2006-01-01

Digital Signal Processing (Second Edition)

Jonathan Blackledge
Technological University Dublin, jonathan.blackledge@tudublin.ie

DIGITAL SIGNAL PROCESSING

"Talking of education, people have now a-days" (he said) "got a strange opinion that every thing should be taught by lectures. Now, I cannot see that lectures can do so much good as reading the books from which the lectures are taken. I know nothing that can be best taught by lectures, except where experiments are to be shown. You may teach chymestry by lectures - You might teach making of shoes by lectures!"

James Boswell: *Life of Dr Samuel Johnson, 1766*

Dedication

To all those students with whom I had the good fortune to work and, in using the material herein, taught me how to teach it.

ABOUT THE AUTHOR

Jonathan Blackledge graduated in physics from Imperial College and music from the Royal College of Music, London, in 1980 and obtained a Doctorate in theoretical physics from the same university in 1983. He was appointed as Research Fellow of Physics at Kings College, London from 1983 to 1988 specializing in inverse problems in electromagnetism and acoustics. During this period, he worked on a number of industrial research contracts undertaking theoretical and computational work on the applications of inverse scattering theory for the analysis of signals and images.

In 1988, he joined the Applied Mathematics and Computing Group at Cranfield University as Lecturer and later, as Senior Lecturer and Head of Group where he promoted postgraduate teaching and research in applied, engineering and industrial mathematics in areas which included computer aided engineering, digital signal processing and computer graphics. In 1994, he was appointed Professor of Applied Mathematics and Computing and Head of the Department of Mathematical Sciences at De Montfort University where he established the Institute of Simulation Sciences. He is currently Professor of Computer Science in the Department of Computer Science at the University of the Western Cape, South Africa nd Professor of of Information and Communications Technology in the Department of Electronics and Electrical Engineering at Loughborough University, England. He is also a co-founder and Director of a group of companies specializing in communications technology and financial analysis based in London and New York.

Professor Blackledge has published over one hundred scientific and engineering research papers and technical reports for industry, six industrial software systems, fifteen patents, ten books and has been supervisor to sixty research (PhD) graduates. He lectures widely to a variety of audiences composed of mathematicians, computer scientists, engineers and technologists in areas that include cryptology, communications technology and the use of artificial intelligence in process engineering, financial analysis and risk management. His current research interests include computational geometry and computer graphics, image analysis, nonlinear dynamical systems modelling and computer network security, working in both an academic and commercial context. He holds Fellowships with England's leading scientific and engineering Institutes and Societies including the Institute of Physics, the Institute of Mathematics and its Applications, the Institution of Electrical Engineers, the Institution of Mechanical Engineers, the British Computer Society, the Royal Statistical Society and the Institute of Directors. He is a Chartered Physicist, Chartered Mathematician, Chartered Electrical Engineer, Chartered Mechanical Engineer, Chartered Statistician and a Chartered Information Technology Professional. He has an additional interest in music for which he holds a Fellowship of the Royal Schools of Music, London.

DIGITAL SIGNAL PROCESSING

Mathematical and Computational Methods, Software Development and Applications

Second Edition

JONATHAN M. BLACKLEDGE†

Professor of Information and Communications Technology,
Department of Electronic and Electrical Engineering,
Loughbourough University, England.

**Horwood Publishing, Chichester,
West Sussex, England**

---

†*Professor of Computer Science, Department of Computer Science, University of the Western Cape, Cape Town, South Africa.*

HORWOOD PUBLISHING LIMITED
Coll House, Westergate, Chichester, West Sussex, PO20 3QL, England.
First published in 2003 and re-printed in 2006 with corrections and additions.

British Library Cataloguing in Publishing Data
A catalogue record of this book is available from the British Library.

ISBN 1-904275-26-5

Typeset and produced by the author using LaTeX, the TeXnicCenter graphical user interface and the stylefile of the Institute of Mathematics and its Applications.

Printed and bound in Great Britain by Antony Rowe Limited.

Foreword to the Second Edition

I was flattered when the publisher, Ellis Horwood asked me to write this Foreword, because my introduction to signal processing began in the Second World War when, as a communications officer in the Royal Corps of Signals, I worked with a war-time teleprinter. We used a system based on 5-bit letters and a pair of copper wires. It provided a bandwidth of about 200Hz that could be separated by filters from the rest of a voice channel without noticeably distorting the speech. Today the bandwidth available is huge by comparison and information can be conveyed through a multitude of channels using tiny glass fibres. However, although the engineering associated with information and communications technology in general has and continues to undergo radical change, many of the underlying mathematical principles remain the same.

G H Hardy in his book *A Mathematician's Apology* wrote that there were 'no interesting applications of pure mathematics'. This is no longer true and Professor Blackledge's book *Digital Signal Processing* will enable many people to make use of their interest in, and perhaps fascination with, mathematics in such a way, and through a field of study, that will help us all communicate our ideas more quickly and conveniently through the digital world of today.

The Earl Kitchener of Khartoum

Preface to the Second Edition

This book provides an account of the mathematical background, computational methods and software engineering associated with digital signal processing. The aim has been to provide the reader with the mathematical methods required for signal analysis which are then used to develop models and algorithms for processing digital signals and finally to encourage the reader to design software solutions for Digital Signal Processing (DSP). In this way, the reader is invited to develop a small DSP library that can then be expanded further with a focus on his/her research interests and applications.

There are of course many excellent books and software systems available on this subject area. However, in many of these publications, the relationship between the mathematical methods associated with signal analysis and the software available for processing data is not always clear. Either the publications concentrate on mathematical aspects that are not focused on practical programming solutions or elaborate on the software development of solutions in terms of working 'black-boxes' without covering the mathematical background and analysis associated with the design of these software solutions. Thus, this book has been written with the aim of giving the reader a technical overview of the mathematics and software associated with the 'art' of developing numerical algorithms and designing software solutions for DSP, all of which is built on firm mathematical foundations. For this reason, the work is, by necessity, rather lengthy and covers a wide range of subjects compounded in four principal parts. Part I provides the mathematical background for the analysis of signals, Part II considers the computational techniques (principally those associated with linear algebra and the linear eigenvalue problem) required for array processing and associated analysis (error analysis for example). Part III introduces the reader to the essential elements of software engineering using the C programming language, tailored to those features that are used for developing C functions or modules for building a DSP library.

The material associated with parts I, II and III is then used to build up a DSP system by defining a number of 'problems' and then addressing the solutions in terms of presenting an appropriate mathematical model, undertaking the necessary analysis, developing an appropriate algorithm and then coding the solution in C. This material forms the basis for part IV of this work.

In most chapters, a series of tutorial problems is given for the reader to attempt with answers provided in Appendix A. These problems include theoretical, computational and programming exercises. Part II of this work is relatively long and arguably contains too much material on the computational methods for linear algebra. However, this material and the complementary material on vector and matrix norms forms the computational basis for many methods of digital signal processing. Moreover, this important and widely researched subject area forms the foundations, not only of digital signal processing and control engineering for example, but also of numerical analysis in general.

The material presented in this book is based on the lecture notes and supplementary material developed by the author for an advanced Masters course 'Digital Signal Processing' which was first established at Cranfield University, Bedford in 1990 and modified when the author moved to De Montfort University, Leicester in 1994.

The programmes are still operating at these universities and the material has been used by some 700++ graduates since its establishment and development in the early 1990s. The material was enhanced and developed further when the author moved to the Department of Electronic and Electrical Engineering at Loughborough University in 2003 and now forms part of the Department's post-graduate programmes in Communication Systems Engineering. The original Masters programme included a taught component covering a period of six months based on two semesters, each Semester being composed of four modules. The material in this work covers the first Semester and its four parts reflect the four modules delivered. The material delivered in the second Semester is published as a companion volume to this work entitled *Digital Image Processing*, Horwood Publishing, 2005 which covers the mathematical modelling of imaging systems and the techniques that have been developed to process and analyse the data such systems provide.

Since the publication of the first edition of this work in 2003, a number of minor changes and some additions have been made. The material on programming and software engineering in Chapters 11 and 12 has been extended. This includes some additions and further solved and supplementary questions which are included throughout the text. Nevertheless, it is worth pointing out, that while every effort has been made by the author and publisher to provide a work that is error free, it is inevitable that typing errors and various 'bugs' will occur. If so, and in particular, if the reader starts to suffer from a lack of comprehension over certain aspects of the material (due to errors or otherwise) then he/she should not assume that there is something wrong with themselves, but with the author!

J M Blackledge, January 2006

Acknowledgements

The material developed in this book has been helped and encouraged by numerous colleagues of the author over the years. The author would like to thank all of his fellow colleagues, but particular thanks go to Prof Roy Hoskins who, for many years, has played a central role in teaching aspects of signal analysis coupled with the mathematical rigour required to come to terms with such entities as the Delta function. Thanks also go to Dr Peter Sherar at Cranfield University who helped the author establish the MSc programme in 'Software Solutions for Digital Signal Processing' from which much of the material presented in this work has been derived. Thanks also go to Dr Martin Turner, Dr Mohammed Jaffar, Dr Martin Crane and Prof Gwynne Evans at De Montfort University who have worked with the author for many years, and as module leaders for De Montfort University's advanced MSc programme in digital signal processing, developed valuable additions to the teaching and learning materials first established by the author. Further, many students of this MSc course were the result of research programmes established by Prof B Foxon at De Montfort University with links to universities and research institutes in Russia, Poland, South Africa and the USA, to whom the author (and the students) are very grateful. The author would also like to thank Prof Peter Smith who as Head of the Department of Electronic and Electrical Engineering at Loughborough University has provided the infrastructure for the author to operate in an internationally acknowledge center of engineering excellence. Thanks also go to Dr S Datta of Loughborough University who has worked with the author for many years and provided him with many valuable recommendations, ideas and research themes. In addition, the author would like to thank all those organizations and industries that have provided funding for the development of his teaching and research activities over the years including: the Engineering and Physical Sciences Research Council, the Defense Evaluation and Research Agency, the International Science and Technology Council, British Coal, British Gas, British Petroleum, British Aerospace, Oxford Instruments, Microsharp, Marconi, Microsoft and British Intelligence. Finally, the author would like to thank all those postgraduate students (both MSc and PhD) who, over the years, have used the material in this book as part of their technical training and educational development and have provided critical and constructive appraisal from one year to the next. The material was written by the author for students undertaking advanced MSc programmes at the universities of Cranfield, De Montfort and more recently, at Loughborough University and has derived great pleasure from presenting it and developing it further as part of his teaching portfolio. In addition to the taught postgraduate programmes from which the material herein has been derived, numerous research students have worked closely with the author providing valuable insights and comments that have helped to enhance and strengthen the material.

Notation

Alphabetic

a_n	Real coefficients of a Fourier cosine series
adj$A$	Adjoint of matrix A
$A \equiv (a_{ij})$	Matrix with element at i^{th} row and j^{th} column
A^T	Transpose of A
A^{-1}	Inverse of A
$[A \mid B]$	Augmented matrix formed from matrices A and B
$\mid A \mid$	Determinant of A
$A(t)$	Amplitude modulation (amplitude envelope)
$A(\omega)$	Amplitude spectrum
b_n	Real coefficients of a Fourier sine series
$\mathbf{b}$	Data vector of linear system $A\mathbf{x} = \mathbf{b}$
chirp(t)	Unit chirp function [with complex form $\exp(-i\alpha t^2)$]
comb(t)	Comb function $[= \sum_n \delta(t - nT)]$
cond(A)	Condition number of matrix A $(=\|A\| \times \|A^{-1}\|)$
c_n	Complex coefficients of a complex Fourier series for example
C	Capacitance or the contour followed by a path of integration in the z-plane
$\mathbf{c}$	Data vector associated with linear system $\mathbf{x} = M\mathbf{x} + \mathbf{c}$
D	Fractal dimension or diagonal matrix
det$A$	Determinant of A (also denoted by $\mid A \mid$)
$\mathbf{e}$	Error vector
$f(t)$	Arbitrary real function - typically object function or system input
$f(z)$	Function of a complex variable
$\mid f \mid$	modulus of complex variable or function f
$\|f(t)\|$	Norm (e.g. a Euclidean norm) of a function $f(t)$
$\|f_i\|$	Norm of an array or vector f_i
$\|f_i\|_2$	Euclidean norm of array or vector f_i
$\|\mathbf{x}\|_p$	p-norm of vector $\mathbf{x}$
$\|\mathbf{x}\|_\infty$	'Infinity' or uniform norm of a vector $\mathbf{x}$
$\|A\|$	Norm of matrix A
$F(\omega)$	Complex spectrum of function $f(t)$
$F_r(\omega)$	Real component of spectrum
$F_i(\omega)$	Imaginary component of spectrum
F_i	Discrete complex spectrum of discrete function f_i
$g(t)$	Arbitrary function
$g(t \mid t_0, \omega)$	Green's function
$H(t)$	Tophat function
I	Unit or identity matrix
Im$[f]$	Imaginary part of complex variable or function f

k	Wavenumber ($= 2\pi/\lambda$)
L	Lower triangular matrix
L_1	Lower triangular matrix with 1's along the leading diagonal
M	Iteration matrix associated with linear system $\mathbf{x} = M\mathbf{x} + \mathbf{c}$
$n(t)$	Noise function
n_i	Discrete noise function
N_i	Noise spectrum
$p(t)$	Instrument function or Impulse Response Function
p_i	Discrete Impulse Response Function
$P(\omega)$	Transfer Function (Fourier transform of p_i)
P_i	Discrete Transfer Function (DFT of p_i
$P(x)$	Probability density function also denoted by $\Pr[x(t)]$
$P(a \mid b)$	Conditional probability of obtaining a given b
$P(\omega)$	Power spectrum ($=\mid F(\omega) \mid^2$) where $F(\omega)$ is the Fourier transform of $f(t)$
P_i	Discrete power spectrum
$\Pr(x)$	Probability occurrence of x
q	Fourier dimension
$q(t)$	Quadrature signal
R	Resistance
R_i	Denotes the i^{th} row of a matrix
$\mathrm{Re}[f]$	Real part of complex variable or function f
$s(t)$	Real or complex (analytic) signal
s_i	Discrete real or complex signal
sgn(t)	Sign function
$\mathrm{sinc}(t)$	Sinc function ($= \sin(t)/t$)
t	Time
U	Upper triangular matrix
U_1	Upper triangular matrix with 1's along the leading diagonal
$U(t)$	Unit step function
$u(x,t)$	Solution to a partial differential equation (e.g. wavefield)
$\mathbf{v}_i$	Eigenvectors of linear system $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$
$WAL(n,t)$	Walsh function
$\mathbf{x}$	Solution vector of linear system $A\mathbf{x} = \mathbf{b}$
$\mathbf{x}_i$	Eigenvectors of linear system $A\mathbf{x}_i = \lambda_i \mathbf{x}_i$
$\mathbf{x}^T$	Transpose of vector $\mathbf{x}$
x_i	i^{th} element of vector $\mathbf{x}$
x_0	Initial value
z	Complex number of the form $a + ib$
z^*	Complex conjugate $a - ib$ of a complex number $a + ib$
$\in$	In (e.g. $x \in [a,b)$ is equivalent to $a \leq x < b$)
$\forall$	Forall (e.g. $f(t) = 0, \quad \forall t \in (a,b]$)

Greek

α	Chirping parameter
$\Gamma(q)$	Gamma function
$\delta(t)$	Dirac delta function
δ_{ij}	Kronecker delta
δt	Small increment (denoted also by Δt)
$\theta(t)$	Instantaneous phase
λ	Wavelength
λ_i	Eigenvalues of linear system $A\mathbf{x}_i = \lambda_i \mathbf{x}_i$
$\psi(t)$	Instantaneous frequency
ϕ_n	Delta sequence function
$\rho(M)$	Spectral radius of (iteration) matrix M
ω	Angular frequency or the relaxation parameter
Ω	Bandwidth of a spectrum

Operators

$\hat{C}$	Cosine transform operator
$\hat{D}$	Linear differential operator
$\hat{D}^q$	Fractional differential operator
$\hat{F}_1$	One dimensional Fourier transform
$\hat{F}_1^{-1}$	One dimensional inverse Fourier transform
$\hat{H}$	Hilbert transform operator
$\hat{I}^q$	Fractional integral operator (e.g. Riemann-Liouville fractional integral)
$\hat{L}$	One sided Laplace transform operator
$\hat{L}^{-1}$	Inverse Laplace transform operator (Bromwich integral)
$\hat{S}$	Sine transform operator
$\hat{W}$	Wavelet transform
$\hat{W}^{-1}$	Inverse wavelet transform
$\otimes$	Convolution operation (continuous or discrete and causal or otherwise, depending on the context specified)
$\odot$	Correlation operation (continuous or discrete and causal or otherwise, depending on the context specified)
$\Longleftrightarrow$	Transformation into Fourier space
$\longleftrightarrow$	Transformation into some transform space (as defined)

Glossary of Terms

Mathematical and Statistical

ACF	Autocorrelation Function
AM	Amplitude Modulations (the amplitude envelope)
BL	BandLimited
CDP	Common Depth Point
CP/M	Control Program for Microprocessors
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
FM	Frequency Modulations
FIR	Finite Impulse Response
GUI	Graphical User Interface
IDFT	Inverse Discrete Fourier Transform
IIR	Infinite Impulse Response
IRF	Impulse Response Function
LCM	Linear Congruential Method
MAP	Maximum *a Posteriori*
ML	Maximum Likelihood
PDE	Partial Differential Equation
PDF	Probability Distribution or Density Function
PSDF	Power Spectral Distribution or Density Function
RSRA	ReScaled Range Analysis
RZC	Real Zero Conversion
STFT	Short Time Fourier Transform
TF	Transfer Function
WV	Wigner-Ville

Computer Science

BCD	Binary Coded Decimal
CASE	Computer Aided Software Engineering
CPU	Central Processing Unit
DSP	Digital Signal Processor
I/O	Input/Output
JCL	Job Control Language
PCNG	Pseudo Chaotic Number Generator
PRNG	Pseudo Random Number Generator
RAM	Random Access Memory
VAX	Virtual Address Extension
VDU	Visual Display Unit
VMS	Virtual Memory System

Organizational and Standards

DOS	Disc Operating System
EMH	Efficient Market Hypothesis
FMH	Fractal Market Hypothesis
LPI	Log Price Increment
MATLAB	Highlevel technical computing language by MathWorks Inc.
MPT	Modern Portfolio Theory
MS	Microsoft
NYA	New York Average
PKI	Public Key Infrastructure
RSA	Rivest, Shamir and Adleman

Contents

Foreword v
Preface vi
Acknowledgements viii
Notation ix
Glossary of Terms xii

Introduction **1**

I Signal Analysis **8**

1 Complex Analysis **9**
1.1 Introduction 9
1.2 Complex Numbers 9
1.2.1 Addition, Subtraction, Multiplication and Division 9
1.2.2 Powers of $i = \sqrt{-1}$ 10
1.2.3 The Complex Conjugate 10
1.2.4 Geometrical Representation (The Argand Diagram) 11
1.2.5 De Moivre's Theorem 12
1.2.6 The Complex Exponential 12
1.3 Complex Functions 13
1.3.1 Differentiability of a Complex Function 14
1.3.2 The Cauchy-Riemann Equations 15
1.3.3 Analytic Functions 16
1.3.4 Some Important Results 18
1.4 Complex Integration 18
1.4.1 Green's Theorem in the Plane 21
1.4.2 Cauchy's Theorem 23
1.4.3 Defining a Contour 25
1.4.4 Example of the Application of Cauchy's Theorem 26
1.4.5 Cauchy's Integral Formula 28
1.5 Cauchy's Residue Theorem 28
1.5.1 Poles and Residues 29
1.5.2 Residues at Simple Poles 29
1.5.3 Residues at Poles of Arbitrary Order 30
1.5.4 The Residue Theorem 31
1.5.5 Evaluation of Integrals using the Residue Theorem 32

1.5.6 Evaluation of Trigonometric Integrals . . . 36
1.6 Summary of Important Results . . . 36
1.7 Further Reading . . . 38
1.8 Problems . . . 38

2 The Delta Function 41
2.1 Some Important Generalized Function . . . 42
2.2 The Delta Function . . . 43
2.2.1 Examples of δ-sequences . . . 44
2.2.2 Integral Representation of the δ-function . . . 45
2.3 Properties of the δ-function . . . 45
2.4 Derivatives of the δ-function . . . 46
2.5 Integration Involving Delta Functions . . . 48
2.6 Convolution . . . 49
2.7 The Green's Function . . . 50
2.8 Summary of Important Results . . . 53
2.9 Further Reading . . . 55
2.10 Problems . . . 55

3 The Fourier Series 57
3.1 Derivation of the Fourier Series . . . 57
3.2 The Half Range Fourier Series . . . 60
3.2.1 Cosine Series . . . 61
3.2.2 Sine Series . . . 61
3.3 Fourier Series for an Arbitrary Period . . . 62
3.4 Applications of the Fourier Series to Circuit Theory . . . 63
3.5 The Complex Fourier Series . . . 67
3.6 The Fourier Transform Pair . . . 69
3.7 The Discrete Fourier Transform . . . 70
3.8 Relationship between the DFT and the Fourier Transform . . . 70
3.9 'Standard' and 'Optical' Forms of the DFT . . . 71
3.10 Summary of Important Results . . . 72
3.11 Further Reading . . . 73
3.12 Problems . . . 73

4 The Fourier Transform 75
4.1 Introduction . . . 75
4.1.1 Notation . . . 76
4.1.2 Physical Interpretation . . . 76
4.1.3 The Spectrum . . . 77
4.1.4 The Inverse Fourier Transform . . . 78
4.1.5 Alternative Definitions and Representations . . . 79
4.1.6 Useful Notation and Jargon . . . 79
4.1.7 Bandlimited Functions . . . 79
4.1.8 The Amplitude and Phase Spectra . . . 80
4.1.9 Differentiation and the Fourier Transform . . . 81
4.1.10 Integration and the Fourier Transform . . . 82
4.2 Selected but Important Functions . . . 82

4.3 Selected but Important Theorems . . . 86
4.4 Convolution and Correlation . . . 87
4.4.1 Convolution . . . 87
4.4.2 Correlation . . . 88
4.4.3 Physical Interpretation . . . 89
4.4.4 Autoconvolution and Autocorrelation . . . 89
4.4.5 The Convolution Theorem . . . 90
4.4.6 The Product Theorem . . . 90
4.4.7 The Correlation Theorem . . . 91
4.4.8 The Autoconvolution and Autocorrelation Theorems . . . 92
4.4.9 Selected but Important Properties . . . 92
4.5 The Sampling Theorem . . . 93
4.5.1 Fourier Transform of the comb Function . . . 94
4.5.2 Proof of the Sampling Theorem . . . 96
4.5.3 Sinc Interpolation . . . 97
4.6 Fourier Filters . . . 97
4.6.1 Lowpass Filters . . . 99
4.6.2 Highpass Filters . . . 99
4.6.3 Bandpass Filters . . . 100
4.6.4 The Inverse Filter . . . 100
4.7 The Kronecker Delta Function and the DFT . . . 101
4.8 Deriving Filters using the DFT . . . 102
4.9 Case Study: Scattering from Layered Media . . . 104
4.9.1 Introduction to the Wave Equation . . . 105
4.9.2 Asymptotic Green's Function Solution . . . 107
4.9.3 Discussion . . . 109
4.10 Summary of Important Results . . . 110
4.11 Further Reading . . . 111
4.12 Problems . . . 111

5 Other Integral Transforms 114
5.1 The Laplace Transform . . . 114
5.2 The Sine and Cosine Transforms . . . 123
5.2.1 The Sine Transform . . . 123
5.2.2 The Cosine Transform . . . 124
5.3 The Walsh Transform . . . 125
5.4 The Cepstral Transform . . . 127
5.5 The Hilbert Transform . . . 129
5.5.1 Modulation and Demodulation . . . 130
5.5.2 Quadrature Detection . . . 131
5.5.3 The Analytic Signal . . . 132
5.5.4 Attributes of the Analytic Signal . . . 134
5.5.5 Phase Unwrapping . . . 135
5.6 Case Study: FM Imaging by Real Zero Conversion . . . 136
5.6.1 Real Zero Conversion . . . 136
5.6.2 Application to Seismic Imaging . . . 138
5.7 STFT and The Gabor Transform . . . 139

5.8 The Wigner and the Wigner-Ville Transforms . . . 141
5.8.1 Origins of the Transforms for Signal Processing . . . 142
5.8.2 Properties of the Wigner-Ville Transform . . . 143
5.8.3 Cross Term Interference . . . 146
5.8.4 Smoothing the Wigner-Ville Distribution . . . 147
5.8.5 The Discrete Wigner-Ville Transform . . . 148
5.8.6 Applications . . . 148
5.9 The Riemann-Liouville and the Wyle Transforms . . . 149
5.10 The z-Transform . . . 150
5.11 The Wavelet Transform . . . 152
5.12 Discussion . . . 154
5.13 Summary of Important Results . . . 154
5.14 Further Reading . . . 157
5.15 Problems . . . 158

II Computational Linear Algebra 161

6 Matrices and Matrix Algebra 162
6.1 Matrices and Matrix Algebra . . . 163
6.1.1 Addition of Matrices . . . 163
6.1.2 Subtraction of Matrices . . . 163
6.1.3 Multiplication of a Matrix by a Scalar Quantity . . . 164
6.1.4 Multiplication of Two Matrices . . . 164
6.1.5 Further Results in Matrix Multiplication . . . 165
6.1.6 The Unit Matrix . . . 166
6.1.7 The Transpose of a Matrix . . . 166
6.1.8 The Inverse Matrix . . . 167
6.1.9 Determinants . . . 168
6.1.10 Properties of Determinants . . . 169
6.2 Systems of Linear Algebraic Equations . . . 170
6.2.1 Formal Methods of Solution . . . 171
6.2.2 Evaluation of adj$A$ . . . 172
6.2.3 Cramers Rule . . . 173
6.3 Linear Systems . . . 175
6.3.1 Inhomogeneous systems . . . 175
6.3.2 Homogeneous Systems . . . 175
6.3.3 Ill-conditioned Systems . . . 176
6.3.4 Under-determined Systems . . . 177
6.3.5 Over-determined Systems . . . 177
6.4 Summary of Important Results . . . 178
6.5 Further Reading . . . 180
6.6 Problems . . . 180

7 Direct Methods of Solution **183**
7.1 Gauss' Method . . . 184
7.1.1 Generalization of the Idea . . . 185
7.1.2 Conversion to Pseudo-Code . . . 186
7.1.3 Pivots and Pivoting . . . 187
7.1.4 Pivotal Strategies . . . 188
7.2 Jordan's Method . . . 190
7.2.1 Matrix Inversion by Jordan's Method . . . 191
7.2.2 Gauss' .v. Jordan's Method . . . 193
7.3 LU Factorization . . . 193
7.3.1 Existence and Uniqueness of LU Factorization . . . 193
7.3.2 Generalization of Crout's Method . . . 196
7.3.3 Generalization of Cholesky's Method . . . 197
7.3.4 Matrix Inversion by *LU* Factorization . . . 198
7.4 Banded Systems . . . 199
7.4.1 Tridiagonal Systems . . . 199
7.4.2 Solution to Tridiagonal Systems . . . 200
7.5 Computational Considerations . . . 201
7.5.1 Computer Storage Requirements . . . 201
7.5.2 Arithmetic Involved in the Computation . . . 202
7.5.3 Scaling of Matrices . . . 203
7.6 Solution to Complex Systems of Equations . . . 203
7.7 Summary of Important Results . . . 204
7.8 Further Reading . . . 205
7.9 Problems . . . 205

8 Vector and Matrix Norms **208**
8.1 Vector Norms . . . 208
8.1.1 Definitions of a Vector Norm . . . 209
8.1.2 Commonly Used Definitions . . . 209
8.1.3 Generalization of Vector Norms - The 'p-Norm' . . . 209
8.1.4 Metric Spaces . . . 210
8.1.5 The Euclidean Norm . . . 210
8.1.6 The Cauchy-Schwarz Inequality for the Euclidean Norm . . . 211
8.2 The Triangle Inequality for the Euclidean Norm . . . 211
8.2.1 Holder's Inequality . . . 212
8.2.2 Minkowski's Inequality . . . 214
8.3 Matrix Norms . . . 215
8.3.1 Types of Matrix Norms . . . 215
8.3.2 Basic Definition of a Matrix Norm . . . 215
8.3.3 Evaluation of ℓ_1 and ℓ_∞ Matrix Norms . . . 216
8.4 The Conditioning of Linear Equations . . . 218
8.4.1 Conditioning of $A\mathbf{x} = \mathbf{b}$ . . . 219
8.4.2 Example of an Ill-conditioned System . . . 222
8.5 Iterative Improvement . . . 223
8.6 The Least Squares Method . . . 226
8.6.1 The Least Squares Principle . . . 226

8.6.2 Linear Polynomial Models . . . 227
8.6.3 The Orthogonality Principle . . . 228
8.6.4 Complex Signals, Norms and Hilbert Spaces . . . 228
8.6.5 Linear Convolution Models . . . 229
8.7 Summary of Important Results . . . 232
8.8 Further Reading . . . 233
8.9 Problems . . . 234

9 Iterative Methods of Solution 237
9.1 Basic Ideas . . . 238
9.1.1 Jacobi's Method . . . 239
9.1.2 The Gauss-Seidel Method . . . 239
9.1.3 The Relaxation or Chebyshev Method . . . 239
9.2 Iterative Methods . . . 240
9.3 Example of the Iteration Method . . . 240
9.4 General Formalism . . . 242
9.5 Criterion for Convergence of Iterative Methods . . . 243
9.5.1 Diagonal Dominance . . . 243
9.5.2 Sufficient Condition for Convergence . . . 244
9.5.3 Proof of the Necessary Condition for Convergence . . . 245
9.5.4 Estimating the Number of Iterations . . . 247
9.5.5 The Stein-Rosenberg Theorem . . . 248
9.6 The Conjugate Gradient Method . . . 248
9.6.1 The Gram-Schmidt Process . . . 249
9.6.2 Example of the Gram-Schmidt Process . . . 249
9.6.3 Practical Algorithm for the Conjugate Gradient Method . . . 250
9.7 Summary of Important Results . . . 251
9.8 Further Reading . . . 252
9.9 Problems . . . 252

10 Eigenvalues and Eigenvectors 255
10.1 Formal Methods of Solution . . . 257
10.2 Properties of Eigenvalues . . . 258
10.3 The Cayley-Hamilton Theorem . . . 263
10.4 The Power Method . . . 265
10.4.1 Basic Algorithm for the Power Method . . . 267
10.4.2 Problems Concerning the Power Method . . . 268
10.4.3 Deflation . . . 268
10.4.4 The Deflation Method for a Non-symmetric Matrix . . . 271
10.4.5 The Deflation Method for a Symmetric Matrix . . . 272
10.5 Jacobi's Method for Symmetric Matrices . . . 272
10.5.1 The Transformation Matrix . . . 273
10.5.2 The Serial Method . . . 274
10.6 Sturm Sequence Iteration . . . 275
10.6.1 Gerschgorin's Theorem . . . 277
10.6.2 Givens' Method . . . 278
10.6.3 Householder's Method . . . 279

10.7 LR and QR Methods . . . 281
10.8 Inverse Iteration . . . 285
10.9 Special Types of Matrices . . . 285
10.10 Summary of Important Results . . . 290
10.11 Further Reading . . . 294
10.12 Problems . . . 295

III Programming and Software Engineering 298

11 Principles of Software Engineering 299
11.1 Introduction . . . 299
11.1.1 Programming Language Development . . . 299
11.1.2 What is Software Engineering ? . . . 302
11.1.3 Applications . . . 303
11.1.4 About Part III . . . 304
11.2 Decimal and Binary Number Systems . . . 305
11.3 Binary Number Systems . . . 306
11.3.1 Binary Coded Decimal (BCD) . . . 306
11.3.2 Binary Arithmetic . . . 307
11.3.3 Decimal to Binary Conversion of Whole Numbers . . . 307
11.3.4 Decimal to Binary Conversion of Decimal Numbers . . . 308
11.3.5 Conversion form Binary to Decimal . . . 308
11.4 Fixed Point Storage and Overflow . . . 309
11.5 Floating Point Representation of Binary Numbers . . . 309
11.6 Numerical Error and Accuracy . . . 310
11.6.1 Errors . . . 311
11.6.2 Types of Errors . . . 311
11.6.3 Accumulation of Errors . . . 313
11.6.4 Types of Error Growth . . . 313
11.6.5 Errors in Computer Arithmetic . . . 313
11.7 Methods that Reduce Errors and Maintain Accuracy . . . 316
11.8 Program Structures . . . 317
11.8.1 Syntax and Semantics . . . 317
11.8.2 Data declarations . . . 317
11.8.3 Input and Output . . . 318
11.8.4 Operations of Data . . . 318
11.8.5 Control . . . 319
11.8.6 Subprograms . . . 320
11.9 Procedures . . . 321
11.10 Processes Specification . . . 322
11.10.1 Pseudocode . . . 322
11.10.2 Program Flowcharts . . . 323
11.10.3 Program Structure Block Diagrams . . . 323
11.10.4 Warnier Diagrams . . . 323
11.10.5 Decision Tables and Trees . . . 323
11.11 Program Specification . . . 324

11.11.1 Program Design, Style and Presentation 324
11.11.2 Program Reliability 325
11.11.3 Program Efficiency 325
11.11.4 Program Development Time 326
11.11.5 Program Documentation 326
11.12 System Design 327
11.12.1 Specification 327
11.12.2 Design 328
11.12.3 Testing 328
11.12.4 Software Maintenance 328
11.13 Requirement Specifications 329
11.14 Modularity 329
11.15 The Jackson Method 330
11.16 Data Analysis 331
11.17 Data Flow Design 331
11.18 Testing and Implementation 332
11.19 Stages of System Design 333
11.19.1 Statement of User Requirements 333
11.19.2 Functional Specification 333
11.19.3 Technical Specification 333
11.19.4 Detailed Design 334
11.19.5 Programming 334
11.19.6 Unit Testing 335
11.19.7 String and System Testing 335
11.19.8 Summary on System Design 335
11.20 Computer Aided Software Engineering Tools 335
11.21 Operating Systems and Languages 336
11.21.1 Functions of an Operating System 337
11.21.2 Types of Operating System 338
11.21.3 DOS 338
11.21.4 UNIX 339
11.22 Programming Languages 340
11.22.1 Factors in the Choice of a Language 340
11.22.2 FORTRAN 341
11.22.3 Pascal 343
11.22.4 Basic 343
11.22.5 COBOL 344
11.22.6 ALGOL 344
11.22.7 PL/1 345
11.22.8 APL 345
11.22.9 C 346
11.22.10 C++ 347
11.22.11 Java 347
11.23 Object Oriented Programming 348
11.23.1 Inheritance 349
11.23.2 Virtual Functions and Abstract Classes 350
11.23.3 Polymorphism 350

11.23.4 Dynamic Binding . . . 351
11.23.5 The C++ Programming Language . . . 351
11.23.6 C++ as an Extension of C . . . 353
11.23.7 The Object Oriented Programming Paradigm . . . 354
11.23.8 Data . . . 356
11.24 Discussion . . . 357
11.25 Summary of Important Results . . . 358
11.26 Further Reading . . . 361
11.27 Problems . . . 362

12 Modular Programming in C 364
12.1 About C . . . 364
12.1.1 Statement Layout . . . 365
12.1.2 Documentation . . . 365
12.1.3 Input/Output . . . 365
12.1.4 Date Type Statements . . . 366
12.1.5 Variable Names . . . 367
12.1.6 Declarations . . . 367
12.1.7 Arrays . . . 367
12.1.8 Operators . . . 367
12.1.9 Expressions . . . 369
12.1.10 Control Statements . . . 369
12.1.11 Looping and Iteration . . . 371
12.1.12 User Defined Functions . . . 372
12.1.13 Passing Variables using Pointers . . . 374
12.1.14 Internal Functions and Libraries . . . 375
12.1.15 Prototyping . . . 375
12.1.16 Advantages and Disadvantages of C . . . 377
12.2 Modular and Structured Programming in C . . . 377
12.2.1 Array Processing . . . 378
12.2.2 Dynamic Memory Allocation . . . 379
12.3 Modularity . . . 381
12.4 Module Size . . . 382
12.5 Structured Programming . . . 386
12.6 Modular Programming using Borland Turbo C++ . . . 390
12.6.1 Speed and Memory . . . 391
12.6.2 Compiling and Linking . . . 392
12.6.3 Developing an Object Library . . . 392
12.6.4 Object Libraries . . . 394
12.6.5 Practical Applications . . . 394
12.7 On Style and Presentation . . . 395
12.8 Summary of Important Results . . . 397
12.9 Further Reading . . . 398
12.10 Problems . . . 399

IV DSP: Methods, Algorithms and Building a Library 403

13 Digital Filters and the FFT 404
13.1 Digital Filters . . . 404
13.2 The Fast Fourier Transform . . . 406
13.2.1 Basic Ideas . . . 406
13.2.2 Bit Reversal . . . 408
13.2.3 The FFT in C . . . 409
13.3 Data Windowing . . . 413
13.4 Computing with the FFT . . . 416
13.5 Discrete Convolution and Correlation . . . 416
13.6 Computing the Analytic Signal . . . 418
13.7 Summary of Important Results . . . 420
13.8 Further Reading . . . 420
13.9 Programming Problems . . . 421
13.9.1 Digital Signal Generation . . . 421
13.9.2 Computing with the FFT . . . 422

14 Frequency Domain Filtering with Noise 426
14.1 Highpass, Lowpass and Bandpass Filters . . . 426
14.2 The Inverse Filter . . . 427
14.3 The Wiener Filter . . . 428
14.3.1 The Least Squares Principle . . . 428
14.3.2 Derivation of the Wiener Filter . . . 429
14.3.3 Signal Independent Noise . . . 430
14.3.4 Properties of the Wiener Filter . . . 430
14.3.5 Practical Implementation . . . 430
14.3.6 FFT Algorithm for the Wiener Filter . . . 431
14.3.7 Estimation of the Signal-to-Noise Power Ratio . . . 434
14.4 Power Spectrum Equalization . . . 435
14.5 The Matched Filter . . . 437
14.5.1 Derivation of the Matched Filter . . . 437
14.5.2 White Noise Condition . . . 438
14.5.3 FFT Algorithm for the Matched Filter . . . 438
14.5.4 Deconvolution of Frequency Modulated Signals . . . 439
14.6 Case Study: Watermarking using Chirp Coding . . . 442
14.6.1 Introduction . . . 443
14.6.2 Matched Filter Reconstruction . . . 445
14.6.3 The Fresnel Transform . . . 445
14.6.4 Chirp Coding, Decoding and Watermarking . . . 446
14.6.5 Code Generation . . . 448
14.6.6 MATLAB Application Programs . . . 450
14.6.7 Discussion . . . 456
14.7 Constrained Deconvolution . . . 457
14.8 Homomorphic Filtering . . . 458
14.9 Noise . . . 459
14.10 Noise Types . . . 460

14.10.1 Multiple Scattering Effects 460
14.10.2 Beam Profile Effects 462
14.11 Pseudo Random Number Generation 462
14.11.1 Pseudo Random Sequences 463
14.11.2 Real Random Sequences 463
14.11.3 Pseudo Random Number Generators 464
14.11.4 Shuffling 467
14.12 Additive Generators 468
14.12.1 Pseudo Random Number Generators and Cryptography . . . 468
14.12.2 Gaussian Random Number Generation 471
14.13 Chaos 474
14.13.1 The Lyapunov Exponent and Dimension 481
14.14 Case Study: Cryptography using Chaos 482
14.14.1 Block Cyphers using Deterministic Chaos 484
14.14.2 Encrypting Processes 485
14.14.3 Key Exchange and Authentication 486
14.15 Summary of Important Results 488
14.16 Further Reading 490
14.17 Programming Problems 491

15 Statistics, Entropy and Extrapolation 494
15.1 Bayes Rule 494
15.1.1 Bayesian Estimation 496
15.1.2 Some Simple Examples of Bayesian Estimation 497
15.1.3 The Maximum Likelihood Estimation 498
15.2 The Maximum Likelihood Method 501
15.3 Maximum a Posteriori Method 502
15.4 The Maximum Entropy Method 503
15.4.1 Information and Entropy 503
15.4.2 Maximum Entropy Deconvolution 505
15.4.3 Linearization 507
15.4.4 The Cross Entropy Method 507
15.5 Spectral Extrapolation 508
15.6 The Gerchberg-Papoulis Method 511
15.7 Application of Weighting Functions 512
15.8 Burg's Maximum Entropy Method 515
15.9 Summary of Important Results 517
15.10 Further Reading 519
15.11 Problems 520

16 Digital Filtering in the Time Domain 522
16.1 The FIR Filter 522
16.2 The FIR Filter and Discrete Correlation 526
16.3 Computing the FIR filter 529
16.3.1 Moving Window Filters 531
16.3.2 Interpolation using the FIR Filter 533
16.3.3 The FIR Filter and the FFT 534

16.4 The IIR Filter . . . 534
16.5 Non-Stationary Problems . . . 535
16.6 Summary of Important Results . . . 537
16.7 Further Reading . . . 539
16.8 Programming Problems . . . 539

17 Random Fractal Signals 541
17.1 Introduction . . . 542
17.2 Stochastic Modelling Revisited . . . 544
17.3 Fractional Calculus . . . 547
17.3.1 The Laplace Transform and the Half Integrator . . . 548
17.3.2 Operators of Integer Order . . . 549
17.3.3 Convolution Representation . . . 550
17.3.4 Fractional Differentiation . . . 551
17.3.5 Fractional Dynamics . . . 554
17.4 Non-stationary Fractional Dynamic Model . . . 556
17.5 Green's Function Solution . . . 559
17.6 Digital Algorithms . . . 565
17.7 Non-stationary Algorithm . . . 566
17.8 General Stochastic Model . . . 572
17.9 Case Study: Fractal Modulation . . . 573
17.9.1 Secure Digital Communications . . . 573
17.9.2 Fractal Modulation and Demodulation . . . 575
17.10 Case Study: Financial Signal Processing . . . 578
17.11 Introduction . . . 579
17.12 The Efficient Market Hypothesis . . . 580
17.13 Market Analysis . . . 583
17.13.1 Risk .v. Return: Arbitrage . . . 584
17.13.2 Financial Derivatives . . . 585
17.13.3 Black-Scholes Analysis . . . 587
17.13.4 Macro-Economic Models . . . 589
17.13.5 Fractal Time Series and Rescaled Range Analysis . . . 595
17.14 Modelling Financial Data . . . 601
17.14.1 Psychology and the Bear/Bull Cycle . . . 603
17.14.2 The Multi-Fractal Market Hypothesis . . . 604
17.15 Summary of Important Results . . . 609
17.16 Further Reading . . . 611
17.17 Problems . . . 611

Summary 612

A Solutions to Problems 616
A.1 Part I . . . 616
A.1.1 Solutions to Problems Given in Chapter 1 . . . 616
A.1.2 Solutions to Problems Given in Chapter 2 . . . 622
A.1.3 Solutions to Problems Given in Chapter 3 . . . 627
A.1.4 Solutions to Problems Given in Chapter 4 . . . 631
A.1.5 Solutions to Problems Given in Chapter 5 . . . 637

A.1.6 Supplementary Problems to Part I 646
A.2 Part II 661
A.2.1 Solutions to Problems Given in Chapter 6 661
A.2.2 Solutions to Problems Given in Chapter 7 665
A.2.3 Solutions to Problems Given in Chapter 8 672
A.2.4 Solutions to Problems Given in Chapter 9 677
A.2.5 Solutions to Problems Given in Chapter 10 685
A.2.6 Supplementary Problems to Part II 693
A.3 Part III 703
A.3.1 Solution to Problems Given in Chapter 11 703
A.3.2 Solutions to Problems Given in Chapter 12 709
A.3.3 Supplementary Problems to Part III 720
A.4 Part IV 731
A.4.1 Solutions to Problems Given in Chapter 13 731
A.4.2 Solutions to Problems Given in Chapter 14 751
A.4.3 Solutions to Problems Given in Chapter 15 762
A.4.4 Solutions to Problems Given in Chapter 16 768
A.4.5 Solutions to Problems given in Chapter 17 774
A.4.6 Supplementary Problems to Part IV 780

B Graphics Utility **788**

Index **800**

Introduction

Many aspects of electrical and electronic engineering have been reduced to the application of programming methods for processing digital signals. In the 'old days', the electrical engineer used to get the soldering iron out and 'make things', e.g. transistor circuits and later, integrated circuits using functional electronics. Moreover, many of these systems were based on analogue technology. Nowadays, much of the electrical engineers job is based on processing information in the form of digital signals using ever more powerful CPUs and increasingly specialist DSP hardware that, more often than not, are just powerful floating point accelerators for processing specialist software.

The design of any electronic system in terms of both the hardware that executes it and the software that 'drives' it is inextricably bound up with the simulation of the system. Indeed, although the jargon and sound bites change radically from one scientific and engineering discipline to another, the use of mathematical modelling for computer simulation has become of major significance in industry. A wealth of excellent computer packages exist for this purpose, which engineers use routinely for design and development. The electrical and electronic engineer now has many highly sophisticated simulators for designing digital signal processors which can then be used to program an appropriate chip directly. Much of the work is then undertaken in the design and generation of code which is invariably based on C and/or C++ depending on the characteristics of the problem. It is within this context that the material herein has been prepared but with the proviso that the reader first comes to terms with the mathematical and computational background upon which all such systems are ultimately based.

The principal purpose of this book is to take the reader through the theoretical and practical aspects required to design and apply a DSP object library using programming techniques that are appropriate to the successful completion of this task. The culmination of this process is the basis for the material given in Part IV and Parts I-III can be considered to be the background to this process for those readers that have no previous experience of the subject. The material is based on a set of lecture notes and supplementary material developed by the author over a number of years as part of an MSc programme in 'Digital Signal Processing' established by the author at the Universities of Cranfield, De Montfort and Loughborough in England over the 1990s. This programme has increasingly made use of MATLAB as a prototyping environment which is ideal for investigating DSP algorithms via application of the MATLAB DSP toolbox. However, emphasis has and continues to be based on instructing students on the design of C code for DSP so that software can be devel-

oped that is independent of a commercial system (other than the C/C++ compiler). The programming approach has been based on the use of Borland Turbo C++ and this is reflected in some of the code discussed in this work, in particular, the graphics utility that is provided for readers to display signals in Appendix B. Apart from this aspect, the C code that is provided here is independent of a specific compiler and the reader may introduce other graphics facilities as required, including those associated with MATLAB which has excellent facilities in this regard and have been used in this book from time to time.

Part I of this book covers the mathematical methods that lie behind the processing of signals and their analysis. Chapter 1 covers the essentials of complex analysis from the introduction of complex numbers through to the principles of complex integration and the results that can be used to evaluate a wide variety of integrals. Such methods are of significant value in providing analytical solutions to problems in signal and circuit analysis such as in the evaluation of the response of time invariant linear systems to periodic and/or aperiodic inputs; this includes the design and characterisation of different filters and their theoretical evaluation. Moreover, complex analysis features routinely in the application of a wide variety of integral transforms such the Fourier transform that is introduced in Chapter 4 for example.

Chapter 2 introduces the reader to a singularly important generalised function, namely, the delta function together with other related generalised functions such as the step function, the sign function and the tophat function for example. All of these functions are of specific importance in signal analysis where the delta function plays a pivotal role especially in the development of generalised Fourier theory and the sampling theorem for example.

The Fourier transform is introduced and studied using two approaches. The first of these is based on the Fourier series which is the basis for the material discussed in Chapter 3 and introduces the Fourier transform (and the discrete Fourier transform) using a classical approach. The second approach to introducing the Fourier transform is via a path that is intrinsically related to the delta function; this is the so called generalised approach to Fourier theory and is discussed in Chapter 4. In practice, both approaches to Fourier theory are important especially when the reader is required to comprehend the connection and synergies that exist between the theoretical basis for many aspects of signal analysis and the systematic design of a computer algorithm for processing digital signals. Although the Fourier transform is arguably the 'work-horse' (both theoretically and computationally) for studying time invariant linear systems, it is not the only integral transform that is of value. Chapter 5 of Part I looks at other integral transforms that are of value to signal analysis. These transforms include the Laplace transform, the sine and cosine transforms which are used for solving causal or initial value problems and for data compression for example, the Gabor, Wigner and Wigner-Ville transforms for studying signals that are time variant, the z-transform (which is used for solving models for signals that are based on discrete control systems for example) and transforms such as the Riemann-Liouville transform which forms the basis for modelling and analysing fractal signals for example. The wavelet transform is also briefly introduced together with a short discussion of its origins and properties with regard to its multi-resolution characteristics.

Part II is on computational linear algebra which starts with a review of matrix algebra and linear algebraic systems of equations given in Chapter 6. The mater-

ial covers methods of solving linear equations using direct methods (Chapter 7) and indirect or iterative methods (Chapter 9) and considers techniques for solving the linear eigenvalue problem in Chapter 10. In addition, a single chapter (Chapter 8) is devoted to a study of vector and matrix norms which are an essential analytical tool used for error analysis, optimization and the quantification of numerical procedures. Part II is relatively extensive and there is arguably too much material on this subject given the remit of this book. However, many years experience in teaching DSP to graduate students by the author has revealed that there is often a fundamental lack of knowledge of computational linear algebra, a subject area that is absolutely fundamental for the formal mathematical definition of digital signals and the digital algorithms that process them. Many of the algorithms used in DSP end up being expressed in matrix form and the numerical solutions to such problems invariable require the solution to systems of linear algebraic equations. Similarly, the computation of eigenvalues and eigenvectors is not only an important aspect of linear algebra but also enters into some practical methods of signal analysis.

Part III covers aspect of the software engineering methodologies and ideas that a programmer should comprehend. This material is deliberately integrated into the C programming language but does not represent a complete or even partial course on C. Instead, those aspects of the language are discussed that are essential only to the programming exercises that are given in Part IV. These 'essentials' are coupled to a discussion on the principles of software engineering with an emphasis on good programming practice; in particular, modular and structured programming and the design of a DSP library. Thus, apart from introducing the reader to the principles of signal analysis and DSP, one of the goals of this work is to help the reader design their own software provision from which further extensions and developments can be made. Chapter 11 provides an introduction to the essentials of number systems and issues concerning numerical accuracy that are associated with them (e.g. errors associated with the binary representation of decimal numbers) and provides a brief overview of programming languages and operating systems that the reader may or otherwise have acquired a working knowledge of. This chapter also provides a discussion on the principles of software engineering which are extended in Chapter 12 via a programming approach working with C. This includes the presentation of some example programs relating to the computational methods discussed in Part II and other numerical procedures that are either useful examples in themselves or relate to methods that are required later on in the work. Specific examples are given of programs that are required to help the reader design, execute and test the DSP modules discussed in Part IV.

Part IV discusses the principles of DSP in terms of a set of specific problems, solutions, the design of an appropriate algorithm and finally, the development of C code, which is left to the reader as an exercise. The material makes both general or specific reference to that presented in Parts I-III and provides examples of specific algorithms using pseudo code via the programming practices discussed in Part III. Chapter 13 discusses the use of digital filters with regard to the application of the Fast Fourier Transform or FFT. This is presented together with the C code that is required by the reader to develop many of the digital signal processors studied in this and later chapters, starting with Chapter 14 which investigates the process of digital filtering in the Fourier or frequency domain. Chapter 15 introduces a statis-

tical approach to the extraction of information from noise using Bayesian estimation theory and techniques based on the application of entropy conscious approaches (e.g. maximum entropy estimation). The chapter expands this theme to a study of the problem of extrapolating the spectrum of bandlimited signals, which like so many inverse problems, is an ill-posed problem. Chapter 16 investigates the computational principles associated with processing signals in the time domain and looks at the definitions, mathematical models and applications of the Finite Impulse Response filter, the Infinite Impulse Response filter and goes on to briefly investigate issues related to processing non-stationary or time variant signals defined by a linear process. Finally, Chapter 17 investigates the world of (random) fractal signals using an approach that is based on fractional partial differential equations which has been a research interest of the author for some years. The use of fractals or self-affine models for analysing and processing signals has been around for many years and is important in that so many naturally occurring signals in nature (speech, radar, seismic, economic and biomedical signals to name but a few) exhibit fractal properties. This chapter includes a brief overview of some of the methods used for analysing and interpreting signals that have been generated by nonlinear and chaotic systems using fractal geometry. This chapter is, in effect a continuation of the material discussed in Chapter 14 which introduces methods of simulating noisy and chaotic signals.

In this book, emphasis is placed on the use of set problems which are given at the end of most chapters. These problems have been designed to instruct the reader on aspects of the material which is either complementary or exploratory in terms of advancing a specific theme further. Some of the questions are aimed at completing aspects of the material which are not covered by the text in full; other questions form the basis for material that occurs later on in the work. These problems have been designed to complement the readers 'learning curve' and should ideally be attempted after reading and comprehending the material provided in a given chapter. The questions represent an important aspect of the readers appreciation of the theoretical and practical programming techniques in order to develop an in-depth understanding of the material herein.

Full solutions are provided in the appropriate Appendix. Providing these solutions clearly adds to the bulk of the book but is of significant value to the reader in terms of completeness and necessary reference. The solutions are in the form of answers to the theoretical questions set in parts I and II and the theoretical and software development questions (given in C) provided in Parts III and IV. In addition to model solutions, the appendix contains a number of supplementary problems. These problems have been taken from a selection of the examination papers prepared by the author for assessment of students undertaking an MSc programme in 'Digital Signal Processing'. Hence, the style of the questions set are different to those of the problems at the end of a chapter. No solutions are provided to these supplementary problems.

Throughout the book, a number of examples and case studies are provided. Some of these cases studies have been designed to extend the material as required. Certain case studies are based on the authors research interests and represent new and novel approaches to some specific problems. The 'software solutions' associated with some of these examples and case studies are given in MATLAB which is an ideal prototyping environment for investigating new approaches to numerical problems. Here, MATLAB's DSP and other toolboxes are used to present prototype code that

can be investigated further as required by the reader. The graphical examples given are not extensive for reasons of page minimization but also, because it is of greater educational value for readers to investigate a processing technique by plotting and interpreting the output of software they have developed for themselves (as discussed in Part IV); a principle, which is the central kernel of this publication. Finally, each chapter contains a list of textbooks which the author has used and in some cases, has been involved in developing. The list is not extensive but is aimed at introducing 'further reading' which in most cases, either complements the material provided in a chapter or (significantly) extends it using works that are of historical as well as academic value and span an appropriate period over which the subject has been significantly developed. It was originally intended to include a list of references based on resources available on the Internet. However, such resources change so rapidly and are available so readily, that it was later considered better to leave the reader to make use of the Internet directly through the available search engines from which the reader can acquire a wealth of excellent material on this subject area and beyond.

Above all, the book attempts to provide a unified and coherent approach to the subject and to give the reader a blend of theory and practice that is, where ever possible, linked together via an appropriate underlying mathematical model. One such model that can be taken by the reader to be a fundamental underlying theme is the equation

$$s(t) = p(t) \otimes f(t) + n(t)$$

where s is the output (a recorded signal), f is the input signal to a system described by the function p and the process of convolution (denoted by the symbol $\otimes$) and n is the noise generated by the system. This is the classic time invariant linear systems model which forms the basis for a wide range of problems, from control engineering to speech processing, from active radar to biomedical signal analysis. The convolution equation is actually an integral equation. It can be considered to be a special form of the so called *inhomogeneous Fredholm equation of the second kind*, namely

$$s(t) = \int_a^b p(t,\tau) f(\tau) d\tau + n(t)$$

where $n(t)$ is a known function and a and b are the fixed points at which s satisfies boundary conditions. A more general case occurs when the integral above runs from a to t giving the so called *inhomogeneous Volterra equation of the second kind*, both cases having homogeneous forms when $n = 0$. From a theoretical point of view, we can consider this book to be on the subject of analysing and solving integral equations of this form except for the fact that the function n is not known, only its probability density function is known (at best).

The convolution process is absolutely fundamental to so many aspects of physics. It describes the smearing or blurring of one function with another which can be seen in terms of the information content of a signal being distorted by that of another. The convolution process is also of fundamental importance to statistics in that it describes the statistical distribution of a system that has evolved from combining two isolated and distinct sub-systems characterised by specific statistical distributions. Moreover, as more and more 'sub-systems' are combined (linearly), the statistics of the output

approaches a normal or Gaussian distribution. This is the so called Central Limit Theorem which is absolutely fundamental to statistical physics and the stochastic behaviour systems in general.

A principle requirement is to established the form of the function p. In the ideal case of a noise free environment (i.e. when $n(t) = 0 \forall t$) this can be achieved by inputting an impulse which is described mathematically in terms of the Dirac delta function δ (as discussed in detail later in Chapter 2). In this case,

$$s(t) = p(t) \otimes \delta(t) = p(t)$$

i.e. the output is $p(t)$. For this reason, p is often referred to as the Impulse Response Function (IPF) because it is in effect, describing the response of a system to an impulse. This fundamental model has an equivalence in frequency space, and, via the convolution theorem can be written as (with $n(t) = 0 \forall t$)

$$S(\omega) = P(\omega)F(\omega)$$

where S, P and F are the spectra of s, p and f respectively and ω is the (angular) frequency. Here, P characterises the way in which the frequency distribution of the input is transferred to the output and for this reason it is commonly referred to as the (frequency) Transfer Function (TF). In this sense, we can define an ideal system as one in which $P(\omega) = 1 \forall \omega$. The addition of noise is an important aspect of signal processing systems because it must always be assumed that no signal or signal processing system is noise free. The physical origins of noise are determined by a range of effects which vary considerably from one system to the next. In each case, suitable statistical models are required to model the noise term which in turn, can be used to design algorithms for processing signals that are robust to noise. This involves the so called extraction of information from noise which is discussed in Chapter 15 using Bayesian estimation methods.

The basic model for a signal, i.e.

$$s(t) = p(t) \otimes f(t) + n(t)$$

can be cast in terms of both piecewise continuous and generalised functions and also discrete functions or vectors. Indeed, many authors present the problem in terms of the equation

$$\mathbf{s} = L\mathbf{f} + \mathbf{n}$$

where L is a linear operator (typically a linear matrix operation) and $\mathbf{s}, \mathbf{p}$ and $\mathbf{n}$ are vectors describing the discrete or digital versions of the functions s, p and n respectively (as given on the front cover this book). Moreover, there is a close connection between the application of this model for signal processing and that associated with the general solution to physical problems specified by certain partial differential equations (PDE's) which are linear, homogeneous or inhomogeneous with homogeneous and/or inhomogeneous boundary conditions. In this case, it is often useful to determine how the system described by a PDE responds to an impulse. The solution to this problem is known as a Green's function named after the English mathematician and physicist George Green whose work dates from early nineteenth century and who provided one of most indispensable mathematical tools of the twentieth century.

However, the Green's function is essentially an impulse response function, an example being the wave generated by dropping a small stone vertically into a large pool of still water. It provides a solution that is based on a convolution process which is an underlying theme of many models in engineering, physics and statistics for example and a central component of the methods and ideas discussed in this work.

As an example, consider the process of diffusion, in which a source of material diffuses into a surrounding homogeneous medium; the material being described by some source function which is a function of both space and time and of compact support (i.e. has limited spatial extent). Physically, it is to be expected that the material will increasingly 'spread out' as time evolves and that the concentration of the material decreases further away from the source. It can be shown that a Green's function solution to the diffusion equation yields a result in which the spatial concentration of material is given by the convolution of the source function with a Gaussian function and that the time evolution of this process is governed by a similar process. Such a solution is determined by considering how the process of diffusion responds to a single point source (a space-time dependent impulse) which yields the Green's function (in this case, a Gaussian function). The connection between the basic convolution model for describing signals and systems and the Green's function solution to PDEs that describe these systems is fundamental. Thus, the convolution model that is the basis for so much of the material discussed in this work is not phenomenological but based on intrinsic methods of analysis in mathematical physics via the application of Green's function solutions. A useful example of this is given in Chapter 4 in terms of a case study which serves to highlight the role of the Fourier transform and the convolution operation in terms of the 'physics' of the propagation and scattering of waves and the signal model (a convolution integral) that this physics produces.

Part I

Signal Analysis

Chapter 1

Complex Analysis

1.1 Introduction

Complex analysis is an essential aspect of signal analysis and processing. Examples of the use of complex analysis for signal processing include: (i) The Fourier transform which provides a complex spectrum; (ii) the analytic signal, which is a complex signal with real and imaginary (quadrature) components; (iii) complex roots or 'complex zeros' for representing a signal; (iv) the amplitude and phase of a signal - essentially a complex plane or 'Argand diagram' representation of a signal. Complex or contour integration is also often used to derive analytical results relating to 'filtering' type operations on spectral (complex) functions.

In this chapter, for the sake of completeness and with regard to the portfolio of this book, a short overview of complex analysis is given primarily for those readers who are not familiar with the subject or need to re-familiarize themselves with it. The aspects taken from the field of complex analysis focus on those areas that have direct relevance to the field of signal analysis.

1.2 Complex Numbers

There are a number of ways to introduce a complex number, but, typically, we can consider the fact that within the real domain, the equation $x^2 + 1 = 0$ has no solution. We therefore specify a solution of the type

$$x = \pm i \quad \text{where} \quad i = \sqrt{-1}.$$

A complex number has the form $z = a + ib$ where a and b are real numbers. The real part of z is defined as $\mathrm{Re}[z] = a$ and the imaginary part of z as $\mathrm{Im}[z] = b$ and we consider pure real numbers as $a + i0$ and pure imaginary numbers as $0 + ib$.

1.2.1 Addition, Subtraction, Multiplication and Division

Addition and subtraction

$$(a + ib) \pm (c + id) = (a \pm c) + i(b \pm d)$$

Multiplication

$$(a+ib)(c+id) = ac + iad + ibc + ibid = ac + i(ad+bc) + i^2bd$$

Now $i^2 = (\sqrt{-1})^2 = -1$, therefore $i^2bd = -bd$ and thus,

$$(a+ib)(c+id) = (ac-bd) + i(ad+bc).$$

Division

$$\frac{a+ib}{c+id} \text{ must be expressed as } A+iB.$$

We note that $(c+id)(c-id) = c^2+d^2$ so that

$$\frac{a+ib}{c+id}\frac{c-id}{c-id} = \frac{(ac+bd)+i(bc-ad)}{c^2+d^2}.$$

Hence,

$$A = \frac{ac+bd}{c^2+d^2} \text{ and } B = \frac{bc-ad}{c^2+d^2}.$$

1.2.2 Powers of $i = \sqrt{-1}$

We note that

$$i = \sqrt{-1}, \quad i^2 = -1, \quad i^3 = -i, \quad i^4 = 1 \text{ etc.}$$

and

$$i^{-1} = \frac{1}{i} = \frac{1}{i}\frac{i}{i} = \frac{i}{-1} = -i$$

$$i^{-2} = -1, \quad i^{-3} = i \text{ etc.}$$

1.2.3 The Complex Conjugate

If $z = a+ib$, then $z^* = a-ib$ and if $z = a-ib$, then $z^* = a+ib$. z^* is called the complex conjugate of z. Note that if $a+ib = c+id$, then formally, $a = c$ and $b = d$. Also note that

$$zz^* = (x+iy)(x-iy) = x^2+y^2$$

and

$$\text{Re}[z] = \frac{1}{2}(z+z^*) = x, \quad \text{Im}[z] = -\frac{i}{2}(z-z^*) = y.$$

1.2.4 Geometrical Representation (The Argand Diagram)

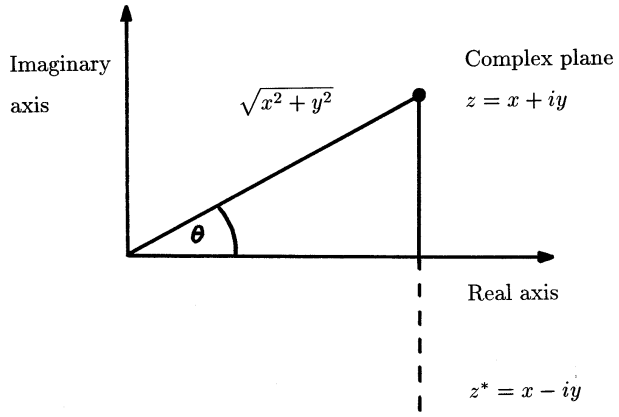


Figure 1.1: Geometrical representation of a complex number.

The Argand diagram is based on considering the real and imaginary parts of a complex number to be the positions on the vertical (imaginary) and horizontal (real) axis of a conventional graph respectively as shown in Figure 1.1. This leads directly to polar or (r, θ) notation. Let

$$r = \sqrt{x^2 + y^2} = \sqrt{zz^*} \equiv \mid z \mid,$$

$\mid z \mid$ being referred to as the modulus of z with alternative notation mod$z$. Then $x = r\cos\theta$ and $y = r\sin\theta$ giving

$$z = r(\cos\theta + i\sin\theta), \quad z^* = r(\cos\theta - i\sin\theta)$$

where

$$\theta = \tan^{-1}\frac{y}{x}.$$

The phase θ is referred to as the argument of z and is therefore sometimes donated by arg$z$. Note that arg$z \equiv \tan^{-1} y/x$ is multi-valued. We can restrict arg$z$ to a principal phase or range between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. The value of arg$z$ is then referred to as the 'principal value'.

Example Consider the complex number $z = 1 + i$. Then $r^2 = 1 + 1 = 2$ or $r = \sqrt{2}$, $\cos\theta = 1/\sqrt{2}$ and $\sin\theta = 1/\sqrt{2}$. Further, $\tan\theta = 1$ so $\theta = \tan^{-1} 1 = 45^o$ or $\pi/4$ radians. Hence, we can write

$$\text{mod}z \equiv \mid z \mid = \sqrt{2}, \quad \text{arg}z = \frac{\pi}{4}.$$

Note that in general,

$$\arg z = \theta + 2\pi n$$

where

$$n = 0, \pm 1, \pm 2, ...$$

1.2.5 De Moivre's Theorem

Suppose that

$$z_1 = r_1(\cos\theta_1 + i\sin\theta_1), \tag{1.2.1}$$
$$z_2 = r_2(\cos\theta_2 + i\sin\theta_2).$$

Then

$$z_1 z_2 = r_1 r_2(\cos\theta_1 + i\sin\theta_1)(\cos\theta_2 + i\sin\theta_2)$$
$$= r_1 r_2[(\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2) + i(\cos\theta_1\sin\theta_2 + \sin\theta_1\cos\theta_2)]$$
$$= r_1 r_2[\cos(\theta_1 + \theta_2) + i\sin(\theta_1 + \theta_2)].$$

Thus, by induction,

$$z_1 z_2 ... z_n = r_1 r_2 ... r_n[\cos(\theta_1 + \theta_2 + ... + \theta_n) + i\sin(\theta_1 + \theta_2 + ... + \theta_n)].$$

Now let

$$z_1 = z_2 = ... = z_n, \quad r_1 = r_2 = ... = r_n \quad \text{and} \quad \theta_1 = \theta_2 = ... = \theta_n.$$

Then

$$z_1^n = r_1^n[\cos(n\theta_1) + i\sin(n\theta_1)].$$

But from equation (1.2.1)

$$z_1^n = r_1^n(\cos\theta_1 + i\sin\theta_1)^n.$$

Hence,

$$(\cos\theta + i\sin\theta)^n = \cos(n\theta) + i\sin(n\theta).$$

1.2.6 The Complex Exponential

Unlike other functions, whose differentiation and/or integration yields different functions (with varying levels of complexity), the exponential function retains is functional form under differentiation and/or integration. Thus, we can define the exponential function as that function $f(x)$ say, such that

$$f'(x) \equiv \frac{d}{dx}f(x) = f(x).$$

What form should such a function have? Suppose we consider the power series

$$f(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + ...$$

Differentiating,

$$f'(x) = 0 + 1 + x + \frac{x^2}{2!} + ... + \frac{x^{n-1}}{(n-1)!} + ... = f(x)$$

as $n \to \infty$. This power series is unique with regard to this fundamental property and is given the special notation e or exp, both of which are used throughout this text. The reason why e appears so commonly throughout mathematics is because of its preservation under differentiation and/or integration. This property can be extended further if we consider an exponential of complex form $\exp(ix)$. Suppose we let

$$f(\theta) = \cos\theta + i\sin\theta. \tag{1.2.2}$$

Then

$$f'(\theta) = -\sin\theta + i\cos\theta = if(\theta).$$

Now, a solution to $f'(\theta) = if(\theta)$ is

$$f(\theta) = A\exp(i\theta) \tag{1.2.3}$$

where A is an arbitrary constant. From equation (1.2.2), $f(0) = 1$ and from equation (1.2.3), $f(0) = A$. Thus $A = 1$ and

$$\exp(i\theta) = \cos\theta + i\sin\theta.$$

Another way of deriving this result is to consider the expansion of $\exp(i\theta)$, i.e.

$$\exp(i\theta) = 1 + i\theta + \frac{i^2\theta^2}{2!} + \frac{i^3\theta^3}{3!} + \frac{i^4\theta^4}{4!} + \frac{i^5\theta^5}{5!} + ... = 1 + i\theta - \frac{\theta^2}{2!} - \frac{i\theta^3}{3!} + \frac{\theta^4}{4!} + \frac{i\theta^5}{5!} - ...$$

$$= 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - ... + i\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - ...\right) = \cos\theta + i\sin\theta.$$

Finally, we note (from De Moivre's theorem) that

$$\exp(in\theta) = (\cos\theta + i\sin\theta)^n = \cos(n\theta) + i\sin(n\theta).$$

1.3 Complex Functions

Some simple examples of complex functions are

$$z + 3, \quad \frac{1}{z}, \quad z^n, \quad \exp(z), \quad \cos z, \quad a_0 + a_1 z + a_2 z^2 + ... + a_n z^n.$$

If $f(x)$ is a function of a real variable, then $f(z)$ is a function of a complex variable. Just as the function $y = f(x)$ defines a mapping from x to y, so the function $w = f(z)$ defines a mapping from the z-plane to a w-plane as illustrated in Figure 1.2. Both planes are complex planes.

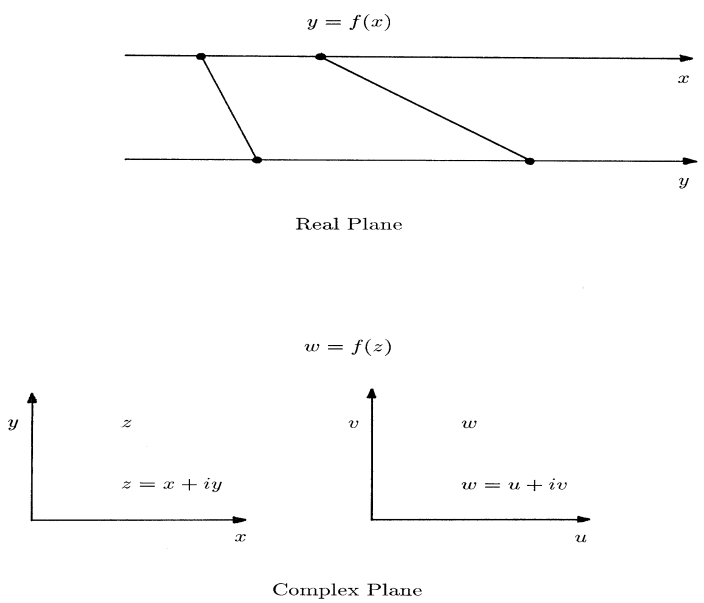


Figure 1.2: Mappings of a real function in the real plane (top) and a complex function in the complex plane (bottom).

Example Consider the function $w = z^2$, then

$$w = z^2 = (x + iy)^2 = x^2 - y^2 + 2ixy = u + iv$$

where $u = x^2 - y^2$ and $v = 2xy$. In general,

$$w = f(z) = u(x, y) + iv(x, y)$$

where $u = \mathrm{Re}[w]$ and $v = \mathrm{Im}[w]$.

1.3.1 Differentiability of a Complex Function

Consider the function $w = f(z)$, then

$$w + \delta w = f(z + \delta z)$$

and

$$\frac{\delta w}{\delta z} = \frac{f(z + \delta z) - f(z)}{\delta z}.$$

Thus

$$\frac{dw}{dz} \equiv f'(z) = \lim_{\delta z \to 0} \frac{\delta w}{\delta z}.$$

which provides a result consistent with the differentiation of a real function.

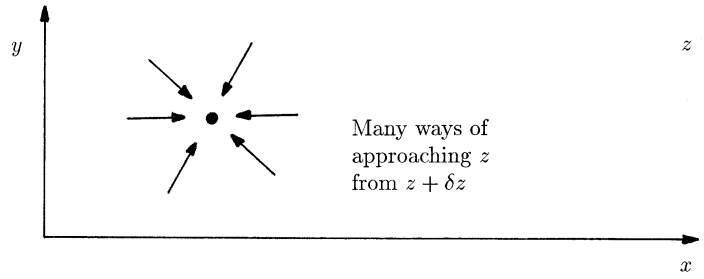


Figure 1.3: There are many way to approach a point in the complex plane. A complex function is differentiable if the same result is obtained irrespective of the path taken to the point.

However, in the complex plane, there are many ways of approaching z from $z + \delta z$. Thus, the function $w = f(z)$ is said to be differentiable if all paths leading to the point z yield the same limiting value for the ratio $\delta w/\delta z$ as illustrated in Figure 1.3.

1.3.2 The Cauchy-Riemann Equations

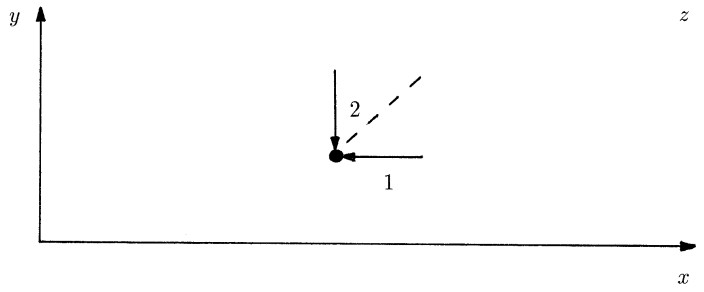


Figure 1.4: Two paths are considered: Path 1 is parallel to the x-axis and path 2 is parallel to the y-axis.

Consider a path that is parallel to the x-axis (path 1 in Figure 1.4) so that $\delta x \neq 0, \delta y = 0$ and $\delta z = \delta x$. Then

$$\frac{\delta w}{\delta z} = \frac{\delta u + i\delta v}{\delta x} = \frac{\delta u}{\delta x} + i\frac{\delta v}{\delta x}$$

and thus,

$$f'(z) = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x}.$$

Now consider a path parallel to the y-axis (path 2 in Figure 1.4) so that $\delta x = 0, \delta y \neq 0$ and $\delta z = i\delta y$. In this case,

$$\frac{\delta w}{\delta z} = \frac{\delta u + i\delta v}{i\delta y} = \frac{\delta v}{\delta y} - i\frac{\delta u}{\delta y}$$

and thus,

$$f'(z) = \frac{\partial v}{\partial y} - i\frac{\partial u}{\partial y}.$$

As $f(z)$ is assumed to be differentiable at $z = x + iy$,

$$\frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} = \frac{\partial v}{\partial y} - i\frac{\partial u}{\partial y}$$

or after equating real and imaginary parts

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \text{and} \quad \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}.$$

These are necessary conditions for the differentiability of a complex function. Now, since $\delta z = \delta x + i\delta y$,

$$dw = du + idv = \frac{\partial u}{\partial x}dx + \frac{\partial u}{\partial y}dy + i\left(\frac{\partial v}{\partial x}dx + \frac{\partial v}{\partial y}dy\right).$$

Using the Cauchy-Riemann equations

$$dw = \left(\frac{\partial u}{\partial x}dx - \frac{\partial v}{\partial x}dy\right) + i\left(\frac{\partial v}{\partial x}dx + \frac{\partial u}{\partial x}dy\right) = \frac{\partial u}{\partial x}(dx + idy) + i\frac{\partial v}{\partial x}(dx + idy)$$

$$= \left(\frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x}\right)(dx + idy) = \left(\frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x}\right)dz.$$

Hence,

$$\frac{dw}{dz} = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x}.$$

Note that the right hand side of the equation above is independent of z.

1.3.3 Analytic Functions

If $f(z)$ is differentiable at every point in a neighbourhood of the point P (see Figure 1.5), we say that it is analytic at the point P.

Example 1 Consider the function $w = zz^* = x^2 + y^2$ in which $u = x^2 + y^2$ and $v = 0$. Then

$$\frac{\partial u}{\partial x} = 2x, \quad \frac{\partial v}{\partial y} = 0,$$

$$\frac{\partial v}{\partial x} = 0, \quad \frac{\partial u}{\partial y} = 2y.$$

In this case, the Cauchy-Riemann equations are satisfied only at the origin $x = y = 0$. Hence, $|\ z\ |^2$ is differentiable at $z = 0$ only and it is not an analytic function anywhere else.

Example 2 Consider the function $w = z^* = x - iy$ where $u = x$ and $v = -y$. In this case,

$$\frac{\partial u}{\partial x} = 1, \quad \frac{\partial v}{\partial y} = -1, \quad \frac{\partial u}{\partial x} \neq \frac{\partial v}{\partial y}$$

and

$$\frac{\partial u}{\partial y} = 0, \quad \frac{\partial v}{\partial x} = 0.$$

Therefore, z^* is not differentiable anywhere.

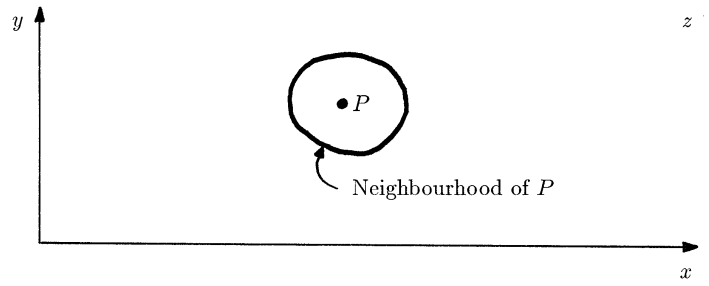


Figure 1.5: The neighbourhood of a point P in the complex plane is the region in the complex plane that (completely) surrounds that point.

Example 3 For the function

$$w = \frac{1}{z} = \frac{1}{x + iy} = \frac{x - iy}{x^2 + y^2},$$

$$u = \frac{x}{x^2 + y^2} \quad \text{and} \quad v = -\frac{y}{x^2 + y^2}.$$

In this case,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}.$$

Therefore, the Cauchy-Riemann equations are satisfied everywhere except at $z = 0$. Hence, $1/z$ is analytic at all points of the z-plane except at $z = 0$.

Example 4 Consider the function

$$w = \exp(z) = \exp(x + iy) = \exp(x)(\cos y + i \sin y),$$

so that

$$u = \exp(x) \cos y, \quad v = \exp(x) \sin y$$

and

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}.$$

In this example, the Cauchy-Riemann equations are satisfied everywhere in the z-plane. Hence, $\exp(z)$ is an analytic function at all points in the complex plane.

Note that $|\exp(z)| = \exp(x)$ and $\arg[\exp(z)] = y + 2\pi n, \quad n = 0, \pm 1, \pm 2, ...$ Thus, $\exp(z)$ is a periodic function with a period of 2π.

1.3.4 Some Important Results

For analytic functions $w_1 = f_1(z)$ and $w_2 = f_2(z)$:

$$\frac{d}{dz}(w_1 + w_2) = \frac{dw_1}{dz} + \frac{dw_2}{dz},$$

$$\frac{d}{dz}(w_1 w_2) = w_1 \frac{dw_2}{dz} + w_2 \frac{dw_1}{dz},$$

$$\frac{d}{dz}\left(\frac{1}{w}\right) = -\frac{1}{w^2}\frac{dw}{dz}, \quad w \neq 0$$

and

$$\frac{dz}{dw} = \frac{1}{\frac{dw}{dz}}, \quad \frac{dw}{dz} \neq 0.$$

If $w = w(\xi)$ and $\xi = \xi(z)$, then

$$\frac{dw}{dz} = \frac{dw}{d\xi}\frac{d\xi}{dz}.$$

Also

$$\frac{d}{dz}z^n = nz^{n-1}$$

where n is an integer.

1.4 Complex Integration

The integral of a complex function is denoted by

$$I = \int_C f(z)dz$$

where

$$f(z) = u(x, y) + iv(x, y), \quad dz = dx + idy$$

and C denotes the 'path' of integration in the complex plane (see Figure 1.6). Hence,

$$I = \int_C (u + iv)(dx + idy) = \int_C (udx - vdy) + i\int_C (udy + vdx)$$

with the fundamental results

$$\int_C [f(z) + g(z)]dz = \int_C f(z)dz + \int_C g(z)dz,$$

$$\int_C kf(z)dz = k\int_C f(z)dz$$

and

$$\int_{C_1+C_2} f(z)dz = \int_{C_1} f(z)dz + \int_{C_2} f(z)dz$$

as illustrated in Figure 1.7.

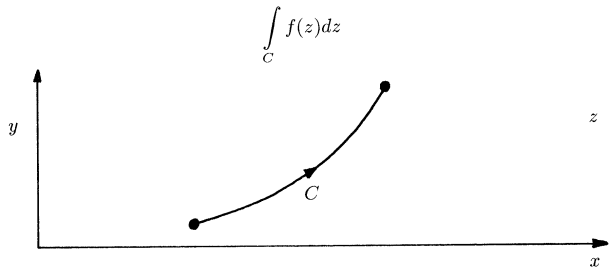


Figure 1.6: Integration in the complex plane is taken to be along a path C.

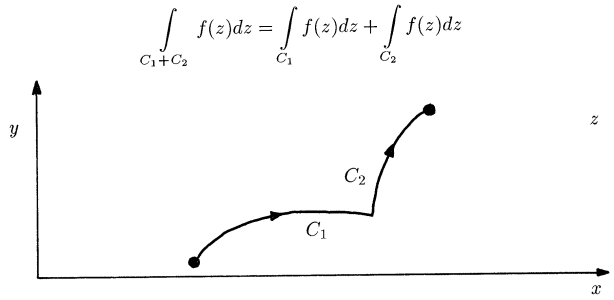


Figure 1.7: Integration over two paths C_1 and C_2 in the complex plane is given by the sum of the integrals along each path.

Example 1 Integrate $f(z) = 1/z$ from 1 to z along a path C that does not pass through $z = 0$ (see Figure 1.8), i.e. evaluate the integral

$$I = \int_C \frac{dz}{z}.$$

Let $z = r\exp(i\theta)$, so that $dz = dr\exp(i\theta) + r\exp(i\theta)id\theta$. Then

$$I = \int_C \frac{\exp(i\theta)(dr + rid\theta)}{r\exp(i\theta)} = \int_C \left(\frac{dr}{r} + id\theta\right)$$

$$= \int_1^{|z|} \frac{dr}{r} + i \int_0^{\theta+2\pi n} d\theta$$

where n is the number of time that the path C encircles the origin in the positive sense, $n = 0, \pm 1, \pm 2, ...$ Hence

$$I = \ln \mid z \mid + i(\theta + 2\pi n) = \ln z.$$

Note, that substitutions of the type $z = r\exp(i\theta)$ are a reoccurring theme in the evaluation and analysis of complex integration.

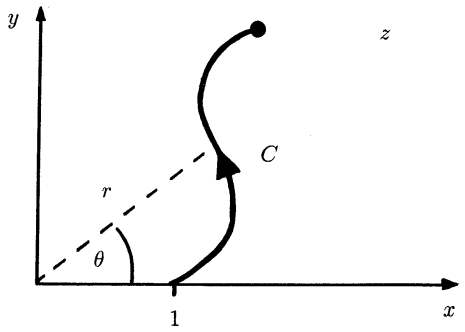


Figure 1.8: Illustration of the path of integration C for integrating $1/z$ from 1 to z.

Example 2 Integrate $f(z) = 1/z$ around $z = \exp(i\theta)$ where $0 \leq \theta \leq 2\pi$ (see Figure 1.9). Here,

$$dz = \exp(i\theta)id\theta$$

and

$$I = \int_C \frac{1}{z}dz = \int_0^{2\pi} id\theta = 2\pi i.$$

The path of integration C in this case is an example of a contour. A Contour is a simple closed path. The domain or region of the z-plane through which C is chosen must be simply connected (no singularities or other non-differentiable features). Contour integrals are a common feature of complex analysis and will be denoted by $\oint$ from here on.

Important Result

$$\oint_C \frac{dz}{z^{n+1}} = 0, \quad n > 0.$$

Proof Let $z = r\exp(i\theta)$, then $dz = r\exp(i\theta)id\theta$ and

$$\int_C \frac{dz}{z^{n+1}} = \int_0^{2\pi} \frac{r\exp(i\theta)id\theta}{r^{n+1}\exp[i(n+1)\theta]} = \frac{i}{r^n}\int_0^{2\pi} \exp(-in\theta)d\theta$$

$$= -\frac{i}{r^n}\frac{1}{in}\left[\exp(-in\theta)\right]_0^{2\pi} = -\frac{1}{r^n}\frac{1}{n}[\exp(-2\pi in) - 1] = 0.$$

Note, that n must be an integer > 0.

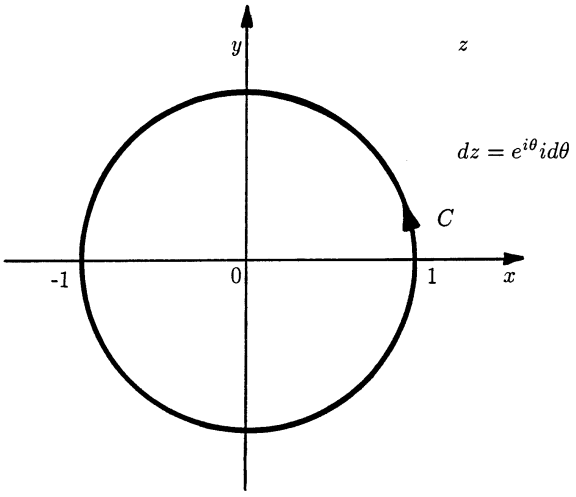


Figure 1.9: Integration of $1/z$ around $z = \exp(i\theta), \quad 0 \leq \theta \leq 2\pi$.

1.4.1 Green's Theorem in the Plane

Theorem If S is a closed region in the $x - y$ plane bounded by a simple closed curve C and if P and Q are continuous function of x and y having continuous derivatives in S, then

$$\oint_C (Pdx + Qdy) = \iint_S \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy$$

Proof Consider the accompanying diagram (see Figure 1.10).

Let curve ACB be described by the equation

$$y = Y_1(x).$$

Let curve BDA be described by the equation

$$y = Y_2(x).$$

Let curve DAC be described by the equation

$$x = X_1(y).$$

Let curve CBD be described by the equation

$$x = X_2(y).$$

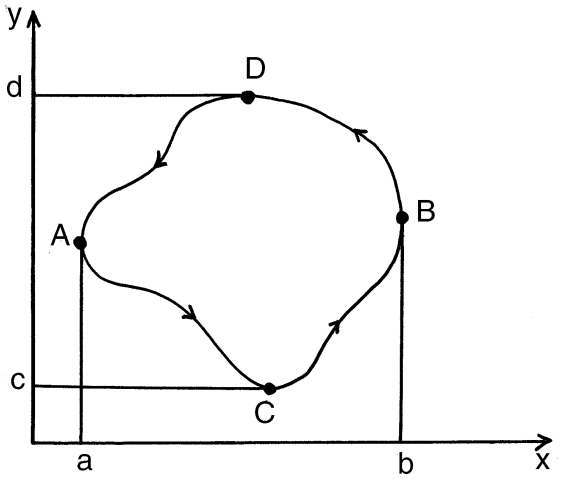


Figure 1.10: Path of integration for the proof of Green's theorem in the plane.

Then,

$$\iint\limits_S \frac{\partial P}{\partial y} dxdy = \int\limits_a^b \left(\int\limits_{Y_1}^{Y_2} \frac{\partial P}{\partial y} dy \right) dx = \int\limits_a^b [P(x, Y_2) - P(x, Y_1)] dx$$

$$= \int\limits_a^b P(x, Y_2) dx - \int\limits_a^b P(x, Y_1) dx = - \int\limits_a^b P(x, Y_1) dx - \int\limits_b^a P(x, Y_2) dx = - \oint\limits_C P dx.$$

Similarly,

$$\iint_S \frac{\partial Q}{\partial x} dxdy = \int_c^d \left(\int_{X_1}^{X_2} \frac{\partial Q}{\partial x} dx \right) dy = \int_c^d [Q(X_2, y) - Q(X_1, y)] dy$$

$$= \int_c^d Q(X_2, y) dy - \int_c^d Q(X_1, y) dy = \int_c^d Q(X_2, y) dy + \int_d^c Q(X_1, y) dy = \oint_C Q dy.$$

1.4.2 Cauchy's Theorem

Theorem If $f(z)$ is analytic and $f'(z)$ is continuous in a simply connected region R, and C is a simple closed curve lying within R, then

$$\oint_C f(z) dz = 0.$$

Proof

$$\oint_C f(z) dz = \oint_C (udx - vdy) + i \oint_C (udy + vdx).$$

Using Green's theorem in the plane, i.e.

$$\oint_C (Pdx + Qdy) = \iint_S \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dxdy$$

where P and Q have continuous partial derivatives, we get

$$\oint_C f(z) dz = \iint_S \left(-\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) dxdy + i \iint_S \left(\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} \right) dxdy.$$

But from the Cauchy-Riemann equations

$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}$$

and

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}.$$

Hence,

$$\oint_C f(z) dz = 0.$$

Corollary 1 If C_1 and C_2 are two paths joining points a and z in the z-plane (see Figure 1.11) then, provided $f(z)$ is analytic at all points on C_1 and C_2 and between C_1 and C_2,

$$\int_{a|C_1\uparrow}^{z} f(z) dz = \int_{a|C_2\uparrow}^{z} f(z) dz.$$

Here, $\uparrow$ denotes that the path of integration is in an anti-clockwize direction and $\downarrow$ is taken to denote that the path of integration is in a clockwize direction. This result comes from the fact that the path taken is independent of the integral since from Cauchy's theorem

$$\left(\int_{a|C_1\uparrow}^{z} + \int_{z|C_2\downarrow}^{a} \right) f(z)dz = 0$$

and thus,

$$\int_{a|C_1\uparrow}^{z} f(z)dz = -\int_{z|C_2\downarrow}^{a} f(z)dz = \int_{a|C_2\uparrow}^{z} f(z)dz.$$

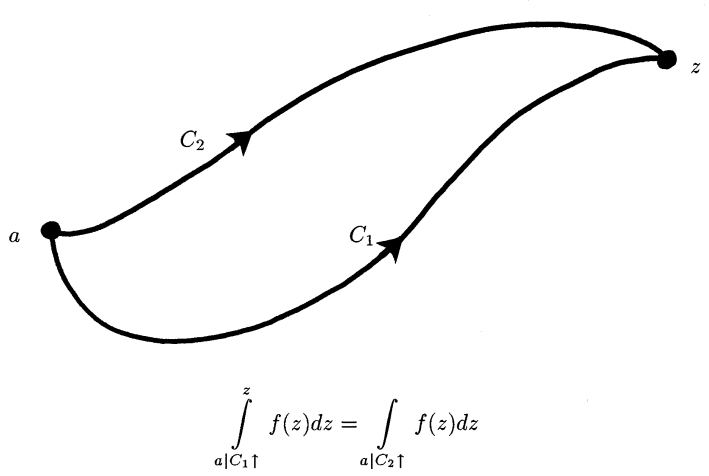


Figure 1.11: An integral is independent of the path that is taken in the complex plane.

Corollary 2 If $f(z)$ has no singularities in the annular region between the contours C_1 and C_2, then

$$\oint_{C_1\uparrow} f(z)dz = \oint_{C_2\uparrow} f(z)dz.$$

We can show this result by inserting a cross-cut between C_1 and C_2 to produce a single contour at every point within which $f(z)$ is analytic (Figure 1.12). Then, from Cauchy's theorem

$$\int_{C_1\uparrow} f(z)dz + \int_{a}^{b} f(z)dz - \int_{C_2\downarrow} f(z)dz + \int_{c}^{d} f(z)dz = 0.$$

Rearranging,

$$\int_{C_1\uparrow} f(z)dz + \int_a^b f(z)dz - \int_d^c f(z)dz = \int_{C_2\uparrow} f(z)dz.$$

But

$$\int_a^b f(z)dz - \int_d^c f(z)dz = 0$$

and hence,

$$\oint_{C_1\uparrow} f(z)dz = \oint_{C_2\uparrow} f(z)dz.$$

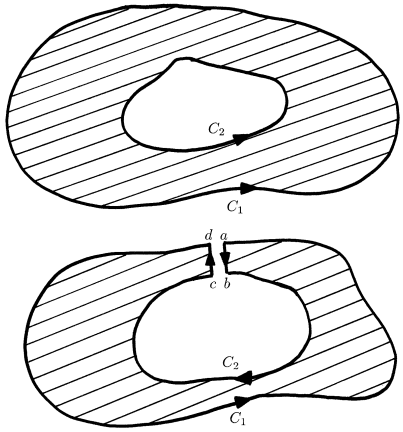


Figure 1.12: Two contours C_1 and C_2 made continuous through a cross-cut.

1.4.3 Defining a Contour

A contour can be defined around any number of points in the complex plane. Thus, if we consider three points z_1, z_2 and z_3, for example, then the paths Γ_1, Γ_2 and Γ_3 respectively can be considered to be simply connected as shown in Figure 1.13. Thus, we have

$$\oint_C f(z)dz = \int_{\Gamma_1} f(z)dz + \int_{\Gamma_2} f(z)dz + \int_{\Gamma_3} f(z)dz.$$

Example

$$I = \oint_C \frac{dz}{z} = \int_\Gamma \frac{dz}{z}$$

where Γ is a circle which can be represented in the form $z = r\exp(i\theta), \quad 0 \leq \theta \leq 2\pi$. Then, $dz = r\exp(i\theta)id\theta$ and

$$\int_{\Gamma} \frac{dz}{z} = \int_{0}^{2\pi} id\theta = 2\pi i.$$

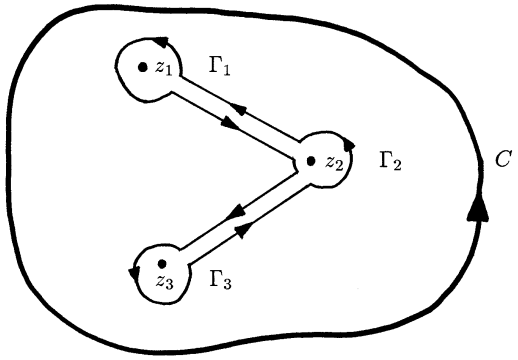


Figure 1.13: Defining the contour C which encloses three points in the complex plane.

1.4.4 Example of the Application of Cauchy's Theorem

Let us consider the evaluation of the integral

$$\int_{0}^{\infty} \frac{\sin x}{x} dx.$$

Consider the complex function $\exp(iz)/z$ which has one singularity at $z = 0$. Note that we have chosen the function $\exp(iz)/z$ because it is analytic everywhere on and within the contour illustrated in Figure 1.14. By Cauchy's theorem,

$$\int_{C_R} \frac{\exp(iz)}{z} dz + \int_{-R}^{-r} \frac{\exp(ix)}{x} dx + \int_{C_r} \frac{\exp(iz)}{z} dz + \int_{r}^{R} \frac{\exp(ix)}{x} dx = 0.$$

We now evaluate the integrals along the real axis:

$$\int_{r}^{R} \frac{\exp(ix)}{x} dx + \int_{-R}^{-r} \frac{\exp(ix)}{x} dx = \int_{r}^{R} \frac{\exp(ix)}{x} dx - \int_{r}^{R} \frac{\exp(-ix)}{x} dx = \int_{r}^{R} \frac{\exp(ix) - \exp(-ix)}{x} dx$$

$$= 2i \int_{r}^{R} \frac{\sin x}{x} dx = 2i \int_{0}^{\infty} \frac{\sin x}{x} dx \quad \text{as} \quad R \to \infty \quad \text{and} \quad r \to 0.$$

Evaluating the integral along C_r:

$$\int_{C_r} \frac{\exp(iz)}{z} dz = \int_{\pi}^{0} \exp[ir(\cos\theta + i\sin\theta)] id\theta \quad [\text{with} \quad z = r\exp(i\theta)]$$

$$= -\int_{0}^{\pi} \exp[ir(\cos\theta + i\sin\theta) id\theta = -\int_{0}^{\pi} \exp(0) id\theta \quad \text{as} \quad r \to 0 \quad (\text{i.e.} -i\pi).$$

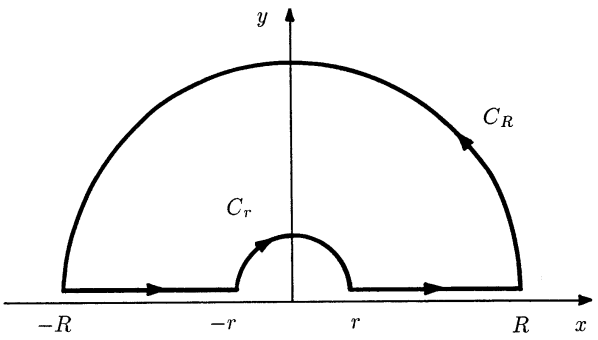


Figure 1.14: Contour used for evaluating the integral of $\sin(x)/x, x \in [0, \infty)$.

Evaluating the integral along C_R:

$$\int_{C_R} \frac{\exp(iz)}{z} dz = \int_{0}^{\pi} \exp[iR(\cos\theta + i\sin\theta) id\theta \quad [\text{with} \quad z = R\exp(i\theta)]$$

$$= \int_{0}^{\pi} \exp(iR\cos\theta)\exp(-R\sin\theta) id\theta = 0 \quad \text{as} \quad R \to \infty.$$

Combining the results,

$$2i \int_{0}^{\infty} \frac{\sin x}{x} dx - i\pi = 0.$$

Hence,

$$\int_{0}^{\infty} \frac{\sin x}{x} dx = \frac{\pi}{2}.$$

1.4.5 Cauchy's Integral Formula

Theorem If $f(z)$ is analytic in a simply connected region R and C is a contour that lies within R and encloses point z_0, then

$$\oint_C \frac{f(z)}{z - z_0} dz = 2\pi i f(z_0).$$

Proof Consider a small circle Γ with center z_0 and radius r, lying entirely within C (see Figure 1.15).

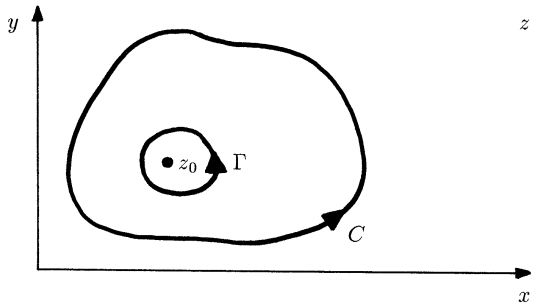


Figure 1.15: Circular contour Γ within arbitrary contour C enclosing a point z_0.

Then

$$I = \oint_C \frac{f(z)}{z - z_0} dz = \oint_\Gamma \frac{f(z)}{z - z_0} dz.$$

Let $z - z_0 = r\exp(i\theta)$, then $dz = r\exp(i\theta) i d\theta$ and

$$I = \int_0^{2\pi} \frac{f[z_0 + r\exp(i\theta)]}{r\exp(i\theta)} r\exp(i\theta) i d\theta = i\int_0^{2\pi} f[z_0 + r\exp(i\theta)] d\theta = i\int_0^{2\pi} f(z_0) d\theta \quad \text{as} \quad r \to 0$$

$$= 2\pi i f(z_0).$$

Hence,

$$f(z_0) = \frac{1}{2\pi i} \oint_C \frac{f(z)}{z - z_0} dz.$$

1.5 Cauchy's Residue Theorem

Cauchy's residue theorem is fundamental to complex analysis and is used routinely in the evaluation of integrals. We start with some important preliminaries. If $f(z)$ is analytic at z_0 it may be expanded as a power series in $(z - z_0)$, i.e. as a Taylor series,

$$f(z) = a_0 + a_1(z - z_0) + a_2(z - z_0)^2 + ... + a_n(z - z_0)^n$$

where

$$a_n = \frac{1}{n!} f^{(n)}(z_0).$$

To expand a function $f(z)$ which is analytic (apart from singular points) about one of its singular points z_0 we can write

$$f(z) = a_0 + a_1(z - z_0) + a_2(z - z_0)^2 + ... + a_n(z - z_0)^n$$

$$+ \frac{b_1}{z - z_0} + \frac{b_2}{(z - z_0)^2} + ... + \frac{b_n}{(z - z_0)^n}.$$

This series has two component parts, i.e. $f(z)$ = analytic part + principal part. Expanding the function in this way allows us to develop the residue theorem.

1.5.1 Poles and Residues

If the principal part terminates at the term $\frac{b_n}{(z-z_0)^n}$, $f(z)$ is said to have a pole of order n at z_0. The coefficient b_1 is called the residue of $f(z)$ at z_0.

Examples

$$f(z) = \frac{3}{z}$$

$z = 0$ is a simple pole and the residue is 3.

$$f(z) = \frac{1}{z^2}$$

$z = 0$ is a double pole with residue is 0.

$$f(z) = \sin z + \frac{5}{z} - \frac{2}{z^3}$$

$z = 0$ is a pole of order 3; the residue is 5.

$$f(z) = \frac{2i}{z^2 - 1} = \frac{i}{z - 1} - \frac{i}{z + 1}$$

has a simple pole at $z = 1$ with a residue i and a simple pole at $z = -1$ with residue $-i$.

1.5.2 Residues at Simple Poles

To find the residue at a simple pole z_0 we write

$$f(z) = g(z) + \frac{b_1}{z - z_0}.$$

Then

$$(z - z_0)f(z) = (z - z_0)g(z) + b_1$$

giving

$$b_1 = \lim_{z \to z_0} [(z - z_0)f(z)].$$

Example

$$f(z) = \frac{1}{(z-1)(z+2)(z+3)}$$

has simple poles at

$$z = 1, \quad z = -2, \quad z = -3$$

with residues

$$\frac{1}{12}, \quad -\frac{1}{3}, \quad \frac{1}{4}.$$

1.5.3 Residues at Poles of Arbitrary Order

Residue at a double pole

$$f(z) = g(z) + \frac{b_1}{(z-z_0)} + \frac{b_2}{(z-z_0)^2}$$

or

$$(z-z_0)^2 f(z) = (z-z_0)^2 g(z) + (z-z_0)b_1 + b_2.$$

Now

$$\frac{d}{dz}[(z-z_0)^2 f(z)] = \frac{d}{dz}[(z-z_0)^2 g(z)] + b_1.$$

Hence,

$$b_1 = \lim_{z \to z_0} \frac{d}{dz}[(z-z_0)^2 f(z)].$$

Residue at a triple pole

$$f(z) = g(z) + \frac{b_1}{(z-z_0)} + \frac{b_2}{(z-z_0)^2} + \frac{b_3}{(z-z_0)^3}$$

or

$$(z-z_0)^3 f(z) = (z-z_0)^3 g(z) + b_1(z-z_0)^2 + b_2(z-z_0) + b_3.$$

Now

$$\frac{d^2}{dz^2}[(z-z_0)^3 f(z)] = \frac{d^2}{dz^2}[(z-z_0)^3 g(z)] + 2b_1.$$

Hence

$$b_1 = \frac{1}{2} \lim_{z \to z_0} \frac{d^2}{dz^2}[(z-z_0)^3 f(z)].$$

Residue at an n-order pole

From the results above, by induction, the residue at a pole of order n is given by

$$b_1 = \frac{1}{(n-1)!} \lim_{z \to z_0} \frac{d^{(n-1)}}{dz^{(n-1)}}[(z-z_0)^n f(z)].$$

1.5.4 The Residue Theorem

Theorem If $f(z)$ is analytic in and on a closed contour C except for a finite number of poles within C, then

$$\oint_C f(z)dz = 2\pi i \sum_i R_i$$

where $\sum_i R_i$=sum of residues of $f(z)$ at those poles which lie in C.

Proof Make cuts and small circles Γ_i around each pole z_i which lies within the contour C (see Figure 1.16).

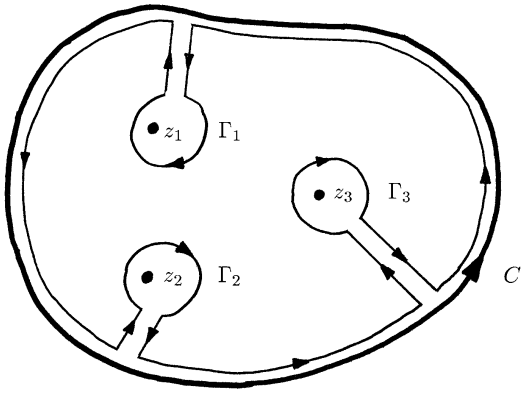


Figure 1.16: Cuts of small circles Γ_i around poles at z_i which lie within the contour C.

Since $f(z)$ is analytic at all points within the new contour, by Cauchy's theorem

$$\int_{C\uparrow} f(z)dz + \sum_i \int_{\Gamma_i\downarrow} f(z)dz = 0$$

or

$$\int_{C\uparrow} f(z)dz = \sum_i \int_{\Gamma_i\uparrow} f(z)dz.$$

Thus,

$$\oint_C f(z)dz = \sum_i \oint_{\Gamma_i} f(z)dz$$

If the point z_i is a pole of order n, then on the small circle Γ_i

$$f(z) = g(z) + \frac{b_1}{z - z_i} + \frac{b_2}{(z - z_i)^2} + ... + \frac{b_n}{(z - z_i)^n}.$$

By Cauchy's Theorem

$$\oint_{\Gamma_i} g(z)dz = 0$$

and

$$\oint_{\Gamma_i} \frac{dz}{(z-z_i)^n} = 0, \quad n = 2, 3, 4, ...$$

Hence,

$$\oint_{\Gamma_i} f(z)dz = b_1 \oint_{\Gamma_i} \frac{dz}{z-z_i} = 2\pi i b_1 \equiv 2\pi i R_i.$$

where R_i is the residue associate with the pole $z = z_i$. Repeating this analysis for all poles within C, we have

$$\oint_C f(z)dz = 2\pi i \sum_i R_i$$

1.5.5 Evaluation of Integrals using the Residue Theorem

Example 1 Evaluate the contour integral

$$\oint_C \frac{1+4z-3z^2}{z-1}dz$$

around:

(i) a circle

$$x^2 + y^2 = \frac{1}{4};$$

(ii) an ellipse

$$\frac{x^2}{9} + \frac{y^2}{4} = 1.$$

The integral has a simple pole at $z = 1$ and the residue at this pole is

$$\lim_{z\to 1}\left(\frac{(z-1)(1+4z-3z^2)}{z-1}\right) = 2.$$

The pole is outside the circle $\mid z \mid = \frac{1}{2}$ and thus,

$$\oint_{|z|=\frac{1}{2}} \frac{1+4z-3z^2}{z-1}dz = 0.$$

The pole is inside the ellipse, hence,

$$\oint_{\text{ellipse}} \frac{1+4z-3z^2}{z-1}dz = 2\pi i \times 2 = 4\pi i.$$

Example 2 Evaluate

$$\oint_C \frac{\sin(\pi z)}{(z-1)^4} dz$$

where C is the circle $\mid z \mid = 3$ (or any other circle that encloses $z = 1$). There is a pole of order 4 at $z = 1$. The residue in this case is

$$\frac{1}{3!} \lim_{z \to 1} \frac{d^3}{dz^3} \sin(\pi z) = -\frac{1}{3!} \lim_{z \to 1} \pi^3 \cos(\pi z) = \frac{\pi^3}{6}.$$

By Cauchy's theorem

$$\oint_C \frac{\sin(\pi z)}{(z-1)^4} dz = 2\pi i \times \frac{\pi^3}{6} = i\frac{\pi^4}{3}.$$

Example 3 Evaluate

$$\int_{-\infty}^{\infty} \frac{dx}{x^2+1}.$$

Consider

$$\frac{1}{z^2+1} = \frac{1}{(z+i)(z-i)}$$

which has poles at $\pm i$ and the contour illustrated in Figure 1.17. The residue at $z = i$ is

$$\lim_{z \to i} \left(\frac{z-i}{(z+i)(z-i)} \right) = \frac{1}{2i}$$

and thus

$$\int_{-R}^{R} \frac{dx}{x^2+1} + \int_{\Gamma} \frac{dz}{z^2+1} = 2\pi i \times \frac{1}{2i} = \pi.$$

The integral over Γ can be written in the form [with $z = R\exp(i\theta)$]

$$\int_{\Gamma} \frac{R\exp(i\theta) i d\theta}{R^2 \exp(2i\theta) + 1} \to 0 \quad \text{as} \quad R \to \infty.$$

Hence

$$\int_{-\infty}^{\infty} \frac{dx}{x^2+1} = \pi$$

and (since the integrand is symmetric about $x = 0$)

$$\int_{0}^{\infty} \frac{dx}{x^2+1} = \frac{\pi}{2}.$$

Example 4 Evaluate

$$\int_0^\infty \frac{\sqrt{x}}{1+x^2}dx.$$

Consider the complex function

$$f(z) = \frac{\sqrt{z}}{1+z^2}$$

and the contour illustrated in Figure 1.18. Let $z = R\exp(i\theta)$, then on the path just above the cut, $\theta = 0$ and

$$f(z) = \frac{\sqrt{R}\exp(0)}{1+R^2\exp(0)} = \frac{\sqrt{x}}{1+x^2}.$$

On the path just below the cut, $\theta = 2\pi$ and

$$f(z) = \frac{\sqrt{R}\exp(\pi i)}{1+R^2\exp(4\pi i)} = -\frac{\sqrt{x}}{1+x^2}.$$

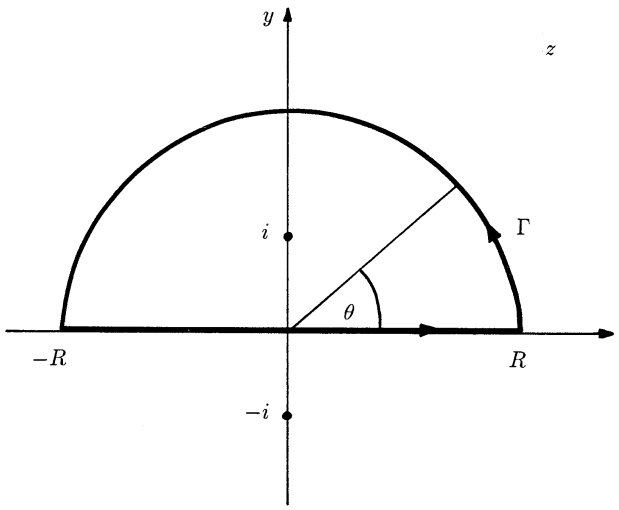


Figure 1.17: Contour used to evaluate the integral of the function $(1+x^2), x \in (-\infty, \infty)$.

The poles inside the contour exist at $z = \pm i$. The residue at $z = i$ is

$$\lim_{z\to i}\left(\frac{(z-i)\sqrt{z}}{(z+i)(z-i)}\right) = \frac{\sqrt{i}}{2i} = \frac{\exp(i\pi/4)}{2i} = \frac{\frac{1}{\sqrt{2}}+\frac{i}{\sqrt{2}}}{2i} = \frac{\sqrt{2}}{4}(1-i).$$

The residue at $z = -i$ is

$$\lim_{z\to -i}\left(\frac{(1+i)\sqrt{z}}{(z-i)(z+i)}\right) = \frac{\sqrt{-i}}{-2i} = \frac{\exp(3i\pi/4)}{-2i} = \frac{\sqrt{2}}{4}(-1-i).$$

By Cauchy's residue theorem

$$\oint_{\Gamma} f(z)dz = 2\pi i \sum_i R_i = 2\pi i \times \frac{\sqrt{2}}{4}(1-i-1-i) = 2\pi i \frac{\sqrt{2}}{4}(-2i) = \sqrt{2}\pi.$$

Hence,

$$\int_r^R \frac{\sqrt{x}}{1+x^2}dx + \int_{\Gamma} \frac{\sqrt{z}}{1+z^2}dz + \int_R^r \frac{-\sqrt{x}}{1+x^2}dx + \int_{\gamma} \frac{\sqrt{z}}{1+z^2}dz = \sqrt{2}\pi$$

$$\implies 2\int_r^R \frac{\sqrt{x}}{1+x^2}dx + \int_0^{2\pi} \frac{\sqrt{R}e^{i\theta/2}Re^{i\theta}id\theta}{1+R^2e^{2i\theta}} + \int_{2\pi}^0 \frac{\sqrt{r}e^{i\theta/2}re^{i\theta}id\theta}{1+r^2e^{2i\theta}} = \sqrt{2}\pi.$$

Finally,

$$\frac{\sqrt{R}R}{1+R^2} \to 0 \quad \text{as} \quad R \to \infty \quad \text{and} \quad \frac{\sqrt{r}r}{1+r^2} \to 0 \quad \text{as} \quad r \to 0$$

and hence

$$\int_0^{\infty} \frac{\sqrt{x}}{1+x^2}dx = \frac{\sqrt{2}\pi}{2} = \frac{\pi}{\sqrt{2}}.$$

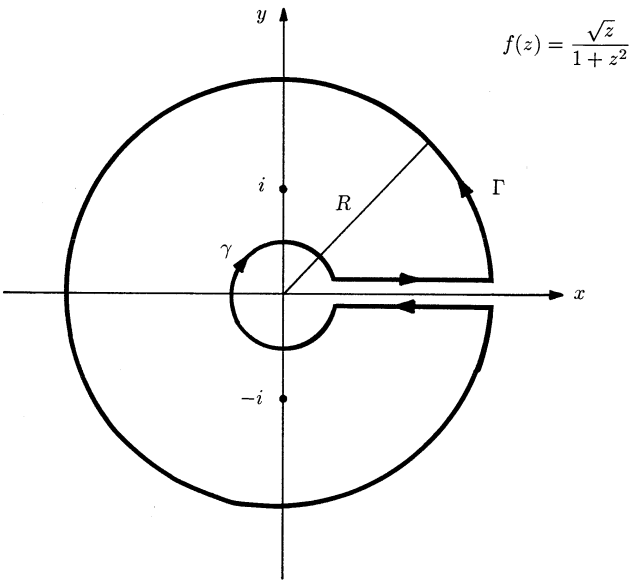


Figure 1.18: Contour used to evaluate the integral of the function $\sqrt{x}/(1+x^2), x \in [0, \infty)$.

1.5.6 Evaluation of Trigonometric Integrals

To evaluate integrals of the form

$$\int_0^{2\pi} f(\cos\theta, \sin\theta) d\theta$$

we can substitute $z = \exp(i\theta)$, so that

$$\cos\theta = \frac{1}{2}\left(z + \frac{1}{z}\right)$$

$$\sin\theta = \frac{1}{2i}\left(z - \frac{1}{z}\right)$$

$$d\theta = \frac{dz}{iz}$$

and integrate round a unit circle $\mid z \mid = 1$.

Example Evaluate

$$I = \int_0^{2\pi} \frac{d\theta}{2 + \cos\theta}.$$

Consider

$$I = \oint_C \frac{dz/iz}{2 + \frac{1}{2}\left(z + \frac{1}{z}\right)} = \frac{2}{i} \oint_C \frac{dz}{z^2 + 4z + 1}.$$

Now, $z^2 + 4z + 1 = 0$ when $z = -2 \pm \sqrt{3}$ and the residue at $z = -2 + \sqrt{3}$ is $\frac{1}{2\sqrt{3}}$. Hence,

$$I = \frac{2}{i} \times 2\pi i \times \frac{1}{2\sqrt{3}} = \frac{2\pi}{\sqrt{3}}.$$

1.6 Summary of Important Results

Complex numbers

$$z = x + iy; \quad x = \mathrm{Re}[z], \quad y = \mathrm{Im}z$$

Complex conjugate

$$z^* = x - iy$$

Polar representation

$$z = r\cos\theta + ir\sin\theta, \quad r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

De Moivre's Theorem

$$(\cos\theta + i\sin\theta)^n = \cos(n\theta) + i\sin(n\theta)$$

Complex exponential

$$\exp(i\theta) = \cos\theta + i\sin\theta$$

Complex functions

$w = f(z)$ is a mapping from the $z = x + iy$ plane to the $w = u + iv$ plane.

Cauchy-Riemann equations

The necessary conditions for the differentiability of a complex function, i.e.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$

and

$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}.$$

Analytic functions

Functions $f(z)$ that are differentiable at everywhere in a neighbourhood of the point z, i.e. complex functions that satisfy the Cauchy-Riemann equations.

Complex integrals

$$I = \int_C f(z)dz$$

where C defines a path in the complex plane; a contour if C is a closed path where the notation is

$$I = \oint_C f(z)dz.$$

Green's theorem in the plane

$$\oint_C (Pdx + Qdy) = \iint_S \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy$$

Cauchy's theorem

$$\oint_C f(z) = 0$$

where $f(z)$ is analytic in a simply connected region.

Cauchy's integral formula

$$\oint_C \frac{f(z)}{z - z_0}dz = 2\pi i f(z_0)$$

Cauchy's residue theorem

$$\oint_C f(z)dz = 2\pi i \sum_i R_i$$

where R_i are the residues of $f(z)$ at the poles which lie within C.

1.7 Further Reading

There are a wide range of text books covering complex analysis that have been published over many years. The following texts provide some examples covering these years.

- Copson E T, *An Introduction to the Theory of Functions of a Complex Variable*, Oxford University Press, 1935.
- Titchmarsh E C, *The Theory of Functions*, Oxford University Press, 1939.
- Knopp K, *Theory of Functions*, Dover Publications, 1945.
- Apostol T M, *Mathematical Analysis*, Addison-Wesley, 1957.
- Fuchs B A and Shabat B V, *Functions of a Complex Variable*, Pergamon Press, 1964.
- Silverman R A, *Complex Analysis with Applications*, Prentice -Hall, 1974.
- Churchill R V and Brown J W, *Complex Variables and Applications*, McGraw-Hill, 1984.
- Paliouras J D and Meadows D S, *Complex Variables for Scientists and Engineers*, Macmillan Publishing Company, 1990.

1.8 Problems

1.1 Write the following complex numbers z in the form $z = a + ib$ where $a = \mathrm{Re}[z]$ and $b = \mathrm{Im}[z]$:

$$\text{(i)}\ \frac{1+i}{2-i},\quad \text{(ii)}\ \frac{1}{i^5},\quad \text{(iii)}\ \frac{4-5i}{(2-3i)^2},$$

$$\text{(iv)}\ \frac{(1+2i)(1+3i)(3+i)}{1-3i},\quad \text{(v)}\ \left(-\frac{1}{2}+i\frac{\sqrt{3}}{2}\right)^2.$$

1.2 Plot the following complex numbers in the z-plane: (i) z, (ii) z^*, (iii) $z + z^*$, (iv) $\sqrt{(zz^*)}$, (v) iz.

1.3 If z_1 and z_2 are complex numbers, prove that

$$(z_1 + z_2)^* = z_1^* + z_2^*$$

and that

$$(z_1 z_2)^* = z_1^* z_2^*.$$

1.4 Find the moduli and the arguments of the following: (i) $1 - i\sqrt{3}$, (ii) $\exp(i\pi/2) + \sqrt{2}\exp(i\pi/4)$, (iii) $(1+i)\exp(i\pi/6)$, (iv) $z_1 z_2$, (v) z_1/z_2 where $z_1 = 2\exp(i\pi/5)$ and $z_2 = 3\exp(i\pi/3)$.

1.5 Show that: (i) $\exp(i\pi) + 1 = 0$, (ii) $\exp(i\pi/2) - i = 0$, (iii) $i^i = 1/\sqrt{e^\pi}$, (iv) $i^{1/i} = \sqrt{e^\pi}$, (v) $i^{1/i} = i^{-i}$.

1.6 If $C = \int \exp(ax)\cos bx dx$ and $S = \int \exp(ax)\sin bx dx$ show that $C + iS = \int \exp[(a+ib)x]dx$ and hence (ignoring constants of integration) that

$$C = \frac{\exp(ax)}{a^2+b^2}(a\cos bx + b\sin bx)$$

and

$$S = \frac{\exp(ax)}{a^2+b^2}(a\sin bx - b\cos bx).$$

1.7 Given that $z = r\exp(i\theta)$ and $z - 1 = R\exp(i\alpha)$, show that

$$\mathrm{Re}[\ln(z-1)] = \frac{1}{2}\ln(1 - r^2 - 2r\cos\theta).$$

1.8 Given that $\int_{-\infty}^{\infty} \exp(-x^2)dx = \sqrt{\pi}$ show that

$$\int_0^{\infty} \exp(-x^2)\cos ax dx = \frac{1}{2}\exp(-a^2/4)\sqrt{\pi}$$

by integrating $\exp(-z^2)$ around the rectangle whose sides are $x = R$, $x = -R$, $y = 0$, $y = a/2$, and letting $R \to \infty$.

1.9 Evaluate the following contour integrals:

(i) $\int \frac{z^3 - z + 2}{z-1}dz$ around $x^2 + 2y^2 = 4$ and around $x^2 + y^2 = \frac{1}{2}$,

(ii) $\int \frac{3z^2 - 2z + 1}{(z^2+1)(z-1)}dz$ around $x^2 + y^2 = 4$.

1.10 Show that

$$\int_{-\infty}^{\infty} \frac{dx}{x^2 + 2x + 2} = \pi$$

by integrating $(z^2 + 2z + 2)^{-1}$ around a large semicircle.

1.11 Evaluate

$$\int_0^{2\pi} \frac{d\theta}{2 + \sin\theta}$$

by writing $z = \exp(i\theta)$ and integrating around the unit circle.

1.12 By integrating $\exp(iz)/(1 + z^2)$ around a large semicircle show that

$$\int_0^{\infty} \frac{\cos x}{1 + x^2} dx = \frac{\pi}{2e}.$$

Chapter 2

The Delta Function

This chapter is primarily concerned with an introduction to the delta or δ-function which is used routinely in the study of signals and systems. However, we also introduce the convolution integral (a natural consequence of the sampling property of the δ-function) and further, consider the role of the Green's function. Although the Green's function is not a necessary pre-requisite for much of the mathematical background required in signal analysis, it does provides an important link between the δ-function and the convolution process - a process that is fundamental to signal analysis and processing and is related to a fundamental approach to solving linear inhomogeneous partial differential equations. Thus, it is introduced to the reader in this chapter for reasons of both interest and continuity.

Since the mid 1930s, engineers and physicists have found it convenient to introduce fictitious functions having idealized properties that no physically significant functions can possibly possess. A typical example of these functions is the so-called Dirac delta function, which is defined by some authors as having the following properties:

$$\delta(t) = \begin{cases} 0, & t \neq 0; \\ \infty, & t = 0. \end{cases}$$

$$\int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0).$$

The δ-function was first introduced by Paul Dirac in the 1930s as part of his pioneering work in the field of quantum mechanics but the idea may well have been around in mathematical circles for some time before that. Nevertheless, the δ-function is sometimes referred to as the Dirac δ-function. Clearly, such a function does not exist in the classical (analysis) sense. It was originally referred to by Dirac as an improper function and he recommended its use in analysis only when it is obvious that no inconsistency will follow from it.

When attempting to provide a rigorous interpretation of the above equations, it is necessary to generalize the concept of a function. It was the work of L Schwartz in the 1950s which put the theory of $\delta(t)$, and another fictitious functions, on a firm foundation. The mathematical apparatus developed by Schwartz is known as

the 'Theory of Distributions' - more accessible and simplified versions of this theory being developed in the classical text books of Lighthill and Zemanian for example which form the basis of the theory of generalized functions.

The δ-function is just one of a class of 'generalized functions' that has fundamental importance in signal analysis. It is a function that needs to be introduced at this stage in order for the reader to be acquainted with its use later on. It is, for example, of fundamental importance in generalized Fourier analysis and in the sampling theorem which are discussed later on in Part I of this work. We shall start by introducing some special functions that are of particular importance in signal analysis and are related to definitions and applications involving the δ-function. We consider the functions discussed here to be functions of t which is taken to denote time.

2.1 Some Important Generalized Function

The Tophat Function

The tophat function is defined as

$$H(t) = \begin{cases} 1, & \mid t \mid \leq T; \\ 0, & \mid t \mid > T. \end{cases}$$

The Unit Step Function

We define the unit step function U by the relation

$$U(t) = \begin{cases} 1, & t > 0; \\ 0, & t < 0. \end{cases}$$

where $U(0)$ is undefined. Where necessary, we adopt the following convention:

$$U_c(t) = \begin{cases} 1, & t > 0; \\ c, & t = 0; \\ 0, & t < 0. \end{cases}$$

where c is any value between 0 and 1, i.e. $c \in [0, 1]$.

The Signum Function

The signum function sgn is defined as

$$\text{sgn}(t) = \begin{cases} 1, & t > 0; \\ -1, & t < 0. \end{cases}$$

Note that

$$\text{sgn}(t) = U(t) - U(-t)$$

and

$$U(t) = \frac{1}{2}[1 + \text{sgn}(t)].$$

Each of these functions have discontinuities and are therefore in the classical sense, not differentiable. If we let f be any function whose only discontinuity is a simple 'jump' discontinuity at $t = a$, then f can always be expressed in the form

$$f(t) = \phi_1(t)U(a-t) + \phi_2(t)U(t-a)$$

where ϕ_1 and ϕ_2 are continuous everywhere.

The Comb Function

Another function that is of importance in signal analysis (in particular, in the analysis of sampled functions) is the comb function given by

$$\text{comb}(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

where T is a constant and δ is the δ-function whose properties are now discussed.

2.2 The Delta Function

If we assume the existence of a δ-function which behaves formally as the derivative of the unit step function U, then:

(i) The pointwize behaviour of δ should be given by

$$\delta(t) = \begin{cases} \infty, & t = 0; \\ 0, & t \neq 0. \end{cases}$$

(ii) If f is any function which is continuous in a neighbourhood of $t = 0$, then

$$\int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0).$$

This last result is known as the sampling property of the δ-function. In particular,

$$\int_{-\infty}^{t} \delta(\tau)d\tau = \int_{-\infty}^{\infty} U(t-\tau)\delta(\tau)d\tau = U(t), \quad \text{for} \quad t \neq 0.$$

Note that if $U(t) = 1 \quad \forall t$ then (normalization property)

$$\int_{-\infty}^{\infty} \delta(t-\tau)d\tau = 1.$$

The properties (i) and (ii) above are actually incompatible. In fact, there is no function, in the ordinary sense of the term, which can behave consistently as the

'derivative' of the unit step function. In what follows, we use the symbol δ as a convenient way of representing the fundamental sampling operation stated in point (ii) above, and construct a consistent set of rules for operating with this symbol. No appeal should be made to the apparent pointwize behaviour of the 'function' $\delta(t)$, except for purely descriptive purposes.

The descriptive or symbolic definition is compounded in point (i) above and describes an infinitely 'thin' spike with infinite amplitude which is clearly not a proper function. It is therefore sometimes convenient to 'visualize' a δ-function in terms of the limit of a sequence of 'proper' functions $S_n(x)$ which are known as δ-sequences, i.e.

$$\delta(t) = \lim_{n\to\infty} S_n(t).$$

The δ-function can then be thought of in terms of the role it plays in the following fundamental definition:

$$\int_{-\infty}^{\infty} \delta(t)f(t)dt = \lim_{n\to\infty} \int_{-\infty}^{\infty} S_n(t)f(t)dt = f(0).$$

2.2.1 Examples of δ-sequences

A delta- or δ-sequence can be generated from any piecewize continuous function that has the property whereby it contracts and increases its amplitude as a parameter is increased. Examples include the following:

The Gaussian function

$$S_n(t) = \frac{n}{\sqrt{\pi}} \exp(-n^2t^2).$$

The tophat function

$$S_n(t) = \begin{cases} \frac{n}{2}, & \mid t \mid \leq 1/n; \\ 0, & \mid t \mid > 1/n. \end{cases}$$

The Cauchy function

$$S_n(t) = \frac{n}{\pi} \frac{1}{(1+n^2t^2)}.$$

The sinc function

$$S_n(t) = \frac{n}{\pi} \operatorname{sinc}(nt)$$

where

$$\operatorname{sinc}(nt) = \frac{\sin(nt)}{nt}.$$

2.2.2 Integral Representation of the δ-function

The last δ-sequence example provides an important and widely used integral representation of the δ-function. Noting that

$$\frac{1}{2\pi}\int_{-n}^{n}\exp(i\omega t)d\omega = \frac{1}{2\pi}\frac{\exp(int)-\exp(-int)}{it} = \frac{n}{\pi}\,\mathrm{sinc}(nt),$$

then, since

$$\lim_{n\to\infty}\int_{-\infty}^{\infty} S_n(t)f(t)dt = \int_{-\infty}^{\infty}\delta(t)f(t)dt$$

we can write

$$\delta(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(i\omega t)d\omega.$$

2.3 Properties of the δ-function

Having introduced the δ-function, we now provide a collection of its fundamental properties.

Generalization Sampling Property

If f is continuous on a neighbourhood of $t = a$, then

$$\int_{-\infty}^{\infty} f(t)\delta(t-a)dt \equiv \int_{-\infty}^{\infty} f(t)\delta_a(t)dt = f(a).$$

Note that we use the symbol ϕ_a to denote the translation of any given function ϕ, i.e.

$$\phi_a(t) \equiv \phi(t-a).$$

Addition

(i)

$$\int_{-\infty}^{\infty} f(t)[\phi(t)+\delta(t)]dt = \int_{-\infty}^{\infty} f(t)\phi(t)dt + f(0).$$

(ii)

$$\int_{-\infty}^{\infty} f(t)[\delta_a(t)+\delta_b(t)]dt = f(a)+f(b).$$

Multiplication

(i) For any number k

$$\int_{-\infty}^{\infty} f(t)[k\delta(t)]dt = kf(0).$$

(ii) For any function ϕ, the formal product $\phi(t)\delta(t)$ is defined by

$$\int_{-\infty}^{\infty} f(t)[\phi(t)\delta(t)]dt = \int_{-\infty}^{\infty} [f(t)\phi(t)]\delta(t)dt = f(0)\phi(0).$$

Differentiation of Discontinuous Functions

Let f be a function defined by

$$f(t) = \phi_1(t)u(a-t) + \phi_2(t)u(t-a)$$

where ϕ_1 and ϕ_2 are differentiable functions, and f has a jump at $t = a$. The classical derivative f' of f is defined everywhere except at $t = a$ as

$$f'(t) = \phi_1'(t)u(a-t) + \phi_2'(t)u(t-a), \quad \text{for} \quad t \neq a.$$

The generalized derivative Df is now given by

$$\begin{aligned} Df(t) &= \phi_1'(t)u(a-t) + \phi_2'(t)u(t-a) + \delta_a(t)[\phi_2(a) - \phi_1(a)] \\ &= f'(t) + [f(a^+) - f(a^-)]\delta(t-a). \end{aligned}$$

Integration of the generalized derivative will recover the original function f, complete with the jump discontinuity at $t = a$. Integration of the classical derivative loses the information about the jump. Often it is clear from the context, which of the derivatives is referred to. The same symbol f' is used for either sense of the term. (But this usage can cause confusion.)

2.4 Derivatives of the δ-function

The derivative δ' of δ is defined in terms of the following fundamental sampling property: For any function f which has a derivative f' continuous in some neighbourhood of $t = 0$,

$$\int_{-\infty}^{\infty} f(t)\delta'(t)dt = -f'(0).$$

More generally, for any given positive integer n, the generalized function $\delta^{(n)}$ is defined by the characteristic sampling property

$$\int_{-\infty}^{\infty} f(t)\delta^{(n)}(t)dt = (-1)^n f^{(n)}(0)$$

where f is any function with continuous derivatives at least up to n^{th} order in some neighbourhood of $t = 0$.

Translation

$$\int_{-\infty}^{\infty} f(t)\delta_a'(t)dt = -f'(a),$$

$$\int_{-\infty}^{\infty} f(t)\delta_a^{(n)}(t)dt = (-1)^n f^{(n)}(a).$$

Addition

$$\int_{-\infty}^{\infty} f(t)[\phi(t) + \delta^{(n)}(t)]dt = \int_{-\infty}^{\infty} f(t)\phi(t)dt + (-1)^n f^{(n)}(a).$$

More generally, if n and m are positive integers with $n \geq m$, and if f is any function with continuous derivatives at least up to the n^{th} order, then,

$$\int_{-\infty}^{\infty} f(t)[\delta_a^{(n)}(t) + \delta_b^{(m)}(t)]dt = (-1)^n f^{(n)}(a) + (-1)^m f^{(m)}(b).$$

Multiplication

(i) For any number k,

$$\int_{-\infty}^{\infty} f(t)[k\delta^{(n)}(t)]dt = (-1)^n k f^{(n)}(0).$$

(ii) If ϕ is continuously differentiable, then $\phi(t)\delta'(t)$ is defined by

$$\int_{-\infty}^{\infty} f(t)[\phi(t)\delta'(t)]dt = -\left[\frac{d}{dt}(f\phi)\right]_{t=0} = -f'(0)\phi(0) - f(0)\phi'(0).$$

Therefore

$$\phi(t)\delta'(t) \equiv \phi(0)\delta'(t) - \phi'(0)\delta(t).$$

(iii) If ϕ has continuous derivatives at least up to order n, then

$$\phi(t)\delta^n(t) = \phi(0)\delta^{(n)}(t) - n\phi'(0)\delta^{(n-1)}(t) + \frac{n(n-1)}{2}\phi''(0)\delta^{(n-2)}(t) + ... + (-1)^n\phi^{(n)}(0)\delta(t).$$

2.5 Integration Involving Delta Functions

(i)

$$\int_{-\infty}^{t} \delta(t)dt = u(t), \quad \text{for} \quad t \neq 0,$$

$$\int_{-\infty}^{\infty} \delta(t)dt = 1.$$

(ii)

$$\int_{-\infty}^{t} \delta^{(n)}(t)dt = \delta^{(n-1)}(t), \quad n \geq 1,$$

$$\int_{-\infty}^{\infty} \delta^{(n)}(t)dt = 0.$$

(iii) If $a < 0 < b$, then

$$\int_{a}^{b} f(t)\delta^{(n)}(t)dt = (-1)^n f^{(n)}(0).$$

(iv) If $0 < a$ or $b < 0$, then

$$\int_{a}^{b} f(t)\delta^{(n)}(t)dt = 0.$$

(v) If $a = 0$ or $b = 0$, then the integrals are not defined.

Change of Variable

To interpret expressions of the form $\delta[\phi(t)]$ we can use one or other of the following methods:

(i) Generalization of the ordinary change of variable rule for integrals. Thus, let $x = \phi(t)$, so that $t = \phi^{-1}(x) \equiv \psi(x)$ say. Then,

$$\int_{a}^{b} f(t)\delta[\phi(t)]dt = \int_{\phi(a)}^{\phi(b)} f[\psi(x)]\delta(x)\psi'(x)dx.$$

(ii) Alternatively, we can use the fact that

$$\delta[\phi(t)] = \frac{d}{dt}U(x) = \frac{d}{dt}U[\phi(t)]\frac{dt}{dx} = \frac{\frac{d}{dt}u[\phi(t)]}{\frac{d\phi}{dt}}.$$

Both methods can be adapted to apply to expressions of the form $\delta^{(n)}[\phi(t)]$.

Special Results

(i)

$$\delta(\alpha t - \beta) = \frac{1}{\mid \alpha \mid}\delta(t - \beta/\alpha).$$

(ii)

$$\delta[(t-\alpha)(t-\beta)] = \frac{1}{\beta - \alpha}[\delta(t-\alpha) + \delta(t-\beta)], \quad \alpha < \beta.$$

(iii)

$$\delta(\sin t) = \sum_{m=-\infty}^{\infty} \delta(t - m\pi).$$

Note that δ behaves like an even function and δ' like an odd function, i.e.

$$\delta(-t) = \delta(t), \quad \delta'(-t) = -\delta'(t).$$

2.6 Convolution

The convolution $f_1 \otimes f_2$ of two ordinary function f_1 and f_2 is defined by

$$(f_1 \otimes f_2)(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t-\tau) d\tau$$

whenever the infinite integral exists. In particular, this will certainly be the case whenever f_1 and f_2 are absolutely integrable, i.e.

$$\int_{-\infty}^{\infty} \mid f_i(t) \mid dt < \infty, \quad i = 1, 2.$$

(i) Convolution is associative,

$$(f_1 \otimes f_2) \otimes f_3 = f_1 \otimes (f_2 \otimes f_3)$$

and commutative,

$$f_1 \otimes f_2 = f_2 \otimes f_1.$$

If h is a function such that $h(t) = 0, \quad \forall t < 0$ then,

$$(f \otimes h)(t) = \int_0^\infty f(t-\tau)h(\tau)d\tau = \int_{-\infty}^t f(\tau)h(t-\tau)d\tau.$$

If, in addition, f is such that $f(t) = 0, \quad \forall t < 0$ then,

$$(f \otimes h)(t) = \int_0^t f(t-\tau)h(\tau)d\tau = \int_0^t f(\tau)h(t-\tau)d\tau.$$

(ii) For any piecewize continuous function f we have

$$(f \otimes \delta)(t) = \int_{-\infty}^\infty f(t-\tau)\delta(\tau)d\tau = \int_{-\infty}^\infty f(\tau)\delta(t-\tau)d\tau = f(t)$$

and

$$(f \otimes \delta_a)(t) = \int_{-\infty}^\infty f(t-\tau)\delta_a(\tau)d\tau = f(t-a).$$

(iii) For any function f which has continuous derivatives at least up to the n^{th} order,

$$(f \otimes \delta^{(n)})(t) = \int_{-\infty}^\infty f(t-\tau)\delta^{(n)}(\tau)d\tau = \int_{-\infty}^\infty f^{(n)}(t-\tau)\delta(\tau)d\tau = f^{(n)}(t).$$

(iv)

$$(\delta_a \otimes \delta_b)(t) = \int_{-\infty}^\infty \delta_a(t-\tau)\delta_b(\tau)d\tau = \delta_a(t-b) = \delta_{a+b}(t),$$

$$(\delta^{(n)} \otimes \delta^{(m)})(t) = \int_{-\infty}^\infty \delta^{(n)}(t-\tau)\delta^{(m)}(\tau)d\tau = \delta^{(m+n)}(t).$$

2.7 The Green's Function

Convolution has been introduced in the previous section as a natural consequence of our discussion of the δ-function. The convolution of a function with a δ-function provides us with a working definition of the δ-function. In this sense, it is not the δ-function itself that is important but the role it plays in this operation, i.e. the sampling property.

Convolution is absolutely fundamental in terms of modelling analogue (the convolution integral) and digital (the convolution sum) signals. For a time invariant linear system, the output signal generated by the system responding to some input signal is modelled in terms of the convolution of the input signal with a function describing the system. To find this function, we can input a δ-function or, in the 'engineers language', an impulse, in order to determine the Impulse Response Function (IRF) since the convolution of a function with the δ-function reproduces the function.

The convolution process occurs in many areas of mathematical analysis, statistics, physics and engineering and can be introduced in many different ways. One way is through the Green's function which, as introduced here, provides a link between the convolution process, the δ-function and a system taken to be described by some inhomogeneous linear partial differential equation.

Green's functions are named after the mathematician and physicist George Green who was born in Nottingham in 1793 and 'invented' the Green's function in 1828. This invention is developed in an essay written by Green entitled *Mathematical Analysis to the Theories of Electricity and Magnetism* originally published in Nottingham in 1828 and reprinted by the George Green Memorial Committee to mark the bicentenary of the birth of George Green in 1993. In this essay, Green's function solutions to the Laplace and Poisson equation are formulated.

The Green's function is a powerful mathematical tool rather than a physical concept and was successfully applied to classical electromagnetism and acoustics in the late nineteenth century. More recently, the Green's function has been the working tool of calculations in particle physics, condensed matter and solid state physics, quantum mechanics and many other topics of advanced applied mathematics and mathematical physics. Just as the Green's function revolutionized classical field theory in the nineteenth century (hydrodynamics, electrostatics and magnetism) so it revolutionized quantum field theory in the mid-twentieth century through the introduction of quantum Green's functions. This provided the essential link between the theories of quantum electrodynamics in the 1940s and 1950s and has played a major role in theoretical physics ever since. It is interesting to note that the pioneering work of Richard Fynman in the 1950s and 1960s, which lead to the development of the Fynman diagram, was based on the Green's function. The Fynman diagram can be considered to be a pictorial representation of a Green's function (a Green's function associated with wave operators) - what Fynman referred to as a 'propagator'.

The Green's function is possibly one of the most powerful analytical tools for solving partial differential equations, a tool that is all the more enigmatic in that the work of George Green was neglected for nearly thirty years after his death in 1841 and to this day no one knows what he looked like or how and why he developed his revolutionary ideas. Green's functions are used mainly to solve certain types of linear inhomogeneous partial differential equations (although homogeneous partial differential equations can also be solved using this approach). In principle, the Green's function technique can be applied to any linear constant coefficient inhomogeneous partial differential equation (either scalar or vector) in any number of independent variables, although in practice, difficulties can arise in computing the Green's function analytically. In fact, Green's functions provide more than just a solution. They transform a partial differential equation representation of a physical problem into a integral equation representation, the kernel of the integral equation being composed

(completely or partly) of the Green's function associated with the partial differential equation. This is why Green's function solutions are considered to be one of the most powerful analytical tool we have for solving partial differential equations, equations that arise in areas such as electromagnetism (Maxwell's equations), wave mechanics (elastic wave equation) optics (Helmholtz equation), quantum mechanics (Schrödinger and Dirac equations) to name but a few.

In one-dimensional time independent problems, the Green's function can be considered to be the solution to an equation of the form[1]

$$\hat{D}g = \delta(x - x_0)$$

where $\hat{D}$ is a linear differential operator and g is the (scalar) Green's function. Green's functions can be used to solve problems which are both time dependent or independent. In one-dimensional time independent problems for example, Green's functions are usually written in the form $g(x \mid x_0)$ or $g(x, x_0)$ both forms being equivalent to $g(\mid x - x_0 \mid)$.

By way of an example, let us consider the inhomogeneous wave equation given by

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x, k) = f(x)$$

where $f(x)$ is some source term. We can think of the function u as being the amplitude of a wave or 'signal' at a point x where k defines the frequency of oscillation. We then introduce the following equation for the Green's function:

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) g(x \mid x_0, k) = \delta(x - x_0).$$

Note that the source term is now replaced with an impulse as described by the δ-function. Multiplying the first equation by g gives

$$g\left(\frac{\partial^2}{\partial x^2} + k^2\right) u = gf$$

and multiplying the second equation by u gives

$$u\left(\frac{\partial^2}{\partial x^2} + k^2\right) g = u\delta(x - x_0).$$

Subtracting the two results and integrating,

$$\int_{-\infty}^{\infty} \left(g\frac{\partial^2 u}{\partial x^2} - u\frac{\partial^2 g}{\partial x^2}\right) dx = \int_{-\infty}^{\infty} fg dx - \int_{-\infty}^{\infty} u\delta(x - x_0) dx.$$

Now, using the generalized sampling property of the δ-function, we get

$$u = \int_{-\infty}^{\infty} fg dx - \int_{-\infty}^{\infty} \left(g\frac{\partial^2 u}{\partial x^2} - u\frac{\partial^2 g}{\partial x^2}\right) dx.$$

[1]It is sometimes useful to define the Green's function as the solution to an equation of the form $\hat{D}g = -\delta(x - x_0)$. Many authors use this definition as it saves having to express the Green's function with a negative sign in front of it (see Question 2.9).

Evaluating the second integral on the right hand side,

$$\int_{-\infty}^{\infty}\left(g\frac{\partial^2 u}{\partial x^2}-u\frac{\partial^2 g}{\partial x^2}\right)dx=\int_{-\infty}^{\infty}\left[\frac{\partial}{\partial x}\left(g\frac{\partial u}{\partial x}\right)-\frac{\partial g}{\partial x}\frac{\partial u}{\partial x}-\frac{\partial}{\partial x}\left(u\frac{\partial g}{\partial x}\right)+\frac{\partial u}{\partial x}\frac{\partial g}{\partial x}\right]dx$$

$$=\int_{-\infty}^{\infty}\frac{\partial}{\partial x}\left(g\frac{\partial u}{\partial x}\right)dx-\int_{-\infty}^{\infty}\frac{\partial}{\partial x}\left(u\frac{\partial g}{\partial x}\right)dx=\left[g\frac{\partial u}{\partial x}\right]_{-\infty}^{\infty}-\left[u\frac{\partial g}{\partial x}\right]_{-\infty}^{\infty}=0$$

provided of course that u and $\frac{\partial u}{\partial x}$ are zero as $x\to\pm\infty$. With these conditions, we obtain the Green's function solution given by

$$u(x_0,k)=\int_{-\infty}^{\infty}f(x)g(x\mid x_0,k)dx\equiv f(x)\otimes g(\mid x\mid,k).$$

Observe, that this result is just a convolution (since $x\mid x_0=x_0\mid x$) of the source term f with the Green's function g. Thus, using the Green's function (which has been defined in terms of the δ-function), we obtain a result that has effectively re-cast a mathematical model for the function u defined in terms of a (inhomogeneous) partial differential equation to one that is defined in terms of a convolution operation (subject to appropriate conditions). Since convolution processes are fundamental to modelling and processing signals (the discrete equivalent being known to engineers as a Finite Impulse Response filter for example), it is important for the reader to appreciate that this process has a deeper significance in terms of general (Green's function) solutions to linear partial differential equations which in turn, are the basis for describing signal supporting and processing systems (e.g. the equation for a transmission line). This is explored further in Chapter 4 using a case study.

To conclude this chapter, the reader should note that the symbol δ may be interpreted as a convenient way of representing the fundamental sampling operation which maps any continuous function $f(t)$ into the value $f(0)$ which is assumed to be at the origin. It is convenient to use the integral notation for this operation but it should be borne in mind, that integrals involving delta functions have at most, only a symbolic meaning. The material provided in this chapter gives a set of practical rules for manipulating expressions containing delta functions which cover most situations of interest.

2.8 Summary of Important Results

Sampling Property

$$f(t)=\int_{-\infty}^{\infty}f(\tau)\delta(t-\tau)d\tau\equiv f(t)\otimes\delta(t).$$

Delta Sequences

Functions $S_n(t)$ such that

$$\int_{-\infty}^{\infty} \delta(t)f(t)dt = \lim_{n\to\infty} \int_{-\infty}^{\infty} S_n(t)f(t)dt = f(0).$$

Normalization

$$\int_{-\infty}^{\infty} \delta(t-\tau)d\tau = 1.$$

Differentiation

$$\int_{-\infty}^{\infty} f(t)\delta_a^{(n)}(t)dt = (-1)^n f^{(n)}(a).$$

Integration

$$\int_{-\infty}^{t} \delta(t)dt = u(t), \quad t \neq 0.$$

Integral Representation

$$\delta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(i\omega t)d\omega$$

Green's Function

The function defined by the solution to equations of the form (time-independent case)

$$\hat{D}u(x) = \delta(x - x_0)$$

where $\hat{D}$ is a linear differential operator.

Green's Function Solution

Solution to equations of the type (time-independent case)

$$\hat{D}u(x) = f(x)$$

given by

$$u(x) = f(x) \otimes g(\mid x \mid)$$

where g is the Green's function.

2.9 Further Reading

- Dirac P A M, *The Principles of Quantum Mechanics*, Oxford University Press, 1947.
- Van der Pol R, *Operational Calculus*, Cambridge University Press, 1955.
- Lighthill M J, *An Introduction to Fourier Analysis and Generalized Functions*, Cambridge University Press, 1960.
- Cannell D M, *George Green: Mathematician and Physicist, 1793-1841*, Athline Press, 1993.
- Hoskins R F, *The Delta Function*, Horwood Publishing, 1999.
- Evans G A, Blackledge, J M and Yardley P, *Analytical Solutions to Partial Differential Equations*, Springer, 1999.

2.10 Problems

Show that:

2.1
$$f(t)\delta(t-a) = f(a)\delta(t-a);$$

2.2
$$x\delta(t) = 0;$$

2.3
$$\delta(a-t) = \delta(t-a);$$

2.4
$$\delta(at) = \frac{1}{\mid a \mid}\delta(t), \quad a \neq 0;$$

2.5
$$\int_{-\infty}^{\infty} f(t)\delta' dt = -f'(0);$$

2.6
$$\delta(a^2 - t^2) = \frac{1}{2a}[\delta(t+a) + \delta(t-a)];$$

2.7

$$\delta(\sin t) = \sum_{n=-\infty}^{\infty} \delta(t - n\pi);$$

2.8 The Fourier transform pair are defined by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt,$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

Determine the Fourier transform of $\delta(t)$ and obtain an integral representation of $\delta(t)$. Hence, evaluate the Fourier transforms of $\cos t$ and $\sin t$. (Hint: use the complex exponential representations for the sine and cosine functions.)

2.9 The Green's function for the one-dimensional wave equation is given by the solution to

$$\left(\frac{\partial^2}{\partial x^2} + k^2 \right) g(x \mid x_0, k) = -\delta(x - x_0)$$

where $k = \omega/c$. Here, ω is the angular frequency of the waves and c is the wave speed. By writing

$$g(X, k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} g(u, k) \exp(iuX) du$$

and

$$\delta(X) = \int_{-\infty}^{\infty} \exp(iuX) du$$

where $X = \mid x - x_0 \mid$, find a solution for g using Cauchy's Residue theorem. Interpret the solution in terms of the propagation of waves to the left and right of the source point x_0.

2.10 Show that the asymptotic (Green's function) solution to the wave equation

$$\left(\frac{\partial^2}{\partial x^2} + k^2 \right) u(x, k) = -f(x)$$

for left travelling waves only is characterized by the Fourier transform of $f(x)$. (Hint: the asymptotic solution for this problem is that obtain by considering the case when $x_0 \longrightarrow \infty$).

Chapter 3

The Fourier Series

In this chapter, we introduce the Fourier series which provides a method of representing a signal in terms of its Fourier coefficients. After discussing the method of representing a signal as a Fourier series, we derive the 'classical' Fourier transform which is then explored further (in terms of the generalized Fourier transform) in Chapter 4.

Many signals are periodic but the period may approach infinity (when the signal is, in effect, non-periodic). Approximations to such signals are important for their analysis and many possibilities exist. The principal idea is to represent a function in terms of some (infinite) series. The problem is to find a series that provides: (i) a useful representation of the function that can manipulated and analysed accordingly; (ii) a series whose component parts (coefficients) can be evaluated relatively easily. We could for example, consider a simple power series

$$f(t) = \sum_n a_n t^n,$$

or a Taylor series

$$f(t) = \sum_n \frac{(t-a)^n}{n!} f^{(n)}(a),$$

but the Fourier series

$$f(t) = \sum_n c_n \exp(int),$$

which is written here in complex form, provides one of the most versatile representations of signals with a finite period.

3.1 Derivation of the Fourier Series

The Fourier series is a trigonometrical series that can be used to represent almost any function $f(t)$ in the range $-\pi \leq t \leq \pi$. Outside this range, the series gives a periodic extension of $f(t)$ with period 2π, i.e.

$$f(t + 2\pi) = f(t).$$

Such a series is useful when: (i) $f(t)$ is already periodic; (ii) $f(t)$ is not periodic but only exists in the range $[-\pi, \pi]$. Let us consider the trigonometrical Fourier series, given by

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nt) + \sum_{n=1}^{\infty} b_n \sin(nt). \tag{3.1.1}$$

Our problem now, is to find expressions for the coefficients a_n and b_n.

Computation of a_0: Integrate both sides of equation (3.1.1) between $-\pi$ and π giving

$$\int_{-\pi}^{\pi} f(t)dt = \int_{-\pi}^{\pi} \frac{a_0}{2} dt = \pi a_0,$$

since all integrals of the type $\int_{-\pi}^{\pi} \sin(nt)dt$ and $\int_{-\pi}^{\pi} \cos(nt)dt$ are zero. Hence

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t)dt.$$

Computation of a_n: Multiply both sides of equation (3.1.1) by $\cos(kt)$ where k is an integer ($k = 1, 2, 3, ...$) and integrate between $-\pi$ and π giving

$$\int_{-\pi}^{\pi} f(t)\cos(kt)dt = \frac{a_0}{2} \int_{-\pi}^{\pi} \cos(kt)dt + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi} \cos(nt)\cos(kt)dt$$

$$+ \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi} \sin(nt)\cos(kt)dt.$$

Now,

$$\int_{-\pi}^{\pi} \sin(nt)\cos(kt)dt = 0 \quad \forall \quad n \quad \text{and} \quad k.$$

Hence, all terms involving b_n are zero. Also,

$$\int_{-\pi}^{\pi} \cos(nt)\cos(kt)dt = \begin{cases} 0, & n \neq k; \\ \pi, & n = k. \end{cases}$$

This property illustrates that the functions $\cos(nt)$ and $\cos(kt)$ are 'orthogonal'. Also, since

$$\int_{-\pi}^{\pi} \cos(kt)dt = 0, \quad \int_{-\pi}^{\pi} f(t)\cos(nt)dt = a_n\pi$$

or

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt.$$

Computation of b_n: Multiply both sides of equation (3.1.1) by $\sin(kt)$ and integrate over t between $-\pi$ and π. Using the results

$$\int_{-\pi}^{\pi}\sin(kt)dt = 0, \quad \int_{-\pi}^{\pi}\cos(nt)\sin(kt)dt = 0$$

and

$$\int_{-\pi}^{\pi}\sin(nt)\sin(kt)dx = \begin{cases} 0, & n \neq k; \\ \pi, & n = k. \end{cases}$$

we obtain

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(nt)dt.$$

Note that these results have been obtained by exploiting the orthogonality of the trigonometrical functions sin and cos. This provides a Fourier series representation of a function in the range $-\pi \leq t \leq \pi$ that can be written out as

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n\cos(nt) + \sum_{n=1}^{\infty} b_n\sin(nt)$$

where

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)dt,$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt, \quad n = 1, 2, 3, ...,$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(nt)dt, \quad n = 1, 2, 3, ...$$

Example Find the Fourier series representation of a 'square wave' signal given by

$$f(t) = \begin{cases} -1, & -\pi \leq t < 0; \\ 1, & 0 \leq t \leq \pi. \end{cases}$$

where $f(t + 2\pi) = f(t)$. In this case,

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)dt = \frac{1}{\pi}\int_{-\pi}^{0}(-1)dt + \frac{1}{\pi}\int_{0}^{\pi} 1dt = 0,$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt = \int_{-\pi}^{0}(-1)\cos(nt)dt + \int_{0}^{\pi}\cos(nt)dt$$

$$= \left[-\frac{\sin(nt)}{n}\right]_{-\pi}^{0} + \left[\frac{\sin(nt)}{n}\right]_{0}^{\pi} = 0,$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(nt)dt = \int_{-\pi}^{0}(-1)\sin(nt)dt + \int_{0}^{\pi}\sin(nt)dt$$

$$= \left[\frac{\cos(nt)}{n}\right]_{-\pi}^{0} + \left[-\frac{\cos(nt)}{n}\right]_{0}^{\pi} = \frac{1}{n}[1-\cos(-n\pi)] - \frac{1}{n}[\cos(n\pi)-1] = \frac{2}{n}[1-\cos(n\pi)].$$

Now, since

$$\cos(n\pi) = \begin{cases} 1, & n \text{ even;} \\ -1, & n \text{ odd.} \end{cases} \quad \text{or} \quad \cos(n\pi) = (-1)^n, \quad n = 1, 2, ...$$

we can write

$$b_n = \frac{2}{\pi n}[1-(-1)^n]$$

and hence, $f(t)$ for the square wave signal in which $f(t) = f(t+2\pi)$ is given by

$$f(t) = \sum_{n=1}^{\infty}\frac{2}{n\pi}[1-(-1)^n]\sin(nt) = \frac{4}{\pi}\left(\frac{\sin t}{1} + \frac{\sin(3t)}{3} + \frac{\sin(5t)}{5} + ...\right).$$

Like any series representation of a function, we may need many terms to get a good approximation, particularly if the function it describes has discontinuities - 'jump' type and other 'sharp' and/or 'spiky' features. Note that the series goes to zero at $t = 0, \pm\pi, \pm 2\pi, ..$ where $f(t)$ has discontinuities. The term $\frac{4}{\pi}\sin t$ is the 'fundamental' frequency and the other terms are the harmonics. Each term of the series represents the harmonic components required to describe a square wave. The values $1, 3, 5, ...$ determine the frequency at which these harmonics oscillate (which becomes increasingly large) and the values $1, \frac{1}{3}, \frac{1}{5}, ...$ determine the amplitudes of these oscillations (which become increasingly small). Note that this square wave is 'odd', i.e. $f(-t) = -f(t)$ and that $\sin(nt)$ is also odd, so only sine terms occur in the Fourier series. Observing this result saves the inconvenience of finding that the cosine terms are all zero. Similarly, an even function, where $f(-t) = f(t)$, only has cosine terms. Thus, if $f(-t) = f(t)$ we need only compute a_n and if $f(-t) = -f(t)$ we need only compute b_n.

3.2 The Half Range Fourier Series

If we require a series representation for $f(t)$ in the range $0 \le t \le \pi$ rather than in the range $-\pi \le t \le \pi$, then we can choose either a sine or a cosine Fourier series.

3.2.1 Cosine Series

Define a new function $g(t)$ where $g(t) = f(t), \quad 0 \le t \le \pi$ and $g(t) = f(-t), \quad -\pi \le t \le 0$. Since $g(-t) = g(t)$, $g(t)$ is even. Hence, the Fourier series has only cosine terms and

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} g(t)\cos(nt)dt = \frac{1}{\pi}\int_{-\pi}^{0} f(-t)\cos(nt)dt + \int_{0}^{\pi} f(t)\cos(nt)dt$$

$$= \frac{1}{\pi}\int_{0}^{\pi} f(t)\cos(nt)dt + \int_{0}^{\pi} f(t)\cos(nt)dt = \frac{2}{\pi}\int_{0}^{\pi} f(t)\cos(nt)dt, \quad n = 0, 1, 2, ...$$

3.2.2 Sine Series

Define $g(t)$ so that $g(t) = f(t), \quad 0 \le t \le \pi$ and $g(t) = -f(-t), \quad -\pi \le t \le 0$. In this case, $g(-t) = -f(t) = -g(t)$ so $g(t)$ is odd. Thus, the series has only sine terms, the coefficients being given by

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} g(t)\sin(nt)dt = -\frac{1}{\pi}\int_{-\pi}^{0} f(-t)\sin(nt)dt + \int_{0}^{\pi} f(t)\sin(nt)dt$$

$$= \frac{1}{\pi}\int_{0}^{\pi} f(t)\sin(nt)dt + \int_{0}^{\pi} f(t)\sin(nt)dt = \frac{2}{\pi}\int_{0}^{\pi} f(t)\sin(nt)dt$$

Example Find a sine series for t^2 in the range $0 \le t \le \pi$. Here,

$$t^2 = \sum_{n=1}^{\infty} b_n \sin(nt)$$

where

$$\frac{\pi}{2}b_n = \int_{o}^{\pi} t^2\sin(nt)dt = \left[-\frac{t^2\cos(nt)}{n}\right]_0^{\pi} + \int_{0}^{\pi} 2t\frac{\cos(nt)}{n}dt$$

$$= -\frac{\pi^2}{n}\cos(n\pi) + \frac{2}{n}\left[\frac{t\sin(nt)}{n}\right]_0^{\pi} - \frac{2}{n}\int_{0}^{\pi}\frac{\sin(nt)}{n}dt$$

$$= -\frac{\pi^2}{n}(-1)^n + \frac{2}{n^2}\left[\frac{\cos(nt)}{n}\right]_0^{\pi} = -\frac{\pi^2}{n}(-1)^n + \frac{2}{n^3}[(-1)^n - 1].$$

Hence,

$$t^2 = \sum_{0}^{\infty}\frac{2}{\pi}\left(\frac{2}{n^3}[(-1)^n - 1] - \frac{\pi^2}{n}(-1)^n\right)\sin(nt), \quad \text{for} \quad 0 \le t \le \pi.$$

3.3 Fourier Series for an Arbitrary Period

The analysis in the previous section was based on a function with a period of 2π (or π in the case of the half range series). If we consider an arbitrary value for this period of $2T$ say, then

$$f(t) = f(x + 2T)$$

and by induction, based on the results and approach covered previously, we have

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\pi nt/T) + \sum_{n=1}^{\infty} b_n \sin(\pi nt/T)$$

where

$$a_n = \frac{1}{T} \int_{-T}^{T} f(t) \cos(n\pi t/T) dt$$

and

$$b_n = \frac{1}{T} \int_{-T}^{T} f(t) \sin(n\pi t/T) dt.$$

Example Expand t as a cosine series in the range $0 \leq t \leq T$. Here,

$$\frac{T}{2} a_n = \int_0^T t \cos(\pi nt/T) dt = \left[\frac{tT}{\pi n} \sin(\pi nt/T) \right]_0^T - \int_0^T \frac{T}{\pi n} \sin(\pi nt/T) dt$$

$$= -\frac{T}{\pi n} \left[-\frac{T}{\pi n} \cos(\pi nt/T) \right]_0^T .$$

Hence,

$$a_n = \frac{2T}{(\pi n)^2} [(-1)^n - 1], \quad n > 0$$

and for $n = 0$,

$$a_0 = \frac{2}{T} \int_0^T t dt = \frac{2}{T} \left[\frac{t^2}{2} \right]_0^T = T$$

giving the result,

$$t = \frac{T}{2} + \sum_{n=1}^{\infty} \frac{2T}{(\pi n)^2} [(-1)^n - 1] \cos(\pi nt/T).$$

This (half-range) Fourier series gives a 'sawtooth' waveform outside the range $0 \leq t \leq T$. If we differentiate both sides of the last equation we obtain a Fourier series representing unity for $0 \leq t \leq T$ and a square wave outside this range, i.e.

$$1 = \sum_{n=1}^{\infty} \frac{2}{\pi n} [1 - (-1)^n] \sin(\pi nt/T), \quad 0 \leq t \leq T$$

which can be compared with the Fourier series for a square wave of period 2π derived earlier. Note that a Fourier series can be differentiated term by term if it is continuous. It can also always be integrated.

3.4 Applications of the Fourier Series to Circuit Theory

Fourier series are used routinely to model the output of electronic circuits. To briefly illustrate this, we shall consider the following problem: Find the steady state output V_o for a RC (Resistor-Capacitor) circuit in which the input voltage V_i is a square wave given by

$$V_i = \sum_{n=1}^{\infty} a_n \sin(nt)$$

where

$$a_n = \frac{2}{\pi n}[1 - (-1)^n].$$

This problem has been chosen to illustrate the use of the Fourier series to analyse RC circuits which can be used to construct highpass and lowpass (analogue) filters.

Highpass Filters

The basic equations for a 'highpass system' are

$$V_o = RI = R\frac{dq}{dt},$$

$$V_i = \frac{q}{C} + RI = \frac{q}{C} + R\frac{dq}{dt}$$

where I is the current and q is the electric charge (see Figure 3.1).

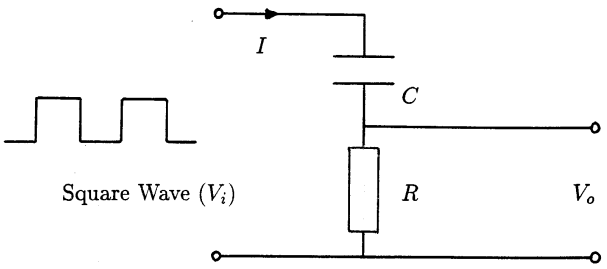


Figure 3.1: RC circuit for a highpass filter.

The method of solution is based on solving the second equation for q and substituting the solution for q into the first equation to obtain V_o. The key to this approach is to

note that the equation for V_i is linear and therefore, we can consider each term in the Fourier series representation of the input square wave separately and solve

$$R\frac{dq_n}{dt} + \frac{q_n}{C} = a_n \sin(nt)$$

or

$$\frac{d}{dt}\left(CRq_n e^{t/RC}\right) = a_n C \sin(nt) e^{t/RC}$$

giving

$$q_n = e^{-t/RC}\frac{1}{RC}\left(a_n C \int e^{t/RC}\sin(nt)dt + \text{constant}\right).$$

Physically, the charge must decay to zero as time increases, i.e.

$$q_n \to 0 \quad \text{as} \quad t \to \infty$$

so that the constant of integration is zero giving

$$q_n = e^{-t/RC}\frac{a_n}{R}\,\text{Im}\left[\int e^{t/RC}e^{int}dt\right] = e^{-t/RC}\frac{a_n}{R}\,\text{Im}\left[\frac{1}{in+1/RC}e^{t/RC}e^{int}\right]$$

$$= \frac{a_n C}{1+(nRC)^2}[\sin(nt) - nRC\cos(nt)].$$

We can 'clean up' this result by letting $nRC = \tan\phi_n$, then

$$\sin(nt) - nRC\cos(nt) = \sin(nt) - \frac{\sin\phi_n}{\cos\phi_n}\cos(nt)$$

$$= \frac{1}{\cos\phi_n}\sin(nt-\phi_n) = \sqrt{1+(nRC)^2}\sin(nt-\phi_n).$$

The solution can then be written in the form

$$q_n = \frac{a_n C}{\sqrt{1+(nRC)^2}}\sin(nt-\phi_n).$$

From the equation for V_o, each component of the output is given by

$$V_n = R\frac{dq_n}{dt} = \frac{RCna_n}{\sqrt{1+(nRC)^2}}\cos(nt-\phi_n).$$

The output is then given by summing over all the components providing the solution

$$V_o = \sum_{n=1}^{\infty} V_n = \sum_{n=1}^{\infty}\frac{RCn}{\sqrt{1+(nRC)^2}}a_n\cos(nt-\phi_n).$$

Note that the n^{th} Fourier component a_n in V_o is changed by the factor

$$H_n = \frac{nRC}{\sqrt{1+(nRC)^2}}.$$

For large n, $H_n \to 1$, so high frequency (i.e. large n) components are unaffected by the circuit. For small n, $H_n \to 0$, so low frequency components are restricted. Thus, the circuit acts as a 'highpass filter' (see Figure 3.2). The output of the filter is illustrated in Figure 3.3 for $RC \sim 1$ and $RC << 1$. In the latter case, the filter behaves as a differentiator.

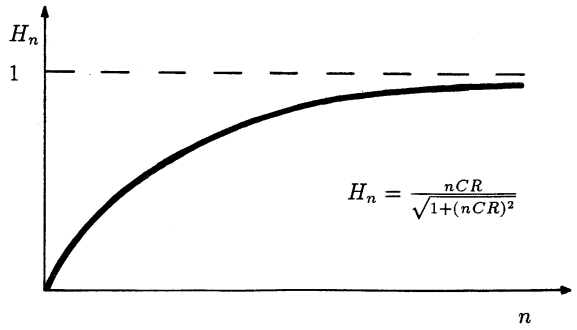


Figure 3.2: Response of the RC highpass filter.

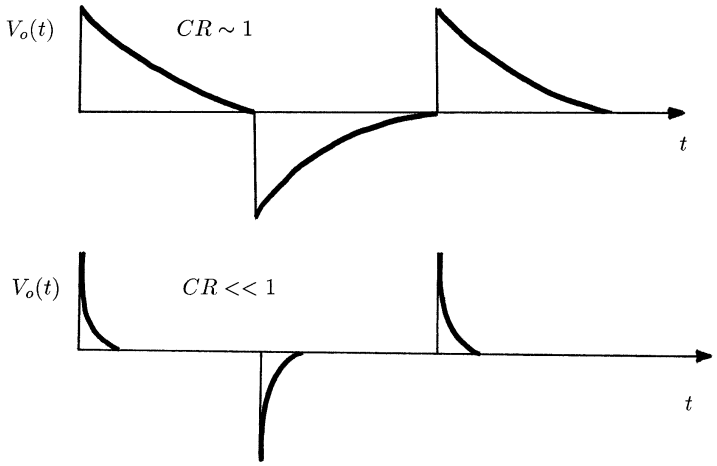


Figure 3.3: Output of RC highpass filter for $RC \sim 1$ and $RC << 1$.

Lowpass Filters

The basic equations for a 'lowpass system' are

$$V_o = \frac{q}{C},$$

$$V_i = \frac{q}{C} + RI = \frac{q}{C} + R\frac{dq}{dt},$$

the latter equation being the same as that for a 'highpass filter' (see Figure 3.4).

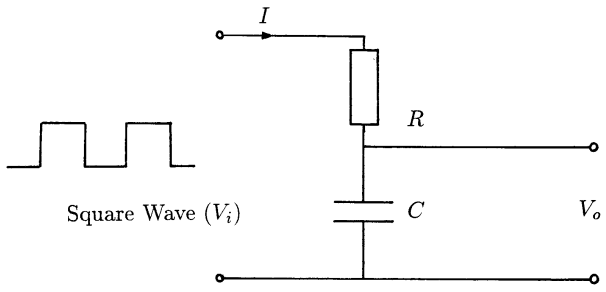


Figure 3.4: RC circuit for a lowpass filter.

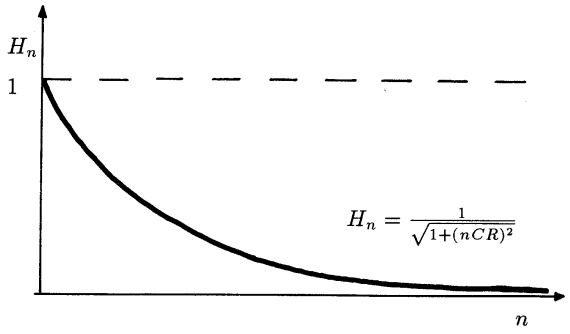


Figure 3.5: Response of the RC lowpass filter.

The solution for the n^{th} component is

$$q_n = \frac{a_n C}{\sqrt{1+(nRC)^2}} \sin(nt - \phi_n).$$

The output component V_n is given by

$$V_n = \frac{a_n}{\sqrt{1+(nRC)}} \sin(nt - \phi_n)$$

and hence

$$V_o = \sum_{n=1}^{\infty} V_n = \sum_{n=1}^{\infty} \frac{1}{\sqrt{1+(nRC)^2}} a_n \sin(nt - \phi_n).$$

In this cases, the n^{th} Fourier component a_n in V_o is changed by the factor

$$H_n = \frac{1}{\sqrt{1+(nRC)^2}}$$

For large n, $H_n \to 0$, so high frequency (i.e. large n) components are 'attenuated' by the circuit. As $n \to 0$, $H_n \to 1$, so low frequency components are unaffected by the circuit which acts as a 'lowpass filter' (see Figure 3.5).

3.5 The Complex Fourier Series

A complex Fourier series performs a similar role to the trigonometrical Fourier series although its implementation is easier and more general. It is just one of a number of linear polynomials which can be used to 'model' a piecewize continuous function $f(t)$. In general, we can consider

$$f(t) = \sum_n c_n B_n(t)$$

where $B_n(t)$ are the basis function and c_n are the coefficients (complex or otherwise). A complex Fourier series is one in which the basis functions are of the form

$$B_n(t) = \exp(int).$$

This series is basic to all Fourier theory and is used to model signals that are periodic. The problem is then reduced to finding the coefficients c_n.

Consider the complex Fourier series for an arbitrary period $2T$ given by

$$f(t) = \sum_{n=-\infty}^{\infty} c_n \exp(int\pi/T).$$

Observe that the summation is now over $(-\infty, \infty)$. To find the coefficients c_n, we multiply both sides of the equation above by $\exp(-imt\pi/T)$ and integrate from $-T$ to T, thus,

$$\int_{-T}^{T} f(t)\exp(-imt\pi/T)dt = \sum_{n=-\infty}^{\infty} c_n \int_{-T}^{T} \exp[i(n-m)t\pi/T]dt.$$

Now, the integral on the right hand side is given by

$$\frac{2T\sin\pi(n-m)}{\pi(n-m)} = \begin{cases} 2T, & n = m; \\ 0, & n \neq m. \end{cases}$$

Note that,

$$\frac{\sin t}{t} = 1 - \frac{t^3}{3!} + \frac{t^5}{5!} - ...$$

so that

$$\left[\frac{\sin t}{t}\right]_{t=0} = 1.$$

Thus, all terms on the right hand side vanish except for the case when $n = m$ and we can therefore write

$$c_n = \frac{1}{2T}\int_{-T}^{T} f(t)\exp(-int\pi/T)dt.$$

By way of an example, suppose we require the complex Fourier series for a square wave signal with period of 2π given by

$$f(t) = \begin{cases} -1, & -\pi \leq t < 0; \\ 1, & 0 \leq x \leq \pi. \end{cases}$$

where $f(t + 2\pi) = f(t)$, then

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \exp(-int) dt = -\frac{1}{2\pi} \int_{-\pi}^{0} \exp(-int) dt + \frac{1}{2\pi} \int_{0}^{\pi} \exp(-int) dt$$

$$= \frac{1}{in\pi}[1 - \cos(n\pi)]$$

and

$$f(t) = \sum_{n=-\infty}^{\infty} \frac{1}{in\pi}[1 - \cos(n\pi)] \exp(in\pi)$$

$$= \sum_{n=0}^{\infty} \frac{1}{in\pi}[1 - (-1)^n] \exp(in\pi) - \sum_{n=0}^{\infty} \frac{1}{in\pi}[1 - (-1)^n] \exp(-in\pi)$$

$$= \sum_{n=1}^{\infty} \frac{2}{n\pi}[1 - (-1)^n] \sin(nt).$$

which recovers the result obtained in Section 3.1 using the trigonometrical Fourier series (but with less computation). As mentioned before, with either the trigonometrical or complex Fourier series, we need many terms to get a good fit to the sharp features and discontinuities. The Fourier series representation of an 'on-off' type signal such as a square wave requires many terms to represent it accurately. Truncation of the series leads to truncation errors which in a Fourier series is generally referred to as 'ringing' . The generalization of this effect is called the Gibbs' phenomenon. This leads to a general rule of thumb which is an important aspect of all signal processing, namely, that 'sharp' features in a signal require many Fourier coefficients to be represented accurately whereas smooth features in a signal require fewer Fourier coefficients. Hence, one way of 'smoothing' a signal is to reduce the number of Fourier coefficients used to represent the signal. This is the basis of lowpass filtering. Moreover, if a signal is relatively smooth, it may require relatively few Fourier coefficients to reconstruct it accurately. In such cases, storing the coefficients c_n instead of the (digital) signal itself can lead to a method of reducing the amount of data required to store the (digital) signal. This approach to data compression is actually applied in practice using the Discrete Cosine Transform (DCT) - the cosine transform is briefly discussed in Chapter 5 - and is the basis for the Joint Photographics Experts Group or JPEG compression scheme. The DCT is used because it has properties that are optimal in terms of using it to design a data compression algorithm, i.e. in expressing a digital signal in terms of its DCT coefficients and reproducing it from them. However, the principal 'philosophy' behind this approach is the same as that discussed above. Actually, the Discrete Fourier Transform or DFT, which is discussed shortly, can also be used for this purpose. It is not such a good compressor as the DCT (because it

is a complex transform with both real and imaginary parts), but it does provide the option of processing the data in compression space using the Fourier amplitude and phase which the DCT does not provide.

3.6 The Fourier Transform Pair

The Fourier transform is the subject of the following chapter and it is prudent at this stage to derive the Fourier transform pair (i.e. the forward and inverse Fourier transforms) from the complex Fourier series. To do this in a way that is notationally consistent, we let $c_n = F_n/2T$ so that

$$f(t) = \frac{1}{2T} \sum_n F_n \exp(int\pi/T)$$

and

$$F_n = \int_{-T}^{T} f(t) \exp(-int\pi/T) dt$$

where

$$\sum_n \equiv \sum_{n=-\infty}^{\infty}.$$

Now, let $\omega_n = n\pi/T$ and $\Delta\omega_n = \pi/T$. We can then write

$$f(t) = \frac{1}{2\pi} \sum_n F_n \exp(i\omega_n t)\Delta\omega_n$$

and

$$F_n = \int_{-T}^{T} f(t) \exp(-i\omega_n t) dt.$$

Then, in the limit as $T \to \infty$, we have

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega,$$

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt.$$

Here, $F(\omega)$ is the Fourier transform of $f(t)$ and $f(t)$ is the inverse Fourier transform of $F(\omega)$. Taken together, these integral transforms form the 'Fourier transform pair'. Note that $f(t)$ is a non-periodic function, since we have used $T \to \infty$ to obtain this result.

3.7 The Discrete Fourier Transform

The discrete Fourier transform or DFT is the 'work horse' for so many of the routine algorithms used for processing digital signals and in Part IV of this work, the basis of a fast algorithm for computing the DFT will be discussed. For now, it is useful and informative to demonstrate the derivation of the DFT from the complex Fourier series.

The complex Fourier series can be written as

$$f(t) = \frac{1}{2T} \sum_n F_n \exp(i\pi nt/T)$$

where

$$F_n = \int_{-T}^{T} f(t) \exp(-i\pi nt\pi/T) dt$$

over the range $[-T, T]$. The DFT can now be derived be considering a discretized form of the function $f(t)$ with uniform sampling at points $t_0, t_1, t_2, ..., t_{N-1}$ giving the discrete function or vector

$$f_m \equiv f(t_m); \quad m = 0, 2, 3, ..., N-1$$

with sampling interval Δt. Now, $t_m = m\Delta t$ and if we let $N = T/\Delta t$, we have

$$f_m = \frac{1}{N} \sum_n F_n \exp(i2\pi nm/N),$$

$$F_n = \sum_m f_m \exp(-i2\pi nm/N).$$

Note that the summations are now finite with n and m running from $-N/2$ to $(N/2)-1$ or alternatively from $n = 0$ to $N-1$.

3.8 Relationship between the DFT and the Fourier Transform

Consider the Fourier transform pair given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt,$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(it) d\omega$$

and the DFT pair, i.e.

$$F_n = \sum_m f_m \exp(-i2\pi nm/N),$$

$$f_m = \frac{1}{N}\sum_n F_n \exp(i2\pi nm/N).$$

To study the relationship between these two results, we can consider the following discretization of the Fourier transform pair:

$$F(\omega_n) = \sum_m f(t_m)\exp(-i\omega_n t_m)\Delta t,$$

$$f(t_m) = \frac{1}{2\pi}\sum_n F(\omega_n)\exp(i\omega_n t_m)\Delta\omega$$

where Δt and $\Delta\omega$ are the sampling intervals. Writing $\omega_n = n\Delta\omega$ and $t_m = m\Delta t$, by inspection (i.e. comparing the results above with the DFT pair) we see that

$$\Delta\omega\Delta t = \frac{2\pi}{N}.$$

This result provides the relationship between the (sampling) interval $\Delta\omega$ of the numbers F_n and the (sampling) interval Δt of the numbers f_m.

3.9 'Standard' and 'Optical' Forms of the DFT

In the previous section, the limits on the sums defining the DFT have been assumed to run from $-N/2$ to $(N/2)-1$ assuming the data is composed of $N-1$ elements. When we consider the case where

$$\sum_n \equiv \sum_{n=-N/2}^{(N/2)-1}$$

the DC (Direct Current) or zero frequency level is taken to occur at the centre of array F_m giving what is termed the optical form of the DFT. In the case when

$$\sum_n \equiv \sum_{n=0}^{N-1}$$

the DC level is taken to occur at F_0 - first value of array F_m. This is known as the standard form of the DFT.

The optical form has some valuable advantages as it provides results that are compatible with those associated with Fourier theory in which the spectrum $F(\omega)$ is taken to have its DC component at the centre of the function. The reason for calling this form of the DFT 'optical' is that there is an analogy between this form and that of the 2D DFT in which the DC component occurs at the centre of a 2D array. In turn the 2D Fourier transform can be used to model the process of a well corrected lens focusing light on to the 'focal plane' in which the zero frequency occurs in the centre of this plane. The standard form of the DFT is often useful for undertaking analytical work with the DFT and in particular, developing the Fast Fourier Transform algorithm that is discussed in Part IV of this text.

3.10 Summary of Important Results

The Trigonometrical Fourier Series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\pi nt/T) + \sum_{n=1}^{\infty} b_n \sin(\pi nt/T)$$

where

$$a_n = \frac{1}{T} \int_{-T}^{T} f(t) \cos(n\pi t/T) dt$$

and

$$b_n = \frac{1}{T} \int_{-T}^{T} f(t) \sin(n\pi t/T) dt$$

for a periodic function with period $2T$.

Complex Fourier Series

$$f(t) = \sum_{n=-\infty}^{\infty} c_n \exp(int\pi/T)$$

where

$$c_n = \frac{1}{2T} \int_{-T}^{T} f(t) \exp(-int\pi/T) dt$$

for a periodic function with period $2T$.

Fourier Transform Pair

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega,$$

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$$

where $f(t)$ is a non periodic function.

Discrete Fourier Transform Pair

$$f_m = \frac{1}{N} \sum_{n=-N/2}^{(N/2)-1} F_n \exp(i2\pi nm/N),$$

$$F_n = \sum_{m=-N/2}^{(N/2)-1} f_m \exp(-i2\pi nm/N)$$

where the relationship between $\Delta\omega$ (the sampling interval of F_n) and Δt (the sampling interval of f_m) is given by

$$\Delta\omega\Delta t = \frac{2\pi}{N}.$$

3.11 Further Reading

- Beauchamp K G, *Signal Processing*, Allen & Unwin, 1973.
- Schwartz M and Shaw L, *Signal Processing*, McGraw-Hill, 1975.
- Papoulis A, *Signal Analysis*, McGraw-Hill, 1977.
- Lynn P A, *An Introduction to the Analysis and Processing of Signals*, Macmillan, 1979.
- Connor F R, *Signals*, Arnold, 1982.
- Chapman M J, Goodall D P and Steel, N C, *Signal Processing in Electronic Communications*, Horwood Publishing, 1997.

3.12 Problems

3.1 Verify that when n and k are integers,

(i)

$$\int_{-\pi}^{\pi} \cos nt \sin \omega t dt = 0 \quad \forall \quad n, \omega;$$

(ii)

$$\int_{-\pi}^{\pi} \sin nt \sin \omega t dt = \begin{cases} 0, & n \neq \omega; \\ \pi, & \text{n=}\omega. \end{cases}$$

3.2 Sketch the following functions in the range $-3\pi \leq t \leq 3\pi$:

(i)

$$f(t) = \begin{cases} 0, & -\pi \leq t \leq 0; \\ t, & 0 \leq t \leq \pi. \end{cases}$$

where

$$f(t + 2\pi) = f(t).$$

(ii)

$$f(t) = \mid t \mid, \quad -\pi \leq t \leq \pi; \quad f(t+2\pi) = f(t).$$

(iii)

$$f(t) = H(t), \quad -\pi \leq t \leq \pi; \quad f(t+2\pi) = f(t)$$

where

$$H(t) = \begin{cases} 1, & t \geq 0; \\ 0, & t < 0. \end{cases}$$

3.3 Show that the function given in question 2(i) above can be represented by the series

$$f(t) = \frac{\pi}{4} - \frac{2}{\pi}\left(\cos t + \frac{\cos 3t}{3^2} + \frac{\cos 5t}{5^2} + ...\right) + \left(\sin t - \frac{\sin 2t}{2} + \frac{\sin 3t}{3} - ...\right).$$

Hence, deduce that

$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + ...$$

3.4 Expand the function $\pi - t$ first as a Fourier sine series and then as a Fourier cosine series valid in the range $0 \leq t \leq \pi$.

3.5 Expand the function $\cos t$ as a Fourier sine series valid in the range $0 \leq t \leq \pi$.

3.6 A function $f(t)$ has period 2π and is equal to t in the range $-\pi \leq x \leq \pi$. Show that

$$F(\omega) = 2\pi i \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}[\delta(\omega + n) - \delta(\omega - n)]$$

where $F(\omega)$ is the Fourier transform of $f(t)$.

3.7 A function $f(t)$ has period 2π and is equal to $\mid t \mid$ in the range $-\pi \leq t \leq \pi$. Show that

$$F(\omega) = \pi^2\delta(\omega) + 2\sum_{n=1}^{\infty} \frac{[(-1)^2 - 1]}{n^2}[\delta(\omega - n) + \delta(\omega + n)]$$

where $F(\omega)$ is the Fourier transform of $f(t)$. Sketch the graph of $f(t)$ and $F(\omega)$.

3.8 Expand the function $f(t) = t^2$ as a Fourier series valid in the range $-\pi \leq t \leq \pi$. Hence, show that

$$F(\omega) = \frac{2\pi^3}{3}\delta(\omega) + 4\pi \sum_{n=1}^{\infty} \frac{(-1)^n}{n^2}[\delta(\omega - n) + \delta(\omega + n)]$$

where $F(\omega)$ is the Fourier transform of $f(t)$ and deduce that

$$\frac{\pi^2}{12} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + ...$$

Chapter 4

The Fourier Transform

The Fourier transform is used extensively in many branches of science and engineering. It is particularly important in signal processing and forms the 'work horse' of many methods and algorithms. This chapter provides an introduction to this transform and presents most of the fundamental ideas, results and theorems that are needed to use Fourier theory for the analysis of signals.

4.1 Introduction

The Fourier transform is attributed to the French mathematician Joseph Fourier who, as scientific adviser and reputedly a spy master to Napoleon, 'invented' the transform as a by product into his investigation of the laws of thermodynamics. In 1798, Napoleon invaded Egypt and during this campaign it became clear that the time period over which artillery could be primed and fired from one round to the next was different from that in the European theater of war. In particular, the period of time required to allow a cannon to be fired operating from one round to the next in a hot climate such as that in Egypt was significantly greater than that in northern Europe and on many occasions the cannon pre-ignited as a charge was being loaded. Although obvious today, the laws of thermal conduction at that time were not fully appreciated. Napoleon asked Fourier to investigate the problem. This led Fourier to develop his famous law of conduction in which the heat flux (i.e. the flow of heat) is proportional to the gradient in temperature. Coupled with the equation of continuity for thermal conduction, Fourier derived a partial differential equation that is now commonly known as the 'diffusion equation' which, for the temperature u, has the basic form

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}.$$

Some years later, Fourier developed his transform to investigate solutions to this equation; a transform that was essentially invented in an attempt solve an equation derived from a military problem of the time. Since then, the Fourier transform has found applications in nearly all areas of science and engineering and is arguably one of the most, if not, the most important integral transforms ever devised.

4.1.1 Notation

The Fourier transform of a function f is usually denoted by the upper case F but many authors prefer to use a tilde above this function, i.e. to denote the Fourier transform of f by $\tilde{f}$. In this work, the former notation is used throughout. Thus, the Fourier transform of f can be written in the form

$$F(\omega) \equiv \hat{F}_1 f(t) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$$

where $\hat{F}_1$ denotes the one-dimensional Fourier transform operator. Here, $F(\omega)$ is referred to as the Fourier transform of $f(t)$ where $f(t)$ is a non-periodic function (see Chapter 3).

The sufficient condition for the existence of the Fourier transform is that f is square integrable, i.e.

$$\int_{-\infty}^{\infty} | f(t) |^2 \, dt < \infty.$$

4.1.2 Physical Interpretation

Physically, the Fourier transform of a function provides a quantitative picture of the frequency content of the function which is important in a wide range of physical problems and is fundamental to the processing and analysis of signals and images. The variable ω has dimensions that are reciprocal to those of the variable t. There are two important cases which arise:

(i) t is time in seconds and ω is the temporal frequency in cycles per second (Hertz). Here, ω is referred to as the angular frequency which is given by $2\pi \times \nu$ where ν is the frequency.

(ii) t is distance in metres (usually denoted by x) and ω and the spatial frequency in cycles per metre (usually denoted by k). Here, k is known as the wavenumber and is given by

$$k = \frac{2\pi}{\lambda}$$

where λ is the wavelength and we note that

$$c = \frac{\omega}{k} = \nu\lambda$$

where c is the wavespeed. The Fourier transform is just one of a variety of integral transforms but it has certain properties which make it particularly versatile and easy to work with. This was expressed eloquently by Lord Kelvin, who stated that:

Fourier's theorem is not only one of the most beautiful results of modern analysis, but it may be said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics.

As discussed at the beginning of this chapter, it is interesting to note that this important transform arose from a scientist having to contemplate a technical problem

that was directly related to a military issue - such is the nature of so many aspects of modern science and engineering!

4.1.3 The Spectrum

The Fourier transform of a function is called its 'spectrum' or frequency distribution - a term that should not be confused with that used in statistics. It is generally a complex function which can be written in the form

$$F(\omega) = F_r(\omega) + iF_i(\omega)$$

where

$$F_r = \mathrm{Re}[F] \quad \text{and} \quad F_i = \mathrm{Im}[F],$$

i.e. the real and imaginary parts of the spectrum respectively. Note that if $f(t)$ is a real valued function, then the real and imaginary parts of its Fourier transform are given by

$$F_r(\omega) = \int_{-\infty}^{\infty} f(t)\cos(\omega t)dt$$

and

$$F_i(\omega) = \int_{-\infty}^{\infty} f(t)\sin(\omega t)dt$$

respectively. An alternative and often more informative (Argand diagram) representation of the spectrum is based on writing it in the form

$$F(\omega) = A(\omega)\exp[i\theta(\omega)]$$

where

$$A(\omega) = \mid F(\omega) \mid = \sqrt{F_r^2(\omega) + F_i^2(\omega)}$$

and

$$\theta(\omega) = \tan^{-1}\left[\frac{F_i(\omega)}{F_r(\omega)}\right].$$

The functions F, A and θ are known as the complex spectrum, the amplitude spectrum and the phase spectrum respectively. In addition to these functions, the function

$$A^2(\omega) = \mid F(\omega) \mid^2$$

is also important in Fourier analysis. This function is known as the the Power Spectral Density Function (PSDF) or just the power spectrum. Finally, the value of the spectrum at $\omega = 0$ is called the zero frequency or DC (after Direct Current) level and is given by the integral of f, i.e.

$$F(0) = \int_{-\infty}^{\infty} f(t)dt.$$

Note that this value provides a measure of the scale of the Fourier transform and is proportional to its mean value.

4.1.4 The Inverse Fourier Transform

The function $f(t)$ can be recovered from $F(\omega)$ by employing the inverse Fourier transform which is given by

$$f(t) = \hat{F}_1^{-1} F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

The operator $\hat{F}_1^{-1}$ is used to denote the inverse Fourier transform. The superscript -1 is used to denote that this operator is an inverse operator (N.B. it does not mean $1/\hat{F}_1$).

In Chapter 3, the inverse Fourier transform was derived from the complex Fourier series in a way that is often referred to as the 'classical approach' and is informative in terms of detailing the relationships between the complex Fourier series, the Fourier transform pair and the discrete Fourier transform pair. However, 'armed' with the 'power' of the δ-function, we can derive this result in an arguably more elegant way; thus, multiplying $F(\omega)$ by $\exp(i\omega t')$ and integrating over ω from $-\infty$ to ∞ we can write

$$\int_{-\infty}^{\infty} F(\omega) \exp(i\omega t') d\omega = \int_{-\infty}^{\infty} dt f(t) \int_{-\infty}^{\infty} \exp[i\omega(t' - t)] d\omega.$$

We now employ the integral representation for the delta function discussed in Chapter 2, i.e.

$$\int_{-\infty}^{\infty} \exp[i\omega(t' - t)] d\omega = 2\pi\delta(t' - t).$$

By substituting this result into the previous equation and using the sampling property of the delta function, we get

$$\int_{-\infty}^{\infty} F(\omega) \exp(i\omega t') d\omega = \int_{-\infty}^{\infty} dt f(t) 2\pi\delta(t' - t) = 2\pi f(t')$$

or

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

The inverse Fourier transform is essentially the same as the forward Fourier transform (ignoring scaling) except for a change from $-i$ to $+i$. This is one of the most unique and important features of the Fourier transform; essentially, computing the inverse Fourier transform is the same as computing a forward Fourier transform which is not the case with other integral transforms such as the Laplace transform for example (see Chapter 5).

4.1.5 Alternative Definitions and Representations

Some authors prefer to define the forward and inverse Fourier transforms as

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt, \quad f(t) = \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega$$

or

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt, \quad f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

Also it is a matter of convention that F is called the Fourier transform of f when $-i$ occurs in the exponential and that f is the inverse Fourier transform of F when $+i$ appears in the exponential. The exact form of the forward and inverse Fourier transforms that are used does not really matter and the user may choose the definition which he or she likes the best. What does matter is that consistency with a given definition is maintained throughout a calculation. Here, the definitions for the forward and inverse Fourier transforms given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$$

and

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega$$

respectively are used throughout. Collectively, these integral transforms are known as the Fourier transform pair.

4.1.6 Useful Notation and Jargon

To avoid constantly having to write integral signs and specify the forward or inverse Fourier transform in full, we can make use of the symbolic form

$$f(t) \Longleftrightarrow F(\omega)$$

which means that F is the Fourier transform of f and f is the inverse Fourier transform of F. This notation is useful when we want to indicate the relationship between a mathematical operation on f and its effect on F. Mathematical operations on f are referred to as operations in real or t-space. Similarly, operations on F are referred to as operations in Fourier space or ω-space.

4.1.7 Bandlimited Functions

A bandlimited function is characterized by a complex spectrum $F(\omega)$ such that

$$F(\omega) = 0, \quad |\, \omega \,| > \Omega.$$

In this case, the inverse Fourier transform is given by

$$f(t) = \frac{1}{2\pi} \int_{-\Omega}^{\Omega} F(\omega) \exp(i\omega t) d\omega.$$

Here, f is known as a bandlimited function and 2Ω is referred to as the bandwidth. A bandlimited function is therefore a function that is composed of frequencies which are limited to a particular finite band.

If $f(t)$ is such that

$$f(t) = 0, \quad | \, t \, | > T$$

then its complex spectrum is given by

$$F(\omega) = \int_{-T}^{T} f(t) \exp(-i\omega t) dt.$$

In this case, f is referred to as a time (t - time in seconds) or space (t - length in meters) limited signal. Note that in practice, all signals are of a finite duration and are therefore space/time limited; in the latter case, the function is said to be of 'compact support'. They are also nearly always bandlimited for a variety of different physical reasons.

4.1.8 The Amplitude and Phase Spectra

Given that the amplitude spectrum and the phase spectrum, taken together, uniquely describe the time signature of a signal, it is pertinent to ask how these spectra, taken separately, contribute to the signal. As a general rule of thumb, the phase spectrum is more important than the amplitude spectrum in that, if the amplitude spectrum is perturbed but the phase spectrum remains intact, then the signal can be reconstructed relatively accurately (via the inverse Fourier transform) particularly in terms of the positions at which the signal is zero. For a real valued signal with $A(\omega) > 0, \quad \forall \omega$, its zeros occur when $\omega + \theta(\omega) = \pm n\pi/2, \quad n = 1, 2, 3...$ since

$$f(t) = \mathrm{Re} \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) \exp[i\theta(\omega)] \exp(i\omega t) d\omega = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) \cos[\omega + \theta(\omega)] d\omega.$$

This can be observed by taking an amplitude only and a phase only reconstruction of a signal, i.e. computing

$$f_A(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) \exp(i\omega t) d\omega$$

and

$$f_\theta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp[i\theta(\omega)] \exp(i\omega t) d\omega.$$

An amplitude only reconstruction basically yields 'rubbish' compared to a phase only reconstruction which yields features that are recognisable in terms of the original signal. Thus, the Fourier phase of a signal is 'more important' than the Fourier amplitude and is less robust to error. In some special applications, only the amplitude or power spectrum can be measured and it is necessary to recover the phase. A well known example is X-ray crystal developing the double helix model for DNA in 1953 for example. Here, X-rays with a constant wavelength of λ are diffracted by the three dimensional molecular structure (in a crystal defined by an object function $O(x, y, z)$ which is of compact support. To a first order (single or weak scattering) approximation, the diffracted field $F(x_0, y_0)$ generated by a plane wave travelling along the z-axis and recorded at a point z_0 in the far field is given by (ignoring scaling)

$$F(x_0, y_0) = \int\int f(x, y) \exp(-i2\pi x_0 x/\lambda z_0) \exp(-i2\pi y_0 y/\lambda z_0) dxdy$$

where

$$f(x, y) = \int O(x, y, z) dz.$$

Now, the X-ray image that is recorded is not F but the intensity $\mid F \mid^2$ and so our problem becomes: given $\mid F(u, v) \mid^2$ find $f(x, y)$ where

$$F(u, v) = \int\int f(x, y) \exp(-iux) \exp(-ivy) dxdy, \quad u = 2\pi x_0/\lambda z_0, \quad v = 2\pi y_0/\lambda z_0.$$

This is equivalent to computing the two-dimensional Fourier transform of the function $f(x, y)$, deleting the phase spectrum and then having to recover it from the amplitude spectrum alone together with any available *a priori* information on the diffractor itself such as its spatial extent (because the diffractor will be of compact support). This is an ill-posed problem and like so many 'solutions' to such problems in signal processing, relies on the application of *a priori* knowledge on the object function coupled with an information theoretic criterion upon which a conditional solution is developed (e.g. application of the least squares method, the maximum entropy criterion, Bayesian estimation etc. as discussed in Part IV).

4.1.9 Differentiation and the Fourier Transform

Given that

$$f(t) \Longleftrightarrow F(\omega)$$

we have

$$\frac{d}{dt} f(t) \Longleftrightarrow i\omega F(\omega).$$

The proof of this result is easily established by making use of the inverse Fourier transform, thus:

$$\frac{d}{dt} f(t) = \frac{1}{2\pi} \frac{d}{dt} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega = \frac{1}{2\pi} \int_{-\infty}^{\infty} i\omega F(\omega) \exp(i\omega t) d\omega.$$

Differentiating n times, by induction, we have

$$\frac{d^n}{dt^n}f(t) \Longleftrightarrow (i\omega)^n F(\omega).$$

This simple and elegant result is the basis for an important and versatile generalization which stems from the question as to the meaning of a fractional differential. In this case, we can consider a definition for a fractional differential based on a generalization of the result above to

$$\frac{d^q}{dt^q}f(t) \Longleftrightarrow (i\omega)^q F(\omega)$$

where $q > 0$ and can be non-integer. Although many other definitions for a fractional differential exist, the one considered here is arguably one of the simplest and most versatile of them. Moreover, this definition can be used to define a fractal signal which is discussed later on in this work (see Chapter 17).

4.1.10 Integration and the Fourier Transform

If we consider integration to be the inverse of differentiation, then it is clear that

$$\int f(t)dt \Longleftrightarrow \frac{1}{i\omega}F(\omega)$$

and that

$$\int ... \int f(t)dt...dt \Longleftrightarrow \frac{1}{(i\omega)^n}F(\omega)$$

where n is the number of times that the integration is performed. Similarly, we can define a fractional integral to be one that is characterized by $(i\omega)^{-q}$ where $q > 0$ is the (fractional) order of integration.

4.2 Selected but Important Functions

There are numerous cases where the Fourier transform of a given function $f(t)$ can be computed analytically and many tables of such results are available. Here, some results which are particularly important in signal analysis are derived, some of them relating to the Fourier transforms of the generalized functions discussed in Chapter 2 and all of them being used later on in this work.

Fourier Transform of the Tophat Function

The tophat function (so called because of its shape) is given by

$$H(t) = \begin{cases} 1, & \mid t \mid \leq T; \\ 0, & \mid t \mid > 0. \end{cases}$$

and the Fourier transform of this function is given by

$$\int_{-\infty}^{\infty} H(t)\exp(-i\omega t)dt = \int_{-T}^{T} \exp(-i\omega t)dt$$

$$= \frac{1}{i\omega}[\exp(i\omega T) - \exp(-i\omega T)] = 2\frac{\sin(\omega T)}{\omega} = 2T\,\mathrm{sinc}(\omega T)$$

where

$$\mathrm{sinc}(\omega T) = \frac{\sin(\omega T)}{\omega T}.$$

The sinc function occurs very often in signal analysis. One reason for this is that the tophat function is routinely used to model real signals of finite duration by windowing (multiplying) hypothetical signals of infinite duration. Whenever this is done, the sinc function emerges in one form or another.

Fourier Transform of the Cosine Function

Consider the cosine function $\cos(\omega_0 t)$, where ω_0 is a constant which determines the rate at which this function oscillates about zero. To compute the Fourier transform of this function, we first write it in terms of complex exponentials, i.e.

$$\cos(\omega_0 t) = \frac{1}{2}[\exp(i_0 t) + \exp(-i\omega_0 t)].$$

The Fourier transform can then be written as

$$\int_{-\infty}^{\infty} \cos(\omega_0 t)\exp(-i\omega t)dt = \frac{1}{2}\int_{-\infty}^{\infty} \exp[-i(\omega-\omega_0)t]dt + \frac{1}{2}\int_{-\infty}^{\infty} \exp[-i(\omega+\omega_0)t]dt$$

$$= \pi[\delta(\omega-\omega_0) + \delta(\omega+\omega_0)]$$

using the integral representation of the δ-function.

Fourier Transform of the Sine Function

By defining $\sin(\omega_0 t)$ in terms of complex exponentials, i.e.

$$\sin(\omega_0 t) = \frac{1}{2i}[\exp(i\omega_0 t) - \exp(-i\omega_0 t)]$$

we get

$$\int_{-\infty}^{\infty} \sin(\omega_0 t)\exp(-i\omega t)dt = \frac{1}{2i}\int_{-\infty}^{\infty} \exp[-i(\omega-\omega_0)t]dt - \frac{1}{2i}\int_{-\infty}^{\infty} \exp[-i(\omega+\omega_0)t]dt$$

$$= i\pi[\delta(\omega+\omega_0) - \delta(\omega-\omega_0)].$$

Fourier Transform of a Gaussian Function

The Gaussian function or normal distribution is given by (ignoring scaling)

$$f(t) = \exp(-at^2)$$

where a is a constant. This function is one of the most important examples of continuous probability distributions found in statistics and frequently arises in the area

of signal analysis, especially when statistical methods (including Bayesian methods for example) are introduced. The Fourier transform of this function is slightly more complicated and than the previous examples given and is based on exploiting the result

$$\int_{-\infty}^{\infty} \exp(-\tau^2) d\tau = \sqrt{\pi}.$$

The Fourier transform is given by

$$F(\omega) = \int_{-\infty}^{\infty} \exp(-at^2) \exp(-i\omega t) dt.$$

Noting that

$$\left(\sqrt{a}t + \frac{i\omega}{2\sqrt{a}}\right)^2 = at^2 + i\omega t - \frac{\omega^2}{4a}$$

we can write

$$F(\omega) = \exp(-\omega^2/4a) \int_{-\infty}^{\infty} \exp\left[-\left(\sqrt{a}t + \frac{i\omega}{2\sqrt{a}}\right)^2\right] dt.$$

If we now let

$$\tau = \left(\sqrt{a}t + \frac{i}{2\sqrt{a}}\right),$$

then $d\tau = \sqrt{a}dt$ and

$$F(\omega) = \frac{1}{\sqrt{a}} \exp(-\omega^2/4a) \int_{-\infty}^{\infty} \exp(-\tau^2) d\tau = \sqrt{\frac{\pi}{a}} \exp(-\omega^2/4a).$$

Fourier Transform of the Sign Function

The sign function $\text{sgn}(t)$ is defined by

$$\text{sgn}(t) = \begin{cases} 1, & t > 0; \\ -1, & t < 0. \end{cases}$$

The Fourier transform of this function can be obtained by computing the Fourier transform of $\exp(-\epsilon \mid t \mid) \text{sgn}(t)$ over the interval $[-\tau, \tau]$ say and then letting $\tau \to \infty$ and $\epsilon \to 0$. Thus,

$$\int_{-\infty}^{\infty} \text{sgn}(t) \exp(-i\omega t) dt$$

$$= \lim_{\epsilon \to 0} \lim_{\tau \to \infty} \left(\int_0^{\tau} \exp(-\epsilon \mid t \mid) \exp(-i\omega t) dt - \int_{-\tau}^{0} \exp(-\epsilon \mid t \mid) \exp(-i\omega t) dt \right)$$

$$= \lim_{\epsilon\to 0}\lim_{\tau\to\infty}\left(\int_0^\tau \exp(-\epsilon t)\exp(-i\omega t)dt - \int_0^\tau \exp(-\epsilon t)\exp(i\omega t)dt\right)$$

$$= \lim_{\epsilon\to 0}\lim_{\tau\to\infty}\left(\frac{\exp[-\tau(\epsilon - i\omega)]}{\epsilon - i\omega} - \frac{\exp[-\tau(\epsilon + i\omega)]}{\epsilon + i\omega} + \frac{1}{\epsilon + i\omega} - \frac{1}{\epsilon - i\omega}\right)$$

$$= \lim_{\epsilon\to 0}\left(\frac{-2i\omega}{(\epsilon + i\omega)(\epsilon - i\omega)}\right) = \frac{2}{i\omega}.$$

Hence,

$$\mathrm{sgn}(t) \Longleftrightarrow \frac{2}{i\omega}.$$

Fourier Transform of the Unit Step Function

The unit or Heaviside step function is defined by

$$U(t) = \begin{cases} 1, & t > 0; \\ 0, & t < 0. \end{cases}$$

To obtain the Fourier transform of this function, we write it in terms of the sign function, i.e.

$$U(t) = \frac{1}{2}[1 + \mathrm{sgn}(t)].$$

We then get,

$$\int_{-\infty}^{\infty} U(t)\exp(-i\omega t)dt = \frac{1}{2}\int_{-\infty}^{\infty}\exp(-i\omega t)dt + \frac{1}{2}\int_{-\infty}^{\infty}\mathrm{sgn}(t)\exp(-i\omega t)dt$$

$$= \pi\left(\delta(t) - \frac{i}{\pi\omega}\right).$$

Fourier Transform of 1/t

Using the result $\mathrm{sgn}(t) \Longleftrightarrow 2/i\omega$, we have

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{2}{i\omega}\exp(i\omega t)d\omega = \mathrm{sgn}(t).$$

Interchanging t and ω and changing the sign of i, this equation becomes

$$\int_{-\infty}^{\infty}\frac{1}{t}\exp(-i\omega t)dt = -i\pi\,\mathrm{sgn}(\omega)$$

and hence,

$$\frac{1}{t} \Longleftrightarrow -i\pi\,\mathrm{sgn}(\omega).$$

Note by taking the inverse Fourier transform of $-i\pi \operatorname{sgn}(\omega)$, we get

$$\frac{i}{\pi t} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \operatorname{sgn}(\omega) \exp(i\omega t) d\omega.$$

4.3 Selected but Important Theorems

Addition Theorem

The Fourier transform of the sum of two functions f and g is equal to the sum of their Fourier transforms F and G respectively.

Proof:

$$\begin{aligned}\int_{-\infty}^{\infty} [f(t) + g(t)] \exp(-i\omega t) dt &= \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt + \int_{-\infty}^{\infty} g(t) \exp(-i\omega t) dt \\ &= F(\omega) + G(\omega).\end{aligned}$$

Similarity Theorem

The Fourier transform of $f(at)$ is $(1/a)F(\omega/a)$ where a is a constant.

Proof:

$$\int_{-\infty}^{\infty} f(at) \exp(-i\omega t) dt = \frac{1}{a} \int_{-\infty}^{\infty} f(at) \exp\left(i\frac{\omega}{a}at\right) d(at) = \frac{1}{a} F\left(\frac{\omega}{a}\right).$$

Shift Theorem

The Fourier transform of $f(t-a)$ is given by $\exp(-i\omega a)F(\omega)$.

Proof:

$$\int_{-\infty}^{\infty} f(t-a) \exp(-i\omega t) dt = \int_{-\infty}^{\infty} f(t-a) \exp[-i\omega(t-a)] \exp(-i\omega a) d(t-a) = \exp(-i\omega a) F(\omega).$$

Parseval's Theorem

If f and g have Fourier transforms F and G respectively, then

$$\int_{-\infty}^{\infty} f(t) g^*(t) dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) G^*(\omega) d\omega$$

where g^* is the complex conjugate of g and G^* is the complex conjugate of G.

Proof:

$$\int_{-\infty}^{\infty} f(t)g^*(t)dt = \int_{-\infty}^{\infty} g^*(t)\left(\frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)d\omega\right)dt$$
$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\left(\int_{-\infty}^{\infty} g^*(t)\exp(i\omega t)dt\right)d\omega$$
$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\left(\int_{-\infty}^{\infty} g(t)\exp(-i\omega t)dt\right)^* d\omega$$
$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)G^*(\omega)d\omega.$$

Rayleigh's Theorem (also known as the energy theorem)

If $f \Longleftrightarrow F$, then

$$\int_{-\infty}^{\infty} |\, f(t)\,|^2\, dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} |\, F(\omega)\,|^2\, d\omega.$$

Proof:

The proof follows directly from setting $g = f$ in Parseval's theorem.

4.4 Convolution and Correlation

Convolution has been introduced in Chapter 2 where it was pointed out that this process is fundamental to the mathematical models and methods used in signal analysis. The process of correlation is very similar to that of convolution but it has certain properties that are critically different. In both cases, these processes are fundamentally associated with the Fourier transform, an association that is compounded in the convolution and correlation theorems.

Notation

Because the convolution and correlation integrals are of such importance and occur regularly, they are usually given a convenient notation. Throughout this work, convolution shall be denoted by the symbol $\otimes$ and correlation by the symbol $\odot$.

4.4.1 Convolution

The convolution of two functions f and g in one dimension is defined by the operation

$$f \otimes g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

where $f \otimes g$ is taken to be a function of t. This is a convolution over the interval $(-\infty, \infty)$ and is sometime written in the form $f(t) \otimes g(t)$ or $(f \otimes g)(t)$ to emphasize the fact that the operation is a function of the independent variable t.

If f and g are of finite extent $\mid t \mid \leq T$, then the convolution is finite and given by

$$f \otimes g = \int_{-T}^{T} f(\tau) g(t - \tau) d\tau.$$

Note that if $g(t) = \delta(t), \quad t \in (-\infty, \infty)$, then

$$f(t) \otimes g(t) = f(t) \otimes \delta(t) = f(t).$$

In other words, the convolution of a function with the delta function replicates the function. Also, note that if we let $t' = t - \tau$ then we can write

$$f \otimes g = \int_{-\infty}^{\infty} f(t - t') g(t') dt'$$

and hence, convolution is commutative, i.e.

$$f \otimes g = g \otimes f.$$

If f and g are both zero for $t < 0$, then

$$f \otimes g = \int_{0}^{\infty} f(\tau) g(t - \tau) d\tau.$$

The equation

$$h(t) = \int_{0}^{\infty} f(\tau) g(t - \tau) d\tau$$

is known as the Wiener-Hopf equation and is important in solving problems which are causal, i.e. problems that are governed by an initial condition at $t = 0$.

4.4.2 Correlation

The correlation (also known as cross-correlation) of two functions f and g in one dimension is defined by the operation

$$f \odot g = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) d\tau.$$

This is very similar to convolution except that the function g is a function of $\tau - t$ and not $t - \tau$, a seemingly small but very important difference. When the functions are complex, it is often useful to define the complex correlation operation

$$f^* \odot g = \int_{-\infty}^{\infty} f^*(\tau) g(\tau - t) d\tau.$$

The important difference between correlation and convolution is that in correlation, the function g is not reversed about the origin as in convolution. Note that for real functions

$$f(t) \otimes g(t) = f(t) \odot g(-t).$$

Also, note that if we let $t' = \tau - t$ then the correlation integral can be written as

$$f(t) \odot g(t) = \int_{-\infty}^{\infty} f(t + t')g(t')dt'.$$

Some authors prefer to define the correlation integral in this way where the independent variable of one of the functions is expressed in terms of an addition rather than a subtraction. However, note that the correlation integral, (unlike the convolution integral) is not generally commutative, i.e.

$$f \odot g \neq g \odot f.$$

4.4.3 Physical Interpretation

Physically, convolution can be thought of as a 'blurring' or 'smearing' of one function by another. Convolution integrals occur in a wide variety of physical problems. Referring to Chapter 2, they occur as a natural consequence of solving linear inhomogeneous partial differential equations using the Green's function method, i.e. if f is some source term for a linear PDE, then a solution of the form $f \otimes g$ can be developed where g is the Green's function. Convolution type processes are important in all aspects of signal and image analysis where convolution equations are used to describe signals and images of many different types. If we describe some system in terms of an input signal $f(t)$ and some output signal $s(t)$ and this system performs some process that operates on $f(t)$, then we can write

$$s(t) = \hat{P}f(t)$$

where $\hat{P}$ is the process operator. In many cases, this operator can be expressed in terms of a convolution with a so called Impulse Response Function p and we can write

$$s(t) = p(t) \otimes f(t).$$

This result is used routinely to model a signal (typically a recorded signal that is output from some passive or active 'system') when it can be assumed that the system processes are linear and stationary (i.e. p does not change with time). Such systems are referred to as time invariant linear systems.

4.4.4 Autoconvolution and Autocorrelation

Two other definitions which are important in the context of convolution and correlation are:

Autoconvolution

$$f \otimes f = \int_{-\infty}^{\infty} f(t)f(\tau - t)dt$$

and

Autocorrelation

$$f \odot f = \int_{-\infty}^{\infty} f(t) f(t-\tau) dt.$$

4.4.5 The Convolution Theorem

The convolution theorem is one of the most important results of Fourier theory. It can be stated thus:

The convolution of two functions in real space is the same as the product of their Fourier transforms in Fourier space.

The proof of this result can be obtained in a variety of ways. Here, we give a proof that is relatively 'short and sweet' and based on the definition for an inverse Fourier transform. Thus, writing

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega,$$

$$g(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} G(\omega) \exp(i\omega t) d\omega$$

we have

$$f \otimes g = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} dt \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega \int_{-\infty}^{\infty} G(\omega') \exp[i\omega'(\tau - t)] d\omega'$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} d\omega F(\omega) \int_{-\infty}^{\infty} d\omega' G(\omega') \exp(i\omega'\tau) \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp[it(\omega - \omega')] dt$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} d\omega F(\omega) \int_{-\infty}^{\infty} d\omega' G(\omega') \exp(i\omega'\tau) \delta(\omega - \omega') = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) G(\omega) \exp(i\omega\tau) d\omega$$

or

$$f(t) \otimes g(t) \Longleftrightarrow F(\omega) G(\omega).$$

4.4.6 The Product Theorem

The product theorem states that the product of two functions in real space is the same (ignoring scaling) as the convolution of their Fourier transforms in Fourier space. To prove this result, let

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$$

and

$$G(\omega) = \int_{-\infty}^{\infty} g(t) \exp(-i\omega t) dt.$$

Then

$$F \otimes G = \int_{-\infty}^{\infty} d\omega \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt \int_{-\infty}^{\infty} g(\tau) \exp[-i\tau(\omega' - \omega)] d\tau$$

$$= \int_{-\infty}^{\infty} dt f(t) \int_{-\infty}^{\infty} d\tau g(\tau) \exp(-i\omega'\tau) \int_{-\infty}^{\infty} \exp[-i\omega(t-\tau)] d\omega$$

$$= \int_{-\infty}^{\infty} dt f(t) \int_{-\infty}^{\infty} d\tau g(\tau) \exp(-i\omega'\tau) 2\pi\delta(t-\tau)$$

$$= 2\pi \int_{-\infty}^{\infty} dt f(t) g(t) \exp(-i\omega' t)$$

or

$$f(t)g(t) \Longleftrightarrow \frac{1}{2\pi} F(\omega) \otimes G(\omega).$$

4.4.7 The Correlation Theorem

The correlation theorem follows from the convolution theorem and can be written in the form

$$f(t) \odot g(t) \Longleftrightarrow F(\omega)G(-\omega)$$

for f and g real and

$$f(t) \odot g^*(t) \Longleftrightarrow F(\omega)G^*(\omega)$$

for f and g complex. Note that if g is a purely real function, then the real part of its Fourier transform $G(\omega)$ is symmetric and its imaginary part is asymmetric, i.e.

$$G_r(-\omega) = G_r(\omega)$$

and

$$G_i(-\omega) = -G_i(\omega).$$

In this case,

$$G(-\omega) = G_r(-\omega) + iG_i(-\omega) = G_r(\omega) - iG_i(\omega) = G^*(\omega)$$

and thus, for real functions f and g, we can write

$$f(t) \odot g(t) \Longleftrightarrow F(\omega)G^*(\omega).$$

4.4.8 The Autoconvolution and Autocorrelation Theorems

From the convolution theorem, we have

$$f(t) \otimes f(t) \Longleftrightarrow [F(\omega)]^2$$

and from the correlation theorem, we have

$$f(t) \odot f(t) \Longleftrightarrow \mid F(\omega) \mid^2 .$$

The last result has a unique feature which is that information about the phase of F is entirely missing from $\mid F \mid^2$ in contrast to the autoconvolution theorem where information about the phase of the spectrum is retained, i.e.

$$[F(\omega)]^2 = A_F^2(\omega) \exp[2i\theta_F(\omega)]$$

where A_F and θ_F are the amplitude and phase spectra of F respectively. Hence, the autocorrelation function $f \odot f$ contains no information about the phase of the Fourier components of f and is consequently unchanged if the phase changes.

4.4.9 Selected but Important Properties

1. Convolution is commutative, i.e.

$$f \otimes g = g \otimes f.$$

2. Convolution is associative, namely,

$$f \otimes (g \otimes h) = (f \otimes g) \otimes h.$$

Multiple convolutions can therefore be carried out in any order.

3. Convolution is distributive, or

$$f \otimes (g + h) = f \otimes g + f \otimes h.$$

4. The derivative of a convolution can be written as

$$\frac{d}{dx}[f(x) \otimes g(x)] = f(x) \otimes \frac{d}{dx} g(x) = g(x) \otimes \frac{d}{dx} f(x).$$

5. Correlation does not in general commute, i.e.

$$f \odot g \neq g \odot f$$

unless both f or g are symmetric functions.

Exercise Prove the results above using the convolution integral directly and then using the convolution theorem.

We conclude this section by providing a table of example Fourier transforms as given below.

Function	Fourier Transform
$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega$	$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$
$\exp(-at^2)$	$\sqrt{\frac{\pi}{a}} \exp(-\omega^2/4a)$
$\frac{1}{\sqrt{4\pi a}} \exp(-t^2/4a)$	$\exp(-a\omega^2)$
$\frac{df}{dt}$	$i\omega F(\omega)$
$\frac{d^2 f}{dt^2}$	$(i\omega)^2 F(\omega)$
$\int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$	$F(\omega)G(\omega)$
$\delta(t - t_0)$	$\exp(-i\omega t_0)$
$f(t-a)$	$\exp(-i\omega a)F(\omega)$
$tf(t)$	$i\frac{dF}{d\omega}$
$H(t)$	$2\frac{\sin(\omega T)}{\omega}$

Table 4.1: Table of example Fourier transforms

4.5 The Sampling Theorem

Digital signals are often obtained from analogue signals by converting them into a sequence of numbers (a digital signal) so that digital computers can be used to process them. This conversion is called digitization. When conversion takes place, it is a naturally common requirement that all the information in the original analogue signal is retained in digital form. To do this, the analogue signal must be sampled at the correct rate. So what is the correct rate? The answer to this question is provided by the sampling theorem. The sampling theorem states that if a continuous function $f(t)$ is bandlimited and has a complex spectrum $F(\omega)$, $\mid \omega \mid \leq \Omega$, then it is fully specified by values spaced at regular intervals

$$\delta t \leq \frac{\pi}{\Omega}.$$

The parameter Ω/π is important and is given a special name. It is called the 'Nyquist frequency'.

To convert an analogue signal into a digital signal with no loss of information, one must choose a sampling rate that is at least equal to the Nyquist frequency of the signal. To show why this is the case, the comb function (which was briefly introduced in Chapter 2) must first be considered. The comb function describes a train of impulses (delta functions) and is given by

$$\text{comb}(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT).$$

This function describes a sequence of delta functions spaced apart by a fixed period T as illustrated in Figure 4.1.

Sampling a function can be described mathematically by multiplying it by this comb function. Thus, if $f(t)$ is the bandlimited function and $g(t)$ is the sampled

function, then we can write

$$g(t) = \text{comb}(t) f(t).$$

The sampling theorem is obtain by analysing the spectrum of the sampled function $g(t)$.

4.5.1 Fourier Transform of the comb Function

The evaluation of $\hat{F}_1 \,\text{comb}(t)$ is important generally and is crucial to the proof of the sampling theorem to follow. Using the definition of comb(t), we can write

$$\int_{-\infty}^{\infty} \text{comb}(t) \exp(-i\omega t) dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - nT) \exp(-i\omega t) dt$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(t - nT) \exp(-i\omega t) dt = \sum_{n=-\infty}^{\infty} \exp(-i\omega nT).$$

Hence, using the product theorem,

$$\text{comb}(t) f(t) \Longleftrightarrow \frac{1}{2\pi} F(\omega) \otimes \sum_{n=-\infty}^{\infty} \exp(-i\omega nT).$$

Although valid, the above result is not in itself very useful. The key result which is required to prove the sampling theorem comes from expressing comb(t) as a complex Fourier series (not a transform). This can be done because, although it is a special case, the comb function is just a periodic function and thus, a Fourier series representation can be used. Hence, using the results discussed in Chapter 3, we consider writing comb in terms of a complex Fourier series given by

$$\text{comb}(t) = \sum_{n=-\infty}^{\infty} c_n \exp(i2\pi nt/T)$$

where the coefficients c_n are obtained by computing the integral

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} \text{comb}(t) \exp(-i2\pi nt/T) dt.$$

Substituting the definition for the comb function into the equation above and noting that comb$(t) = \delta(t)$ in the interval $[-T/2, T/2]$, we get

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} \delta(t) \exp(-i2\pi nt/T) dt = \frac{1}{T}.$$

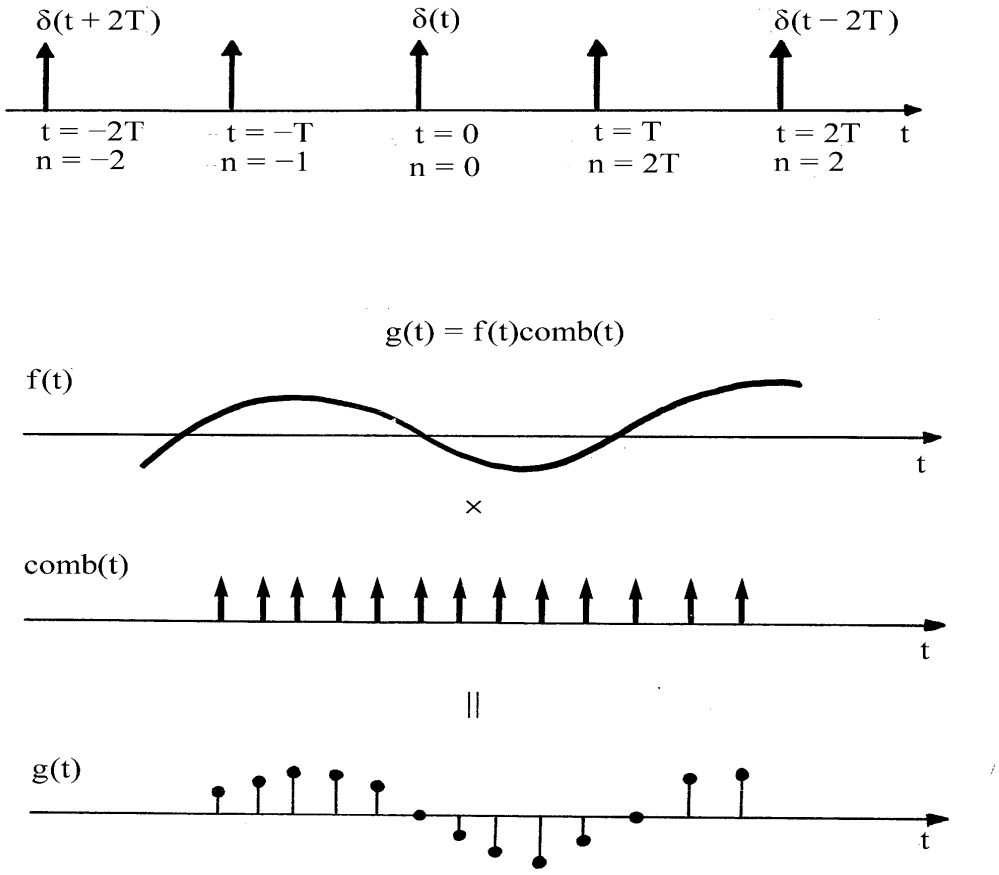


Figure 4.1: Illustration of the sampling of a function by multiplication with a comb function.

Hence, we can represent the comb function by the complex Fourier series

$$\text{comb}(t) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \exp(i2\pi nt/T)$$

and the Fourier transform of the comb function can now be written as

$$\int_{-\infty}^{\infty} \frac{1}{T} \sum_{n=-\infty}^{\infty} \exp(i2\pi nt/T) \exp(-i\omega t) dt$$

$$= \frac{1}{T} \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} \exp[-it(\omega - 2\pi n/T)]dt = \frac{2\pi}{T} \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n/T).$$

We have therefore obtained the fundamental and important result (crucial to the proof of the sampling theorem)

$$\sum_{n=-\infty}^{\infty} \delta(t - nT) \Longleftrightarrow \frac{2\pi}{T} \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n/T).$$

4.5.2 Proof of the Sampling Theorem

Suppose we sample a function at regular intervals δt. The sampled function $g(t)$ is then given by

$$g(t) = f(t) \sum_{n=-\infty}^{\infty} \delta(t - n\delta t).$$

Using the product theorem, in Fourier space, this equation becomes

$$G(\omega) = F(\omega) \otimes \frac{2\pi}{\delta t} \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n/\delta t)$$

$$= \frac{2\pi}{\delta t} \sum_{n=-\infty}^{\infty} F(\omega - 2\pi n/\delta t).$$

This result demonstrates that sampling the function f, creates a new spectrum G which is a periodic replica of the spectrum F spaced at regular intervals $\pm 2\pi/\delta t, \pm 4\pi/\delta t, \pm 6\pi/\delta t$ and so on as illustrated in Figure 4.2. Since F is a bandlimited function, the total width of the spectrum is $\Omega - (-\Omega) = 2\Omega$, i.e. the bandwidth of F. Thus, if $2\pi/\delta t < 2\Omega$, then the replicated spectra will overlap. This effect is known as aliasing. To ensure that aliasing does not occur, we require that

$$\frac{2\pi}{\delta t} \geq 2\Omega$$

or a sampling rate where

$$\delta t \leq \frac{\pi}{\Omega}.$$

A digital signal that has been sampled according to the condition

$$\delta t = \frac{\pi}{\Omega}$$

is called a Nyquist sampled signal where Ω/π is the Nyquist frequency (equal to twice the frequency bandwidth of the signal). This is the optimum sampling interval required to avoid aliasing and to recover the information of an analogue signal in digital form. It is the fundamental result used in all A-to-D (Analogue-to-Digital) conversion schemes.

4.5.3 Sinc Interpolation

If the condition given above provides the necessary sampling rate that is required to convert an analogue signal into a digital signal without loss of information, then one should be able to recover the analogue signal from the digital signal also without loss of information (i.e. undertake D-to-A or Digital-to-Analogue conversion). Assuming that f has been sampled at the Nyquist frequency, the only difference between g and f is that the spectrum of g consists of F repeated at regular interval $2\pi n/\delta t$; $n = \pm 1, \pm 2, \pm 3, ..., \pm\infty$. Clearly, f can be obtained from g by retaining just the part of G for values of $\mid \omega \mid$ less than or equal to Ω and setting all other values in the spectrum to zero, i.e.

$$F(\omega) = G(\omega)$$

provided we set

$$G(\omega) = 0 \quad \forall \mid \omega \mid > \Omega.$$

We can describe this process mathematically by multiplying G with the Tophat function

$$H(\omega) = \begin{cases} 1, & \mid \omega \mid \leq \Omega; \\ 0, & \mid \omega \mid > 0. \end{cases}$$

Thus, F is related to G by the equation

$$F(\omega) = H(\omega)G(\omega).$$

Using the convolution theorem, we then obtain

$$f(t) = 2\Omega\,\mathrm{sinc}(\Omega t) \otimes g(t).$$

This result describes the restoration of a continuous function $f(t)$ from a sampled function $g(t)$ and therefore demonstrates that a function can be interpolated by convolving it with the appropriate sinc function. This is known as sinc interpolation. In practice, a sampled function can be sinc interpolated by 'zero padding' its complex spectrum.

4.6 Fourier Filters

The design of filters is an inherent aspect of signal processing. Many of these (Fourier) filters operate on the spectrum of the signal. An operation which changes the distribution of the Fourier components of a function (typically via a multiplicative process) may be defined as a (Fourier) filtering operation. Thus, in an operation of the form

$$S(\omega) = P(\omega)F(\omega)$$

P may be referred to as a filter and S can be considered to be a filtered version of F. In this sense, many of the operations already discussed can be thought of as just filtering operations. For example, differentiation is characterized by the (highpass) filter

$$P(\omega) = i\omega$$

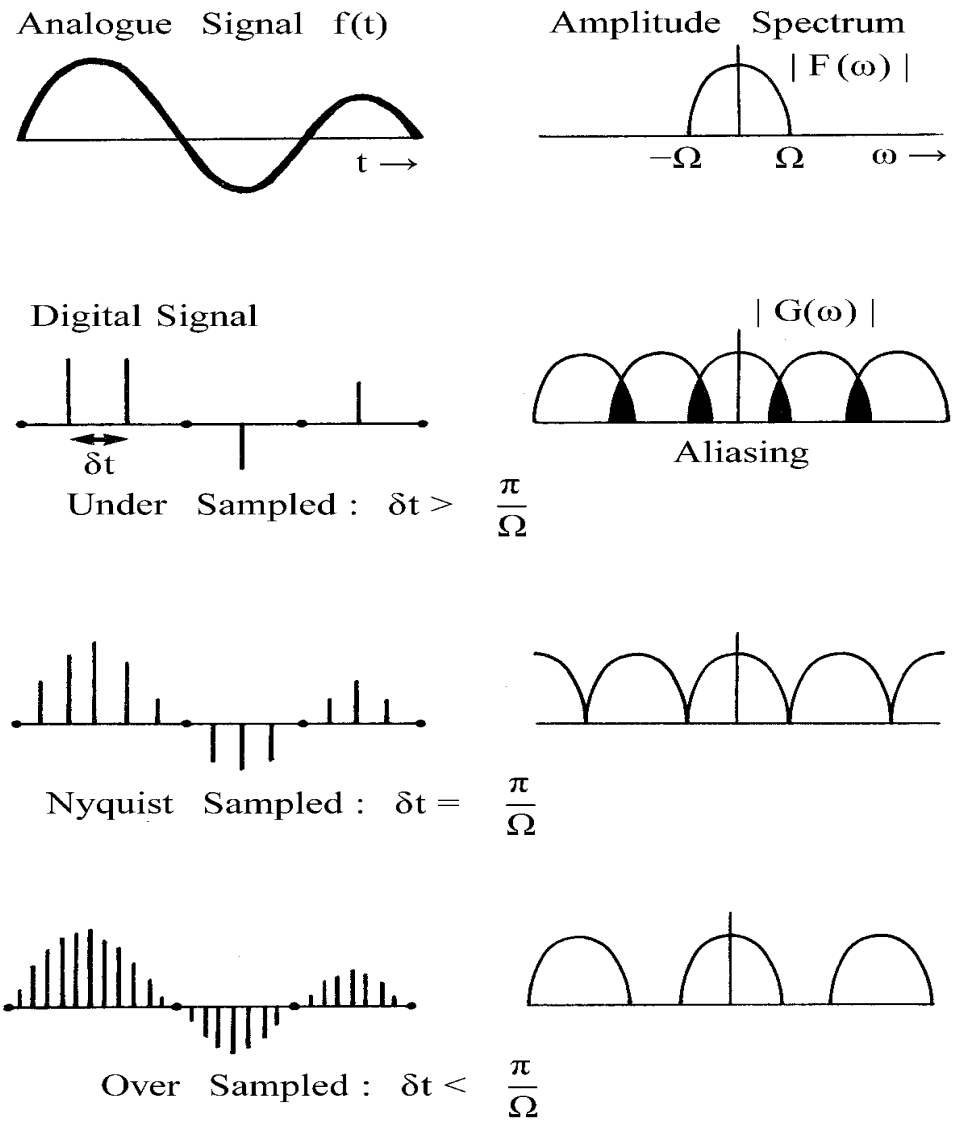


Figure 4.2: Illustration of the sampling theorem: The spectrum of the analogue signal becomes replicated when sampling takes place. If an analogue signal of bandwidth 2Ω is under sampled, then the digital signal does not contain the same amount of information and the data is aliased. The information that is lost is associated with that part of the spectrum which overlaps (the shaded regions). Aliasing is avoided by sampling the analogue signal at a rate that is greater or equal to π/Ω.

and integration is characterized by the (lowpass) filter

$$P(\omega) = \frac{1}{i\omega}.$$

In general, (Fourier) filters tend to fall into one of three classes: lowpass filters, highpass filters and bandpass filters.

4.6.1 Lowpass Filters

A lowpass filter is one which suppresses or attenuates the high frequency components of a spectrum while 'passing' the low frequencies within a specified range. Some examples of lowpass filters include the following:

(i) The ideal lowpass filter (the tophat function)

$$P(\omega) = \begin{cases} 1, & |\,\omega\,| \leq \Omega; \\ 0, & |\,\omega\,| > \Omega. \end{cases}$$

The bandwidth of the filter is 2Ω.

(ii) The Gaussian lowpass filter

$$P(\omega) = \exp(-\omega^2/\sigma^2)$$

where σ is the standard deviation of the filter, i.e. its half width when $P = \exp(-1)$.

(iii) The Butterworth lowpass filter

$$P(\omega) = \frac{1}{1 + \left(\frac{\omega}{\Omega}\right)^{2n}}, \quad n = 1, 2, ...$$

where Ω is the 'cut-off' frequency which defines the bandwidth of the filter and n is the 'order' of the filter which determines its 'sharpness'.

4.6.2 Highpass Filters

A highpass filter does exactly the opposite to a lowpass filter, i.e. it attenuates the low frequency components of a spectrum while 'passing' the high frequencies within a specified range. Some examples of highpass filters include the following:

(i) The ideal highpass filter

$$P(\omega) = \begin{cases} 0, & |\,\omega\,| < \Omega; \\ 1, & |\,\omega\,| \geq \Omega. \end{cases}$$

(ii) The Gaussian highpass filter

$$P(\omega) = \exp(\omega^2/\sigma^2) - 1.$$

(iii) The Butterworth highpass filter

$$P(\omega) = \frac{1}{1 + \left(\frac{\Omega}{\omega}\right)^{2n}}; \quad n = 1, 2, ..., \quad \omega \neq 0.$$

4.6.3 Bandpass Filters

A bandpass filter only allows those frequencies within a certain band to pass through. In this sense, lowpass and highpass filters are just special types of bandpass filters. Examples include:

(i) The ideal bandpass filter (with $\omega_0 > \Omega$)

$$P(\omega) = \begin{cases} 1, & \text{if} \quad \omega_0 - \Omega \leq \omega \leq \omega_0 + \Omega \quad \text{and} \quad -\omega_0 - \Omega \leq \omega \leq -\omega_0 + \Omega; \\ 0 & \text{otherwise.} \end{cases}$$

Here, ω_0 is the centre frequency of the filter which defines the location of the frequency band and Ω defines the bandwidth.

(ii) The quadratic-Gaussian bandpass filter given by

$$P(\omega) = \omega^2 \exp(-\omega^2/\sigma^2).$$

Note that as σ increases, the frequency band over which the filter operates increases. In general, bandpass filters have at least two control parameters; one to adjust the bandwidth and another to adjust the position of the band.

A large number of signals can be modelled in terms of some bandpass filter modifying the distribution of the (complex) Fourier components associated with an information source. Processing may then be required to restore the out-of-band frequencies in order to recover the complex spectrum of the source. This requires a good estimate of the original bandpass filter.

4.6.4 The Inverse Filter

Suppose that $S(\omega)$ and $P(\omega)$ are known functions and it is required to find $F(\omega)$ from which $f(t)$ can be computed (via the inverse Fourier transform). In this case

$$F(\omega) = \frac{S(\omega)}{P(\omega)}.$$

Here, $1/P(\omega)$ is known as the inverse filter and can be written in the form

$$\frac{P^*(\omega)}{\mid P(\omega) \mid^2}, \quad \mid P(\omega) \mid \neq 0.$$

4.7 The Kronecker Delta Function and the DFT

Given the analysis provided in the previous chapter and the results discussed in this chapter, and coupled with the fact that many of the results and ideas developed using the Fourier transform can be implemented using the Discrete Fourier Transform or DFT, it is pertinent to ask how the results developed here can be rigorously justified using the DFT. In other words, how can we develop results (such as the convolution theorem for example) that are based explicitly on the use of the DFT. If we consider a generalized approach, then it is imperative that we consider a discrete version of the delta function since nearly all of the results discussed in this chapter have been based on using the generalized Fourier transform (compared with those discussed in the previous chapter which are based on a classical approach) which in turn have been dependent on the application of the delta function and its properties. The solution to this problem is based on the introduction of the so called Kronecker delta function which can be defined as

$$\delta_{nm} = \begin{cases} 1, & n = m; \\ 0, & n \neq m. \end{cases}$$

to which we can extend the following properties:

$$\sum_n f_n \delta_{nm} = f_m$$

and

$$\delta_{nm} = \frac{1}{N} \sum_k \exp[2\pi i k(n-m)/N]$$

where N is the array size and the limits of the sums are taken to run from $-\infty$ to ∞. The first of these properties is the definition of this discrete delta function in terms of its fundamental sampling property. The second of these properties can be derived directly from a discretization of the integral representation of the delta function, i.e.

$$\delta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(i\omega t) d\omega.$$

Thus, consider

$$\delta(t_n) = \frac{1}{2\pi} \sum_m \exp(i\omega_m t_n) \Delta\omega$$

which can be written in the form

$$\delta(t_n) = \frac{1}{2\pi} \sum_m \exp(imn\Delta\omega\Delta t) \Delta\omega$$

where Δt and $\Delta\omega$ denote the sampling intervals between the elements t_n and ω_m respectively. Now, with

$$\Delta\omega\Delta t = \frac{2\pi}{N},$$

$$\delta(t_n) = \frac{1}{\Delta t N} \sum_m \exp(2\pi i n m/N)$$

or

$$\Delta t \delta(t_n) = \frac{1}{N} \sum_m \exp(2\pi i n m/N).$$

Hence, defining the Kronecker delta function as $\delta_n \equiv \delta(t_n)\Delta t$ we obtain the desired result.

With these results, we can derive a number of important results in a way that is directly analogous to those derived for the Fourier transform. For example, suppose we want to prove the convolution theorem for the DFT, i.e. show that

$$f_n \otimes g_n \Longleftrightarrow F_n G_n$$

where

$$f_n \otimes g_n = \sum_n f_n g_{m-n}$$

and F_n and G_n are the DFTs of f_n and g_n respectively. Let

$$f_n = \frac{1}{N} \sum_m F_m \exp(2\pi i n m/N)$$

and

$$g_n = \frac{1}{N} \sum_m G_m \exp(2\pi i n m/N).$$

Then

$$\sum_n f_n g_{m-n} = \frac{1}{N^2} \sum_n \sum_k F_k \exp(2\pi i k n/N) \sum_\ell G_\ell \exp[2\pi i \ell (m-n)/N]$$

$$= \frac{1}{N} \sum_k F_k \sum_\ell G_\ell \exp(2\pi i \ell m/N) \sum_n \exp[2\pi i n (k-\ell)/N]$$

$$= \frac{1}{N} \sum_k F_k \sum_\ell G_\ell \exp(2\pi i \ell m/N) \delta_{k\ell}$$

$$= \frac{1}{N} \sum_k F_k G_k \exp(2\pi i k m/N).$$

Hence,

$$f_n \otimes g_n \Longleftrightarrow F_n G_n.$$

4.8 Deriving Filters using the DFT

Digital filters can be derived from application of the DFT in much the same way as analogue filters can from application of the Fourier transform. For example, consider the differentiation of a digital signal using forward differencing and whose digital gradient is given by (ignoring scaling by the 'step length')

$$g_n = f_{n+1} - f_n.$$

Now, with

$$f_n = \frac{1}{N}\sum_m F_m \exp(i2\pi nm/N),$$

we have

$$g_n = \frac{1}{N}\sum_m F_m \exp(i2\pi nm/N)[\exp(i2\pi m/N) - 1]$$

which shows that the DFT filter for this operation is given by

$$\exp(i2\pi m/N) - 1 = [\cos(2\pi m/N) - 1] + i\sin(2\pi m/N).$$

Similarly, the DFT filter that characterizes the center differenced approximation to a second order derivative (i.e. $f_{n+1} - 2f_n + f_{n-1}$) is given by

$$\exp(i2\pi m/N) + \exp(-i\pi m/N) - 2 = 2[\cos(2\pi m/N) - 1].$$

Note that these filters differ from the application of Fourier filters in discrete form, i.e. $i\omega_m$ and $-\omega_m^2$. In particular, their response at high frequencies is significantly different. Also, note that a process such as $f_{n+1} - 2f_n + f_{n-1}$ for example can be considered in terms of a discrete convolution of the signal f_n with $(1, -2, 1)$ and in this sense we can write

$$(1, -2, 1) \otimes f_n \Longleftrightarrow 2[\cos(2\pi m/N) - 1]F_m.$$

Processes of this type (i.e. discrete convolutions and other 'moving window' operations) are discussed extensively in Part IV of this work.

We conclude this chapter with a schematic diagram of the relationships between the Fourier transform, the complex Fourier series (Chapter 3) and the discrete Fourier transform as given below. This diagram illustrates qualitatively the principal properties associated with each of these approaches to Fourier analysis which should be born in mind when their utilities are exercised. In particular, the relationship between the Fourier transform and the corresponding discrete Fourier transform (and hence between analogue and digital signal processing) should be understood in terms of the fact that the DFT does not produce identical results (due to numerical error associated with the discretization of arrays that are of finite duration) to those of the Fourier transform. Nevertheless, the use of Fourier theory can be used to investigate the analysis of (analogue) signals to design processing methods that can be implemented using the DFT. Details of the numerical considerations that need to be addressed with regard to this statement are discussed in Part IV of this book.

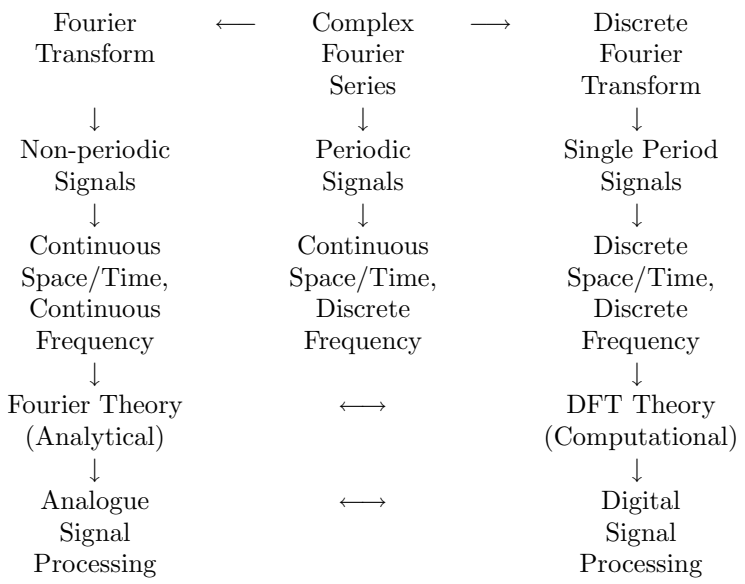


Diagram 4.1: Schematic diagram of the relationship between the Fourier transform, the (complex) Fourier series and the discrete Fourier transform (DFT).

4.9 Case Study: Scattering from Layered Media

This case study is devoted to illustrating how the Fourier transform, the convolution process and the reflection or (back) scattering of waves are related and further, how the equation $s(t) = p(t) \otimes f(t)$ for modelling a signal can be established from first principles. Thus, we consider the case where a pulse of radiation is normally incident upon a material which has a layer The pulse can be considered to be an electromagnetic pulse reflected from a layered dielectric or an acoustic pulse in which a spectrum of acoustic waves are reflected from a layered structure with variations in density for example. In any case, we assume that a pulsed radiation field or wavefield is emitted and recorded through some appropriate technology (depending of the frequency range that is being used) and that the time history of the reflected field is obtained giving a signal that can then be digitized for appropriate processing in order to perhaps analyse the layered structure under investigated. This is a classical example of the type of situation that occurs in many areas of non-destructive evaluation, radar, seismology and medical ultrasonic imaging to name but a few. The problem is to develop a suitable model for the recorded signal. The approach that is considered here is not confined to layered materials but this assumption provides a method of analysis that can be confined to a one-dimensional model.

4.9.1 Introduction to the Wave Equation

We start be considering the wave equation which is a second order partial differential (hyperbolic) equation of the form

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x,k) = 0.$$

Here, u describes the wavefield which is a function of space x and the wavenumber k which in turn is given by

$$k = \frac{\omega}{c}$$

where ω is the angular frequency ($=2\pi\times$ frequency) and c is the speed at which the wavefield propagates. If the medium through which the wavefield propagates is homogeneous with no variations of the wavespeed c, then the above equation is homogeneous with a solution of the form

$$u(x,k) = P\exp(-ikx)$$

where P is the amplitude of the wavefield. Note that this solution holds if P is a function of k and so we can write

$$u(x,k) = P(k)\exp(-ikx).$$

Here, $P(k)$ can be taken to describe the spectrum of the pulse that is emitted. If P is a constant forall $k \in (-\infty, \infty)$, then the pulse becomes an ideal impulse or delta function. On the other hand, if P only exist for $k = k_0$ say, then the wavefield can be taken to be continuous - a Continuous Wave or CW field.

Suppose that at some position along x, material is introduced, that in effect, changes the wave speed. For example, suppose a dielectric is introduced with permittivity ϵ, then the electromagnetic wave speed will change from

$$c_0 = \frac{1}{\sqrt{\epsilon_0\mu_0}} \quad \text{to} \quad c = \frac{1}{\sqrt{\epsilon\mu_0}}$$

where ϵ_0 and μ_0 are the permittivity and permeability of free space. A change in the wave speed will of course lead to a change in the refractive index leading directly to partial reflection of a light wave from a glass plate for example. Similarly, if the wavefield was an acoustic field, and a change in density ρ was introduced, then the acoustic wave speed would change from

$$c_0 = \frac{1}{\sqrt{\rho_0\kappa_0}} \quad \text{to} \quad c = \frac{1}{\sqrt{\rho\kappa_0}}$$

where ρ_0 and κ_0 are the density and compressibility of a homogeneous space. Note that in electromagnetics, if ϵ_0 is taken to be the permittivity of a vacuum then $\epsilon > \epsilon_0$ always. However, in acoustics it is possible for the density ρ to be both greater or less than the ambient density ρ_0, i.e. the speed of propagation of an acoustic wave can increase or decrease above or below the ambient velocity, but in electromagnetics, the wave speed can only decrease form that of the speed of light.

In addition to introducing a material at some position x, suppose we now extend the problem and consider the material to have a characteristic profile defined by $c(x)$. Further, let us consider this profile to be defined in terms of a perturbation of an ambient constant velocity c_0 so that

$$c(x) = c_0 + v(x).$$

The effect of introducing an inhomogeneity characterized by $v(x)$ is to produce a change or perturbation to the behaviour of the wavefield u. Suppose we model this change in a similar way to that of the speed profile, i.e. consider

$$u(x,k) = u_0(x,k) + w(x,k)$$

where $u_0(x,k) = P(k)\exp(-ikx)$ which is the solution to the homogeneous equation

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u_0(x,k) = 0.$$

Having defined the problem above, we shall now apply suitable approximations to the problem in order to establish a first order solution that in many cases is valid physically (a solution that is actually compounded in a single convolution). The principal conditions are that v and w are small perturbations to c_0 and u_0 respectively, i.e. $v(x) << c_0, \forall x$ and $w(x,k) << u_0(x,k), \forall x$ and k. In other words, the change in the wave speed generated by an inhomogeneity is relatively small leading to a small or weak effect on the wavefield. The latter condition is referred to as the weak scattering condition and is generally attributed to Max Born who was the first to introduce such an approximation applied to quantum scattering theory in the 1930s. These conditions allow us to formulate a modified inhomogeneous wave equations as follows. First, we evaluate $1/c^2$, thus,

$$\frac{1}{c^2} = \frac{1}{(c_0+v)^2} = \frac{1}{c_0^2}\left(1 + \frac{v}{c_0}\right)^{-2} = \frac{1}{c_0^2}\left(1 - \frac{2v}{c_0} + ...\right) \simeq \frac{1}{c_0^2} - \frac{2v}{c_0^3}$$

since $v/c_0 << 1$ and the nonlinear terms in the binomial expansion given above can therefore be neglected. Now,

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\omega^2}{c_0^2} - \omega^2\frac{2v}{c_0^3}\right)(u_0 + w)$$

$$= \left(\frac{\partial^2}{\partial x^2} + \frac{\omega^2}{c_0^2}\right) u_0 + \left(\frac{\partial^2}{\partial x^2} + \frac{\omega^2}{c_0^2}\right) w - \frac{2\omega^2}{c_0^3}(vu_0 + vw)$$

$$= \left(\frac{\partial^2}{\partial x^2} + \frac{\omega^2}{c_0^2}\right) w - \frac{2\omega^2}{c_0^3}(vu_0 + vw)$$

$$\simeq \left(\frac{\partial^2}{\partial x^2} + \frac{\omega^2}{c_0^2}\right) w - \frac{2v\omega^2}{c_0^3}u_0 = 0$$

or

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) w = k^2\frac{2v}{c_0}u_0$$

where

$$k = \frac{\omega}{c_0}.$$

This equation facilitates a first order approximation to the solution of the inhomogeneous Helmholtz equation, i.e.

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x,k) = k^2 f(x) u(x,k)$$

where u is the wavefield and $f(x)$ is the inhomogeneity. Note that this equation can be developed directly from the electromagnetic or acoustic field equations. In doing so, the functional form of $f(x)$ will be defined directly in terms of the electromagnetic and acoustic 'scatter generating parameters' (i.e. the permittivity, permeability in electromagnetic problems and the density and compressibility in acoustics).

4.9.2 Asymptotic Green's Function Solution

Referring to the Green's function method discussed in Chapter 2 (see Question 2.9), the solution for w can be written in the form (for right travelling waves and where v is now a dimensionless quantity used to denote the quotient v/c_0)

$$\begin{aligned} w(x_0,k) &= 2k^2 \int_{-\infty}^{\infty} g(x \mid x_0,k) v(x) u_0(x,k) dx \\ &= 2k^2 \int_{-\infty}^{\infty} \tfrac{i}{2k} \exp(ik \mid x - x_0 \mid) v(x) P(k) \exp(-ikx) dx \\ &= ikP(k) \int_{-\infty}^{\infty} \exp(ik \mid x - x_0 \mid) v(x) \exp(-ikx) dx. \end{aligned}$$

Note that using $\otimes$ to denote the convolution operation, we can write this result in the form

$$w(x,k) = 2k^2 P(k) g(\mid x \mid, k) \otimes v(x) \exp(-ikx).$$

Thus, we observe that the basic solution to this problem is compounded in a convolution operation involving the Green's function. Now, as $x_0 \longrightarrow \infty$, $\mid x - x_0 \mid \longrightarrow x_0 - x, \quad \forall x \in (-\infty, \infty)$ and hence,

$$w(x_0,k) = \exp(ikx_0) ikP(k) \int_{-\infty}^{\infty} \exp(-2ikx) v(x) dx$$

and the solution for the wavefield u required is thus given by

$$u(x_0,k) = u_0(x_0,k) + w(x_0,k)$$

$$= P(k) \exp(-ikx_0) + \exp(ikx_0) ikP(k) \int_{-\infty}^{\infty} \exp(-2ikx) v(x) dx.$$

This result expresses the wavefield in term of the sum of the incident field $P(k)\exp(-ikx_0)$ and the reflected wavefield $\exp(ikx_0)S(k)$ where S is the reflection coefficient which

can be written in terms of the angular frequency ω as

$$S(\omega) = P(\omega) i\omega \int_{-\infty}^{\infty} \exp(-2i\omega t) v(t) dt$$

or, replacing t by $t/2$,

$$S(\omega) = \frac{1}{2} P(\omega) i\omega \int_{-\infty}^{\infty} \exp(-i\omega t) v(t/2) dt.$$

Here, we see that the spectrum of the reflected signal is characterized by the Fourier transform of the velocity profile which is a function of the 'half-way' travel time. We can therefore write

$$S(\omega) = \frac{1}{2} P(\omega) i\omega V(\omega)$$

where

$$V(\omega) = \int_{-\infty}^{\infty} v(t/2) \exp(-i\omega t) dt.$$

Finally, if we Fourier invert this equation, then, noting that

$$i\omega \Longleftrightarrow \frac{d}{dt}$$

and using the convolution theorem, we get

$$s(t) = p(t) \otimes f(t)$$

where

$$f(t) = \frac{1}{2} \frac{d}{dt} v(t/2).$$

Thus, we arrive at the convolution model for the reflected signal which demonstrates that the amplitude modulations of this signal are not determined by the velocity profile itself but by its first derivative. Thus, sharp changes in the velocity profile of a layered material will produce large amplitude variations in the reflected signal. Finally, note that s, p and f are functions of t which is actually the two-way travel time, since, replacing $t/2$ by t, we have

$$s(2t) = p(2t) \otimes f(2t)$$

where

$$f(2t) = \frac{1}{4} \frac{d}{dt} v(t)$$

in which the signal is a function of the time taken for the wavefield to propagate to the inhomogeneity from the source and return back to the detector after interacting (i.e. reflecting) from it.

4.9.3 Discussion

The purpose of this case study has been to show how it is possible to produce the convolution model for a signal based on a more fundamental appreciation of a physical problem. In undertaking this study, it has been shown that the convolution model is a direct result of applying conditions on the strength of the wavefield that define a physical process in which only weak or single interactions (i.e. reflections) take place. Thus, with regard to active pule-echo type signal and imaging systems, the convolution model is actually based on the Born scattering approximation in which multiple scattering events are neglected. Finally, it is of value to quantify the condition under which this approximation holds, namely that w is small compared to u_0 or that

$$\|w(x,k)\| << \|u_0(x,k)\|$$

where we consider the norm (for $x \in L$)

$$\|f(x)\| \equiv \left(\int_L \mid f(x) \mid^2 dx \right)^{\frac{1}{2}}.$$

Now

$$\|w(x_0,k)\| = \|2k^2 \int_{-\infty}^{\infty} g(x \mid x_0,k) v(x) u_0(x,k) dx\|$$

$$\leq 2k^2 \|u_0(x_0,k)\| \times \| \int_{-\infty}^{\infty} g(x \mid x_0,k) v(x) dx\|$$

and the condition for weak scattering can be written as

$$2k^2 \| \int_{-\infty}^{\infty} g(x \mid x_0,k) v(x) dx\| << 1.$$

We can progress further by noting that

$$\|g(\mid x \mid,k) \otimes v(x)\| \leq \|g(\mid x \mid,k)\| \times \|v(x)\|$$

and that

$$\|g(\mid x \mid,k)\| = \frac{\sqrt{L}}{2k}.$$

Thus, it is clear that our condition reduces to

$$\|v(x)\| << \frac{1}{k\sqrt{L}}.$$

Finally, suppose that $v(x)$ is of compact support $x \in [-L/2, L/2]$ where L is the scale length over which the inhomogeneity exists. Then, by defining the root mean square value to be given by (noting that v is real)

$$\langle v^2 \rangle \equiv \left(\frac{1}{L} \int_{-L/2}^{L/2} [v(x)]^2 dx \right)^{\frac{1}{2}}$$

we can write (noting that $k = 2\pi/\lambda$ and ignoring scaling by 2π)

$$\langle v^2 \rangle << \lambda.$$

This result demonstrates that for arbitrary values of $\|v\|$, the wavelength needs to be significantly large compared to the scale length of the inhomogeneity (specifically, the square root). However, if an experiment is undertaken in which the wavelength of the wavefield is of the same order of magnitude as the scale length of the scatterer (which is usual in pulse-echo systems where a wavelength of the order of the scale length of the scatterer is required) then with $\lambda \sim 2\pi\sqrt{L}$ say, we have

$$\|v\| << 1.$$

Now, v is actually the dimensionless quantity v/c_0 where c_0 is the ambient velocity and v is the change in this velocity. Thus, provided v is a relatively small perturbation to the velocity of free space, the Born approximation holds, reducing the problem to a time invariant convolution as shown above. Note that this condition is consistent with the condition used earlier on to develop an approximate form for $(c_0+v)^{-2}$ using a binomial expansion and the result is therefore self-consistent.

4.10 Summary of Important Results

Convolution Theorem

$$f(t) \otimes g(t) \Longleftrightarrow F(\omega)G(\omega)$$

Product Theorem

$$f(t)g(t) \Longleftrightarrow \frac{1}{2\pi}F(\omega) \otimes G(\omega)$$

Correlation Theorem

$$f(t) \odot g(t) \Longleftrightarrow F(\omega)G^*(\omega)$$

Autocorrelation Theorem

$$f(t) \odot f(t) \Longleftrightarrow \mid F(\omega) \mid^2$$

Sampling Theorem

$$f(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \Longleftrightarrow \frac{2\pi}{T} \sum_{n=-\infty}^{\infty} F(\omega - 2\pi n/T)$$

where $f(t)$ is a bandlimited function with (angular) bandwidth Ω.

Nyquist Sampling Rate

$$T = \frac{\pi}{\Omega}$$

Nyquist Frequency

$$\Omega/\pi = 2 \times \text{Bandwidth} \quad \text{frequency}$$

Sinc Interpolation

$$f(t) = 2\Omega \text{sinc}(\Omega t) \otimes f(t) \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

4.11 Further Reading

- Bateman H, *Tables of Integral Transforms*, McGraw-Hill, 1954.
- Papoulis A, *The Fourier Integral and its Applications*, McGraw-Hill, 1962.
- Bracewell R N, *The Fourier Transform and its Applications*, McGraw-Hill, 1978.
- Oppenheim A V, Willsky A S and Young I T, *Signals and Systems*, Prentice-Hall, 1983.
- Kraniauskas P, *Transforms in Signals and Systems*, Addison-Wesley, 1992.

4.12 Problems

4.1 Show that the Fourier transform of the function

$$f(t) = \begin{cases} t, & \mid t \mid \leq a; \\ 0, & \mid t \mid > a. \end{cases}$$

is

$$F(\omega) = \begin{cases} \frac{2ia}{\omega}\left(\cos \omega a - \frac{\sin \omega a}{\omega a}\right), & \omega \neq 0; \\ 0, & \omega = 0. \end{cases}$$

4.2 Find the Fourier transform of the 'comb function'

$$\text{comb}(t) = \sum_{i=1}^{n} \delta(t - t_i).$$

4.3 Find the Fourier transform of

$$f(t) = \begin{cases} 1 - \frac{|t|}{a}, & \mid t \mid \leq a; \\ 0, & \mid t \mid > a. \end{cases}$$

and by inversion, prove that

$$\int_{-\infty}^{\infty} \frac{\sin^2 t}{t^2} dx = \pi.$$

4.4 Find the Fourier transform of

$$f(x) = \begin{cases} 1/2a, & \mid x \mid \leq a; \\ 0, & \mid x \mid > a. \end{cases}$$

and by considering the limit as $a \to 0$, verify that

$$\int_{-\infty}^{\infty} \delta(t) \exp(-i\omega t) dt = 1.$$

4.5 Find the Fourier transform of $\exp(- \mid t \mid)$ and, by inversion, prove that

$$\int_{-\infty}^{\infty} \frac{\cos t}{1+t^2} dx = \frac{\pi}{e}.$$

4.6 Find the Fourier transform of the function

$$f(t) = \begin{cases} 1, & \mid t \mid < a; \\ 0, & \mid x \mid > a. \end{cases}$$

and by applying the convolution theorem to the Fourier transform of $[f(t)]^2$, show that

$$\int_{0}^{\infty} \frac{\sin^2 t}{t^2} dt = \frac{\pi}{2}.$$

4.7 Show that

$$f(t) \otimes \exp(i\omega t) = F(\omega) \exp(i\omega t).$$

Hence, prove the convolution theorem for Fourier transforms.

4.8 Show that if α is small enough, then

$$f(t) \otimes \exp(i\alpha t^2) \simeq [F(2\alpha t) - i\alpha F''(2\alpha t)] \exp(i\alpha t^2).$$

4.9 If $f(t) \Longleftrightarrow F(\omega)$ and $1/\pi t \Longleftrightarrow -i \operatorname{sgn}(\omega)$ where

$$\operatorname{sgn}(\omega) = \begin{cases} 1, & \omega > 0; \\ -1, & \omega < 0. \end{cases}$$

show that

$$f'(t) \otimes \frac{1}{\pi t} \Longleftrightarrow \mid \omega \mid F(\omega),$$

$$\frac{1}{\pi t^2} \Longleftrightarrow \mid \omega \mid$$

and

$$-\frac{1}{\pi t} \otimes \frac{1}{\pi t} = \delta(t).$$

4.10 Show that

$$f_n \odot g_n \Longleftrightarrow F_n G_n^*$$

where

$$f_n \odot g_n = \sum_n f_n g_{n+m}$$

and that

$$f_n g_n \Longleftrightarrow \frac{1}{N} F_n \otimes G_n$$

where

$$F_n \otimes G_n = \sum_n F_n G_{m-n}.$$

4.11 Show that the inverse solution for f to the equation

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x,k) = k^2 f(x) u(x,k)$$

is

$$f(x) = \frac{1}{u(x,k)} \left(\frac{\partial^2}{\partial x^2}(s(x) \otimes [u(x,k) - u_0(x,k)] + \frac{1}{k^2}[u(x,k) - u_0(x,k)])\right),$$

$$\mid u(x,k) \mid > 0$$

where u_0 is the solution to

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u_0 = 0$$

and

$$s(x) = \begin{cases} x, & x \geq x_0; \\ 0, & x < x_0. \end{cases}$$

Chapter 5

Other Integral Transforms

Although it is fair to state that the Fourier transform is absolutely fundamental to signal analysis, there are a number of other integral transforms that play an important role in signal analysis, simulation and processing. These transforms include: the Laplace transform, the sine and cosine transforms, the Hilbert transform, the z-transform, the Wigner and Wigner-Ville transform, the Riemann-Liouville or fractional integral transform and the wavelet transform. In this chapter, we briefly introduce these transforms and discuss some of their properties.

5.1 The Laplace Transform

The single sided Laplace transform (usual case) is given by

$$F(p) = \hat{L}f(t) \equiv \int_0^\infty f(t)\exp(-pt)dt$$

and the double sided Laplace transform is defined as

$$F(p) = \int_{-\infty}^\infty f(t)\exp(-pt)dt.$$

The Laplace transform together with the Fourier transform is arguably the mathematical signature of the communications engineer and electrical engineering in general and it has a long history of applications to problems in engineering. Like the Fourier transform, it changes or transforms some of the most important differential equations of physics into algebraic equations, which are generally much easier to solve. The essential difference, is that the (one-sided) Laplace transform can be used to solve causal problems, i.e. those that are characterized by initial conditions. This is not the only transform that can be used to solve causal problems. Both the Fourier sine and cosine transforms have applications in this respect. However, these transforms are not unique to this type of problem solving either. The essential difference lies in whether a two-sided transform (where the integral 'runs' from $-\infty$ to ∞) or a

single-sided transform (where the limits of the integral are from 0 to ∞, in the so called 'positive half-space').

By 1785, Laplace was working with transforms of the type

$$F(p) = \int_0^\infty t^p f(t) dt$$

which is today used to solve certain differential equations with variable coefficients for example and is known as the Mellin transform. In 1807, Laplace's fellow countryman, Joseph Fourier, published the first monograph describing the heat equation which is more commonly known as the diffusion equation and Laplace attempted to solve this equation in a way that inspired Fourier to developed his own transform. The connection between the Fourier transform to study the power and energy spectra of signals and the Laplace transform is intimate but they are not equivalent. While the Fourier transform is useful for finding the steady state output of a linear circuit in response to a periodic input for example (see Chapter 3), the Laplace transform can provide both the steady state and transient responses for periodic and aperiodic signals. However, one of the most important features of both transforms is their ability to map the convolution operation into a multiplicative operation in Fourier or Laplace space. Interestingly and fundamentally, the convolution integral also plays a central role in the theory of random variables; namely, if $P_f(x)$ and $P_g(x)$ are the probability density functions of two independent random variables or signals $f(t)$ and $g(t)$, then the probability density function of the random variable $h(t) = f(t) + g(t)$ is given by the convolution of $P_f(x)$ with $P_g(x)$ - a result that was first introduced by Gauss using the Laplace transform. Moreover, if we have a number of independent random functions $f_1(t), f_2(t), ..., f_n(t)$, then in the limit as $n \to \infty$, the probability density function of the function $f_1(t) + f_2(t) + ... + f_n(t)$ is given by a Gaussian or normal distribution, a result that is fundamental to so many statistical systems and is commonly referred to as the Central Limit Theorem.

Another important development involving the Laplace transform was first advanced by the rather eccentric Englishman Oliver Heaviside (1850-1925) who reduced differential equations directly to algebraic ones by representing time differentiation as an operator. He used p, as in $px \equiv dx/dt$ and $p^2x \equiv d^2x/dt^2$ and $1/p(x) \equiv \int x dt$, and then manipulated these equations using any algebraic trick that could be invented, including his famous Heaviside expansion theorem which is essentially the partial fraction expansion of modern Laplace theory. Heaviside also introduced fractional operators such as $p^{1/2}$ which can be used to investigate properties relating to the fractional calculus which is explored in Part IV (Chapter 17). Toward the end of his life, Heaviside communicated with John Bromwich (1875-1929), who provided greater rigor to Heaviside's work in terms of complex, Laplace-like integrals such as the Bromwich integral which is in effect the inverse Laplace transform. On this note, it is important to understand that the Fourier transform has one major advantage over the Laplace transform, which is that the form of the inverse Fourier transform is the same as the forward Fourier transform. In terms of applications to signal processing, the Fourier transform is more readily applicable because it is rare that data is provided subject to a set of initial conditions, i.e. signal processing is essentially a non-causal 'art-form' and for this reason, together with the intrinsic symmetry of

the Fourier transform, the application of the Laplace transform is not as extensive. Nevertheless, the Laplace transform has and continues to have value in some forms of signal and circuit analysis.

Some Elementary Laplace Transforms

(i) The function $f(t) = 1 \forall t \geq 0$ (the Laplace transform of unity) has the Laplace transform

$$F(p) = \int_0^\infty \exp(-pt)dt = \left[-\frac{1}{p}\exp(-pt)\right]_0^\infty = \frac{1}{p}, \quad p > 0.$$

(ii) The Laplace transform of the function $f(t) = \exp(at)$ is given by

$$F(p) = \int_0^\infty \exp[(a-p)t]dt = \left[\frac{1}{a-p}\exp[(a-p)t]\right]_0^\infty = \frac{1}{p-a}, \quad p > a.$$

(iii) Consider the function $f(t) = t^n$ where n is an integer, then

$$F(p) = I_n = \int_0^\infty t^n \exp(-pt)dt$$

$$= -\frac{1}{p}\left[t^n \exp(-pt)\right]_0^\infty + \frac{n}{p}\int_0^\infty t^{n-1}\exp(-pt)dt$$

$$= \frac{n}{p}I_{n-1} = \frac{n}{p}\frac{(n-1)}{p}I_{n-2} = \frac{n}{p}\frac{(n-1)}{p}\frac{(n-2)}{p}I_{n-3} = ...$$

provided $p > 0$. Hence

$$F(p) = \frac{n!}{p^{n+1}}.$$

Note that this result provides us with a integral representation for a factorial, i.e. with $p = 1$,

$$n! = \int_0^\infty t^n \exp(-t)dt.$$

From this equation, we can quite naturally extend the definition of a factorial of an integer n to that for a non-integer q. This provides us with the definition of the Gamma function given by

$$\Gamma(q) = \int_0^\infty t^{q-1}\exp(-t)dt, \quad q - 1 > 0$$

so that

$$\Gamma(q+1) = q!$$

Thus, note that for $q > 0$

$$\hat{L}t^q = \frac{\Gamma(1+q)}{p^{1+q}}.$$

(iv) The Laplace transform of $t^n f(t)$ can be evaluated thus: Since

$$F(p) = \int_0^\infty f(t)\exp(-pt)dt,$$

$$\frac{dF}{dp} = \frac{d}{dp}\int_0^\infty f(t)\exp(-pt)dt = -\int_0^\infty f(t)t\exp(-pt)dt$$

and

$$\frac{d^2F}{dp^2} = -\frac{d}{dp}\int_0^\infty f(t)t\exp(-pt)dt = \int_0^\infty f(t)t^2\exp(-pt)dt.$$

By induction,

$$\frac{d^nF}{dp^n} = (-1)^n\hat{L}[t^n f(t)].$$

Hence,

$$\hat{L}[t^n f(t)] = (-1)^n\frac{d^nF(p)}{dp^n}.$$

Similarly, the Laplace transform of $t^{-n}f(t)$ is given by

$$\hat{L}[t^{-n}f(t)] = (-1)^n\int ...\int_n F(p)d^np.$$

(v) The Laplace transform of a periodic function can be evaluated if we consider the function $f(t)$ to have a period a say. Then

$$F(p) = \int_0^\infty f(t)\exp(-pt)dt = \int_0^a f(t)\exp(-pt)dt + \int_a^{2a} f(t)\exp(-pt)dt + ...$$

$$= \int_0^a f(t)\exp(-pt)dt + \int_0^a f(t)\exp[-p(t+a)]dt + ...$$

$$= [1 + \exp(-pa) + \exp(-2pa) + ...]\int_0^a f(t)\exp(-pt)dt.$$

Now, the series above is a geometric series and if

$$s = 1 + \exp(-pa) + \exp(-2pa) + ... + \exp(-npa)$$

then

$$s\exp(-pa) = \exp(-pa) + \exp(-2pa) + \exp(-3pa) + ... + \exp[-(n+1)pa].$$

Subtracting these equations gives

$$s = \frac{1-\exp[-(n+1)pa]}{1-\exp(-pa)} = \frac{1}{1-\exp(-pa)} \quad \text{as} \quad n \to \infty.$$

Hence, the Laplace transform of a function with period a can be written as

$$\frac{1}{1+\exp(-pa)} \int_0^a f(t)\exp(-pt)dt.$$

It is important to note that not all Laplace transforms can be evaluated, i.e. in some cases the integral does not converge. For example, for the function $f(t) = 1/t$,

$$F(p) = \int_0^\infty \frac{1}{t}\exp(-pt)dt$$

and divergence occurs at the lower limit. Therefore the integral does not exist and there is no Laplace transform of $1/t$ whereas the Fourier transform of this function does exist (i.e. $t^{-1} \Longleftrightarrow = -i\pi\text{sgn}(\omega)$ - see Chapter 4). Another example includes the function $f(t) = \exp(t^2)$ whose Laplace transform is

$$F(p) = \int_0^\infty \exp(t^2)\exp(-pt)dt.$$

In this case, the upper limit takes the integrand to infinity; the integral does not exist and there is no Laplace transform of $\exp(t^2)$.

Laplace Transform of a Derivative

$$\hat{L}\left(\frac{df}{dt}\right) = \int_0^\infty \frac{d}{dt}\left[f(t)\exp(-pt)\right]dt - \int_0^\infty f\frac{d}{dt}\exp(-pt)dt$$

$$= \left[f(t)\exp(-pt)\right]_0^\infty + pF(p) = pF(p) - f(0)$$

and

$$\hat{L}\left(\frac{d^2f}{dt^2}\right) = \int_0^\infty \frac{d}{dt}\left(\frac{df}{dt}\exp(-pt)\right)dt - \int_0^\infty \frac{df}{dt}\frac{d}{dt}\exp(-pt)dt$$

$$= \left[\frac{df}{dt}\exp(-pt)\right]_0^\infty + p\int_0^\infty \frac{df}{dt}\exp(-pt)dt$$

$$= p^2F(p) - pf(0) - \left[\frac{df}{dt}\right]_{t=0}.$$

Thus, by induction

$$\hat{L}\left[\frac{d^n f}{dt^n}\right] = p^n F(p) - p^{n-1} f(0) - p^{n-2}\left[\frac{df}{dt}\right]_{t=0} ... - \left[\frac{d^{n-1} f}{dt^{n-1}}\right]_{t=0}.$$

Compare this result with the equivalent result for the Fourier transform, i.e.

$$\hat{F}_1\left[\frac{d^n f}{dt^n}\right] = (i\omega)^n F(\omega).$$

For the Laplace transform, the values of the function and its derivatives must be known at $t = 0$ which is not the case with the Fourier transform. This highlights the use of the Laplace transform for solving initial value problems compared to the Fourier transform which is non-causal. However, since few signals are provided together with an initial condition, the Fourier transform dominates the field compared to the Laplace transform.

Laplace Transform of an Integral

$$\hat{L}\left(\int^t f(\tau)d\tau\right) = \int_0^\infty \exp(-pt)\int^t f(\tau)d\tau dt.$$

Since

$$-\frac{1}{p}\frac{d}{dt}\exp(-pt) = \exp(-pt)$$

the integral above can be written as

$$-\frac{1}{p}I$$

where

$$I = \int_0^\infty \frac{d}{dt}[\exp(-pt)]\int^t f(\tau)d\tau dt$$

$$= \int_0^\infty \frac{d}{dt}\left(\exp(-pt)\int^t f(\tau)d\tau\right)dt - \int_0^\infty \exp(-pt)\left(\frac{d}{dt}\int^t f(\tau)d\tau\right)dt$$

$$= \left[\exp(-pt)\int^t f(\tau)d\tau\right]_0^\infty - \int_0^\infty \exp(-pt)f(t)dt$$

$$= -\int^t f(\tau)d\tau \mid_{t=0} -F(p).$$

Hence,

$$\hat{L}\left(\int^t f(\tau)d\tau\right) = \frac{1}{p}F(p) + \frac{1}{p}f^{-1}(0)$$

where

$$f^{-1}(0) \equiv \int^{t} f(\tau)d\tau \mid_{t=0} .$$

Important Theorems

- Linearity

$$\hat{L}[af(t) + bg(t)] = aF(p) + bG(p)$$

- Scaling

$$\hat{L}[f(at)] = \frac{1}{a}F\left(\frac{p}{a}\right)$$

- Convolution

$$\hat{L}\left[\int_0^t f(\tau)g(t-\tau)d\tau\right] = F(p)G(p)$$

- Shift Theorem

$$\hat{L}[f(t-a)] = \exp(-ap)F(p)$$

- Inverse Shift Theorem

$$\hat{L}[\exp(at)f(t)] = F(p-a)$$

The Inverse Laplace Transform

Given that

$$F(p) = \int_0^{\infty} f(t)\exp(-pt)dt$$

with $f(t) = 0, \quad t < 0$ and $p = i\omega$ we can view the Laplace transform as a Fourier transform of the form

$$F(i\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)dt$$

whose inverse is

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(i\omega)\exp(i\omega t)d\omega.$$

Now, with $\omega = p/i$, this integral can be written in the form

$$f(t) = \hat{L}^{-1}F(p) \equiv \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} F(p)\exp(px)dp$$

where $\hat{L}^{-1}$ is the inverse Laplace operator and c denotes an arbitrarily small constant which takes the path of the integral away from the imaginary axis. This result is often called the 'Bromwich integral'. Unlike the Fourier transform, the Laplace transform is not symmetrical, i.e. the forward and inverse Laplace transforms are not of the same form. For analytical problems, the Laplace transforms are inverted using tables generated by computing the forward Laplace transform for a range of different functions. Direct evaluation of the inverse Laplace transform requires contour integration of the type discussed in Chapter 1.

Causal Convolution

In cases where f and g are zero for $t < 0$ we can define the causal convolution integral (Wiener-Hopf integral)

$$f \otimes g = \int_0^\infty f(t)g(t'-t)dt.$$

In this case, the convolution theorem is

$$f \otimes g \Longleftrightarrow FG$$

where F and G are the Laplace transforms of f and g respectively (and not the Fourier transforms), i.e.

$$F(p) = \int_0^\infty f(t)\exp(-pt)dt,$$

$$G(p) = \int_0^\infty g(t)\exp(-pt)dt.$$

This is the convolution theorem for Laplace transforms. It can be proved using a similar approach to that adopted to prove the convolution theorem for the Fourier transform. Thus, consider f and g in terms of their respective inverse Laplace transforms

$$f(t) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} F(p)\exp(pt)dp,$$

$$g(t) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} G(p)\exp(pt)dp.$$

Then

$$\int_0^\infty f(t)g(t'-t)dt = \int_0^\infty dt \frac{1}{(2\pi i)^2} \int_{c-i\infty}^{c+i\infty} F(p)\exp(pt)dp \int_{c-i\infty}^{c+i\infty} G(p')\exp[p'(t'-t)]dp'$$

$$= \frac{1}{(2\pi i)^2} \int_{c-i\infty}^{c+i\infty} dp' G(p')\exp(p't') \int_{c-i\infty}^{c+i\infty} \frac{F(p)}{p-p'}dp$$

$$= \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} G(p')F(p')\exp(p'x')dp'$$

since (Cauchy's integral formula)

$$\oint_C \frac{F(p)}{p-p'}dp = 2\pi i F(p').$$

Hence, for Laplace transforms

$$f \otimes g \Longleftrightarrow FG.$$

The convolution theorem (for both the Fourier and Laplace transform) provides a method for solving integral equation which frequently occurs in signal analysis. In the case of non-causal equations we use the Fourier transform. For example, suppose we are required to solve the equation

$$u(t) = f(t) - \int_{-\infty}^{\infty} u(\tau)g(t-\tau)d\tau$$

for u. Since $t \in (-\infty, \infty)$, we take Fourier transforms and use the convolution theorem to obtain

$$U(\omega) = F(\omega) - U(\omega)G(\omega).$$

Hence,

$$U(\omega) = \frac{F(\omega)}{1+G(\omega)}$$

and we can write a solution of the type

$$u(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{F(\omega)}{1+G(\omega)} \exp(i\omega t)d\omega, \quad G(\omega) \neq -1.$$

Now suppose we are required to solve the (causal) equation

$$u(t) = f(t) - \int_0^\infty u(\tau)g(t-\tau)d\tau$$

for u. Since $t \in [0, \infty)$, we use Laplace transforms and the convolution theorem to obtain

$$U(p) = F(p) - U(p)G(p)$$

so that

$$U(p) = \frac{F(p)}{1 + G(p)}$$

and on inversion

$$u(t) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} \frac{F(p)}{1 + G(p)} \exp(pt) dp.$$

To complete this section, the following table gives some examples of the Laplace transform.

Function	Laplace Transform
$f(t) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} F(p) \exp(pt) dp$	$F(p) = \hat{L}[f(t)] \equiv \int_0^\infty f(t) \exp(-pt) dt$
1	$\frac{1}{p}$
$t^n, \quad n > -1$	$n! p^{-(n+1)}$
$\exp(at)$	$\frac{1}{p-a}$
$\sin(\omega t)$	$\frac{\omega}{p^2+\omega^2}$
$\cos(\omega t)$	$\frac{p}{p^2+\omega^2}$
$\sinh(at) = \frac{1}{2}(e^{at} - e^{-at})$	$\frac{1}{2}\left(\frac{1}{p-a} - \frac{1}{p+a}\right) = \frac{a}{p^2-a^2}$
$\cosh(at) = \frac{1}{2}(e^{at} + e^{-at})$	$\frac{1}{2}\left(\frac{1}{p-a} - \frac{1}{p+a}\right) = \frac{p}{p^2-a^2}$
$\frac{df}{dt}$	$pF(p) - f(0)$
$\frac{d^2 f}{dt^2}$	$p^2 F(p) - pf(0) - f'(0)$
$-tf(t)$	$\frac{dF}{dp}$
$\exp(at) f(t)$	$F(p - a)$
$U(t - b) f(t - b)$	$\exp(-bp) F(p) \quad b > 0$
$\int_0^t f(t - \tau) g(\tau) d\tau$	$F(p) G(p)$
$\delta(t - b)$	$\exp(-bp), \quad b \geq 0$

Table 5.1: Table of example Laplace transforms

5.2 The Sine and Cosine Transforms

5.2.1 The Sine Transform

Let $f(t) = 0, \quad t < 0$ and consider the imaginary part of the Fourier transform. We can then consider the transform

$$F(\omega) = \hat{S}[f(t)] \equiv \int_0^\infty f(t) \sin(\omega t) dt$$

and

$$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \sin(\omega t) d\omega.$$

A table giving some examples of the sine transform is given below.

Function	Sine Transform
$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \sin(\omega t) d\omega$	$\hat{S}[f(t)] = F(\omega) = \int_0^\infty f(t) \sin(\omega t) dt$
$\frac{df}{dt}$	$-\omega \hat{C}[f(t)]$
$\frac{d^2 f}{dt^2}$	$\omega f(0) - \omega^2 F(\omega)$
$\exp(-\epsilon t)$	$\frac{\omega}{\epsilon^2 + \omega^2}$
1	$\frac{1}{\omega}$

Table 5.2: Some examples of the sine transforms

Note that, like the one-sided Laplace transform, the sine transform is causal.

5.2.2 The Cosine Transform

Let $f(t) = 0, \quad t < 0$ and consider the real part of the Fourier transform. Then

$$F(\omega) = \hat{C}[f(t)] \equiv \int_0^\infty f(t) \cos(\omega t) dt$$

and

$$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \cos(\omega t) d\omega.$$

A table giving some examples of the cosine transform is given below.

Function	Cosine Transform
$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \cos(\omega t) d\omega$	$\hat{C}[f(t)] = F(\omega) = \int_0^\infty f(t) \cos(\omega t) dt$
$\frac{df}{dt}$	$-f(0) + \omega \hat{S}[f(t)]$
$\frac{d^2 f}{dt^2}$	$-f'(0) - \omega^2 F(\omega)$
$\exp(-\epsilon t)$	$\frac{\epsilon}{\epsilon^2 + \omega^2}$
$\exp(-\alpha t^2)$	$\sqrt{\frac{\pi}{4\alpha}} \exp(-\omega^2/4\alpha)$

Table 5.3: Some examples of the cosine transforms

As with the Laplace and sine transforms, the cosine transform is causal and can be used to solve initial value problems when $f(0)$ and $f'(0)$ are known. The convolution theorem for the cosine transform is

$$\int_0^\infty f(\tau)[g(t+\tau)+g(t-\tau)]d\tau \Longleftrightarrow F(\omega)G(\omega).$$

For the sine transform, the convolution theorem is

$$\int_0^\infty f(\tau)[g(t+\tau)-g(t-\tau)]d\tau \Longleftrightarrow \hat{S}[f(t)]\hat{C}[g(t)]$$

5.3 The Walsh Transform

The origins of the Fourier transform (as discussed in Chapter 3) are based on the application of the function $\exp(in\pi t/T)$ to describe a signal with a period T given by

$$f(t) = \sum_n c_n \exp(i\pi nt/T)$$

where

$$\frac{1}{T}\int_{-T/2}^{T/2} \exp(i2\pi nt/T)\exp(-i2\pi mt/T)dt = \begin{cases} 1, & n = m; \\ 0, & n \neq m. \end{cases}$$

the functions $\exp(in\pi t/T)$, forming an orthogonal system. However, the complex exponential (together with the sin and cosine) is not the only function to form such a system; many such functions are available. Instead of considering cosinusoidal wave functions oscillating at different frequencies, suppose we consider a set of rectangular waveforms taking on just two amplitude values of +1 and -1. These wave functions are another example of an orthogonal set of functions. Known as Walsh functions, they are defined over a limited time interval T (the time-base), which requires to be known if quantitative values are to be assigned to the function. Like the sine-cosine functions, two arguments are required for complete definition, a time period t and an ordering number n which is related to the frequency, the function being written in the form $\mathrm{WAL}(n,t)$ with the property:

$$\frac{1}{T}\int_0^T \mathrm{WAL}(m,t)\mathrm{WAL}(n,t)dt = \begin{cases} 1, & n = m; \\ 0, & n \neq m. \end{cases}$$

With this property, we can define the Wash transform pair as

$$f(t) = \frac{1}{T}\sum_{n=0}^{N-1} F_n \mathrm{WAL}(n,t),$$

$$F_n = \int_0^T f(t)\mathrm{WAL}(n,t)dt.$$

Discretising, we then obtain the discrete Walsh transform given by

$$f_m = \frac{1}{N}\sum_{n=0}^{N-1} F_n \mathrm{WAL}(n,m),$$

$$F_n = \sum_{m=0}^{N-1} f_m \mathrm{WAL}(n,m).$$

The Walsh transform has many of the properties of the Fourier transform including the convolution theorem for example and the Walsh coefficients F_n can be analysed as a Walsh spectrum to which filters can be applied. Thus, Walsh function series can be applied to many of those areas where sinusoidal techniques dominate. However, the applications for which the Walsh transform are ideal are those in which the binary form (i.e. -1, +1 or 0,1) of the function can be used to match binary logic and binary computer algorithmic techniques. Additionally, there are certain characteristics of the Walsh transform that have no equivalence to the Fourier transform which permit new approaches to the design of hardware/software. The main focus for the applications of the Walsh transform are in digital communications and coding. These include multiplexing (the simultaneous transmission of many independent signals over a common communications channel), binary group coding and digital image transmission and processing. Compared with the Fourier transform, which requires a broad spectrum to describe high frequency features in a signal, the bandwidth of the Walsh transform can be relatively low. This is because relatively few terms in the Walsh series representation of an 'on-off' type signal (which requires many Fourier coefficients to be represented accurately) are required in comparison with a Fourier series; the Fourier transform being based on smooth orthogonal basis functions and the Walsh transform being based on discontinuous or 'on-off' type orthogonal basis functions.

Walsh functions are, like cosinusoids, just one of a number of basis functions that form an orthogonal series $B_n(t)$ and so in general, we can consider a signal of the form

$$f(t) = \sum_{n=0}^{N-1} c_n B_n(t).$$

Suppose the series $B_n(t), \quad n = 0, 1, 2, ...$ is orthogonal over the interval $0 \le t \le T$, i.e.

$$\int_0^T B_n(t)B_m(t)dt = \begin{cases} T, & n = m; \\ 0, & n \neq m. \end{cases}$$

where n and m have integer values. Now, ideally we want $N \to \infty$ to represent $f(t)$ exactly. However, in practice, only a finite number of terms N is possible for the evaluation of $f(t)$ (leading to truncation error) and thus, the coefficients c_n need to be chosen to provide the best approximation to $f(t)$. This can be achieved by solving for c_n such that the mean-square error defined by

$$e(c_n) = \int_0^T \left[f(t) - \sum_{n=0}^{N-1} c_n B_n(t) \right]^2 dt$$

is a minimum. Thus occurs when

$$\frac{d}{dc_m}e(c_n) = \int_0^T 2\left[f(t) - \sum_{n=0}^{N-1} c_n B_n(t)\right] B_m(t)dt = 0$$

or when

$$\int_0^T f(t)B_m(t)dt = \sum_{n=0}^{N-1} c_n \int_0^T B_n(t)B_m(t)dt = \sum_{n=0}^{N-1} c_n \int_0^T B_n(t)B_m(t)dt = Tc_m$$

and hence,

$$c_m = \frac{1}{T}\int_0^T f(t)B_m(t)dt.$$

Clearly, the ability to compute this set of coefficients depends exclusively on the fact that $B_n(t)$ are orthogonal and given this condition, we see that the Fourier transform and the Walsh transform are essentially based on the application of certain orthogonal basis functions which are part of a much wider class of functions in general.

5.4 The Cepstral Transform

When the spectrum of a signal is computed via the Fourier transform, a common characteristic is for the high frequency components to have amplitudes that are relatively small compared to those at the low frequency end of the spectrum especially when the signal is dominatad by a large DC component. Typically, this leads to a plot of the amplitude or power spectrum being dominated by the low frequency characteristics of the signal. A simple method of enhancing the high frequency spectrum is to apply a logarithmic transform and instead of plotting the amplitude spectrum $\mid F(\omega) \mid$, we plot and analyse $\log \mid F(\omega) \mid$ given that $F(\omega) > 0 \ \ \forall\omega$ or $\log(1+ \mid F \mid)$ given that $F(\omega) \geq 0 \ \ \forall\omega$. When the logarithm is to base 10, the spectral scale is measured in decibels. This simple transform for enhancing the high frequency spectral characteristics leads directly to the idea of computing its inverse Fourier transform. Thus, given that $F(\omega)$ is the Fourier transform of $f(t)$, instead of computing

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)d\omega$$

we consider computing

$$\hat{f}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \log \mid F(\omega) \mid \exp(i\omega t)d\omega.$$

The function $\hat{f}(t)$ is referred to as the real cepstrum (or just the cepstrum), the complex cepstrum being given by

$$\hat{f}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \log[F(\omega)]\exp(i\omega t)d\omega.$$

Note that since $F(\omega)$ is a complex function which can be written as $A_F(\omega)\exp[i\theta_F(\omega)]$ where A_F and θ_F are the amplitude spectrum and the phase spectrum respectively,

$$\log[F(\omega)] = \log A_F(\omega) + i\theta_F(\omega).$$

Also, note that the signal $f(t)$ can be reconstructed from the complex cepstrum but not from the real cepstrum because the phase spectrum is missing. In the analysis of θ_F, as with all phase plots, it is usually necessary to phase unwrap; a technique that is discussed further in the following section.

The cepstral transform and cepstrum based filtering has value in the analysis of homomorphic systems. The idea is to transform a multiplicative model for a process into an additive one with the use of a logarithmic transform. Thus, if a spectrum is given by $F(\omega) = F_1(\omega)F_2(\omega)$ where it can be assumed F_1 and F_2 have distinct characteristics in the low and high frequency range of the spectrum respectively, then by computing

$$\log F(\omega) = \log F_1(\omega) + \log F_2(\omega)$$

and applying an appropriate lowpass or highpass filter to $\log F$, it is possible to recover (approximately) F_1 or F_2 respectively. This approach has significant value in speech processing for example, in which a speech signal can often be well modelled in terms of the convolution of a low frequency signal (generated by the vibration of the vocal chords which is a low frequency noise source) and a high frequency signal (generated by the movement of the mouth, tong and lips to form an utterance). Cepstral speech analysis can be used to compute templates of speech signals that are used in speech recognition systems for example. The templates produced are usually more sensitive and robust to those generated from an analysis of the speech signal and/or its spectrum directly.

In general, cepstral analysis is a logarithmic based enhancement method used to diagnose the presence of certain features hidden in a signal. For example, situations arise which cause the spectrum to have periodic components. Using the cepstrum, it is often possible to detect these components and relate them back to features of a signal which are not apparent in the time domain. A number of phenomena can cause oscillatory patters in the power spectrum such as:

- non-linearities in the signal generating mechanism where the periodicities are associated with harmonic components of a fundamental frequency;
- delayed replicas of a signal component added to the original time series causing a periodic oscillation in the power spectrum.

Since ceptral analysis can be used to detect peaks in the power spectrum, attention needs to focus in obtaining accurate power spectral estimates that are free from spectral leakage (windowing errors - see Section 13.3) otherwise the spectral components to be analysed can be corrupted. The quality of a cepstral analysis is influenced by a number of factors that include:

- differentiation or high pass filtering of the power spectrum or source data which emphasises echo-type phenomena and can lead to more clearly defined ceptral peaks;

- the use of different windows on a signal which can enhance the echo discriminating aspects of a cepstrum in certain cases.

The cepstrum was originally designed for pulse-echo-delay systems and was applied to seismic signal processing, an area that is typified by data consisting of a primary wave form together with filtered and delayed versions (or echos) of the primary signal, modelled in terms of low frequency acoustic waves scattering from a layered media (see Section 4.9). These echos are typically hidden in noise and, as such, are often indistinguishable from the original time trace. Ceptral analysis can often detect such echos and their relative positions in the seismic signal. However, cepstral analysis is, in general, a rather subjective signal analysis tool and can yield useful results in one application while providing inexplicably poor results in another. It is most likely to be helpful in situations involving a basic time function that has been convolved with a sequence of impulse functions.

5.5 The Hilbert Transform

The Hilbert transform is named after David Hilbert, who's career spanned many years of both the nineteenth and twentieth centuries and covered many aspects of mathematical analysis, logic and aspects of mathematical physics. In signal analysis, this transform provides a way of analysing a signal in the complex plane (i.e. using an Argand diagram). Given a real signal, the Hilbert transform converts the signal into its quadrature or imaginary component. With both real and imaginary parts available, complex plane analysis can be undertaken. This transformation also allows the signal attributes to be computed such as the amplitude envelope and the phase and it is closely associated with methods of modulation and demodulation which are also addressed in this section. Taken in combination, a real signal together with its Hilbert transform is known as the analytic signal.

The Hilbert transform $q(t)$ of a real function $f(t)$ can be defined by

$$q(t) = \hat{H}f(t) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(\tau)}{t-\tau} d\tau$$

where $\hat{H}$ is the Hilbert transform operator. Note that the more usual definition of the Hilbert transform (as quoted by most authors) is

$$q(t) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(\tau)}{\tau - t} d\tau$$

and the definition used here is sometimes referred to as the Stieltjes-Riemann integral. Also, note that in both cases, it is common to prefix the integral with P to indicate that the integral refers to the principal part in order to take account of the singularity at $\tau = t$. Nevertheless, this transform is a just a convolution, a convolution of $f(t)$ with $1/\pi t$. Hence, we may write the Hilbert transform in the following way:

$$q(t) = \frac{1}{\pi t} \otimes f(t).$$

Using the convolution theorem and noting (as derived in Chapter 4) that

$$\hat{F}_1\left(\frac{1}{\pi t}\right) = -i\,\mathrm{sgn}(\omega),$$

the Fourier transform of q can be written as

$$Q(\omega) = -i\,\mathrm{sgn}(\omega)F(\omega).$$

Hence,

$$q(t) = \hat{F}_1^{-1}[-i\,\mathrm{sgn}(\omega)F(\omega)].$$

This result immediately provides the following method for computing the Hilbert transform of a function: (i) Take the Fourier transform of the function; (ii) multiply the result by $-i\,\mathrm{sgn}(\omega)$; (iii) compute the inverse Fourier transform of the result. Thus, for example, it is easy to show that the Hilbert transform of $\cos(\omega t)$ is $\sin(\omega t)$ and the Hilbert transform of $\sin(\omega t)$ is $-\cos(\omega t)$. In this sense, the Hilbert transform can be considered to change the phase of a cosinusoidal wave by 90^o. A table of example Hilbert transforms is given below.

Function	Hilbert Transform
$f(t)$	$q(t) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{f(\tau)}{\tau - t}d\tau$
$\delta(t)$	$\frac{1}{\pi t}$
$U(t-t_0)$	$\frac{1}{\pi}\ln\mid t - t_0 \mid$
$U(t-b) - U(t-a)$	$-\frac{1}{\pi}\ln\left\lvert\frac{t-a}{t-b}\right\rvert$
$\cos(\omega t), \quad \omega > 0$	$\sin(\omega t)$
$\sin(\omega t), \quad \omega > 0$	$-\cos(\omega t)$
$\frac{\sin(\omega t)}{\omega t}, \quad \omega > 0$	$\frac{\sin^2(\omega t/2)}{\omega t/2}$
$\frac{t}{a^2+t^2}, \quad \mathrm{Re}\,a > 0$	$-\frac{a}{a^2+t^2}$
$\frac{a}{a^2+t^2}, \quad \mathrm{Re}\,a > 0$	$\frac{t}{a^2+t^2}$

Table 5.4: Table of example Hilbert transforms

5.5.1 Modulation and Demodulation

If $f(t)$ is a bandlimited function with spectrum $F(\omega), \quad \mid \omega \mid \leq \Omega$, then multiplying this function by $\cos(\omega_0 t)$ say, shifts or modulates the location of the centre of $F(\omega)$ from $\omega = 0$ to $\omega = \pm\omega_0$. The frequency ω_0 is commonly referred to as the carrier frequency. This processes is known as modulation and can be described mathematically by

$$f(t)\cos(\omega_0 t) \Longleftrightarrow F(\omega) \otimes \pi[\delta(\omega - \omega_0) + \delta(\omega + \omega_0)]$$

or, using the sampling property of the delta function

$$f(t)\cos(\omega_0 t) \Longleftrightarrow \pi[F(\omega - \omega_0) + F(\omega + \omega_0)].$$

A modulated spectrum of this kind is known as a sideband spectrum. If a spectrum has its centre at $\omega = 0$, then it is known as a baseband spectrum. Shifting a sideband spectrum back to baseband is called demodulation. Demodulating a modulated

spectrum (modulated to $\pm\omega_0$) is accomplished by multiplying by another cosine function oscillating at the same (carrier) frequency ω_0. This process can be described mathematically as follows:

$$f(t)\cos(\omega_0 t)\cos(\omega_0 t) \Longleftrightarrow \pi[F(\omega-\omega_0)+F(\omega+\omega_0)] \otimes \pi[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)]$$

or using the sampling property of the delta function

$$f(t)\cos(\omega_0 t)\cos(\omega_0 t) \Longleftrightarrow \pi^2[2F(\omega)+F(\omega-2\omega_0)+F(\omega+2\omega_0)].$$

By neglecting the part of the spectrum which is modulated to $\pm 2\omega_0$ by this process, the baseband spectrum $F(\omega)$ can be extracted. In practice this can be achieved by applying a lowpass filter.

5.5.2 Quadrature Detection

The Hilbert transform of a signal is often referred to as the quadrature signal which is why it is usually denoted by the letter q. Electronic systems which perform Hilbert transforms are also known as quadrature filters. These filters are usually employed in systems where the signal is a continuous wave or a narrowband signal (i.e. a signal whose bandwidth is a small percentage of the dominant carrier frequency). They operate on the basis that if $y(t)$ is a real valued and bandlimited signal with a spectrum $Y(\omega), \mid \omega \mid \leq \Omega$ and

$$f(t) = y(t)\cos(\omega_0 t)$$

where $\omega_0 > \Omega$, then the quadrature signal can be obtained by simply multiplying $y(t)$ by $\sin(\omega_0 t)$, i.e.

$$q(t) = y(t)\sin(\omega_0 t).$$

This result can be derived by analysing the Fourier transform of $f(t)$ which is given by (using the convolution theorem)

$$F(\omega) = Y(\omega) \otimes \pi[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)] = \pi[Y(\omega-\omega_0)+Y(\omega+\omega_0)].$$

Now, the spectrum of the Hilbert transform of F is obtained by multiplying F by $-i\,\mathrm{sgn}(\omega)$. Hence, provided $\omega_0 > \Omega$,

$$Q(\omega) = -i\pi\,\mathrm{sgn}(\omega)[Y(\omega-\omega_0)+Y(\omega+\omega_0)] = i\pi[Y(\omega+\omega_0)-Y(\omega-\omega_0)]$$

$$= Y(\omega) \otimes i\pi[\delta(\omega+\omega_0)-\delta(\omega-\omega_0)].$$

Using the product theorem, in real space, the last equation is

$$q(t) = y(t)\sin(\omega_0 t).$$

The quadrature filter is mostly used in systems where frequency demodulation is required to extract a baseband signal from a narrowband signal.

5.5.3 The Analytic Signal

In signal analysis (where the independent variable is usually time), a real valued signal can be represented in terms of the so called analytic signal. The analytic signal is important because it is from this signal that the amplitude, phase and frequency modulations of the original real valued signal can be determined.

If $f(t)$ is a real valued signal with spectrum $F(\omega)$, then $f(t)$ can be computed from $F(\omega)$ via the inverse Fourier transform

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

Note that as usual, this involves applying a two-sided integral where integration is undertaken over ω from $-\infty$ to ∞. The analytic signal is obtained by integrating only over the positive half of the spectrum which contains the 'physically significant' frequencies, i.e. integrating over ω from 0 to ∞. In a sense, this is the key of the answer to the question: what is the meaning of a negative frequency? If s is used to denote the analytic signal of f, then by definition

$$s(t) = \frac{1}{\pi} \int_{0}^{\infty} F(\omega) \exp(i\omega t) d\omega.$$

From this result, it is possible to obtain an expression for s in terms of f which is done by transforming s into Fourier space and analysing the spectral properties of the analytic signal. This leads to the following theorem:

Theorem The analytic signal is given by

$$s(x) = f(x) + iq(x)$$

where q is the Hilbert transform of f.

Proof In Fourier space, the analytic signal can be written as

$$S(\omega) = 2U(\omega)F(\omega)$$

where S and F are the Fourier transforms of s and f respectively and $U(\omega)$ is the unit step function given by

$$U(\omega) = \begin{cases} 1, & \omega \geq 0; \\ 0, & \omega < 0. \end{cases}$$

We now employ a simple but useful analytical trick by writing the step function in the form

$$U(\omega) = \frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\omega)$$

where

$$\operatorname{sgn}(\omega) = \begin{cases} 1, & \omega > 0; \\ -1, & \omega < 0. \end{cases}$$

The inverse Fourier transform of this function can then be written as

$$u(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{1}{2}\exp(i\omega t)d\omega + \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{1}{2}\operatorname{sgn}(\omega)\exp(i\omega t)d\omega$$

$$= \frac{1}{2}\delta(t) + \frac{i}{2\pi t}$$

where we have used the results

$$\delta(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(i\omega t)d\omega$$

and

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\operatorname{sgn}(\omega)\exp(i\omega t)d\omega = \frac{i}{\pi t}.$$

Now, since

$$2U(\omega)F(\omega) \Longleftrightarrow 2u(t)\otimes f(t)$$

we have

$$s(t) = 2u(t)\otimes f(t) = f(t)\otimes\left(\delta(t) + \frac{i}{\pi t}\right) = f(t) + \frac{i}{\pi t}\otimes f(t)$$

or

$$s(t) = f(t) + iq(t)$$

where q is the Hilbert transform of f, i.e.

$$q(t) = \frac{1}{\pi t}\otimes f(t)$$

which completes the proof.

From the last result it is clear that the analytic signal associated with a real valued function f can be obtained by computing its Hilbert transform to provide the quadrature component. This process is called quadrature detection. The analytic signal is a complex function and therefore contains both amplitude and phase information. The important feature about the analytic signal is that its spectrum (by definition) is zero for all values of ω less than zero. This type of spectrum is known as a single sideband spectrum because the negative half of the spectrum is zero. An analytic signal is therefore a single sideband signal.

This result provides another way of computing the Hilbert transform of a function $f(t)$:

(i) Compute the Fourier transform of $f(t)$;

(ii) set the components of the complex spectrum $F(\omega)$ in the negative half space to zero;

(iii) compute the inverse Fourier transform which will have real and imaginary parts $f(t)$ and $q(t)$ respectively.

5.5.4 Attributes of the Analytic Signal

As with any other complex function, the behaviour of the analytic signal can be analysed using an Argand diagram and may be written in the form

$$s(t) = A(t)\exp[i\theta(t)]$$

where

$$A(t) = \sqrt{f^2(t) + q^2(t)}$$

and

$$\theta(t) = \tan^{-1}\left(\frac{q(t)}{f(t)}\right).$$

Here, the real part of $s(t)$ is $f(t) = A(t)\cos\theta(t)$, as illustrated in Figure 5.1, and the imaginary component (computed by taking the Hilbert transform of f) is given by $q(t) = A(t)\sin\theta(t)$.

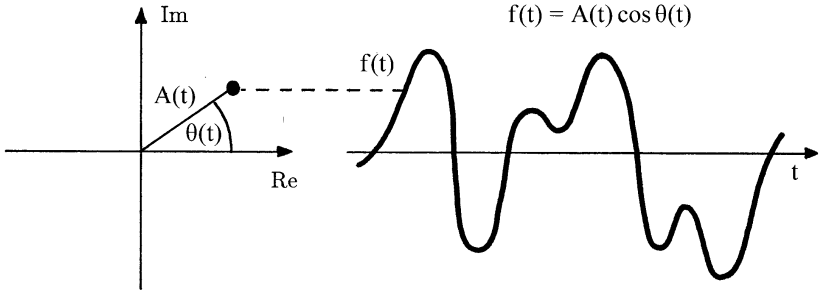


Figure 5.1: Complex plane representation of a real signal.

The function A describes the dynamical behaviour of the amplitude modulations of f. For this reason, it is sometimes referred to as the amplitude envelope. The function θ measures the phase of the signal at an instant in time and is therefore known as the instantaneous phase. Note that because the arctangent function is periodic, this function is multi-valued. Hence, strictly speaking, the analytic function should be written as

$$s(t) = A(t)\exp[i(\theta(t) + 2\pi n)], \quad n = 0, \pm 1, \pm 2, ...$$

If we confine the value of the phase to a fixed period (i.e. we compute the phase using only one particular value of n), then it is referred to as the wrapped phase. In this case, there is only one unique value of the phase within a fixed period. However, any other interval of length 2π can be chosen. Any particular choice, decided upon *a priori*, is called the principal range and the value of the phase within this range, is called the principal value.

5.5.5 Phase Unwrapping

The wrapped phase is not an ideal way of analysing the phase of the analytic signal. The problem comes from the fact that the arctangent based definition of the phase is inherently multi-valued. Overcoming this problem to produce an unwrapped phase function requires a different approach. Thus, the problem is to find another definition of the phase function that does not rely on the conventional arctangent definition. What follows is an intriguing method of doing this and relies on taking the logarithm of the analytic signal. Since

$$s(t) = A(t)\exp[i(\theta(t) + 2\pi n)], \quad n = 0, \pm 1, \pm 2, ...,$$

taking the natural logarithm of both sides yields the result

$$\ln s(t) = \ln A(t) + i(\theta(t) + 2\pi n)$$

from which it follows that

$$\theta(t) + 2\pi n = \mathrm{Im}[\ln s(t)].$$

This result of course relies on the condition that $A(t) > 0 \forall t$. Now, another important property of the analytic signal is its instantaneous frequency. This parameter (denoted by ψ say) measures the rate of change of phase, i.e.

$$\psi(t) = \frac{d}{dt}\theta(t).$$

The effect of considering this function is to eliminate the constant $2\pi n$ from the equation for θ, i.e.

$$\psi(t) = \mathrm{Im}\left[\frac{d}{dt}\ln s(t)\right] = \mathrm{Im}\left[\frac{1}{s(t)}\frac{d}{dt}s(t)\right]$$

or alternatively (since $s = f + iq$)

$$\psi(t) = \mathrm{Im}\left[\frac{s^*(t)}{\mid s(t) \mid^2}\frac{d}{dt}s(t)\right] = \frac{1}{A^2(t)}\left[f(t)\frac{d}{dt}q(t) - q(t)\frac{d}{dt}f(t)\right].$$

The instantaneous frequency provides a quantitative estimate of the frequency of the real valued signal f at any instant in time. From this function, we may obtain another closely related description for the signal in terms of its frequency modulations which are given by $\mid \psi \mid$. The phase θ can be obtained from ψ by integrating. Using the initial conditions

$$\int^t \psi(\tau)d\tau = 0, \quad t = 0$$

and

$$\theta = \theta_0 \quad \text{when} \quad t = 0$$

we get

$$\theta(t) = \theta_0 + \int^t \psi(\tau)d\tau.$$

This phase function is called the unwrapped phase. It is not multi-valued and therefore the problem of choosing a principal range to compute the phase does not occur. Collectively, the amplitude modulations, frequency modulations and the (unwrapped) instantaneous phase are known as the attributes of the analytic signal. They all provide different but related information on the time history of a signal.

5.6 Case Study: FM Imaging by Real Zero Conversion

In addition to the amplitude modulations of a signal, analysis of the unwrapped phase and the frequency modulations can be of interest. FM imaging is based on displaying the Frequency modulations (FM) given by $\mid \psi \mid$. In practice, the nonlinear dependence of ψ on A^2 means that ψ is very sensitive to relatively small scale variations in A; it is ill-conditioned. Hence, a straight forward display of the frequency modulations tends to 'highlight' those positions in time where the amplitude modulations are small. The problem with this is that it produces an image of a signal that is entirely contrary to conventional AM (Amplitude Modulated) imaging. Without any additional pre-processing of the data, FM images do not convey useful information.

A variety of different methods can be used to pre-process the data in a form that is suited to the display of the frequency modulations. One method is to smooth the original signal $f(t)$ be applying a lowpass filter prior to computing ψ. The effect of doing this is to reduce the dynamic range of $1/A^2$. The smoothed FM image is then given by $\mid \psi \mid$ and the smoothed phased image obtained by integrating ψ using a filter of the type $\frac{1}{(i\omega)}$ for example.

5.6.1 Real Zero Conversion

Another way of overcoming the problems associated with FM and phase imaging is to real zero convert the signal $f(t)$. One way of expressing a real valued function is in terms of the polynomial

$$f(t) = \sum_{n=0}^{N} a_n t^n \equiv a_0 + a_1 t + a_2 t^2 + ... + a_N t^N.$$

In this form, it is clear that we can associate N roots with the signal by solving the equation $f(z) = 0$ where we have switched from t to z to indicate that these roots may be complex. If the roots of this equation are $z_1, z_2, ..., z_N$ say, then the equation $f(z) = 0$ can be written in the factored form

$$\prod_{n=1}^{N} (z - z_n) \equiv (z - z_1)(z - z_2)...(z - z_N) = 0.$$

In general, these roots can have both real and complex values and in signal analysis, are referred to as the real and complex zeros respectively. In addition to Fourier analysis and other integral transform analysis, a digital signal can be analysed by studying the location and distribution of its zeros. If the zeros are computed accurately enough, then a signal can be described in terms of its roots in the same way

that it can be described by its Fourier coefficients for example. Such signals can then be reconstructed from 'zero information'! Moreover, in some cases, the signal can be usefully processed by manipulating the values of its roots or by 'root filtering'. This is known as complex zero or root analysis.

The real zeros of a signal are just the positions in time where it changes polarity and therefore passes through zero (i.e. they are the points in time where the amplitude of the signal vanishes). The complex zeros of a signal correspond to those more subtle attributes where the amplitude is modulated but the polarity of the signal remains the same as illustrated in Figure 5.2. Complex zero signals are among the most common types of signals and a signal of this type can be uniquely determined from its zeros.

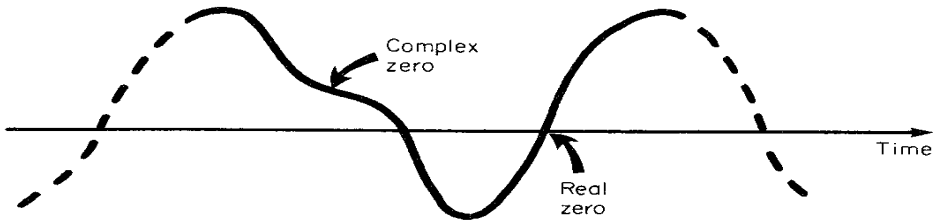


Figure 5.2: Example features in a real valued signal that correspond to its real and complex zeros.

In terms of a descriptive framework, frequency modulations access the information encoded primarily in the real zeros of a signal. It is therefore appropriate to modify the information contained in the original signal in such a way that it is completely encoded in the real zeros. Signals of this form are known as real zero signals and methods of analysing and processing them can be developed which are based entirely on the zero-crossings of the signal. The conversion of a complex zero signal to a real zero signal is known as Real Zero Conversion of RZC. Compared to the complex zeros, the real roots of a digital signal are relatively straightforward to obtain. One simply has to evaluate the points in time where the amplitude of the signal changes sign subject to a suitable interpolation algorithm for computing a good approximation to the value at which the amplitude is zero between the two data points that lie above and below the axis.

There are two ways in which a signal can be real zero converted. One way is to successively differentiate it. The problem here, is that in practice, this method requires extremely sharp filters to control out-of-band noise. Also, multiple differentiation is not strictly invertible because of the loss of addition constants. The second and more practical method (if only because of its sheer simplicity) is to add a sine wave $a\sin(\omega_0 t)$ say with a frequency ω_0 and an amplitude a equal to or greater than the highest frequency and amplitude present in the signal respectively. Thus, given a bandlimited real signal $f(t)$ whose spectrum is $F(\omega), \mid \omega \mid \leq \Omega$, the corresponding RZC signal is given by

$$f_{\mathrm{RZC}}(t) = f(t) + a\sin(\omega_0 t), \quad a \geq \mid f(t) \mid, \quad \omega_0 \geq \Omega.$$

This method is very easy to perform in analogue of digital form. A RZC signal which is generated in this way amounts to a sine wave with 'jitter' in the position of its real zeros. This jitter is determined by both the amplitude and frequency modulation in the original signal. The information on the original signal is completely encoded in the displacement of the zeros from the regular sine wave locations. We can therefore consider a RZC signal as a sine wave (whose amplitude and frequency modulations are constant in time), perturbed by a signal whose amplitude and frequency modulation vary in time. The result is to produce a signal with fluctuations in the amplitude envelope A about a, fluctuations in ψ about ω_0 and changes in the phase which perturb an otherwise linear phase function $\omega_0 t$. By adjusting the magnitude of a to an appropriate value, information encoded in the frequency modulations and phase of the original signal can be assessed in terms of how they perturb the regular sine wave. Once the signal has been real zero converted, the frequency modulations can be obtained by computing $\mid \psi \mid$.

5.6.2 Application to Seismic Imaging

The propagation and reflection of seismic waves through the earth has been studied intensively for many years, especially in the context of the exploration of oil, coal and gas. In this case, an image is generated of the interior structure of the ground at a variety of depths. This provides information on the geology of regions which are mostly or to a reasonable approximation, stratified. Hence, seismic imaging is often based on the principles of the reflection of (low frequency, e.g. 1-500 Hz) acoustic radiation from layered media. Seismic waves are typically generated by chemical explosions (for deep penetration) or vibrating impacts of short duration. By recording the time history of the reflected seismic waves using a array of geophones, information on the nature and geological significance of the earth's interior can be obtained. A typical seismic image is produced by collecting together and displaying side-by-side the seismic signals that are produced at different 'shot locations' (the location of a point source of seismic energy). Each trace is the sum of all the signals that have been detected and recorded by a linear array of geophones extending outward from the shot location after they have been corrected for normal moveout (i.e. aligned to coincide with reflections form points at a common depth). This is called 'stacking' and a seismic image of this type is therefore commonly known as a Common Depth Point (CDP) stack. It provides a set of seismic signals with an improved signal-to-noise ratio compared to the original pre-stacked data. It is convention to shade in the area under the positive lobes of each signal to emphasize the lateral correlation of the data. This makes it easier for a geologist to distinguish between layers of geological interest.

An example of a conventional CDP stack is given in Figure 5.3 together with the corresponding FM image after real zero conversion. In seismology, an image of the instantaneous frequency has an application in direct detection techniques because it is often observed (qualitatively) that a shift toward lower FM values occurs immediately below hydrocarbon-bearing structures. This is precisely what happens in the example given here in which the instantaneous frequency decreases and remains so for a relatively long period of time after the main events (as indicated by $>$). It is interesting to note that this seismic image is of one of the main coal seems of the

South Yorkshire coal field in England.

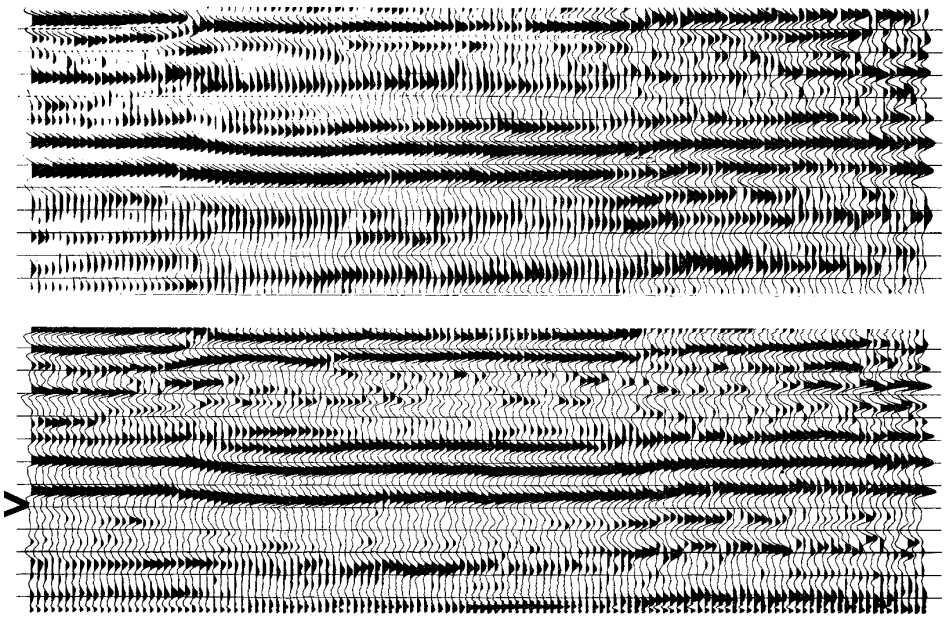


Figure 5.3: Conventional seismic image or CDP stack (top) and the RZC FM image (bottom).

5.7 STFT and The Gabor Transform

The Fourier transform, i.e.

$$F(\omega) = \int f(t) \exp(-i\omega t) dt,$$

provides a function that characterizes the spectral properties of a signal $f(t)$ in its entirety. This assumes that the signal is the result of a stationary process, the Fourier transform being a stationary transform. Suppose, however, that we wanted to investigate the spectral properties of just part of the signal and study how these properties changed as the signal developed under the assumption that the signal was the result of some non-stationary process. We could segment the signal into sections or blocks and take the Fourier transform of each of these blocks. Thus, if we consider the case where

$$f(t) = \begin{cases} f_1(t), & t \in [0, T) \\ f_2(t), & t \in [T, 2T) \\ \vdots & \\ f_N(t), & t \in [(N-1)T, NT) \end{cases}$$

T being the 'length' of each block, then we could consider an analysis of the spectrum $F_n(\omega)$ where

$$F_n(\omega) = \int_{-T/2}^{T/2} f_n(t)\exp(-i\omega t)dt, \quad n = 1, 2, ..., N.$$

This process would allow us to investigate how the spectral properties of the signal change (or otherwise) as it develops over its 'lifetime'. If the signal was composed of frequencies whose distribution are continuous over its duration (i.e. the signal is stationary), then one would expect the spectrum to be constant for each value of n. However, if the signal had a marked change in its frequency content over its life time, then this feature would be observed via a marked difference in the characteristics of $F_n(\omega)$ as a function of n. In this case, the partitioning scheme involves a 'window' that segments the signal into consecutive blocks that are of equal length. Alternatively, we can consider a windowing function w say, that extracts data as it 'slides' over the signal. We can write this process as

$$F(\omega, \tau) = \int_{-\infty}^{\infty} w(\tau - t)f(t)\exp(-i\omega t)dt.$$

The window usually has a short time duration (i.e. it windows data whose length is small compared to the duration of the signal) and the approach is thereby called the 'Short Time Fourier Transform' or STFT. The STFT yields a 'spectrogram' $F(\omega, \tau)$ which is an image of the time variations in the frequency components of the signal. Compared to the Fourier transform of $f(t)$, the Fourier transform of $f(t)w(\tau - t)$ reflects the signal's local frequency characteristics and thus provides an image of how a signal's frequency content evolves over time. It is assumed here, that the windowing function is an 'on-off' function of the form

$$w(t) = \begin{cases} 1, & |\ t\ | \leq T; \\ 0, & |\ t\ | > T \end{cases}$$

where T is the 'length' of the window. However, the functional form of the window can be changed to provide an output that is conditioned by its form in order to provide a 'spectrogram' that is not distorted by frequency components that have been generated by the discontinuity associated with the window function defined above.

If the time duration and frequency bandwidth of $w(t)$ are Δt and $\Delta\omega$ respectively, then the STFT extracts the signal's behaviour in the vicinity of $(t - \Delta t, t + \Delta t) \times (\omega - \Delta\omega, \omega + \Delta\omega)$. In order to analyse a signal at a particular time and frequency (t, ω), it is natural to desire that Δt and $\Delta\omega$ be as narrow as possible. However, the selections of Δt and $\Delta\omega$ are not independent but related via the Fourier transform. If we make the time interval around the time t small, then the bandwidth becomes high as applied to the windowed signal, i.e. to the short interval that is artificially constructed for the purpose of the analysis. For a digital signal of length N, the product $\Delta t\Delta\omega$ satisfies the relationship (see Section 3.8)

$$\Delta t\Delta\omega = \frac{2\pi}{N}$$

and thus, there is a trade-off in the selection of the time and frequency resolution. If $w(t)$ is chosen to have good time resolution (smaller Δt), then its frequency resolution deteriorates (larger $\Delta\omega$) and vice versa. A window that provides an optimum solution to this problem (i.e. in terms of providing optimum resolution in both the time and frequency domains) is the Gaussian function given by (ignoring scaling)

$$w(t) = \exp(-at^2)$$

where a is a constant. This function provides a 'taper' at the extreme ends of the signal over which the Fourier transform is to be taken. Formally, the use of a Gaussian window gives the Gabor transform which provides the common basis for computing the spectrogram of a signal, i.e. an analysis of the frequency content of a signal as a function of time. It assumes that the signal is stationary when seen through a window of limited extent at time location τ. Its application includes passive sonar and speech analysis to name but a few in which the analysis of a time varying Fourier space is required. Such analysis is typically undertaken by viewing a grey level or pseudo colour image of $\mid F(\omega, \tau) \mid$ or, using a logarithmic scale to enhance the spectrogram, an image of $\log(1+ \mid F(\omega, \tau) \mid$ for example, working of course, with digital data and a DFT.

5.8 The Wigner and the Wigner-Ville Transforms

The Wigner transform originated as a technique in quantum mechanics for computing the time varying spectral response of a multimode wavefield. Originally, the Wigner transform was derived by Wigner in the early 1930s for calculating the quantum correction terms to the Boltzman formula in which a joint distribution of position and momentum was required. Wigner devised a joint distribution that gave the quantum mechanical distributions of position and momentum.

The Wigner-Ville transform originated from the work designed to develop a transform for the analysis of the Dirac field equations in a manner that is similar to the use of Fourier transform for analysis relating to the Schrödinger equation. The Schrödinger equation was first proposed in the 1920s as a partial differential equation describing quantum physics at a time when a mathematical development of quantum mechanics was being undertaken in parallel by Heisenberg using an eigenvalue approach. In both cases, the 'spin' of a particle was not considered until Dirac derived his famous field equations which can be considered to be an extension and in someways, a completion, of Schrödinger's and Heisenberg's work to include particle spin.

The Wigner transform is defined by

$$F(\omega, t) = \int_{-\infty}^{\infty} f(t + \tau/2) f(t - \tau/2) \exp(-i\omega\tau) d\tau$$

where $f(t)$ is a real signal. The Wigner-Ville transform is an extension of this result to

$$S(\omega, t) = \int_{-\infty}^{\infty} s(t + \tau/2) s^*(t - \tau/2) \exp(-i\omega\tau) d\tau$$

where $s(t)$ is the analytic function given by

$$s(t) = f(t) + iq(t),$$

$q(t)$ being the quadrature component that is itself, given by taking the Hilbert transform of $f(t)$. In both cases, the transform can be understood in terms of 'sliding' two functions over each other and taking the resultant products as the inputs to a Fourier transform. Applications of this transform include speech analysis and control engineering, including the analysis of noise generated by the critical components of rotation contact mechanical systems such as a gear box in order to monitor pre-failure (noise) signatures.

5.8.1 Origins of the Transforms for Signal Processing

In signal analysis, the Wigner and/or Wigner-Ville transform can be justified in terms of an approach to solving the problems associated with the application of the Gabor transform and the STFT in general. With STFT analysis the result is strongly influenced by the choice of the window function $w(t)$. In the interpretation of a spectrogram, it is always necessary to take into account the shape and size of the window used. The best results are usually achieved if a window is used that matches the characteristics of the signal. The effect of the window on the spectrum will be minimal if the characteristics of the signal are not altered by the application of the window. However, in many applications this may be difficult, or even impossible, especially when the characteristics of the signal change rapidly as a function of time. In such cases, it is necessary to find an appropriate time dependent window. A simple way to achieve this is to use the signal itself as the window and consider the transform

$$S(t,\omega) = \int_{-\infty}^{\infty} s(\tau)s(t-\tau)\exp(-i\omega\tau)d\tau$$

or

$$S(t,\omega) = \int_{-\infty}^{\infty} s(\tau)s^*(t-\tau)\exp(-i\omega\tau)d\tau$$

with symmetric forms

$$S(t,\omega) = \int_{-\infty}^{\infty} s(t+\tau/2)s(t-\tau/2)\exp(-i\omega\tau)d\tau$$

and

$$S(t,\omega) = \int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\exp(-i\omega\tau)d\tau$$

respectively, which, in the latter case, is the Wigner-Ville transform of the analytic signal $s(t)$ defined earlier - sometimes referred to as the auto-Wigner-Ville distribution. Thus, one of the interpretations of the Wigner and/or Wigner-Ville transform is that it is the Fourier transform of the signal after it has been windowed with the

reverse of itself. Clearly, for applications in digital signal processing, the discrete Fourier transform can be applied to compute this transform.

The application of the Wigner transform in signal analysis is non-standard but is useful for specialised diagnostic purposes. It is sensitive to features in noise that are not easily discernible in conventional time-frequency spectral analysis. The transform provides a time-frequency distribution that, in general, yields excellent time and frequency resolution together with other valuable properties. However, it also yields some undesirable outputs such as aliasing and cross-term interference.

5.8.2 Properties of the Wigner-Ville Transform

Realness

The Wigner-Ville (WV) transform is real since if

$$S(t,\omega) = \int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\exp(-i\omega\tau)d\tau$$

then

$$S^*(t,\omega) = \int_{-\infty}^{\infty} s^*(t+\tau/2)s(t-\tau/2)\exp(i\omega\tau)d\tau = -S(t,\omega).$$

Time-shift Invariance

An important feature of the WV transform is that it is invariant to time and frequency shifts. If the WV transform of a signal $s(t)$ is $S(t,\omega)$, then the WV transform of the time-shifted version $s'(t) = s(t-t',\omega)$ is a time-shifted WV transform of $s(t)$, i.e.

$$S'(t,\omega) = S(t-t',\omega).$$

Frequency Modulation Invariance

If the WV transform of a signal $s(t)$ is $S(t,\omega)$, then the WV transform of the frequency modulated version $s'(t) = s(t)\exp(i\omega' t)$ is a frequency shifted WV transform of $s(t)$, i.e.

$$S'(t,\omega) = S(t,\omega-\omega').$$

Power Spectrum

Integration of the WV transform over time yields the power spectrum of the signal $s(t)$ since

$$\int_{-\infty}^{\infty} S(t,\omega)dt = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\exp(-i\omega\tau)dtd\tau$$

$$= \int_{-\infty}^{\infty} \exp(-i\omega\tau) \int_{-\infty}^{\infty} s(t)s^*(t-\tau)d\tau dt = S(\omega)S^*(\omega) = \mid S(\omega) \mid^2 .$$

Instantaneous Energy

Integration of the WV transform over frequency yields an expression for the instantaneous energy of the signal, i.e.

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} S(t,\omega)d\omega = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\exp(-i\omega\tau)d\tau d\omega$$

$$= \int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(-i\omega\tau)d\omega d\tau$$

$$= \int_{-\infty}^{\infty} s(t+\tau/2)s^*(t-\tau/2)\delta(\tau)d\tau = \mid s(t) \mid^2 .$$

Thus, from Rayleigh's theorem (i.e. the energy theorem), since

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} \mid S(\omega) \mid^2 d\omega = \int_{-\infty}^{\infty} \mid s(t) \mid^2 dt$$

we have

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S(t,\omega)d\omega dt = \int_{-\infty}^{\infty} \mid s(t) \mid^2 dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mid S(\omega) \mid^2 d\omega.$$

Instantaneous Frequency

Let $s(t) = A(t)\exp[i\theta(t)]$ where A and θ are real. Then

$$\langle\omega\rangle_t = \mathrm{Im}\frac{\frac{1}{2\pi}\int_{-\infty}^{\infty} i\omega\bar{S}(t,\omega)d\omega}{\frac{1}{2\pi}\int_{-\infty}^{\infty}\bar{S}(t,\omega)d\omega} = \mathrm{Im}\frac{\frac{1}{2\pi}\int_{-\infty}^{\infty} i\omega\bar{S}(t,\omega)d\omega}{[s(t)]^2} = 2\frac{d}{dt}\theta(t)$$

where

$$\bar{S}(t,\omega) = \int_{-\infty}^{\infty} s(t+\tau/2)s(t-\tau/2)\exp(-i\omega\tau)d\tau$$

and $\langle\ \ \rangle_t$ denotes the mean or average value at a time t. This result comes from the fact that

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega\ \ i\omega\int_{-\infty}^{\infty} d\tau s(t+\tau/2)s(t-\tau/2)\exp(-i\omega\tau)$$

$$= -\int_{-\infty}^{\infty} d\tau s(t+\tau/2)s(t-\tau/2)\frac{d}{d\tau}\frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega \exp(-i\omega\tau)$$

$$= -\int_{-\infty}^{\infty} d\tau s(t+\tau/2)s(t-\tau/2)\frac{d}{d\tau}\delta(\tau) = \frac{d}{dt}[s(t)]^2 = 2s(t)\frac{d}{dt}s(t).$$

Thus,

$$\langle\omega\rangle_t = 2\mathrm{Im}\frac{1}{s(t)}\frac{d}{dt}s(t) = 2\mathrm{Im}\frac{d}{dt}\ln s(t)$$

and the instantaneous frequency can be defined as (see Section 5.5.5)

$$\frac{d}{dt}\theta(t) = \mathrm{Im}\left[\frac{d}{dt}\ln s(t)\right].$$

This result shows that at an instant in time t, the mean frequency content of the signal for the WV distribution is given by the instantaneous frequency of the signal. Thus, another way of computing the instantaneous frequency (and hence the unwrapped phase) of a signal (see Section 5.5.5) is by computing the mean frequency using the WV distribution.

Group Delay

Assume that the Fourier transform of a signal $s(t)$ is given by $S(\omega) = A(\omega)\exp[i\theta(\omega)]$. Then the first derivative of the Fourier phase θ is called the group delay. Noting that

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} dt \quad it\int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\exp(-i\Omega t)$$

$$= -\int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\frac{d}{d\Omega}\frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(-i\Omega t)dt$$

$$= -\int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\frac{d}{d\Omega}\delta(\Omega) = \frac{d}{d\omega}[S(\omega)]^2 = 2S(\omega)\frac{d}{d\omega}S(\omega)$$

and that

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} dt\int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\exp(-i\Omega t)$$

$$= \int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(-i\Omega t)dt$$

$$= \int_{-\infty}^{\infty} d\Omega S(\omega+\Omega/2)S(\omega-\Omega/2)\delta(\Omega) = [S(\omega)]^2$$

then, since

$$\frac{d}{d\omega}\theta(\omega) = \mathrm{Im}\left[\frac{d}{d\omega}\ln S(\omega)\right]$$

we can write

$$\langle t\rangle_\omega = \mathrm{Im}\frac{\frac{1}{2\pi}\int_{-\infty}^{\infty} it\bar{s}(t,\omega)dt}{\frac{1}{2\pi}\int_{-\infty}^{\infty}\bar{s}(t,\omega)dt} = 2\frac{d}{d\omega}\theta(\omega).$$

where

$$\bar{s}(t,\omega) = \int_{-\infty}^{\infty} S(\omega+\Omega/2)S(\omega-\Omega/2)\exp(-i\Omega t)d\Omega$$

Thus, the mean time of the WV distribution at a frequency ω is given by the group delay.

The properties of the WV transform compared with those of the Gabor transform (a spectrogram) are summarized in the table below which is based on those properties that a time-frequency distribution should ideally have and which of these properties is fulfilled by each transform.

Property	Gabor Transform (Spectrogram)	Wigner-Ville Transform
Shift in time	YES	YES
Shift in frequency	YES	YES
Realness	YES	YES
Finite support in time	NO	YES
Finite support in frequency	NO	YES
Positivity	YES	NO
Integration of time =power spectrum	NO	YES
Integration of frequency =instantaneous energy	NO	YES
Average frequency at time t =instantaneous frequency	NO	YES
Average time at frequency ω =group delay	NO	YES

5.8.3 Cross Term Interference

Suppose we have a signal that is composed of the sum of two signals s_1 and s_2. Then

$$s(t) = s_1(t) + s_2(t)$$

and the spectrum of s will be

$$S(\omega) = S_1(\omega) + S_2(\omega).$$

The power spectrum is given by

$$\mid S \mid^2 = \mid S_1 + S_2 \mid^2 = \mid S_1 \mid^2 + \mid S_2 \mid^2 + 2\mathrm{Re}[S_1^* S_2] \neq \mid S_1 \mid^2 + \mid S_2 \mid^2 .$$

Thus, the power spectrum is not the sum of the power spectrum of each signal. The physical reason for this is that when two signals are added, the waveforms may add and interfere in a number of ways to give different weights to the original frequencies. For the WV transform

$$S(t,\omega) = S_{s_1}(t,\omega) + S_{s_2}(t,\omega) + 2\mathrm{Re}[S_{s_1,s_2}(t,\omega)]$$

In addition to the two auto-terms, this expression contains a cross-term. Because this cross-term has an amplitude that is twice as large of the auto-terms (i.e. a coefficient of 2), it tends to obscure the auto-terms. Methods to reduce the cross-term interference patterns without destroying the useful properties of the WV distribution are therefore important in developing algorithms for WV transform based time-frequency analysis. Note that the WV distribution of an N-component signal consists of N auto-terms and $N(N-1)/2$ cross-terms. Thus while the number of auto-terms grows linearly with the number of signal components, the number of cross-terms grows quadratically with N.

With real signals, the cross-terms produce significant interference which can be reduced by smoothing the output. However, by using the analytic signal, the cross-term interference is reduced because the negative frequency components are zero (the analytic signal is a single side-band signal). Thus, use of the analytic function avoids all cross-terms that are associated with negative frequency components. However, this is only the case for real valued signals in which the Hilbert transform is computed to generate the analytic signal as in Section 5.5.3. For complex signals, the spectra are not symmetric and therefore it is not possible to neglect the negative frequency components.

5.8.4 Smoothing the Wigner-Ville Distribution

Since most signals have multiple components, the WV distribution is 'confused' by cross-terms which grow quadratically with the number of signal components and makes the interpretation of the time-frequency distribution difficult. In general, the auto-terms of the WV distribution are relatively smooth compared to the cross-terms which are strongly oscillatory. This allows for the application of a lowpass filter $P(t,\omega)$ to the WV distribution to reduce cross-term interference obtained via the convolution process

$$R(T,\Omega) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S(t,\omega)P(T-t,\Omega-\omega)dtd\omega$$

where R denotes the smoothed distribution. With regard to the nature of the low-pass smoothing filter, a Gaussian filter provides for symmetry in both the time and frequency domains. Although application of a Gaussian lowpass filter, for example, can substantially suppress the cross-term interference, this is obtained at the expense of resolution and a trade-off must be introduced between the degree of smoothing and the resolution acceptable for a time-frequency analysis. In the application of a

Gaussian lowpass filter, this trade-off is compounded in the value of the standard deviation that is used.

5.8.5 The Discrete Wigner-Ville Transform

The definition a the discrete WV distribution is not straightforward and several approaches are possible. By letting $u = \tau/2$ we can write

$$S(t,\omega) = 2\int s(t+u)s^*(t-u)\exp(-2i\omega u)du.$$

For a regularly sampled signal, with sampling interval Δt, we can write

$$S(t_n,\omega_m) = 2\Delta t\sum_k s(t_n+k\Delta t)s(t_n-k\Delta t)\exp(-2i\omega_m k\Delta t)$$

and for $t_n = n\Delta t$ and $\omega_m = m\Delta\omega$ we have

$$S(n\Delta t, m\Delta\omega) = 2\Delta t\sum_k s[(n+k)\Delta t]s^*[(n-k)\Delta t]\exp(-2imk\Delta\omega\Delta t).$$

Hence, for a regular array s_k, the discrete WV distribution is given by

$$S_{nm} = \sum_k s_{n+k}s^*_{n-k}\exp(-2\pi imk/N)$$

where

$$\Delta\omega\Delta t = \frac{\pi}{N}$$

and is thus, compounded in the DFT of the array $s_{n+k}s^*_{n-k}$ instead of s_k. However, note that the sampling period in the frequency domain is now $\pi/(N\Delta t)$ rather than $2\pi/(N\Delta t)$ as is the case with the discrete Fourier transform.

5.8.6 Applications

There has been an increasing interest in the applications of the WV transform to the analysis of signals particularly with regard to the detection of information in noise. For example, the WV distribution can be used to analyse time domain averaged vibration signals of a gearbox in which different faults, such as tooth cracking and pitting, produce different and significant (i.e. detectable) patterns in the WV distribution (e.g. Staszewski W J, Worden K and Tomlinson G R, 1997, *Time-Frequency Analysis in Gearbox Fault Detection using the Wigner-Ville Distribution and Pattern Recognition*, Mechanical Systems and Signal Processing, **11**(5), 673-692). As with a spectrogram, the patterns produced (if significant) can be used to generate a set of templates that can be correlated with an input WV distribution or used for more advanced pattern recognition systems (e.g. for training a neural pattern recognition engine).

Other applications of the WV distribution include seismic signal processing (acoustic well logging) and fuel injection monitoring via analysis of the injector vibrations obtained using a transducer positioned on the outer surface of the injector body. Noise and vibration analysis is applicable to a wide range of engineering applications and within the context of mechanical, control and process engineering, the WV distribution has an important role to play.

5.9 The Riemann-Liouville and the Wyle Transforms

These transforms are an essential basis for the description and analysis of random fractal or statistical self-affine signals. The Riemann-Liouville transform is given by

$$s(t) = \frac{1}{\Gamma(q)} \int_0^t \frac{f(\tau)}{(t-\tau)^{1-q}} d\tau$$

where $q > 0$ and the Weyl transform, by

$$s(t) = \frac{1}{\Gamma(q)} \int_t^\infty \frac{f(\tau)}{(t-\tau)^{1-q}} d\tau, \quad q > 0.$$

In both cases, Γ denotes the 'Gamma function' given by

$$\Gamma(q) = \int_0^\infty x^{q-1} \exp(-x) dx.$$

These transforms are the classical fractional integral operators and for integer values of q (i.e. when $q = n$ where n is a non-negative integer), the Riemann-Liouville transform reduces to the standard Riemann integral. Note that the Riemann-Liouville transform is a causal convolution of the function $f(t)$ with $t^{q-1}/\Gamma(q)$. In the case when $f(t)$ is white noise, i.e. noise whose power spectrum is uniform or 'white', these transforms describe a signal whose Power Spectral Density Function (PSDF) is characterized by $1/(i\omega)^q$ or

$$\mid S(\omega) \mid^2 \propto \frac{1}{\mid \omega \mid^{2q}}.$$

Many noise types found in nature have a PSDF of this form. Moreover, this spectral form is characteristic of signals whose statistical distribution is the same at different scales. This can be shown by taking the Riemann-Liouville transform of a function in which a scale length parameter λ has been introduced; thus with $x = \lambda\tau$,

$$\frac{1}{\Gamma(q)} \int_0^t \frac{f(\lambda\tau)}{(t-\tau)^{1-q}} d\tau = \frac{1}{\lambda^q \Gamma(q)} \int_0^{\lambda t} \frac{f(x)}{(\lambda t - x)^{1-q}} dx = \frac{1}{\lambda^q} s(\lambda t).$$

Hence, with Pr denoting the Probability Density Function or PDF of the function s, we can write

$$\lambda^q \mathrm{Pr}[s(t)] = \mathrm{Pr}[s(\lambda t)]$$

which is the equation describing statistical self-affinity, namely, the PDF of the signal is the same over different scale lengths λ subject to a change in amplitude determined by λ^q. In other words, the 'shape' of the PDF is the same at different scales. This is the fundamental principle of statistical self-affinity and is discussed further in Part IV of this work (Chapter 17).

5.10 The z-Transform

The z-transform is used as a tool for the analysis and design of linear sampled data systems, i.e. typically in cases, when a processing system is modelled via linear difference equations for example. The z-transform is defined by

$$\hat{Z}[f_k] = F(z) = \sum_{k=0}^{\infty} f_k z^{-k}$$

where z is an arbitrary complex number. $F(z)$, which is the sum of complex numbers, is also a complex number. This transform has many similar properties to the (discrete) Fourier transform such as the convolution theorem for example and the shifting property, i.e.

$$\hat{Z}[f_{k+1}] = zF(z) - zf_0$$

for which the general result is

$$\hat{Z}[f_{k+n}] = z^n F(z) - \sum_{j=0}^{n-1} f_j z^{n-j}, \quad k \geq -n.$$

However, the process of determining the inverse z-transform is similar to that of determining the inverse Laplace transform using tabulated results rather than through direct computation. As with other transforms, it is typical to compute the z-transform for a range of different functions. One such function that is important with regard to the z-transform is the geometric sequence

$$f_k = \begin{cases} a^k, & k \geq 0; \\ 0, & k < 0 \end{cases}$$

for any real number a where

$$F(z) = \sum_{k=0}^{\infty} a^k z^{-k} = \sum_{k=0}^{\infty} (az^{-1})^k = \frac{1}{1 - az^{-1}} = \begin{cases} \frac{z}{z-a}, & \mid z \mid > \mid a \mid; \\ \text{unbounded}, & \mid z \mid < \mid a \mid . \end{cases}$$

Thus, for example, the z-transform of a unit step function is

$$U_k \leftrightarrow \frac{z}{z-1}$$

where

$$U_k = \begin{cases} 1, & k \geq 0; \\ 0, & k < 0. \end{cases}$$

and $\leftrightarrow$ denotes the z-transform process.

Suppose that a linear discrete system is described by the difference equation

$$c_{k+1} - c_k = r_k, \quad c_0 = 0$$

and we want to find the impulse response function satisfying the equation

$$c_{k+1} - c_k = \delta_k$$

where δ_k is the discrete unit impulse function (the Kronecker delta function) given by

$$\delta_k = \begin{cases} 1, & k = 0; \\ 0, & k \neq 0. \end{cases}$$

Taking the z-transform, we can transform this equation into z-space to give

$$(z-1)C(z) = 1$$

since

$$\delta_k \leftrightarrow 1.$$

Thus,

$$C(z) = \frac{1}{z-1}.$$

Now,

$$\frac{z}{z-1} \leftrightarrow (-1)^k$$

and hence, from the shifting property

$$zC(z) - zc_0 \leftrightarrow c_{k+1}$$

we have

$$\hat{Z}^{-1}[C(z)] = (-1)^{k-1}, \quad k \geq 1.$$

The z-transform has traditionally been used as a key analytical tool for designing control systems that are described by linear difference equations and linear discrete systems in general. Like the Fourier and Laplace transforms, we can consider a model for the output of a time invariant linear system of the form

$$s_k = p_k \otimes f_k \equiv \sum_{j=0}^{\infty} p_{k-j} f_j$$

where f_k and s_k are the inputs and output respectively and p_k is the impulse response function (the weighting sequence as it is sometime called in this context). Now

$$\hat{Z}[s_k] = S(z) = \sum_{k=0}^{\infty} s_k z^{-k}$$

or

$$S(z) = \sum_{k=0}^{\infty} \left[\sum_{j=0}^{\infty} p_{k-j} f_j \right] z^{-k} = \sum_{j=0}^{\infty} f_j \sum_{k=0}^{\infty} p_{k-j} z^{-k}.$$

Now, with $\ell = k - j$,

$$S(z) = \sum_{j=0}^{\infty} f_j \sum_{\ell=-j}^{\infty} p_\ell z^{-\ell-j} = \sum_{j=0}^{\infty} f_j z^{-j} \sum_{\ell=0}^{\infty} p_\ell z^{-\ell} = F(z)P(z).$$

Thus, we have

$$P(z) = \frac{S(z)}{F(z)}$$

which is the z-transfer function of a linear discrete system.

5.11 The Wavelet Transform

Wavelet signal analysis is based on a correlation-type operation which includes a scaling property in terms of the amplitude and temporal extent of the correlation kernel under the assumption that the signal is non-stationary. It provides an alternative to the classical short time Fourier transform (the Gabor transform). Wavelet theory has been developed as a unifying framework for non-stationary signal analysis only recently, although similar ideas have been around for many years. The idea of looking at a signal at various scales and analysing it with various resolutions emerged independently in many different fields of mathematics, physics and engineering. The term wavelet or 'Ondelette' originates primarily from theoretical work undertaken mainly in France in the mid-1980s which was inspired by geophysical problems and seismic signal processing.

The wavelet transform can be evaluated in terms of the effect of a correlation over different scales. If a signal contains a specific identifiable feature (i.e. an amplitude variation of finite support within the data stream) then a correlation of such a signal with an identical feature (a template) provides a correlation function with a flag - a 'spike' - in the locality of the data stream in which the feature occurs, a process that is one of the most basic methods of pattern recognition in signal an image analysis. However, if the feature has a different scale (in terms of its amplitude and/or temporal extent) then a correlation process will not provide a flag. This is because, although the feature may look the same as the template, it does not provide a match in terms of its scale; the correlation integral is not scale invariant. In order to address this shortcoming, one could apply a multiple correlation process in which a variety of different templates are used, each of which are of a different scale or resolution. Applying such a process in a systematic fashion, a number of correlation functions will be obtained, one of which will provide a flag in the case when the scale of the template is the same as that of the feature in the signal the template describes. If $f(t)$ is the signal in question and $g(t)$ is the template, then we can consider the approach considered above to be compounded mathematically as

$$s(t,\lambda) = \lambda \int f(\tau) g\left[\lambda(\tau - t)\right] d\tau$$

where λ is a scaling parameter.

The multi-resolution properties of the wavelet transform have been crucial to their development and success in the analysis and processing of signals and images. Wavelet transformations play a central role in the study of self-similar or fractal signals and images. The transform constitutes as natural a tool for the manipulation of self-similar or scale invariant signals as the Fourier transform does for translation invariant such as stationary and periodic signals.

In general, the (continuous) wavelet transformation of a signal $f(t)$ say

$$f(t) \leftrightarrow F_L(t)$$

is defined in terms of projections of $f(t)$ onto a family of functions that are all normalized dilations and translations of a prototype 'wavelet' function w, i.e.

$$\hat{W}[f(t)] \equiv F_L(t) = \int f(\tau) w_L(\tau, t) d\tau$$

where

$$w_L(t,\tau) = \frac{1}{\sqrt{\mid L \mid}} w\left(\frac{\tau - t}{L}\right).$$

Here, we assume that the wavelet function is real; for complex wavelets, the complex conjugate of w is taken. The parameters L and τ are continuous dilation and translation parameters respectively and take on values in the range $-\infty < L, \quad \tau < \infty, \quad L \neq 0$. Note that the wavelet transformation is essentially a correlation in which $w(t)$ is the kernel but with a factor L introduced. The introduction of this factor provides dilation and translation properties into the correlation integral that gives it the ability to analyse signals in a multi-resolution role. Note that the correlation integral is now a function of L and that wavelet analysis is self-similar. In general, a multi-resolution signal analysis is a framework for analysing signals based on isolating variations in the signal that occur on different temporal or spatial scales. The basic analysis involves approximating the signal at successively coarser scales through repeated application of a smoothing (correlation) operator. Just as the short time Fourier transform can be used to compute a spectrogram, i.e. an image of the power spectrum $\mid F(\omega,\tau) \mid^2$ say, so by computing an image of $\mid F_L(t) \mid^2$, we can analyse the wavelet decomposition of a signal. This is known as a 'scalogram'. It is a distribution of the energy of the signal in the time-scale plane and is thus, expressed in power per unit frequency, like the spectrogram. However, in contrast to the spectrogram, the energy of the signal is here, distributed with different resolutions.

Wavelet analysis results in a set of wavelet coefficients which indicate how close the signal is to a particular basis or wavelet function w. Thus, we expect that any general signal can be represented as a decomposition into wavelets, i.e. that the original waveform is synthesized by adding elementary building blocks, of constant shape but different size and amplitude. Another way to say this is that we want the wavelets $w_L(t)$ to behave just like an orthogonal basis. As with many of the other integral transforms discussed so far, orthogonality can be used to develop an inverse transform and the (discrete) wavelet transform is no exception. A necessary and sufficient condition for this transformation to be invertible is that $w_L(t)$ satisfy the *admissibility condition*

$$\int \mid W(\omega) \mid^2 \mid \omega \mid^{-1} d\omega = C_w < \infty$$

where W is the wavelets Fourier transform, i.e.

$$W(\omega) = \int w(t) \exp(-i\omega t) dt.$$

Provided w has reasonable decay at infinity and smoothness, as is usually the case in practice, the admissibility condition above is equivalent to the condition

$$\int w(t) dt = 0.$$

For any admissible $w(t)$, the wavelet transform has an inverse given by

$$f(t) = \hat{W}^{-1}[F_L(\tau)] = \frac{1}{C_w} \int \int F_L(\tau) w_L(t,\tau) L^{-2} dL d\tau.$$

There are a wide variety of wavelets available (i.e. functional forms for w_L) which are useful for processing signals in 'wavelet space' when applied in discrete form. The discrete wavelet transform of the digital signal $f_n, \quad n = 0, 1, ..., N$ say, can be written as

$$F_{ij} = L^{-j/2} \sum_{n=0}^{N} f_n w\left(\frac{n-i}{L^j}\right)$$

where the value of L is equal to 2 (for a so called 'dyadic transform'). The properties of the wavelets vary from one application to another but in each case, the digital signal f_i is decomposed into a matrix F_{ij} (a set of vectors) where j is the 'level' of the decomposition.

5.12 Discussion

There are a wide variety of integral transforms that can be applied in discrete form to digital signals for their analysis and processing. Many of the these transforms yield properties that are better or worse in terms of doing 'something useful' with the signal. Thus, if one is interested in observing and working with the properties of a signal at different resolutions, then the wavelet transform provides an ideal 'working environment' or if analysis is required of the response of a system to a known input where the system can be described by a set of discrete linear difference equations, then the z-transform plays a central role. However, in terms of linking the characteristics of a signal to the 'physics' of the system that produces it, particularly with regard to the interactions of wavefields with matter and the design of sensors to record such interactions, the Fourier transform has and continues to rein supreme. This is because it provides the most general and comprehensive solution (via the Green's function for example) to physical systems described by differential equations and most of the laws of physics can be expressed in the form of differential equations; that is our legacy from Isaac Newton.

5.13 Summary of Important Results

The Laplace Transform

$$F(p) = \int_0^\infty f(t) \exp(-pt) dt,$$

$$f(t) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} F(p) \exp(px) dp.$$

The Sine Transform

$$F(\omega) = \int_0^\infty f(t) \sin(\omega t) dt,$$

$$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \sin(\omega t) d\omega.$$

The Cosine Transform

$$F(\omega) = \int_0^\infty f(t) \cos(\omega t) dt,$$

$$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \cos(\omega t) d\omega.$$

The (discrete) Walsh Transform

$$f(t) = \sum_{n=0}^{N-1} F_n \mathrm{WAL}(n, t),$$

$$F_n = \frac{1}{T} \int_0^T f(t) \mathrm{WAL}(n, t);$$

where the Walsh functions $\mathrm{WAL}(n, t)$ describe a set of rectangular or square waveforms taking only two amplitude values, namely +1 and -1, and form an orthogonal set of functions.

The Cepstral Transform

$$\hat{f}(t) = \frac{1}{2\pi} \int_{-\infty}^\infty \log[F(\omega)] \exp(i\omega t) d\omega$$

where

$$F(\omega) = \int_{-\infty}^\infty f(t) \exp(-i\omega t) dt.$$

The Hilbert Transform

$$q(t) = \frac{1}{\pi} \int_{-\infty}^\infty \frac{f(\tau)}{\tau - t} d\tau$$

The Stieltjes-Riemann Transform

$$q(t) = \frac{1}{\pi} \int_{-\infty}^\infty \frac{f(\tau)}{t - \tau} d\tau$$

The Analytic Signal

$$s(t) = f(t) + iq(t), \quad q(t) = \frac{1}{\pi t} \otimes f(t).$$

Amplitude Modulations

$$A(t) = \sqrt{f^2(t) + q^2(t)}$$

Frequency Modulations

$$| \psi |$$

where

$$\psi = \frac{1}{A^2(t)} \left[f(t) \frac{d}{dt} q(t) - q(t) \frac{d}{dt} f(t) \right].$$

Unwrapped Phase

$$\theta(t) = \theta(t = 0) + \int^{t} \psi(\tau) d\tau$$

The Gabor Transform

$$F(\omega, \tau) = \int_{-\infty}^{\infty} w(t + \tau) f(t) \exp(-i\omega t) dt$$

where

$$w(t) = \exp(-at^2).$$

The Wigner Transform

$$F(\omega, \tau) = \int_{-\infty}^{\infty} f(t + \tau/2) f(t - \tau/2) \exp(-i\omega t) dt$$

The Wigner-Ville Transform

$$F(\omega, \tau) = \int_{-\infty}^{\infty} s(t + \tau/2) s^*(t - \tau/2) \exp(-i\omega t) dt$$

The Riemann-Liouville Transform

$$F(t) = \frac{1}{\Gamma(q)} \int_{0}^{t} \frac{f(\tau)}{(t - \tau)^{1-q}} d\tau, \quad q > 0$$

where

$$\Gamma(q) = \int_0^\infty x^{q-1} \exp(-x) dx.$$

The Weyl Transform

$$F(t) = \frac{1}{\Gamma(q)} \int_t^\infty \frac{f(\tau)}{(t-\tau)^{1-q}} d\tau, \quad q > 0.$$

The z-Transform

$$F(z) = \sum_{k=0}^{\infty} f_k z^{-k}$$

The Wavelet Transform

$$F_L(t) = \int_{-\infty}^{\infty} f(\tau) w_L(t, \tau) d\tau$$

where

$$w_L(t, \tau) = \frac{1}{\sqrt{\mid L \mid}} w\left(\frac{\tau - t}{L}\right).$$

5.14 Further Reading

- Jury E I, *Theory and Application of the z-transform Method*, Wiley, 1964.
- Oberhettinger F and Badii L, *Tables of Laplace Transforms*, Springer, 1973.
- Oldham K B and Spanier J, *The Fractional Calculus*, Academic Press, 1974.
- Rabiner L R and Gold B, *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1975.
- Beauchamp K G, *Walsh Functions and their Applications*, Academic Press, 1975.
- Watson E J, *Laplace Transforms and Applications*, Van Nostrand Reinhold, 1981.
- Candy J V, *Signal Processing*, McGraw-Hill, 1988.

- Cohen L, *Time-Frequency Analysis*, Prentice-Hall, 1995.

- Mecklenbräuker W and Hlawatch F (Eds.), *The Wigner Distribution*, Elsevier, 1997.

5.15 Problems

5.1 If

$$\hat{L}f(t) \equiv \int_0^\infty f(t)\exp(-pt)dt \quad \text{and} \quad \int_0^\infty \exp(-u^2)du = \frac{\sqrt{\pi}}{2}$$

show that

$$\hat{L}\cosh(at) = \frac{p}{p^2-a^2}, \quad p > \mid a \mid; \quad \hat{L}t\exp(at) = \frac{1}{(p-a)^2}, \quad p > a;$$

$$\hat{L}\left(\frac{1}{\sqrt{t}}\right) = \sqrt{\frac{\pi}{p}}, \quad p > 0 \quad \text{and} \quad \hat{L}\delta(t-a) = \exp(-ap).$$

5.2 Given that

$$J_0(at) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2}\left(\frac{at}{2}\right)^{2n},$$

prove that

$$\hat{L}J_0(at) = \frac{1}{\sqrt{a^2+p^2}}.$$

5.3 Using the convolution theorem for Laplace transforms, show that if

$$y(x) = f(x) + \int_0^x g(x-u)y(u)du$$

then

$$\hat{L}y(x) = \frac{\hat{L}f(x)}{1-\hat{L}g(x)}.$$

Hence, solve the integral equation

$$y(x) = \sin(3x) + \int_0^x \sin(x-u)y(u)du.$$

5.4 Using the convolution theorem for Laplace transforms, find the general solution to the equation

$$\frac{d^2y}{dx^2} + 4y = f(x)$$

where $f(x)$ is an arbitrary function and where $y(0) = 1$ and $y'(0) = 1$.

5.5 Solve

$$\phi(x) = x - \int_0^x (t-x)\phi(t)dt$$

using Laplace transforms.

5.6 Solve the equation

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$$

in the region $x > 0, t > 0$, using the sine transform subject to the conditions: $u(0,t) = 1$ for $t > 0, u(x,0) = 0$ for $x > 0$, and $u \to 0$ as $x \to \infty$.

5.7 Compute the z-transform of the sinusoidal function

$$f(k) = \begin{cases} \sin(k\omega T), & k \geq 0; \\ 0, & k < 0. \end{cases}$$

and the cosinusoidal function

$$f(k) = \begin{cases} \cos(k\omega T), & k \geq 0; \\ 0, & k < 0. \end{cases}$$

5.8 A linear discrete control system is described by

$$c(k+2) + a_1 c(k+1) + a_2 c(k) = b_0 r(k+2) + b_1 r(k+1) + b_2 r(k)$$

where $c(k) = 0$ and $r(k) = 0$ for $k < 0$. Find the z-transfer function for this system. Derive the transfer function for the general case when

$$c(k) + a_1 c(k-1) + ... + a_n c(k-n) = b_0 r(k) + b_1 k - 1) + ... + b_n r(k-n)$$

when $c(k) = 0$ and $r(k) = 0$ for $k < 0$ (i.e. zero initial conditions and a causal input).

5.9 The central limit theorem stems from the result that the convolution of two functions generally yields a function which is smoother than either of the functions that are being convolved. Moreover, if the convolution operation is repeated, then the result starts to look more and more like a Gaussian function. Prove the central limit theorem for the case when the functions being convolved are tophat functions. In particular, show that as the number of convolutions of a tophat function increases, the result approximates a Gaussian function; specifically, if

$$f(t) = \begin{cases} 1, & \mid t \mid \leq \frac{1}{2}; \\ 0, & \text{otherwise} \end{cases}$$

then

$$\prod_{n=1}^{N}{}^{\otimes} f_n(t) \equiv f_1(t) \otimes f_2(t) \otimes ... \otimes f_N(t) \simeq \sqrt{\frac{6}{\pi N}} \exp(-6t^2/N)$$

where $f_n(x) = f(x), \ \forall n$ and N is large. (Hint: Consider the effect of multiple convolutions in Fourier space and then work with a series representation of the result.)

5.10 The Morlet wavelet (without the correction term) is given by

$$w(t) = \pi^{-\frac{1}{4}} \exp(i\omega_0 t - t^2/2)$$

where ω_0 is the characteristic frequency of the wavelet.

(i) Calculate the Fourier transform of this wavelet, noting that

$$\exp(-at^2) \Longleftrightarrow \sqrt{\frac{\pi}{a}} \exp(-\omega^2/4a)$$

(ii) Define analytically the centre frequency of the Morlet wavelet, i.e. the frequency maximising the power spectrum of this wavelet function.

(ii) Explain how the Morlet wavelet can detect a 'brick wall' discontinuity, i.e. a signum function with amplitude a, in other words, when a constant positive signal $f(t) = a$ suddenly drops to a constant negative value $f(t) = -a$.

5.11 (i) Compute the Wigner-Ville transform of the multi-component signal composed of four complex sinusoids, i.e.

$$f(t) = \sum_{n=1}^{4} x_n(t)$$

where

$$x_n = a_n \exp(i\omega_n t)$$

and characterise the cross-terms.

(ii) Describe the relationship between the Wigner-Ville transform and the instantaneous auto-correlation function of a signal.

Part II

Computational Linear Algebra

Chapter 6

Matrices and Matrix Algebra

Many of the algorithms associated with processing digital signals are based on the mathematical ideas and analysis discussed in Part I. However, in practice, the processing of a digital signal requires such analysis to be converted into discrete or digital form. In other words, instead of working with a continuous function $f(t)$ (an analogue signal) and indefinite or definite integrals, we are required to work with discrete functions (digital signals), vectors or arrays of the type $\mathbf{f} \equiv f_i = f_1, f_2, ..., f_N$ (where N is the size of the array) and consider indefinite or definite sums. This inevitably leads to problems being specified in terms of matrices and matrix equations for which knowledge of matrix algebra is a necessary requirement. Moreover, such matrix equations typically describe sets of linear simultaneous equations for which solutions need to be found. Further, it is often necessary to extend this study to encompass the linear eigenvalue problem.

A typical and re-occurring theme is related to the convolution of two discrete functions in which the convolution integral is replaced by a convolution sum, i.e. the integral equation

$$s(t) = \int p(t-\tau)f(\tau)d\tau$$

is replaced with

$$s_i = \sum_j p_{i-j} f_j.$$

In turn, this convolution sum can be expressed in terms of the matrix equation (see Chapter 16)

$$\mathbf{s} = P\mathbf{f}$$

where P is a matrix whose elements are composed from those of $\mathbf{p}$. Note that some authors prefer to use the notation $f[i]$ and $p[i-j]$ for example to describe digital signals. In this work, the notation f_i, p_i etc. is used throughout to denote digital signals.

Part II of this book covers the computational methods associated with solving sets of linear algebraic equations and briefly considers the linear eigenvalue problem. In addition, a study is given that is concerned with the use of vector and matrix norms which provide the analysis required to establish conditions for numerical stability

and application of the least square methods for example which is used to construct solutions to certain classes of digital signal processors.

We shall start be first reviewing the algebra of matrices and matrix based equations.

6.1 Matrices and Matrix Algebra

A matrix (which shall usually be denoted by A) is a rectangular array of numbers

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

Here, A is said to be an $m \times n$ matrix (with m rows and n columns) - a matrix of 'order' mn. When $m = n$, the array is called a square matrix of order n. The element in the i^{th} row and j^{th} column of A is denoted by a_{ij} and we can write

$$A = (a_{ij}).$$

Two matrices A and B are equal if they are identical (they must be of the same type). In this case, each element of A is equal to the corresponding element of B. If all the elements of a matrix are zero then it is referred to as a null matrix.

6.1.1 Addition of Matrices

Two matrices A and B can be added together if and only if they are of the same type. B must have the same number of rows as A and the same number of columns as A. A and B are then said to be conformable for addition. Let $A = (a_{ij})$ and $B = (b_{ij})$, each being $m \times n$ matrices. Then

$$A + B = C$$

where C is also an $m \times n$ matrix. Then, the ij^{th} element in $C = (c_{ij})$ is given by

$$c_{ij} = a_{ij} + b_{ij}.$$

For example,

$$\begin{pmatrix} 1 & 0 & 3 \\ 5 & -1 & 2 \end{pmatrix} + \begin{pmatrix} 4 & -7 & 2 \\ 5 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 5 & -7 & 5 \\ 10 & 2 & 3 \end{pmatrix}.$$

6.1.2 Subtraction of Matrices

The matrix $-A$ is a matrix of the same type as A where the elements are all multiplied by -1, i.e. if

$$A = (a_{ij})$$

then

$$-A = (-a_{ij}).$$

If A and B are conformable for addition, then $A - B = A + (-B)$ and

$$C = A - B$$

or

$$c_{ij} = a_{ij} - b_{ij}.$$

6.1.3 Multiplication of a Matrix by a Scalar Quantity

If λ is a scalar and A is a matrix, then λA is a matrix of the same type as A and each element of λA is equal to λ multiplied by the corresponding element of A. Also,

$$\lambda(A + B) = \lambda A + \lambda B.$$

6.1.4 Multiplication of Two Matrices

Let A and B be two matrices. They are conformable for the product AB, written in that order, if the numbers of columns of A is equal to the number of rows in B. With this restriction, the product AB can be defined thus: Let A be an $m \times n$ matrix and let B be an $n \times p$ matrix. Then $C = AB$ is an $n \times p$ matrix whose elements c_{ij} are defined by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

For example, if

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 2 & -1 & 5 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2 & 1 & 4 \\ -7 & 5 & 2 \\ -1 & 3 & 1 \end{pmatrix},$$

then

$$C = AB = \begin{pmatrix} 1 & 3 & 4 \\ 2 & -1 & 5 \end{pmatrix} \begin{pmatrix} 2 & 1 & 4 \\ -7 & 5 & 2 \\ -1 & 3 & 1 \end{pmatrix} = \begin{pmatrix} -23 & 28 & 14 \\ 6 & 12 & 11 \end{pmatrix}.$$

In general, if A is an $m \times n$ matrix and B is an $n \times p$ matrix, then BA will only exist if $p = m$. Even so, $AB \neq BA$ in general. The product may be of matrices of different types. If A is $m \times n$ and B is $n \times m$ then AB is $m \times n$ and BA is $n \times m$. If the product AB can be formed, then B is said to be pre-multiplied by A and A is said to be post-multiplied by B. The matrices A and B for which

$$AB = BA$$

are said to commute under multiplication.

6.1.5 Further Results in Matrix Multiplication

The Distributive Law

Let A, B and C be three matrices where A, B are conformable for addition and A, C are conformable for the product AC. Then

$$(A+B)C = AC + BC.$$

If A, C are conformable for the product CA, then

$$C(A+B) = CA + CB.$$

The Associative Law

Suppose that A is a $m \times n$ matrix, B is a $n \times p$ matrix and C is a $p \times q$ matrix. The product AB can be formed $(m \times p)$. This can be post-multiplied by C (since C is $p \times q$) to give $(AB)C$ $(m \times p)$. Alternatively, BC can be formed $(n \times q)$ and pre-multiplied by A (since A is $m \times n$) to give $A(BC)$ $(m \times q)$. We can then prove that

$$(AB)C = A(BC).$$

Proof

$A = (a_{ij})$ $(m \times n)$, $B = (b_{ij})$ $(n \times p)$ and $C = (c_{ij})$ $(p \times q)$

Let $AB = D = (d_{ij})$ $(m \times p)$, then

$$d_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj.}$$

The ij^{th} element in $(AB)C$ is given by

$$= \sum_{\ell=1}^{p} d_{i\ell} c_{\ell j} = \sum_{\ell=1}^{p} \sum_{k=1}^{n} a_{ik} b_{k\ell} c_{\ell j}. \tag{6.1}$$

Now let $BC = E = (e_{ij})$ $(n \times q)$, where

$$e_{ij} = \sum_{\ell=1}^{p} b_{i\ell} c_{\ell j}.$$

The ij^{th} element in $A(BC)$ is then given by

$$= \sum_{k=1}^{n} a_{ik} e_{kj} = \sum_{k=1}^{n} \sum_{\ell=1}^{p} a_{ik} b_{k\ell} c_{\ell j}. \tag{6.2}$$

Equations (6.1) and (6.2) are equal, hence the result is proved.

In view of the proof given above, we can write

$$(AB)C = A(BC) = ABC,$$

i.e. the order must be preserved - brackets are not valid. If A is a square $n \times n$ matrix, then the products AA or AAA all exist and are denoted by A^2, A^3 etc.

6.1.6 The Unit Matrix

A square matrix in which all the elements are zero except those along the leading diagonal where they are all equal to 1 is called a unit matrix and is usually denoted by I. For example, the matrix

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is a unit matrix of order 3 and in general, the unit matrix is given by

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & 1 \end{pmatrix}$$

We denote the ij^{th} element in I by using the Hermitian or Kronecker delta symbol,

$$I = \delta_{ij} = \begin{cases} 0, & i \neq j; \\ 1, & i = j. \end{cases}$$

Note that $I = I^2 = I^3 = ...$ are all $n \times n$ matrices if I is $n \times n$ since if a_{ij} is the ij^{th} element in I^2, then

$$a_{ij} = \sum_{k=1}^{n} \delta_{ik}\delta_{kj} = \delta_{ij}.$$

For $IA = A$, I and A must be conformable for the product IA and for $BI = B$, B and I must be conformable for the product BI.

6.1.7 The Transpose of a Matrix

The transpose of A $(m \times n)$ is a matrix denoted by A^T $(n \times m)$. If $A = (a_{ij})$ then $A^T = (a_{ji})$. The elements of the i^{th} row of A^T are the elements of the i^{th} column of A.

Theorem If A and B are conformable for the product AB, then

$$(AB)^T = B^T A^T.$$

Proof Let A be an $m \times n$ matrix and B be an $n \times p$ matrix. Then

$$C = AB \qquad (m \times p)$$

where

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Hence, if $C^T = (z_{ij})$, then

$$z_{ij} = (c_{ij})^T = \sum_{k=1}^{n} a_{jk} b_{ki}.$$

Now

$$A^T = (a_{ji}), \quad B^T = (b_{ji})$$

and B^T and A^T are conformable for the product $B^T A^T = (w_{ij})$ where

$$w_{ij} = \sum_{k=1}^{n} b_{ki} a_{jk} = z_{ij}.$$

Hence,

$$(AB)^T = B^T A^T.$$

Theorem If $A_1, A_2, ..., A_n$ are conformable for the product $A_1 A_2 ... A_n$, then

$$(A_1 A_2 ... A_n)^T = A_n^T A_{n-1}^T ... A_1^T.$$

Proof (by induction)

$$(A_1 A_2 ... A_n)^T = [(A_1 A_2 ... A_{n-1}) A_n]^T = A_n^T (A_1 A_2 ... A_{n-1})^T$$
$$= A_n^T [(A_1 A_2 ... A_{n-2}) A_{n-1}]^T = A_n^T A_{n-1}^T (A_1 A_2 ... A_{n-2}) = ... = A_n^T A_{n-1}^T ... A_1^T.$$

6.1.8 The Inverse Matrix

Let A be a square $(n \times n)$ matrix, then it may be possible to construct a second matrix A^{-1} (also $n \times n$) such that

$$AA^{-1} = A^{-1}A = I.$$

Theorem The inverse of a non-singular matrix A is unique, i.e. if $AX = I$ then $X = A^{-1}$.

Proof If $AX = I$ then pre-multiplying by A^{-1} gives

$$A^{-1}AX = A^{-1}I = A^{-1}.$$

Now, $IX = A^{-1}$ so $X = A^{-1}$. Similarly, if $YA = I$ then $Y = A^{-1}$.

Theorem Suppose A, B are both $n \times n$ and non-singular, then $(AB)^{-1} = B^{-1}A^{-1}$.

Proof Let $C = AB$, then

$$B^{-1}A^{-1}C = B^{-1}A^{-1}AB = B^{-1}IB = B^{-1}B = I.$$

Hence

$$B^{-1}A^{-1} = C^{-1}.$$

6.1.9 Determinants

The determinant of a matrix A, which is denoted by $\mid A \mid$ or by $\det A$, arises naturally in the solution to a set of linear equations. They are essentially a notational convenience from which we can derive a number of useful properties. For example, consider the solution to the equations

$$a_{11}x_1 + a_{12}x_2 = b_1,$$

$$a_{21}x_1 + a_{22}x_2 = b_2.$$

Solving for x_1 and x_2 gives

$$x_1 = \frac{b_1a_{22} - b_2a_{12}}{a_{11}a_{22} - a_{21}a_{12}},$$

$$x_2 = \frac{b_2a_{11} - b_1a_{21}}{a_{11}a_{22} - a_{21}a_{12}}.$$

Note that this solution requires that

$$a_{11}a_{22} - a_{21}a_{12} \neq 0.$$

Suppose we now define the following 'second order determinants':

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21},$$

$$\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix} = b_1a_{22} - b_2a_{12},$$

$$\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix} = b_2a_{11} - b_1a_{21}.$$

We can then write

$$\frac{x_1}{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}} = \frac{x_2}{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}.$$

Now, consider and extension of the idea to the 3×3 system

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

The solution for x_1 is

$$x_1 = \frac{b_1a_{22}a_{33} - b_1a_{32}a_{23} + b_2a_{32}a_{13} - b_2a_{12}a_{33} + b_3a_{12}a_{23} - b_3a_{22}a_{13}}{a_{11}a_{22}a_{33} - a_{11}a_{32}a_{23} + a_{21}a_{32}a_{13} - a_{21}a_{12}a_{33} + a_{31}a_{12}a_{23} - a_{31}a_{22}a_{13}}.$$

We write the denominator in terms of the 'third order determinant'

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21} \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + a_{31} \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix},$$

and write the numerator in terms of the third order determinant

$$\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix} = b_1 \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - b_2 \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + b_3 \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}.$$

Similar analysis of the solutions for x_2 and x_3 leads to the result

$$\frac{x_1}{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}} = \frac{x_2}{\begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}} = \frac{x_3}{\begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}} = \frac{1}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}}.$$

We can extend this approach to larger systems of equations. This is the basis for a method of solution often referred to a 'Cramers rule'. In general,

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = a_{11} \mid M_{11} \mid -a_{21} \mid M_{21} \mid +a_{31} \mid M_{31} \mid -... + a_{n1} \mid M_{n1} \mid$$

where M_{i1} are the matrices obtained by deleting the first column and i^{th} row. M_{i1} is called the minor corresponding to the element a_{i1} and it is usual to write

$$\det A = a_{11}A_{11} + a_{21}A_{21} + a_{31}A_{31} + ... + a_{n1}A_{n1}$$

where A_{i1} is the cofactor of the element a_{i1} given by

$$A_{i1} = (-1)^{i+1} \mid M_{i1} \mid .$$

Note that a square $n \times n$ matrix A will be singular if and only if $\det A = 0$.

6.1.10 Properties of Determinants

The value of a determinant is unaltered by interchanging the elements of all corresponding rows and columns, i.e.

$$\mid A \mid = \mid A^T \mid .$$

For example,

$$\begin{vmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 3 & 4 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 2 & 2 & 3 \\ 3 & 1 & 4 \end{vmatrix} = 2.$$

The sign of a determinant is reversed by interchanging any two of its rows (or columns):

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = - \begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}.$$

The value of a determinant is zero if any two of its rows (or columns) are identical. Scalar multiplication of a row or a column by a constant λ gives $\lambda \mid A \mid$, i.e.

$$\begin{vmatrix} \lambda a_{11} & a_{12} & a_{13} \\ \lambda a_{21} & a_{22} & a_{23} \\ \lambda a_{31} & a_{32} & a_{33} \end{vmatrix} = \lambda \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}.$$

If any two rows (or columns) of a determinant have proportional elements, the value of the determinant is zero. For example,

$$\begin{vmatrix} 1 & -1 & 3 \\ 2 & -2 & 4 \\ 3 & -3 & 5 \end{vmatrix} = 0.$$

If the elements of any row (or column) are the sums or differences of two or more terms, the determinant may be written as the sum or difference of two or more determinants. For example,

$$\begin{vmatrix} a_{11}+x_1 & a_{12} & a_{13} \\ a_{21}+x_2 & a_{22} & a_{23} \\ a_{31}+x_3 & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} x_1 & a_{12} & a_{13} \\ x_2 & a_{22} & a_{23} \\ x_3 & a_{32} & a_{33} \end{vmatrix}.$$

The value of a determinant is unchanged if equal multiples of the elements of any row (or column) are added to the corresponding elements of any other row (or column). For example,

$$\begin{vmatrix} a_{11}+\lambda a_{12} & a_{12} & a_{13} \\ a_{21}+\lambda a_{22} & a_{22} & a_{23} \\ a_{31}+\lambda a_{32} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} \lambda a_{12} & a_{12} & a_{13} \\ \lambda a_{22} & a_{22} & a_{23} \\ \lambda a_{32} & a_{32} & a_{33} \end{vmatrix}$$

$$= \lambda \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}.$$

Finally, we note that

$$\mid AB \mid = \mid A \mid \mid B \mid .$$

6.2 Systems of Linear Algebraic Equations

There are two types of systems:

(i) Inhomogeneous equations.

(ii) Homogeneous equations.

Inhomogeneous equations are a set of equations given by

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1,$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2,$$

$$\vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n.$$

where a_{ij} and b_i are known values which may be positive, negative, zero or complex. The basic problem is: given a_{ij} and b_i find $x_1, x_2, ..., x_n$ - the 'solution vector', which we can denote by

$$\mathbf{x} = (x_1, x_2, ..., x_n)^T.$$

These equations can be cast as

$$A\mathbf{x} = \mathbf{b}$$

where $\mathbf{b} \neq \mathbf{0}$. Homogeneous equations are a set of equations in which $b_i = 0 \forall i$ or $\mathbf{b} = \mathbf{0}$.

6.2.1 Formal Methods of Solution

Formal methods of solution involve writing the system of equations in matrix form:

$$A\mathbf{x} = \mathbf{b}$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv (x_1, x_2, ..., x_n)^T$$

and

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \equiv (b_1, b_2, ..., b_n)^T.$$

The formal solution to $A\mathbf{x} = \mathbf{b}$ is based on constructing an inverse matrix A^{-1} such that

$$A^{-1}A = I$$

where I is the unit or identity matrix given by

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}.$$

Then,

$$A^{-1}A\mathbf{x} = A^{-1}\mathbf{b}$$

and

$$I\mathbf{x} = A^{-1}\mathbf{b}.$$

Now,

$$I\mathbf{x} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{x}.$$

and therefore

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

In 'formal' methods, the problem is reduced to finding the inverse or reciprocal matrix A^{-1}. This is given by

$$A^{-1} = \frac{\text{adj}A}{\det A}$$

where adj$A$ is the adjoint of matrix A and $\det A$ is the determinant of matrix A. Computation of A^{-1} requires that $\det A \neq 0$, i.e. A is non-singular.

6.2.2 Evaluation of adj$A$

adj$A$ is the transpose of the matrix whose elements are the cofactors of A. The cofactors A_{ij} of a_{ij} are given by

$$A_{ij} = (-1)^{(i+j)} M_{ij}$$

where M_{ij} are the minors of the elements a_{ij}. The minors are obtained by deleting the row and column in which a_{ij} occurs and computing the determinant of the remaining elements. For example, suppose we are required to solve the following set of linear equations by computing the inverse matrix:

$$4x_1 - 3x_2 + x_3 = 11,$$

$$2x_1 + x_2 - 4x_3 = -1,$$

$$x_1 + 2x_2 - 2x_3 = 1.$$

In matrix form, $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 4 & -3 & 1 \\ 2 & 1 & -4 \\ 1 & 2 & -2 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 11 \\ -1 \\ 1 \end{pmatrix}.$$

Now, the formal solution is $\mathbf{x} = A^{-1}\mathbf{b}$ where $A^{-1} = \text{adj}A/\det A$. The computation of $\det A$ is as follows:

$$\begin{vmatrix} 4 & -3 & 1 \\ 2 & 1 & -4 \\ 1 & 2 & -2 \end{vmatrix} = 4 \begin{vmatrix} 1 & -4 \\ 2 & -2 \end{vmatrix} - (-3) \begin{vmatrix} 2 & -4 \\ 1 & -2 \end{vmatrix} + 1 \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}$$

$$= 4 \times 6 - (-3) \times 0 + 1 \times 3 = 27.$$

The computation of the cofactors is as follows:

$$A_{11} = (-1)^{1+1} \begin{vmatrix} 1 & -4 \\ 2 & -2 \end{vmatrix} = 6,$$

$$A_{12} = (-1)^{1+2} \begin{vmatrix} 2 & -4 \\ 1 & -2 \end{vmatrix} = 0,$$

$$A_{13} = (-1)^{1+3} \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix} = 3,$$

$$A_{21} = (-1)^{2+1} \begin{vmatrix} -3 & 1 \\ 2 & -2 \end{vmatrix} = -4,$$

$$A_{22} = (-1)^{2+2} \begin{vmatrix} 4 & 1 \\ 1 & -2 \end{vmatrix} = -9,$$

$$A_{23} = (-1)^{2+3} \begin{vmatrix} 4 & -3 \\ 1 & 2 \end{vmatrix} = -11,$$

$$A_{31} = (-1)^{3+1} \begin{vmatrix} -3 & 1 \\ 1 & -4 \end{vmatrix} = 11,$$

$$A_{32} = (-1)^{3+2} \begin{vmatrix} 4 & 1 \\ 2 & -4 \end{vmatrix} = 18,$$

$$A_{33} = (-1)^{3+3} \begin{vmatrix} 4 & -3 \\ 2 & 1 \end{vmatrix} = 10.$$

The matrix of cofactors is therefore given by

$$\begin{pmatrix} 6 & 0 & 3 \\ -4 & -9 & -11 \\ 11 & 18 & 10 \end{pmatrix}$$

and thus,

$$\mathrm{adj}A = \begin{pmatrix} 6 & -4 & 11 \\ 0 & -9 & 18 \\ 3 & -11 & 10 \end{pmatrix}$$

and

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{1}{27} \begin{pmatrix} 6 & -4 & 11 \\ 0 & -9 & 18 \\ 3 & -11 & 10 \end{pmatrix} \begin{pmatrix} 11 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}.$$

6.2.3 Cramers Rule

The formal solution to the equation $A\mathbf{x} = \mathbf{b}$ is

$$\mathbf{x} = A^{-1}\mathbf{b} = \frac{\mathrm{adj}A}{\det A}\mathbf{b} = \frac{1}{\mid A \mid} \begin{pmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \dots & A_{nn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

where A_{ij} are the cofactors of the matrix A, the solutions being given by

$$x_1 = \frac{1}{\mid A \mid}(b_1A_{11} + b_2A_{21} + ... + b_nA_{n1}),$$

$$x_2 = \frac{1}{\mid A \mid}(b_1A_{12} + b_2A_{22} + ... + b_nA_{n2}),$$

$$\vdots$$

$$x_n = \frac{1}{\mid A \mid}(b_1A_{1n} + b_2A_{2n} + ... + b_nA_{nn}).$$

Cramers rule comes from observing that

$$b_1A_{11} + b_2A_{21} + ... + b_nA_{n1} = \begin{pmatrix} b_1 & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_n & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

$$b_1A_{12} + b_2A_{22} + ...{}_+ b_nA_{n2} = \begin{pmatrix} a_{11} & b_1 & \dots & a_{1n} \\ a_{12} & b_2 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & b_n & \dots & a_{nn} \end{pmatrix}$$

and so on. Hence,

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}}, \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ a_{12} & b_2 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}}$$

with similar results for the elements $x_3, x_4, ..., x_n$. For example, suppose we want to solve the set of equations

$$4x_1 - 3x_2 + x_3 = 11,$$

$$2x_1 + x_2 - 4x_3 = -1,$$

$$x_1 + 2x_2 - 2x_3 = 1.$$

We first compute the determinant of the system

$$\mid A \mid = \begin{vmatrix} 4 & -3 & 1 \\ 2 & 1 & -4 \\ 1 & 2 & -2 \end{vmatrix} = 27.$$

Then,

$$x_1 = \frac{1}{27}\begin{pmatrix} 11 & -3 & 1 \\ -1 & 1 & -4 \\ 1 & 2 & -2 \end{pmatrix} = \frac{1}{27}(66 + 18 - 3) = 3,$$

$$x_2 = \frac{1}{27}\begin{pmatrix} 4 & 11 & 1 \\ 2 & -1 & -4 \\ 1 & 1 & -2 \end{pmatrix} = \frac{1}{27}(24 + 0 + 3) = 1,$$

$$x_3 = \frac{1}{27}\begin{pmatrix} 4 & -3 & 11 \\ 2 & 1 & -1 \\ 1 & 2 & 1 \end{pmatrix} = \frac{1}{27}(12 + 9 + 33) = 2.$$

6.3 Linear Systems

6.3.1 Inhomogeneous systems

Inhomogeneous systems are characterized by $A\mathbf{x} = \mathbf{b}$ for which the formal solution is

$$\mathbf{x} = A^{-1}\mathbf{b} = \frac{\text{adj}A}{\det A}\mathbf{b}.$$

This solution exists provided A^{-1} exists, i.e. provided $\mid A \mid \neq 0$. The system $A\mathbf{x} = \mathbf{b}$ is consistent if at least one solution exists. If $\mid A \mid = 0$, then no formal solution of the equations exists; the equations are inconsistent or incompatible. For example, the equations

$$2x_1 - 3x_2 = 4,$$

$$2x_1 - 3x_2 = 6,$$

have no solution since $\mid A \mid = 0$. The geometrical interpretation of this result is that there are two parallel lines and such lines do not intersect. If both $\det A$ and $(\text{adj}A)\mathbf{b}$ are zero, then an infinity of solutions exist. In this case, the equations are linearly dependent. For example, consider the equations

$$2x_1 - 3x_2 = 4,$$

$$-4x_1 + 6x_2 = -8.$$

These are the same equations and therefore, geometrically describe the same line. Hence, the solution to the system is any point along the line.

6.3.2 Homogeneous Systems

Homogeneous systems are of the type

$$A\mathbf{x} = \mathbf{0}$$

whose formal solution is $\mathbf{x} = A^{-1}\mathbf{0}$. If $\mid A \mid \neq 0$ and A^{-1} exists, then $\mathbf{x} = \mathbf{0}$ which is the trivial solution. If $\mid A \mid = 0$, then an infinity of non-trivial solutions exist. For example, consider the equations

$$2x_1 - 3x_2 = 0,$$

$$4x_1 - 6x_2 = 0.$$

These equations are linearly dependent and represent the same line on which an infinity of solutions exist.

6.3.3 Ill-conditioned Systems

If the solution to $A\mathbf{x} = \mathbf{b}$ is very sensitive to relatively small changes in a_{ij} and b_i, then the system is ill-conditioned. It is usually important to establish whether a system of linear equations is ill-conditioned in cases where a_{ij} and b_i are derived from data with experimental error or noise (inevitable in all but a few cases). To illustrate the problem, consider the system of equations

$$3x_1 + 1.52x_2 = 1,$$

$$2x_1 + 1.02x_2 = 1.$$

Now

$$| A |= \begin{pmatrix} 3 & 1.52 \\ 2 & 1.02 \end{pmatrix} = 3.06 - 3.04 = 0.02$$

and using Cramers rule

$$x_1 = \frac{1}{0.02}\begin{pmatrix} 1 & 1.52 \\ 1 & 1.02 \end{pmatrix} = -\frac{0.5}{0.02} = -25,$$

$$x_2 = \frac{1}{0.02}\begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} = \frac{1}{0.02} = 50.$$

Now, suppose we change a_{22} from 1.02 to 1.03 which represents a 1% change in the value of a_{22}. Then

$$| A |= \begin{pmatrix} 3 & 1.52 \\ 2 & 1.03 \end{pmatrix} = 3.09 - 3.04 = 0.05$$

and using Cramers rule again

$$x_1 = \frac{1}{0.05}\begin{pmatrix} 1 & 1.52 \\ 1 & 1.03 \end{pmatrix} = -\frac{0.49}{0.05} = -9.8,$$

$$x_2 = \frac{1}{0.05}\begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} = \frac{1}{0.05} = 20.$$

This simple example demonstrates that a 1% change in the value of this coefficient leads to a 200-300% change in the solution and no reliance can be placed on the solution of such a system in practice.

6.3.4 Under-determined Systems

An under-determined system is one where there are more unknowns than equations. For example, consider the system $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 1 & -3 & 0 & 5 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 4 \\ -7 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

In this case

$$x_1 - 3x_2 + 5x_4 = 4,$$

$$x_3 + 2x_4 = -7,$$

$$x_5 = 1,$$

or

$$x_1 = 3x_2 - 5x_4 + 4,$$

$$x_3 = -2x_4 - 7,$$

$$x_5 = 1.$$

Hence

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 3a - 5b + 4 \\ a \\ -2b - 7 \\ b \\ 1 \end{pmatrix}$$

for any a and b.

6.3.5 Over-determined Systems

Over-determined systems take the form $A\mathbf{x} = \mathbf{b}$ where A has more rows than columns, i.e. there are more equations than unknowns. Formally, such systems have no solution. However, an estimate for $\mathbf{x}$ can be obtained based on the application a specific criterion. A widely exploited example of such a criterion is the least squares method. Here, given an over determined system $A\mathbf{x} = \mathbf{b}$ we find $\mathbf{x}$ such that

$$\|A\mathbf{x} - \mathbf{b}\|_2^2$$

is a minimum where

$$\|x\|_2 = \left(\sum_{i=1}^{n} | x_i |^2 \right)^{\frac{1}{2}}$$

which is an example of the application of the Euclidean norm. Vector and matrix norms are discussed further in Chapter 8.

6.4 Summary of Important Results

Matrices

$$C = A + B = C \quad \text{where} \quad c_{ij} = a_{ij} + b_{ij},$$

$$C = AB \quad \text{where} \quad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

$$(AB)^T = A^T B^T, \quad (AB)^{-1} = B^{-1} A^{-1}, \quad A^{-1} A = I$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Determinants

$$\det A \equiv \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = a_{11}A_{11} + a_{21}A_{21} + a_{31}A_{31} + ... + a_{n1}A_{n1}$$

where A_{i1} is the cofactor of the element a_{i1} given by

$$A_{i1} = (-1)^{i+1} \mid M_{i1} \mid$$

and M_{i1} are the minors obtained by deleting the first column and i^{th} row of A.

Properties

$$\mid A \mid = \mid A^T \mid, \quad \mid AB \mid = \mid A \mid \mid B \mid .$$

Adjoint

$\text{adj}A$ is the transpose of the matrix whose elements are the cofactors of A.

Inverse Matrix

$$A^{-1} = \frac{\text{adj}A}{\det A}$$

Formal solution

If $A\mathbf{x} = \mathbf{b}$, then

$$\mathbf{x} = A^{-1}\mathbf{b}$$

where

$$x_1 = \frac{1}{\mid A \mid}(b_1 A_{11} + b_2 A_{21} + ... + b_n A_{n1}),$$

$$x_2 = \frac{1}{\mid A \mid}(b_1 A_{12} + b_2 A_{22} + ... + b_n A_{n2}),$$

$$\vdots$$

$$x_n = \frac{1}{\mid A \mid}(b_1 A_{1n} + b_2 A_{2n} + ... + b_n A_{nn}).$$

Cramers Rule

Solution to $A\mathbf{x} = \mathbf{b}$ is

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}}, \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ a_{12} & b_2 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}}, ...,$$

$$x_n = \frac{\begin{vmatrix} a_{11} & a_{12} & \dots & b_1 \\ a_{12} & a_{22} & \dots & b_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & b_n \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}}.$$

Inhomogeneous Systems

$$A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} = A^{-1}\mathbf{b} = \frac{\text{adj}A}{\det A}\mathbf{b}.$$

If $\det A \neq 0$, then a solution exists.

If at least one solution exists, then the system $A\mathbf{x} = \mathbf{b}$ is consistent.

If $\det A = 0$, then no formal solution exists.

If $\det A$ and $\text{adj}A = 0$, then an infinity of solutions exist.

Homogeneous Systems

$$A\mathbf{x} = \mathbf{0}, \quad \mathbf{x} = A^{-1}\mathbf{0} = \frac{\text{adj}A}{\det A}\mathbf{0}.$$

If $\det \neq 0$, then $\mathbf{x} = \mathbf{0}$.

If $\det = 0$, then an infinity of solutions exist.

Ill-conditioned Systems

$$A\mathbf{x} = \mathbf{b}$$

where, if $\mathbf{b}+\delta\mathbf{b}$ and/or $A+\delta A$ yields $\mathbf{x}+\delta\mathbf{x}$ and $\delta\mathbf{x}$ is large for relatively small values of $\delta\mathbf{b}$ and/or δA.

Under-determined Systems

$$A\mathbf{x} = \mathbf{b}$$

where there are more unknowns that equations. The solution is parameter dependent where the number of parameters is determined by the number of unknowns that exceeds the number of equations.

Over-determined Systems

$$A\mathbf{x} = \mathbf{b}$$

where there are more equations than unknowns. Formally, no solution exists but criteria based solutions (e.g. least squares criterion) can be applied.

6.5 Further Reading

- Ayres F, *Matrices*, Schaum, 1962.
- Stephenson G, *An Introduction to Matrices, Sets and Groups*, Longman, 1974.
- Broyden C G, *Basic Matrices*, Macmillan, 1975.
- Lipschutz S, *Linear Algebra*, Schaum (McGraw-Hill), 1981.
- Perry W L, *Elementary Linear Algebra*, McGraw-Hill, 1988.
- Anton H and Rorres C, *Elementary Linear Algebra*, Wiley, 1994

6.6 Problems

6.1 (i) Show that the equations

$$-2x + y + z = 1,$$

$$x - 2y + z = 2,$$

$$x + y - 2z = -3,$$

are linearly dependent and find the infinity of solutions which satisfies this system.

(ii) Find the value of k for which the equations

$$x + 5y + 3z = 0,$$

$$5x + y - kz = 0,$$
$$x + 2y + kz = 0,$$

have non-trivial solutions. Show that the equations are linearly dependent and find the infinity of solutions.

6.2 (i) Find the value(s) of λ for which the equations

$$x_1 + 2x_2 + x_3 = \lambda x_1,$$
$$2x_1 + x_2 + x_3 = \lambda x_2,$$
$$x_1 + x_2 + 2x_3 = \lambda x_3,$$

have non-trivial solutions and obtain the general solution for one such value of λ.

(ii) Find the most general form of solution of the equations

$$x_1 - x_2 + 2x_3 - x_4 = 1,$$
$$2x_1 - x_2 + 3x_3 - 4x_4 = 2,$$
$$-x_1 + 3x_2 - x_3 - x_4 = -1.$$

(iii) Show that the equations

$$x_1 + 2x_2 - 3x_3 = 0,$$
$$2x_1 - x_2 + 2x_3 = 0,$$
$$x_1 + 7x_2 - 11x_3 = 0,$$

have a non-trivial solution and find a solution which satisfies

$$x_1^2 + x_2^2 + x_3^2 = 1.$$

6.3 Investigate the ill-conditioned nature of the equations

$$x + 1.52y = 1,$$
$$2x + (3.05 + \delta)y = 1,$$

for $\delta = -0.02, \quad -0.01, \quad 0.0, \quad 0.01$ and 0.02 using Cramers rule.

6.4 Using the least squares method, show that the 'best value' for x_1 and x_2 in the over-determined system

$$a_{11}x_1 + a_{12}x_2 = b_1,$$
$$a_{21}x_1 + a_{22}x_2 = b_2,$$
$$a_{31}x_1 + a_{32}x_2 = b_3,$$

is given by the solution to the following equations

$$\sum_{i=1}^{3} a_{i1}(a_{i1}x_1 + a_{i2}x_2 - b_i) = 0,$$

$$\sum_{i=1}^{3} a_{i2}(a_{i1}x_1 + a_{i2}x_2 - b_i) = 0.$$

Find the 'best value' of x and y for the over-determined systems

$$2x + 3y = 8,$$

$$3x - y = 1,$$

$$x + y = 4,$$

and

$$x - y = 2,$$

$$x + y = 4,$$

$$2x + y = 8.$$

Compute the associated root mean square error in each case.

Chapter 7

Direct Methods of Solution

Formal methods of solving linear equations such as Cramers rule are useful as hand calculations when only a few equations are involved. With larger systems of equations, formal methods are not appropriate because they require an excessive number of operations. For n equations they typically require $n!$ cross-products (e.g. to solve a set of 10 linear equations requires $\sim 10^6$ cross products). Hence, such methods are very slow. In addition to the number of operations required, errors can 'propagate' through the calculation (due primarily to arithmetic 'round-off errors') growing at each stage. A large number of computations can therefore lead to a final result which is inaccurate or worse, completely wrong, especially if the equations are ill-conditioned.

Computationally practical methods of solution such as those of Gauss, Jordan, Crout and others as discussed in this chapter, typically require $\sim n^3$ multiplications and additions to solve systems consisting of n linear equations. Such methods are known as direct methods of solution. Indirect or iterative methods (techniques attributed to Jacobi, Gauss, Seidel and others) involve a different approach to the problem in which the number of computations depends mainly on the number of iterations required to achieve an accurate solution. In this case, the principal aim is to obtain a solution which requires as few iterations as possible.

As a general rule of thumb, direct methods are preferable when the characteristic matrix of the system is dense, whereas indirect methods are often implemented when the matrix is sparse. In turn, there is a close synergy between the approaches taken to solving sets of linear systems of equations using iterative methods and finite difference approaches to solving elliptic equations for example such as the Laplace ($\nabla^2 u = 0$) and Poisson ($\nabla^2 u = f$) equations. However, direct methods of solution are usually more efficient, except in cases where: (i) the equations to be solved are of a form especially suited to an iterative technique; (ii) a good starting approximation is available. Direct and indirect techniques are sometimes combined. For example, the initial solution obtained by a direct method of a system of ill-conditioned equations may be improved upon by using an iterative method. This is known as iterative improvement. Further, in addition to both direct and indirect methods, there are semi-iterative approaches such as the 'Conjugate Gradient Method'.

The solution of a system of linear equations by a computer is almost invariably

subject to rounding errors. In most practical cases, the coefficients of the equations are non-integer, but even if this is not so, the methods to be described give rise to non-integer numbers during the computation. Such numbers, unless they can be expressed as fractions whose denominators are powers of 2, have non-terminating representations in the binary arithmetic used by a computer. Each number must be stored in a finite space, and hence only the most significant figures can be retained. It is important to minimize the effects of the rounding errors which arise in this way, so that the highest possible accuracy is achieved in the solution. In other words, the solution to $A\mathbf{x} = \mathbf{b}$ on a digital computer is limited by calculations involving a finite number of decimal places. Errors induced by limited decimal place calculation can lead to seriously inaccurate results. To illustrate this, consider the solution to the system of equations (using Cramers rule)

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 = 1,$$

$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 = 0,$$
$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = 0,$$

which has the exact solution $x_1 = 9, \quad x_2 = -36, \quad x_3 = 30$ and compare this solution with that of the system

$$1.00x_1 + 0.50x_2 + 0.33x_3 = 1.00,$$

$$0.50x_1 + 0.33x_2 + 0.25x_3 = 0.0, 0$$
$$0.33x_1 + 0.25x_2 + 0.20x_3 = 0.00,$$

using 2 decimal place computations which gives $x_1 = 7.00, \quad x_2 = -24.00, \quad x_3 = 17.00$, i.e. significantly different results to the exact solution.

7.1 Gauss' Method

The solution of a set of linear equations is unaffected if any of the following elementary row operations is performed: (i) altering the order of the equations; (ii) multiplying any equation by a non-zero constant; (iii) addition or subtraction of any two equations.

Also known as Gauss elimination or Gauss reduction, Gauss' method is based on using points (i)-(iii) above to implement the following basic idea: Process the system $A\mathbf{x} = \mathbf{b}$ in such a way that we can write $U\mathbf{x} = \mathbf{h}$ where U is an upper triangular matrix of the form

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

and $\mathbf{h} = (h_1, h_2, ..., h_n)^T$ is a new column vector formed from the data $\mathbf{b}$ as a result of applying this process.

Once this process has been completed, the solution can be obtained by back-substitution, i.e.

$$u_{nn}x_n = h_n, \quad x_n = \frac{h_n}{u_{nn}};$$

$$u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = h_{n-1}, \quad x_{n-1} = \frac{h_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}$$

and so on. Hence,

Triangularization and Back-substitution ≡ Gauss' method.

For example, suppose we want to solve the set of equations

$$\begin{array}{lcl} 2x_1 - x_2 + 3x_3 & = & 2, \\ 4x_1 + x_2 + x_3 & = & -1, \\ -2x_1 + 3x_2 + 2x_3 & = & 3. \end{array}$$

Since the problem is completely defined by the matrix of coefficients of the variables and the column vector of constants, we may suppress all unnecessary symbols and operate upon an 'augmented matrix'. The process of triangularization can be performed using this augmented matrix which is essentially a type of 'counting board' which is just a useful way of displaying the information. Thus, with R_i referring to the i^{th} row, the augmented matrix for this case is given by

$$\left(\begin{array}{ccc|c} 2 & -1 & 3 & 2 \\ 4 & 1 & 1 & -1 \\ -2 & 3 & 2 & 3 \end{array}\right)$$

and the process of triangularization is as follows:

$$\begin{array}{c} R_2 - 2R_1 \\ R_3 + R_1 \end{array} \left(\begin{array}{ccc|c} 2 & -1 & 3 & 2 \\ 0 & 3 & -5 & -5 \\ 0 & 2 & 5 & 5 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_3 - \frac{2}{3}R_2 \end{array} \left(\begin{array}{ccc|c} 2 & -1 & 3 & 2 \\ 0 & 6 & -10 & -10 \\ 0 & 0 & \frac{25}{3} & \frac{25}{3} \end{array}\right).$$

Back-substituting:

$$\begin{array}{ll} 25x_3 = 25, & x_3 = 1; \\ 6x_2 - 10 = -10, & x_2 = 0; \\ 2x_1 - 0 + 3 = 2, & x_1 = -\frac{1}{2}. \end{array}$$

The denominator occurring in term $\frac{2}{3}R_2$ and terms like it is called a 'pivot'. Note that we use | to distinguish between those elements of the augmented matrix that are part of the characteristic matrix A and those that belong to the data $\mathbf{b}$. The notation $\longrightarrow$ is used to denote a subsequent processing stage.

7.1.1 Generalization of the Idea

Consider the general 3×3 system:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1,$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2, \tag{7.1.1}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \tag{7.1.2}$$

We can now eliminate x_1 from equation (7.1.1) using $R_2 - R_1\frac{a_{21}}{a_{11}}$ and then eliminate x_1 from equation (7.1.2) with the process $R_3 - R_1\frac{a_{31}}{a_{11}}$. The system then becomes

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1, \\ a'_{22}x_2 + a'_{23}x_3 &= b'_2, \\ a'_{32}x_2 + a'_{33}x_3 &= b'_3, \end{aligned} \tag{7.1.3}$$

where

$$\begin{aligned} a'_{22} = a_{22} - \frac{a_{21}a_{12}}{a_{11}}, \quad a'_{23} = a_{23} - \frac{a_{21}a_{13}}{a_{11}}, \\ a'_{32} = a_{32} - \frac{a_{12}a_{31}}{a_{11}}, \quad a'_{33} = a_{33} - \frac{a_{13}a_{31}}{a_{11}}, \\ b'_2 = b_2 - b_1\frac{a_{21}}{a_{11}}, \quad b'_3 = b_3 - b_1\frac{a_{31}}{a_{11}}. \end{aligned}$$

Eliminating x_2 from equation (7.1.3) using the process $R_3 - R_2\frac{a'_{32}}{a'_{22}}$, the system becomes

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1, \\ a'_{22}x_2 + a'_{23}x_3 &= b'_2, \\ a''_{33}x_3 &= b''_3, \end{aligned}$$

where

$$a''_{33} = a'_{33} - \frac{a'_{32}a'_{23}}{a'_{22}}, \quad b''_3 = b'_3 - \frac{b'_2a'_{32}}{a'_{22}}.$$

In this process of triangularization, the pivots a_{11} and a'_{22} must clearly be non-zero. Back-substituting

$$\begin{aligned} x_3 &= \frac{b''_3}{a''_{33}} \\ x_2 &= \frac{b'_2 - a'_{23}x_3}{a'_{22}} \\ x_1 &= \frac{b_1 - a_{13}x_3 - a_{12}x_2}{a_{11}}. \end{aligned}$$

7.1.2 Conversion to Pseudo-Code

Based on the algebraic form of multipliers obtained for the 3×3 system above and generalizing to a $n \times n$ system, the essential lines of pseudo code for Gaussian elimination are:

```
a(i,j)=a(i,j)-a(i,k)*a(k,j)/a(k,k)
```

and

```
b(i)=b(i)-b(k)*a(i,k)/a(k,k)
```

where k runs from 1 to n-1, i runs from k+1 to n and j runs from k to n. Note that the term

```
a(i,k)/a(k,k)
```

is common to both lines of code given above. Therefore, we can let

```
c=a(i,k)/a(k,k)
```

so that the pseudo code for Gaussian elimination becomes

```
for k=1 to n-1, do:

    for i=k+1 to n,  do:
        c=a(i,k)/a(k,k)

        for j=k to n, do:
            a(i,j)=a(i,j)-a(k,j)*c
        enddo

    b(i)=b(i)-b(k)*c
    enddo

 enddo
```

The pseudo code for back-substitution is

```
x(n)=b(n)/a(n,n)
for k=n-1 to 1 in steps of -1, do:
    sum=b(k)

        for j=k+1 to n, do:
            sum=sum-a(k,j)*x(j)
        enddo

x(k)=sum/a(k,k)
enddo
```

Observe, that the way in which this code is developed is based on establishing a systematic processing path using a small set of equations and then generalizing the result for an arbitrary size. This is typical of the process of developing software solutions for computational linear algebra.

7.1.3 Pivots and Pivoting

Zero pivots in the Gaussian elimination process must clearly be avoided. However, in practice, small pivots can cause problems. To illustrate this, let us consider a system given by

$$-0.001x_1 + x_2 = 1,$$

$$x_1 + x_2 = 2.$$

Applying Gaussian elimination working to 3 decimal places only gives

$$-0.001x_1 + x_2 = 1,$$

$$1001x_2 = 1002,$$

from which we derive the results $x_2 = 1.000$, $x_1 = 0$ which is not a solution since $0 + 1 \neq 2$. However, if we interchange the rows and work with

$$x_1 + x_2 = 2,$$

$$-0.001x_1 + x_2 = 1,$$

Gaussian elimination gives

$$x_1 + x_2 = 2,$$

$$1.001x_2 = 1.002$$

and hence, $x_2 = 1.000$ and $x_1 = 1$ which is the 3 decimal place hypothetical 'machine solution'.

7.1.4 Pivotal Strategies

There are two pivotal strategies available: (i) partial pivoting; (ii) full pivoting.

Partial Pivoting

Partial pivoting is based on: (i) searching the relevant column of the reduced matrix for the largest entry at each stage of the elimination process; (ii) interchange the relevant rows.

Example Consider the equations

$$\begin{array}{lcl} 10x_2 - 3x_3 & = & -5, \\ 2x_1 + x_2 - x_3 & = & 7, \\ 4x_1 + 10x_2 - 2x_3 & = & 10, \end{array}$$

with augmented matrix

$$\left(\begin{array}{ccc|c} 0 & 10 & -3 & -5 \\ 2 & 1 & -1 & 7 \\ 4 & 10 & -2 & 10 \end{array}\right).$$

Interchange R_3 and R_1, then

$$\left(\begin{array}{ccc|c} 4 & 10 & -2 & 10 \\ 2 & 1 & -1 & 7 \\ 0 & 10 & -3 & -5 \end{array}\right)$$

and

$$\begin{array}{c} R_1 \\ R_2 - \frac{2}{4}R_1 \\ R_3 \end{array} \left(\begin{array}{ccc|c} 4 & 10 & -2 & 10 \\ 0 & -4 & 0 & 2 \\ 0 & 10 & -3 & -5 \end{array}\right).$$

Interchange R_3 and R_2, then

$$\left(\begin{array}{ccc|c} 4 & 10 & -2 & 10 \\ 0 & 10 & -3 & -5 \\ 0 & -4 & 0 & 2 \end{array}\right)$$

and

$$\begin{matrix} R_1 \\ R_2 \\ R_3 - \frac{(-4)}{10}R_2 \end{matrix} \begin{pmatrix} 4 & 10 & -2 & | & 10 \\ 0 & 10 & -3 & | & -5 \\ 0 & 0 & -\frac{6}{5} & | & 0 \end{pmatrix}.$$

Back-substituting, we get

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 15/4 \\ -1/2 \\ 0 \end{pmatrix}.$$

The extension to partial pivoting is 'full' or 'complete pivoting'.

Full Pivoting

In this case, we search the matrix of coefficients for the largest entry and interchange relevant rows.

Example Solve $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 6 & 4 \\ 3 & 7 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 14 \\ 3 \\ -8 \end{pmatrix}.$$

Here, the augmented matrix is

$$\begin{pmatrix} 2 & 1 & 3 & | & 14 \\ 1 & 6 & 4 & | & 3 \\ 3 & 7 & 2 & | & -8 \end{pmatrix}$$

Interchanging R_1 and R_3,

$$\begin{pmatrix} 3 & 7 & 2 & | & -8 \\ 1 & 6 & 4 & | & 3 \\ 2 & 1 & 3 & | & 14 \end{pmatrix}.$$

Then

$$\begin{matrix} R_1 \\ R_2 - \frac{6}{7}R_1 \\ R_3 - \frac{1}{7}R_1 \end{matrix} \begin{pmatrix} 3 & 7 & 2 & | & -8 \\ -11/7 & 0 & 16/7 & | & 69/7 \\ 11/7 & 0 & 19/7 & | & 106/7 \end{pmatrix}.$$

Interchanging R_3 and R_2,

$$\begin{pmatrix} 3 & 7 & 2 & | & -8 \\ 11/7 & 0 & 19/7 & | & 106/7 \\ -11/7 & 0 & 16/7 & | & 69/7 \end{pmatrix}$$

and

$$\begin{matrix} R_1 \\ R_2 \\ R_3 - \frac{16}{19}R_2 \end{matrix} \begin{pmatrix} 3 & 7 & 2 & | & -8 \\ 11/7 & 0 & 19/7 & | & 106/7 \\ -385/133 & 0 & 0 & | & -385/133 \end{pmatrix}.$$

Back-substituting:

$$x_1 = 1,$$

$$\frac{11}{7} + \frac{19}{7}x_3 = \frac{106}{7} \longrightarrow x_3 = 5,$$
$$3 + 7x_2 + 10 = -8 \longrightarrow x_2 = -3.$$

In practice, application of full pivoting requires more computing time because the matrix must be searched for the largest entry. Also full pivoting requires the output data to be reordered.

7.2 Jordan's Method

As with Gauss' method, Jordan's method relies on a process of reduction. However, instead of processing the augmented matrix to become triangular, processing is applied to reduce the matrix to a diagonal form. Thus, the basic idea is: given $A\mathbf{x} = \mathbf{b}$, via a process of elimination, reduce this system to the form $I\mathbf{x} = \mathbf{h}$ where

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}, \quad \mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{pmatrix}.$$

The solution is then trivial, since

$$x_1 = h_1,$$
$$x_2 = h_2,$$
$$\vdots$$
$$x_n = h_n.$$

Note that in this case, no back-substitution is required.

Example of Jordan's Method Solve

$$\begin{aligned} x_1 - 2x_2 + x_3 &= 7, \\ 2x_1 - 5x_2 + 2x_3 &= 6, \\ 3x_1 + 2x_2 - x_3 &= 1, \end{aligned}$$

which has the augmented matrix

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 7 \\ 2 & -5 & 2 & 6 \\ 3 & 2 & -1 & 1 \end{array}\right).$$

Then,

$$\begin{matrix} \\ R_2 - 2R_1 \\ R_3 - 3R_1 \end{matrix} \left(\begin{array}{ccc|c} 1 & -2 & 1 & 7 \\ 0 & -1 & 0 & -8 \\ 0 & 8 & -4 & -20 \end{array}\right) \longrightarrow \begin{matrix} \\ -R_2 \\ \\ \end{matrix} \left(\begin{array}{ccc|c} 1 & -2 & 1 & 7 \\ 0 & 1 & 0 & 8 \\ 0 & 8 & -4 & -20 \end{array}\right) \longrightarrow$$

$$\begin{matrix} R_1 + 2R_2 \\ \\ R_3 - 8R_2 \end{matrix} \left(\begin{array}{ccc|c} 1 & 0 & 1 & 23 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & -4 & -84 \end{array}\right) \longrightarrow \begin{matrix} \\ \\ -R_3/4 \end{matrix} \left(\begin{array}{ccc|c} 1 & 0 & 1 & 23 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 21 \end{array}\right) \longrightarrow$$

$$R_1 - R_3 \quad \left(\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 21 \end{array} \right).$$

Hence, the solution is

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 21 \end{pmatrix}.$$

7.2.1 Matrix Inversion by Jordan's Method

The solution of a set of linear equations amounts to finding, for a given matrix A and column vector $\mathbf{b}$, a column vector $\mathbf{x}$ satisfying

$$A\mathbf{x} = \mathbf{b}.$$

The related inversion problem is that of finding A^{-1} such that $AA^{-1} = I$, where I is a unit matrix of appropriate dimensions. To take a 3×3 example, we have

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where we wish to determine the elements α_{ij}. From the laws of matrix multiplication, it is apparent that the elements of the first column on the right-hand side are equal to the scalar product of the first, second and third rows of A respectively with the first column of A^{-1}. We may therefore calculate

$$\begin{pmatrix} \alpha_{11} \\ \alpha_{21} \\ \alpha_{31} \end{pmatrix}$$

by taking $(1 \;\; 0 \;\; 0)^T$ as the right-hand side of a set of three linear equations; the elements of the second and third columns of A^{-1} are found in a similar manner. Thus, the basic idea is to setup an augmented matrix of the form $[A \mid I]$ and apply a process which transforms $[A \mid I]$ into $[I \mid B]$; then, $B = A^{-1}$. The inversion of an $n \times n$ matrix is therefore equivalent to the solution of a set of n linear equations for n different right-hand sides - the columns of the unit matrix. We may deal with all the right-hand sides simultaneously, as illustrated in the following example which uses Jordan's method.

Example Consider the matrix

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 3 & -1 \\ 2 & 1 & -6 \end{pmatrix}$$

and the augmented matrix

$$\left(\begin{array}{ccc|ccc} 1 & 2 & -2 & 1 & 0 & 0 \\ 1 & 3 & -1 & 0 & 1 & 0 \\ 2 & 1 & -6 & 0 & 0 & 1 \end{array} \right).$$

We can then carry out the following process:

$$\begin{array}{c} \\ R_2 - R_1 \\ R_3 - 2R_1 \end{array} \left(\begin{array}{ccc|ccc} 1 & 2 & -2 & 1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 1 & 0 \\ 0 & -3 & -2 & -2 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 - 2R_2 \\ \\ R_3 + 3R_2 \end{array} \left(\begin{array}{ccc|ccc} 1 & 0 & -4 & 3 & -2 & 0 \\ 0 & 1 & 1 & -1 & 1 & 0 \\ 0 & 0 & 1 & -5 & 3 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 + 4R_3 \\ R_2 - R_3 \\ \\ \end{array} \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -17 & 10 & 4 \\ 0 & 1 & 0 & 4 & -2 & -1 \\ 0 & 0 & 1 & -5 & 3 & 1 \end{array}\right)$$

and hence,

$$A^{-1} = \left(\begin{array}{ccc} -17 & 10 & 4 \\ 4 & -2 & -1 \\ -5 & 3 & 1 \end{array}\right).$$

As a further example of this method, consider the following computation (using Jordan's method with partial pivoting):

$$\left(\begin{array}{cccc|cccc} 3 & 1 & -2 & -1 & 1 & 0 & 0 & 0 \\ 2 & -2 & 2 & 3 & 0 & 1 & 0 & 0 \\ 1 & 5 & -4 & -1 & 0 & 0 & 1 & 0 \\ 3 & 1 & 2 & 3 & 0 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_2 - \frac{2}{3}R_1 \\ R_3 - \frac{1}{3}R_1 \\ R_4 - R_1 \end{array} \left(\begin{array}{cccc|cccc} 3 & 1 & -2 & 1 & 1 & 0 & 0 & 0 \\ 0 & -8/3 & -10/3 & 11/3 & -2/3 & 1 & 0 & 0 \\ 0 & 14/3 & -10/3 & -2/3 & -1/3 & 0 & 1 & 0 \\ 0 & 0 & 4 & 4 & -1 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 - \frac{3}{14}R_3 \\ R_3 \\ R_2 + \frac{4}{7}R_3 \\ \end{array} \left(\begin{array}{cccc|cccc} 3 & 0 & -9/7 & -6/7 & 15/14 & 0 & -3/14 & 0 \\ 0 & 14/3 & -10/3 & -2/3 & -1/3 & 0 & 1 & 0 \\ 0 & 0 & 10/7 & 23/7 & -6/7 & 1 & 4/7 & 0 \\ 0 & 0 & 4 & 4 & -1 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 + \frac{9}{28}R_4 \\ R_2 + \frac{5}{6}R_4 \\ R_4 \\ R_3 - \frac{5}{14}R_4 \end{array} \left(\begin{array}{cccc|cccc} 3 & 0 & 0 & 3/7 & 3/4 & 0 & -3/14 & 9/28 \\ 0 & 14/3 & 0 & 8/3 & -7/6 & 0 & 1 & 5/6 \\ 0 & 0 & 4 & 4 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 13/7 & -1/2 & 1 & 4/7 & -5/14 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 - \frac{3}{13}R_4 \\ R_2 - \frac{56}{39}R_4 \\ R_3 - \frac{28}{13}R_4 \\ \end{array} \left(\begin{array}{cccc|cccc} 3 & 0 & 0 & 0 & 45/52 & -3/13 & -9/26 & 21/52 \\ 0 & 14/3 & 0 & 0 & -35/78 & -56/39 & 7/39 & 35/26 \\ 0 & 0 & 4 & 0 & 1/13 & -28/13 & -16/13 & 23/13 \\ 0 & 0 & 0 & 13/7 & -1/2 & 1 & 4/7 & -5/14 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \frac{1}{3}R_1 \\ \frac{3}{14}R_2 \\ \frac{1}{4}R_3 \\ \frac{7}{13}R_4 \end{array} \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 15/12 & -1/13 & -3/26 & 7/52 \\ 0 & 1 & 0 & 0 & -5/52 & -4/13 & 1/26 & 15/52 \\ 0 & 0 & 1 & 0 & 1/52 & -7/13 & -4/13 & 23/52 \\ 0 & 0 & 0 & 1 & -7/26 & 7/13 & 4/13 & -5/26 \end{array}\right).$$

7.2.2 Gauss' .v. Jordan's Method

Jordan's method requires more arithmetic than Gauss' method and there is no simple pivotal strategy available for this method. Although the method of Jordan is less efficient than that of Gauss for solving a single set of equations, the latter loses ground in the inversion problem, since it requires a separate back-substitution for each column of the inverse. Jordan's method involves no back-substitution, and in fact the total number of arithmetic operations proves to be identical for inversion by either method. Gauss' method of matrix inversion is the more difficult method to program, though it permits easier analysis of rounding errors. However, it is obviously inefficient from the computational point of view to solve a system of equations by first evaluating the inverse, which should in fact only be calculated if it is needed explicitly. In addition, the error analysis associated with Jordan's method for both the solution to linear systems of equations and matrix inversion is more difficult. In general, Jordan's method is usually used to invert matrices and Gauss' method (typically with partial pivoting) is used to solve systems of linear equations. It is often useful to design algorithms to do both to produce the so called the 'Gauss-Jordan' method.

7.3 LU Factorization

The basic idea of LU factorization is, given $A\mathbf{x} = \mathbf{b}$, factorize A into lower L and upper U triangular matrices. Thus, let

$$A = LU,$$

then

$$A\mathbf{x} = LU\mathbf{x} = \mathbf{b}.$$

Now, let

$$U\mathbf{x} = \mathbf{y}$$

so that

$$L\mathbf{y} = \mathbf{b}.$$

The problem then becomes a two stage process: (i) Solve the equation $L\mathbf{y} = \mathbf{b}$ by forward-substitution; (ii) solve the equation $U\mathbf{x} = \mathbf{y}$ by back-substitution. Now both forward and back-substitution is trivial and thus, the main problem is reduced to finding L and U given A. In Gauss' method, rounding errors occur each time an element of the reduced matrix is computed; LU factorization avoids this.

7.3.1 Existence and Uniqueness of LU Factorization

Consider the 3×3 system of equations with the characteristic matrix

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

and write this system in the form

$$A = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}.$$

As written, there are more unknowns than equations since $3 \times 3 \neq 6 + 6$.

Crout's Method

The problem becomes unique if we consider

$$A = LU_1 = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{pmatrix}.$$

Writing 1's along the diagonal of U $(= U_1)$ gives Crout's method.

Doolittle's Method

Alternatively, we can consider

$$A = L_1U = \begin{pmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}.$$

With 1's along the diagonal of L $(= L_1)$, the method is known as Doolittle's method.

Cholesky's Method

A special case arises if A is symmetric and positive definite. In this case, we can consider

$$A = CC^T = \begin{pmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \begin{pmatrix} c_{11} & c_{21} & c_{31} \\ 0 & c_{22} & c_{32} \\ 0 & 0 & c_{33} \end{pmatrix}.$$

This factorization is known as Cholesky's method.

Example Solve the equations

$$\begin{aligned} 2x_1 + x_2 + 3x_3 &= 14, \\ x_1 + 6x_2 + 4x_3 &= 3, \\ 3x_1 + 7x_2 + 2x_3 &= -8. \end{aligned}$$

Using Crout's method

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 6 & 4 \\ 3 & 7 & 2 \end{pmatrix} = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{pmatrix}.$$

Now, from R_1 :

$$\ell_{11} = 2; \quad \ell_{11}u_{12} = 1, \quad u_{12} = 1/2; \quad \ell_{11}u_{13} = 3, \quad u_{13} = 3/2.$$

From R_2 :

$$\ell_{21} = 1; \quad \ell_{21}u_{12} + \ell_{22} = 6, \quad \ell_{22} = 11/2; \quad \ell_{21}u_{13} + \ell_{22}u_{23} = 4, \quad u_{23} = 5/11,$$

and from R_3 :

$$\ell_{31} = 3; \quad \ell_{31}u_{12} + \ell_{32} = 7, \quad \ell_{32} = 11/2; \quad \ell_{31}u_{13} + \ell_{32}u_{23} + \ell_{33} = 2, \quad \ell_{33} = -5.$$

Hence,

$$A = LU_1 = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 11/2 & 0 \\ 3 & 11/2 & -5 \end{pmatrix} \begin{pmatrix} 1 & 1/2 & 3/2 \\ 0 & 1 & 5/11 \\ 0 & 0 & 1 \end{pmatrix}.$$

We can now solve $L\mathbf{y} = \mathbf{b}$ by forward-substitution, i.e. solve

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 11/2 & 0 \\ 3 & 11/2 & -5 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 14 \\ 3 \\ -8 \end{pmatrix}$$

giving the results

$$2y_1 = 14, \quad y_1 = 7;$$

$$y_1 + \frac{11}{2}y_2 = 3, \quad y_2 = -\frac{8}{11};$$

$$3y_1 + \frac{11}{2}y_2 - 5y_3 = -8, \quad y_3 = 5.$$

We can then solve $U\mathbf{x} = \mathbf{y}$ by back-substitution, i.e. solve

$$\begin{pmatrix} 1 & 1/2 & 3/2 \\ 0 & 1 & 5/11 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ -8/11 \\ 5 \end{pmatrix}$$

giving

$$x_3 = 5;$$

$$x_2 + \frac{5}{11}x_3 = -\frac{8}{11}, \quad x_2 = -3;$$

$$x_1 + \frac{1}{2}x_2 + \frac{3}{2}x_3 = 7, \quad x_1 = 1;$$

and the solution vector is

$$\mathbf{x} = (1, -3, 5)^T.$$

Example of Cholesky's Method

Solve the equations

$$\begin{array}{rcl} 2x_1 - x_2 & = & 1, \\ -x_1 + 2x_2 - x_3 & = & 0, \\ -x_2 + 2x_3 & = & 1. \end{array}$$

In this case

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} = \begin{pmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \begin{pmatrix} c_{11} & c_{21} & c_{31} \\ 0 & c_{22} & c_{32} \\ 0 & 0 & c_{33} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{2} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & 0 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \sqrt{\frac{3}{2}} & -\sqrt{\frac{2}{3}} \\ 0 & 0 & \sqrt{\frac{4}{3}} \end{pmatrix}.$$

We can then solve $C\mathbf{y} = \mathbf{b}$ by forward-substitution, i.e. solve

$$\begin{pmatrix} \sqrt{2} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & 0 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

giving

$$\mathbf{y} = \left(\frac{1}{\sqrt{2}}, \frac{1}{2}\sqrt{\frac{2}{3}}, \frac{4}{3}\sqrt{\frac{3}{4}} \right)^T.$$

Finally we solve $C^T\mathbf{x} = \mathbf{y}$ by back-substitution, i.e. solve

$$\begin{pmatrix} \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \sqrt{\frac{3}{2}} & -\sqrt{\frac{2}{3}} \\ 0 & 0 & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{2}\sqrt{\frac{2}{3}} \\ \frac{4}{3}\sqrt{\frac{3}{4}} \end{pmatrix}$$

to give the solution vector

$$\mathbf{x} = (1, 1, 1)^T.$$

7.3.2 Generalization of Crout's Method

We start be considering the general 3×3 system

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{pmatrix}.$$

Then,

R_1 : $\ell_{11} = a_{11}$; $\ell_{11}u_{12} = a_{12}$, $u_{12} = a_{12}/\ell_{11}$; $\ell_{11}u_{13} = a_{13}$, $u_{13} = a_{13}/\ell_{11}$.

R_2 : $\ell_{21} = a_{21}$; $\ell_{21}u_{12} + \ell_{22} = a_{22}$, $\ell_{22} = a_{22} - u_{12}\ell_{21}$; $\ell_{21}u_{13} + \ell_{22}u_{23} = a_{23}$,

$u_{23} = (a_{23} - \ell_{21}u_{13})/\ell_{22}$.

R_3 : $\ell_{31} = a_{31}$; $\ell_{31}u_{12} + \ell_{32}$, $\ell_{32} = a_{32} - \ell_{31}u_{12}$; $\ell_{33} = a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23}$..

Hence, by induction for an $n \times n$ system of equations we can write

$$u_{ij} = \frac{1}{\ell_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj} \right) \quad \text{for} \quad j = i+1, i+2, ..., n;$$

$$\ell_{ij} = a_{ji} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj} \quad \text{for} \quad j = i, i+1, i+2, ..., n$$

where $i = 1, 2, ..., n$ and requires that $\ell_{ii} \neq 0$.

The method is closely related to that of Gauss. The final matrix U is, in both cases, the same and, in fact, the element ℓ_{ij} of L is identical with the number by which the pivotal row is multiplied in Gauss' process before its subtraction from the i^{th} row to generate a zero in the position of a_{ij}. In Gauss' process, each time an element of a reduced matrix is calculated and recorded, a rounding error is likely to occur. With Crout's process, these errors may be largely avoided by the use of double-precision arithmetic in the calculation of the elements of L and U from the foregoing formulae. The results are then rounded to single precision and recorded on the completion of each calculation. The removal of the necessity for calculating and recording several intermediate matrices has therefore localized what might otherwise be a significant source of error to a single step in the determination of each element of L and U. The use of double precision arithmetic in this step leads to a degree of accuracy comparable to that which would result if the entire Gauss process were carried out with double precision. The total number of additions and of multiplications is the same as for Gauss' method. Since division by ℓ_{ii} is involved in calculating the elements of L, the method fails if any diagonal element of L is found to be zero. A row interchanging modification (analogous to the use of partial pivoting in Gauss' process) is available which overcomes this problem.

7.3.3 Generalization of Cholesky's Method

Consider the 3×3 system in which

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \begin{pmatrix} c_{11} & c_{21} & c_{31} \\ 0 & c_{22} & c_{32} \\ 0 & 0 & c_{33} \end{pmatrix}.$$

Then,

$R_1 : \quad c_{11}^2 = a_{11}, c_{11} = \sqrt{a_{11}}; \quad c_{11}c_{21} = a_{12}, \quad c_{21} = a_{12}/c_{11}; \quad c_{11}c_{31} = a_{13}, \quad c_{31} = a_{13}/c_{11}.$

$R_2 : \quad c_{21}^2 + c_{22}^2 = a_{22}, \quad c_{22} = \sqrt{a_{22} - c_{21}^2}; \quad c_{21}c_{31} + c_{22}c_{32} = a_{23}, \quad c_{32} = (a_{23} - c_{21}c_{31})/c_{22}.$

$R_3 : \quad c_{31}^2 + c_{32}^2 + c_{33}^2 = a_{33}, \quad c_{33} = \sqrt{a_{33} - c_{31}^2 - c_{32}^2}.$

Thus, for an $n \times n$ system of equation, we have

$$c_{ii} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} c_{jk}^2}, \quad \text{for} \quad i = j;$$

$$c_{ij} = \frac{1}{c_{jj}} \left(a_{ji} - \sum_{k=1}^{j-1} c_{ik} c_{jk} \right) \quad \text{for} \quad i > j$$

where $j = 1, 2, ..., n$ and $c_{jj} > 0 \forall j$. Note that, as with Gaussian elimination, row interchanges should be used to avoid any small diagonal elements which occur in the process. It is normal to use the largest available pivot at any step of the process.

It may be found that one of the diagonal elements of C is the square root of a negative number. This does not invalidate the factorization, but it does mean that other imaginary elements will arise in the course of the computation. No complex numbers will occur however and complex arithmetic need not be resorted to provided that an imaginary element is stored as its modulus and has a marker symbol associated with it. All that is necessary is for appropriate marking and sign changing on multiplication or division by such a marked element. The occurrence of a zero diagonal element causes the method to break down and in general, the use of row interchanging destroys the symmetry of the system. For this reason, Cholesky's method is perhaps best reserved for use with positive definite matrices, when all the diagonal elements will be real and greater than zero. In the positive definite case, there is an additional advantage, since if the equivalent of full pivoting is used, with appropriate row and column interchanges, the symmetry of the system is retained and a useful gain in accuracy may be achieved. Cholesky's method, used in the manner just described, may be expected to be slightly slower than the method of Gauss elimination used with full pivoting (as is possible for positive definite matrices), owing to the necessity for the evaluation of square roots. However, Cholesky's method has a slight advantage in accuracy and double-precision arithmetic may be used in computing inner products to gain a further improvement. Thus, Cholesky's method is an attractive proposition for problems involving positive definite matrices.

7.3.4 Matrix Inversion by LU Factorization

As with the Gauss and Jordan methods, inversion of an $n \times n$ matrix A may be accomplished, using triangular decomposition, by solving the equation $A\mathbf{x} = \mathbf{b}$ for n right-hand sides, namely the columns of the appropriate unit matrix. Improved accuracy is achieved, at little expense of extra computing time, if double precision arithmetic is used for the accumulation of inner products. For symmetric positive definite matrices, the method of Cholesky may be used to good advantage.

Our problem is to invert $A = LU$. Now,

$$LUU^{-1} = AU^{-1}$$

and

$$L = AU^{-1} \quad \text{since} \quad UU^{-1} = I.$$

Also,

$$LL^{-1} = AU^{-1}L^{-1}$$

and

$$I = AU^{-1}L^{-1} \quad \text{since} \quad LL^{-1} = I.$$

Hence,

$$A^{-1} = U^{-1}L^{-1}.$$

The recipe for matrix inversion using LU factorization is therefore as follows:

(i) Find L and U;

(ii) compute L^{-1} and U^{-1};

(iii) compute $L^{-1}U^{-1}$.

The advantage of this approach over Jordan's method is that the computation of L^{-1} and U^{-1} is relatively easy and computationally efficient.

7.4 Banded Systems

Many numerical problems lead to linear systems of equations with a banded structure. Examples include the computation of splines for modelling signals and surfaces, the solution of boundary value problems for differential equations using finite difference schemes and the discrete convolution of two discrete functions when the impulse response function consists of a few elements only.

A matrix $A = (a_{ij})$ is called a banded matrix if there is a positive integer k substantially smaller than the order n of the matrix such that

$$a_{ij} = 0 \quad \text{if} \quad | i - j | \geq k.$$

The parameter k is called the 'width' of the band. The storage requirements of banded matrices are significantly less than those required for general matrices of the same order.

7.4.1 Tridiagonal Systems

Tridiagonal systems are particularly important because of their commonality and simplicity of solution. They involve matrices whose elements are all zero except for those along the leading diagonal and the two diagonals just above and just below the leading diagonal (the sub-diagonals), i.e.

$$A = (a_{ij}) = 0 \quad \forall i, j \quad \text{if} \quad | i - j | \geq 2.$$

A $n \times n$ tridiagonal matrix must have at most $n + 2(n-1) = 3n - 2$ non-zero elements. Consider the following tridiagonal system

$$\begin{array}{lcl} d_1x_1 + c_1x_2 & = & b_1, \\ a_1x_1 + d_2x_2 + c_2x_3 & = & b_2, \\ a_2x_2 + d_3x_3 + c_3x_4 & = & b_3, \\ & \vdots & \\ a_{n-2}x_{n-2} + d_{n-1}x_{n-1} + c_{n-1}x_n & = & b_{n-1}, \\ a_{n-1}x_{n-1} + d_nx_n & = & b_n. \end{array}$$

Instead of storing an $n \times n$ matrix we need only store the vectors a_i, d_i, c_i which have dimensions of $n-1, n$ and $n-1$ respectively.

7.4.2 Solution to Tridiagonal Systems

By choosing the diagonal element d_1 as a pivot, we need only eliminate x_1 from the second equation, all other equations remaining the same. This gives the following:

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1, \\ d_2^{(1)}x_2 + c_2x_3 &= b_2^{(1)}, \\ a_2x_2 + d_3x_3 + c_3x_4 &= b_3, \\ &\vdots \end{aligned}$$

where

$$\begin{aligned} d_2^{(1)} &= d_2 - \frac{a_1}{d_1}c_1, \\ b_2^{(1)} &= b_2 - \frac{a_1}{d_1}b_1. \end{aligned}$$

Choosing $d_2^{(1)}$ as the pivot, we get

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1, \\ d_2^{(1)}x_2 + c_2x_3 &= b_2^{(1)}, \\ d_3^{(1)}x_3 + c_3x_4 &= b_3^{(2)}, \\ &\vdots \end{aligned}$$

where

$$\begin{aligned} d_3^{(2)} &= d_3 - \frac{a_2}{d_2^{(1)}}c_2, \\ b_3^{(2)} &= b_3 - \frac{a_2}{d_2^{(1)}}b_1^{(2)}. \end{aligned}$$

Repeating this operation, we obtain an upper triangular matrix of the form

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1, \\ d_2^{(1)}x_2 + c_2x_3 &= b_2^{(1)}, \\ d_3^{(2)}x_3 + c_3x_4 &= b_3^{(2)}, \\ &\vdots \\ d_{n-1}^{(n-2)}x_{n-1} + c_{n-1}x_n &= b_{n-1}^{(n-2)}, \\ d_n^{(n-1)}x_n &= b_n^{(n-1)}, \end{aligned}$$

where

$$\begin{aligned} d_{k+1}^{(k)} &= d_{k+1} - \frac{a_k}{d_k^{(k-1)}}c_k, \\ b_{k+1}^{(k)} &= b_{k+1} - \frac{a_k}{d_k^{(k-1)}}b_k^{(k-1)}, \end{aligned}$$

for $k = 1, 2, ..., n-1$. Back-substitution then gives

$$x_n = \frac{b_n^{(n-1)}}{d_n^{(n-1)}}$$

and

$$x_k = \frac{b_k^{(k-1)} - c_k x_{k+1}}{d_k^{(k-1)}}, \quad k = n-1, n-2, ..., 1.$$

The total number of additions and multiplications in this algorithm is $2(n-1)+n-1 = 3n-3$ and the number of divisions is $(n-1)+n = 2n-1$. For large n, this is small compared to the $\sim n^3$ additions and multiplications required by Gaussian elimination.

Example Solve

$$\begin{array}{lcl} x_1 + 2x_2 & = & 2, \\ x_1 + 3x_2 + 2x_3 & = & 7, \\ x_2 + 4x_3 + 2x_4 & = & 15, \\ 4x_3 + x_4 & = & 11. \end{array}$$

The augmented matrix is

$$\left(\begin{array}{cccc|c} 1 & 2 & 0 & 0 & 2 \\ 1 & 3 & 2 & 0 & 7 \\ 0 & 1 & 4 & 2 & 15 \\ 0 & 0 & 4 & 1 & 11 \end{array}\right)$$

and we undertake the processing to upper triangular form as follows:

$$R_2 - R_1 \left(\begin{array}{cccc|c} 1 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 0 & 5 \\ 0 & 1 & 4 & 2 & 15 \\ 0 & 0 & 4 & 1 & 11 \end{array}\right) \longrightarrow R_3 - R_2 \left(\begin{array}{cccc|c} 1 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 0 & 5 \\ 0 & 0 & 2 & 2 & 10 \\ 0 & 0 & 4 & 1 & 11 \end{array}\right) \longrightarrow$$

$$R_4 - 2R_3 \left(\begin{array}{cccc|c} 1 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 0 & 5 \\ 0 & 0 & 2 & 2 & 10 \\ 0 & 0 & 0 & -3 & -9 \end{array}\right)$$

Back-substituting:

$x_4 = \frac{-9}{-3} = 3, \quad x_3 = \frac{10-2\times 3}{2} = 2, \quad x_2 = 5 - 2 \times 2 = 1$ and $x_1 = 2 - 2 \times 1 = 0$.

7.5 Computational Considerations

7.5.1 Computer Storage Requirements

Let us first consider the storage needed by a program for solving a single set of linear equations, which is effectively independent of the method used. Unless a copy of the original matrix of coefficients is required, this original matrix may be overwritten as

the computation proceeds; the space needed is therefore that for $n(n+1)$ variables plus a little working space, where the order of the system is $n \times n$.

With LU factorization, whichever method is used, it is customary to store not only the final upper triangular matrix but also the corresponding lower triangular matrix. The latter is obtained as previously described in the case of the decomposition methods; when using the elimination methods we obtain its element ℓ_{ij} as the number by which the j^{th} pivotal row is multiplied before subtraction from the i^{th} row in the process of generating zeros. All this information is stored for a number of right-hand sides, not all of which are known at the time when the elimination or decomposition is being performed. Under such circumstances, we must be able to apply to the new right-hand sides, the same transformations as would normally be applied during the reduction process; hence, the necessity for storing the multipliers. The elements of both the upper and lower triangles may be recorded in the same rectangular array. The diagonal elements of this array, however, should be used to store the reciprocals of the pivots (or of the diagonal elements of U in Doolittle's method or the diagonal elements of L in Crout's and Cholesky's methods), since these reciprocals are used in calculating the multipliers ℓ_{ij} and subsequently in the back-substitution. In Doolittle's method, for example, an original 4×4 matrix A would be overwritten by

$$\begin{pmatrix} u_{11}^{-1} & u_{12} & u_{13} & u_{14} \\ \ell_{21} & u_{22}^{-1} & u_{23} & u_{24} \\ \ell_{31} & \ell_{32} & u_{33}^{-1} & u_{34} \\ \ell_{41} & \ell_{42} & \ell_{43} & u_{44}^{-1} \end{pmatrix}.$$

The inversion problem amounts to the solution of n linear equations for n right-hand sides, and storage is therefore needed for $2n^2$ variables plus working space.

7.5.2 Arithmetic Involved in the Computation

The following tables give the number of arithmetic operations necessary for the solution of an $n \times n$ system of equations and inversion of an $n \times n$ matrix.

Methods of Solution

Method	Reciprocals	Multiplications	Additions
Gauss	n	$n(\frac{1}{3}n^2 + n - \frac{1}{3})$	$n(\frac{1}{3}n^2 + \frac{1}{2}n - \frac{5}{6})$
Decomposition	n	$n(\frac{1}{3}n^2 + n - \frac{1}{3})$	$n(\frac{1}{3}n^2 + \frac{1}{2}n - \frac{5}{6})$
Jordan	n	$\frac{1}{2}n^3 + n^2 - \frac{1}{2}n$	$\frac{1}{2}n^3 - \frac{1}{2}n$

Methods of Inversion

Method	Reciprocals	Multiplications	Additions
Gauss	n	$n^3 - 1$	$n^3 - 2n^2 + n$
Decomposition	n	$n^3 - 1$	$n^3 - 2n^2 + n$
Jordan	n	$n^3 - 1$	$n^3 - 2n^2 + n$

In general, all three methods of solution are equally efficient for inverting a matrix, though Jordan's method is less so than the others for solving a single system of equations. When symmetric positive definite matrices arise, a significant saving may be made using Cholesky's method or the symmetric version of Gauss' method. The use of row (or row and column) interchanges with the above methods involves no extra arithmetic, but the necessary search for the largest pivot takes additional time, as does the process of reshuffling the rows at each stage.

7.5.3 Scaling of Matrices

It is good practice to scale the rows and/or columns of a matrix before commencing a computation so that the maximum modulus of the elements in any row and column lies between 0.5 and 1. Unfortunately, it is not easy to define a 'best' method of doing this, and justification for it in the literature seems to be based more on experience and intuition than on theory. The effect appears to be that the numerical stability of the computation is enhanced. Furthermore, most error analyses are concerned with matrices which have been 'scaled' in this manner, so that it is easier to ascertain what reliance may be placed on computed solutions. Row scaling, of course, does not affect the computed solutions, but column scaling has the effect of dividing the solution component for a particular column by the scaling multiplier. It is usual to scale by powers of 2 so that (in the binary arithmetic used by the computer) no rounding errors arise.

7.6 Solution to Complex Systems of Equations

Complex systems of equations may be solved by any of the standard methods with the use of complex arithmetic. Alternatively, real arithmetic may be used throughout if advantage is taken of the fact that the matrices

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

have the same algebraic properties as 1 and i, the real and imaginary units. A complex number $a + ib$ may therefore be represented as the 2×2 matrix

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}.$$

If every complex element of an $n \times n$ matrix is replaced by a 2×2 matrix of this type, the result is a $2n \times 2n$ matrix whose elements are all real, and any of the methods discussed here may be employed to deal with it. In the following example Jordan elimination is used (without pivoting) in order that the final result is obtained without back-substitution.

Example Solve the equations

$$(1 + i)z_1 - 2iz_2 = -5 + 3i,$$

$$(3 - 2i)z_1 + (2 + i)z_2 = 18 - 5i.$$

The augmented matrix and its transformations are

$$\left(\begin{array}{cccc|cc} 1 & 1 & 0 & -2 & -5 & 3 \\ -1 & 1 & 2 & 0 & -3 & -5 \\ 3 & -2 & 2 & 1 & 18 & -5 \\ 2 & 3 & -1 & 2 & 5 & 18 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_2 + R_1 \\ R_3 - 3R_1 \\ R_4 - 2R_1 \end{array} \left(\begin{array}{cccc|cc} 1 & 1 & 0 & -2 & -5 & 3 \\ 0 & 2 & 2 & -2 & -8 & -2 \\ 0 & -5 & 2 & 7 & 33 & -14 \\ 0 & 1 & -1 & 6 & 15 & 12 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 - \frac{1}{2}R_2 \\ \\ R_3 + \frac{5}{2}R_2 \\ R_4 - \frac{1}{2}R_2 \end{array} \left(\begin{array}{cccc|cc} 1 & 0 & -1 & -1 & -1 & 4 \\ 0 & 2 & 2 & -2 & -8 & -2 \\ 0 & 0 & 7 & 2 & 13 & -19 \\ 0 & 0 & -2 & 7 & 19 & 13 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 + \frac{1}{7}R_3 \\ R_2 - \frac{2}{7}R_3 \\ \\ R_4 + \frac{2}{7}R_3 \end{array} \left(\begin{array}{cccc|cc} 1 & 0 & 0 & -5/7 & 6/7 & 9/7 \\ 0 & 2 & 0 & -18/7 & -82/7 & 24/7 \\ 0 & 0 & 7 & 2 & 13 & -19 \\ 0 & 0 & 0 & 53/7 & 159/7 & 53/7 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 + \frac{5}{53}R_4 \\ R_2 + \frac{18}{53}R_4 \\ R_3 - \frac{14}{53}R_4 \\ \\ \end{array} \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 3 & 2 \\ 0 & 2 & 0 & 0 & -4 & 6 \\ 0 & 0 & 7 & 0 & 7 & -21 \\ 0 & 0 & 0 & 53/7 & 159/7 & 53/7 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \frac{1}{2}R_2 \\ \frac{1}{7}R_3 \\ \frac{7}{53}R_4 \end{array} \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 3 & 2 \\ 0 & 1 & 0 & 0 & -2 & 3 \\ 0 & 0 & 1 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 & 3 & 1 \end{array}\right).$$

The solutions are clearly

$$z_1 = 3 + 2i, \quad z_2 = 1 - 3i.$$

7.7 Summary of Important Results

Gauss' Method

$$A\mathbf{x} = \mathbf{b} \longrightarrow U\mathbf{x} = \mathbf{h}$$

where U is an upper triangular matrix.

Jordan's Method

$$A\mathbf{x} = \mathbf{b} \longrightarrow I\mathbf{x} = \mathbf{h}$$

where I is the unit matrix.

Crout's Method

$$A\mathbf{x} = \mathbf{b} \longrightarrow LU_1\mathbf{x} = \mathbf{b}$$

Doolittle's Method

$$A\mathbf{x} = \mathbf{b} \longrightarrow L_1 U\mathbf{x} = \mathbf{b}$$

Cholesky's Method

$$A\mathbf{x} = \mathbf{b} \longrightarrow CC^T\mathbf{x} = \mathbf{b}$$

7.8 Further Reading

- Goult R J, Hoskins R F, Milner J A and Pratt M J, *Computational Methods in Linear Algebra*, Thornes, 1974.

  Atkinson K E, *An Introduction to Numerical Analysis*, Wiley, 1978.

- Johnson L W and Dean Riess R, *Introduction to Linear Algebra*, Addison-Wesley, 1981.

- Yakowitz S and Szidarovszky F, *An Introduction to Numerical Computations*, Macmillan, 1989.

- Fraleigh J B and Beauregard R A, *Linear Algebra*, Addison-Wesley, 1990.

7.9 Problems

7.1 Use Gaussian Elimination (GE) with natural pivoting to solve the following equations

$$x_1 - x_2 = 1,$$

$$x_2 + x_4 = -1,$$

$$x_1 - x_3 = 2,$$

$$x_1 + x_2 + x_3 + x_4 = 1.$$

7.2 Use GE with (i) natural pivoting and (ii) partial pivoting to solve the equations

$$x_1 + 2x_2 - x_3 = -3,$$

$$3x_1 + 7x_2 + 2x_3 = 1,$$

$$4x_1 - 2x_2 + x_3 = -2.$$

7.3 Use GE with (i) natural pivoting, (ii) partial pivoting and (iii) complete pivoting to solve the equations

$$x_1 + 3x_2 + 2x_3 - 4x_4 = 10,$$

$$-2x_1 + x_2 + 4x_3 + x_4 = 11,$$
$$x_1 - 2x_2 - 3x_3 + 2x_4 = -9,$$
$$3x_1 - 3x_2 - 5x_3 - 2x_4 = -17.$$

7.4 (i) Obtain the inverse of the matrices

$$\begin{pmatrix} 3 & 2 & -1 \\ 1 & -1 & 2 \\ 2 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$$

using Jordan's method.

7.5 Use Jordan's method with natural pivoting to find the inverse of the matrix

$$\begin{pmatrix} 1 & 2 & 0 & 1 \\ -1 & -1 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 4 & -1 & 5 \end{pmatrix}$$

and hence solve,

$$x_1 + 2x_2 + x_4 = 2,$$
$$-x_1 - x_2 + x_3 = 1,$$
$$2x_1 + 3x_2 = 5,$$
$$x_1 + 4x_2 - x_3 + 5x_4 = 0.$$

When is it be computationally efficient to compute the inverse of the characteristic matrix of a system of linear equations?

7.6 Use Crout's method with exact arithmetic to solve the following system of linear equations

$$\begin{aligned} 2x_1 + x_2 - 3x_3 &= -5, \\ x_1 - 2x_2 - x_3 &= -6, \\ x_1 + x_2 + x_3 &= 6. \end{aligned}$$

7.7 Using Cholesky factorization and exact arithmetic, solve the equations

$$\begin{aligned} 2x_1 - x_2 &= 1, \\ -x_1 + 2x_2 - x_3 &= 0, \\ -x_2 + 2x_3 &= 1. \end{aligned}$$

7.8 Obtain the Doolittle factorization of the matrix

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$$

and hence solve the system of equations given by

$$A\mathbf{x} = \mathbf{b}$$

where $\mathbf{b} = (1 \quad 2 \quad 3 \quad 4)^T$.

7.9 Solve the system of linear equations given in the last question using Cholesky factorization and exact arithmetic.

7.10 Using an LU factorization method of your choice, solve the equations

$$\begin{aligned} x_1 + 2x_2 - 3x_3 + x_4 &= -4, \\ 2x_1 + 5x_2 - 4x_3 + 6x_4 &= -1, \\ -x_1 + x_2 + 13x_3 + 31x_4 &= 53, \\ 2x_1 + 3x_2 - 5x_3 + 15x_4 &= 8. \end{aligned}$$

Chapter 8

Vector and Matrix Norms

This chapter is devoted to a study of vector and matrix norms and the role they play in the analysis of linear algebraic systems. In particular, we derive results and study approaches to the analysis of discrete variables that are used routinely in computational algorithms for a wide range of applications inclusive of those designed for solving sets of linear simultaneous equations and digital signal processing in general. For example, we use the principal results of vector and matrix norms to derive criterion on the stability of linear systems and investigate the role norms play in the optimization methods for solving these systems.

8.1 Vector Norms

Consider the set of vectors

$$\mathbf{x}_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 1 \\ 10 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} 1 \\ 2.5 \end{pmatrix}.$$

Which is the 'biggest' vector? We could just choose the maximum value of each vector, i.e. 3, 10, 2.5. We could also just add the elements together (5, 11, 3.5) or we could also use Pythagoras' theorem to get $\simeq (3.6, 10.0, 2.7)$. In general, for an arbitrary vector $\mathbf{x} \in \mathrm{R}^n$, i.e.

$$\mathbf{x} = (x_1, x_2, ..., x_n)^T$$

we can define various different numbers which provide a measure of the size of the vector such as

$$x_1 + x_2 + ... + x_n,$$
$$(x_1^2 + x_2^2 + ... + x_n^2)^{\frac{1}{2}},$$
$$(x_1^3 + x_2^3 + ... + x_n^3)^{\frac{1}{3}},$$
$$\vdots$$

Alternatively, we could choose $\mathbf{x}_{\max}$, the element of the vector with the largest magnitude value.

8.1.1 Definitions of a Vector Norm

For a vector $\mathbf{x}$, the norm is denoted by $\|\mathbf{x}\|$ and satisfies the following:

$$\|\mathbf{x}\| = 0 \quad \text{if} \quad \mathbf{x} = 0, \quad \text{otherwise} \quad \|\mathbf{x}\| > 0;$$

$$\|k\mathbf{x}\| = \mid k \mid \|\mathbf{x}\| \quad (\text{the homogeneity condition})$$

where k is a scalar and

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|,$$

$$\|\mathbf{x} \cdot \mathbf{y}\| \leq \|\mathbf{x}\|\|\mathbf{y}\|.$$

8.1.2 Commonly Used Definitions

Three useful and commonly used classes of vector norm are:

(i) The ℓ_1 norm given by

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} \mid x_i \mid .$$

(ii) The ℓ_2 norm or Euclidean norm defined as

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} \mid x_i \mid^2 \right)^{\frac{1}{2}}.$$

(iii) The ℓ_∞ norm (the 'infinity' or 'uniform' norm) given by

$$\|\mathbf{x}\|_\infty = \max_i \mid x_i \mid .$$

8.1.3 Generalization of Vector Norms - The 'p-Norm'

Vector norms can be generalized by introducing the so called 'p-norm' defined as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} \mid x_i \mid^p \right)^{\frac{1}{p}}, \quad p \geq 1.$$

Then,

$$\|\mathbf{x}\|_1 \equiv \|\mathbf{x}\|_{p=1}$$

$$\|\mathbf{x}\|_2 \equiv \|\mathbf{x}\|_{p=2}$$

$$\vdots$$

Theorem

$$\lim_{p \to \infty} \|\mathbf{x}\|_p = \|\mathbf{x}\|_\infty$$

Proof Let

$$\mid x_m \mid = \max_i \mid x_i \mid \equiv \|\mathbf{x}\|_\infty.$$

Then,

$$\|\mathbf{x}\|_p = \left[\sum_{i=1}^{n} \mid x_m \mid^p \left(\frac{\mid x_i \mid}{\mid x_m \mid}\right)^p\right]^{\frac{1}{p}} = \left[\mid x_m \mid^p + \mid x_m \mid^p \sum_{\substack{i=1 \\ i\neq m}}^{n} \left(\frac{\mid x_i \mid}{\mid x_m \mid}\right)^p\right]^{\frac{1}{p}}$$

$$= \mid x_m \mid \left[1 + \sum_{\substack{i=1 \\ i\neq m}}^{n} \left(\frac{\mid x_i \mid}{\mid x_m \mid}\right)^p\right]^{\frac{1}{p}}.$$

Hence,

$$\lim_{p\to\infty} \|\mathbf{x}\|_p = \mid x_m \mid \lim_{p\to\infty} \left[1 + \sum_{\substack{i=1 \\ i\neq m}}^{n} \left(\frac{\mid x_i \mid}{\mid x_m \mid}\right)^p\right]^{\frac{1}{p}} = \mid x_m \mid \equiv \|\mathbf{x}\|_\infty.$$

8.1.4 Metric Spaces

It is sometimes convenient to define a distance $d(\mathbf{x}, \mathbf{y})$ between any two vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$$

which defines a metric or measure and where the norm can be arbitrary. A sequence of vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, ...$ converge to $\mathbf{x}$ if

$$\lim_{k\to\infty} d(\mathbf{x}, \mathbf{x}^{(k)}) = \lim_{k\to\infty} \|\mathbf{x} - \mathbf{x}^{(k)}\| = 0.$$

Metrics of this type are typically used to study the convergence of iterative solutions involving vector and/or matrix algebra for example.

8.1.5 The Euclidean Norm

The Euclidean norm is given by

$$\|\mathbf{x}\|_2 = (x_1^2 + x_2^2 + ... + x_n^2)^{\frac{1}{2}}.$$

Other forms of notation of this important and widely used vector norm include

$$\sqrt{\mathbf{x} \cdot \mathbf{x}} \equiv \mid \mathbf{x} \mid \quad \text{and} \quad \langle \mathbf{x}, \mathbf{x} \rangle.$$

The Euclidean norm is one of the most commonly used norms. It has a variety of applications in functional and numerical analysis. The ℓ_2 norm defines the length of modulus of $\mathbf{x}$ and $\mathbf{x} \cdot \mathbf{y}$ defines the scalar product in R^n given by

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + ... + x_n y_n.$$

Two important inequalities relating to the Euclidean norm are:

(i) the Cauchy-Schwarz inequality given by

$$| \mathbf{x} \cdot \mathbf{y} | \leq | \mathbf{x} | | \mathbf{y} |;$$

(ii) the triangle inequality, namely

$$| \mathbf{x} + \mathbf{y} | \leq | \mathbf{x} | + | \mathbf{y} | .$$

8.1.6 The Cauchy-Schwarz Inequality for the Euclidean Norm

For any real number θ, let $(\mathbf{x} \cdot \mathbf{y})\theta\mathbf{x} - \mathbf{y}$ be a vector in R^n. Then

$$[(\mathbf{x} \cdot \mathbf{y})\theta\mathbf{x} - \mathbf{y}] \cdot [(\mathbf{x} \cdot \mathbf{y})\theta\mathbf{x} - \mathbf{y}] \geq 0.$$

Expanding, we get

$$(\mathbf{x} \cdot \mathbf{y})^2\theta^2 | \mathbf{x} |^2 - 2(\mathbf{x} \cdot \mathbf{y})^2\theta + | \mathbf{y} |^2 \geq 0$$

and with

$$\theta = \frac{1}{| \mathbf{x} |^2},$$

we have

$$(\mathbf{x} \cdot \mathbf{y})^2 \frac{1}{| \mathbf{x} |^2} - 2(\mathbf{x} \cdot \mathbf{y})^2 \frac{1}{| \mathbf{x} |^2} + | \mathbf{y} | \geq 0,$$

$$| \mathbf{x} |^2 | \mathbf{y} |^2 - (\mathbf{x} \cdot \mathbf{y})^2 \geq 0,$$

$$| \mathbf{x} \cdot \mathbf{y} | \leq | \mathbf{x} | | \mathbf{y} | .$$

The generalization of this result is 'Holder's inequality', i.e.

$$\sum_{i=1}^{n} | x_i | | y_i | \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q.$$

8.2 The Triangle Inequality for the Euclidean Norm

We start with

$$| \mathbf{x} + \mathbf{y} |^2 = (\mathbf{x} + \mathbf{y}) \cdot (\mathbf{x} + \mathbf{y}) = \mathbf{x} \cdot \mathbf{x} + 2\mathbf{x} \cdot \mathbf{y} + \mathbf{y} \cdot \mathbf{y}$$

$$\leq | \mathbf{x} |^2 + 2 | \mathbf{x} | | \mathbf{y} | + | \mathbf{y} |^2$$

by the Cauchy-Schwarz inequality. Now,

$$| \mathbf{x} |^2 + 2 | \mathbf{x} | | \mathbf{y} | + | \mathbf{y} |^2 = (| \mathbf{x} | + | \mathbf{y} |)^2$$

and thus,

$$| \mathbf{x} + \mathbf{y} |^2 \leq (| \mathbf{x} | + | \mathbf{y} |)^2$$

or

$$| \mathbf{x} + \mathbf{y} | \leq | \mathbf{x} | + | \mathbf{y} | .$$

Hence,

$$\|\mathbf{x} + \mathbf{y}\|_2 \leq \|\mathbf{x}\|_2 + \|\mathbf{y}\|_2.$$

The generalization of this result to a 'p-norm' is given by the 'Minkowski inequality', i.e.

$$\left(\sum_{i=1}^{n} | x_i + y_i |^p \right)^{\frac{1}{p}} \leq \left(\sum_{i=1}^{n} | x_i |^p \right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n} | y_i |^p \right)^{\frac{1}{p}}.$$

8.2.1 Holder's Inequality

Holder's inequality is based on application of the result

$$a^{\frac{1}{p}} b^{\frac{1}{q}} \leq \frac{a}{p} + \frac{b}{q}$$

where $p > 1$ and

$$\frac{1}{p} + \frac{1}{q} = 1.$$

Let us first look at a proof of this result. If

$$\frac{1}{p} + \frac{1}{q} = 1$$

then

$$\frac{1}{q} = 1 - \frac{1}{p}, \quad q = \frac{p}{(p-1)}; \Longrightarrow q > 1 \quad \text{if} \quad p > 1.$$

Thus,

$$t^{-\frac{1}{q}} \leq 1, \quad \text{if} \quad t \geq 1$$

and therefore

$$\int_1^x t^{-\frac{1}{q}} dt \leq \int_1^x 1 dx \quad \text{for} \quad x \geq 1.$$

Evaluating the integrals, we have

$$\left[\frac{t^{1-\frac{1}{q}}}{1-\frac{1}{q}} \right]_1^x \leq [x]_1^x$$

or

$$\frac{x^{1-\frac{1}{q}}}{1-\frac{1}{q}} - \frac{1}{1-\frac{1}{q}} \leq x - 1.$$

Rearranging, we get

$$p x^{\frac{1}{p}} - p \leq x - 1$$

or

$$x^{\frac{1}{p}} \leq \frac{x}{p} + 1 - \frac{1}{p}.$$

Now, since

$$\frac{1}{q} = 1 - \frac{1}{p}$$

we obtain

$$x^{\frac{1}{p}} \leq \frac{x}{p} + \frac{1}{q}.$$

Suppose that $a \geq b > 0$ and let $x = a/b$. Using the result derived above we get

$$\frac{a^{\frac{1}{p}}}{b^{\frac{1}{p}}} \leq \frac{a}{bp} + \frac{1}{q}$$

and therefore

$$a^{\frac{1}{p}} b^{(1-\frac{1}{p})} \leq \frac{a}{p} + \frac{b}{q}$$

or

$$a^{\frac{1}{p}} b^{\frac{1}{q}} \leq \frac{a}{p} + \frac{b}{q}.$$

By symmetry, p and q can be interchanged giving

$$a^{\frac{1}{q}} b^{\frac{1}{p}} \leq \frac{a}{q} + \frac{b}{p}.$$

Having derived the above, we can now go on to prove Holder's inequality by introducing the following definitions for a and b:

$$a = \frac{\mid x_i \mid^p}{\sum_{i=1}^{n} \mid x_i \mid^p}$$

and

$$b = \frac{\mid y_i \mid^q}{\sum_{i=1}^{n} \mid y_i \mid^q}.$$

Using these results, we have

$$\left(\frac{\mid x_i \mid^p}{\sum_{i=1}^{n} \mid x_i \mid^p} \right)^{\frac{1}{p}} \left(\frac{\mid y_i \mid^q}{\sum_{i=1}^{n} \mid y_i \mid^q} \right)^{\frac{1}{q}} \leq \frac{\mid x_i \mid^p}{p \sum_{i=1}^{n} \mid x_i \mid^p} + \frac{\mid y_i \mid^q}{q \sum_{i=1}^{n} \mid y_i \mid^q}$$

and summing over i gives

$$\frac{\sum_{i=1}^{n} \mid x_i \mid\mid y_i \mid}{\left(\sum_{i=1}^{n} \mid x_i \mid^p \right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} \mid y_i \mid^q \right)^{\frac{1}{q}}} \leq \frac{\sum_{i=1}^{n} \mid x_i \mid^p}{p \sum_{i=1}^{n} \mid x_i \mid^p} + \frac{\sum_{i=1}^{n} \mid y_i \mid^q}{q \sum_{i=1}^{n} \mid y_i \mid^q}.$$

Hence,

$$\frac{\sum_{i=1}^{n} \mid x_i \mid\mid y_i \mid}{\left(\sum_{i=1}^{n} \mid x_i \mid^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} \mid y_i \mid^q\right)^{\frac{1}{q}}} \leq \frac{1}{p} + \frac{1}{q} = 1$$

or

$$\sum_{i=1}^{n} \mid x_i \mid\mid y_i \mid \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q.$$

8.2.2 Minkowski's Inequality

If $p > 1$, then using the result

$$\mid x_i + y_i \mid \leq \mid x_i \mid + \mid y_i \mid$$

together with Holder's inequality, we can write

$$\sum_{i=1}^{n} \mid x_i+y_i \mid^p = \sum_{i=1}^{n} \mid x_i+y_i \mid^{p-1} \mid x_i+y_i \mid \leq \sum_{i=1}^{n} \mid x_i \mid\mid x_i+y_i \mid^{p-1} + \sum_{i=1}^{n} \mid y_i \mid\mid x_i+y_i \mid^{p-1}$$

$$\leq \left(\sum_{i=1}^{n} \mid x_i \mid^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} \mid x_i + y_i \mid^{q(p-1)}\right)^{\frac{1}{q}} + \left(\sum_{i=1}^{n} \mid y_i \mid^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} \mid x_i + y_i \mid^{q(p-1)}\right)^{\frac{1}{q}}.$$

But $q(p-1) = p$ and therefore

$$\sum_{i=1}^{n} \mid x_i + y_i \mid^p \leq \left(\sum_{i=1}^{n} \mid x_i + y_i \mid^p\right)^{\frac{1}{q}} \left(\sum_{i=1}^{n} \mid x_i \mid^p\right)^{\frac{1}{p}}$$

$$+ \left(\sum_{i=1}^{n} \mid x_i + y_i \mid^p\right)^{\frac{1}{q}} \left(\sum_{i=1}^{n} \mid y_i \mid^p\right)^{\frac{1}{p}}.$$

Dividing through by

$$\left(\sum_{i=1}^{n} \mid x_i + y_i \mid^p\right)^{\frac{1}{q}}$$

gives

$$\left(\sum_{i=1}^{n} \mid x_i + y_i \mid^p\right)^{1-\frac{1}{q}} \leq \left(\sum_{i=1}^{n} \mid x_i \mid^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n} \mid y_i \mid^2\right)^{\frac{1}{p}}.$$

Finally, since $1 - 1/q = 1/p$, we get

$$\left(\sum_{i=1}^{n} \mid x_i + y_i \mid^p\right)^{\frac{1}{p}} \leq \left(\sum_{i=1}^{n} \mid x_i \mid^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n} \mid y_i \mid^p\right)^{\frac{1}{p}}.$$

8.3 Matrix Norms

The norm of a matrix A is, like the vector norm, denoted by $\|A\|$. A matrix norm satisfies the following conditions:

$$\|A\| = 0 \quad \text{if} \quad A = 0 \quad \text{otherwise} \quad \|A\| > 0;$$

$$\|kA\| = \mid k \mid \|A\| \quad (\text{the homogeneity condition});$$

$$\|A + B\| \leq \|A\| + \|B\|;$$

$$\|AB\| \leq \|A\|\|B\|.$$

8.3.1 Types of Matrix Norms

Matrix norms are in many ways similar to those used for vectors. Thus, we can consider an ℓ_2 (matrix) norm (analogous to the Euclidean norm for vectors) given by

$$\|A\|_2 = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \mid a_{ij} \mid^2 \right)^{\frac{1}{2}}.$$

Then there is the ℓ_1 (matrix) norm,

$$\|A\|_1 = \max_j \sum_{i=1}^{n} \mid a_{ij} \mid$$

and the ℓ_∞ (matrix) norm defined as

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} \mid a_{ij} \mid .$$

8.3.2 Basic Definition of a Matrix Norm

Let $\|\mathbf{x}\|$ be any (vector) norm on the n-dimensional vector space R^n. For any real $n \times n$ matrix A we define the 'subordinate matrix norm' of A as

$$\|A\| = \max \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$

where the maximum is taken for all vectors $\mathbf{x} \in R^n$. The main point of defining a matrix norm in this way is that because $\|A\mathbf{x}\|$ is a vector norm, the results and properties derived for vector norms can be applied directly to matrix norms. This is compounded in the following theorems.

Theorem If A and B are both $n \times n$ matrices then for any matrix norm

$$\|A + B\| \leq \|A\| + \|B\|.$$

Proof By Minkowski's inequality

$$\|(A+B)\mathbf{x}\| = \|A\mathbf{x}+B\mathbf{x}\| \leq \|A\mathbf{x}\| + \|B\mathbf{x}\|.$$

Dividing through by $\|\mathbf{x}\|$, we have

$$\frac{\|(A+B)\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} + \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|}$$

and therefore

$$\max\frac{\|(A+B)\mathbf{x}\|}{\|\mathbf{x}\|} \leq \max\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} + \max\frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|}$$

or

$$\|A+B\| \leq \|A\| + \|B\|.$$

Theorem If A and B are both $n \times n$ matrices then for any matrix norm

$$\|AB\| \leq \|A\|\|B\|.$$

Proof

$$\|AB\| = \max\frac{\|(AB)\mathbf{x}\|}{\|\mathbf{x}\|} = \max\frac{\|(AB)\mathbf{x}\|}{\|\mathbf{x}\|}\frac{\|B\mathbf{x}\|}{\|B\mathbf{x}\|}, \quad B\mathbf{x} \neq 0$$

$$= \max\frac{\|A(B\mathbf{x})\|}{\|B\mathbf{x}\|}\frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|}.$$

Now,

$$\frac{\|A(B\mathbf{x})\|}{\|B\mathbf{x}\|} \leq \|A\|$$

and

$$\frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|B\|.$$

Hence,

$$\|AB\| \leq \|A\|\|B\|.$$

8.3.3 Evaluation of ℓ_1 and ℓ_∞ Matrix Norms

The definition for $\|A\|$ can be re-phrased to define $\|A\|$ as the maximum of $\|A\mathbf{x}\|$ for all $\mathbf{x} \in R^n$ with $\|\mathbf{x}\| = 1$. Consider $\|A\|_1$ and let $\mathbf{x}$ be any vector with $\|\mathbf{x}\|_1 = 1$, i.e.

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} \mid x_i \mid = 1.$$

Let $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ be the column vectors of A, then

$$\|A\mathbf{x}\|_1 = \|x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + ... + x_n\mathbf{v}_n\|_1 \leq \mid x_1 \mid \|\mathbf{v}_1\|_1 + \mid x_2 \mid \|\mathbf{v}_2\|_1 + ... + \mid x_n \mid \|\mathbf{v}_n\|_1$$

$$\leq (\mid x_1 \mid + \mid x_2 \mid + ... + \mid x_n \mid) \max_i \|\mathbf{v}_i\|_1 \leq \max_i \|\mathbf{v}_i\|_1 \quad .$$

Hence,

$$\|A\|_1 = \max_j \sum_{i=1}^{n} \mid a_{ij} \mid .$$

Now, consider $\|A\|_\infty$ and let $\mathbf{x}$ be any vector with $\|\mathbf{x}\|_\infty = 1$, i.e.

$$\|\mathbf{x}\|_\infty = \max_i \mid x_i \mid = 1.$$

Then,

$$\|A\mathbf{x}\|_\infty = \|x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + ... + x_n\mathbf{v}_n\|_\infty$$

$$\leq \mid x_1 \mid \|\mathbf{v}_1\|_\infty + \mid x_2 \mid \|\mathbf{v}_2\|_\infty + ... + \mid x_n \mid \|\mathbf{v}_n\|_\infty$$

$$\leq \max_i \mid x_i \mid (\|\mathbf{v}_1\|_\infty + \|\mathbf{v}_2\|_\infty + ... + \|\mathbf{v}_n\|_\infty)$$

$$\leq \max_i \mid x_i \mid \max_i \sum_{i=1}^{n} \|\mathbf{v}_i\|_\infty = \max_i \sum_{i=1}^{n} \|\mathbf{v}_i\|_\infty.$$

Hence,

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} \mid a_{ij} \mid .$$

Example Let

$$A = \begin{pmatrix} 2 & 2 & -1 \\ 3 & 1 & -2 \\ -3 & 4 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 2 \\ -1 & 2 & 3 \\ 1 & 3 & -2 \end{pmatrix}.$$

Then $\|A\|_1 = 8$ - the modular sum of column 1.

$\|A\|_\infty = 8$ - the modular sum of row 3.

$\|B\|_1 = 7$ - the modular sum of column 3.

$\|B\|_\infty = 6$ - the modular sum of row 2 or row 3.

Also,

$$AB = \begin{pmatrix} -1 & 1 & 12 \\ 0 & -4 & 13 \\ -8 & 5 & 8 \end{pmatrix}$$

and $\|AB\|_1 = 33$ and $\|AB\|_\infty$=21. Note that for each of these norms

$$\|AB\| \leq \|A\|\|B\|.$$

8.4 The Conditioning of Linear Equations

A fundamental problem in computational linear algebra is to estimate how a perturbation in $\mathbf{b}$ or A or both $\mathbf{b}$ and A will effect the solution $\mathbf{x}$. The solution to this problem is important because if small perturbations in A or $\mathbf{b}$ produce large changes in $\mathbf{x}$ for a given method of solution, then the equations are 'ill-conditioned' and numerical solutions may have significant errors associated with them. For example, consider the system of equations

$$\begin{aligned} x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 &= 1, \\ \frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 &= 0, \\ \frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 &= 0. \end{aligned}$$

The characteristic matrix A of this set of equations is known as a Hilbert matrix. Using Gaussian elimination with natural pivoting and exact arithmetic, we construct the augmented matrix

$$\left(\begin{array}{ccc|c} 1 & \frac{1}{2} & \frac{1}{3} & 1 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & 0 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 0 \end{array}\right)$$

and proceed as follows:

$$\begin{array}{r} R_2 - \frac{1}{2}R_1 \\ R_3 - \frac{1}{3}R_1 \end{array} \left(\begin{array}{ccc|c} 1 & \frac{1}{2} & \frac{1}{3} & 1 \\ 0 & \frac{1}{12} & \frac{1}{12} & -\frac{1}{2} \\ 0 & \frac{1}{12} & \frac{4}{45} & -\frac{1}{3} \end{array}\right) \longrightarrow \begin{array}{r} \\ R_3 - R_2 \end{array} \left(\begin{array}{ccc|c} 1 & \frac{1}{2} & \frac{1}{3} & 1 \\ 0 & \frac{1}{12} & \frac{1}{12} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{180} & \frac{1}{6} \end{array}\right).$$

Back-substituting, the solution vector is

$$\mathbf{x} = (9, -36, 30)^T.$$

Now consider a solution to the same system of equations using exactly the same technique but working to an accuracy of just 2 decimal places. In this case, the augmented matrix is given by

$$\left(\begin{array}{ccc|c} 1.00 & 0.50 & 0.33 & 1.00 \\ 0.50 & 0.33 & 0.25 & 0.00 \\ 0.33 & 0.25 & 0.20 & 0.00 \end{array}\right)$$

and

$$\begin{array}{r} R_2 - 0.50R_1 \\ R_3 - 0.33R_1 \end{array} \left(\begin{array}{ccc|c} 1.00 & 0.50 & 0.33 & 1.00 \\ 0.00 & 0.08 & 0.08 & -0.50 \\ 0.00 & 0.08 & 0.09 & -0.33 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} \\ R_3 - R_2 \end{array} \left(\begin{array}{ccc|c} 1.00 & 0.50 & 0.33 & 1.00 \\ 0.00 & 0.08 & 0.08 & -0.50 \\ 0.00 & 0.00 & 0.01 & 0.17 \end{array}\right).$$

Back-substituting, we have

$$\mathbf{x} = (7.00, -24.00, 17.00)^T$$

using a 2 decimal place calculation which should be compared with $\mathbf{x} = (9, -36, 30)^T$ using exact arithmetic. The 2 decimal place computation has sizable errors. A small perturbation in a_{ij} can therefore yield large changes in the solution vector when computations are performed to a limited number of decimal places as is the case in practice (with a digital computer). The size of the error illustrated here, however, is based on a special type of matrix (the Hilbert matrix) which is relatively highly ill-conditioned and has been used to highlight the problem of numerical instability.

8.4.1 Conditioning of $A\mathbf{x} = \mathbf{b}$

In terms of the conditioning of linear systems, there are three cases of interest:

(i) $\mathbf{b} \to \mathbf{b} + \delta\mathbf{b}$, $\mathbf{x} \to \mathbf{x} + \delta\mathbf{x}$ for which

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond(A)} \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

(ii) $A \to A + \delta A$, $\mathbf{x} \to \mathbf{x} + \delta\mathbf{x}$ for which

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x} + \delta\mathbf{x}\|} \leq \text{cond}(A) \frac{\|\delta A\|}{\|A\|}.$$

(iii) $\mathbf{b} \to \mathbf{b} + \delta\mathbf{b}$, $A \to A + \delta A$, $\mathbf{x} \to \mathbf{x} + \delta\mathbf{x}$ for which

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(A) \left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|} \right)$$

where

$$\text{cond}(A) = \|A\| \|A^{-1}\|$$

which is known as the 'condition number'. The stability of a solution to $A\mathbf{x} = \mathbf{b}$ is determined by the magnitude of the condition number which provides a measure of how ill-conditioned the system is.

Derivation of the Results

All results are based on the application of inequalities associated with vector and matrix norms (i.e. Holder's and Minkowski's inequalities) together with a couple of tricks.

Case 1: Perturbation of Data $\mathbf{b} \to \mathbf{b} + \delta\mathbf{b}$

We note that

$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b},$$

$$A\mathbf{x} + A\delta\mathbf{x} = \mathbf{b} + \delta\mathbf{b},$$

$$A\delta\mathbf{x} = \delta\mathbf{b} \quad \text{since} \quad A\mathbf{x} - \mathbf{b} = 0.$$

$$\therefore \quad \delta\mathbf{x} = A^{-1}\delta\mathbf{b}.$$

Taking norms of both sides

$$\|\delta\mathbf{x}\| = \|A^{-1}\delta\mathbf{b}\| \leq \|A^{-1}\|\|\delta\mathbf{b}\|.$$

Now $\|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|$ giving

$$\|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\|A\|}.$$

Hence,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\|A^{-1}\|\|\delta\mathbf{b}\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

or

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

where

$$\text{cond}(A) = \|A\|\|A^{-1}\|.$$

The magnitude of cond(A) (for a matrix of a given size) determines how much a small change in $\mathbf{b}$ effects the solution where any norm may be used. If cond(A) is large, then the system is likely to be ill-conditioned so that conversely, we require cond(A) to be relatively small to have confidence in the solution. For example, consider the characteristic matrix

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 3 & -1 \\ 2 & 1 & -6 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} -17 & 10 & 4 \\ 4 & -2 & -1 \\ -5 & 3 & 1 \end{pmatrix}.$$

In this case

$$\|A\|_\infty = \max_i \sum_{j=1}^{3} \mid a_{ij} \mid = \mid 2 \mid + \mid 1 \mid + \mid -6 \mid = 9.$$

Similarly,

$$\|A^{-1}\|_\infty = \mid -17 \mid + \mid 10 \mid + \mid 4 \mid = 31.$$

Hence,

$$\text{cond}(A) = 9 \times 31 = 279.$$

Case 2: Perturbation of Matrices $A \to A + \delta A$

In this case,

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b},$$

$$\mathbf{x} + \delta\mathbf{x} = (A + \delta A)^{-1}\mathbf{b},$$

$$\delta\mathbf{x} = (A + \delta A)^{-1}\mathbf{b} + A^{-1}\mathbf{b}, \quad \text{since} \quad A\mathbf{x} = \mathbf{b}.$$

We now use the following result (basic trick)

$$(B^{-1} - A^{-1})\mathbf{b} = A^{-1}(A - B)B^{-1}\mathbf{b}$$

since

$$A^{-1}(A-B)B^{-1}\mathbf{b} = (A^{-1}AB^{-1} - A^{-1}BB^{-1})\mathbf{b} = (B^{-1} - A^{-1})\mathbf{b}.$$

This gives

$$\delta\mathbf{x} = A^{-1}[A-(A+\delta A)](A+\delta A)^{-1}\mathbf{b} = -A^{-1}\delta A(A+\delta A)^{-1}\mathbf{b} = -A^{-1}\delta A(\mathbf{x}+\delta\mathbf{x}).$$

Taking norms of both sides,

$$\|\delta\mathbf{x}\| = \| - A^{-1}\delta A(\mathbf{x}+\delta\mathbf{x})\| \leq \|A^{-1}\|\|\delta A\|\|\mathbf{x}+\delta\mathbf{x}\|.$$

Therefore

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}+\delta\mathbf{x}\|} \leq \text{cond}(A)\frac{\|\delta A\|}{\|A\|}, \quad \text{cond}(A) = \|A\|\|A^{-1}\|.$$

Case 3: Perturbation of Data and Matrices (General Case)

The general case is concerned with the effect on the solution vector when $A \to A+\delta A$ and $\mathbf{b} \to \mathbf{b}+\delta\mathbf{b}$. Here,

$$(A+\delta A)(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}+\delta\mathbf{b},$$

$$A\mathbf{x} + A\delta\mathbf{x} + \delta A\mathbf{x} + \delta A\delta\mathbf{x} = \mathbf{b}+\delta\mathbf{b},$$

$$A\delta\mathbf{x} = \delta\mathbf{b} - \delta A\mathbf{x} - \delta A\delta\mathbf{x}, \quad \text{since} \quad A\mathbf{x}-\mathbf{b} = 0.$$

$$\therefore \quad \delta\mathbf{x} = A^{-1}\delta\mathbf{b} - A^{-1}\delta A\mathbf{x} - A^{-1}\delta A\delta\mathbf{x}.$$

Taking norms of both sides,

$$\|\delta\mathbf{x}\| = \|A^{-1}\delta\mathbf{b} - A^{-1}\delta A\mathbf{x} - A^{-1}\delta A\delta\mathbf{x}\| \leq \|A^{-1}\delta\mathbf{b}\| + \|A^{-1}\delta A\mathbf{x}\| + \|A^{-1}\delta A\delta\mathbf{x}\|$$

$$\leq \|A^{-1}\|\|\delta\mathbf{b}\| + \|A^{-1}\|\|\delta A\|\|\mathbf{x}\| + \|A^{-1}\|\|\delta A\|\|\delta\mathbf{x}\|.$$

Dividing through by $\|\mathbf{x}\|$ and noting that

$$\|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\|A\|}$$

we get

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A\|\|A^{-1}\|\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \|A^{-1}\|\|\delta A\| + \|A^{-1}\|\|\delta A\|\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}.$$

Rearranging, we can write

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}\left(1 - \text{cond}(A)\frac{\|\delta A\|}{\|A\|}\right) \leq \text{cond}(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \text{cond}(A)\frac{\|\delta A\|}{\|A\|}$$

where

$$\text{cond}(A) = \|A\|\|A^{-1}\|$$

giving

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\text{cond}(A)}{1 - \text{cond}(A)\|\delta A\|/\|A\|}\left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|}\right).$$

A further simplification to the above result can be made via the following argument: To have confidence in the solution $A\mathbf{x} = \mathbf{b}$ we require that

$$\text{cond}(A)\frac{\|\delta A\|}{\|A\|} << 1$$

If this is the case, then

$$\frac{1}{1 - \text{cond}(A)\|\delta A\|/\|A\|} \sim 1$$

giving

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(A)\left(\frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|}\right).$$

In this case, cond(A) provides a bound to possible errors in a solution due to perturbations in $\mathbf{b}$ and A. For any of the three cases considered above, if cond(A) is (relatively) large, then the equations are 'ill-conditioned' and if cond(A) is (relatively) small, then the equations are 'well-conditioned'. The term 'relative' used here relates to systems that are known to be ill-conditioned.

8.4.2 Example of an Ill-conditioned System

Consider the Hilbert matrix

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{pmatrix}.$$

For an ℓ_1 norm:

$$\|A\|_1 = \max_j \sum_{i=1}^{n} \mid a_{ij} \mid = \mid 1 \mid + \left|\frac{1}{2}\right| + \left|\frac{1}{3}\right| = \frac{11}{6}$$

and

$$\|A^{-1}\|_1 = \mid -36 \mid + \mid 192 \mid + \mid -180 \mid = 408.$$

Therefore, the condition number of this matrix for an ℓ_1 norm is 748. For an ℓ_∞ norm:

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} \mid a_{ij} \mid = \mid 1 \mid + \left|\frac{1}{2}\right| + \left|\frac{1}{3}\right| = \frac{11}{6}$$

and

$$\|A^{-1}\|_\infty = \mid -36 \mid + \mid 192 \mid + \mid -180 \mid = 408.$$

Therefore, the condition number of this matrix for an ℓ_∞ norm is 748. For the characteristic matrix

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 3 & -1 \\ 2 & 1 & -6 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} -17 & 10 & 4 \\ 4 & -2 & -1 \\ -5 & 3 & 1 \end{pmatrix}$$

the condition number for the ℓ_1 norm is 234 and the condition number for the ℓ_∞ norm is 279. These condition numbers are significantly smaller than the condition

number for a Hilbert matrix of equal size. Linear systems characterized by Hilbert matrices are very unstable as indicated by the magnitude of the condition number. Another well known example involves sets of linear equations characterized by a van der Monde matrix of the type

$$\begin{pmatrix} 1 & t_1 & t_1^2 & t_1^3 & \dots & t_1^n \\ 1 & t_2 & t_2^2 & t_2^3 & \dots & t_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & t_n^3 & \dots & t_n^n \end{pmatrix}.$$

In fact, as a general rule of thumb, any system of equations with a characteristic matrix whose elements vary significantly in value from one part of the matrix to another especially via some geometric progression tend to be 'ill-conditioned'. This situation often occurs with the numerical inversion of certain integral transforms, a notorious example being the inversion of the Laplace transform. Given that

$$F(p) = \int_0^\infty f(t) \exp(-pt) dt$$

we can discretize this result and write

$$F_m = \sum_{n=0}^{N-1} f_n \exp(-p_m t_n) = \sum_{n=0}^{N-1} f_n \exp[-(m\Delta p)(n\Delta t)] = \sum_{n=0}^{N-1} T_{nm} f_n$$

where

$$T_{nm} = t_m^n, \quad t_m = \exp(-m\Delta p \Delta t).$$

Thus, we can write the discrete Laplace transform using matrix form as

$$\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & t_0 & t_0^2 & t_0^3 & \dots & t_0^{N-1} \\ 1 & t_1 & t_1^2 & t_1^3 & \dots & t_1^{N-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{N-1} & t_{N-1}^2 & t_{N-1}^3 & \dots & t_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}.$$

Because the characteristic matrix associated with the discrete Laplace transform is of van der Monde type, numerical methods for computing the inverse Laplace transform are notoriously ill-conditioned. This is another reason why the Fourier transform and its inverse are preferable (provided the problem is non-causal) or the cosine/sin transforms are preferable in the application of problems that are causal (if applicable).

8.5 Iterative Improvement

If the condition number of a linear system of equations is small, then we can use one of the direct methods of solution (e.g. Gaussian elimination with partial pivoting or LU factorization). If the condition number is large so that the solution vector obtained has errors, we can apply the process of 'iterative improvement' (also known as 'successive improvement'). Essentially, iterative improvement is a post-solution modification to

improve the accuracy of the solution. The basic idea behind iterative improvement is to solve $A\mathbf{x} = \mathbf{b}$ using some (direct) method giving some approximation $\mathbf{x}_0$ to the exact solution $\mathbf{x}$. We then construct a residual $\mathbf{r}_0$ associated with $\mathbf{x}_0$ given by

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0.$$

Now, since

$$0 = \mathbf{b} - A\mathbf{x}$$

by subtracting, we get

$$\mathbf{r}_0 = A\mathbf{z}_0, \quad \mathbf{z}_0 = \mathbf{x} - \mathbf{x}_0$$

and can solve $A\mathbf{z}_0 = \mathbf{r}_0$ to get $\mathbf{z}_0$. Now

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0$$

and if $\mathbf{x}_1$ is the exact solution, then

$$A\mathbf{x}_1 = A\mathbf{x}_0 + A\mathbf{z}_0 = A\mathbf{x}_0 + \mathbf{r}_0 = A\mathbf{x}_0 + \mathbf{b} - A\mathbf{x}_0 = \mathbf{b}$$

exactly. If $\mathbf{x}_1$ is not an exact solution, then we can repeat the process, i.e. iterate and find the next residual

$$\mathbf{r}_1 = \mathbf{b} - A\mathbf{x}_1$$

and then solve

$$A\mathbf{z}_1 = \mathbf{r}_1, \quad \mathbf{z}_1 = \mathbf{x} - \mathbf{x}_1$$

giving

$$\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{z}_1$$

where $\mathbf{x}_2$ is a second approximation to $\mathbf{x}$. In general, we use an iterative loop and introduce the following algorithm:

For all $n = 1, 2, 3, ...,$

Step 1: Solve $A\mathbf{x}_n = \mathbf{b}$.

Step 2: Compute $\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n$.

Step 3: Solve $A\mathbf{z}_n = \mathbf{r}_n$ for $\mathbf{z}_n$.

Step 4: Compute $\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{z}_n$.

Step 5: Compute $\mathbf{r}_{n+1} = \mathbf{b} - A\mathbf{x}_{n+1}$ or stop if $\|\mathbf{r}_{n+1}\|$ is sufficiently small.

Each system $A\mathbf{z}_n = \mathbf{r}_n$ has the same matrix A. Therefore we can make use of the multipliers already stored from the first solution via LU factorization. Hence, iterative improvement adds only a moderate amount of computing time to the algorithm once LU factorization has been accomplished. It is essential that the residuals $\mathbf{r}_n$ are computed with higher precision than the rest of the computation. If $A\mathbf{x} = \mathbf{b}$ gives $\mathbf{x}_0$ to single precision, then $\mathbf{r} = \mathbf{b} - A\mathbf{x}_0$ must be computed to double precision.

As an example of iterative improvement, consider the system

$$A\mathbf{x} = \mathbf{b}$$

where

$$A = \begin{pmatrix} 1.1 & 2.3 \\ 2.2 & 3.4 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 5.7 \\ 9.0 \end{pmatrix}$$

which has the exact solution

$$\mathbf{x} = (1, 2)^T.$$

We solve the equations using L_1U factorization and 2 digit precision, i.e.

$$L_1U = \begin{pmatrix} 1.0 & 0.0 \\ 2.0 & 1.0 \end{pmatrix} \begin{pmatrix} 1.1 & 2.3 \\ 0.0 & -1.2 \end{pmatrix}$$

giving

$$\mathbf{x}_0 = (1.6, 1.7)^T.$$

The first iteration involves computing the residual using double precision (i.e. 4 digits), thus,

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 = \begin{pmatrix} 0.030 \\ -0.460 \end{pmatrix}.$$

Solving $A\mathbf{z}_0 = \mathbf{r}_0$ via L_1U factorization (using the same factors) gives

$$\mathbf{z}_0 = \begin{pmatrix} -0.872 \\ 0.430 \end{pmatrix}$$

and

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0 = \begin{pmatrix} 0.828 \\ 2.030 \end{pmatrix}.$$

Iterating again, we compute the residual

$$\mathbf{r}_1 = \mathbf{b} - A\mathbf{x}_1 = \begin{pmatrix} 0.120 \\ 0.276 \end{pmatrix}.$$

Solving $A\mathbf{z}_1 = \mathbf{r}_1$ gives

$$\mathbf{z}_1 = \begin{pmatrix} 0.172 \\ -0.030 \end{pmatrix}$$

and

$$\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{z}_1 = \begin{pmatrix} 1.000 \\ 2.000 \end{pmatrix}.$$

The justification of this technique is based on the following: Suppose the computer to which the method is applied, has an effective precision of p digits and let the first approximation $\mathbf{x}_0$ have d digits correct, then, the residual $\mathbf{r}_0$ has $p - d$ correct digits (initially). The solution to $A\mathbf{z}_0 = \mathbf{r}_0$ should have the same relative precision as the initial solution to $A\mathbf{x}_0 = \mathbf{b}$. $\mathbf{z}_0$ should then have a minimum of d digits correct and $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0$ will then have $2d$ digits correct. Each iteration will correct d digits until full p digit precision is obtained.

In conclusion to this section, the diagram below provides a rough guide to the use of direct methods of solution coupled with iterative improvement and the conditions under which they should be applied.

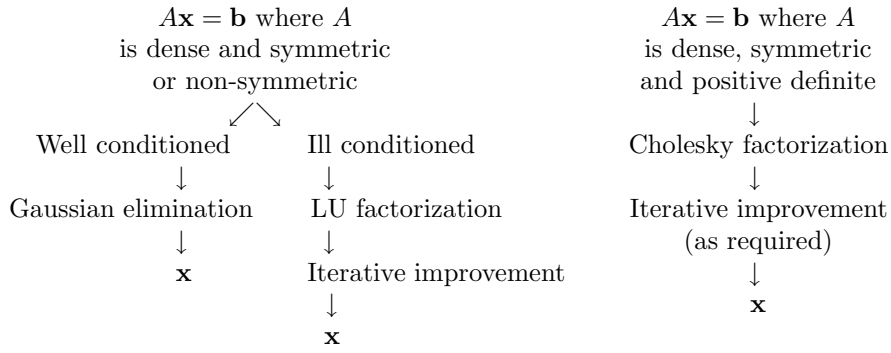


Diagram 8.1: Schematic diagram illustrating the application of direct methods.

8.6 The Least Squares Method

The least squares method and the orthogonality principle are used extensively in signal processing and we shall conclude this chapter with a discussion of this method.

8.6.1 The Least Squares Principle

Suppose we have a real function $f(t)$ which we want to approximate by a function $\hat{f}(t)$. We can construct $\hat{f}$ in such a way that its functional behaviour can be controlled by adjusting the value of a parameter a say. We can then adjust the value of a to find the best estimate $\hat{f}$ of f. So what is the best value of a to choose? To solve this problem, we can construct the mean square error

$$e = \int [f(t) - \hat{f}(t,a)]^2 dt$$

which is a function of a. The value of a which produces the best approximation $\hat{f}$ of f is therefore the one where $e(a)$ is a minimum. Hence, a must be chosen so that

$$\frac{\partial e}{\partial a} = 0.$$

Substituting the expression for e into the above equation and differentiating we obtain

$$\int [f(t) - \hat{f}(t,a)] \frac{\partial}{\partial a} \hat{f}(t,a) dt = 0.$$

Solving this equation for $\hat{f}$ provides the minimum mean square estimate for f. This method is known generally as the least squares method. However, in order to use the least squares method, some sought of model for the estimate $\hat{f}$ must be introduced. There are a number of such models that can be used.

8.6.2 Linear Polynomial Models

Suppose we expand $\hat{f}$ in terms of a linear combination of (known) basis functions $y_n(t)$, i.e.

$$\hat{f}(t) = \sum_n a_n y_n(t)$$

where

$$\sum_n \equiv \sum_{n=-N/2}^{N/2}$$

and where, for simplicity, let us first assume that f is real. Since the basis functions are known, to compute $\hat{f}$, the coefficients a_n must be found. Using the least squares principle, we require a_n such that the mean square error

$$e = \int \left(f(t) - \sum_n a_n y_n(t) \right)^2 dt$$

is a minimum. This occurs when

$$\frac{\partial e}{\partial a_m} = 0 \quad \forall \quad m.$$

Differentiating,

$$\frac{\partial}{\partial a_m} \int \left(f(t) - \sum_n a_n y_n(t) \right)^2 dt$$

$$= 2 \int \left(f(x) - \sum_n a_n y_n(t) \right) \frac{\partial}{\partial a_m} \left(f(t) - \sum_n a_n y_n(t) \right) dt.$$

Noting that

$$\frac{\partial}{\partial a_m} \left(f(t) - \sum_n a_n y_n(t) \right) = -\frac{\partial}{\partial a_m} \sum_n a_n y_n(t)$$

$$= -\frac{\partial}{\partial a_m} \left(... + a_1 y_1(t) + a_2 y_2(t) + ... + a_n y_n(t) + ... \right)$$

$$= -y_1(t), \quad m = 1;$$

$$= -y_2(t), \quad m = 2;$$

$$\vdots$$

$$= -y_m(t), \quad m = n;$$

we have

$$\frac{\partial e}{\partial a_m} = -2 \int \left(f(t) - \sum_n a_n y_n(t) \right) y_m(t) dt = 0.$$

The coefficients a_n which minimize the mean square error for a linear polynomial model are therefore obtained by solving the equation

$$\int f(t) y_m(t) dt = \sum_n a_n \int y_n(t) y_m(t) dt$$

for a_n.

8.6.3 The Orthogonality Principle

The previous result can be written in the form

$$\int \left(f(t) - \sum_n a_n y_n(t) \right) y_m(t) dt = 0$$

which demonstrates that the coefficients a_n are such that the error $f - \hat{f}$ is orthogonal to the basis functions y_m. It is common to write this result in the form

$$\langle f - \hat{f}, y_m \rangle \equiv \int [f(t) - \hat{f}(t)] y_m(t) dt = 0.$$

This is known as the orthogonality principle.

8.6.4 Complex Signals, Norms and Hilbert Spaces

Let us consider the case where f is a complex signal. In this case, $\hat{f}$ must be a complex estimate of this signal and we should also assume that both y_n and a_n (which we shall denote by c_n) are also complex. The mean square error is then given by

$$e = \int \mid f(t) - \sum_n c_n y_n(t) \mid^2 dt.$$

Now, since the operation

$$\left(\int \mid f(t) \mid^2 dt \right)^{1/2}$$

defines the (Euclidean) norm of the function f which shall be denoted by $\| \bullet \|_2$, we can write the mean square error in the form

$$e = \| f(t) - \hat{f}(t) \|_2^2$$

which saves having to write integral signs all the time. The error function above is an example of a 'Hilbert space'. The error function above is a function of the complex coefficients c_n and is a minimum when

$$\frac{\partial e}{\partial c_m^r} = 0$$

and

$$\frac{\partial e}{\partial c_m^i} = 0$$

where

$$c_m^r = \mathrm{Re}[c_m]$$

and

$$c_m^i = \mathrm{Im}[c_m].$$

The above conditions lead to the result

$$\int \left(f(t) - \sum_n c_n y_n(t) \right) y_m^*(t) dt = 0$$

or

$$\langle f - \hat{f}, y_m^* \rangle = 0.$$

This result can be shown as follows:

$$e = \int \mid f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \mid^2 dt$$

$$= \int \left(f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \right) \left(f^*(t) - \sum_n (c_n^r - ic_n^i) y_n^*(t) \right) dt.$$

Now

$$\frac{\partial e}{\partial c_m^r} = \int \left(f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \right) y_m^*(t) dt - \int (f^*(t) - \sum_n (c_n^r - ic_n^i) y_n^*(t) y_m(t) dt = 0$$

and

$$\frac{\partial e}{\partial c_m^i} = i \int \left(f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \right) y_m^*(t) dt - \int \left(f^*(t) - \sum_n (c_n^r - ic_n^i) y_n^*(t) \right) y_m(t) dt = 0$$

or

$$\int \left(f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \right) y_m^*(t) dt - \int \left(f^*(t) - \sum_n (c_n^r - ic_n^i) y_n^*(t) \right) y_m(t) dt = 0.$$

Subtracting these results gives

$$\int \left(f(t) - \sum_n (c_n^r + ic_n^i) y_n(t) \right) y_m^*(t) dt = 0$$

or

$$\int \left(f(t) - \sum_n c_n y_n(t) \right) y_m^*(t) dt = 0.$$

8.6.5 Linear Convolution Models

So far we have demonstrated the least squares principle for approximating a function using a model for the estimate $\hat{f}$ of the form

$$\hat{f}(t) = \sum_n a_n y_n(t).$$

Another important type of model that is used in the least squares method for signal processing and has a number of important applications is the convolution model, i.e.

$$\hat{f}(t) = y(t) \otimes a(t).$$

In this case, the least squares principle can again be used to find the function a. A simple way to show how this can be done, is to demonstrate the technique for digital signals and then use a limiting argument for continuous functions.

Real Digital Signals

If f_i is a real digital signal consisting of a set of numbers $f_1, f_2, f_3, ...$, then we may use a linear convolution model for the discrete estimate $\hat{f}_i$ given by

$$\hat{f}_i = \sum_j y_{i-j} a_j.$$

In this case, using the least squares principle, we find a_i by minimizing the mean square error

$$e = \sum_i (f_i - \hat{f}_i)^2.$$

This error is a minimum when

$$\frac{\partial}{\partial a_k} \sum_i \left(f_i - \sum_j y_{i-j} a_j \right)^2 = 0.$$

Differentiating, we get

$$-2 \sum_i \left(f_i - \sum_j y_{i-j} a_j \right) \frac{\partial}{\partial a_k} \sum_j y_{i-j} a_j = -2 \sum_i \left(f_i - \sum_j y_{i-j} a_j \right) y_{i-k} = 0$$

and rearranging, we have

$$\sum_i f_i y_{i-k} = \sum_i \left(\sum_j y_{i-j} a_j \right) y_{i-k}.$$

The left hand side of this equation is just the discrete correlation of f_i with y_i and the right hand side is a correlation of y_i with

$$\sum_j y_{i-j} a_j$$

which is itself just a discrete convolution of y_i with a_i. Hence, using the appropriate symbols (as discussed in Chapter 4) we can write this equation as

$$f_i \odot y_i = (y_i \otimes a_i) \odot y_i.$$

Real Analogue Signals

With real analogue signals, the optimum function a which minimizes the mean square error

$$e = \int [f(t) - \hat{f}(t)]^2 dt$$

where

$$\hat{f}(t) = a(t) \otimes y(t)$$

is obtained by solving the equation

$$[f(t) - a(t) \otimes y(t)] \odot y(t) = 0.$$

This result is based on extending the result derived above for digital signals to infinite sums and using a limiting argument to integrals.

Complex Digital Signals

If the data are a complex discrete function f_i where f_i corresponds to a set of complex numbers $f_1, f_2, f_3, ...$, then we use the mean square error defined by

$$e = \sum_i \mid f_i - \hat{f}_i \mid^2$$

and a linear convolution model of the form

$$\hat{f}_i = \sum_j y_{i-j} c_j.$$

In this case, the error is a minimum when

$$\frac{\partial e}{\partial c_k} = \sum_i \left(f_i - \sum_j y_{i-j} c_i \right)^* y_{i-k} = 0$$

or

$$f_i \odot y_i^* = (y_i \otimes c_i) \odot y_i^*.$$

Complex Analogue Signals

If $\hat{f}(t)$ is a complex estimate given by

$$\hat{f}(t) = c(t) \otimes y(t)$$

then the function $c(t)$ which minimizes the error

$$e = \|f(t) - \hat{f}(t)\|_2^2$$

is given by solving the equation

$$[f(t) - c(t) \otimes y(t)] \odot y^*(t) = 0.$$

This result is just another version of the orthogonality principle.

Points on Notation

Notice that in the work presented above, the signs $\otimes$ and $\odot$ have been used to denote convolution and correlation respectively for both continuous and discrete data. With discrete signals $\otimes$ and $\odot$ denote convolution and correlation sums respectively. This is indicated by the presence of subscripts on the appropriate functions. If subscripts are not present, then the functions in question should be assumed to be continuous and $\otimes$ and $\odot$ are taken to denote convolution and correlation integrals respectively.

8.7 Summary of Important Results

Vector Norms

$$\|\mathbf{x}\| \equiv \|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} \mid x_i \mid^p \right)^{\frac{1}{p}}, \quad p \geq 1; \quad \|\mathbf{x}\|_\infty = \max_i \mid x_i \mid;$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| \|\mathbf{y}\|, \quad \|\mathbf{x} \cdot \mathbf{y}\| \leq \|\mathbf{x}\| \|\mathbf{y}\|.$$

Matrix Norms

$$\|A\| = \max \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}.$$

$$\|A + B\| \leq \|A\| + \|B\|, \quad \|AB\| \leq \|A\| \|B\|.$$

$$\|A\|_1 = \max_j \sum_{i=1}^{n} \mid a_{ij} \mid, \quad \|A\|_\infty = \max_i \sum_{j=1}^{n} \mid a_{ij} \mid,$$

$$\|A\|_2 = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \mid a_{ij} \mid^2 \right)^{\frac{1}{2}}.$$

Condition Number

$$\text{cond}(A) = \|A\| \|A^{-1}\|.$$

Conditioning of Equations

If $A\mathbf{x} = \mathbf{b}$, then $(A + \delta A)(\mathbf{x} + \delta \mathbf{x}) = (\mathbf{b} + \delta \mathbf{b})$ and

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(A) \left(\frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|} \right), \quad \text{cond}(A) \frac{\|\delta A\|}{\|A\|} << 1.$$

Iterative Improvement

Iterative procedure for improving upon the accuracy of the solution to $A\mathbf{x} = \mathbf{b}$ using LU factorization. The procedure is as follows:

For $n = 1, 2, 3, ...,$

Step 1: Solve $A\mathbf{x}_n = \mathbf{b}$ for $\mathbf{x}_n$ (single precision).

Step 2: Compute $\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n$ (double precision).

Step 3: Solve $A\mathbf{z}_n = \mathbf{r}_n$ for $\mathbf{z}_n$.

Step 4: Compute $\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{z}_n$.

Step 4: Compute $\mathbf{r}_{n+1} = \mathbf{b} - A\mathbf{x}_{n+1}$

Step 5: If $\|\mathbf{r}_{n+1}\| \leq \epsilon$ (the tolerance value) then Stop, else goto Step 3.

The Orthogonality Principle

1. If (linear polynomial model)

$$\hat{f}(t) = \sum_n c_n y_n(t)$$

is an estimate of $f(t)$, then the Hilbert space,

$$e = \|f(t) - \hat{f}(t)\|_2^2 \equiv \int |f(t) - \hat{f}(t)|^2 dt$$

is a minimum when

$$\langle f - \hat{f}, y_m \rangle \equiv \int [f(t) - \hat{f}(t)] y_m^*(t) dt = 0.$$

2. If (linear convolution model)

$$\hat{f}(t) = y(t) \otimes c(t) \equiv \int y(t - \tau) c(\tau) d\tau$$

is an estimate of $f(t)$, then

$$e = \|f(t) - \hat{f}(t)\|_2^2$$

is a minimum when

$$[f(t) - \hat{f}(t)] \odot y^*(t) = 0.$$

8.8 Further Reading

- Wilkinson J H, *Rounding Errors in Algebraic Processes*, Prentice-Hall, 1963.
- Forsyth G E and Moler C B, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, 1967.
- Hamming R W, *Introduction to Applied Numerical Analysis*, McGraw-Hill, 1971.
- Conte S D and de Boor C, *Elementary Numerical Analysis*, McGraw-Hill, 1980.
- Skelton R E, *Dynamic Control Systems*, Wiley, 1988.

8.9 Problems

8.1 Find the condition number cond(A) of the matrix

$$A = \begin{pmatrix} 1 & 3 & 1 \\ 2 & 5 & -3 \\ -1 & -1 & 8 \end{pmatrix}$$

for both $\|A\|_1$ and $\|A\|_\infty$. Obtain exact solutions to

$$A\mathbf{x} = \mathbf{b}; \quad \mathbf{b} = (1 \quad -3 \quad 8)^T$$

using a method of your choice. For $\|A\|_1$, verify that if b_1 is changed from 1 to 1.01, then

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} = \text{cond}(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

8.2 Consider the following linear system

$$A\mathbf{x} = \mathbf{b}$$

where

$$A = \begin{pmatrix} 3.1 & 1.3 & 2.4 \\ 2.0 & 4.1 & 1.2 \\ 1.1 & -2.8 & 1.1 \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} 4.4 \\ 6.1 \\ -1.7 \end{pmatrix}.$$

Solve this system of equations using Doolittles method working to 2 decimal places only and compare your result with the exact solution vector $\mathbf{x} = (1 \quad 0 \quad 0)^T$. Working to double precision, find an improved solution to this system of equations.

8.3 Show that if $\mathbf{r}$ is the residual associated with an approximate solution $\hat{\mathbf{x}}$ of the equation $A\mathbf{x} = \mathbf{b}$, then

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \text{cond}(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

where cond(A) is the condition number of A.

8.4 Prove that

$$\text{cond}(A) \geq \frac{\|B^{-1} - A^{-1}\| \quad \|A\|}{\|A - B\| \quad \|B^{-1}\|}$$

where A and B are matrices of the same order and cond(A) is the condition number of the matrix A.

8.5 Prove Banach's Lemma

$$\|(I + A)^{-1}\| \leq \frac{1}{1 - \|A\|}; \quad \|A\| < 1.$$

Hint: $(I+A)^{-1}(I+A) \equiv I$.

8.6 If $\hat{\mathbf{x}}$ is an approximation of the vector $\mathbf{x}$ which is given by

$$\hat{x}_i = \sum_{j=1}^{n} y_{i-j} a_j, \quad i = 1, 2, ..., n$$

show that the optimum coefficients a_j which minimizes

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$$

may be obtained by solving the equation

$$\sum_{i=1}^{n} \left(x_i - \sum_{j=1}^{n} y_{i-j} a_j \right) y_{i-k} = 0; \quad k = 1, 2, ..., n$$

for a_j.

8.7 Use the approximate Doolittle factorization

$$A = \begin{pmatrix} 4 & 3 & 1 & 2 \\ 1 & 4 & 3 & 1 \\ 2 & 3 & 6 & 1 \\ -2 & 1 & 3 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ .25 & 1 & 0 & 0 \\ .50 & .46 & 1 & 0 \\ -.50 & .77 & .33 & 1 \end{pmatrix} \begin{pmatrix} 4 & 3 & 1 & 2 \\ 0 & 3.30 & 2.70 & .50 \\ 0 & 0 & 4.2 & -.25 \\ 0 & 0 & 0 & 8.7 \end{pmatrix}$$

to solve to 4 significant digit accuracy, the linear equations:

$$4x_1 + 3x_2 + x_3 + 2x_4 = 5,$$

$$x_1 + 4x_2 + 3x_3 + x_4 = 6,$$

$$2x_1 + 3x_2 + 6x_3 + x_4 = 7,$$

$$-2x_1 + x_2 + 3x_3 + 8x_4 = 8.$$

8.8 Find, correct to 2 decimal places only, the approximate Cholesky factorization of

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}.$$

Use this factorization, together with an iterative improvement method, to find the solution of

$$A\mathbf{x} = (5 \quad 6 \quad 7 \quad 8)^T$$

correct to 4 decimal places.

8.9 Working to 2 decimal places only, find a Cholesky factorization of the matrix

$$A = \begin{pmatrix} 10 & 5 & 3.3 & 0 \\ 5 & 3.3 & 2.5 & 2 \\ 3.3 & 2.5 & 4 & 1.7 \\ 0 & 2 & 1.7 & 6.5 \end{pmatrix}.$$

Use this factorization together with iterative improvement to find a solution to the equation

$$A\mathbf{x} = (10 \quad 8 \quad 6 \quad 4)^T$$

which is correct to 4 decimal places.

Chapter 9

Iterative Methods of Solution

Iterative or indirect method are based on reformulating $A\mathbf{x} = \mathbf{b}$ as

$$\mathbf{x}^{(n+1)} = M\mathbf{x}^{(n)} + \mathbf{c}$$

where n refers to the iteration number, M is known as the iteration matrix and $\mathbf{c}$ is a new vector formed by the reformulation process. This approach requires that $\mathbf{x}^{(n)}$ converges to $\mathbf{x}$, the exact solution, i.e.

$$\mathbf{x} = \lim_{n\to\infty} \mathbf{x}^{(n)}$$

Convergence depends on the choice of M and should not depend critically upon the choice of $\mathbf{x}^{(0)}$. Ideally, the choice of $\mathbf{x}^{(0)}$ should be based on *a priori* information on $\mathbf{x}$ but, in practice, we can set $\mathbf{x}^{(0)} = 0$.

Advantages of Iterative Methods

(i) The iteration matrix is virtually unaltered by most methods.

(ii) The methods are generally self correcting.

(iii) The methods are generally simple to program and require little additional storage.

(iv) The methods lend themselves to the application of parallel processing.

Disadvantages of Iterative Methods

(i) In general, there is no guaranty of convergence.

(ii) Even when the method converges, the number of iterations can be large.

(iii) The computing time can be large if many iteration are required and the system of equations is large.

As a general rule of thumb, iterative methods are recommended only for large linear systems of equations $A\mathbf{x} = \mathbf{b}$ when A is a large sparse matrix. Systems of this type,

frequently occur in numerical solutions to partial differential equations using finite difference analysis and (to a lesser extent) finite element analysis. They also occur in cases when the impulse response function of a system is characterized by a matrix which is banded and sparse.

The types of iterative techniques that are discussed in this chapter include:

Jacobi's method,
the Gauss-Seidel method,
the relaxation method,
The conjugate gradient method which is a semi-iterative technique.

The conditions for convergence (which are discussed later on in this chapter) are as follows:

(i) A sufficient condition for convergence is that A is strictly diagonally dominant, i.e.

$$\mid a_{ii} \mid > \sum_{\substack{j=1 \\ j \neq i}}^{n} \mid a_{ij} \mid .$$

(ii) A necessary and sufficient condition for convergence is that

$$\rho(M) < 1$$

where $\rho(M)$ is the 'spectral radius', given by

$$\rho(M) = \max_{i} \mid \lambda_i \mid .$$

Here, λ_i are the eigenvalues of M. The largest eigenvalue of the iteration matrix must therefore be less than unity to ensure convergence.

9.1 Basic Ideas

Consider the following system of equations:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

We can rewrite this set of equations as

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - ... - a_{1n}x_n),$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - ... - a_{2n}x_n),$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - ... - a_{n,n-1}x_{n-1}).$$

Clearly this requires that $a_{ii} \neq 0$, i.e. the diagonal elements are non zero.

9.1.1 Jacobi's Method

Jacobi's method involves the most basic iteration process of the type:

$$x_1^{(k+1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - ... - a_{1n}x_n^{(k)}),$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - ... - a_{2n}x_n^{(k)}),$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k)} - ... - a_{n(n-1)}x_{n-1}^{(k)}).$$

9.1.2 The Gauss-Seidel Method

The Gauss-Seidel method involves updating the sub-diagonal elements as the computation proceeds. The iteration process is

$$x_1^{(k+1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - ... - a_{1n}x_n^{(k)}),$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - ... - a_{2n}x_n^{(k)}),$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k+1)} - ... - a_{n(n-1)}x_{n-1}^{(k+1)}).$$

9.1.3 The Relaxation or Chebyshev Method

The relaxation method involves using a 'relaxation parameter' to 'tune' the system in order to reduce the number of iterations required. The iteration process is

$$x_1^{(k+1)} = x_1^{(k)} + \frac{\omega}{a_{11}}(b_1 - a_{12}x_2^{(k)} - ... - a_{1n}x_n^{(k)} - a_{11}x_1^{(k)}),$$

$$x_2^{(k+1)} = x_2^{(k)} + \frac{\omega}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - ... - a_{2n}x_n^{(k)} - a_{22}x_2^{(k)}),$$

$$\vdots$$

$$x_n^{(k+1)} = x_n^{(k)} + \frac{\omega}{a_{nn}}(b_n - a_{n1}x_1^{(k+1)} - ... - a_{n(n-1)}x_{n-1}^{(k+1)} - a_{nn}x_n^{(k)}).$$

where ω is called the 'relaxation parameter'. The value of this parameter affects (increases) the rate of convergence. There are two cases in which $1 < \omega < 2$ giving the so called 'Successive-Over-Relaxation' or SOR method and the case when $0 < \omega < 1$ known as the 'Successive-Under-Relaxation' or SUR method.

9.2 Iterative Methods

In the previous section the basic ideas were introduced. In this section, we develop the approach further. We start with $\mathbf{b} = A\mathbf{x}$ where A is a $n \times n$ matrix and write this equation in the form

$$b_i = \sum_{j=1}^{n} a_{ij}x_j = \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j + a_{ii}x_i.$$

Rearranging,

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j\right)$$

and iterating,

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right), \quad k = 1, 2, ...$$

where

$$x_i^{(1)} = \frac{b_i}{a_{ii}}$$

which is equivalent to setting $x_i^{(0)} = 0 \forall i$. The above iterative scheme is Jacobi's method. Gauss-Seidel iteration is given by

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

and the relaxation method is compounded in the result

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_i^{(k)} - a_{ii}x_i^{(k)}\right).$$

It is easy to translate the formulae above into code which requires a loop to compute the sums, a loop over i and a loop over k. Note that if $\omega = 1$, then the relaxation method reduces to Gauss-Seidel iteration. For $\omega \neq 1$, the relaxation method can increase the rate of convergence, depending on the characteristics of the system.

9.3 Example of the Iteration Method

Suppose we are required to solve

$$10x_1 - x_2 + x_3 = 20,$$

$$x_1 - 20x_2 - 2x_3 = -16,$$

$$2x_1 - x_2 + 20x_3 = 23.$$

We re-write this system as

$$x_1 = \frac{1}{10}(20 + x_2 - x_3),$$

$$x_2 = \frac{1}{20}(16 + x_1 - 2x_3),$$

$$x_3 = \frac{1}{20}(23 - 2x_1 + x_2).$$

Note that the characteristic matrix A of this system is diagonally dominant and that convergence is likely because $1/a_{ii}$ is small. Applying Jacobi iteration, we have

$$x_1^{(n+1)} = (20 + x_2^{(n)} - x_3^{(n)})/10,$$

$$x_2^{(n+1)} = (16 + x_1^{(n)} - 2x_3^{(n)})/20,$$

$$x_3^{(n+1)} = (23 - 2x_1^{(n)} + x_2^{(n)})/20,$$

and taking $\mathbf{x}^{(0)} = 0$, gives

$$\mathbf{x}^0 = (0, 0, 0)^T,$$

$$\mathbf{x}^{(1)} = (2.0, 0.8, 1.15)^T,$$

$$\mathbf{x}^{(2)} = (1.965, 1.015, 0.990)^T,$$

$$\mathbf{x}^{(3)} = (2.0025, 0.99725, 1.003)^T,$$

Applying Gauss-Seidel iteration, we get

$$x_1^{(n+1)} = (20 + x_2^{(n)} - x_3^{(n)})/10,$$

$$x_2^{(n+1)} = (16 + x_1^{(n+1)} - 2x_3^{(n)})/20,$$

$$x_3^{(n+1)} = (23 - 2x_1^{(n+1)} + x_2^{(n+1)})/20,$$

and with $\mathbf{x}^0 = 0$, we have

$$\mathbf{x}^{(0)} = (0, 0, 0)^T,$$

$$\mathbf{x}^{(1)} = (2.0, 0.9, 0.995)^T,$$

$$\mathbf{x}^{(2)} = (1.995, 0.999475, 1.000024)^T,$$

$$\mathbf{x}^{(3)} = (2.000055, 1.000050, 0.999995)^T.$$

The exact solution vector is $\mathbf{x} = (2, 1, 1)^T$.

9.4 General Formalism

Given $A\mathbf{x} = \mathbf{b}$, we decompose the characteristic matrix A into lower triangular L, diagonal D and upper triangular U matrices, i.e.

$$A = L + D + U$$

or Characteristic Matrix = Lower triangular + Diagonal + Upper triangular.

Then,

$$(L + D + U)\mathbf{x} = \mathbf{b},$$

$$D\mathbf{x} = -L\mathbf{x} - U\mathbf{x} + \mathbf{b},$$

and

$$\mathbf{x} = -D^{-1}L\mathbf{x} - D^{-1}U\mathbf{x} + D^{-1}\mathbf{b}$$

where

$$D^{-1} = \text{diag}(a_{11}^{-1}, a_{22}^{-1}, ..., a_{nn}^{-1}).$$

The iteration methods can then be written as follows:

Jacobi iteration

$$\mathbf{x}^{(n+1)} = -D^{-1}(L + U)\mathbf{x}^{(n)} + D^{-1}\mathbf{b} = M_J\mathbf{x}^{(n)} + \mathbf{c}_J$$

Gauss-Seidel iteration

$$\mathbf{x}^{(n+1)} = -D^{-1}L\mathbf{x}^{(n+1)} - D^{-1}U\mathbf{x}^{(n)} + D^{-1}\mathbf{b}$$

or after rearranging,

$$\mathbf{x}^{(n+1)} = -(D + L)^{-1}U\mathbf{x}^{(n)} + (D + L)^{-1}\mathbf{b} = M_G\mathbf{x}^{(n)} + \mathbf{c}_G$$

Relaxation iteration

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \omega(-D^{-1}L\mathbf{x}^{(n+1)} - D^{-1}U\mathbf{x}^{(n)} + D^{-1}\mathbf{b} - \mathbf{x}^{(n)})$$

which after rearranging gives

$$\mathbf{x}^{(n+1)} = -(D + \omega L)^{-1}[(\omega - 1)D + \omega U]\mathbf{x}^{(n)} + (D + \omega L)^{-1}\omega\mathbf{b} = M_R\mathbf{x}^{(n)} + \mathbf{c}_R$$

These results are compounded in the table below.

Method	**Iteration Matrix** M	Vector $\mathbf{c}$
Jacobi	$-D^{-1}(L + U)$	$D^{-1}\mathbf{b}$
Gauss-Seidel	$-(D + L)^{-1}U$	$(D + L)^{-1}\mathbf{b}$
Relaxation	$-(D + \omega L)^{-1}[(\omega - 1)D + \omega U]$	$\omega(D + \omega L)^{-1}\mathbf{b}$

SOR .v. SUR

When do we use under ($\omega < 1$) and over ($\omega > 1$) relaxation? Experimental results suggest the following:

(i) Use $\omega > 1$ when Gauss-Seidel iteration converges monotonically (most common).

(ii) Use $\omega < 1$ when Gauss-Seidel iteration is oscillatory.

Note that as a general rule of thumb, if A is diagonally dominant, SOR or SUR iteration is only slightly faster than Gauss-Seidel iteration. However, If A is near or not diagonally dominant, then SOR or SUR can provide a noticeable improvement.

9.5 Criterion for Convergence of Iterative Methods

There are two criterion for convergence:

(i) Sufficient condition: A is strictly diagonally dominant.

(ii) Necessary and sufficient condition: The iteration matrix M has eigenvalues λ_i such that

$$\max_i \mid \lambda_i \mid < 1.$$

9.5.1 Diagonal Dominance

The matrix

$$A = \begin{pmatrix} 5 & 2 & -1 \\ 1 & 6 & -3 \\ 2 & 1 & 4 \end{pmatrix}$$

is diagonally dominant because

$$5 > 2 + \mid -1 \mid,$$

$$6 > 1 + \mid -3 \mid,$$

$$4 > 2 + 1$$

and the matrix

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}$$

is not diagonally dominant because

$$1 < 2 + \mid -2 \mid,$$

$$1 < 1 + 1,$$

$$1 < 2 + 2.$$

In general, A is diagonally dominant if

$$\mid a_{ii} \mid > \sum_{\substack{j=1 \\ j \neq i}}^{n} \mid a_{ij} \mid \forall i.$$

9.5.2 Sufficient Condition for Convergence

Let us define the error $\mathbf{e}^{(n)}$ at each iteration (n) as

$$\mathbf{x}^{(n)} = \mathbf{x} + \mathbf{e}^{(n)}$$

where $\mathbf{x}$ is the exact solution. Now, since

$$\mathbf{x}^{(n+1)} = M\mathbf{x}^{(n)} + \mathbf{c}$$

we have

$$\mathbf{x} + \mathbf{e}^{(n+1)} = M(\mathbf{x} + \mathbf{e}^{(n)}) + \mathbf{c}.$$

If this process is appropriate to obtain a solution, then we must have

$$\mathbf{x} = M\mathbf{x} + \mathbf{c}.$$

From the equations above, we get

$$\mathbf{e}^{(n+1)} = M\mathbf{e}^{(n)},$$

i.e.

$$\begin{aligned}
\mathbf{e}^{(1)} &= M\mathbf{e}^{(0)}, \\
\mathbf{e}^{(2)} = M\mathbf{e}^{(1)} &= M(M\mathbf{e}^{(0)}) = M^2\mathbf{e}^{(0)}, \\
\mathbf{e}^{(3)} = M\mathbf{e}^{(2)} &= M(M^2\mathbf{e}^{(0)}) = M^3\mathbf{e}^{(0)}, \\
&\vdots \\
\mathbf{e}^{(n)} &= M^n\mathbf{e}^{(0)}.
\end{aligned}$$

Now, for global convergence, we require that

$$\mathbf{e}^{(n)} \to 0 \quad \text{as} \quad n \to \infty$$

or

$$\lim_{n\to\infty} M^n\mathbf{e}^{(0)} = 0 \quad \forall \quad \mathbf{e}^{(0)}.$$

This will occur if

$$\|M^n\| < 1.$$

But

$$\|M^n\| \leq \|M\|^n$$

and hence, we require that

$$\|M\|^n < 1$$

or

$$\|M\| < 1.$$

Consider Jacobi iteration, where

$$M_J = -D^{-1}(L+U) = -\frac{a_{ij}}{a_{ii}}, \quad j \neq i.$$

For the ℓ_∞ norm

$$\|M_J\|_\infty = \max_i \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{\mid a_{ij} \mid}{\mid a_{ii} \mid} < 1$$

and therefore

$$\mid a_{ii} \mid > \sum_{\substack{j=1 \\ j\neq i}}^{n} \mid a_{ij} \mid .$$

Diagonal dominance is a sufficient but not a necessary condition for convergence. However, in general, the greater the diagonal dominance, the greater the rate of convergence.

9.5.3 Proof of the Necessary Condition for Convergence

For convergence, we require that

$$\lim_{n\to\infty} \mathbf{e}^{(n)} = 0.$$

Let M be the iteration matrix associated with a given process and consider the eigenvalue equation

$$(M - I\lambda_i)\mathbf{v}_i = 0$$

where λ_i are the eigenvalues of M and $\mathbf{v}_i$ are the eigenvectors of M. (N.B. Eigenvalues and eigenvectors are discussed in the following chapter.) Let us now write the error vector $\mathbf{e}^{(0)}$ as

$$\mathbf{e}^{(0)} = \sum_{i=1}^{n} a_i \mathbf{v}_i$$

where a_i are appropriate scalars. From previous results we have

$$\mathbf{e}^{(n)} = M^n \mathbf{e}^{(0)}.$$

Hence,

$$M\mathbf{e}^{(0)} = M \sum_i a_i \mathbf{v}_i = \sum_i a_i M \mathbf{v}_i = \sum_i a_i \lambda_i \mathbf{v}_i$$

since

$$M\mathbf{v}_i = \lambda_i \mathbf{v}_i.$$

Further,

$$M(M\mathbf{e}^{(0)}) = M \sum_i a_i \lambda_i \mathbf{v}_i = \sum_i a_i \lambda_i M \mathbf{v}_i = \sum_i a_i \lambda_i^2 \mathbf{v}_i$$

and by induction,

$$\mathbf{e}^{(n)} = M^n \mathbf{e}^{(0)} = \sum_i a_i \lambda_i^n \mathbf{v}_i$$

Now, we want

$$\mathbf{e}^{(n)} \to 0 \quad \text{as} \quad n \to \infty.$$

But this will only occur if

$$\lambda_i^n \to 0 \quad \text{as} \quad n \to \infty,$$

a condition which requires that

$$\rho(M) < \max_i \mid \lambda_i \mid .$$

The parameter $\rho(M)$ is called the 'spectral radius'. This result leads to the following recipe for establishing the convergence of an iterative process for solving $A\mathbf{x} = \mathbf{b}$:

Given

$$\mathbf{x}^{(n+1)} = M\mathbf{x}^{(n)} + \mathbf{c},$$

(i) find the largest eigenvalue of M;

(ii) if the eigenvalue is < 1 in modulus, then the process converges;

(iii) if the eigenvalue if > 1 in modulus, then the process diverges.

Example Find the spectral radii for M_J and M_G given that

$$A = \begin{pmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{pmatrix}.$$

The iteration matrix for Jacobi's method is

$$M_J = -D^{-1}(L+U) = \begin{pmatrix} 0 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 0 \end{pmatrix}.$$

The eigenvalues of M_J can be found by solving equation (see Chapter 10)

$$\mid M_J - \lambda I \mid = 0 \quad \text{for} \quad \lambda, \quad \text{i.e.}$$

$$\begin{vmatrix} -\lambda & 1/4 & 0 \\ 1/4 & -\lambda & 1/4 \\ 0 & 1/4 & -\lambda \end{vmatrix} = 0$$

or

$$-\lambda \begin{vmatrix} -\lambda & 1/4 \\ 1/4 & -\lambda \end{vmatrix} - \frac{1}{4} \begin{vmatrix} 1/4 & 1/4 \\ 0 & -\lambda \end{vmatrix} = 0.$$

Thus,

$$\lambda \left(\lambda^2 - \frac{1}{16} \right) - \frac{\lambda}{16} = 0$$

and

$$\lambda = 0 \quad \text{or} \quad \pm \frac{1}{\sqrt{8}}.$$

Hence,

$$\rho(M_J) = \max_i \mid \lambda_i \mid = \frac{1}{\sqrt{8}} \simeq 0.3536.$$

The Gauss-Seidel iteration matrix is given by

$$M_G = -(D+L)^{-1}U.$$

Using Jordan's method (see Chapter 7) to find $(D+L)^{-1}$ we get

$$M_G = \begin{pmatrix} 0 & 1/4 & 0 \\ 0 & 1/16 & 1/4 \\ 0 & 1/64 & 1/16 \end{pmatrix}.$$

The eigenvalues of M_G are then obtained by solving

$$\begin{vmatrix} -\lambda & 1/4 & 0 \\ 0 & 1/16-\lambda & 1/4 \\ 0 & 1/64 & 1/16-\lambda \end{vmatrix} = 0$$

giving

$$\lambda = 0, 0 \quad \text{or} \quad \frac{1}{8}.$$

Therefore

$$\rho(M_G) = 0.125.$$

9.5.4 Estimating the Number of Iterations

The smaller $\mid \lambda \mid_{\max}$, the faster $\mathbf{e}^{(n)} \to 0$ as $n \to \infty$. Hence, the magnitude of $\rho(M)$ indicates how fast the iteration processes converges. This observation can be used to estimate the number of iterations. If n is the number of iterations required to obtain a solution vector accurate to d decimal places, then

$$\rho^n \simeq 10^{-d}$$

so that n is given by

$$n \simeq -\frac{d}{\log_{10}\rho}.$$

In the SOR method, the iteration matrix is a function of the relaxation parameter ω. Hence the spectral radius ρ is a functional of ω. The optimum value of ω is the one for which ρ is a minimum. In some simple cases, the optimum value of ω can be computed analytically. However, in practice, it is common to sweep through a range of values of ω (in steps of 0.1 for example) using an appropriate computer program and study the number of iterations required to generate a given solution. The relaxation parameter ω is then set at the value which gives the least number of iterations required to produce a solution with a given accuracy.

9.5.5 The Stein-Rosenberg Theorem

In general, $\rho(M_G) \leq \rho(M_J)$. Therefore Gauss-Seidel iteration usually converges faster than Jacobi iteration for those cases in which both methods of iteration can be applied. The comparisons between Jacobi and Gauss-Seidel iteration are compounded in the Stein-Rosenberg theorem:

If M_J contains no negative elements, then there exists the following four possibilities:

Convergence

(i) $\rho(M_J) = \rho(M_G) = 0$;

(ii) $0 < \rho(M_G) < \rho(M_J) < 1$.

Divergence

(iii) $\rho(M_J) = \rho(M_G) = 1$,

(iv) $1 < \rho(M_J) < \rho(M_G)$.

9.6 The Conjugate Gradient Method

The conjugate gradient method can be used to solve $A\mathbf{x} = \mathbf{b}$ when A is symmetric positive definite. We start by defining the term 'conjugate'. Two vectors $\mathbf{v}$ and $\mathbf{w}$ are conjugate if $\mathbf{v}^T A\mathbf{w} = 0$, $\mathbf{v}^T A\mathbf{v} > 0 \forall \mathbf{v}$. The basic idea is to find a set of linearly independent vectors $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)$ which are conjugate with respect to A, i.e.

$$\mathbf{v}_i^T A\mathbf{v}_j = 0 \quad \text{if} \quad i \neq j.$$

If we expand $\mathbf{x}$ as a linear series of basis vectors

$$\mathbf{x} = \sum_{j=1}^{n} \alpha_j \mathbf{v}_j$$

then the problem is reduced to finding $\alpha_j \forall j$. Now, the equation $A\mathbf{x} = \mathbf{b}$ becomes

$$\sum_{j=1}^{n} \alpha_j A\mathbf{v}_j = \mathbf{b}.$$

Multiplying both sides of this equation by $\mathbf{v}_i^T$ we get

$$\sum_{j=1}^{n} \alpha_j \mathbf{v}_i^T A\mathbf{v}_j = \mathbf{v}_i^T \mathbf{b}.$$

Now, since $\mathbf{v}_i^T A\mathbf{v}_j = 0$ if $i \neq j$, we have

$$\sum_{j=1}^{n} \alpha_j \mathbf{v}_i^T A\mathbf{v}_j = \alpha_i \mathbf{v}_i^T A\mathbf{v}_i = \mathbf{v}_i^T \mathbf{b}.$$

Thus,

$$\alpha_i = \frac{\mathbf{v}_i^T \mathbf{b}}{\mathbf{v}_i^T A \mathbf{v}_i}$$

and the solution becomes

$$\mathbf{x} = \sum_{j=1}^{n} \left(\frac{\mathbf{v}_i^T \mathbf{b}}{\mathbf{v}_i^T A \mathbf{v}_i} \right) \mathbf{v}_i.$$

This solution only requires the computation of $A\mathbf{v}_i$ and the scalar products. Therefore, the computation is easy once the conjugate directions $\mathbf{v}_i$ are known. In principal, the conjugate gradient method is a direct method. However, in practice, an iterative solution is required to find the conjugate directions using a 'Gram-Schmidt' type process.

9.6.1 The Gram-Schmidt Process

Our problem is, given $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)$ of a vector space R^n with inner product denoted by $\langle \ \ \rangle$, find an orthonormal basis $(\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n)$ of R^n. The Gram-Schmidt process is based on the following procedure:

$$\mathbf{w}_1 = \mathbf{v}_1,$$

$$\mathbf{w}_2 = \mathbf{v}_2 - \frac{\langle \mathbf{v}_2, \mathbf{w}_1 \rangle}{\|\mathbf{w}_1\|^2} \mathbf{w}_1,$$

$$\mathbf{w}_3 = \mathbf{v}_3 - \frac{\langle \mathbf{v}_3, \mathbf{w}_2 \rangle}{\|\mathbf{w}_2\|^2} \mathbf{w}_2 - \frac{\langle \mathbf{v}_3, \mathbf{w}_1 \rangle}{\|\mathbf{w}_1\|^2} \mathbf{w}_1,$$

$$\vdots$$

$$\mathbf{w}_n = \mathbf{v}_n - \frac{\langle \mathbf{v}_n, \mathbf{w}_{n-1} \rangle}{\|\mathbf{w}_{n-1}\|^2} \mathbf{w}_{n-1} - ... - \frac{\langle \mathbf{v}_n, \mathbf{w}_2 \rangle}{\|\mathbf{w}_2\|^2} \mathbf{w}_2 - \frac{\langle \mathbf{v}_n, \mathbf{w}_1 \rangle}{\|\mathbf{w}_1\|^2} \mathbf{w}_1,$$

where

$$\langle \mathbf{x}, \mathbf{y} \rangle \equiv \mathbf{x}^T \mathbf{y}, \quad \|\mathbf{x}\| \equiv \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}.$$

The orthonormal basis is then given by

$$\left(\frac{\mathbf{w}_1}{\|\mathbf{w}_1\|}, \frac{\mathbf{w}_2}{\|\mathbf{w}_2\|}, ..., \frac{\mathbf{w}_n}{\|\mathbf{w}_n\|} \right),$$

i.e.

$$\langle \mathbf{w}_i / \|\mathbf{w}_i\|, \mathbf{w}_j / \|\mathbf{w}_j\| \rangle = 0, \quad i \neq j.$$

9.6.2 Example of the Gram-Schmidt Process

Consider

$$\mathbf{v}_1 = (1, 1, 0)^T \quad \text{and} \quad \mathbf{v}_2 = (-1, 0, 1)^T.$$

Then

$$\mathbf{w}_1 = \mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix},$$

$$\mathbf{w}_2 = \mathbf{v}_2 - \frac{\langle \mathbf{v}_2, \mathbf{v}_1 \rangle}{\|\mathbf{w}_1\|^2}\mathbf{w}_1$$

$$= \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} - \frac{(-1)}{2}\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix} = \begin{pmatrix} -1/2 \\ 1/2 \\ 1 \end{pmatrix},$$

$$\mathbf{w}_1^T \mathbf{w}_2 = 0,$$

$$\mathbf{w}_2^T \mathbf{w}_1 = 0.$$

Therefore $\mathbf{w}_1$ and $\mathbf{w}_2$ are orthogonal and the orthonormal basis is

$$\frac{\mathbf{w}_1}{\|\mathbf{w}_1\|} = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix}, \quad \frac{\mathbf{w}_2}{\|\mathbf{w}_2\|} = \frac{2}{3}\begin{pmatrix} -1/2 \\ 1/2 \\ 1 \end{pmatrix}.$$

9.6.3 Practical Algorithm for the Conjugate Gradient Method

To solve $A\mathbf{x} = \mathbf{b}$, we begin with an arbitrary starting vector $\mathbf{x}_0 = (1, 1, ..., 1)^T$ say and compute the residual $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$. We then set $\mathbf{v}_0 = \mathbf{r}_0$ and feed the result into the following iterative loop:

(i) compute coefficients of conjugate gradients

$$\alpha_j = \frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{v}_i^T A \mathbf{v}_i};$$

(ii) update $\mathbf{x}$,

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{v}_i;$$

(iii) compute the new residual

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \alpha_i A \mathbf{v}_i;$$

(iv) compute the next conjugate gradient using the Gram-Schmidt process

$$\mathbf{v}_{i+1} = \mathbf{r}_i + \frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i}\mathbf{v}_i.$$

Important points

1. The conjugate directions are $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$.
2. The matrix A is only used to compute $A\mathbf{v}_i$.
3. The process is stopped when $\|\mathbf{r}_i\| \leq \epsilon$, i.e. the required tolerance.
4. The algorithm works well when A is large and sparse.

We conclude this chapter with a schematic diagram illustrating the principal iterative methods of approach to solving $A\mathbf{x} = \mathbf{b}$ as given below.

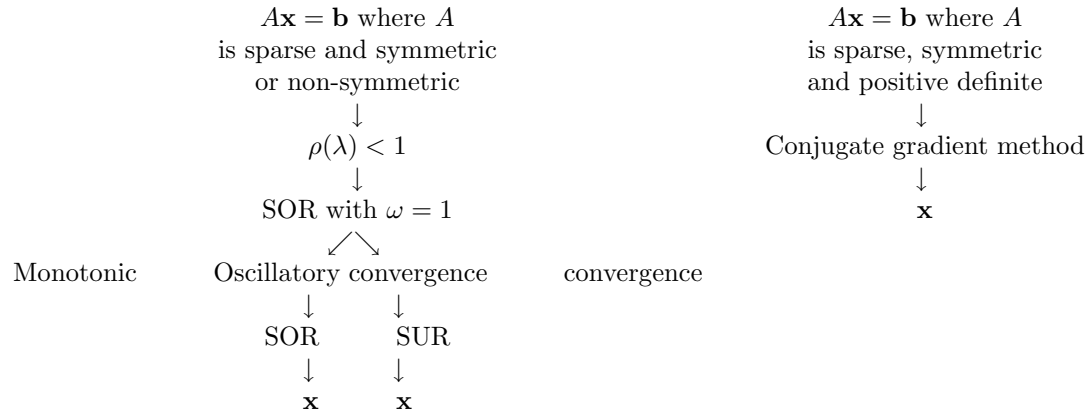


Diagram 9.1: Schematic diagram illustrating the iterative methods

9.7 Summary of Important Results

Iterative Method

$$\mathbf{x}^{(n+1)} = M\mathbf{x}^{(n)} + \mathbf{c}$$

Jacobi Iteration

$$M = -D^{-1}(L+U), \quad \mathbf{c} = D^{-1}\mathbf{b}$$

Gauss-Seidel Iteration

$$M = -(D+L)^{-1}U, \quad \mathbf{c} = (D+L)^{-1}\mathbf{b}$$

Relaxation Iteration

$$M = -(D+\omega L)^{-1}[(\omega-1)D + \omega U], \quad \mathbf{c} = \omega(D+\omega L)^{-1}\mathbf{b}$$

SOR Method

$$1 < \omega < 2$$

SUR Method

$$0 < \omega < 1$$

Condition for Convergence

$$\|M\| << 1$$

Sufficient Condition for Convergence

$$| a_{ii} | > \sum_{\substack{j=1 \\ j \neq i}}^{n} | a_{ij} |$$

Necessary Condition for Convergence

$$\rho(M) < \max_i | \lambda_i |$$

Conjugate Gradient Solution

$$\mathbf{x} = \sum_{j=1}^{n} \left(\frac{\mathbf{v}_i^T \mathbf{b}}{\mathbf{v}_i^T A \mathbf{v}_i} \right) \mathbf{v}_i, \qquad \mathbf{v}_i^T A \mathbf{v}_j = 0 \quad \text{if} \quad i \neq j.$$

9.8 Further Reading

- Traub J F, *Iterative Methods for the Solution of Equations*, Prentice-Hall, 1964.
- Smith G D, *Numerical Solution of Partial Differential Equations*, Oxford University Press, 1978.
- Fogeil M, *The Linear Algebra Problem Solver*, Research and Education Association, 1986.
- Gerald C F and Weatley P O, *Applied Numerical Analysis*, Addison-Wesley, 1989.
- Ciarlet O G, *Introduction to Numerical Linear Algebra and Optimization*, Cambridge University Press, 1989.

9.9 Problems

9.1 (i) Solve the following system of equations using Gauss-Seidel iteration:

$$-2x_1 + x_2 = -1,$$

$$x_1 - 2x_2 + x_3 = 0,$$

$$x_2 - 2x_3 + x_4 = 0,$$

$$x_3 - 2x_4 = 0.$$

Observe the number of iterations that are required for the process to converge, giving a solution that is correct to 2 decimal places.

(ii) Compare the number of iterations using Jacobi and then Gauss-Seidel method that are required to solve $A\mathbf{x} = (2, 4, 6, 8)^T$ where

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}.$$

9.2 Consider the following equations:

$$5x_1 + 2x_2 - x_3 = 6,$$

$$x_1 + 6x_2 - 3x_3 = 4,$$

$$2x_1 + x_2 + 4x_3 = 7.$$

State whether or not the characteristic matrix of this system is diagonally dominant. Compute the Jacobi and Gauss-Seidel iteration matrices for this system and solve these equations using Jacobi iteration and then Gauss-Seidel iteration. Use a 'stopping criterion' given by

$$\max_i \mid x_i^{(n+1)} - x_i^{(n)} \mid \leq \frac{1}{2} \times 10^{-2}.$$

If we wanted to improve the rate of convergence of the Gauss-Seidel process, should we use SOR or SUR. By choosing a suitable value for the relaxation parameter, solve the system of equations above using the relaxation method. Is the rate of convergence significantly improved? Comment on your result.

9.3 Compare the convergence rates of GS and SOR iteration using a relaxation parameter value of 1.6 to solve the equations:

$$-3x_1 + x_2 + 3x_4 = 1,$$

$$x_1 + 6x_2 + x_3 = 1,$$

$$x_2 + 6x_3 + x_4 = 1,$$

$$3x_1 + x_3 - 3x_4 = 1.$$

Use a 'stopping criterion' of 5×10^{-3}. Comment on the result.

9.4 For which of the following system of equations does the (i) Jacobi method and (ii) Gauss-Seidel method converge? Start with the zero vector and use a tolerance of 0.001:

(a)

$$-4x_1 + x_2 = 1,$$

$$x_1 - 4x_2 + x_3 = 1,$$

$$x_2 - 4x_3 + x_4 = 1,$$

$$x_3 - 4x_4 = 1.$$

(b)

$$2x_1 + x_2 + x_3 = 4,$$
$$x_2 + 2x_2 + x_3 = 4,$$
$$x_1 + x_2 + 2x_3 = 4.$$

(c)

$$x_1 + 2x_2 - 2x_3 = 3,$$
$$x_1 + x_2 + x_3 = 0,$$
$$2x_1 + 2x_2 + x_3 = 1.$$

9.5 Find the spectral radii of the Jacobi and Gauss-Seidel iteration matrices corresponding to each system given in question 9.4. Hence, comment on the convergence or divergence in each case.

9.6 Which, if any, of the coefficient matrices in question 9.4 are diagonally dominant? Is diagonal dominance a necessary condition for convergence of the Jacobi and Gauss-Seidel method?

9.7 $\|\mathbf{v}\|$ denotes any norm of R^n and $\|A\|$ is the associated norm for an $n \times n$ matrix A. Prove that if $\rho(A)$ denotes the spectral radius of A, then

$$\rho(A) \leq \|A\|.$$

9.8 Find a set of vectors which are mutually conjugate with respect to the matrix

$$A = \begin{pmatrix} 3 & 1 & -1 \\ 1 & 2 & 1 \\ -1 & 1 & 3 \end{pmatrix}.$$

Hence, solve the equation

$$A\mathbf{x} = (4, 6, 4)^T.$$

9.9 Use the Gram-Schmidt process to solve the equation $A\mathbf{x} = (4 \;\; 4 \;\; 4)^T$ where

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & -1 \\ 0 & -1 & 2 \end{pmatrix}.$$

Chapter 10

Eigenvalues and Eigenvectors

The problem of computing the eigenvalues and eigenvectors occur in the study of systems described by:

(i) homogeneous systems where

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i;$$

(ii) inhomogeneous systems in which

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i + \mathbf{b}$$

where $\mathbf{x}_i$ are the eigenvectors and λ_i are the eigenvalues. To illustrate where such a problem originates, consider the homogeneous wave equation given by

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x,k) = 0$$

which has a general solution

$$u(x,k) = A\cos(kx) + B\sin(kx)$$

where A and B are constants. Suppose we want the solution to the case of standing waves on a string of length L. The boundary conditions in this case are $u(0,k) = 0$ and $u(L,k) = 0$. The first of these conditions implies that $A = 0$. The second conditions implies that either $B = 0$ which gives a trivial solution or that $kL = n\pi$ where n is an integer. The solution is then given by (for unit amplitude)

$$u_n(x,k) = \sin(k_n x)$$

where

$$k_n = \frac{n\pi}{L}.$$

Here, the frequency of oscillation is determined by π/L which defines the eigenvalues k_n - discrete frequencies - and the wavefield is determined by the set of eigenfunctions $u_n(x,k)$. Suppose we now investigate the discrete solution to this problem over an array of size N say, by centre differencing it so that we have

$$u_{n+1} - 2u_n + u_{n-1} + \lambda u_n = 0, \quad n = 1, 2, ..., N$$

where $\lambda = k^2\Delta x^2$ (Δx being the 'step length'). This difference equation defines a set of coupled equations given by

$$-u_2 + 2u_1 - u_0 = \lambda u_1,$$

$$-u_3 + 2u_2 - u_1 = \lambda u_2,$$

$$\vdots$$

$$-u_{N+1} + 2u_N - u_{N-1} = \lambda u_N.$$

If we now introduce the (boundary) conditions $u_0 = 0$ and $u_{N+1} = 0$, then we can write this set of equations in the form

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ & & \vdots & & \ddots & \vdots \\ & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{pmatrix}$$

which is the eigenvalue problem for a characteristic matrix that is tridiagonal. The vector $\mathbf{u} \equiv (u_1, u_2, ..., u_N)$ - known as the eigenvector - is the (discrete) wave-amplitude subject to the boundary conditions imposed which stipulate that the amplitude is zero at the beginning and the end of the array (a wave on a string which is fixed at both ends). The eigenvalue λ describes the frequency with which this wave can oscillate. There may be a set of eigenvalues and eigenvectors that satisfy this system. Physically, they describe the frequencies at which standing waves are supported by the string. In problems of this type, in which the characteristic matrix is tridiagonal, a method known as Sturm sequence iteration can be efficiently applied to compute the eigenvalues and eigenvectors as discussed later on in this chapter.

There is clearly an association between the eigenvalues of such a system and its Fourier representation. For example, if we consider u_n in terms of the Discrete Fourier Transform (see Chapter 3), i.e.

$$u_m = \frac{1}{N}\sum_n U_n \exp(2\pi i n m/N),$$

then the equation

$$u_{m+1} - 2u_m + u_{m-1} + \lambda u_m = 0$$

transforms to

$$\frac{1}{N}\sum_n [\exp(2\pi i n/N) - 2 + \exp(-2\pi i n/N)] U_n \exp(2\pi i n m/N)$$

$$+\frac{\lambda}{N}\sum_n U_n \exp(2\pi i n m/N) = 0$$

from which it is clear that

$$\exp(2\pi i n/N) + \exp(-2\pi i n/N) - 2 + \lambda = 0$$

or

$$\lambda_n = 2[1 - \cos(2\pi n/N)].$$

The eigenvectors are then given by

$$u^n_{m+1} = 2u^n_m - u^n_{m-1} - \lambda_n u^n_m, \quad m = 0, 1, 2, ..., N-1$$

given u^n_0 and u^n_{-1}.

10.1 Formal Methods of Solution

If we consider the homogeneous case, then we can write

$$A\mathbf{x}_i = \lambda_i I \mathbf{x}_i$$

and hence,

$$(A - I\lambda_i)\mathbf{x}_i = 0.$$

The equation $A\mathbf{x} = 0$ is generally not soluble unless $\mid A \mid = 0$. Hence, to compute the eigenvalues we require that

$$\mid A - I\lambda_i \mid = 0.$$

This is the 'characteristic equation' or 'characteristic polynomial' of A because we can write

$$\mid A - I\lambda \mid = \begin{vmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ & a_{22} - \lambda & & \\ & & \ddots & \\ & & & a_{nn} - \lambda \end{vmatrix} = f(\lambda)$$

where $f(\lambda)$ is a polynomial of degree n. Our problem is two-fold. First we need to solve $\mid A - I\lambda_i \mid = 0$ for λ_i, i.e. find the roots of $f(\lambda) = 0$. Having obtained the eigenvalues we then need to solve $(A - I\lambda_i)\mathbf{x}_i = \mathbf{0}$ for $\mathbf{x}_i$ given λ_i.

Example Consider the characteristic matrix

$$A = \begin{pmatrix} -1 & 2 & -2 \\ -2 & 3 & -1 \\ 2 & -2 & 4 \end{pmatrix}.$$

The characteristic equation is

$$\begin{vmatrix} -1-\lambda & 2 & -2 \\ -2 & 3-\lambda & -1 \\ 2 & -2 & 4-\lambda \end{vmatrix} = 0.$$

Factorizing, we get

$$(2 - \lambda)(\lambda - 1)(\lambda - 3) = 0$$

whose roots are $\lambda_1 = 1, \quad \lambda_2 = 2$ and $\lambda_3 = 3$. For $\lambda_1 = 1, \quad \mathbf{x}_1 = (x_1, x_2, x_3)^T$ is given by solution to

$$\begin{pmatrix} -2 & 2 & -2 \\ -2 & 2 & -1 \\ 2 & -2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{0}.$$

Now, the augmented matrix is

$$\left(\begin{array}{ccc|c} -2 & 2 & -2 & 0 \\ -2 & 2 & -1 & 0 \\ 2 & -2 & 3 & 0 \end{array}\right)$$

and using Gaussian elimination we obtain

$$\left(\begin{array}{cccc} -2 & 2 & -2 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right)$$

Therefore $x_3 = 0$ and $x_1 = x_2$ giving $\mathbf{x}_1 = (1, 1, 0)^T$ which is the eigenvector of the eigenvalue λ_1. The same approach can then be used to find the eigenvectors for eigenvalues λ_2 and λ_3.

10.2 Properties of Eigenvalues

The characteristic equation may be written in the form

$$f(\lambda) = \mid A - I\lambda \mid = (-1)^n[\lambda^n - \alpha_{n-1}\lambda^{n-1} + \alpha_{n-2}\lambda^{n-2} - ... + (-1)^n\alpha_0].$$

Let $\lambda_1, \lambda_2, ..., \lambda_n$ be eigenvalues (roots) of the equation $f(\lambda) = 0$. Then

$$f(\lambda) = (\lambda_1 - \lambda)(\lambda_2 - \lambda)(\lambda_3 - \lambda)...(\lambda_n - \lambda).$$

Comparing these equations,

$$\alpha_{n-1} = (\lambda_1 + \lambda_2 + ... + \lambda_n),$$

$$\alpha_{n-2} = \lambda_1\lambda_2 + \lambda_1\lambda_3 + ... + \lambda_1\lambda_n + \lambda_2\lambda_3 + \lambda_2\lambda_4 + ... + \lambda_2\lambda_n + ... + \lambda_{n-1}\lambda_n,$$

$$\alpha_{n-3} = \lambda_1\lambda_2\lambda_3 + ... + \lambda_{n-2}\lambda_{n-1}\lambda_n,$$

$$\vdots$$

$$\alpha_0 = \lambda_1\lambda_2\lambda_3...\lambda_n.$$

Important Results

(i) Let $\lambda = 0$, then $\mid A \mid = \alpha_0 = \lambda_1\lambda_2...\lambda_n$ or

$$\det A = \prod_{i=1}^{n} \lambda_i.$$

(ii) By inspection

$$\alpha_{n-1} = a_{11} + a_{22} + ... + a_{nn} = \lambda_1 + \lambda_2 + ... + \lambda_n$$

or

$$\mathrm{Tr}A \equiv \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i.$$

Orthogonal Properties of Eigenvectors

From the properties of determinants (see Chapter 6), A and A^T have the same eigenvalues. However, A and A^T have different eigenvectors unless A is symmetric. If A is non-symmetric, then

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i \tag{10.1}$$

and

$$A^T \mathbf{y}_i = \lambda_i \mathbf{y}_i. \tag{10.2}$$

Taking the transpose of equation (10.2) gives

$$\mathbf{y}_i^T A = \lambda_i \mathbf{y}_i^T. \tag{10.3}$$

From equation (10.1)

$$\mathbf{y}_j^T A\mathbf{x}_i = \lambda_i \mathbf{y}_j^T \mathbf{x}_i \tag{10.4}$$

and from equation (10.3)

$$\mathbf{y}_i^T A\mathbf{x}_j = \lambda_i \mathbf{y}_i^T \mathbf{x}_j$$

or

$$\mathbf{y}_j^T A\mathbf{x}_i = \lambda_j \mathbf{y}_j^T \mathbf{x}_i. \tag{10.5}$$

Now, subtracting equation (10.5) from equation (10.4) gives

$$(\lambda_j - \lambda_i)\mathbf{y}_j^T \mathbf{x}_i = 0.$$

Hence, if $\lambda_i \neq \lambda_j$ then

$$\mathbf{y}_j^T \mathbf{x}_i = 0.$$

In other words, the eigenvectors are orthogonal.

Normalized Eigenvectors

The eigenvectors

$$(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$$

can be normalized by considering re-scaling them as

$$\left(\frac{\mathbf{x}_1}{\|\mathbf{x}_1\|}, \frac{\mathbf{x}_2}{\|\mathbf{x}_2\|}, ..., \frac{\mathbf{x}_n}{\|\mathbf{x}_n\|} \right)$$

where

$$\|\mathbf{x}_i\| = \sqrt{\langle \mathbf{x}_i, \mathbf{x}_i \rangle}$$

Linearly Dependent Eigenvectors

If there exists a relationship

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n = 0 \tag{10.6}$$

where the coefficients c_i are not all zero, then the eigenvectors are linearly dependent.

Linearly Independent Eigenvectors

If equation (10.6) above can only be satisfied when $c_i = 0 \forall i$ then the eigenvectors are linearly independent.

Theorem If the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ of an $n \times n$ matrix A are all distinct, then the corresponding eigenvectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ are linearly independent.

Proof If $\mathbf{x}_i$ are linearly independent, then

$$c_1\mathbf{x}_1 = -(c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n)$$

and

$$A(c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n) = -c_1A\mathbf{x}_1 = -c_1\lambda_1\mathbf{x}_1 = \lambda_1(c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n).$$
$$\implies c_2\lambda_2\mathbf{x}_2 + ... + c_n\lambda_n\mathbf{x}_n = c_2\lambda_1\mathbf{x}_2 + ... + c_n\lambda_1\mathbf{x}_n$$

or

$$c_2(\lambda_2 - \lambda_1)\mathbf{x}_2 + c_3(\lambda_3 - \lambda_1)\mathbf{x}_3 + ... + c_n(\lambda_n - \lambda_1)\mathbf{x}_n = 0.$$

The effect is that $\mathbf{x}_1$ is eliminated. Further,

$$c_2(\lambda_2 - \lambda_1)\mathbf{x}_2 = -[c_3(\lambda_3 - \lambda_1)\mathbf{x}_3 + ... + c_n(\lambda_n - \lambda_1)\mathbf{x}_n] = 0$$

and

$$Ac_2(\lambda_2 - \lambda_1)\mathbf{x}_2 = c_2(\lambda_2 - \lambda_1)\lambda_2\mathbf{x}_2 = -[c_3(\lambda_3 - \lambda_1)\lambda_2\mathbf{x}_3 + ... + c_n(\lambda_n - \lambda_1)\lambda_2\mathbf{x}_n).$$
$$\implies [c_3(\lambda_3-\lambda_1)\lambda_2\mathbf{x}_3+...+c_n(\lambda_n-\lambda_1)\lambda_2\mathbf{x}_n] = c_3(\lambda_3-\lambda_1)\lambda_3\mathbf{x}_3+...+c_n(\lambda_n-\lambda_1)\lambda_n\mathbf{x}_n$$

or

$$c_3(\lambda_3 - \lambda_1)(\lambda_3 - \lambda_2)\mathbf{x}_3 + ... + c_n(\lambda_n - \lambda_1)(\lambda_n - \lambda_2)\mathbf{x}_n = 0$$

which eliminates $\mathbf{x}_2$. Eliminating $\mathbf{x}_3$ using the same method gives

$$c_4(\lambda_4 - \lambda_1)(\lambda_4 - \lambda_2)(\lambda_4 - \lambda_3)\mathbf{x}_4 + ... + c_n(\lambda_n - \lambda_1)(\lambda_n - \lambda_2)(\lambda_n - \lambda_3)\mathbf{x}_n = 0$$

Hence, by induction, if we repeat the same method n times we will get

$$c_n(\lambda_n - \lambda_1)(\lambda_n - \lambda_2)(\lambda_n - \lambda_3)...(\lambda_n - \lambda_{n-1})\mathbf{x}_n = 0.$$
$$\implies c_n = 0 \quad \text{if} \quad \lambda_i \quad \text{are} \quad \text{distinct} \quad \forall i.$$

By a similar argument

$$c_1 = c_2 = c_3 = ... = c_i = 0$$

and therefore $\mathbf{x}_i$ are linearly independent.

Diagonalization of Matrices

The matrix A is diagonalizable if matrix T exists such that

$$T^{-1}AT = \begin{pmatrix} b_{11} & & & \\ & b_{22} & & \\ & & \ddots & \\ & & & b_{nn} \end{pmatrix}.$$

Theorem An n^{th} order matrix is diagonalizable if it possesses a complete set of linearly independent eigenvectors.

Proof Let $\mathbf{x}_i$ be an eigenvector of A with eigenvalues λ_i. Consider a 'model matrix' $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ then

$$AX = A(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n) = (A\mathbf{x}_1, A\mathbf{x}_2, ..., A\mathbf{x}_n) = (\lambda_1\mathbf{x}_1, \lambda_2\mathbf{x}_2, ..., \lambda\mathbf{x}_n).$$

Hence,

$$AX = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n) \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

and thus,

$$X^{-1}AX = \Lambda \equiv \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n).$$

Integral Powers of A

Theorem If n is a positive integer, then A^n has the same system of eigenvectors as A but its eigenvalues are λ_i^n.

Proof Consider $A\mathbf{x}_i = \lambda_i\mathbf{x}_i$. Then

$$A^n\mathbf{x}_i = A^{n-1}A\mathbf{x}_i = A^{n-1}\lambda_i\mathbf{x}_i = \lambda_i A^{n-1}\mathbf{x}_i.$$

Repeating the process,

$$\lambda_i A^{n-1}\mathbf{x}_i = \lambda_i A^{n-2}A\mathbf{x}_i = \lambda_i A^{n-2}\lambda_i\mathbf{x}_i = \lambda_i^2 A^{n-2}\mathbf{x}_i$$

and

$$\lambda_i^2 A^{n-2}\mathbf{x}_i = ...$$

$$\vdots$$

$$... = \lambda_i^n\mathbf{x}_i$$

Hence,

$$A^n\mathbf{x}_i = \lambda_i^n\mathbf{x}_i.$$

Theorem If n is a positive integer, then $A^n = X\Lambda^n X^{-1}$.

Proof

$$A = X\Lambda X^{-1},$$
$$A^2 = (X\Lambda X^{-1})(X\Lambda X^{-1}) = X\Lambda^2 X^{-1},$$
$$A^3 = (X\Lambda X^{-1})(X\Lambda^2 X^{-1}) = X\Lambda^3 X^{-1},$$
$$\vdots$$

$$A^n = X\Lambda^n X^{-1}.$$

Example Given

$$A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$$

compute A^8. The characteristic equation is

$$| A - \lambda I | = \begin{vmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{vmatrix} = (\lambda - 2)(\lambda - 5) = 0$$

and the eigenvalues are 2 and 5. For $\lambda = 2$:

$$\begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

so that

$$2x_1 + x_2 = 0 \quad \text{twice},$$
$$x_1 = -\frac{1}{2}x_2.$$

For the normalized eigenvector, $x_1^2 + x_2^2 = 1$ which gives

$$x_1 = -\frac{1}{\sqrt{5}}, \quad x_2 = \frac{2}{\sqrt{5}}.$$

For $\lambda = 5$:

$$\begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

and

$$x_1 = x_2 \quad \text{twice}.$$

Normalising, we get

$$x_1 = \frac{1}{\sqrt{2}}, \quad x_2 = \frac{1}{\sqrt{2}}.$$

Then,

$$\mathbf{x}_1 = \begin{pmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix},$$

the model matrix is

$$X = (\mathbf{x}_1, \mathbf{x}_2) = \begin{pmatrix} -1/\sqrt{5} & 1/\sqrt{2} \\ 2/\sqrt{5} & 1/\sqrt{2} \end{pmatrix}$$

and

$$X^{-1} = \begin{pmatrix} -\sqrt{5}/3 & \sqrt{5}/3 \\ 2\sqrt{2}/3 & \sqrt{2}/3 \end{pmatrix}.$$

Now,

$$X^{-1}AX = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix} = \Lambda$$

and

$$A^8 = X\Lambda^8 X^{-1}.$$

Thus,

$$\lambda^8 = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}^8 = \begin{pmatrix} 2^8 & 0 \\ 0 & 5^8 \end{pmatrix} = \begin{pmatrix} 256 & 0 \\ 0 & 390625 \end{pmatrix}$$

so that

$$A^8 = \begin{pmatrix} -1/\sqrt{5} & 1/\sqrt{2} \\ 2/\sqrt{5} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 256 & 0 \\ 0 & 390625 \end{pmatrix} \begin{pmatrix} -\sqrt{5}/3 & \sqrt{5}/3 \\ 2\sqrt{2}/3 & \sqrt{2}/3 \end{pmatrix}$$

$$= \begin{pmatrix} 260502 & 130123 \\ 260246 & 130379 \end{pmatrix}.$$

Important Points

Non-symmetric matrices with different eigenvalues are diagonalizable. For symmetric matrices

$$\lambda = X^{-1}AX = X^T AX$$

since $X^T X = I$ and $X^T = X^{-1}$. For Hermitian matrices

$$\Lambda = X^{-1}AX = X^\dagger AX$$

since $X^\dagger X = I$ or $X^\dagger = X^{-1}$ where $X^\dagger = (X^*)^T$.

10.3 The Cayley-Hamilton Theorem

The Cayley-Hamilton theorem states that a a square matrix satisfies its own characteristic equation. Thus, if

$$f(\lambda) = \mid A - \lambda I \mid$$

then

$$f(A) = 0.$$

Example Consider

$$A = \ left(\begin{matrix} 0 & -2 \\ 1 & -3 \end{matrix}$$

then

$$f(\lambda) = \begin{vmatrix} -\lambda & -2 \\ 1 & -3-\lambda \end{vmatrix} = \lambda^2 + 3\lambda + 2 = 0.$$

From the Cayley-Hamilton theorem

$$f(A) = A^2 + 3A + 2I = 0.$$

Now,

$$A^2 = \begin{pmatrix} -2 & 6 \\ -3 & 7 \end{pmatrix}, \quad 3A = \begin{pmatrix} 0 & -6 \\ 3 & -9 \end{pmatrix}, \quad 2I = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

and hence,

$$f(A) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Proof of the Cayley-Hamilton Theorem

Let

$$f(\lambda) = (-1)^n[\lambda^n - \alpha_{n-1}\lambda^{n-1} + \alpha_{n-2}\lambda^{n-2} - ... + (-1)^n\alpha_0] = 0$$

and

$$f(A) = (-1)^n[A^n - \alpha_{n-1}A^{n-1} + \alpha_{n-2}A^{n-2} - ... + (-1)^n\alpha_0].$$

Now, does $f(A) = 0$? Since

$$A\mathbf{x}_i = \lambda_i\mathbf{x}_i$$

and

$$A^n\mathbf{x}_i = \lambda_i^n\mathbf{x}_i,$$

$$\begin{aligned} f(A)\mathbf{x}_i &= (-1)^n[A^n\mathbf{x}_i - \alpha_{n-1}A^{n-1}\mathbf{x}_i + \alpha_{n-2}A^{n-2}\mathbf{x}_i - ... + (-1)^n\alpha_0\mathbf{x}_i] \\ &= (-1)^n[\lambda^n\mathbf{x}_i - \alpha_{n-1}\lambda^{n-1}\mathbf{x}_i + \alpha_{n-2}\lambda^{n-2}\mathbf{x}_i - ... + (-1)^n\alpha_0\mathbf{x}_i] \\ &= (-1)^n[\lambda^n - \alpha_{n-1}\lambda^{n-1} + \alpha_{n-2}\lambda^{n-2} - ... + (-1)^n\alpha_0]\mathbf{x}_i = f(\lambda)\mathbf{x}_i = 0. \end{aligned}$$

This result shows that

$$[f(A)\mathbf{x}_1, f(A)\mathbf{x}_2, ..., f(A)\mathbf{x}_n] = 0$$

or

$$f(A)U = 0, \quad U = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n).$$

Now $\mathbf{x}_i$ are linearly independent and therefore U^{-1} exists. Thus,

$$f(A)UU^{-1} = 0$$

or

$$f(A)I = f(A) = 0.$$

Applications of the Cayley-Hamilton Theorem

From the Cayley-Hamilton theorem,

$$(-1)^n[A^n - \alpha_{n-1}A^{n-1} + \alpha_{n-2}A^{n-2} - ... + (-1)^n\alpha_0] = 0.$$

Pre-multiplying both sides of the equation above by A^{-1} gives

$$A^{-1}A^n - \alpha_{n-1}A^{-1}A^{n-1} + \alpha_{n-2}A^{-1}A^{n-2} - ... + (-1)^n\alpha_0A^{-1} = 0$$

or

$$A^{n-1} - \alpha_{n-1}A^{n-2} + \alpha_{n-2}A^{n-3} - ... + (-1)^n\alpha_0A^{-1} = 0.$$

Rearranging

$$A^{-1} = \frac{(-1)^n}{\alpha_0}(-A^{n-1} + \alpha_{n-1}A^{n-2} - \alpha_{n-2}A^{n-3} + ...).$$

There are some practical problems associated with this approach to computing A^{-1} since the powers of A must be formed and stored and $\sim n^4$ multiplications are needed for an $n \times n$ matrix.

Example Consider

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 1 & 0 & 1 \\ 2 & 2 & 3 \end{pmatrix}.$$

Then

$$| A - I\lambda | = \begin{vmatrix} 1-\lambda & 2 & -1 \\ 1 & -\lambda & 1 \\ 2 & 2 & 3-\lambda \end{vmatrix} = -\lambda^3 + 4\lambda^2 - \lambda - 6.$$

The characteristic equation is

$$\lambda^3 - 4\lambda^2 + \lambda + 6 = 0.$$

From the Cayley-Hamilton theorem, we have

$$A^3 - 4A^2 + A + 6I = 0,$$

$$A^2 - 4A + I + 6A^{-1} = 0,$$

$$A^{-1} = \frac{1}{6}(-A^2 + 4A - I),$$

$$A^2 = \begin{pmatrix} 1 & 0 & -2 \\ 3 & 4 & 2 \\ 10 & 10 & 9 \end{pmatrix}.$$

Hence,

$$A^{-1} = \frac{1}{6}\begin{pmatrix} 2 & 8 & -2 \\ 1 & -5 & 2 \\ -2 & -2 & 2 \end{pmatrix}.$$

10.4 The Power Method

The power method is used to find the eigenvalue (and eigenvector) of largest modulus. Suppose that the eigenvectors are $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$. Also, consider an arbitrary initial approximation $\mathbf{x}_0$ (any vector). Let

$$\mathbf{x}_0 = \sum_{i=1}^{N} \alpha_i \mathbf{v}_i.$$

Then

$$A^n \mathbf{x}_0 = \sum_{i=1}^{N} \alpha_i A^n \mathbf{v}_i = \sum_{i=1}^{N} \alpha_i \lambda_i^n \mathbf{v}_i$$

because

$$A^n \mathbf{v}_i = \lambda_i^n \mathbf{v}_i.$$

Now, if λ_1 is the largest eigenvalue, then

$$\frac{A^n\mathbf{x}_0}{\lambda_1^n} = \sum_{i=1}^{N} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^n \mathbf{v}_i = \alpha_1\mathbf{v}_1 + \sum_{i=2}^{N} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^n \mathbf{v}_i = \alpha_1\mathbf{v}_1 \quad \text{as} \quad n \to \infty$$

since

$$\frac{\lambda_i}{\lambda_1} < 1 \quad \forall \quad 2 \leq i \leq N.$$

For large n

$$\frac{A^n\mathbf{x}_0}{\lambda_1^n} = \alpha_1\mathbf{v}_1$$

and

$$\frac{A^{n+1}\mathbf{x}_0}{\lambda_1^{n+1}} = \alpha_1\mathbf{v}_1.$$

If we now choose some vector $\mathbf{y}$ not orthogonal to $\mathbf{v}_i$, then

$$\frac{1}{\lambda_1^n}(A^n\mathbf{x}_0 \cdot \mathbf{y}) = \alpha_1\mathbf{v}_1 \cdot \mathbf{y}$$

and

$$\frac{1}{\lambda_1^{n+1}}(A^{n+1}\mathbf{x}_0 \cdot \mathbf{y}) = \alpha_1\mathbf{v}_1 \cdot \mathbf{y}.$$

Hence,

$$\frac{1}{\lambda_1^{n+1}}(A^{n+1}\mathbf{x}_0 \cdot \mathbf{y}) = \frac{1}{\lambda_1^n}(A^n\mathbf{x}_0 \cdot \mathbf{y}) \neq 0$$

or

$$\frac{A^{n+1}\mathbf{x}_0 \cdot \mathbf{y}}{A^n\mathbf{x}_0 \cdot \mathbf{y}} = \frac{\lambda_1^{n+1}}{\lambda_1^n} = \lambda_1.$$

If we now choose $\mathbf{y} = A^n\mathbf{x}_0$, then

$$\lambda_1 = \frac{A^{n+1}\mathbf{x}_0 \cdot A^n\mathbf{x}_0}{A^n\mathbf{x}_0 \cdot A^n\mathbf{x}_0} = \frac{A(A^n\mathbf{x}_0) \cdot A^n\mathbf{x}_0}{A^n\mathbf{x}_0 \cdot A^n\mathbf{x}_0}.$$

Eigenvector Associated with λ_1

We know that

$$\lambda_1 = \frac{A(A^n\mathbf{x}_0) \cdot A^n\mathbf{x}_0}{A^n\mathbf{x}_0 \cdot A^n\mathbf{x}_0}.$$

But $A\mathbf{v}_i = \lambda\mathbf{v}_i$ or

$$\lambda = \frac{A\mathbf{v}_i \cdot \mathbf{v}_i}{\mathbf{v}_i \cdot \mathbf{v}_i}.$$

Thus, $A^n\mathbf{x}_0 \equiv \mathbf{v}_1$ must be the approximate eigenvector corresponding to λ_1. The quotient

$$\frac{A\mathbf{v}_1 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}$$

is called the 'Rayleigh quotient'.

Example of the Power Method

Compute the dominant eigenvalue and associated eigenvector of

$$A = \begin{pmatrix} -7 & 2 \\ 8 & -1 \end{pmatrix}.$$

noting that the exact results are $\lambda = -9$ and $\mathbf{x} = (-1, 1)^T$. We start with $\mathbf{x}_0 = (1, 1)^T$. Then

$$A\mathbf{x}_0 = \begin{pmatrix} -5 \\ 7 \end{pmatrix},$$

$$A^2\mathbf{x}_0 = \begin{pmatrix} 49 \\ -47 \end{pmatrix},$$

$$A^3\mathbf{x}_0 = \begin{pmatrix} -437 \\ 439 \end{pmatrix}$$

and

$$\lambda_1 \simeq \frac{A^3\mathbf{x}_0 \cdot A^2\mathbf{x}_0}{A^2\mathbf{x}_0 \cdot A^2\mathbf{x}_0} = -\frac{42046}{4610} = -9.121.$$

Overflow

In the previous example, relatively large numbers were produced. If large powers of A are computed then very large numbers start to accumulate which may exceed the storage facility of the computer. This is called 'overflow'. Overflow can be avoided by scaling, i.e. dividing the vectors by their maximum modulo component. Repeating the previous example with scaling, we have:

$$A\mathbf{x}_0 = \begin{pmatrix} -5 \\ 7 \end{pmatrix} \quad \text{or with scaling,} \quad \frac{1}{7}\begin{pmatrix} -5 \\ 7 \end{pmatrix} = \begin{pmatrix} -0.714 \\ 1 \end{pmatrix},$$

$$A^2\mathbf{x}_0 = \begin{pmatrix} -7 & 2 \\ 8 & -1 \end{pmatrix}\begin{pmatrix} -0.714 \\ 1 \end{pmatrix} = \begin{pmatrix} 6.998 \\ -6.712 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 \\ -0.959 \end{pmatrix} \quad \text{with scaling,}$$

$$A^3\mathbf{x}_0 = \begin{pmatrix} -7 & 2 \\ 8 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ -0.959 \end{pmatrix} = \begin{pmatrix} 8.918 \\ 8.959 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} -0.995 \\ 1 \end{pmatrix} \quad \text{with scaling.}$$

Then

$$\lambda_1 = \frac{A\mathbf{v}_1 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}$$

where $\mathbf{v}_1 = (-1, 1)^T$ giving a value of -9 for the dominant eigenvalue.

10.4.1 Basic Algorithm for the Power Method

The basic steps required in developing an algorithm for the power method are as follows:

Step 1: Compute $\mathbf{y}_i = A\mathbf{x}_i$ where $\mathbf{x}_0$ is arbitrary but usually set to $(1, 1, ..., 1)^T$.

Step 2: Evaluate $\mathbf{x}_{i+1} = \mathbf{y}_i / \max \mid \mathbf{y}_i \mid$, i.e. scale eigenvector to prevent overflow.

Step 3: Terminate the process when $\|\mathbf{x}_{i+1} - \mathbf{x}_i\| \leq \epsilon$ where ϵ is some user defined tolerance.

Step 4: $\mathbf{x}_{i+1}$ is the dominant eigenvector $\equiv \mathbf{v}_1$.

Step 5: The dominant eigenvalue is then given by computing

$$\lambda_1 = \frac{A\mathbf{v}_1 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}.$$

Note that $(\lambda_1, \mathbf{v}_1)$ are sometimes referred to as the 'dominant eigenpair'.

10.4.2 Problems Concerning the Power Method

The power method is not applicable when A does not have a dominant eigenvalue or when λ_2 is just less than λ_1. When λ_2/λ_1 is just less than 1, $(\lambda_2/\lambda_1)^n \to 0$ as $n \to \infty$ slowly. Therefore many iterations are required. If the entries of A contain significant errors, powers A^n of A have significant round-off errors in their entries. Therefore errors accumulate as the iteration proceeds. Hence, the basic rule of thumb for using the Power Method is:

1. Try the power method and if

$$\frac{(A\mathbf{x}_i) \cdot \mathbf{x}_i}{\mathbf{x}_i \cdot \mathbf{x}_i} \quad \text{with} \quad \mathbf{x}_0 = (1, 1, ..., 1)^T$$

approaches a single number λ_1 goto 2.

2. Check whether

$$A\mathbf{v}_1 = \lambda_1 \mathbf{v}_1.$$

3. If 2 checks out, then accept $(\lambda_1, \mathbf{v}_1)$ as the dominant eigenpair.

10.4.3 Deflation

Deflation of a matrix is based on 2 Lemmas:

Lemma 1. If a matrix A has eigenvalues λ_i corresponding to eigenvectors $\mathbf{x}_i$, then $P^{-1}AP$ has the same eigenvalues as A but with eigenvectors $P^{-1}\mathbf{x}_i$ for any nonsingular matrix P.

Proof The eigenvalue equation is $A\mathbf{x}_i = \lambda_i \mathbf{x}_i$, thus

$$(P^{-1}AP)P^{-1}\mathbf{x}_i = P^{-1}A(PP^{-1})\mathbf{x}_i = P^{-1}A\mathbf{x}_i = P^{-1}(\lambda_i \mathbf{x}_i) = \lambda_i(P^{-1}\mathbf{x}_i).$$

The matrices A and $P^{-1}AP$ are said to be similar because they have the same eigenvalues. The transform

$$A \to P^{-1}AP$$

is called a similarity transform. Many numerical techniques for computing the eigenvalues of a matrix are based on some type of similarity transform.

Lemma 2. Let

$$B = \begin{pmatrix} \lambda_1 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & c_{22} & c_{23} & \dots & c_{2n} \\ 0 & c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_{n2} & c_{n3} & \dots & c_{nn} \end{pmatrix}$$

and let C be the $(n-1)\times(n-1)$ matrix obtained by deleting the first row and column of B. Then:

(i) The matrix B has eigenvalues λ_1 together with the $n-1$ eigenvalues of C.

(ii) If $(\beta_2, \beta_3, ..., \beta_n)^T$ is an eigenvector of C with eigenvalue $\mu \neq \lambda_1$, then the corresponding eigenvector of B is $(\beta_1, \beta_2, ..., \beta_n)^T$ where

$$\beta_1 = \frac{\sum_{j=2}^{n} a_{1j}\beta_j}{\mu - \lambda_1}.$$

Proof

The characteristic polynomial of B is

$$\begin{vmatrix} \lambda_1 - \lambda & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & c_{22} - \lambda & c_{23} & \dots & c_{2n} \\ 0 & c_{32} & c_{33} - \lambda & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_{n2} & c_{n3} & \dots & c_{nn} - \lambda \end{vmatrix}.$$

Expanding this determinant down, the first column gives

$$(\lambda_1 - \lambda) \begin{vmatrix} c_{22} - \lambda & c_{23} & \dots & c_{2n} \\ c_{32} & c_{33} - \lambda & \dots & c_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n2} & c_{n3} & \dots & c_{nn} - \lambda \end{vmatrix} = (\lambda_1 - \lambda) \mid C - \lambda I \mid$$

The eigenvalues of C are therefore the eigenvalues of B, i.e. the eigenvalues of B are given by

$$(\lambda_1 - \lambda) \mid C - \lambda I \mid = 0.$$

Hence, one eigenvalue is λ_1, the rest being obtained by solving

$$\mid (C - \lambda I) \mid = 0.$$

Further, if $(\beta_1, \beta_2, ..., \beta_n)^T$ is an eigenvector of B with eigenvalue μ, then

$$\begin{pmatrix} \lambda_1 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & c_{22} & c_{23} & \dots & c_{2n} \\ 0 & c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_{n2} & c_{n3} & \dots & c_{nn} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix} = \mu \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix}.$$

These equations can be written as

$$\begin{pmatrix} c_{22} & c_{23} & \dots & c_{2n} \\ c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n2} & c_{n3} & \dots & c_{nn} \end{pmatrix} \begin{pmatrix} \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix} = \mu \begin{pmatrix} \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix}$$

and

$$\lambda_1\beta_1 + a_{12}\beta_2 + a_{13}\beta_3 + ... + a_{1n}\beta_n = \mu\beta_1.$$

From the last equation we have

$$\beta_1 = \frac{\sum_{j=2}^{n} a_{1j}\beta_j}{\mu - \lambda_1}.$$

From Lemma (2) we know that if λ_1 is known (obtained from A via the power method for example) then we can concentrate attention on finding another eigenvalue by extracting the reduced matrix C and applying the power method again. This is the principle of deflation. The remaining question is what transformation matrix P gives $B = P^{-1}AP$ of the appropriate form. The matrix required is given by

$$P = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \xi_2 & 1 & 0 & \dots & 0 \\ \xi_3 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \xi_n & 0 & 0 & \dots & 1 \end{pmatrix}$$

and if $\mathbf{x}_1 = (x_1, x_2, ..., x_n)^T$ is the eigenvector associated with λ_1, then

$$\xi_i = \frac{x_i}{|\ \mathbf{x}_1\ |}; \quad 2 \leq i \leq n.$$

The inverse of P is

$$P^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\xi_2 & 1 & 0 & \dots & 0 \\ -\xi_3 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\xi_n & 0 & 0 & \dots & 1 \end{pmatrix}.$$

10.4.4 The Deflation Method for a Non-symmetric Matrix

Deflation for the case when A is non-symmetric is based on the following important property: If A is any real matrix and P is a real non-singular matrix, then A and $B = P^{-1}AP$ have the same eigenvalues and related eigenvectors given by $\mathbf{v}_i = P^{-1}\mathbf{x}_i$ where $\mathbf{x}_i$ are the eigenvectors of A. The application of a similarity transform of this type is a good idea if the eigenpairs of B are easier to compute than the eigenpairs of A. Note that $\mathbf{x}_i$ can be recovered from $\mathbf{v}_i = P^{-1}\mathbf{x}_i$ since

$$P\mathbf{v}_i = PP^{-1}\mathbf{x}_i = \mathbf{x}_i.$$

Example Consider the matrix

$$A = \begin{pmatrix} 10 & -6 & -4 \\ -6 & 11 & 2 \\ -4 & 2 & 6 \end{pmatrix}$$

where the dominant eigenpair $(\lambda_1, \mathbf{v}_1)$ is given by $\lambda_1 = 18$ and $\mathbf{x}_1 = (1, -1, -1/2)^T$. The transform matrix we require for deflation is given by

$$P = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}, \quad P^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix}$$

so that

$$P^{-1}AP = \begin{pmatrix} 18 & -6 & -4 \\ 0 & 5 & -2 \\ 0 & -1 & 4 \end{pmatrix}.$$

The deflated matrix is

$$C = \begin{pmatrix} 5 & -2 \\ -1 & 4 \end{pmatrix}.$$

The eigenvalues (λ_2, λ_3) of this matrix are $(6, 3)$ and the eigenvectors of this matrix are $(1, -1/2)^T$ and $(1, 1)^T$. Now

$$\begin{aligned} \beta_1 &= \frac{\sum_{j=2}^{3} a_{1j}\beta_j}{\mu - \lambda_i} \\ &= 1/3, \quad i = 2; \\ &= 2/3, \quad i = 3. \end{aligned}$$

Hence,

$$\mathbf{v}_2 = (1/3, 1, -1/2)^T \quad \text{and} \quad \mathbf{v}_3 = (2/3, 1, 1)^T.$$

Finally, $\mathbf{v}_i = P^{-1}\mathbf{x}_i$ giving

$$\mathbf{x}_2 = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/3 \\ 1 \\ -1/2 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 2/3 \\ -2/3 \end{pmatrix}.$$

Similarly

$$\mathbf{x}_3 = \begin{pmatrix} 2/3 \\ 1/3 \\ 2/3 \end{pmatrix}.$$

10.4.5 The Deflation Method for a Symmetric Matrix

If A is symmetric and $(\lambda_1, \mathbf{v}_1)$ is the dominant eigenpair of A, then

$$B = A - \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T \quad \text{where} \quad \mathbf{u}_1 = \mathbf{v}_1 / \mid \mathbf{v}_1 \mid$$

has eigenvalues $0, \lambda_2, \lambda_3, ..., \lambda_n$ and the eigenvectors of B are the eigenvectors of A. To find λ_2 (the second dominant eigenvalue) we can apply the power method to B. To find all eigenpairs of a symmetric matrix A, we can apply the following algorithm:

Step 1: Given that $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$ apply the power method starting with $i = 1$ to compute $(\lambda_i, \mathbf{v}_i)$.

Step 2: Compute $\mathbf{u}_i = \mathbf{v}_i / \mid \mathbf{v}_i \mid$ and then $A_{i+1} = A_i - \lambda_i \mathbf{u}_i \mathbf{u}_i^T$.

Step 3: Use the power method to compute $(\lambda_{i+1}, \mathbf{v}_{i+1})$.

Step 4: Repeat the process until all the eigenpairs are obtained.

Note that errors in λ_1 and $\mathbf{v}_1$ can propagate through the calculation. Therefore, λ_i and $\mathbf{v}_i$ become more inaccurate as i increases.

10.5 Jacobi's Method for Symmetric Matrices

The principal idea behind Jacobi's method is that since $P = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ where $\mathbf{x}_i$ are linearly independent eigenvectors of a matrix A, then $P^{-1}AP = \Lambda$ where Λ is a diagonal matrix and that the elements of the leading diagonal of Λ are the eigenvalues of A. Hence, if we can find some transformation matrices $P_1, P_2, ..., P_n$ such that

$$P_n^{-1} ... P_2^{-1} P_1^{-1} A P_1 P_2 ... P_n = \Lambda$$

then, after n steps, the eigenvalues of A are the elements along the leading diagonal of Λ and the eigenvectors of A are the columns of the matrix $P_1 P_2 ... P_n$.

10.5.1 The Transformation Matrix

The transformation matrix which does the trick is the 'rotation matrix' given by

$$P = \begin{pmatrix} \ddots & & & & & & & & & & \\ & 1 & & & & & & & & & \\ & & \cos\theta & & & & & \sin\theta & & & \\ & & & 1 & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & 1 & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & 1 & & & \\ & & -\sin\theta & & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}$$

where

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{qq} - a_{pp}}\right).$$

P has the following important properties:

1. $P^{-1} = P^T$ therefore P^{-1} is easy to compute.

2. Given A, $P^T AP$ gives a matrix with 0's at positions pq and pp, i.e. 0's occur in $P^T AP$ at the locations of $-\sin\theta$ and $\sin\theta$ in P. By repeating the process, i.e. using

$$A_{m+1} = P_m^T A_m P_m$$

where the positions of $\sin\theta$ and $-\sin\theta$ in P change as m increases, the off-diagonal elements of A can be eliminated.

Example Find the eigenvalues of the matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$$

using Jacobi's method.

$$P = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

where

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\times 1}{3-2}\right) = \frac{1}{2}\tan^{-1}(2) = 31.72.$$

Working to 2 decimal places only,

$$P = \begin{pmatrix} 0.85 & 0.53 \\ -0.53 & 0.83 \end{pmatrix}$$

and

$$P^{-1} = P^T = \begin{pmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{pmatrix}$$

giving

$$A_1 = P^{-1}AP = \begin{pmatrix} 1.39 & -8.9 \times 10^{-3} \\ -8.9 \times 10^{-3} & 3.63 \end{pmatrix}.$$

Thus, the eigenvalues are given approximately by $(1.39, 3.63)$ which should be compared with the formal method of solution which gives

$$\lambda = \frac{5 \pm \sqrt{5}}{2} \simeq 1.38, \quad 3.62.$$

10.5.2 The Serial Method

The serial method is based on eliminating the off-diagonal elements in row order using an appropriate rotation matrix. To illustrate this technique schematically, consider Jacobi's method for the 3×3 system (where $*$ denotes an element of the matrix)

$$A = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}.$$

Then,

$$P_1 = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad P_1^T A P_1 = A_1 = \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix},$$

$$P_2 = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad P_2^T A_1 P_2 = A_2 = \begin{pmatrix} * & 0 & 0 \\ 0 & * & * \\ 0 & * & * \end{pmatrix},$$

$$P_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}, \quad P_3^T A_2 P_3 = A_3 = \begin{pmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix}.$$

Note that in practice, all the 0's above, are relatively small numbers. The procedure can therefore be repeated to improve the result, i.e. to get smaller numbers in the off-diagonal positions. To illustrate the method further, consider the application of Jacobi's method for a 4×4 system using the serial technique when

$$A = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}.$$

In this case,

$$P_1 = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_1^T A P_1 = A_1 = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix},$$

$$P_2 = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_2^T A_1 P_2 = A_2 = \begin{pmatrix} * & 0 & 0 & * \\ 0 & * & * & * \\ 0 & * & * & * \\ * & * & * & * \end{pmatrix},$$

$$P_3 = \begin{pmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta & 0 & 0 & \cos\theta \end{pmatrix}, \quad P_3^T A_2 P_3 = A_3 = \begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}.$$

Similarly, we compute A_4 and A_5 and finally A_6 given by

$$A_6 = \begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{pmatrix}.$$

10.6 Sturm Sequence Iteration

Sturm sequence iteration is used for the evaluation of the eigenvalues of a matrix A when A is symmetric and tridiagonal. Consider the matrix

$$A = \begin{pmatrix} d_1 & c_1 & & & & \\ c_1 & d_1 & c_2 & & & \\ & c_2 & d_3 & & & \\ & & & \ddots & & \\ & & & & c_{n-1} & \\ & & & c_{n-1} & d_n & \end{pmatrix}$$

where d_i and c_i are non-zero $\forall i$. The eigenvalues of A are the roots of the equation

$$f(\lambda) = \mid A - \lambda I \mid = 0.$$

The Sturm Sequence

The principal leading minors of $\mid A - \lambda I \mid$ are as follows:

$$f_1(\lambda) = (d_1 - \lambda),$$

$$f_2(\lambda) = (d_2 - \lambda)(d_1 - \lambda) - c_1^2 = (d_2 - \lambda) f_1(\lambda) - c_1^2 f_0(\lambda),$$

where by definition, $f_0(\lambda) = 1$. Similarly,

$$f_3(\lambda) = (d_3 - \lambda) f_2(\lambda) - c_2^2 f_1(\lambda),$$

$$f_4(\lambda) = (d_4 - \lambda)f_3(\lambda) - c_3^2 f_2(\lambda),$$

$$\vdots$$

$$f_{n+1}(\lambda) = (d_{n+1} - \lambda)f_n(\lambda) - c_n^2 f_{n-1}(\lambda).$$

These polynomials form a 'Sturm sequence'. We can evaluate $\mid A - \lambda I \mid$ using this result if A is tridiagonal. For $\lambda = 0$, we can evaluate $\mid A \mid$ using the Sturm sequence.

Computing the Eigenvalues using a Sturm Sequence

This is based on the fundamental result that for any value of λ, the number of agreements in sign of successive terms of the Sturm sequence is equal to the number of eigenvalues of A which are strictly greater than λ, the sign of a zero being taken to be opposite to that of the previous term. The basic idea is to find the number of eigenvalues in a given range, count the number of agreements in sign $s(\lambda)$ in the sequence $f_0(\lambda), f_1(\lambda), f_2(\lambda), ..., f_n(\lambda)$ and use the result

$$s(\lambda) = \text{number of roots} > \lambda.$$

Example Find the number of eigenvalues of the matrix

$$A = \begin{pmatrix} 2 & 4 & 0 & 0 \\ 4 & 10 & 3 & 0 \\ 0 & 3 & 9 & -1 \\ 0 & 0 & -1 & 5 \end{pmatrix}$$

lying in the interval $[0, 5]$. Take $\lambda = 0$ giving $f_0(0) = 1$ (by definition) then

$$f_1(0) = (d_1 - 0) = 2 - 0 = 2,$$

$$f_2(0) = (d_2 - 0)f_1(0) - c_1^2 f_0(0) = (10 - 0) \times 2 - 4^2 \times 1 = 4,$$

$$f_3(0) = (d_3 - 0)f_2(0) - c_2^2 f_1(0) = (9 - 0) \times 4 - 3^2 \times 2 = 18,$$

$$f_4(0) = (d_4 - 0)f_3(0) - c_3^2 f_2(0) = (5 - 0) \times 18 - (-1)^2 \times 4 = 86.$$

The signs of the terms are + + + + + which gives 4 agreements in sign. Hence, there are 4 eigenvalues of A greater than 0. Now, if $\lambda = 5$, then

$$f_0(5) = 1, \quad f_1(5) = -3, \quad f_2(5) = -31, \quad f_3(5) = -97, \quad f_4(5) = 31.$$

The signs of the terms are + - - - + which gives 2 agreements in sign. Hence, there are 2 eigenvalues of A greater than 5. Now, there are 4 eigenvalues altogether, 4 eigenvalues > 0 and 2 eigenvalues > 5. Hence, there must be 2 eigenvalues in the range $[0, 5]$. Note that $f_4(5) =\mid A - I5 \mid \neq 0$ and therefore 5 is not an eigenvalue.

Bisection

To estimate any individual eigenvalues we can bisect the interval in which the desired eigenvalue is known to lie and repeat the process.

Example Consider

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

If all the eigenvalues lie in the interval $[0, 4]$, $0 \leq \lambda_3 < \lambda_2 < \lambda_1 \leq 4$, find λ_2 with an error at most of 0.25.

First stage Compute the Sturm sequence for $\lambda = 2$ (i.e. the mid point or bisection of interval $[0, 4]$):

$$f_0(2) = 1, \quad f_1(2) = 0, \quad f_2(2) = -1, \quad f_3(2) = 1,$$

giving 1 agreement in sign. Hence, only 1 eigenvalue > 2 (i.e. λ_1) and $\lambda_2 \in [0, 2]$.

Second stage Compute the Sturm sequence for $\lambda = 1$ (bisection of the interval $[0, 2]$):

$$f_0(1) = 1, \quad f_1(1) = 1, \quad f_2(1) = 0, \quad f_3(1) = -1$$

giving 2 agreements in sign. Hence, there are two eigenvalues > 1 (i.e. λ_3 and λ_2) and $\lambda_2 \in [1, 2]$.

Third stage Compute the Sturm sequence for $\lambda = 1.5$:

$$f_0(1.5) = 1, \quad f_1(1.5) = 0.5, \quad f_2(1.5) = -0.75, \quad f_3(1.5) = -0.125$$

giving 2 agreements in sign. Hence $\lambda_2 \in [1.5, 2]$ and

$$\lambda_2 = 1.75 \pm 0.25.$$

10.6.1 Gerschgorin's Theorem

Gerschgorin's theorem is used to find the intervals in which eigenvalues lie. The theorem states that every eigenvalue of a matrix $A = (a_{ij})$ lies in at least one of the circles in the complex plane with centre a_{ii} and radius

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} | \, a_{ij} \, | \, .$$

Example Consider

$$A = \begin{pmatrix} 5 & 1 & 0 \\ -1 & 3 & 1 \\ -2 & 1 & 10 \end{pmatrix}.$$

Then

Centre	Radius
5	$\mid 1 \mid + \mid 0 \mid = 1$
3	$\mid -1 \mid + \mid 1 \mid = 2$
10	$\mid -2 \mid + \mid 1 \mid = 3$

By Gerschgorin's, theorem the eigenvalues lie in the ranges

$$[5-1, 5+1] = [4, 6]; \quad [3-2, 3+2] = [1, 5] \quad \text{and} \quad [10-3, 10+3] = [7, 13].$$

Gerschgorin's Theorem and Sturm Sequence Iteration

Gerschgorin's theorem can be used to find the range in which an eigenvalue lies. Bisection using the Sturm sequence iteration is then used to 'focus down' on an eigenvalue. From Gerschgorin's theorem, the range in which all the eigenvalues lie must be $[-\|A\|_\infty, \|A\|_\infty]$.

10.6.2 Givens' Method

If A is tridiagonal, we can use Sturm sequence iteration to compute the eigenvalues. Givens' method converts A (assumed to be symmetric) to tridiagonal form so that Sturm sequence iteration can be used to solve the eigenvalue problem. The basic idea is to use Jacobi's method with a subtle change in tactics to create zeros in A by application of suitable rotation matrices. In Jacobi's method, zeros are created at (p, q) and (q, p); in Givens' method, the zeros are created at $(p-1, q)$ and $(q, p-1)$. Givens' method works by using the formula

$$\theta = -\tan^{-1}\left(\frac{a_{p-1,q}}{a_{p-1,p}}\right)$$

instead of

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{qq} - a_{pp}}\right)$$

as used for Jacobi's method.

Schematic Example of Givens' Method

Consider the matrix

$$A = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

and the transformation matrix

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \theta = -\tan^{-1}\left(\frac{a_{13}}{a_{12}}\right)$$

where $c \equiv \cos\theta$ and $s \equiv \sin\theta$. Then

$$A_2 = P_1^T A P_1 = \begin{pmatrix} * & * & 0 & * \\ * & * & * & * \\ 0 & * & * & * \\ * & * & * & * \end{pmatrix}.$$

Now consider

$$P_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & s \\ 0 & 0 & 1 & 0 \\ 0 & -s & 0 & c \end{pmatrix}, \quad \theta = -\tan^{-1}\left(\frac{a_{14}^{(2)}}{a_{12}^{(2)}}\right)$$

where $a_{ij}^{(2)}$ are elements of A_2 Then

$$A_3 = P_2^T A_2 P_2 = \begin{pmatrix} * & * & 0 & 0 \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

and with

$$P_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c & s \\ 0 & 0 & -s & c \end{pmatrix}, \quad \theta = -\tan^{-1}\left(\frac{a_{24}^{(3)}}{a_{23}^{(3)}}\right)$$

and

$$A_4 = P_3^T A_3 P_3 = \begin{pmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix}.$$

The matrix A_4 is now tridiagonal. We can therefore apply Sturm sequence iteration together with Gerschgorin's theorem to compute the eigenvalues.

10.6.3 Householder's Method

Householder's method is used to convert a symmetric matrix A to tridiagonal form so that Sturm sequence iteration can be applied. It does the same thing as Givens' method but with increased efficiency using partitioning. The method is based on a similarity transform of the type

$$A_{i+1} = P_i^{-1} A_i P_i$$

with

$$P_i = \begin{pmatrix} I_i & \mathbf{0}^T \\ \mathbf{0} & Q_i \end{pmatrix}.$$

Here, Q_i are Householder matrices given by

$$Q_i = I_i - \frac{2\mathbf{u}_i\mathbf{u}_i^T}{\mathbf{u}_i^T\mathbf{u}}$$

where

$$\mathbf{u}_i = \mathbf{x}_i - k_i \mathbf{e}_i,$$

$\mathbf{e}_i$ is the first column of I_i, i.e. $(1, 0, ..., 0)^T$,

$$k_i = \pm(\mathbf{x}_i^T \mathbf{x}_i)^{\frac{1}{2}},$$

and

$$\mathbf{x}_i = \begin{pmatrix} a_{i+1,i} \\ a_{i+2,i} \\ \vdots \\ a_{ni} \end{pmatrix}.$$

Note that P_i is orthogonal and symmetric, therefore

$$P_i^{-1} = P_i^T = P_i.$$

The sign of k_i is chosen to be opposite to that of the first component of $\mathbf{x}_i$. At each step i, Q_i is a $(n-i) \times (n-i)$ Householder matrix satisfying

$$Q_i \mathbf{x}_i = k_i \mathbf{e}_i$$

Schematic Illustration

Consider the matrix

$$A_1 = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & \mathbf{x}_1^T \\ \mathbf{x}_1 & B_1 \end{pmatrix}.$$

Let

$$P_1 = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & Q_1 \end{pmatrix} \quad \text{where} \quad Q_1 \mathbf{x}_1 = k_1 \mathbf{e}_1.$$

Then,

$$A_2 = P_1^{-1} A_1 P_1 = P_1 A_1 P_1 = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & Q_1 \end{pmatrix} \begin{pmatrix} a_{11} & \mathbf{x}_1^T \\ \mathbf{x}_1 & B_1 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & Q_1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & Q_1 \end{pmatrix} \begin{pmatrix} a_{11} & (Q_1 \mathbf{x}_1)^T \\ \mathbf{x}_1 & B_1 Q_1 \end{pmatrix} = \begin{pmatrix} a_{11} & (Q_1 \mathbf{x}_1)^T \\ Q_1 \mathbf{x}_1 & Q_1 B_1 Q_1 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} & k_1 \mathbf{e}_1^T \\ k_1 \mathbf{e}_1 & Q_1 B_1 Q_1 \end{pmatrix} = \begin{pmatrix} a_{11} & k_1 & 0 & \dots & 0 \\ k_1 & & & & \\ 0 & & & & \\ \vdots & & Q_1 B_1 Q_1 & & \\ 0 & & & & \end{pmatrix}.$$

Hence,

$$A_2 = \begin{pmatrix} a_{11}^{(2)} & a_{12}^{(2)} & 0 & \dots & 0 \\ a_{21}^{(2)} & a_{22}^{(2)} & a_{32}^{(2)} & \dots & a_{n2}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \dots & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \dots & & a_{nn}^{(2)} \end{pmatrix} = \begin{pmatrix} A & X^T \\ X & B_2 \end{pmatrix}.$$

Let

$$P_2 = \begin{pmatrix} I_2 & 0^T \\ 0 & Q_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & Q_2 & \\ 0 & 0 & & & \end{pmatrix}.$$

Then,

$$A_3 = P_2 A_2 P_2 = \begin{pmatrix} A & (Q_2 X)^T \\ Q_2 X & Q_2 B_2 Q_2 \end{pmatrix}.$$

Now,

$$Q_2 X = Q_2 \begin{pmatrix} 0 & a_{32}^{(2)} \\ 0 & a_{42}^{(2)} \\ \vdots & \vdots \\ 0 & a_{n2}^{(2)} \end{pmatrix} = (\mathbf{0}, Q_2 \mathbf{x}_2) = (\mathbf{0}, k_2 \mathbf{e}_2) = \begin{pmatrix} 0 & k_2 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix}$$

and

$$(Q_2 X)^T = \begin{pmatrix} 0 & 0 & \dots & 0 \\ k_2 & 0 & \dots & 0 \end{pmatrix}.$$

Hence,

$$A_3 = \begin{pmatrix} a_{11}^{(2)} & a_{12}^{(2)} & 0 & 0 & \dots & 0 \\ a_{21}^{(2)} & a_{22}^{(2)} & k_2 & 0 & \dots & 0 \\ 0 & k_2 & & & & \\ 0 & 0 & & & & \\ \vdots & \vdots & & Q_2 B_2 Q_2 & & \\ 0 & 0 & & & & \end{pmatrix}.$$

Continued application of this process produces a tridiagonal matrix. At each stage, application of the Householder matrix produces 0's along all appropriate entries. The Householder method is therefore more efficient than Givens' method which produces near zeros (i.e. small numbers) at only two entries each time the rotation matrix is applied.

10.7 LR and QR Methods

These methods are used for finding the eigenvalues of non-symmetric matrices. Both methods are based on similarity transforms. Here LR means Left-Right and QR is taken to denote Orthogonal-Right. The LR method is based on LU decomposition.

The QR method is based on decomposing the matrix into the product of an orthogonal matrix Q and an upper triangular matrix R.

The LR Method

Consider

$$A_1\mathbf{x}_i = \lambda_i\mathbf{x}_i$$

to which we apply Doolittle factorization $A_1 = L_1R_1$. Now compute $A_2 = R_1L_1$, factorize A_2 and repeat the process. The general equations for this process are:

$$A_i = L_iR_i,$$

$$A_{i+1} = R_iL_i,$$

for $i = 1, 2, ...$ As $i \to \infty$ the diagonal elements of the matrix A_i reduce to the eigenvalues of A_1, i.e.

$$\lim_{i\to\infty} A_i = \begin{pmatrix} \lambda_1 & & & * \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix}.$$

Note that $R_i = L_i^{-1}A_i$. Therefore $A_{i+1} = L_i^{-1}A_iL_i$ is a similarity transform.

The QR Method

The QR method is based on the same idea as the LR method, the general equations being:

$$A_i = Q_iR_i,$$

$$A_{i+1} = R_iQ_i,$$

for $i = 1, 2, ...$ and

$$\lim_{i\to\infty} A_i = \begin{pmatrix} \lambda_1 & & & * \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix}.$$

To find Q and R, we pre-multiply A_1 by the transpose of a plane rotation matrix P^T with

$$\theta = -\tan^{-1}\left(\frac{a_{qp}}{a_{pp}}\right).$$

This produces a zero in the qp position of P^TA_1. The process is repeated to produce an upper triangular matrix R, i.e.

$$P_n^T...P_2^TP_1^TA_1 = R.$$

Here, P_i are orthogonal so $P^T = P^{-1}$ and hence

$$A_1 = (P_1P_2...P_n)R \equiv QR.$$

Conversion to Hessenberg Form

This is used to improve the efficiency of the QR algorithm. The basic idea is to apply some similarity transform to A of the form $M^{-1}AM$ to obtain an upper Hessenberg matrix H. We then only need to generate zeros in the lower off-diagonal entries to get A and hence Q in the QR algorithm. This is a particularly good idea because RQ is also of Hessenberg form. Therefore (the crucial point) Hessenberg form is preserved. The basis for the algorithm is as follows: Transform A to H using the similarity transform. Then the eigenvalues of H are the same as A and

$$P_n^T ... P_2^T P_1^T H = R, \quad H = QR, \quad Q = P_1 P_2 ... P_n.$$

The general equations are:

$$H_i = Q_i R_i,$$
$$H_{i+1} = R_i Q_i.$$

Reduction to Upper Hessenberg Form

The reduction of a matrix to upper Hessenberg form is accomplished using transformation matrices resembling those used in the analysis of Gauss elimination. Consider the 3×3 example

$$A_1 = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

To reduce this matrix to upper Hessenberg form, we need to annihilate the 31 entry. This can be done by pre-multiplying A by

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{31} & 1 \end{pmatrix}, \quad \text{where} \quad m_{31} = \frac{a_{31}}{a_{21}}.$$

We require the eigenvalues of our upper Hessenberg matrix to be the same as A. Hence, it is necessary to post-multiply the inverse of M which is given by

$$M_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & m_{31} & 1 \end{pmatrix}.$$

Computing $A_2 = M_1^{-1} A_1 M_1$ gives

$$A_2 = \begin{pmatrix} a_{11} & a_{12}^{(2)} & a_{13} \\ a_{21} & a_{22}^{(2)} & a_{23} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix}$$

where $a_{ij}^{(2)}$ denotes the new element in position ij. As a further illustration of the method, consider the 4×4 case when

$$A_1 = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}.$$

To transform A_1 to upper Hessenberg form, we need to annihilate the entries at 31, 41 and 42. To do this we first form the matrix

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -m_{31} & 1 & 0 \\ 0 & -m_{41} & 0 & 1 \end{pmatrix}, \quad m_{31} = \frac{a_{31}}{a_{21}}, \quad m_{41} = \frac{a_{41}}{a_{21}}$$

with

$$M_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & m_{31} & 1 & 0 \\ 0 & m_{41} & 0 & 1 \end{pmatrix}.$$

Thus, $A_2 = M_1^{-1} A_1 M_1$ is given by

$$A_2 = \begin{pmatrix} a_{11} & a_{12}^{(2)} & a_{13} & a_{14} \\ a_{21} & a_{22}^{(2)} & a_{23} & a_{24} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{pmatrix}.$$

We now construct the matrix

$$M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -m_{42} & 1 \end{pmatrix}, \quad \text{where} \quad m_{42} = \frac{a_{42}^{(2)}}{a_{32}^{(2)}}$$

with

$$M_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & m_{42} & 1 \end{pmatrix}$$

and compute $A_3 = M_2^{-1} A_2 M_2$ which is given by

$$A_3 = \begin{pmatrix} a_{11} & a_{12}^{(2)} & a_{13}^{(3)} & a_{14} \\ a_{21} & a_{22}^{(2)} & a_{23}^{(3)} & a_{24} \\ 0 & a_{32}^{(2)} & a_{33}^{(3)} & a_{34}^{(2)} \\ 0 & 0 & a_{43}^{(3)} & a_{44}^{(3)} \end{pmatrix}$$

and is an upper Hessenberg matrix. A general $n \times n$ matrix A_1 can be reduced to upper Hessenberg form in exactly $n-2$ steps. Step j annihilates elements in positions $(j+2, j), (j+3, j), ..., (n, j)$ which is achieved by performing the similarity transform

$$A_{j+1} = M_j^{-1} A_j M_j.$$

The columns of M_j are those of the $n \times n$ identity matrix with the exception of the $(j+1)^{\text{th}}$ column which is

$$(0, 0, ..., 1, m_{j+2,j}, m_{j+3,j}, ..., m_{nj})^T$$

where

$$m_{ij} = \frac{a_{ij}^{(j)}}{a_{j+1,j}^{(j)}}.$$

This reduction method fails if any $a_{j+1,j}^{(j)} = 0$. As in Gauss elimination, row and column interchanges can be used to avoid this difficulty.

10.8 Inverse Iteration

If the eigenvalues of a matrix are known (computed via some method) then the associated eigenvectors can be computed using a technique called 'inverse iteration'. Inverse iteration can be used for any matrix for which an approximate eigenvalue is known.

Basic algorithm

If λ is an approximate eigenvalue then

(i) Solve $(A - I\lambda)\mathbf{z}_1 = \mathbf{y}_0$ where $\mathbf{y}_0 = (1, 1, ..., 1)^T$ via LU factorization;

(ii) set $\mathbf{y}_1 = \mathbf{z}_1/\|\mathbf{z}_1\|$;

(iii) solve $(A - I\lambda)\mathbf{z}_2 = \mathbf{y}_1$;

(iv) set $\mathbf{y}_2 = \mathbf{z}_2/\|\mathbf{z}_2\|$;

(v) solve $(A - I\lambda)\mathbf{z}_3 = \mathbf{y}_2$;

(vi) continue the process until $\mathbf{y}_n$ is sufficiently small.

Diagram 10.1 provides a schematic guide to the application of the methods discussed in this chapter.

10.9 Special Types of Matrices

We conclude this chapter and Part II in general with a brief review of the matrices and some of their special forms or types.

$$\mathrm{A} = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

1. Row Matrix or Row Vector A matrix of one row of order $1 \times n$.

$$\mathbf{a} = (a_1, a_2, ..., a_n)$$

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

where A is a square matrix

$\downarrow$

Largest eigenvalue	A tridiagonal	A symmetric		A non symmetric	
$\downarrow$	$\downarrow$	$\downarrow$	$\searrow$	$\downarrow$	$\searrow$
Power method	Sturm sequence iteration	Jacobi's method	Givens' method	Householder's method	Conversion to Hessenberg form
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
			Sturm sequence iteration	Sturm sequence iteration	QR algorithm
			$\downarrow$	$\downarrow$	$\downarrow$
$(\lambda_1, \mathbf{x}_1)$	λ_i	λ_i	λ_i	λ_i	λ_i
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
	Inverse iteration	Inverse iteration	Inverse iteration	Inverse iteration	Inverse iteration
	$\mathbf{x}_i$	$\mathbf{x}_i$	$\mathbf{x}_i$	$\mathbf{x}_i$	$\mathbf{x}_i$

Diagram 10.1: Schematic diagram illustrating solutions to the linear eigenvalue problem.

2. Column Matrix or Column Vector A matrix of one column of order $m \times 1$.

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}$$

3. Null or Zero Matrix A matrix where every element is zero - it may be of order $m \times n$.

$$\mathrm{O} = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

4. Transpose of a Matrix Denoted by A^T or $\widetilde{\mathrm{A}}$ and obtained from A by interchanging rows and columns.

$$\mathrm{A}^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}$$

If A is of order $m \times n$ then A^T is of order $n \times m$.

5. Square Matrix A matrix of order $n \times n$.

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

6. Trace of a Matrix The sum of the elements on the leading diagonal $(a_{11}, a_{22}, ..., a_{nn})$.

$$\mathrm{TrA} = \sum_{i=1}^{n} a_{ii}$$

7. Symmetric Matrix A square matrix $\mathrm{A} = (a_{ij})$ such that $a_{ij} = a_{ji}$ for all i, j.

$$\mathrm{A}^T = \mathrm{A}$$

8. Skew-Symmetric Matrix A square matrix $\mathrm{A} = (a_{ij})$ such that $a_{ij} = -a_{ji}$ for all i, j.

$$\mathrm{A}^T = -\mathrm{A}$$

9. Diagonal Matrix A square matrix whose off-diagonal elements are all zero, i.e. $a_{ij} = 0$ whenever $i \neq j$.

10. Unit or Identity Matrix A matrix of order n whose diagonal elements are all equal to one and whose off-diagonal elements are all zero.

$$\mathrm{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

11. Singular Matrix A square matrix A whose determinant is zero.

$$\mathrm{detA} = 0$$

12. Non-singular Matrix A square matrix A whose determinant is non-zero.

$$\mathrm{detA} \neq 0$$

13. Adjoint of a matrix Denoted by adjA, this is defined for square matrices A only. It is the transpose of the matrix whose elements are the cofactors of the elements of A. The cofactors A_{ij} of a_{ij} are given by

$$A_{ij} = (-1)^{i+j} M_{ij}$$

where M_{ij} are the minors of the elements a_{ij}, obtained by deleting the row and column in which a_{ij} occurs and computing the determinant of the elements that remain.

14. Inverse or Reciprocal matrix Denoted by A^{-1}, this type of matrix is uniquely defined for square matrices only and exists if and only if A is non-singular (i.e. if $\mathrm{detA} \neq 0$). A^{-1} is defined by

$$\mathrm{A}^{-1} = \frac{\mathrm{adjA}}{\mathrm{detA}}.$$

15. Orthogonal Matrix A square matrix such that

$$\mathrm{AA}^T = \mathrm{A}^T\mathrm{A} = \mathrm{I}$$

16. Upper Triangular Matrix A square matrix for which all elements below the leading diagonal are zero.

$$\mathrm{U} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

$\mid \mathrm{U} \mid$ is equal to the product of all the elements on the leading diagonal. U^{-1} exist only if no element on the leading diagonal of U is zero. In this case, U^{-1} is another upper triangular matrix.

17. Lower Triangular Matrix A square matrix for which all elements above the leading diagonal are zero.

$$\mathrm{L} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

The determinant of this type of matrix is given by

$$\mathrm{detL} = \prod_{i=1}^{n} a_{ii}$$

and is non-zero only if $a_{ii} \neq 0, \quad \forall i$. Then, L^{-1} exists and is another lower triangular matrix.

18. Idempotent Matrix A square matrix A which satisfies the relation

$$\mathrm{A}^2 = \mathrm{A}.$$

19. Nilpotent Matrix A square matrix which satisfies the relation

$$\mathrm{A}^k = 0$$

where k is any positive integer.

20. Complex Conjugate of a Matrix If A is a matrix of order $m \times n$ with complex elements a_{ij}, then the complex conjugate of A - denoted by A^* - is obtained by taking the complex conjugate of all the elements a_{ij}.

21. Hermitian Conjugate of a Matrix Denoted by $\mathrm{A}^\dagger$ and defined by

$$\mathrm{A}^\dagger = (\mathrm{A}^*)^T = (\mathrm{A}^T)^*.$$

22. Hermitian Matrix A is hermitian if

$$\mathrm{A}^\dagger = \mathrm{A}.$$

23. Skew Hermitian Matrix A is skew hermitian if

$$\mathrm{A}^\dagger = -\mathrm{A}.$$

24. Unitary Matrix A square matrix A such that

$$\mathrm{A}\mathrm{A}^\dagger = \mathrm{A}^\dagger\mathrm{A} = \mathrm{I}.$$

25. Sparse Matrix A matrix which has a large number of elements which are equal to zero.

26. Dense Matrix A matrix which has relatively few elements equal to zero.

27. Tridiagonal Matrix A matrix with elements on the leading diagonal and the diagonals on each side of the leading diagonal (the subdiagonals).

$$\mathrm{A} = \begin{pmatrix} * & * & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ * & * & * & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & * & * & * & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots\ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & * & * & * & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & * & * & * \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & * & * \end{pmatrix}$$

28. Hilbert Matrix A matrix of order $n+1$ given by

$$\mathrm{A} = \begin{pmatrix} 1 & 1/2 & \dots & 1/(n+1) \\ 1/2 & 1/3 & \dots & 1/(n+2) \\ \vdots & \vdots & \ddots & \vdots \\ 1/(n+1) & 1/(n+2) & \dots & 1/(2n+1) \end{pmatrix}$$

29. Hessenberg matrix A matrix A = (a_{ij}) where $a_{ij} = 0 \forall i > j+1$ (Upper Hessenberg) or where $a_{ij} = 0 \forall j > i+1$ (Lower Hessenberg).

Upper Hessenberg (all elements below subdiagonal are zero)

$$\mathrm{A} = \begin{pmatrix} * & * & \dots & * & * \\ * & * & \dots & * & * \\ 0 & * & \dots & * & * \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & * & * \\ 0 & 0 & \dots & * & * \end{pmatrix}$$

Lower Hessenberg (all elements above subdiagonal are zero)

$$\mathrm{A} = \begin{pmatrix} * & * & 0 & \dots & 0 & 0 \\ * & * & * & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ * & * & * & \dots & * & * \\ * & * & * & \dots & * & * \end{pmatrix}$$

30 Toeplitz Matrix A matrix A whose leading diagonal consists of elements of the same value.

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

where

$$a_{11} = a_{22} = ... = a_{nn}.$$

10.10 Summary of Important Results

Eigenvector Equation

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i.$$

$$\mathrm{detA} = \prod_{i=1}^{n} \lambda_i.$$

$$\mathrm{Tr}A \equiv \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i.$$

Orthogonality

If

$$A^T \mathbf{y}_i = \lambda_i \mathbf{y}_i$$

then

$$y_j^T \mathbf{x}_i = 0$$

provided $\lambda_i \neq \lambda_j$.

Linearly Dependent Eigenvectors

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + ... + c_n x_n = 0$$

where $c_i, \quad i = 1, 2, ..., n$ are not all zero.

Linearly Independent Eigenvectors

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + ... + c_n x_n = 0$$

where $c_i, \quad i = 1, 2, ..., n$ are all zero.

Diagonalization

$$X^{-1}AX = \Lambda \equiv \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$$

where

$$X = (\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n).$$

Integral Powers

$$A^n = X\Lambda^n X^{-1}$$

Cayley-Hamilton Theorem

If

$$f(\lambda) = \mid A - I\lambda \mid$$

then

$$f(A) = 0.$$

The Power Method

$$\lambda_1 = \frac{A(A^n\mathbf{x}_0) \cdot A^n\mathbf{x}_0}{A^n\mathbf{x}_0 \cdot A^n\mathbf{x}_0}$$

and

$$\mathbf{v}_1 = A^n\mathbf{x}_0$$

where $\mathbf{x}_0 = (1, 1, ..., 1)^T$.

Similarity Transform

$$AP^{-1}AP$$

where $P^{-1}AP$ has the same eigenvalues as A.

Jacobi Method

Based on the result

$$P_n^{-1}...P_2^{-1}P_1^{-1}AP_1P_2...P_n = \Lambda$$

where

$$P = \begin{pmatrix} \ddots & & & & & & & & & & \\ & 1 & & & & & & & & & \\ & & \cos\theta & & & & & \sin\theta & & & \\ & & & 1 & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & 1 & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & 1 & & & \\ & & -\sin\theta & & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix},$$

$P^{-1} = P^T$ and

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{qq} - a_{pp}}\right)$$

giving zeros at (p, q) and (q, p).

Sturm Sequence Iteration

Given

$$A = \begin{pmatrix} d_1 & c_1 & & & & \\ c_1 & d_1 & c_2 & & & \\ & c_2 & d_3 & & & \\ & & & \ddots & & \\ & & & & & c_{n-1} \\ & & & & c_{n-1} & d_n \end{pmatrix}$$

the Sturm sequence is

$$f_1(\lambda) = (d_1 - \lambda),$$
$$f_2(\lambda) = (d_2 - \lambda)f_1(\lambda) - c_1^2 f_0(\lambda),$$
$$f_3(\lambda) = (d_3 - \lambda)f_2(\lambda) - c_2^2 f_1(\lambda),$$
$$\vdots$$
$$f_{n+1}(\lambda) = (d_{n+1} - \lambda)f_n(\lambda) - c_n^2 f_{n-1}(\lambda).$$

where $f_0(\lambda) = 1$.

The number of agreements in sign of the Sturm sequence gives the number of roots greater than λ.

Givens' Method

Based on the application of

$$P_n^{-1}...P_2^{-1}P_1^{-1}AP_1P_2...P_n = T$$

where T is a tridiagonal matrix,

$$P = \begin{pmatrix} \ddots & & & & & & & & & & \\ & 1 & & & & & & & & & \\ & & \cos\theta & & & & & \sin\theta & & & \\ & & & 1 & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & 1 & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & 1 & & & \\ & & -\sin\theta & & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix},$$

$P^{-1} = P^T$ and

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{a_{p-1,q}}{a_{p-1,q}}\right)$$

giving zeros at $(p-1,q)$ and $(q,p-1)$. Sturm sequence iteration is then used to find the eigenvalues.

Householder Method

Based on the similarity transform

$$A_{i+1} = P_i^{-1}A_iP_i$$

where

$$P_i = \begin{pmatrix} I_i & \mathbf{0}^T \\ \mathbf{0} & Q_i \end{pmatrix},$$

Q_i are Householder matrices given by

$$Q_i = I_i - \frac{2\mathbf{u}_i\mathbf{u}_i^T}{\mathbf{u}_i^T\mathbf{u}}$$

where

$$\mathbf{u}_i = \mathbf{x}_i - k_i\mathbf{e}_i,$$

$\mathbf{e}_i$ is the first column of I_i, i.e. $(1,0,...,0)^T$,

$$k_i = \pm(\mathbf{x}_i^T\mathbf{x}_i)^{\frac{1}{2}},$$

and

$$\mathbf{x}_i = \begin{pmatrix} a_{i+1,i} \\ a_{i+2,i} \\ \vdots \\ a_{ni} \end{pmatrix}.$$

Used to convert the characteristic matrix to tridiagonal form so that Sturm sequence iteration can be used to compute the eigenvalues.

QR Method

$$A_i = Q_i R_i,$$
$$A_{i+1} = R_i Q_i,$$

for $i = 1, 2, ...$ and

$$\lim_{i \to \infty} A_i = \begin{pmatrix} \lambda_1 & & & * \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix}$$

where

$$P_n^T ... P_2^T P_1^T A_1 = R,$$
$$A_1 = (P_1 P_2 ... P_n) R \equiv QR.$$

Q and R are obtained by pre-multiply A_1 by the transpose of a rotation matrix P^T with

$$\theta = -\tan^{-1}\left(\frac{a_{qp}}{a_{pp}}\right)$$

which produces a zero in the qp position of $P^T A_1$.

Inverse Iteration

Iterative method of computing the eigenvectors given the eigenvalues (assumed to have been obtained by an appropriate method). Based on solving

$$(A - \lambda I)\mathbf{z}_{i+1} = \mathbf{y}_i$$

where $\mathbf{y}_0 = (1, 1, ..., 1)^T$ and $\mathbf{y}_i = \mathbf{z}_i / \|z_i\|$.

10.11 Further Reading

- Wilkinson J H, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

- Gill P E, Murray W and Wright M H, *Numerical Linear Algebra and Optimization*, Addison-Wesley, 1991.

- Hageman L A and Young D M, *Applied Iterative Methods*, Academic Press, 1981.

- Cheney W and Kincaid D, *Numerical Mathematics and Computing*, Monterey, 1985.

- Barnett S and Cameron R G, *Introduction to Mathematical Control Theory*, Clarendon Press, 1992.

10.12 Problems

10.1 Starting with initial approximation $\mathbf{x}_0 = (1 \quad 1)^T$, use the power method to find the dominant eigenpair of the matrix

$$\begin{pmatrix} 5 & -2 \\ -2 & 8 \end{pmatrix}$$

correct to 2 decimal places. Normalize each iterate so that the largest element in modulus is 1.

10.2 The matrix

$$A = \begin{pmatrix} 8 & -2 & -3 & 1 \\ 7 & -1 & -3 & 1 \\ 6 & -2 & -1 & 1 \\ 5 & -2 & -3 & 4 \end{pmatrix}$$

has an eigenvector $(1 \quad 1 \quad 1 \quad 1)^T$ with eigenvalue 4. Use the deflation method with exact arithmetic to find the remaining eigenvalues and eigenvectors of A.

10.3

$$P = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad A = \begin{pmatrix} 5 & 0 & 1 \\ 0 & -3 & 0.1 \\ 1 & 0.1 & 2 \end{pmatrix}.$$

Calculate $P^T AP$ and deduce that if

$$\theta = -\frac{1}{2}\tan^{-1}\left(\frac{2}{3}\right)$$

then the $(1 \quad 3)$ and $(3 \quad 1)$ entries of this product are zero. Hence, working to 4 decimal places, write down approximations to the eigenvalues and eigenvectors of A.

10.4 If

$$P = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$A = \begin{pmatrix} 2 & 0.1 & 2 & 0.1 \\ 0.1 & 3 & 0.1 & 0.2 \\ 2 & 0.1 & 6 & 0.05 \\ 0.1 & 0.2 & 0.05 & 1 \end{pmatrix}$$

calculate $P^T AP$ and show that A is nonsingular.

10.5 Prove that every eigenvalue of a matrix $A = (a_{ij})$ lies in at least one of the circles in the complex plane with centre a_{ii} and radius

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \mid a_{ii} \mid .$$

(Gerschgorin's theorem)

10.6 Use Gerschgorin's theorem to find the intervals in which the eigenvalues of

$$\begin{pmatrix} 1 & 2 & 0 & 0 \\ 2 & -1 & 3 & 0 \\ 0 & 3 & 6 & 4 \\ 0 & 0 & 4 & -3 \end{pmatrix}$$

lie. Use the Sturm sequence to find the number of eigenvalues which lie in the interval $[1, 2]$.

10.7 Show that the range in which all the eigenvalues of

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & -1 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

lie is $[-3, 3]$. Hence find, with an error of less than 0.25, all the eigenvalues of this matrix.

10.8 The Householder matrix is given by

$$Q = I - 2\mathbf{w}\mathbf{w}^T$$

where I is the $n \times n$ identity matrix and $\mathbf{w}$ is some $n \times 1$ vector satisfying $\mathbf{w}^T\mathbf{w} = 1$. Verify that Q is both symmetric and orthogonal and show that the form of the Householder matrix required to satisfy the equation

$$Q\mathbf{x} = k\mathbf{e}$$

form some scalar k where $\mathbf{e}$ is the first column of I, is given by

$$Q = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}$$

where

$$\mathbf{u} = \mathbf{x} - k\mathbf{e}.$$

10.9 Construct Householder matrices Q_1 and Q_2 such that

$$Q_1 \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix} = k_1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

and

$$Q_2 \begin{pmatrix} -3 \\ -1 \end{pmatrix} = k_2 \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Hence, reduce the matrix

$$\begin{pmatrix} 3 & 2 & 1 & 2 \\ 2 & -1 & 1 & 2 \\ 1 & 1 & 4 & 3 \\ 2 & 2 & 3 & 1 \end{pmatrix}$$

to tridiagonal form.

10.10 Obtain an orthogonal (QR) decomposition of the matrix

$$\begin{pmatrix} 1 & 4 & 2 \\ -1 & 2 & 0 \\ 1 & 3 & -1 \end{pmatrix}$$

working to 4 decimal places.

Part III

Programming and Software Engineering

Chapter 11

Principles of Software Engineering

11.1 Introduction

Software engineering is a discipline that has and continues to develop rapidly in parallel with advances in hardware specifications and the sophistication of computer applications, especially in information and communications technology. It is arguable that the first example of software engineering (albeit of a primitive form) took place at Bletchley Park (Station X) in Buckinghamshire, England in 1944 when the first programmable computer - the 'Colossus' - was developed as part of the Allied war effort to decrypt the ciphers of the Axis powers. What Winston Churchill called his 'Ultra' secrete remained classified for nearly fifty years after the end of the second world war and it is only relatively recently that information on the technology developed at station X and its impact on the development of computer science has been declassified.

11.1.1 Programming Language Development

Since the development of the first commercial mainframe programmable computers in the 1950s, a range of programming languages have been developed. Those languages that became most popular early on were FORTRAN used by the scientific and engineering communities and COBOL used in business, the banking industry and the financial sector in general. FORTRAN concentrated on the use of computers for the numerical evaluation of formulas, hence the name, FORmula TRANslation. COBOL or the COmmon Business Oriented Language focused on aspects of computing associated with data storage, data access, relational databases and I/O. However, the sophistication of the applications to which these programming languages were applied was relatively low and the idea of developing methods for 'engineering' the software developed was not an issue. What was an over-riding issue in those early days of program development was the importance of designing software that was as efficient as possible in terms of: (i) minimal integer and, in particular, floating point computations; (ii) minimal use of core memory. The reasons for this were the limi-

tations placed on the speed of the central processor and the core memory as well as (to a lesser extent) external memory storage (e.g. magnetic tapes). The 1960s saw a rapid expansion in the development of computing hardware and, in parallel with this, the development of new programming languages and standards. These included languages such as BASIC (Beginner's All-purpose Symbolic Instruction Code) which became central to the development for programming microprocessors and ALGOL (Algorithmic Language) which has been particularly important in the development of programming languages, no other single language having had such a far-reaching influence on the design and definition of languages. Throughout this period, the facilities offered by the hardware and the programming languages available lead to the need to develop methodologies for designing software especially with regard to specifying standards and protocols needed for employing teams of programmers. Even so, many such programmers were from a scientific and/or engineering background with little or no formal training in the engineering of software components or the design of a complete system. In the 1970s, as computing facilities and accessibility to them grew, it started to become clear that new approaches to engineering software were required in order to enhance the performance and delivery of software packages that were being used in an increasing number of applications. Thus, programming techniques were developed that were based on utilising specific approaches and rationale for training the increasing number of programmers and software engineers needed for a rapidly growing market. This included the introduction of Pascal which was developed in 1970 to make available a language suitable to teach programming as a systematic discipline and to develop implementations which were both reliable and efficient. Pascal became one of the major 'teaching languages' in the USA and Europe. The 1970s was a period of rapid expansion in the area of computing and many other new languages were developed during this time. These included PL/1 (Programming Language 1) for example that was designed to be a multi-purpose programming language and whose original goal was to succeed FORTRAN with the inclusion of more extensive data structuring facilities. Many other languages were considered, some of which were adopted by specific industries while others 'fell by the wayside'. Operating systems also developed significantly over this period, one of the more successful results being the development of the VAX/VMS (Virtual Address Extension/Virtual Memory System) by the American based company *Digital* which significantly improved on the memory capacity of a mainframe by utilising external memory as if it were part of the core memory; hence the term 'Virtual Memory System'. However, in hindsight, it is arguable that the most significant development of all, if only in terms of the personal computing we now benefit from, was the development by Microsoft of DOS (Digital Operating System) and independently, the development of the C programming language and the UNIX operating system at the AT & T Bell Laboratories in the early 1970s.

From the foundations laid in the 1970s, the development of microprocessor based computing over the 1980s has forged an explosion in the utilization of computers for a broad spectrum of applications in which the development of Microsoft Windows from DOS was independently accompanied by the development of C++ from C. Although many specialised mainframes (such as super computers) and UNIX based workstations have and continue to be utilized for specialist applications, the use of microprocessors (personal computers) in general has become standard throughout the world. This

has resulted in the development of distributed computing using a local network of microprocesors to enhance the overall performance and computing power and thus, to the concept of grid computing, which is the same basic idea but on a much larger (world wide) scale.

There are two developments that have significantly enhanced computer applications and the utilities provided that were pioneered over the 1980s and the 1990s: (i) Graphical User Interfaces (GUIs); (ii) internet access. Until the concept of a visual or graphical approach to operating a computer was considered at the Xerox Laboratories in the mid-late 1970s, (leading directly to the Apple Macintosh series of PCs - the first commercial microprocessors to utilise GUIs effectively), most operating systems were based on a command line language which required extensive experience to learn properly. The applications of GUIs is now common in all areas of computer science and many aspects of software engineering are concerned, and rightly so, with the development of GUIs for which specific systems have been designed (e.g. the X-windows series of GUI builders developed at MIT in the early 1980s). GUIs are now the principal focus for user operation of a microprocessor and significant advances have been made in the design methodologies associated with making an applications package 'user friendly'. This includes the development of systems that are designed to enhance the task of a programmer and help configure a programming project - Computer Aided Software Engineering.

A major revolution in computer science has occurred since the concept of a distributed information communications network was invented at the CERN particle physics Laboratories, Switzerland, in the 1980s. The Internet and the World Wide Web, together with electronic mailing facilities, information search engines and so on has radically changed the way in which we utilize the computer. Consequently, computer science has changed many of the original themes upon which it was originally based and has now become dominated by the engineering associated with Information and Communications Technology (ICT). Techniques for software engineering have followed this trend and the field is now dominated by issues associated with the exchange and processing of digital information. However, throughout the revolution in ICT, there are a number of issues that have remained constant in terms of the programming language(s) used and the operating systems implemented. The principal programming language is C and its derivatives including C++ (which has become well associated with the principles of object oriented programming), Java (a development of C/C++ with a focus on Internet based applications), J++ (Java with a number of enhanced facilities), C# (Microsoft's version of Java for compatibility with MS-Windows) and so on. The principal operating systems are Microsoft Windows and Linux (originally a UNIX emulator for PCs) both of which have been and continue to be developed using the C/C++ programming language.

Coupled with the use of various CASE tools, the quasi-standardisation of the programming language, operating system(s) and emphasis on design techniques such as object orientation with open source policies or at least public domain object libraries, an individual software engineer can now achieve results that only a few years ago would have required a team of software engineers and associated management infrastructure to bring a complex software engineering project to a successful conclusion. This has been made possible, at least in part, by the development of object oriented programming and component based software engineering which have many

profound advantages over the previous programming styles such as modular and 'procedure oriented' programming once the idea of an object is understood, upon which, the task of a software engineer becomes easier.

Procedure oriented programming is concerned with the following:

Modular programming
Within the context of a given problem, identifying a sequence of specific processes which can be used to form the basis of a set of procedures (C void functions).

Structured programming
Designing functions in such a way that they are based almost exclusively on a set of well defined control statements where the code can be read as a book (i.e. from left to right and from top to bottom - 'line by line' and 'page by page').

Testing procedures
Designing a processor - a test unit - to test the procedure/function/module using data that is consistent with the real thing.

In object oriented programming, an object contains two principal features: attributes and behaviour. Attributes, which can be member variables or data members, describe an object and determine how it is different from other objects. The member variables can be either instance variables of class variables. Behaviour is determined by what an object does, implemented through its member functions. Objects are instances of classes and so objects are instantiated individually. The behaviour of an object is activated by invoking one of its methods such as sending it a message. When an object receives a message, it either performs an action or modifies its state or both. Sending a massage to an object is equivalent to calling a function using a procedure oriented approach. Objects become, in effect, 'black boxes' which send and receive messages. This approach enhances the development time of software engineering projects and, if properly used, improves the maintenance, reusability and modifiability of software. It is arguable that 'reusability' is one of the most important features of object oriented programming leading directly to component based software engineering in which various components or objects can be assembled to form an applications package. Consequently, many object oriented programming languages have been developed since the development of this methodology in the late 1980s although the leading commercial object oriented languages are far fewer in number, e.g. C++, Smalltalk and Java, the latter example being designed principally for Web-enabled object oriented programming. This includes the modification and extension of some of the more conventional languages to provide object oriented facilities, C being one of the best known examples.

11.1.2 What is Software Engineering ?

The origins and meaning of the term 'Software Engineering' are related to the development of the earliest programmable computers. Although the term was used occasionally in the late 1950s and 1960s it was first popularised by the 1968 NATO Software Engineering Conference held in Garmisch, Germany. Software engineering describes the broad range of activities that were formally called programming and

systems analysis and incorporates all aspects of the practice of computer programming as opposed to the theory of computer programming. The IEEE Standard 610.12 states that software engineering is:

'(1) the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software' and '(2) the study of approaches as in (1).'

It is arguable that the term *software development* is a more appropriate term than *software engineering* for the process of creating software. Software engineering can imply levels of rigor and proven processes that are not appropriate for all types of software development. If Bill Gates had adopted for a rigorous formal methods approach to software engineering, he would probably still be working from of his garage rather than running one of the worlds most successful software engineering companies, i.e. Microsoft Corporation. Thus, in practice, 'craftsmanship' is an important and appropriate metaphor because it focuses on the skills of the developer as the key to success of the 'manufacturing' process. It is also arguable that this field is not mature enough to warrant the term *engineering* since, over the last few decades, at least one radical new approach has entered mainstream software development, e.g. structured programming, modular programming, procedure oriented programming, object oriented programming, component based programming and agile development. This implies that the field is changing too rapidly to be considered an engineering discipline. On the other hand, radical new approaches can be viewed as evolutionary rather than revolutionary; they are mere introductions of new tools rather than fundamental changes.

11.1.3 Applications

The applications of software engineering are countless. However, the nature of the applications can be losely categorised into the following:

- System software
  - operating system components;
  - drivers;
  - editors;
  - file management;
  - compilers.
- Real-time software
  - data gathering from external environments;
  - transformation of information, e.g. DSP;
  - systems monitoring.

- Business software
  - management information systems;
  - relational data bases;
  - financial time series analysis.
- Scientific and engineering software
  - simulation ('number crunching' algorithms);
  - CAD/CAM/CAE;
  - CFD.
- Embedded software
  - limited functions usually residing in ROM;
  - digital control functions (A/D convertors);
  - mobile communication systems.
- PC software
  - computer graphics;
  - LAN and WWW;
  - games.
- Artificial intelligence software
  - knowledge based systems;
  - fuzzy systems control;
  - ANN - pattern recognition (image and voice).

11.1.4 About Part III

The primary goal of Part III of this book is to provide readers with an overview a programming language, namely C, and through this language, discuss techniques for structured and modular programming. Part III is divided into two chapters which cover those elements of the following subjects considered to be important for the software development of digital signal processors, namely: an understanding of number systems, numerical error and accuracy; programming and software engineering including a short discussion on operating systems and programming languages; an overview of the C programming language. The material discusses programming and software engineering using C and prepares the reader for the programming problems given at

the end of each chapter. However, the principles presented are independent of the programming language used and in this chapter, a brief overview of the principles of software engineering is provided without specifying any particular programming language or operating system. This material is followed by a chapter which continues with the theme of programming and software engineering using the C programming language and a procedure oriented approach to software engineering which is the principal focus for developing DSP algorithms and systems.

A set of problems are included for the reader to enhance their understanding of the material and to practice programming in C (Chapter 12). These problems are related to some of the numerical methods used to design algorithms for DSP which are discussed in more detail in Part IV. They are also related to some of the computational methods discussed in Part II. The emphasis of the material is on structured programming in modular form.

Although the material given in Chapter 12 discusses issues using C, it should be understood, that it is not a complete survey of the C programming language for which there is a large variety of excellent text books available. Instead, those aspects of programming and software engineering that are required for the design of a DSP system are discussed using C with the aim of addressing those aspects of the language that are pertinent to the material presented in Part IV. The principal aspects of the language presented in Chapter 12 are concerned with array processing and the design of specific functions using defined array inputs and outputs together with associated numerical I/O parameters. This includes the use of dynamic memory management during run time.

11.2 Decimal and Binary Number Systems

Computers store numbers in some approximate form that can be packed into a fixed number of bits. A decimal system counts in 10's using the marks 0-9 to represent values so that for example $32 = 3 \times 10^1 + 2 \times 10^0$ and $4281 = 4 \times 10^3 + 2 \times 10^2 + 8 \times 10^1 + 1 \times 10^0$. Each digit in the number has positional value and represents the number of occurrences of the weight of that position. Non-integral numbers (real numbers) can be represented in a similar way. For example $4281.82 = 4 \times 10^3 + 2 \times 10^2 + 8 \times 10^1 + 1 \times 10^0 + 8 \times 10^{-1} + 2 \times 10^{-2}$. Here, the decimal 'weights' are:

$$10^3, \quad 10^2, \quad 10^1, \quad 10^0, \quad 10^{-1}, \quad 10^{-2}.$$

Fixed Point, Floating Point and Normalized Floating Point Representation

Consider the fixed point numbers 8340000.00, 0.00000834, 22.18 and 3.1415926. Floating point representation of these numbers could include the representation 834.00×10^4, 0.834×10^{-5}, 22.18×10^0 and 3.1415926×10^0. Normalized floating point representation involves the standardization of the floating point and we write 0.834×10^6, 0.834×10^{-5} 0.2218×10^2 and 0.31415926×10^1. In general, a decimal number can be written in the form

decimal number $= \mathrm{M} \times 10^{\mathrm{E}}$

where M is the Mantissa and E is the Exponent.

11.3 Binary Number Systems

Binary number system count in 2's instead of 10's using the marks 0 and 1 to represent a number. The binary weights are:

$$..., \quad 2^5, \quad 2^4, \quad 2^3, \quad 2^2, \quad 2^1, \quad 2^0, \quad 2^{-1}, \quad 2^{-2}, \quad 2^{-3}, \quad ...$$

which are of course equivalent to the decimal numbers

$$..., \quad 32, \quad 16, \quad 8, \quad 4, \quad 2, \quad 1, \quad 0.5, \quad 0.25, \quad 0.125, \quad ...$$

Thus, $32 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$ or has binary representation 100000. Similarly, $41 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ or binary representation 101001. We can binary encode the decimal digits as follows:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Similarly, we can binary encode floating point decimal numbers using a binary point as illustrate by the examples given below

Decimal numbers	Binary numbers
5.0	101.0
5.5	101.1
6.25	110.01
8.75	1000.111

11.3.1 Binary Coded Decimal (BCD)

BCD is used to represent decimal digits directly. For example,

Decimal	19.25
Binary	10011.01
BCD	00011001.00100101

Here, each decimal digit is represented by its binary form directly instead of attempting to represent the decimal number in binary form directly.

11.3.2 Binary Arithmetic

Binary arithmetic is based on the following:

Addition	Multiplication
$0+0=0$	$0\times 0=0$
$0+1=1$	$0\times 1=0$
$1+0=1$	$1\times 0=0$
$1+1=10$	$1\times 1=1$

For example,

Decimal	Binary
12	1100
+5	+101
17	10001

and

Decimal	Binary
11	01011
+6	+00110
17	10001

Binary numbers (and binary arithmetic) are the basis switches in which 0 is represented by a current that is off and 1 is represented by a current that is on.

11.3.3 Decimal to Binary Conversion of Whole Numbers

For whole number conversion, we divide by 2 and write result beneath with a remainder to the side (1 or 0) and repeat process until a fraction is obtained. For example, to covert 41 to binary form,

$$\begin{array}{ll} 41/2 & 1 \\ 20/2 & 0 \\ 10/2 & 0 \\ 5/2 & 1 \\ 2/2 & 0 \\ 1/2 & 1 \end{array} \qquad 41_{10} \rightarrow 101001_2$$

As a further example, consider the conversion of 63 to binary form based on the following:

$$
\begin{array}{ll}
63/2 & 1 \\
31/2 & 1 \\
15/2 & 1 \\
7/2 & 1 \\
3/2 & 1 \\
1/2 & 1
\end{array}
\qquad 63_{10} \rightarrow 111111_2
$$

11.3.4 Decimal to Binary Conversion of Decimal Numbers

For conversion, we multiply by 2, write the result beneath and remove the digit before the point to form the binary fraction. The process is repeated until a zero fraction value occur, or the required accuracy is reached. For example, consider the conversion of 0.625 to binary form based on the following:

$$
\begin{array}{ll}
0.625\times2 & \\
1.25 & 1 \\
0.25\times2 & \\
0.5 & 0 \\
0.5\times2 & \\
1.0 & 1 \\
0.0\times2 &
\end{array}
\qquad 0.625_{10} \rightarrow .101_2
$$

As a further example consider the conversion of 27.375 to binary form. The first step is to convert the whole number part, i.e.

$$
\begin{array}{ll}
27/2 & 1 \\
13/2 & 1 \\
6/2 & 0 \\
3/2 & 1 \\
1/2 & 1
\end{array}
\qquad 27_{10} \rightarrow 11011_2
$$

The second step is to convert the decimal part, i.e.

$$
\begin{array}{ll}
0.375\times2 & \\
0.75 & 0 \\
0.75\times2 & \\
1.5 & 1 \\
0.5\times2 & \\
1.0 & 1 \\
0.0\times2 &
\end{array}
\qquad .375_{10} \rightarrow .011_2
$$

Combining the results: $27.375_{10} \rightarrow 11011.011_2$.

11.3.5 Conversion form Binary to Decimal

This process is straight forward and can be illustrated using the following example in which the binary number 11.101 is converted to a decimal value: Since,

	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
Decimal	2	1	.	0.5	0.25	0.125
Binary	1	1	.	1	0	1

conversion is given by

$$11.101_2 = 1 \times 2 + 1 \times 1 + 1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 = 3.625_{10}.$$

11.4 Fixed Point Storage and Overflow

Consider a 16 bit word (for demonstration purposes only) where the binary point is taken to occur between the 8^{th} and 9^{th} entries. Thus,

Decimal	Binary	Fixed Point Storage
5.0	101.0	0000010100000000
5.5	101.1	0000010110000000
5.75	101.11	0000010111000000
5.875	101.111	0000010111100000

A problem occurs if the number becomes so large that the number of bits required to represent it using fixed point storage exceeds the word length. For example, in the storage of the number 500.875 we would have

Decimal	Binary	Fixed Point Storage
500.875	111110100.111	11111010011100000

In this case, 17 bits are required to store the number. If the number of bits required exceeds the word length, the result is called 'overflow'. The word length sets a limit on the magnitude of the number that can be stored.

11.5 Floating Point Representation of Binary Numbers

In this representation, we use a Mantissa/Exponent to represent the number as illustrated below.

Decimal	Binary	Normalized Floating Point Binary
0.5	0.1	0.1×10^0
1.5	1.1	0.11×10^1
2.5	10.1	0.101×10^{10}
33.25	100001.01	$0.10000101 \times 10^{110}$
500.875	111110100.111	$0.111110100111 \times 10^{1001}$

In a 16 bit word for example, normalized floating point representation of the numbers above is given by

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0
1	1	1	1	1	0	1	0	0	1	1	1	1	0	0	1

Here, the bits from 1 to 12 are taken to represent the mantissa and the last four bits (from 13 to 16) represent the exponent. Observe, that the first column is always 1 and can be ignored providing further storage space. Also, note that in this case, there is no overflow for the decimal number 500.875. This is the principal upon which number storage is based with VAX (Virtual Address Extension) systems for example in which 32, 64 or 128 bit words are used instead of 16 bit. Integer decimal numbers have an exact representation in a binary number system and integer arithmetic is exact. However, most real decimal numbers can not be represented exactly using a binary number system and hence, real arithmetic is not exact which is an important point to bare in mind with numerical computing. Examples of exact binary representation are for example $0.5 = 1/2^1$ so that the binary representation is 0.1. Similarly $0.25 = 1/2^2$ which has an exact binary representation of 0.01. An example of an approximate binary representation includes decimal numbers such as 0.2 with binary representation 0.0010011001... This is illustrated as follows:

Binary Weight	Decimal Value	Appropriate bit
2^{-1}	0.5	0
2^{-2}	0.25	0
2^{-3}	0.175	1
2^{-4}	0.0625	0
2^{-5}	0.03125	0
2^{-6}	0.015625	1
2^{-7}	0.0078125	1
2^{-8}	0.00390625	0
2^{-9}	0.001953125	0
2^{-10}	0.0009765625	1
2^{-11}	0.00048828125	1

Now,

$$\frac{1}{2^3} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^{10}} + \frac{1}{2^{11}} = 0.199902343$$

and the approximate binary representation of 0.2 is 0.00100110011.

11.6 Numerical Error and Accuracy

Computing essentially consists of information input and information output ('Information or Data Processing'). The method of computing is known as the algorithm

and we can consider the essential underlying process in terms of the following:

$$\textbf{Input information} \longrightarrow \textbf{The algorithm} \longrightarrow \textbf{Output information}$$

A programmer is often concerned with the design of the algorithm. This involves a number of factors: (i) principle; (ii) architecture; (iii) effectiveness; (iv) computational efficiency; (v) accuracy (in the presence of error); (vi) stability. The algorithm typically induces further errors, i.e.

$$\textbf{Input errors} \longrightarrow \textbf{Algorithm errors} \longrightarrow \textbf{Output errors}$$

An essential algorithm design feature is to minimize the output errors. Loosely speaking, the clever manipulation of errors (truncation errors) is practically the entire basis of numerical analysis (although don't tell numerical analysts this because in tends to make them insecure and cross, but not necessarily in that order).

11.6.1 Errors

A numerical error occurs when the value use to represent some quantity is not the true value of that quantity.

Absolute error is the difference between the exact value of a number and an approximation to that number, i.e.

$$\epsilon = n - N$$

where ϵ is the error, n is the approximate value and N is the exact value.

Relative Error allows the importance of an error to be better appreciated by comparing it to the quantity being approximated, e.g.

$$\text{Relative error} = \left| \frac{\epsilon}{N} \right|$$

Relative errors are particularly important if the exact value of a number N is very small or very large. For example, $\epsilon = 0.00001$ sounds acceptable. However, if $N = 0.00002$, the approximation of N by n might be only half the size of N. Similarly, $\epsilon = 2340$ sounds very large, but if $N = 4 \times 10^{20}$, ϵ is negligible in comparison.

Error Analysis is an essential element of numerical analysis because of the inherent use of approximations to exact numbers.

11.6.2 Types of Errors

Numerical errors include:

(i) data errors;
(ii) transcription errors;
(iii) conversion errors;
(iv) rounding errors;
(v) computational errors;
(vi) truncation errors;
(vii) algorithmic errors.

Data errors occur when the input to the computer may be subject to error because of limitations on the method of data collection. These limitations may include the accuracy to which it was possible to make measurements, the skill of the observer and the resources available to obtain the data.

Transcription errors are mistakes in copying from one form to another. Examples could include typing 369 for 396 and mixed doubles, i.e. typing 3226 for 3326. Transcription errors may be reduced or avoided by using direct encoding and validation checks.

Conversion errors occur when converting data from its input form to its stored form (e.g. binary coded decimal to pure binary form). Such errors may occur because of the practical limits placed on accuracy. On output similar errors may occur.

Rounding errors occur when not all the significant digits of a decimal number are given. Suppose that for a given real number α, the digits after the decimal point are $\alpha_1, \alpha_2, ...\alpha_n, \alpha_{n+1}....$ To round down or truncate α to n decimal places, all digits after the n^{th} place are removed. To round up α to n decimal places, then if $\alpha_{n+1} \geq 5$, α_n is increased by one and all digits after the n^{th} place are removed. In either case, the magnitude of the rounding error does not exceed $\frac{1}{2} \times 10^{-n}$. Rounding off involves rounding up or down according to which of these processes makes the least difference in the stated value. For example $\pi = 3.14159265...$, which rounded up to 3 decimal places gives $\pi = 3.142$ and rounded down to 3 decimal places gives $\pi = 3.141$. In general, suppose x is a number and let u be an approximation to x obtained by rounding up to n decimal places and d be an approximation to x obtained by rounding down after n decimal places. Then,

rounding up error $\epsilon_{\text{up}} = |u - x| \leq \frac{1}{2} \times 10^{-n}$;

rounding down error $\epsilon_{\text{down}} = |d - x| \leq 10^{-n}$.

Thus, for example, π rounded up to 3 decimal places $= 3.142$ and $\epsilon_{\text{up}} = |3.142 - 3.14159265...| = 0.000407... < \frac{1}{2} \times 10^{-3}$ and π rounded down to 3 decimal places $= 3.141$ and $\epsilon_{\text{down}} = |3.141 - 3.14159265...| = 0.000592... < 10^{-3}$.

Computational errors occur as a result of performing arithmetic operations and are usually caused by overflow or rounding to intermediate results.

Truncation errors occur when a limited number of terms of an infinite series (representing a certain number for example) are taken to approximate that number. For example, since

$$\tan^{-1}x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + ... \quad | \, x \, | \leq 1$$

and

$$\frac{\pi}{4} = \tan^{-1}1,$$

π can be evaluated to any accuracy using the infinite series

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - ...\right).$$

If the series is truncated to

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7},$$

then a truncation error results. Truncation errors are particularly important in numerical procedures that are based on application of arrays of finite length in algorithms that are based on piecewise continuous functions over an infinite domain. For example, in the application of the Discrete Fourier Transform as discussed in Chapters 3 and 4, truncations errors lead to effects such as the Gibbs' phenomenon. In addition, the finite length of the arrays lead to an effect known as spectral leakage which is discussed further in Part IV (Chapter 13).

Algorithmic errors are concerned with an algorithm which is a set of procedure steps used in the solution of a given problem. Errors incurred by the execution of an algorithm are called algorithmic errors.

11.6.3 Accumulation of Errors

If we add n numbers together, each of which has been rounded to K decimal places, there could be a maximum total error of $n \times \frac{1}{2} \times 10^{-K}$. In practice, it is unlikely to be as large as this and it can be shown that

$$\text{probable error} \propto \sqrt{n} \times 10^{-K}.$$

This result needs to be taken into account when a calculation involving rounded numbers is involved and an answer to a certain accuracy is required. Relative error may be affected even more. For example, consider $f(x) = 1 - \cos x$ where $x = 1^o$; then $\cos 1^o = 0.9998476952...$ If the system being used can only cope say with 4 digits, then $\cos 1^o = 0.9998$ and the approximate value of $f(1^o) = 1 - 0.9998 = 0.0002$ whereas the true value of $f(1^o) = 1 - 0.99984769... = 0.0001523...$ In this case, $\epsilon = 0.0002 - 0.0001523... = 0.00004769...$ and the relative error $= 0.00004769.../0.0001523... = 0.31312...$ or 31.316 % .

11.6.4 Types of Error Growth

Suppose, that in a algorithm involving rounded numbers, the error ϵ associated with each calculation is determined by the function $R(\epsilon)$. Also suppose, that there are n operations involved. If $R(\epsilon) \propto n\epsilon$, we say that the growth of the error is linear. This is normal, unavoidable and usually does not matter too much. If $R(\epsilon) \propto K^n(\epsilon)$ where K is some constant > 1, we say that the growth is exponential. This can be disastrous and must be avoided at all costs. A process which exhibits exponential error growth is unstable.

11.6.5 Errors in Computer Arithmetic

Rounding errors in stored data

Since all computers have a finite word length, there is always a limit to the accuracy of the stored data. The following factors are relevant:

(i) For fixed point integer representation, there is good control over accuracy within the allowed range since there is no fractional part to be rounded.

(ii) For other fixed point representations where part of or all of the number is fractional, rounding will occur often. However, the precision provided may still allow reasonable control over accuracy during addition and subtraction.

(iii) In floating point representations, almost all number storage and calculations can lead to rounding errors.

(iv) Rounding should be unbiased if possible, i.e. numbers should be rounded off rather than rounded up or rounded down.

Conversion errors

In converting fractions from decimal to binary for storage, rounding errors are often introduced. For example,

$$0.8_{10} \rightarrow 0.1100110011001100..._2$$

and

$$0.110011_2 \rightarrow 0.796875_{10}.$$

Computational error

In general, every arithmetic operation performed by a computer may produce a rounding error. The cause of this error will be one of:

(i) the limited number of bits available to store the result;

(ii) overflow and underflow;

(iii) rounding in order to normalize a result.

The size of the error will depend on the size of the word length and the method of rounding up, rounding down or rounding off. Another aspect of computational error relates to the use of intrinsic mathematical functions which are invariably computed using their series representation. Since most such series are infinite series, a truncation error is inevitable.

Errors in Stored Results

All results stored in mantissas of the same length will be subject to the same maximum relative error. This will be the case for any floating point arithmetic operation; add, subtract, multiply or divide.

Floating Point Error Formula

This formula is used to find the maximum error in the result of a floating-point arithmetic operation and is given by

$$R(x \ \ o \ \ y) = (x \ \ o \ \ y)(1 \pm \epsilon)$$

where o represents one of the operations $+, -, \times$ or $/$. Here, ϵ is the maximum relative error for the given method of representation and $R(x \ \ o \ \ y)$ represents the worst result of the operation $x \ \ o \ \ y$. For example, assuming 3 digit decimal representations with $\epsilon = 0.005$, let us consider the maximum error associated with the operations $195+23.4 = 218.4$ and $195 \times 23.4 = 4563$. In this case, $R(195+23.4) = (195+23.4)(1 \pm 0.005) = 218(1 \pm 0.005) = 218 \pm 1.09$ and $R(195 \times 23.40) = (195 \times 23.4)(1 \pm 0.005) = 4560(1 \pm 0.005) = 4560 \pm 22.8$ respectively.

Further Errors in Computer Arithmetic

The order of operations can critically effect the output. In general, it is better to add 'floating-point' numbers in order of magnitude if possible. For example, suppose we compute 0.273000+0.001480+0.000862 (=0.275342) by truncating intermediate results stored with mantissas 3 digits long; then,

$0.273000 + 0.001480 = 0.274480 \rightarrow 0.274000$

and

$0.27400 + 0.000862 = 0.274862 \rightarrow 0.274000$

whereas

$0.000862 + 0.001480 = 0.002342 \rightarrow 0.002340$

and

$0.002340 + 0.27300 = 0.275340 \rightarrow 0.275000$

Algorithmic Errors

The errors produced when using an algorithm will frequently depend on:

(i) the order of the operations performed;

(ii) the number of operations performed.

If the errors from one stage of the algorithm are carried over to successive stages, then the size of the error may grow. These accumulated errors ultimately make the results obtained unreliable. For example, suppose we want to compute the values in the sequence

$$\frac{1}{3}, \ \ \frac{2}{3}, \ \ 1, \ \ \frac{4}{3}, \ \ \frac{5}{3}, ..., \ \ 6$$

using 3 digit accuracy at each stage. Using pseudo code, suppose we design a first algorithm as follows:

```
Algorithm_1

    * Define variables *
      float x
```

```
Begin:
    x=0

* Perform calculation *
    repeat
        x=x+0.333
    until x>6
end
```

The result is

0.333 0.666 0.999 1.33 1.66 1.99 2.32 2.65 ...

Now consider a second algorithm:

```
* Define variables *
      integer n
      float x

Begin:
      n=0

* Perform calculation *
      repeat
          n=n+1
          x=n/3
      until n>18
end
```

The result is:

0.333 0.667 1.00 1.33 1.67 2.00 2.33 2.67 ...

11.7 Methods that Reduce Errors and Maintain Accuracy

Nesting

One of the most significant methods of maintaining accuracy that can be performed by the programmer is nesting. Nesting reduces the number of operations, thereby reducing error accumulation. For example, to evaluate

$$3x^3 + 2x^2 + 5x + 1$$

for a given value of x we use

$$((3x + 2)x + 5)x + 1$$

starting with the innermost bracket and working outwards.

Batch Adding

Batch adding is based on grouping numbers of similar magnitude and adding them. The result is then added to the number of a group of numbers with a similar magnitude and so on.

Conditioning

Conditioning is important when solutions are required to systems that are ill-conditioned in which a small change to the data yields large changes in the solution. Methods of conditioning can be used to assess the size of error that is likely to occur with such systems before implementing an algorithm on the computer and techniques such as iterative improvement used to improve the accuracy of the output (see Chapter 8).

11.8 Program Structures

Program structures are the forms in which program components are constructed, organised and interrelated. The structure of a program covers:

(i) Its syntax and semantics.

(ii) Data declarations.

(iii) Basic operations on the data.

(iv) Control structures.

(v) Subprograms.

11.8.1 Syntax and Semantics

The syntax of a language are the grammatical rules that govern the ways in which words, symbols, expressions and statements may be formed and combined. The semantics of a language are the rules that govern its meaning. In the case of a computer language, meaning is defined in terms of what happens when the program is executed.

11.8.2 Data declarations

Data types such as constants and variables must be defined within a program so that the appropriate operations may be performed upon the data values. Such information is typically given at the start of a program. Declarative statements are used to state the properties of the data, i.e. data that may be of type integer, floating point or character.

In the declaration of a real constant, a value is explicitly assigned to the identifier. A variable name does not have an associated value until it has been assigned one. This is done by either using an assignment statement or by using them as arguments in input functions. The processes of assigning values to variables before using them is called initialization and must be done in a systematic way. In some programming languages, certain characters are given default properties. For example, in the older Fortran compilers, the variables

```
i,j,k,l,m,n
```

have traditionally been taken to be integer by default because of the routine use of these variables for indexing the elements of arrays (as used in linear algebra for example). Other languages dictate that all variables types are declared explicitly.

11.8.3 Input and Output

Many programming languages have special functions for dealing with input and output. Common names for these functions are: input, get, accept, output, write, print, put, display and so on. Input and Output or I/O typically concerns: (i) interacting with the user via the Visual Display Unit or VDU and (ii) writing and reading data to and from a file respectively.

The most immediate type of I/O is the type that outputs a statement on the VDU and inputs a string from the keyboard, i.e.

```
output('statement')
input(string)
```

When input, the string will be stored in main storage and referred to by the appropriate identifier. Depending on the original context and applications under which a language was first developed, its sophistication in terms of its I/O facilities and their functionality can change radically. For example, Cobol and its many derivatives were originally designed for business and administration which required first and foremost, methods of storing, transferring, manipulating and processing data compared to a language such as Fortran which was originally designed for scientists and engineers to translate their formulas. Thus, the I/O facilities available in Cobol were, by way of the application, significantly more sophisticated than those of Fortran.

11.8.4 Operations of Data

Operations on data include:

(i) arithmetic operations;

(ii) operations on integers;

(iii) operations on reals;

(iv) operations on characters;

(v) logical relational operations.

11.8.5 Control

The order in which program instructions are performed must be carefully controlled and programming languages contain features that allow the order of instruction execution to be controlled. All programming problems may be reduced to combinations of controlled sequences, selections or repetitions of basic operations on data.

Control Structures

Program control structures are used to regulate the order in which program statements are executed and fall into three categories:

(i) sequences;

(ii) selections;

(iii) repetitions.

Sequences

In the absence of selections or repetitions, program statements are executed in the sequence in which they appear in the program:

```
statement_1
statement_2
statement_3
```

Selections

Selections form part of the decision-making facilities within a programming language. They allow alternative actions to be taken according to the conditions that exist at particular stages in execution. Versions of the if-then-else statement are available in most high-level languages. The form of syntax for this statement is:

```
if
  condition
then
  statement_sequence_1
else
  statement_sequence_2
endif
```

Selections include nested if-then-else and case statements.

Repetitions

There are many programming problems in which the same sequence of statements needs to be performed again and again a definite number of times. The repeated

performance of the same statement is referred to as looping. Loop constructs for repetitions can be generated using entities such as the 'while' loop, a 'repeat' loop, a 'for' loop and a 'do' loop. Efficient looping schemes form an essential aspect of array processing.

11.8.6 Subprograms

The term 'subprogram' may be used to describe a component part of a program. A well constructed subprogram should be:

(i) self-contained;

(ii) perform well-defined operations on well-defined data;

(iii) have an internal structure that is independent of the program in which it is contained.

The two basic types of subprogram are:

(i) functions;

(ii) procedures, subroutines or void functions;

Functions

Many high-level programming languages have in-built functions or provide the facility for the user to define and construct a function. For example, consider the following program to find the square root of a number using an in-built square root function called *sqrt*.

```
      Program Find_root

* This program has the name "Find_root". It inputs a number and
  outputs the value of the square root of the number. *

  Define variables:
             real number
             real root

  Begin:
         output("Input number")
         input(number)

            if
               number<0
            then
                   output "No root."

                   else
                   root=sqrt(number)
```

```
                    output("Square root is", root)

            endif
    end
```

11.9 Procedures

Any defined way of carrying out some actions may be called a 'procedure'. Programming procedures are defined operations on defined data and may be used as program components. These components are referred to as subroutines or void function. Procedure definitions are similar to function definitions, but procedure calls are statements whereas function calls appear in programs as expressions. For example, consider the following procedure for computing the area of a rectangle.

```
    procedure area(IN: real first_side, real second_side; OUT: real area)

  * This procedure takes in the values of first_side and second_side.

    If the value is not positive then the procedure returns
    area with zero value. Otherwise, the procedure multiplies
    the first_side by the second_side and returns area with the
    value of the result *

    Begin:

         if
         (first_side<=0) OR (second_side<=0)
         then
           area=0
         else
           area=first_side*second_side
         endif
    end
```

The input and output of this procedure are called parameters. The input parameters of a procedure correspond to the arguments of a function. The output parameters of a procedure serve the same purpose as the value returned by a function. Procedures may have I/O parameters which control the way in which data is passed into a procedure, manipulated by it in some defined way and then passed out again. The input to a procedure may or may not be overwritten. As a general rule of thumb, it is better not to overwrite the input so that after the procedure has been used the input parameters are unaltered. Parameters which are internal to the procedure should always but always be initialized. Non-initialization of internal parameters is a common way of generating algorithmic errors.

11.10 Processes Specification

The specification of how the processing is to take place needs to give a precise definition of what processing is needed by giving the relationship between the input data and the output data. Some standard ways of specifying processes are:

(i) pseudocodes;

(ii) flowcharts;

(iii) program structure block diagrams;

(iv) Warnier diagrams;

(v) decision tables and trees.

11.10.1 Pseudocode

Pseudocode is a set of statements whose aim is to quantify the process without obscuring its function with the syntax and semantics of a particular programming language. We have already seen some examples of pseudo code in the previous section which was introduced to present the principle of procedures. In general, the syntax used for pseudo code is arbitrary and user dependent and typically reflects the programming language the user is most familiar with. The key to using pseudo code is to convey the process clearly and accurately in a way that real code using some programming language can not necessarily do as well, otherwise, one might as well write out the code directly - many programmers do! The following examples illustrate the use of pseudo-code.

Example 1 Pseudo-code to read in a number from the keyboard, square it and write out the result to the VDU.

```
output("Input number")
   input(number)
number=number*number
   output("Number squared is", number)
```

Here, the data I/O is assumed to be controlled by the functions output and input.

Example 2 Pseudo-code to compute the square-root of an array containing 10 elements.

```
for i=1,2,...,10; do:
   array(i)=sqrt(array(i))
enddo
```

Example 3 Pseudo-code to read a number (assumed to be non-zero), check whether it is positive or negative and output the result.

```
output("Input positive or negative numbers")
      input(number)

  Begin:
      if number > 0
  then
      output("Number is positive")
  endif

      if number < 0
  then
      output("Number is negative")
      endif
  end
```

11.10.2 Program Flowcharts

Flowcharts are a traditional means of showing, in diagrammatic form, the sequence of steps in performing a programming task. There are two levels of flowchart:

(i) outline program flowcharts;

(ii) detailed program flowcharts.

Outline program flowcharts represent the first stage of turning a systems flowchart into the necessary detail to enable a programmer to write the program. They present the actual computer operations in outline only. Detailed program flowcharts are charts that contain the detailed computer steps necessary to perform the particular task. It is from these charts that the programmer may generate code.

11.10.3 Program Structure Block Diagrams

Structure block diagrams are intended to replace the traditional flowcharts to aid the writing of structured code. There are many varieties of these diagrams but they are all only suitable for low-level specifications.

11.10.4 Warnier Diagrams

Warnier diagrams are a simple way of writing out processing requirements in a structured form. They are mostly used for drafting out simple procedures before attempting to express the procedures in more precise forms such as pseudocode. Warnier diagram operate from left to right and typically use a left brace { to segment a process in to its component parts.

11.10.5 Decision Tables and Trees

Decision tables and trees are a means of expressing process logic. Decision tables are used to analyse a problem. The conditions applying in the particular problem are set

out, and the actions taken (as a result of any combination of the conditions arising) are shown. Decision trees are a graphical representation of decision tables. Their purpose is to aid the construction of decision tables.

11.11 Program Specification

Program specification usually forms part of system specification which defines the whole system. Much of the detail within a program specification is specific to the particular problem to be solved but some general design aims may be expressed explicitly as part of the program specification irrespective of the particular problem. They are:

(i) program design, style and presentation;

(ii) program readability;

(iii) program efficiency;

(iv) program development time;

(v) program development costs;

(vi) program documentation.

11.11.1 Program Design, Style and Presentation

Structured programming can directly contribute to the overall quality of programs and the achievement of many design aims. Structured programs are not only more comprehensible, they are also much easier to test. In general, one should think of reading a program in a similar way to reading a paper or book; although references and flags to other parts of the text will occur from time to time, the paper should read naturally from top to bottom and from left to right. Good programming provides a natural and logical flow to the code in the same way as good writing does. This includes the overall presentation of the text on the page. Aids to program presentation include:

(i) The use of meaningful identifiers for programs, subprograms, variables and constants.

(ii) The indenting of code to highlight its structure (as used in some of the previous pseudo code examples).

(iii) The use of appropriate program structures.

(iv) Restricting the size of subprograms to manageable lengths, e.g. insisting that no subprogram occupies more than one page say of A4, especially when hard copies are required.

(v) Incorporating comments within the program that explain what is being done and how it is done, a particularly important aspect of coding to which many programmers would argue that for every one line of code there should at least two lines of comment.

Point (v) cannot be stressed enough. For all the flow charts, data flow diagrams etc. that may be used to establish a process, the potential reader and modifier of a program will typically only have access to the source code. It is therefore very important for the code to be well commented in order to make a future programmer comprehend the code. In some respects, it is arguable that clarity and comprehension of code is more important than efficiency especially when software engineering projects involve a high turn-over of programming staff.

11.11.2 Program Reliability

Program specification must always be accompanied by carefully considered methods for testing the program. Even if the principles of the program and its coding (automatic or otherwise) are proved, in practice, testing procedures are required, i.e. until such a time as formal methods in computer science become compatible with practical software engineering projects. A program that passes its test may be certified and put to use and if the program continues to work for some appreciable time without failure, it may be regarded as reasonably reliable. The reliability of a system which is composed of many functions is often related to the cohesiveness of the functions which in all cases should be maximized with little or no coupling and interdependence from one function to another.

One useful measure of reliability is the mean time between failures. This is the average time that elapses between program failures being detected - a failure being anything from the output of a wrong result to a non-recoverable and unexpected termination of execution (i.e. a crash).

Formal methods provide an approach that ultimately may allow the programmer to do away with test procedures as the output from some automatic code generator would be based on a process specification that is rigorously proved correct in all cases. However, such methods are only applicable (at present) to problems that are relatively simple. They are typically applied to systems that are not necessarily complex but safety critical. Many such systems are related to digital signal process in control engineering including areas such as avionics, weapons control systems and secure communications systems.

11.11.3 Program Efficiency

The efficiency of a program may be expressed in terms of its use of recourses such as time and storage space. Efficiency is achieved by adopting a sound design method rather than by using programming 'tricks' that result in compact but unreadable programs. A program should be designed to work first and then, if possible, made to work more efficiently. A good design may turn out to be efficient enough by itself. If it is not, it will be much easier to make efficient than a poorly designed program. The use of subprograms - which is the key to structured programming - can slow down execution slightly because of the time taken for parameter passing, but this can be turned to advantage in dealing with the efficient use of restricted storage space. This happens when segmentation is possible.

A segmented program is one that, when executed, allows some parts of the program (segments) to take turns in occupying the same area of main storage. Segments will

occupy main storage when they are executed and held on disk for example the rest of the time. Copying programs or subprograms into the same area of main storage during execution is known as performing an overlay.

11.11.4 Program Development Time

There is usually some limit set on the time allocated to designing and implementing a program. Time limits can cause the quality of a program to suffer but the quality should not be allowed to suffer unnecessarily. Time limitations typically cause a programmer to economize on the design and rush on to the implementation stage without implementing appropriate test procedures. Rushing the coding stage of a program makes matters worse. The best solution is to put most of the time allocated to program development into producing a good design from which a quick and simplified implementation can be produced as necessary. Working toward specific dates by which certain parts of the programming task must be completed is also of value. There is an obvious close relationship between development times and budgets. Significant improvements have been made through the introduction of Computer Aided Software Engineering or CASE tools. One of the most significant results has been the ability for a complex system to be developed with a software engineering team that is composed of fewer personnel, allowing the system to be developed with significantly less overheads and time management.

11.11.5 Program Documentation

Program documentation is very important and varies according to its intended use. Three main areas of use are:

- For the programmer's own present or future use and as an aid to all stages in programming.
- For the present or future use of other programmer, including the programmer's supervisor, e.g. for maintenance, modification, debugging, etc.
- For the users of the program, who may themselves vary in expertise.

The following items may be expected as part of the complete process of documentation:

(i) A statement of the problem (system and program specification).

(ii) Documents specifying the format of inputs and outputs, including checks on the validity of data.

(iii) Details of the data structures used, plus details of how data in files is organised, accessed and kept secure.

(iv) Details of the algorithms and procedures used to solve the problem presented in a suitable form such as pseudocode.

(v) A carefully devised set of test data with evidence to show that the test data has been used to good effect.

(vi) Evidence to show that the program not only works but also that it has simple, effective, unambiguous and error-free methods of input and output for the programmer's use (i.e. a good 'user interface').

(vii) Detailed instructions to the installer/user such as:

(a) limitations of the program;
(b) requirements in order to run the program (e.g. hardware);
(c) details of how to run the program;
(d) instructions (with examples) on how to use the program.

Instructions on how to use the program may take the form of a user manual and on-line help with additional instructions output by the program when it runs. Online documentation and a run-time help facility are valuable aids and becoming more and more common.

11.12 System Design

System design is crucially dependent on the so called software life cycle and can be described schematically as follows:

Specification

⇓

Design

⇓

Testing

⇓

Software Maintenance

Within the context of the above, we now discuss the stages of this life-cycle.

11.12.1 Specification

Specification is concerned with the formulation of software requirements in terms of:

- functions;
- operational constraints;
- external system behaviour;

- support environment;
- hardware on which software is to perform.

11.12.2 Design

Design deals with the realisation of code on the target system. This activity is dependent upon individual skill, attention to detail, knowledge of how best to use available tools and management organisation.

11.12.3 Testing

Testing involves:

- exercising the program using data which is similar to the real thing;
- observing the outputs;
- inferring program errors or inadequacies from anomalies in the output.

This can only be achieved through the establishment of a suitable design strategy which:

(i) tests to see if the individual components meet their requirements;

(ii) ensures that the integrated system functions perform correctly.

In practice, testing a module is done using a set of carefully selected 'test data'. Testing may be conducted by executing the program on the computer or by simulating its execution by a manual paper exercise called a 'dry run'. There are two basic types of testing:

1. Functional testing or black box testing which is based upon typical, extreme and invalid data values that are representative of those covered by the specification.

2. Logical testing or white box testing which is based upon examining the internal structure of the program and selecting data which gives rise to the alternative cases of control flow, e.g. both paths through an if..then..else.

Functional testing is used at the final stage of programming as a basis for accepting or rejecting the system.

11.12.4 Software Maintenance

Upon completion of the implementation stage, the software is typically transferred to operations staff where it must be maintained. Problems associated with software maintenance can invariably be traced to deficiencies in the way the software was designed and developed. A lack of control and discipline in the early stages of the

software life-cycle nearly always translates into problems in the last stage. This leads to 'defect amplification' which refers to the following phenomenon: *During any phase of software development, errors may be generated. Errors that are not removed will be passed through to the next phase. Some of the errors that are passed through will have more significant ramifications on the next and/or subsequent phases.*

An important aspect of the software life-cycle is that it is dynamic. In other words, as the software is designed, coded, tested etc., any or all of the requirements (from specification to software maintenance) will invariably change and cycle back to one or all of the previous stages. This feature of software engineering is known as 'Bersoff's law of system engineering' which states:

'No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle'.

11.13 Requirement Specifications

In general terms, a requirement specification will comprise of six sections as follows:

Function
A 'user-eye' view of the system, dealing with all aspects of the user interface.

Allocation
Delineation of responsibility, for each of the functions that have been defined, to hardware/software/operator.

Constraints
Any particular limitations on the way the system is to do its work, together with their level of importance.

Standards
A definition of existing (i.e. known) and documented techniques which have to be applied to the development of the system.

Quality
Details of the quality control procedures under which the projects progress will be monitored.

Schedule
Targets, i.e. what is expected to be ready and by when.

11.14 Modularity

High modularity of software in which each module is well structured is the key to all good programming design. Moreover, if these modules can be designed in such a way that there implementation can be orientated toward numerous applications, then their existence saves on the duplication of tasks and hence the duplications of software which leads naturally to the principles of objected oriented programming and systems design. The following guide-lines are of value in deciding what to do during the act of

design (leading to programs that are clear, simple and flexible) and provide a criteria for assessing the structure of an existing piece of software.

Restricting module size is one crude way to reduce complexity. An extreme view is to restrict all modules to no more than seven statements. This argument is based on experimental evidence of the human mind being able to comprehend the interrelationship or otherwise of only seven or less statements. Various quantitative measures of complexity are available. A useful measure of a modules complexity is one plus the number of decisions within a module (usually as a result of an if-type statement) - the McCabe cyclometric complexity measure. The principle of information hiding holds that data should be inaccessible other than by means of the procedures that are specially provided for accessing data.

Coupling and cohesion are terms that describe the character of the interaction between modules and within modules, respectively. Coupling and cohesion are complementary. Strong coupling and weak cohesion are bad; weak coupling and strong cohesion are good. Coupling can be characterized into a number of types, ranging from one module altering another's code to modules communicating by exchanging serial data streams only. Coupling and cohesion provide a terminology and a qualitative analysis of modularity. Modules that are used in more than one place should not arise from a top-down design process, but from a bottom-up approach.

11.15 The Jackson Method

This method (first published by Michael Jackson, *Principles of Program Design*, Academic Press, 1975) is arguably the most systematic method in existence for structured programming. The basic thesis is that the structure of a program should match the structure of the file or files that the program is going to act on. The Jackson method is only useful for designing the detail of a software function - the structure of individual programs and modules. It gives little help in designing the overall structure of a large piece of software, which needs to be broken down into programs and modules. The steps in the method are as follows:

1. Draw a diagram (a data structure diagram) describing the structure of each file that the program uses.

2. Derive a single program structure diagram from the set of data structure diagrams.

3. Associate the elementary operations with their appropriate positions in the program structure diagram.

5. Transform the program structure diagram to schematic logic.

The main advantages of this method are:

- There is high 'proximity' between the structure of the program and the structure of the files. Hence a minor change to a file structure will lead only to a minor change in the program.

- There is a series of well-defined steps leading from the specification to the design. Each stage creates a well defined product.

From a programmer's point of view, the method can be seen as an increased regimentation of programming and a means of subjecting work to closer control.

11.16 Data Analysis

Data analysis is concerned with identifying the requirements for data to be input to a system, and output from it. It may concentrate solely on I/O requirements - the interface with the environment - for the system as a whole. Since every system should be designed in terms of a sequence of sub-systems each of which have their own I/O, data analysis can equally well be used to cover sub-systems as well. In practice, the activities of the processing sub-system may well be functionally decomposed into a number of separate modules. Each module of the processing sub-system operates in series with another. The input received by one module is the output generated by a previous module operating at a sub-functional level.

11.17 Data Flow Design

Data flow design examines the flow of data within a prospective piece of software and the transformations that will act upon these flows. The end product is a structure chart for the software that shows:

- the modules of which the software is composed;
- the interaction between the modules.

Data flow diagrams are used in the functional description of systems and/or sub-systems by showing how the input data of a processing component is transformed to the output data. This typically consist of two components:

- circles or 'bubbles' representing the functions performed at the intermediate stages;
- arrowheaded lines with suitable annotations alongside highlighting what the data object is.

The bubbles represent the transformations performed on the data objects identified by the arrowheaded lines. By convention, inputs to a process should arrive from the left; all outputs from an intermediate process are generated on the right. The information flow through the entire group of functions should be from left to right. Drawing a data flow diagram for a proposed piece of software is a vital step in the method. There are three alternative approaches:

1. Start with a single bubble identifying the overall function of the software and its I/O data flows. Break this function down into a set of smaller functions. Continue

this process until it is impossible to continue any further. This is typical of a 'top down' approach which introduces a number of levels - level 01, level 02, ... etc.

2. Start with the output data flow from the system and identify the final transformation that has produced this data. Then identify the previous transformation and so on until the diagram is complete.

3. Start from the input flow to the system and construct the sequence of transformations that should be applied.

Once the data flow diagram is completed, one can regard each function described by a given bubble as a module (an individual function) that inputs and outputs a serial stream of data from one function to another.

11.18 Testing and Implementation

Testing can account for a significant amount of effort required to develop software. In the bottom-up approach to modular programming, each module is tested by inputting appropriate data and analysing the output against a suitable 'bench mark' - a known analytical result for example. Bottom-up testing requires the time consuming construction of 'test beds', and it often tests new modules in groups rather than individually. In the bottom-up approach, the lowest level modules are designed first and tested independently of each other. The next level of subprograms is tested in the same way until the top level program (an interface for example) is reached. In principle, top-level or top-down testing can lead to less programming time, less computer time and above all else, gives better awareness of project progress leading to more reliable software. Here, the highest level of the software is coded first. Program 'stubs' are used to stand in for invoked but as yet unwritten lower level components. These stubs are rudimentary replacements for missing subprograms. Implementation proceeds by selecting lower-level components (formerly stubs) for design and coding and incorporation into the system. At any stage in the software development there are:

(i) higher level components which have already been tested;

(ii) a single component under test;

(iii) stubs.

In practice, it is rare that the development of a software system is based solely on a top-down or top-up scenario but more on a hybrid approach in which all levels are interlinked in terms of the software cycle and the evolution of the design. The elements of software engineering as discussed here, should be seen in terms of a lose guide as to the way in which software evolves. Many of the ideas that are developed under the heading of 'software engineering' have significant credit in terms of their rational and pragmatic approach. However, as any programmer knows, it does not always work out that way and there is nothing more illuminating than getting 'stuck-in' to having to program a system from scratch. It is arguable that many of the principles of software engineering are based on common sense, but this can only be

fully appreciated by those who have undertaken a significant amount of programming and systems development. It is all to easy for over-paid and over-fed project managers who have never written a program in their lives, to come up with fatuous phases such as '...its all just a matter of programming'.

11.19 Stages of System Design

The stages of system design are as follows:

1. Statement of User Requirements.
2. Functional Specification.
3. Technical Specification.
4. Detailed Design.
5. Programming and Unit Testing.
6. System Testing.
7. Conversion and 'Going-Live'

11.19.1 Statement of User Requirements

The statement of user requirements describes what the user expects from the system. This may form part of a proposal for the work or it may be part of an 'invitation to tender' from a client or it may be put together by talking to the client at a high level. Typically 'high level' discussions relates to 'functional' issues. It is particularly important to get the user requirement correct because:

- it gives the user(s) confidence in what you can do for them;
- it enables you to fix the scope of the project early on;
- it ensures that you are able to proceed to the next phase.

11.19.2 Functional Specification

This document expands on the functional description of the system. It specifies all inputs, outputs and processing requirements at a functional level. It generally contains sample input/inquiry screen and sample report layouts together with conversation flows for the system. It is important to get client input on the functional specification and to get them involved in the system design process.

11.19.3 Technical Specification

This document describes how the functionality of the system will be achieved in practical terms. It typically contains:

- database design;
- technical/performance figures;
- recommendations for platform/architecture and language/operating system;
- a breakdown of system functions into programmable modules.

In some cases, smaller projects may combine the functional and technical specifications to avoid unnecessary administrative overhead.

11.19.4 Detailed Design

A detailed design considers the process by which module descriptions in the technical specifications are turned into program designs. Here, it is important to adopt a consistent approach to variable/file names. Two of the most common design approaches are:

- flow diagrams;
- structure charts.

In both cases, it is essential to adopt a modular approach to program design, because:

(i) it avoids code redundancy;

(ii) it increases code efficiency;

(iii) it facilitates code portability.

11.19.5 Programming

With regard to turning the designs into code, it is useful to consider the following points:

- Stay with to the design specification; if you change it make sure you change the document.
- Develop standards for naming program variables.
- Introduce comments which are updated for the purpose code maintainability.
- Avoid the temptation to design whilst coding; it can lead to some seriously doggy code.

- Avoid code redundancy by putting common logic into subprograms.

- Introduce machine specific code such as graphics and file handling into separate modules.

11.19.6 Unit Testing

Unit testing is concerned with introducing a test or test set that verifies that the code does what the design says it should do. The following points are valid:

- Obtain unit test conditions from the detailed design; do not use the code itself.

- Use a structured approach to listing test conditions, so you know which ones have been completed.

- Avoid the temptation to skip the conditions that you are 'sure' will work, they tend to be the ones that don't !

11.19.7 String and System Testing

String and system testing involves tests that ensure that all the unit tested modules work properly together to perform the functions specified in the functional specification. This testing is done in close collaboration with the users of the system and should be based on 'realistic' test data. Particular importance should be attached to testing parts of the system where data is passed between modules. Security/performance and external interfaces may also be tested as part of the system test.

11.19.8 Summary on System Design

The above stages are not rigid, and on smaller projects they tend to be relaxed. However, documentation, although sometimes a pain, is very important. Setting out standards for variable file names etc. at the beginning of a project is also important. It means that on larger projects, analysts and programmers can work as a team. Not adhering to standards can mean a lot of work.

11.20 Computer Aided Software Engineering Tools

There are a wide range of Computer Aided Software Engineering or CASE tools currently available. For the most part, these tools help to make designing and installing systems easier. They are not however a 'cure-all', but they can help system designers concentrate on what the system does, rather than how to achieve it. CASE tools are typically used for

- prototyping;

- screen and report layouts;
- design of the database;
- detailed design;
- code generation;
- generation of test data.

Case tools have their place, but they should not be seen as a natural solution for making life easy. It may be of interest to reader for the author to recollect on an interesting scenario concerning CASE tools. Some time ago, I was presenting a short course on 'Computational Methods for Engineering Design' which was a hands-on approach to designing software that implements some of the more important numerical procedures for computational geometry and finite element analysis. The delegates were composed of six graduates of computer science and IT degrees which I shall call Group I and six graduates of electrical, mechanical engineering and physics which I shall call Group II. When it came to the practical sessions, in which delegates had to design, code and test various modules, inevitably hands went up for assistance. It became immediately clear that Group I were using an inordinate number of windows for editing, compiling, visualising etc. and every time I asked them to show me the problem there appeared to be a large amount of 'pointing and clicking' required in order to illustrate the problem. Not only was the problem difficult to find but the 'mess' on the VDU was enough to give the computer a head-ache. Group II on the other hand, all opted for a simple editor and a command line approach based on a systematic rather old-fashioned approach, in which the material displayed on the VDU was reminiscent of the days before windows was introduced. The problem was easy to evaluate and solve and clarity rained supreme (most of the time). Moreover, Group II managed to accomplished the programming tasks set and developed a working, if primitive object library, whereas Group I appeared to have accomplished very little. I can't help thinking that Group I achieved less because of the way in which they had been trained to use current 'bloatware' which led to significant confusion and diffusion (of the process a module was supposed to perform into the operating environment of the machine). However, it all looked very impressive on the screen and sounded good as well!

11.21 Operating Systems and Languages

An operating system is a suite of programs that operates and controls the various units of a computer. It controls the way software uses hardware. This control ensures that the computer not only operates in the way intended by the user but does it in a systematic, reliable and efficient manner, i.e. schematically, we can think of an operating system in terms of the following:

Users

⇓

Other software

⇓

Operating system

⇓

Hardware

Part of the operating system remains in main storage permanently during the running of the computer. This part is called the kernel and is the controlling part of the operating system. The rest of the operating system is usually stored on a direct-access storage device from which any particular program can be called into main memory by the kernel when required.

11.21.1 Functions of an Operating System

The principal functions of an operating system can be summarized as follows:

- The scheduling and loading of programs, or subprograms, in order to provide a continuous sequence of processing or to provide appropriate responses to events.
- Control over hardware resources, e.g. control over the selection and operation of devices used for input, output or storage.
- Protecting hardware, software and data from improper use.
- Calling into main storage programs and functions as and when required.
- Passing of control from one job (program) to another under a system of priority when more than one application program occupies main storage.
- Provision of error correction routines.
- Furnishing a complete record of all that happens during processes.
- Communication with the computer operator usually by means of the keyboard/mouse and visual display unit.

The operator typically communicates with the operating system by means of a Job Control Language (JCL) which is translated and executed by an interactive command interpreter.

11.21.2 Types of Operating System

Single program systems are the basis for the majority of small microcomputer-based systems. They allow a single user to operate the machine in an interactive mode but normally only allow one user program to be in main storage and processed at a time. (e.g. MS-DOS - MicroSoft Disc Operating System, or DR-DOS - Digital Research Disc Operating System).

Simple batch systems provide multi-programming of batch programs but have few facilities for interaction or multi-access.

Multi-access and time-sharing systems represent the majority of systems but there is a wide range of complexity. On large microcomputers, there are a number of operating systems available for use on different machines produced by many different manufacturers (e.g. UNIX). On large minicomputers and mainframes, operating systems are normally specific to a particular machine and manufacturer.

Real time systems cater for the type of real-time system being used and fall in to three basic categories:

(i) complex multi-access time-sharing;

(ii) commercial real-time systems in which there is an essential job usually making extensive use of data bases;

(iii) process control systems to control and operate processes in which response to change must be as fast and reliable as possible.

11.21.3 DOS

DOS (Disk Operating System) has been the basis for most single-user operating systems of IBM and IBM compatible personal computers. It is one of the most widely used operating system in existence and underpins the development and execution of Microsoft's generation of windows based graphical user interface, from Windows 3.1 through to windows 95, 98 and 2000. Microsoft DOS (MS-DOS) was first introduced in 1981 and was marketed by IBM as PC-DOS; the two systems being virtually indistinguishable. The origins of this operating system lie in CP/M (Control Program for Microprocessors), the operating system for 8-bit computers popular in the late 1970s which used the Intel 8080 microprocessors. CP/M was created in the late 1970s as floppy disk drives became available for early personal computers, designed with as little as 16K RAM. MS-DOS is basically a clone of CP/M and was originally designed to facilitate the transition of 8-bit CP/M business software so that the software would run in the new 16-bit IBM PC environment. IBM originally approached CP/M's publisher, Digital Research, to write the operating system for its new computer, but as the result of a now legendary communications breakdown, Microsoft Corporation got the job instead.

Important Features of DOS

DOS is a command-line operating system with an interface that requires users to

memorise a limited set of commands, arguments, and syntax to use MSDOS computers successfully. After mastering DOS commands, users can achieve a high degree of control over the operating system's capabilities. The most severe limitation of DOS was for many years, the 640K RAM barrier that the operating system imposed on IBM PC-compatible computing. The 640K RAM 'barrier' was first broken by a new successor to DOS called OS/2 (Operating System 2) introduced in 1987.

11.21.4 UNIX

UNIX was the product of work at AT & T Bell laboratories during the early 1970s. Because Bell Laboratories were prohibited from marketing UNIX by the antitrust regulations then governing AT& T UNIX (the first version to gain significant distribution), the compiler was provided without charge to universities throughout North America, beginning in 1976. In 1979, the University of California at Berkely developed an enhanced version of UNIX for VAX computers. This version of UNIX led to other versions being made available commercially. In the early 1980s, AT& T gained the right to market the system and released System V in 1983.

UNIX is a comprehensive C-based operating system or programming environment that expresses a unique programming 'philosophy'. It provides:

(i) a filing system, with tree-structured directories;

(ii) a textural command language, based on command verbs followed by parameters;

(iii) a unique facility for joining pieces of software together called 'pipes';

(iv) a primitive set of facilities, e.g. a file copy tool.

UNIX is based on the concept of creating 'software tools', each of which performs one (and only one) function and together constitute the 'operating system'. In this environment, application programs need not rely on their own features to accomplish functions, but can take advantage of the software tools available. This philosophy helps keep application programs within manageable bounds. In UNIX, each element of software is called a 'filter'. A filter is a program that inputs a serial stream of information, processes it and outputs another serial stream. The communication of data from one software tool to another is accomplished via a 'pipe', a user command that couples the output of one command to the input of another. Pipes are highly flexible and enable the user to control virtually every aspect of the operating environment. It is possible to extend the command set to create commands for situations not anticipated in the operating systems development. Compared with DOS, UNIX is a highly flexible and powerful operating environment. However, it can exact a heavy toll on end users. With more than 200+ basic commands, inadequate error messages and a cryptic command syntax, UNIX requires a long 'learning curve' and imposes heavy burdens on users who do not use the system regularly. The main virtues of UNIX as a basis for software tools are:

(i) it encourages the use of existing software;

(ii) software is constructed in a highly modular fashion;

(iii) UNIX readily supports the dataflow software design method.

Once a dataflow diagram has been designed, it can be directly implemented as UNIX filters, connected by pipes. Unlike most PC operating systems, UNIX was designed as a multi-user system. With its multitasking capabilities, UNIX can perform more than one function at a time. In the past, these features have been in little demand by PC users of stand alone machines for running one application at a time. With current trends in personal computing, including the linking of workstations to corporate microcomputers and mainframes, UNIX is destined for greatness. UNIX for PC's is now readily available. Known as Linux, it is an emulating system to run UNIX on a PC environment and is one of the fastest growing operating systems currently available.

11.22 Programming Languages

There have been many programming languages developed since the early 1950s, some of which have come and gone while others have stayed and been continuously upgraded and enhanced. However, nearly all programming language have fundamental attributes in common and it is arguable that, in theory, the development of computer science is language independent. However, in practice, it is imperative that a language should be mastered as completely as possible, especially when it is based on the application of a specific field of interest as discussed in this book.

11.22.1 Factors in the Choice of a Language

Clarity, Simplicity and Unity of the Language

A programming language should provide the conceptual framework for thinking about an system as well as providing the means for expressing the system for machine execution. It is the semantic clarity which is arguably the most significant factor in selecting a language.

Clarity of Structure

The syntax of a language affects the ease with which a program may be written, tested, and later understood and modified. A language should have the property that semantic differences should be mirrored syntactically.

Naturalness of Application

The language should provide appropriate data structures, operations, control structures and a natural syntax for the problem to be solved. A language particularly suited (in both syntax and semantics) to a certain class of applications may greatly simplify the creation of individual programs in that area.

Ease of Extension

A substantial part of a programming task is to construct a library of functions specific to the application. This may be viewed in terms of language extension and the

language should allow extension through simple, natural and elegant mechanisms.

External Support

The technical structure of a programming language and its implementation is only one aspect affecting its unity. The presence of complete and usable documentation and a tested and error-free implementation are also strong determinants of the unity of a language.

Portability

An important aspect of many programming tasks is that programs are portable. A language which is widely available and whose definition is independent of the features of a particular machine forms a useful base for the production of portable programs.

Efficiency

Efficiency is one of the major concerns in the evaluation of a programming language. Efficiency of program execution is important in language design and of primary importance for large production programs designed to be executed many times. The efficiency of program compilation in certain languages rather than execution may be paramount. In such cases, it is important to have a fast and efficient compiler rather than a compiler which produces optimized executable code. Efficiency of program creation, testing and use is important in problems whose solution must be designed, coded, tested and modified with minimum waste of the programmers time and energy. In the following section, a number of programming languages are briefly reviewed giving a brief history of their development and some of their advantages and disadvantages.

11.22.2 FORTRAN

FORTRAN (FORmula TRANslation) was the first high level language to be developed. Earliest versions date from the mid 1950s but the first standard compiler to have widespread use (FORTRAN IV) was released in 1962. Over the 1960s and 1970s, various additional feature were added leading to the FORTRAN 77 version released in 1977. A new version of this language was due to be released in 1988 but failed to appear on the market, due primarily to the competition from the UNIX/C environment which is (for good reasons) currently dominating the computer market. A Fortran 90 compiler (first released in 1991 and upgraded in 1994) included a number of superior features such as dynamic memory management.

FORTRAN is a language which resembles elementary algebra, augmented by certain English words. Because of its similarity to ordinary algebra, FORTRAN is well suited to problems in mathematics, physics and engineering. FORTRAN is also applied to a wide variety of other areas, virtually any area requiring extensive manipulation of numerical data or (post 1977) character information. The large number of application areas together with the simplicity of learning and usage makes FORTRAN one of the most popular computer languages for scientific computing. There

is a significant amount of 'legacy code' which is FORTRAN based in the scientific and engineering communities.

Important Features

FORTRAN was designed with the primary goal of execution efficiency. It can be implemented on most computers so that execution is extremely efficient. It uses hardware structures directly for almost everything except I/O which makes it ideal for scientific computing. A FORTRAN program consists of a main program and a set of separately compiled subprograms, with translated subprograms linked together during loading. No run time storage management (dynamic memory allocation) is provided in versions that pre-date Fortran 90. Subprograms can only communicate by passing arrays and parameters through subprogram calls. Hence, the run time structure of a FORTRAN program is static.

Advantages and Disadvantages

Advantages

- The semantic structure of the language is perfect for translation of complex algebraic equations and algorithms used in engineering.
- Iteration algorithms are particularly easy to construct using FORTRAN DO-loops.
- Execution of load modules produced by most FORTRAN compilers is extremely efficient.
- FORTRAN provides a number of useful intrinsic functions for number and character manipulation.
- Because FORTRAN is an 'old' well established language it can be implemented on most computers and in most operating environments, given an appropriate compiler.

Disadvantages

- It is a poorly structured language and allows inexperienced programmers to produce a 'mess', e.g. 'spaghetti program' with uncontrolled use of GOTO statement.
- It was originally unable to access features associated with the operating system during run time. This was overcome to a limited extent on VAX/VMS systems (for example) in the mid 1970s using the run time library facilities; a sequence

of commands/statements which can be included in a FORTRAN program to access certain features of the VAX/VMS operating environment.

- FORTRAN has poor data typing and I/O facilities in general.
- The run time structure of a FORTRAN (pre-Fortran 90) program is static.
- There is no run time storage management.

Of all the disadvantages of the original FORTRAN programming language, its run time memory management is possibly its worse feature. In any programming language, this feature actively hinders modular programming especially on computers with low memory capacity and operating systems which can only address a low RAM (i.e. PC/DOS systems).

11.22.3 Pascal

Pascal was developed in 1970 to make available a language suitable to teach programming as a systematic discipline and to develop implementations which are both reliable and efficient. It is one of the major 'teaching languages' in the USA and Europe.

Important Features of Pascal

A Pascal program is composed of a data definition section, executable block definitions and an executable 'main' program block. The standard data types are those of integer and real numbers, logical values and the printable characters. Data may be allocated dynamically and there is a facility to declare new data types with symbolic constants; this extensionability is one of Pascal's most significant features. All objects such as constants, variables, procedures etc. must be declared before they are referenced. Pascal allows procedures and functions to be called with recursion.

11.22.4 Basic

Basic (Beginner's All-purpose Symbolic Instruction Code) was developed in the early 1960s. One of the objectives was to develop a language that could be learned quickly and was powerful enough to be used in solving problems from small scale to medium scale in any discipline. Basic statements and operands are very close to both the English language and algebra and is very easy to comprehend.

Important Features of Basic

Basic is commonly available on time sharing systems but is most commonly implemented on mini- and micro-computer systems. A Basic program consists of a single block of numbered statements. Subroutines and functions are within the same block and no independent subprograms are allowed. Therefore Basic does not facilitate any form of modular programming.

11.22.5 COBOL

COBOL (COmmon Business-Oriented Language) was developed in the late 1950's to provide a relatively machine-independent language for solving business data processing problems. A standard version was approved by the American National Standards Institute (ANSI). This standard version (ANSI COBOL) was revised in 1974 and has now been implemented by all major manufacturers. COBOL is a business data processing language designed to implement relatively simple algorithms coupled with high volume I/O. Because I/O is a prime concern, COBOL is designed with emphasis on features for specification of the properties and structure of I/O files.

Important Features of COBOL

A COBOL program is organised into four divisions:

1. The **procedure division** which includes the algorithm.

2. The **data division** which contains the data descriptions.

3. The **environment division** which specifies the machine dependent I/O devices to be used.

4. The **identification division** which begins the program and serves to identify it and its author, which also helps in providing program documentation.

The basic data types are numbers and character strings, with the basic structure being the record. COBOL has an English-like syntax which provides good self-documentation. This syntax makes writing even the simplest program a fairly lengthy process.

11.22.6 ALGOL

ALGOL (Arithmetic Language) was originally designed by an international committee during the late 1950's and early 1960's (ALGOL-60, ALGOL-68). The definition of ALGOL was particularly important in the development of programming languages. No other single language has had such a far-reaching influence on the design and definition of languages. ALGOL has been the central language on which much research in programming languages has been based. It is usually classified as a language for scientific computations because of its emphasis on numeric data and homogeneous array data structures. ALGOL has a particularly clear and elegant structure. The manner of its definition and control structure have influenced later versions of other languages (in particular FORTRAN and Pascal). The language has served for many years as the primary publication language for algorithms in a variety of computer science journals. Hence, a knowledge of ALGOL is a prerequisite to much advanced work in programming languages and related areas. ALGOL programs in general cannot be executed as efficiently as equivalent FORTRAN programs on conventional hardware. This is one major reason why ALGOL has never replaced FORTRAN in much of scientific computing.

Important Features of ALGOL

ALGOL is a language designed for compilation; programs and subprograms are compiled into machine code, with only a minimum of software simulation required for some of the primitive operations. Owing to the dynamic storage allocation and referencing environment, updating is necessary during execution. Data in ALGOL is restricted to simple homogeneous arrays of integer, real or Boolean elements. An extensive set of arithmetic, relational, and logical operations are provided. ALGOL provides two options for transmission of actual parameters to subprograms: transmission by value, and transmission by name. An ALGOL program is composed of a main program and a set of subprograms constructed of blocks - a set of declarations followed by a sequence of statements. The main program is just a single block.

11.22.7 PL/1

PL/1 (Programming Language 1) is a multi-purpose programming language. The original goal of PL/1 was to succeed FORTRAN and to include more extensive data structuring facilities. It is designed to be used by both scientific and commercial programmers and is constructed so that the individual programmer can program at his/her own level of experience.

Important Features of PL/1

A PL/1 program consists of one or more separately compiled external procedures. Internal and external procedures are identical syntactically but differ in that any variable in the external procedure is accessible to the internal procedure. PL/1 contains a wide variety of elementary data types, including an extensive set of possible type specifications for numeric data. The language is complex and is not easy to read or write. Its wide versatility makes it a relatively hard language to learn in its entirety.

11.22.8 APL

APL (A Programming Language) is a versatile programming language providing a direct means of problem solving. It is used interactively on a computer terminal. It is particularly well suited for the interactive environment because only a few characters typed on the terminal can lead to a relatively large amount of computing. APL is not suitable for the construction of large programs for repeated use in production computing.

Important Features of APL

There is no concept of a main program in APL. Subprogram execution is initiated either by a call from another subprogram, or by the programmer at a terminal through entry and execution of an expression containing a function call. In this sense, a programmer creates and executes the main program line by line during a session at the terminal.

11.22.9 C

C is a general-purpose and highly portable programming language designed for and implemented on the UNIX operating environment which is itself written in C. C has been closely associated with the UNIX operating system and like UNIX is a product of work at the Bell laboratories during the early 1970's. It is sometimes known as a 'systems programming language' because it is useful for writing operating systems. Indeed, the reason for being called 'C' is because the first attempts at writing a systems programming language were called A and B, with the third attempt leading to acceptable performance characteristics. Its generality and absence of restrictions makes it more convenient and effective for many tasks.

Important Features of C

C is a relatively low level language. It provides no operations to deal directly with composites such as character strings, sets, lists or arrays considered as a whole. It does not define any storage allocation facility other than static definition and the stack discipline provided by the local variables or functions. There are no read or write type statements and no wired-in file access methods. C provides the fundamental flow control constructions required for well-structured programs. The functions of a C program may be compiled separately. Variables may be internal to a function or external which are known only within a single source file or are completely global.

Procedure Oriented Programming in C

The basic principle of procedure oriented programming is to design libraries of modules which perform well defined functions or procedures that can be used by other systems as required. When the appropriate libraries have been completed, the software engineer can concentrate on the functionality of the 'system' (compounded in the design of a GUI for example) instead of having to design and/or re-design lower level processors. This approach to software engineering, for which the C programming language is ideal, tends to be used in software engineering projects in which the processes are relatively complex and require careful design and analysis but where the data types and throughputs are relatively simple. In DSP, many of the procedures are oriented toward the efficient processing of single and/or multi-dimensional arrays (2D DSP) of floating point numbers. It is this approach (i.e. a procedure oriented approach) that is emphasised in Part III of this work and extended further in Part IV with regard to building a DSP module library.

C versus FORTRAN

Although C is a general purpose language and can therefore be used in 'number crunching' applications, FORTRAN usually performs better in such applications. This is because: (i) a large number of efficient mathematical libraries have been developed using FORTRAN (e.g. numerical solutions of Partial Differential Equations for Computational Fluid Dynamics); (ii) FORTRAN is a language which can easily be optimised by advanced compilers.

The case for using C in numerical computations rests mainly with the fact that it

supports memory allocation. In linear algebra or signal processing applications, this can be very important. However, with the advent of Fortran 90, the situation has changed somewhat. Fortran 90 supports a range of features originally attributed to C such as memory allocation and pointers at a higher level, so that it is, in general, easier to use. Care has also been taken during the definition of the language to preserve the potential for optimising compilers. Finally, FORTRAN 77 code can be mixed with Fortran 90. For strictly mathematical applications, Fortran 90 may arguably be a more useful language. However, for a large number of domains such as interface development, operating systems, interaction with hardware and so on, C/C++ is generally superior and should be used whenever possible.

11.22.10 C++

C++ was developed as an extension of C and in one sense can be seen as a version of C with greater functionality. However, in a significantly more important sense, the language was developed to enhance the programming methodology of C to include the principles of object orientation which provides an approach to the re-use of software.

C++ can be viewed as C with classes, a *class* being the C++ term for an abstract data type. C has a way of aggregating data into a complex type - the *struct*; a class is a struct with a few more features. Part of the data can be kept 'private', with the only access to this data through the 'methods' associated with the class. A *method* is just a C function, but is bound to a particular class and has access to all of the class's data. A class that is entirely 'public' is the same thing as a *struct* and can be defined as a struct. Classes also have the capability of inheritance. The programmer can 'inherit' all of the data, and methods, of another classes and add what is needed. This is particularly useful for code reuse. In C and other conventional languages, if some code does not meet our needs, we at least must modify the code. Inheritance allows us to take an existing class and augment or modify it as necessary. In fact, a C++ class has the capability of 'multiple inheritance'; a class can inherit data from more than one other class.

11.22.11 Java

Java is a relatively new language but borrows from existing ones, the best parts being taken from C++, Smalltalk and Lisp. It tries to balance functionality, speed and portability. Compared with C/C++ which is fast, unsecure and reasonably portable, Java is relatively fast, secure and very portable.

Java originated from a programme to develop software for information appliances (e.g cellular phones and personal digital assistants etc). The goal was to enable the transfer of information and real time applications over packet based networks. This required small, efficient code which was safe and robust. The development of the language began with C++ but this was found it too big and complex. Instead the language was developed from scratch leading to a new language called Oak. Attention shifted to interactive TV set top boxes and although oak fitted the requirements well the market was not there. Instead Sun targeted the Internet and World Wide Web. Oak was developed as a small, robust, architecture independent and object oriented language which were the ideal requirements for a universal, network based

programming language. Hence oak was modified a little to the new environment and renamed Java.

In comparison with C++, Java does not allow the explicit use of pointers. Instead it uses references, a kind of safe pointer (no arithmetic allowed). Java has no operator overloading (except + for strings) and only allows single inheritance. Multiple inheritance is replaced with Java interfaces and all methods (unless explicitly declared) are virtual or dynamically bound, i.e. Java is a late-binding language. Hence all over-ridden methods are selected at run-time. There is no preprocessor and hence no define type statements are required with Java and there are no *include* header files. All basic types (e.g. int, float, double etc.) have a platform independent fixed size. Java has automatic garbage collection and there is no need for destructors and delete operators. There are no pointers to methods (simulated by use of interfaces and callbacks). Instead, Java has true arrays, i.e. arrays are first class objects.

11.23 Object Oriented Programming

Object-oriented programming (OOP) is characterized by many concepts: inheritance (single or multiple), dynamic binding, polymorphism, and information hiding. These concepts overlap somewhat but are relatively easy to define. OOP itself involves designing software around the 'objects' in question. These objects are instances of 'abstract data types' or 'classes' (the C++ handle for an abstract data type). An abstract data type is simply a definition of a complex data type *as well* as the 'methods' that can act upon the data. Nothing can get at the data that constitutes an abstract data type without using the data type's methods.

Another way of defining OOP is just 'data centered design'. Design the program around the data you will be acting on. It is unlikely that the fundamental data you are working with will change significantly. However, the functionality (what you will do with the data) will be constantly enhanced and modified as time goes on. Typical structured programming techniques, such as 'functional decomposition', concentrate on what a program *does*, rather than what it does it *to*. Such design methods require large-scale overhaul of the software when changes to the functionality are required.

A *class* may be described as an 'abstract data type'. It is an aggregation of related data elements together with all the 'methods' that may operate on that data type to represent a unified concept. For a true abstract data type, the only access to the data itself is through the defined methods. An *object* is an instantiation of a class. Anything that is of the abstract data type that the class defines is an object. In a pure object-oriented program, we work only with objects. Our only function calls are messages to objects (or invocations of the objects' methods).

Information hiding refers to the concept that our only access to the data in an abstract data type (or a class in C++ terminology) is through the 'methods' defined in those classes. In fact, all of the data and methods have some attribute of information hiding associated with them. These attributes in C++ terminology are referred to as 'public', 'private', and 'protected'. 'Public' methods or data are available to any other class. Most methods are public. Exceptions are those methods that are used internally by the class to implement its public methods. Most data is not public, and there should be few exceptions to this rule. 'Private' methods and data are available only to objects of the specified class. In general, all data used to represent an object

of a particular class should be private. Private methods are used only to implement methods of the specified class. 'Protected' methods or data are available only to objects of derived classes, where a derived class is a class that 'inherits' its properties from a parent class.

Information hiding when used to maximum effect has several benefits: reliability, understandibility, and, in some cases, efficiency. Software developed with information hiding tends to be more reliable since access to the data representation is restricted to those implementing the abstract data type. 'Clients' of these abstract data types (developers using the code in their own applications) have no direct access to the data itself. Any actions on the data are taken through the controlled gateways of the class's methods, which presumably are written such that the integrity of the data is ensured. Once the methods are written and debugged, use of the class should be completely safe.

Understandibility is enhanced since client developers need not understand the inner workings of the class's methods or even the data representation of the object. The only thing a developer using the class needs to understand is the 'class interface', or the set of methods available for that class. In a C++ program, this can be reduced to a procedure as simple as reading the method prototypes listed in the public section of a class definition in a header file. Of course, the developer also needs to know what each method requires to operate and what state the object is in after the method is called. It is important that the documentation for any class interface contain this information in a very accessible way.

11.23.1 Inheritance

Inheritance is the creation of a new class as an extension or specialization of an existing class. It allows the conceptual relationship between different classes to be made explicit. For example, the class *Apple* might be a descendant of the class *Fruit*. It is mostly identical to the class *Fruit*, but it may include some additional methods or data. Instead of duplicating the identical parts of the two classes, we can say that *Apple* 'inherits' its definition from *Fruit*. In this case, only the additional methods and data need to be specified in the child class. Thus, inheritance allows not only easy comprehension of relationships between classes but also much easier construction of the classes themselves.

Inheritance also allows existing classes and class libraries to be easily modified to suit the job at hand. Inheritance enhances reusability of code, since existing code need not be modified at all. Existing classes can simply be inherited and the changes made to the derived class. Inheritance may be single or multiple. Single inheritance implies obtaining characteristics from just one parent class. The preceding example used single inheritance. Multiple inheritance obtains methods and data from multiple parent classes. An example might be an object of class *CircularWindow* that is derived from classes *Window* and *Circle*; it has available all the methods peculiar to circles as well as the methods appropriate to windows.

Another characteristic of inheritance is the ability to treat objects of different classes as instances of the 'generic' parent class. This allows us to handle groups of disparate objects as instances of a more uniform class. An example might be a generic class called *Shape* that has several derived classes, such as *Rectangle, Triangle*, and

Circle. These derived classes will share methods and characteristics of the data of the parent class.

If we have a heterogeneous group of objects that share a parent class, we can treat it as a group of homogeneous objects as if they were of the same type. For example, we could group all the different *Rectangle, Triangle*, and *Circle* objects in a linked list of shapes. If we have the capability of sending identical commands to these different objects, this feature is even more useful. There is another type of 'genericness' that does not involve inheritance. Parametrised types allow the creation of a class whose data representation can be changed as the need arises. For example, a vector class could be constructed such that the data representation appears as a parametrized. This allows a generic class to which a parameter can be supplied to create specific classes for the different types of vectors. Such classes in C++ are called 'template' classes.

11.23.2 Virtual Functions and Abstract Classes

If the only purpose of a class is to be a parent for more specific derived classes, we may wish to have some methods in this base class be 'virtual methods'. In this case, if a pointer to the parent object invokes the method name and the object is actually of the derived class, the derived class method is invoked. For example, the class *Shape* may have a *draw* method that is defined (has some code associated with it) but is listed as 'virtual'. In this case, if the pointer is of class *Shape* but the actual instantiated object at run-time is of a derived class such as *Rectangle*, the *Rectangle draw* method will be invoked.

If we decide that a particular method will never be defined for a class, we can define the method as a 'pure virtual' method, and no code will be associated with it. This makes the class an 'abstract class'. No object of an abstract class can ever be instantiated.

Pointers may be of the class type, but they must actually refer to objects of derived classes. The notion of an abstract class is especially powerful in conjunction with a system that lets us group disparate objects on a common parent class (such as a linked list of shapes, where the individual shapes actually might be a circle, a triangle, and a rectangle) and send identical messages to each of the objects, which must each respond appropriately to them.

11.23.3 Polymorphism

The concept of sending different messages to different types of objects in object-oriented programming is referred to as *polymorphism*. Polymorphism allows us to send identical messages to different objects and have each object respond appropriately. An example above was given where we might want to invoke methods on disparate objects with the same parent class. We may invoke a *display* method on each member of a linked list of shapes, the individual objects of which might be triangles, rectangles, and circles. However, it is not necessary for the objects to be grouped or related for polymorphism to come into play.

At the simplest level, two completely unrelated objects of two different classes may each receive a *display* command, and each will respond to the command differently. In

this case, polymorphism is required, but the specific code to be executed can actually be determined at compile-time.

In the previous example, objects of several different classes were grouped together as a linked list of pointers to objects of the parent class. The linked list might be built dynamically, in which case we would not know at compile-time what specific code would be executed for the *display* method. This would have to be determined on the fly at run-time for each invocation of a method on any element of the linked list. Often when polymorphism is mentioned, this is the phenomenon that is referred to. This definition of polymorphism is a bit stricter, implying the capability to send different commands to different objects that are grouped as heterogeneous collections of descendants of a common base class. Under this definition, it is the combination of inheritance and the ability to determine the actual code at run-time (*dynamic binding*) that enables us to perform polymorphism.

11.23.4 Dynamic Binding

Dynamic binding might also be referred to as 'late binding' because the method is not bound to specific code until as late as possible (when the method is invoked at run-time). Dynamic binding requires that a 'method table' be maintained during execution. When the *display* method is called for an object, the class of the object is determined first. Then the actual function to be executed for the method is determined by looking up the method name and class identifier in the method table. Thus, with dynamic binding we incur not just the overhead of the function call when invoking a method, but also the expense of the method lookup.

11.23.5 The C++ Programming Language

During the 1980s the C language emerged as one of the most universal programming languages. It made it possible to write code that was portable to a wide class of computers. Software could be written faster and projects grew in size. This in turn led to increased complexity and development times. The language C++ was developed to make programming easier. To enable this goal to be achieved the language itself had to be more complex than C. C++ is a compatible superset of the C programming language, providing extensions to support the object-oriented programming (OOP) methodology and was developed in the early 80's at AT&T Bell Labs by Dr Bjarne Stroustrup. All the added features of C++ are designed with the aim of reducing levels of difficulty. The language's name C++ is a play on C's ++ increment operator. The language C++ is literally 'one step beyond C'. The inventor of the language, Bjarne Stroustrup of AT&T originally called the language 'a better C'. The major extension to the syntax of C was the implementation of the 'class' construct. Since a great deal of C code already existed, C++ compilers had to be designed to ensure that the language was compatible with C; any C program should compile and execute properly in the C++ environment. In addition, extensions to C were added and in the end a very high degree of compatibility was achieved. There are few changes, as opposed to additions, to the syntax of C++ from ANSI C and almost all ANSI C programs run in the C++ environment.

An important advantage of using C++ is that C programmers can switch gradually

to C++ without wasting their C programming skills. Today ANSI C and C++ are close counterparts and most ANSI C programs are compatible with C++. Hence existing C code can normally be compiled without modification by C++ compilers.

Most aspects with regard to programming in ANSI C apply to C++. ANSI C and C++ programs look pretty much the same. They use nearly identical syntax, as well as the same kind of loops, data types, pointers, and other elements. Although C++ was designed to make programming easier the mere adoption of the language does not automatically guarantee better or simpler software. To reap the real rewards of C++ it is necessary to adopt a new programming methodology, commonly referred to as *object oriented programming* or *OOP*. To enable this methodology to be implemented, in addition to the facilities provided by C, C++ provides a variety of new facilities such as those for:

- data abstraction
- multiple inheritance
- strong type checking
- passing arguments by reference
- guaranteed initialization
- automatic cleanup
- operator and function overloading
- type conversions

The terms *data abstraction* and *inheritance* are the basic ingredients of the OOP methodology which allows programmers to write applications using concepts and notations natural to the application domain. An object is a collection of code and data designed to emulate a physical or abstract entity. Data abstraction allows programmers to elegantly model application domain objects such as input/output devices, robots, and employees. Inheritance allows for the preservation of relationships between object types that are related to each other, e.g. employees and managers, students and tutors etc. In addition, the twin concepts of data abstraction and inheritance together encourage the reuse of existing code by making it available in packaged form and allowing it to be enhanced or specialized.

OOP is a methodology that gives great importance to relationships between objects rather than implementation details. Relationships are ties between objects and are usually developed through genealogical trees in which new objects are derived from others. Hiding the implementation details of an object results in the user being more concerned with an objects relation to the rest of the system rather than how an objects behaviours are implemented. This distinction is important and represents a fundamental departure from earlier languages in which functions and function calls were the centre of activity. C++ is an important software engineering tool, especially in the context of writing large programs.

Software engineering aims to produce quality programs that take less time to debug, maintain and port. When used well and as intended, C++ facilities lead to

programs that are easier to understand and maintain because data can be modelled in terms of the application domain.

11.23.6 C++ as an Extension of C

C++ extends C in three principal ways:

- It encourages and supports development of *Abstract Data Types (ADTs)*.
- It supports program design and construction using *object-oriented programming (OOP)* principles.
- It implements many smaller extensions and improvements to the syntax of the C language.

ADTs are types defined by the programmer in addition to those supplied as part of the language. In normal C the nearest approach to ADTs is through the *typedef* mechanism. Here the user creates a new type based on a structure declaration. An ADT in the OOP context also includes definition of the operations which may be carried out on its data components. The internal implementation of an ADT is hidden from all other operations which are not part of the ADT. C++ implements ADTs using the OOP approach. Because of the way ADTs package their data and operations, software designed with them tends to be more modular and less complex than in traditional methods. The interfaces between modules are well-defined and interdependences across modules are few. This promotes easy division of tasks among software developers. Also, because the internal implementation of an ADT is hidden, there is some assurance that a module will not be tampered with by outside code.

C++ implements ADTs and OOP in the following ways:

- It implements *objects*, defined as *classes*, which incorporate not just the data definitions found in C structures but also declarations and definitions of functions which operate on that data. This *encapsulation* of data and functions in a single object is the main innovation of C++.
- Instances of classes may automatically be initialised and discarded using *constructors* and *destructors*. This prevents program initialisation errors.
- The way in which C++ classes are defined enforces *data hiding*; data defined in a class is by default only available to the member functions of that class. External, or client code which uses a class cannot tamper with the internal implementation of the class but is restricted to accessing the class by calling its member functions.
- C++ allows *overloading* of operators and functions. More than one definition of a function may be made having the same name. The compiler identifies the appropriate definition for a function call. Ordinary operators such as '++' and '-¿' can also be overloaded with additional meanings.

- C++ allows the characteristics of the 'class' type - data and functions - to be *inherited* by sub classes, called *derived classes*. The class being derived from is called a *base class*. The derived classes may in turn add further data and function definitions. This facility encourages it code re-use and the generation of shareable *class libraries*. This gives cost and time savings in the software development process.
- Multiple inheritance is possible; here sub classes can inherit characteristics from more than one base class.
- C++ allows classes to define *virtual functions*; more than one definition of the functions with the same prototype. The decision as to which one is selected is taken at *run-time*. This concept is called *polymorphism*, and the run-time selection among function definitions is called *late binding* or *dynamic binding*.
- Template classes can be defined. These allow different instances of the same class to be used with data of different types but with unchanged code. This also promotes code re-use.

C++ facilities for OOP are characterised by classes, inheritance and virtual functions. This makes C++ a good language for writing software to handle a multitude of related objects. A particularly appropriate use for C++ is in the design and implementation of *Graphical User Interfaces* (*GUIs*). Here many different but related objects are represented on the screen and allowed to interact. Using the OOP approach, these objects are stored in class hierarchies. By use of virtual functions a generic interface to the objects can be presented. Hence the programmer does not need to know the detail of how the objects are manipulated.

C++ is a hybrid language. It is intended to be used for OOP but can be used in a procedure way (C programs run in the C++ environment). There are other OOP languages (e.g Smalltalk) which force the OOP approach. Data type checking and conversion in C++ are stronger than in traditional C. Function prototypes make the transfer of parameters at the function call interface less error prone. C++ provides a large number of small syntactic improvements over C. There is a new reference mechanism, simplifying the pointer dereferencing approach of C. Allocating and deallocating memory is simplified. A stream (I/O) library is implemented. This defines in a class hierarchy streams for input and output. C++ is an extension of C which, by facilitating improved design practices, provides an environment for the development of software that better reflects the 'real world'. It also reduces complexity and increases reliability.

11.23.7 The Object Oriented Programming Paradigm

The OPP paradigm and other programming paradigms can be characterized as follows:

Procedure Programming

The program is designed as a collection of functions which are then coded using optimal algorithms. Almost all programming languages support the procedure programming paradigm.

Modular Programming

The program is designed as a collection of modules. Each module hides its data from the other modules and provides a set of interface functions which must be used to access the data. Hence, provided the interface functions remain the same, module internals can be modified without effecting client modules, i.e. users of the interface functions. The client modules do not even have to be recompiled. The languages Modula, Ada, and C (files can be used as modules) are examples that support the modular programming paradigm.

User-Defined (or Abstract Data) Types

Designing a program by defining new types and manipulating objects of these types is essentially the same as modular programming with one exception: user-defined data types, unlike modules, are fully fledged types. C++ supports the programming paradigm based on abstract data types. Note that the C *typedef* facility is not really a fully-fledged facility for defining new types because it does not support *data encapsulation* and operations cannot be associated with the new types.

Object-Oriented Programming

This approach is similar to the programming paradigm based on abstract data types except that inheritance can be used to define related types. Inheritance supports code reuse by factoring out the common components of related types. C++ is a prime example of a language that supports the OOP paradigm.

Object-oriented software design is the construction of a software system as a structured collection of objects that interact with each other. Programs are written, as far as possible, using concepts that are natural to the application domain. Object-oriented software design enhances program readability and, at the same time, it speeds up the program development process and makes it easier to maintain programs.

An OPP language should provide the following facilities (in addition to the conventional facilities found in languages like C):

- A mechanism for defining first class object types which have the same rights and privileges as the predefined types.
- Inheritance, i.e. a mechanism for 'deriving' new types from old ones.
- Dynamic binding, i.e. invocation of the appropriate interface function to operate upon an object even though the object type may not be known at compile time.

OOP languages provide a mechanism (in particular the *class* facility in C++) to define new types to easily and naturally model entities in the application domain. Proper design of the class 'user interface' is critical because otherwise the class may not have the right functionality. This may result in it becoming awkward to use or it may simply not be used at all.

11.23.8 Data

Most languages provide a set of primitive data types, which are used by programmers to build more complicated data structures such as symbol tables, lists, trees etc. These data items must be manipulated according to the semantics specified by the implementer. However, in may high level languages such as FORTRAN and Pascal, the data structure implementer has no way of ensuring that the data is manipulated correctly and consistently with respect to the specified semantics. Even if the implementer provides routines to manipulate the data, the compiler cannot insure that these 'interface' routines are actually used to manipulate the data. It is often the case that programmers will tend to manipulate the data structures directly, by-passing the interface routines to increase program execution speed, functionality and so on. This can lead to errors and it makes the resulting code become dependent upon the specifics of the data structure implementation.

Using data abstraction facilities, data structures are defined as 'classes' which are fully-fledged types that can be used like the predefined types. Users of class objects can manipulate them (i.e manipulate the data encapsulated by the objects) using only the items ('data members' and 'function members') specified in the 'public' part of the class specification. Data abstraction ensures that programmers use and manipulate objects without knowing about or directly accessing their internal representations. That is, data abstraction facilities are mechanisms for defining 'black-box' classes. Just as procedure abstraction allows programmers to build libraries of types such as strings, complex numbers, polynomials, simulation objects, and so on.

Simple data abstraction does not allow customization of existing classes and does not support code reuse. For example, assume that a class *employee* has been defined. Suppose now that the programmer wants to define another class called *manager*, which is similar to the class *employee* but it has some additional components ('members' in C++).

Without inheritance, two strategies can be used for defining both employees and managers:

(i) A new *manager* class can be defined from scratch. This strategy results in code duplication because *employee* and *manager* will have many members in common. Hence, changes to class *employee* may require corresponding changes to class *manager*. Another disadvantage of this strategy is that is does not preserve the relationship between the *employee* and *manager* classes, i.e, class *manager* is a specialization or customization of class *employee*.

(ii) Class *employee* can be modified to include additional information for managers. Modifying an exiting type violates the concept of modularity and it may break existing code that works. Also, users of class *employee* will now have to understand the fact that class *employee* contains members especially for implementing *manager* objects. Further changes to manager-specific members of class *employee* will affect all users of class *employee* regardless of whether or not they refer to employees who are managers. Finally, modifying an existing type will require recompilation of existing code.

The 'type inheritance' mechanism in languages such as C++ allows new types to be 'derived' from existing types. The new types inherit properties of the types they are derived from (i.e. the properties of the base types). Type inheritance allows the

development of customized types by allowing new attributes to be added to existing types without requiring modification to these types. Hence, inheritance eliminates code duplication and at the same time it preserves the relationship between the 'derived' and 'base' types. Data abstraction and inheritance are orthogonal concepts and they represent real advances in the repertoire of software building facilities.

The primary extension C++ makes to C is the addition of the *class* facility, which supports both data abstraction and inheritance. The language also extends C in many other ways that facilitate the development of both small and large programs. For example, C++ allows users to define constant variables, pass arguments by reference, overload functions and operators, and so on.

11.24 Discussion

The brief overview of the programming languages discussed here has been designed to introduce the reader to some of the more common languages available. However, like any natural language, a programming language is not static and is continually being upgraded and improved upon in line with technology and competition from the facilities available with other languages. Also, many of the languages discussed here have been used to create more specialized versions based on 'variations on a theme' in order to satisfy the demands of a particular technology. This is particularly so with COBOL which has been re-cast into many different forms for applications in the business, financial and banking industries. Further, over the 1990s, nearly all such languages and compilers have been modified to operate in a windows environment and have a variety of 'visual' forms (e.g. visual Basic, visual C++ etc.). Moreover, CASE tools have been introduced where possible and desirable to help the programmer undertake projects that were beyond the scope of a single individual in former times. In all languages, it is typical to introduce more and more libraries of intrinsic functions, I/O facilities and graphics options especially for the development of graphical user interfaces.

The use of a programming language depends critically on its historical legacy and the code and systems that have been developed through its application. However, more recently, the commercial environment in which software products are expected to perform has had an even more significant effect on dictating the programming language and operating environment that is used. With the rapid increase in the availability and computing power of personal computers, many companies have been forced to develop versions of their products (originally developed for workstations or even a mainframe for example) for the PC market in order to remain in business. The reason for the growth in the PC market is not that PC computing and the operating environment that comes with it is better of worse, but that it provides the user with a degree of independence (a time invariant human need). It could be argued that the PC has, to the 1990s person (to be PC), become what the Ford model T automobile was to the 1920s man. In addition to the above, the IT revolution, which in a sense, involves the application of digital signal processors as much as it does central processing units, has become dominated by systems which are, for better or worse, based on the exclusive use of the C programming language and its derivatives (C++, Java, Java plus and many others, now and in the future). The C programming language and PC based operating environments, whether they are

based on Linux or Microsoft windows, have become the principle attributes of the majority of software engineers in nearly all areas of commerce, science and engineering. Further, many high-level programming environments are actually based on C. One such environment is MATLAB which includes a DSP toolbox and provides the facility to output MATLAB code (.m files) in C. It is within this context that the following chapter should be studied.

11.25 Summary of Important Results

Decimal Number Systems
Systems that count in 10's using the marks 0-9 to represent a number and weights of the type 10^n.

Binary Number Systems
Systems that count in 2's instead of 10's using the marks 0 and 1 to represent a number and weights of the type 2^n.

Binary Coded Decimal (BCD)
Used to represent decimal digits directly where each decimal digit is replace by its binary form.

Overflow
The effect caused when the number of bits required exceeds the word length which sets a limit on the magnitude of the number that can be stored.

Rounding Error
Error that occur when not all the significant digits of a number are given.

Computational Error
Error as a result of performing arithmetic operations, usually caused by overflow or rounding to intermediate results.

Truncation Error
Error that occurs when a limited number of terms of an infinite series (representing a certain number for example) are taken to approximate that number.

Algorithmic Error
Error incurred by the execution of an algorithm, an algorithm being a set of procedural steps used in the solution of a given problem.

Accumulation Error
Error that accumulates as a numerical solution evolves which is typically based on a poor representation of floating point numbers by a limited number of decimal places for example.

Linear Error Growth

$$R(\epsilon) \propto n\epsilon$$

where n is the number of operations and ϵ is the absolute error that occurs with each operation.

Exponential Error Growth

$$R(\epsilon) \propto K^n(\epsilon)$$

where K is some constant > 1, n is the number of operations and ϵ is the absolute error that occurs with each operation.

Nesting
A method of maintaining accuracy by reducing the number of operations, thereby reducing error accumulation used in the evaluation of polynomials for example.

Batch Adding
Grouping numbers of similar magnitude, adding them together and then adding the result to the output obtained from adding another group of numbers with a similar magnitude and so on.

Conditioning
Assessment of the possible ill-conditioned nature of a numerical solution to a problem (e.g. see Chapter 8).

Subprogram
A component part of a program referred to as a subroutine, function, void function or module which undertakes a specific and well defined processing task.

Function
A module that processes data and returns the results via the function name, e.g. intrinsic functions such as cos, sin, exp, log etc.

Procedure
Any defined way of carrying out some action or actions using well defined operations on well defined data using program components; a module which inputs data and returns the result through some parameter, set of parameters or an array for example.

Modular Programming
Design of a system based on constructing a library of functions which process serial streams of data in a specific and well defined way that is unique to each module.

Structured Programming
Design of a module that is based on a well defined set of procedures excluding the use of the goto statement.

Object Library
A collection of objects or compiled functions and procedures in a library that is specific to a given task or a set of tasks.

Bottom-up Design
Development of a system that is based on developing object libraries that start with functions that are one level above those provided by the compiler and progressively working upwards level by level, each level being ideally based on those modules that have been developed for the object library at the level below.

Top-down Design
Development of a system that is based on considering an overview of the system at a top level first and progressive working down level by level developing object libraries as required which are initially designated by stubs at the level above.

Jackson Method
Design of a module in which the logic and structure of the procedure that processes the data reflects that of the data itself.

Requirement Specification
Specification of those aspects of the system design that include a suitable road-map, functionality, function definition, constraints and a schedule.

Process Specification
Specification of the processes that a system requires in order to accomplish the task(s).

System Specification
Statement of the system design which includes and statement of user requirements, functional specification, technical specification and module, object library and system testing.

Program Specification
Part of the system specification which defines issues including program design, efficiency, development time, costs and documentation.

Software Life Cycle
The cycle of events that covers the development of a software system from specification through design to its implementation, testing and maintenance.

Bersoff's Law of System Engineering
'No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle'

Object Oriented Design
Application of programming techniques and paradigms that aims to produce re-usable

software or objects, i.e. software engineering which is oriented toward the use of object libraries composed of functions that are accessible and in the widest possible context.

CASE Tools
Computer Aided Software Engineering systems that aid the programmer in the design of software from the graphical user interface to code generation.

11.26 Further Reading

- Dahl O J, Dijkstra E W and Hoare C A R, *Structured Programming*, Academic Press, 1972.
- Coleman M and Pratt S, *Software Engineering for Students*, Chartwell-Bratt, 1986
- Pressman R S, *Software Engineering: A Practitioners Approach*, McGraw-Hill, 1987.
- Abel P, IBM PC Assembler Language and Programming, Prentice-Hall, 1987.
- Yakowitz S and Szidarovszky F, *An Introduction to Numerical Computations*, Macmillan, 1989.
- Morgan D, *Numerical Methods*, M& T Publishing, 1992.
- Bell D, Morrey I and Pugh J, *Software Engineering: A Programming Approach*, Prentice-Hall, 1992.
- Denton N and Hill G, *Systems Construction and Analysis: A Mathematical and Logical Framework*, McGraw-Hill, 1993.
- Booch G, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, 1994.
- Runciman C and Wakeling D, *Applications of Functional Programming*, UCL Press, 1995.
- Fogiel M, *Handbook and Guide for Comparing and Selecting Computer Languages*, Research and Education Association, 1985.

11.27 Problems

11.1 Write the decimal number 8.25 in the following forms:

- Binary
- Binary Coded Decimal

11.2 Explain the meaning of:

- Fixed point storage of binary numbers
- Floating point representation of binary numbers

What are the advantages of using a floating point representation?

11.3 What are the three basic constructs of a structured program? Briefly explain their purpose.

11.4 Review the arguments for and against *goto* statements in the context of structured programming.

11.5 Convert the following pseudo code into a structured form: [5]

```
      i:=start
loop:
      IF array(i)=x, goto Found
      IF i=end, goto notFound
      i:=i+1
      goto loop
notFound:
      write 'not Found'
      action1
      goto finish
Found:
      write 'Found'
      action2
finish
```

Compare and contrast the two programs in terms of their structure and their respective cyclometric complexity measures.

11.6 Within the context of the 'system life cycle', discuss the principal issues associated with system design commenting on the following:

- specification;
- design;
- testing;
- software maintenance and defect amplification.

11.7 Explain why the 'system life cycle' is inherently dynamic and how this process is compounded in terms of 'Bersoff's law of systems engineering'.

11.8 An aeroengine company require a micro processor-based monitoring system for a prototype gas turbine engine that receives input from the following sensors each of which has a boolean output:

- Temperature Sensors: detect temperature threshold of combustion chamber.
- Pressure Sensors: detect pressure threshold of primary compressor.
- Vibration Sensors: detect critical mode of vibration.
- Fuel Sensor: detect fuel flow threshold.

When activated, the system should be capable of performing the following Activation Responses:

- Generating Alarms
- Turning On Lights
- Printing Out Messages

The control module has buttons that allow the following programming:

- Adjusting Sensor Levels: Each sensor can be adjusted to be activated by a range of INPUT (environmental) conditions.
- Selecting Response List: Each sensor can activate any combination of Activation Responses.
- Setting Mode: The system has two modes: ON, OFF.
- Inactivating: After activation, it is possible to halt the responses and reset the system by inactivating it.

(i) Design a functional description for the system that would suffice for the 'System Specification'.

(ii) Prepare Data Flow Diagrams for levels 01 (toplevel) and 02 (next level).

Chapter 12

Modular Programming in C

This chapter has been written to provide the reader with a quick-guide to programming in C with an emphasis on those features of the language that are primarily required to undertake the problems set in Part IV. Clearly, it is not possible to give a complete account of the language in a single chapter, but, as in many applications that focus on a specific theme, the aspects of the C programming language that are required to develop DSP modules form a relatively small subset of the languages full capabilities. It is this subset, together with some necessary examples and discussions that are presented here.

12.1 About C

C was crated by Dennis Ritchie of the Bell Labs in 1972 and first appeared in the public domain with the publication of *The C programming Language* in 1978 by Kernighan and Ritchie. C is a general purpose programming language which, from its conception, has been closely associated with the UNIX operating system which is written in C. C is a relatively low-level language because it puts the programmer 'very close to the machine'. It incorporates the control features that computer science theory and practice find desirable. The design of C makes it natural for users to use top-down planning, structured programming and modular design. It is a relatively efficient language because its design takes advantage of the abilities of current computers. C is also a portable language which means that C programs written on one system can be run with minimal or no modification on another system. It exhibits some of the fine control usually associated with assembly languages. Programs can be fined tuned for maximum efficiency. It is also sufficiently structured to encourage good programming. Finally, C has become one of the most widely used languages available. There is an abundance of compilers available, many of them free and accessible via the Internet and is currently one of the preferred language for communications engineering and signal processing in general.

12.1.1 Statement Layout

There are no restrictions placed on the layout of C code. It can be written in free form with any number of spaces and/or new lines used to separate the words and symbols. Each statement, which is case sensitive, must by separated from the next by a semicolon. A C program or subprogram (a function) always starts with a left brace { and ends with a right brace }. For example, the code

```
main()
{
        printf(".c is the extension used
                   for a C program file name.\n");
}
```

defines a main program with no I/O which prints the text to the screen using the

```
printf
```

function together with a carriage return using

```
\n
```

12.1.2 Documentation

Comments can be inserted anywhere in the program provided they are enclosed by the symbols /* and */. Comments do not nest but can appear anywhere that a blank or new line can appear. For example,

```
/* This is a C comment line */
```

12.1.3 Input/Output

C provides no I/O facilities, i.e. there are no read or write statements and no wired-in file access methods.

Unformatted Output

Unformatted output text to the screen is achieved using the *puts* function which has the form

```
puts("Text.....");
```

Formatted Output

Formatted output is achieved using the *printf* function which has the form

```
printf("control string", arg1, arg2, ...);
```

which converts, formats and prints its arguments on the standard output under control of the string control. The control string contains two types of objects: ordinary characters, which are copied to the output stream and conversion specifications, each of which causes conversion and printing of the next successive argument to *printf*. Each argument is represented in order within the string by the specification

```
%type
```

where type includes one of the following:

```
d /* decimal integer                              */
u /* unsigned decimal integer                     */
c /* single character\index                       */
s /* string (null-terminated)                     */
f /* floating-point number                        */
e /* floating-point number in exponential format */
g /* %e or %f whichever is shorter               */
```

The 'length' n of the type is specified thus,

```
%ntype
```

Input is undertaken with a format control based on the function *scanf* which has the form

```
scanf("control string", &var1, &var2, ...)
```

12.1.4 Date Type Statements

There are only basic data types in C:

char is a single byte, capable of holding one character in the local character set.

int is an integer, typically reflecting the natural size of integers on the host machine.

float is for single-precision floating point.

double is for double-precision floating point.

N.B. by default, most C compilers store floating point numbers using double precision.

There are a number of different qualifiers which can be applied to integers such as *short*, *long* and *unsigned*. *short* and *long* refer to different sizes of integers. *unsigned* numbers obey the laws of modulus 2^n arithmetic were n is the number of bit in an integer. *unsigned* numbers are always positive. For example:

```
short int x;
long int y;
unsigned int z;
float a,b;
double c,d,e;
```

Data can be 'cast' from one type using another using casting. For example, if x is of type *int* and its value is required to be assigned to a which is of type *float*, then we can cast as follows:

```
a=(float)x;
```

12.1.5 Variable Names

An identifier in C is a sequence of letters and digits where the first character must be a letter. The underscore _ counts as a letter and upper and lower case letters are different. A conventional rule of thumb is to use lower case letters for variable names and upper case letters for symbolic constants. Keywords such as *if, else, int, float* etc. are reserved and cannot be used as variable names.

12.1.6 Declarations

All variables must be declared before use, although certain declarations can be made implicitly by context. A declaration specifies a type and is followed by a list of one or more variables of that type.

12.1.7 Arrays

In C, array subscripts always start at zero instead of 1. For example, the line

```
int ndigit [10];
```

declares *ndigit* to be an array of 10 integers with elements

```
ndigit[0], ndigit[1], ..., ndigit[9]
```

and

```
float array[10][10];
```

declares *array* to be an array of 100 floating point numbers. A subscript can be any integer expression, which includes integer variables like *i, j, k* and integer constants.

12.1.8 Operators

The arithmetic operators are

```
+, -, *, /
```

and are grouped from left to right. Integer division truncates any fractional part. The modulus operator is

```
%
```

and the expression

`x%y`

provides the remainder when x is divided by y and thus, is zero when y divides x exactly. The operator

`%`

cannot be applied to a float or double. The relational operators are

`>, >=, <, <=`

and the equality operators are

`== /* (equal to) */`

and

`!= /* (not equal to) */`

. The logical connectives AND and OR are

`&&`

and

`||`

respectively. C provides an incremental operator

`++`

which adds 1 to its operand The decremental operator

`--`

subtracts 1 from its operand.

Bitwise Logical Operators

C provides a number of operators for bit manipulation; these may not be applied to float or double types. They include:

```
&  /* bitwise AND              */
|  /* bitwise inclusive OR     */
^  /* bitwise exclusive OR     */
<< /* left shit               */
>> /* right shift              */
~  /* one's complement (unary) */
```

Assignment Operators

Most binary operators have a corresponding assignment operator *op*=, where *op* is one of the following:

`+, -, *, /, %, <<, &, >>, |, ^`

. For example, the expression

```
i=i+2
```

can be written in the compressed for

```
i+=2
```

. Finally note that C has no exponentiation operator (like $**$ in Fortran). For this type of operation the *pow* function is required.

12.1.9 Expressions

Numerical quantities are represented by arithmetic expressions , while Boolean quantities are represented by logical expressions. Examples of arithmetic expressions are

```
x*y, x%y, x+y
```

a Assignment expressions include

```
x>=y, x!=z
```

and assignment expressions are of the form

```
i=i+3
```

or

```
i+=3, x=x*(y+1)
```

or

```
x*=y+1
```

.

12.1.10 Control Statements

Unconditional Branching

Control may be transferred unconditionally by means of the statement

```
go to identifier;
```

where the identifier must be label located in the current function. For example, the code

```
go to start;

start:  .....
```

provides the facility for unconditionally transferring the process before the statement

```
go to start;
```

to the processes associated with the code after the statement

```
start:
```

Conditional Branching

Conditional branching is achieved by if-else, else-if and switch constructs. Nesting is possible. An example of a if-else construct is

```
if (n > 0)
   {
   if (a > 0)
      z=a;
   }
  else
      z=b;
```

An example of an else-if construct is

```
if (i > 0)
   {
   if (x > y)
       T=x+1;
   else if (x < y)
       s=x-1;
   }
   else
       T=y;
```

The Switch Construct

The switch statement is a special multi-way decision maker that tests whether an expression matches one of the numbers of constant value and branches accordingly. The basic form is as follows:

```
switch ( expression )
 {
 case label1: statement1;
              break;
 case label2: statement2;
              break;
 default:     statement3;
 }
```

There can be more than two labeled statements and the default case is optional. For example

```
switch ( letter )
  {
  case 'a' :
       printf("This line is printed if letter is a\n");
       break;
  case 'b' :
       printf("This line is printed if letter is b\n");
       break;
  default :
       printf("This line is printed if there is no match\n");
  }
```

12.1.11 Looping and Iteration

C provides three forms of loop constructs: *for, while* and *do*. All three can be nested.

The For Loop

The basic form of the for loop is

```
for (expr1; expr2; expr3)
     statement;
```

For example, to print the integers 1 through to 6 and their squares, we can construct the following code:

```
main()
{
  int num;
          printf(" n     n squared\n");
                 for ( num=1; num <= 6; num++)
          printf("%5d %5d\n", num, num*num);
}
```

The While Loop

The basic form is

```
while (expression)
statement;
```

For example, considering the problem above, using the while loop we have

```
main()
   {
   int num;
     printf(" n     n squared\n");
          num = 1;
          while ( num <= 6)
          {
                 printf("%5d %5d\n", num, num*num);
                 num++;
           }
    }
```

The Do Loop

The basic form of the do loop is

```
     do
       statement;
       while ( expression ):
```

Taking the previous problem again, we have

```
main()
{
  int num;
     printf(" n     n squared\n")
    num=1;
    do  {
        printf("%5d %5d\n", num, num*num);
        num++;
        }
    while(num <= 6);
 }
```

12.1.12 User Defined Functions

In C, a function is equivalent to a subroutine or function in Fortran for example. A function is a convenient way to compound a computational procedure. Each function has the same form:

```
   name(argument list)
   argument declarations
   {
      declarations
      statements
   }
```

For example consider a main program and function to compute m^n where m and n are the integers as given below:

```
main()
     {
     int i;
           for (i=1; i<=10; i++)
           printf("%d %d %d\n", i, power(2,i), power(-3,i));
     }

power(x,n) /*Raise x to the nth power for n > 0*/
int x,n;
{
  int i,p;
  p=1;
  for(i=1; i <= n; i++)
      p=p*x;
  return(p);
}
```

Here, the program computes the powers of 2 and -3 for n=1,2,...,10. In this case, the function returns the numerical value through its name. Another important use of a function is in passing and returning variables via the functions utility. Such functions are known as void functions. Taking, the previous example, we have

```
main()
     {
     int i,x,y;
           for (i=1; i <= 10; i++)
           {
           power(2,i,x);
           power(-3,i,y);

           printf("%d %d %d\n", i, x, y);
           }
     }

void power(x,n,y)
int x,y,n;
{
  int i,p;
  p=1;
  for(i=1; i <= n; i++) p=p*x;
  y=p;
}
```

Observed that in the void function above, the variables x, n and y are declared after specifying the function with its I/O. This is an example of K & R (Kernighan

and Ritchie) C. The ANSI (American National Standards Institute) form, which is arguably more informative, is to declare the variable type within the functions parenthesis, i.e.

```
void power(int x, int n, int y)
```

It is ultimately a matter of taste, but it is useful if the name of a void function is specified in upper case which helps to identify subprograms in the labyrinth of (lower case) code. This practice is adopted form here on.

12.1.13 Passing Variables using Pointers

Pointers provide the facility to 'point to' an area of the stack (random access memory) directly, and can be used to pass variables to and from a void function. For example, suppose a void function is designed to compute a variable from an array x which is of type *float*, then

```
void MY_FUNCTION(float x[ ], float *var)
{
/* Process */
.
.
.
*var=result
}
```

Here, the variable name var is a pointer to the result which is passed through the function as indicated in the parenthesis. To address this variable the & is used. Thus, to use this function in a main program, we use

```
MY_FUNCTION(x, &var)
```

For example, C has an intrinsic function *scanf* which resides in the standard I/O library (*stdio.h*) and allows data to be input from the keyboard. Suppose it is required to input a value from the keyboard which is of decimal integer type and assign it to the variable *var*, then we may use

```
scanf("%d",&var);
```

Two dimensional arrays can be addressed using pointers in a similar way, e.g.

```
void MATRIX(float *a[ ], float *b[ ])
```

passes the two-dimensional arrays a and b to the void function *matrix*.

12.1.14 Internal Functions and Libraries

In addition to the basic facilities discussed so far, C compilers offers a range of intrinsic functions. This facility has expanded rapidly over the years to provide functions that are compatible with other languages and beyond. The functions are assigned to different libraries. The most common of these libraries are *stdio* (standard input/output) which provides functions for inputting and outputting standard data from standard I/O facilities such as the keyboard for example and the math library *math* which contain functions for computing mathematical operations such as a cosine and sine for example. These libraries of functions are .h functions and need to be included at the very beginning of a program or subprogram using the hash include statement, e.g.

```
#include<stdio.h> /* Standard input/output library */
#include<io.h>    /* Input/output library          */
#include<math.h>  /* Maths library                 */
```

The hash can also be used to define variables that are global to a main or subprogram. For example

```
#define N 256
```

sets the value of N to 256. The decimal integer N can then be used throughout the program with a value of 256.

12.1.15 Prototyping

Prototyping involves stating the name and type of any functions that another function is to use for data processing. The compiler needs to be informed *a priori* of the functions that a module is going to use. This is done be copying the function names together with their parameter lists prior to defining the the module. For example, suppose a void function called *PROCESS* has been designed that is dependent on two other void functions *F1* and *F2* and that all three operate on a serial stream of floating point data of size *n*. *PROCESS* is then prototyped with respect to *F1* and *F2* as follows:

```
#define<math.h>
#define<io.h>
...

void F1(float x[ ], int n);
void F2(float x[ ], int n);

void PROCESS(float x[ ], int n)
{

...
```

```
F1(float x[ ], int n);

...

F2(float x[ ], int n);

...

}
```

As the number of functions increases, it is good practice to list them in a header file which can then be specified at the beginning of a module. In the example given above, we have

```
#define<math.h>
#define<io.h>
...

#include "modules.h"

void PROCESS(float x[ ], int n)
{

...

F1(float x[ ], int n);

...

F2(float x[ ], int n);

...

}
```

where the file modules.h is composed of the following:

```
void F1(float x[ ], int n);
void F2(float x[ ], int n);

...
```

12.1.16 Advantages and Disadvantages of C

Advantages

- C is a well structured language; in general, it forces inexperienced programmers to produce relatively well structured code although it is still possible to produce 'spaghetti programs' with uncontrolled use of the goto statement for example.
- C allows access to many of the features associated with an operating system during run time (particularly Unix or Linux).
- There is good data typing facilities.
- The run time structure of a C program is dynamic.

Disadvantages

- The semantic structure of the C language is not ideal for translation of complex algebraic equations and algorithms used in engineering when compared to a language such as Fortran.
- Execution of load modules produced by many C compilers are not as efficient as some languages particularly when it comes to floating point arithmetic. On the other hand, C can be used effectively to drive DSP chips which are essentially floating point accelerators for developing real time DSP systems.
- C does not provide (especially with some of the older compilers) as many useful intrinsic functions as some other languages for number and character manipulation.

For all the disadvantages C may have, its widespread use, facilities and rapid development over-ride any minor irritations that the language might legitimately be criticized for. For modular and structured programming, the C programming language is ideal.

12.2 Modular and Structured Programming in C

One of the principle advantages of the original C programming language was its run time memory management facilities. A programming language that does not provide this feature actively hinders modular programming especially on computers with low memory capacity and operating systems which can only address a low RAM (i.e. early PC/DOS systems). In fact, it may be argued, that memory management more than any other feature, led to the instant popularity of C in the early 1980s with regard to the growing use of microcomputing for engineering in which, especially in

the early years when the standard operating system (i.e. MS and DR DOS) had a 640K memory threshold.

To emphasize this important point, it is worth taking a look at the principles of memory management with regard to modular programming by taking a specific example which is common to many cases and is particularly important in DSP where efficient array processing is essential.

12.2.1 Array Processing

In C, an array is declared as:

```
type array[wordlength];
```

For example, the code

```
float a[10];
int k[20];
```

declares and array of type *float* with 10 words of memory and an array of type *int* with 20 words of memory. It is important to note that by default, the compiler assumes that an array that has been given n words of memory is of the form a[0], [1], a[2], ..., a[n-1] and not a[1], a[2], ..., a[n] so that when we process arrays using a for loop for example, the counter is from 0 to n-1 and not from 1 to n, e.g. to initialize an array of size n, we perform the following:

```
float array[n];
for (i=0; i< n; i++) array[i]=0.0;
```

If it is desired to loop from 1 to n, then n+1 words of memory must be assigned, e.g.

```
float array[n+1];
for (i=1; i <= n; i++) array[i]=0.0;
```

In this case, the element

```
array[0]
```

is not used. This feature can lead to some difficulties in designing C code that reflects the conventional use of indices for defining vectors and matrices. Typically, we define a vector as $\mathbf{x} \in R^n$ which is taken to represent the set of elements $x_1, x_2, ..., x_n$ and not $x_0, x_2, ..., x_{n-1}$ as in part II of this book. Providing consistency is preserved, the default array assignment of C can and should be used. Array processing which is based on processing the array elements from 1 to n instead of from 0 to n-1 requires that arrays of size n+1 are specified *a priori*. In this case, a useful tip is to initialize the first element of the array, i.e. set x[0]=0.

12.2.2 Dynamic Memory Allocation

Consider the following void function which undertakes a general 'process':

```
       void PROCESS(float x[ ], int n)
       {

/* Declare internal workspace arrays */

       float y[1000], z[1000];

/* Do processing */

        ...

        }
```

Suppose this subprogram is just one of a number of modules contained in a library; it inputs an array x of type *float* and of size n. It performs some process on x and outputs the result by over-writing x. The integer variable n is a control parameter which remains unaltered on output. The arrays y and z are internal workspace each with 1000 words of memory. This internal workspace is typically required to carry out the process on x. It is usual to set an upper limit on the work space so that the subprogram can be used to process arbitrary size arrays up to and including the number of words allocated to the workspace that is internal to the subprogram. Hence, in this subprogram n$\leq$1000. There are two problems with this:

(i) If we wanted to process an array x with more than 1000 elements, we would have to edit the subprogram to change the memory allocated to the internal workspace accordingly.

(ii) If the input to this subprogram contains less than 1000 data points (n=500 for example), then the memory allocated to the internal workspace is in excess of that required, wasting RAM which, on a small PC, may be in short supply.

There is an obvious solution to this problem which is to input the arrays y and z together with x, thus:

```
       void PROCESS(float x[ ], float y[ ], float z[ ], int n)
       {

/* Do processing. */

        ...

        }
```

This solution requires the user of this subprogram to know details about its construction; in particular, the work space required. The user must pass arrays through the

subprogram call:

```
int n;
 n=1000;
  float x[n],y[n],z[n];
  ...

  PROCESS(x,y,z,n);
  ...
```

So what's the problem? At first sight there does not appear to be a problem as such, until the discussion above is seen in the context of a library containing many modules. When a library is completed, the user should not want or need to know how a specific module actually works, only that a call to a certain subprogram will perform a specific function. This of course, includes the internal workspace required to carry out that function on arrays of arbitrary size. Although the above solution will work, it involves the user having to know the internal work space required for a subprogram *a priori*. When the user is designing a program consisting of calls to many different subprograms stored in a given library, this can become very tedious and in a sense, defeats the point of modular programming. On the other hand, if the former method is used (where the internal workspace is declared with an upper limit on the word space required), then the total memory needed to implement an entire library of such modules can become extreme, in the knowledge that this memory may not necessarily be required!

With respect to the arguments above, the whole problem can be overcome if we could write

```
void PROCESS(float x[ ], int n)
...

/* Declare internal workspace. */

float y[n], z[n];

...

}
```

In this case, the size of n would completely control the memory allocation required to run this subprogram. Indeed, the value of n could possibly be input during the run time of a program which called this subprogram; an example of dynamic memory allocation. Many early compilers did not have this facility. For example, until the release of Fortran 90, no FORTRAN compiler provided dynamic memory allocation.

The discussion above has been presented to illustrate the meaning of a 'static' programming language with 'poor' memory management. It also demonstrates one

of the principal reasons why modular programming ideally requires dynamic memory management if RAM is to be utilized efficiently.

In C, the solution above can be implemented but not using the code as written. Instead, we need to make use of the memory allocation function *malloc* or *calloc*:.

```
      void PROCESS(float x[ ], int n)
      ...

/* Declare internal workspace. */

      y = (float *) calloc( n, sizeof( float ) );
      z = (float *) calloc( n, sizeof( float ) );
      ...

/* Do processing. */
      ...

/* Free memory allocated to internal arrays. */

      free(y);
      free(z);
      }
```

In this example, *calloc* assigns n words of memory (using the function *sizeof*) to arrays y and z which are of type *float*. After processing the input array with the use of these internal arrays, the function *free*, removes the memory that has been allocated. The function *malloc* operates in a similar way, the difference being, that after the arrays have been created, they are not initialized so that the internal arrays could take on spurious values. Thus, the use of *malloc* should always be coupled with a for loop to set the internal array(s) to zero, i.e.

```
y = (float *) malloc( n*sizeof( float ) );
    for(i=0; i<n; i++)
    y[i]=0.0;
```

The function *calloc* initializes the arrays automatically and is therefore recommended. Note, that if arrays are processed using for loops for example from 1 to n instead of from 0 to n-1, then *calloc* must be used to assign n+1 words of memory.

12.3 Modularity

In modular programming, the principal issues that govern the system 'lifecycle' can be quantified as follows:

Requirement analysis

$\Downarrow$

Design

⇓

Module specification

⇓

Module design

⇓

Module Coding

⇓

Module Unit Testing

⇓

System Test

⇓

Maintenance

There are three important questions concerned with modular programming in general:

- How big should a module be?
- How complex is the module?
- How can we minimize interactions between modules?

12.4 Module Size

Clearly, the smaller the module, the easier it should be to comprehend. On the other hand, if each module is too small, the system can become overwhelmed by their proliferation. Hence, low complexity in terms of the number of statements per module increases the complexity of the number of levels. A common view point is that a module should occupy no more than a page of text (40 - 50 lines of code). An extreme view point is that a module should normally take up about seven lines or less. This viewpoint is based on psychological research which indicates that the human brain is only capable of comprehending seven things (as a complete set) at once. Other factors to consider in judging the length of a module are:

(i) A large number of smaller modules gives rise to slower programs because of the increased overhead of subprogram linkage.

(ii) A library composed of many small modules leads to confusing 'bitty' structures.

(iii) The logic of a module should be self-contained (and self-evident); at no time should we need to comprehend two of more modules simultaneously.

(iv) Editing a module should be self-consistent and not necessarily lead to the modification of other modules.

Complexity of a Module

In the days when CPU and RAM acted as stringent constraints, it was usual for programmers to work hard to make their programs as efficient as possible. Nowadays, the emphasis is to reduce the development time of programs and ease the burden of maintenance, i.e. to write programs that are clear and simple, and therefore easy to check, understand and modify. In considering the complexity of a module, there can be a conflict between whether it is more important for a program to be easy to understand or whether it is more important for it to run quickly, which may involve the introduction of complex coding features. The speed of a program is not so critically related to features such as subprogram linkage but more to its structure, form and iteration schemes (for loops). As a rule of thumb, programs spend most of the time ($\sim$50%) executing a small fraction ($\sim$10%) of the code and it is the optimization of these small parts that leads to minimum run times.

Arguments for Simplicity

Finding a simple solution may not necessarily be easy. A simple program is more likely to work quickly and then go on working after it is put into practice. Further, locating a bug in a simple program or modifying it to meet a changed specification is made easier if the program design is easy to follow, rather than riddled with clever but incomprehensible tricks.

Complexity

Complexity is not just the number of lines associated with a module but involves:

- **Comprehensibility**
  For the purpose of design, checking, testing and maintenance, it should be possible to understand individual modules independently of others.

- **Changeability**
  If various design decisions are changed, such as a file structure, changes should be confined to as few modules as possible (preferably one).

- **Independent development**
  The interfaces between modules should be as simple as possible. Interfaces should therefore, be by means of calls on subprograms rather than by means of access to shared data and file structures.

- **Number of decisions**
  The number of if..then..else type statements which lead to a transfer of control.

The last factor in the complexity of a module is the basis for the so cal complexity measure (McCabe T J, 'A complexity measure', *IEEE Transactions of Software Engineering*, Vol. SE-2, No. 4, 1976). This complexity measure is based on the assertion that module complexity does not depend on the number of statements in the module but on the decision structure of the module. To calculate the cyclometric complexity of a program, one counts the number of predicates and add one. For example, the line

```
if((i > j) && (n > m)) then
```

has a cyclometric complexity of 2. The complexity of a module may also refer to its computational complexity which describes whether problems and algorithms are feasible in terms of their usage of computer time and space.

Coupling of Modules

Coupling describes the way in which individual modules affect each other. The most important forms of coupling are:

(i) Procedure calls with a small number of data parameters.

(ii) Passing a serial stream of data from one module to another.

These are the weakest and best forms of module coupling. Another form of coupling which is sometimes useful involves a procedure call with a switch as a parameter. This is where a module is passed an indicator telling the procedure which action to take from amongst a number of available actions. This indicator is sometimes called a switch. For example, consider the following procedure for a general purpose I/O (read/write) procedure.

```
IO(switch, array, length);
```

The parameter 'switch' could have values 0 and 1 or 'r' and 'w' that specify whether the operation is a read or write respectively. This single module could replace two individual modules such as:

```
READ(array, length);
```

and

```
WRITE(array, length);
```

The function IO is a single multi-purpose module. The functions READ and WRITE eliminate a parameter from the interaction and at the same time create well-defined modules, each with a specific function.

Cohesion

Cohesion describes the nature of the interactions within a software module. The programmer should aim at functional cohesion, in which every operation in the module contributes toward the performance of a single well-defined task giving 'high-cohesion'. Sequential cohesion can be useful where the operations in a module collaborate in modifying a piece of data. Typically, such a module accepts data from one module, modifies them, and passes them on to another module. Coupling and cohesion provide a qualitative analysis of modularity. A general rule of thumb for modular programming is:

Strong coupling and weak cohesion are bad.

Weak coupling and strong cohesion are good.

Shared Modules

In many cases, the functionality of a specific block of code in a particular module may be better assigned to another subprogram for reasons of clarity, structure and form. If the block in question is specific to the functionality of a module alone and has no further use 'outside' this module, then it is generally better to write the subprogram in a form that is internal to the module itself. In some cases, many of the modules which constitute a library may require a common process to be performed in order to execute their functionality. In this case, a module which performs this process should be designed beforehand so that it can be shared by other modules. For example, consider the case when a common prompt is required by two function *F1* and *F2*.

```
      void F1(float x[ ], int n)
      {
      ...

/* Prompt user for name of data file. */

      PROMPT;
      ...

      }


      void F2(float y[ ], int n)
      {

      ...

/* Prompt user for name of data file. */

      PROMPT;
      ...
```

```
}


POMPT()
{
/* FUNCTION: Prompts user for name of data file. */

printf("Input name of data file\n");
}
```

In this example, at a given point in the code, functions *F1* and *F2* require the user to input the name of a file. The user must be prompted for this name which in this example is done by calling the function *PROMPT*. Both subprograms share this module.

Sharing modules is an important consideration in building a library. It is common to undertake this task using a bottom-up design strategy. In this approach, we consider the computer/compiler to provide features that can be considered to be basic 'primitives' to our software solution. We then add a 'layer' of software that enhances the basic computer/compiler features to make them look more convenient and appropriate for the system problem we wish to solve. This creates a more powerful abstract machine, built 'on top' of the basic environment provided. The process is repeated by designing and building other, higher-level layers until we have constructed a system that matches the problem well. Modules that are used in more than one place should not arise from a top-down design process, but from a bottom-up approach.

12.5 Structured Programming

The basic idea behind structured programming is that any program, however complex, should be written in such a way that it can be read by a human who starts at the beginning and follows it through from top to bottom (and left to right). One of the principal views of structured programming is that programs should be built from three components:

- **sequences** which are normally written in the order in which the statements are to be executed;
- **selections**, e.g. if...then...else;
- **repetitions**, e.g. do ... enddo.

The consequences of the application of this principal are that:

(i) Each component has only one entry and exit.

(ii) None of the constructs consist of more than three 'features'.

(iii) The *goto* statement is obsolete.

Programs whose design is based on a rigorous application of this principal have a logical and well structured form. A fundamental feature of structured program concerns arguments related ton the *goto* statement.

Arguments Against the Goto Statement

- They are unnecessary and any program written using a goto statement can be transformed into an equivalent program that uses only the structured constructs.
- Experimental evidence suggests that a structured program (without goto statements) can be understood by another programmer more rapidly than a non-structured program.
- Goto statements do not convey the meaning of a block of code as clearly as a structured construct; the goto statement has a lack of 'expressive power'.
- Goto statements lead to difficulties in finding out where you came from in a program.
- Goto statements require that a program must be read 'backwards' or by 'skipping forwards'.
- The goto statement leads to difficulty in proving the correctness of a program.

Arguments for the Goto Statement

- Goto statements have use in exceptional circumstances.
- They are sometimes necessary to make a program perform well.
- It is sometime 'natural' to use goto statements.

The main goal of structured programming is to yield programs with optimum clarity. In this sense, the cons of the goto statement outweigh the pros. By way of an example, consider the following block of C code which is designed to find whether the integer number 2 is present in a serial stream of 10 integers (held in the integer array n).

```
        i=1;
flag1:  continue;
         if(n[i]==2)  goto flag2;
         if(i==10)    goto flag3;
            i=i+1;
```

```
                    goto flag1;

 flag2: printf("found number 2\n");
        goto flag4;
 flag3: printf("not found number 2\n");
        goto flag4;
 flag4: end;
```

An equivalent structured program is:

```
       k=0;
       for(i=1; i<=10; i++)
                         {
                           if(n[i] == 2)k=1;
                          }
        if(k == 1) printf("found number 2\n");
        if(k == 0) printf("not found number 2\n");
```

As an example of the logical use of the goto statement, consider the following block of C code consisting of three modules written as void functions which process a serial stream of data held in array x and whose performance (determined by a function $EVALUATOR$ which analyses x in some way) depends on the value of a user specified parameter *param*.

```
start:  printf("Input value of param\n");
        ...

        PROCESS_1(x,n,param);
        PROCESS_2(x,n,param);
        PROCESS_3(x,n,param);
        ...

        EVALUATOR(x,output);
        if(output=='bad')
        {
        printf("Process data again (y/n)?");
        scanf("%s", &answer);
        if(answer=='y')goto start;
        }
```

Reading and Writing Data to and from a File

By way of an example, we consider two functions for writing and reading floating point data to and from files that are named (with no more than 16 characters long)

and accessed in run time. This is a standard necessary requirement. Each program that follows has been designed to be self explanatory and uses standard C functions for specifying, opening and closing a data file, namely *gets*, *fopen* and *fclose* respectively. The functions for writing and reading the data to and from a file are *fprintf* and *fscanf* respectively.

```
#include <io.h>
#include <stdio.h>

void WRITE( float s[], int n )
{
  char filename[16];
  FILE *fp;
  int i;

/* FUNCTION: Writes data of type float to a named file.   */

/* Prompt user to input name of data file.                */

     puts("Input name of file to write data>");
     gets( filename );

/* Open file.                                             */

     if ((fp = fopen(filew, "w")) == NULL)
     {
     printf("error\n");
     return;
     }

/* Send message to use 'writing data ...'                 */

printf("WRITING DATA ...");


/* Write array s to data file.                            */

     for (i=0; i<n; i++)
     fprintf(fp,"%f\n",s[i]);

/* Close the file.                                        */

     fclose( fp );
}


#include <io.h>
```

```
#include <stdio.h>

void READ( float s[], int n )
{
  char filename[16];
  int i;
  FILE *fp;

/* FUNCTION: Read data of type float from a named file.    */

/* Prompt user to input name of data file.                 */

  puts("Input name of file\n");
  gets( filename );


/* Open file.                                              */

    if ((fp = fopen(filename,"r")) == NULL)
    {
    printf("error in reading file\n");
    exit(1);
    }

/* Send message to use 'READING DATA ...'                  */

printf("READING DATA ...");

/* Read in array s from data file.                         */

   for (i=0; i<n; i++)
   fscanf(fp,"%f",&s[i]);

/* Close the file.                                         */

   fclose( fp );

}
```

12.6 Modular Programming using Borland Turbo C++

The principle components of a typical computer can be summarized as follows:

Input/Output (Terminal, Hard Disc etc.)	$\Longleftrightarrow$	Central Processing Unit (CPU)	$\Longleftrightarrow$	Internal Memory (RAM)

The central processing unit performs basic operations. Internal Memory (Random Access Memory or RAM) holds the program specifying operations to be performed and information or data upon which these operations are to act. Input/Output devices are those through which the algorithm and the data are fed into the memory and through which the computer communicates the results of the activities.

12.6.1 Speed and Memory

Speed

The CPU of a typical micro computer can perform the order of 10^9 operations a second. For example a 1GHz machine will perform 10^9 (typical Pentium III Intel CPU) Boolean operations each second which are carried out in series on a so called 'von Neumann' machine. The speed of operations can be dramatically increased by using parallel processors. However, producing parallel architectures is not trivial and the speed at which serial processors improve in performance has often out-paced that of parallel processing and its applications to engineering problems. Random access memory (RAM) is one of the primary characteristics of a computer. Memory capacity and access speed vary widely according to the storage medium used. The larger the RAM, the larger the number of bits of data that can be stored without having to read/write to an I/O device, e.g. hard disc.

Virtual Memory System (VMS)

The basic idea of a VMS is to increase the memory capacity of a computer by utilizing disc space automatically, i.e.

$$\text{RAM} + \text{Disc space} = \text{Virtual Memory}$$

Every time a segment of data is transferred from RAM to disc or disc to RAM, a 'page fault' is generated which takes a finite period of time. This approach to increasing processor memory was first pioneered by Digital Equipment Corporation but it is now a feature of nearly all computers. One aspect of programming is to minimize page faults.

Minimising Page Faults

There are a number of ways of minimizing page faults but one of the most important features associated with the type of programming required for DSP is ensuring that nested loops are constructed in the correct order. Iteration forms an important part of all aspects of programming particularly in engineering. Hence, optimising the speed at which iterative processes are executed is very important. A common mistake is nesting loops in an order that is incompatible with the compiler being used. With 2D

array processing for example, it is necessary to process both the rows and columns. In a serial machine, the data is not actually stored as a 2D array but as vector. Depending on the nature of the compiler, this vector is made up either of row segments or column segments. The way in which the compiler assembles the vector determines the order of the process. With C programming, the rule of thumb is to process the rows first and then the columns, e.g.

```
for(i=0; i < n; i++)
   {
    for(j=0; j < n; j++)
        a[i][j]=0.0;
   }
```

12.6.2 Compiling and Linking

A typical session will involve the following:

- Write a module (typically a void function).
- Compile the module.
- Design a test unit and compile it.
- Link the test unit with the module.
- Run the program and inspect the output for a given input.

All C programs have the extension .c, e.g. main.c, test.c etc. Assuming that the user is using a command line approach (rather than a visual interface), the command for compiling a function called *process.c* say with Borland Turbo C++ is *tcc -c process.c* which outputs a *process.obj* file providing there are no fatal errors. It is at this point that syntactical errors are to be corrected. For those who are new to C programming, a standard mistake is to miss out a semi-colon ; at the end of a statement for example. Typically, the module is a processor and although it may compile perfectly, no diagnosis of its functionality will have been undertaken. Suppose that we have now designed a test function or unit called *tprocess.c* say, which inputs data that is processed by the function *process.c* and outputs the result (assumed to be alpha-numerical). After compiling the test unit, we can then link the test unit with the module using the command *tcc tprocess.obj process.obj* or alternatively using the source code files *tcc tprocess.c process.c*. This produces an executable file *tprocess.exe* which can be run by just typing the name (with CR).

12.6.3 Developing an Object Library

The typical components of a program are:

- Main Program
- User Interface
- Data Input
- Data Processing
- Data Output

In the programming practice adopted here, all programs should be designed around sequences of sub-processes or subprograms - preferably void function modules; hence the term 'modular programming'. Subroutines can of course be part of a main program, e.g.

```
main()

/* program main.c */

{
 float x[512];
 ...
 PROCESS_1(x);
 PROCESS_2(x);
 ...
 }

 void PROCESS_1(float x[])
 {
 ...
 }

 void PROCESS_2(float x[])
 {
 ...
 }
```

In this case, the modules form part of the same source code file. In such a case, using Borland Turbo C++, application of the command *tcc main.c* will provide an executable file main.exe. However, in modular programming, it is far more appropriate to design the functions as separate files, in which the name of the of the function is reflected by the name of the file containing the source code (in upper case or otherwise). Each subprogram or module must then be compiled separately. For example, suppose program *test.c* introduces processes undertaken by modules *proc1.c* and *proc2.c* which are external to *test.c*. In this case, we compile and link to produce an executable file *test.exe* using the command *tcc test.c proc1.c proc2.c*. Here, all the files are taken to exist in the same directory.

12.6.4 Object Libraries

The basic idea of an object library is to establish a special directory in which all modules required by a main program are stored and compiled. Instead of linking to subprograms separately, we link to an 'object library' in which compiled subroutines (i.e. object files) are known to exist. To accomplish this using Borland Turbo C++, we use the command *tlib olname +mod1 +mod2 +mod3 ...* where *olname* is the preferred name of the object library (which is produced with extension .lib) and *mod1, mod2,...* are the objects, i.e. *mod1.obj, mod2.obj,....* Linking a main program that uses all, or a subset of these modules is then simple using the command *tcc main.c olname.lib*. To generate a list of the objects from which the library is composed, the command *tlib olname.lib, olname.lst* can be used, the file *olname.lst* can then be inspected as required. Removal of an object *mod4* say from the library is undertaken using the command *tlib olname -mod4*. Further variations are of course available in the appropriate compiler documentation.

Ideally, the main program should essentially be dominated by a sequence of functions. It is a good practice to make the name of the directory in which the modules are placed the same as the name of object library. In addition to the object library(s) developed by the programmer, it may be necessary to link with specialist object libraries provided by the compiler. For example, Borland Turbo C++ provides a graphics library which must be linked to when graphics functions are used, i.e. *graphics.lib*.

12.6.5 Practical Applications

The practice of modular programming applies to software development for research programs, software development and the generation of commercial software packages. In each case, we study how to break the problem down into a set of essential processes, e.g. reading data, writing data, displaying data, graphical output, computational operations etc. Each process becomes a module which is tested and made 'bug proof'. When satisfied with the 'condition' and 'performance' of a function or module, it is added to the appropriate library. When the library is finished, main programs can be written which concentrate on the user interface. The principles of modular programming are essential for research. Once a library is established, ideas can be tested by just calling different processing routines. If a new process is required, then it can be written as a module and added to the appropriate library. It is upon this basis that the problems are presented in Part IV of this book.

A number of low to high level processes can be designed in the form of a function. Such subprograms can be collected together in a file to serve as an object library. User created libraries of this type are very useful and can greatly simplify programming. Instead of labouring to add a new feature to some old program, one simply calls on one of the appropriate subprograms in the library, and the desired facility appears. The principal advantage of subprogram libraries lies in the fact that they cultivate highly modular programming which is the basis of nearly all modern programming techniques. This involves nothing more than coupling together several well-developed, carefully tested and debugged modules. When a new feature is required, it can be developed and then added to the library. The mechanics of creating and maintaining a library varies from system to system. However, in general, subprograms can be

compiled in isolation and subsequently stored in a library. Once such a library exists, you no longer need to append the source versions of the subprograms to your main program. Instead, you simply allow the library to be associated with your compiled main programs at run time.

12.7 On Style and Presentation

Style and presentation are particularly important features of good programming. Some useful tips are:

(i) Provide a header to all modules giving information on all aspects of the program or subprogram including the author, date, function of program, list of variables, I/O arrays and parameters, a modification history and so on.

(ii) Comment each functional block of a program with a description of what the process is, aiming at a comment to code ratio of (at least) 2:1.

(iii) Use comment lines to breakup blocks of source code.

(iv) Indent the source code so that important features can easily be read.

The following is an example of a header for a subprogram:

```
       void EXAMPLE(float x[ ], float y[ ], int n)

/************************************************

 FUNCTION: Provides an example of the type of
           header that should be used when
           writing a subprogram function.


 AUTHOR:                       DATE:

 MODIFICATION HISTORY:

      Date      Author      Reason

************************************************

 PARAMETERS

 Input: x - input array of type float
        n - size of array of type int

 Output: y - output array of type float

 GLOBAL VARIABLES:
```

```
  LOCAL VARIABLES:

  INTERNAL FUNCTIONS:

  EXTERNAL FUNCTIONS:

  OBJECT LIBRARIES REQUIRED:

  ************************************************/
```

Discussion

The use of extended variable and parameter names can lead to a significant increase in the perceived complexity of the code. Moreover, if long variable and parameter names are used, more errors can be made in typing them out when designing the code and/or correcting it. In general, it is a good idea to let the code reflect as much as possible the notation used in the mathematical development of the algorithm that is to be coded. For example, suppose that we are processing a two dimensional array which is based on some matrix algebra related to a matrix $A = (a_{ij})$ as used in the mathematical notation. In such a case, the code should be based on the application of

```
a[i][j]
```

rather than something like

```
array_value [column_element] [row_element ]
```

which is unnecessary and cumbersome but unfortunately typical of the coding undertaken by computer scientists who have not undertaken a rigorous course in linear algebra for example and have an aversion to certain subjects that underpin computer science (mathematics for example). Using variables names that are long, even with the best intentions, tends to yield code that is dense and sometimes off-putting. Using notation in developing code that is as close as possible to that used in the development of a mathematical algorithm yields a certain expressive power. Some programmers like to use an underscore at the end of a variable, e.g.

```
variable_name_
```

This idea has value when undertaking a search in an editor for example, by searching for all elements of the code that end with an underscore. While of value in editing large modules, if the size of a module is kept to minimum length (as is should be), this feature becomes redundant.

Another important features of good coding practice is to space the code out. The principle is similar to that of designing a document. If the document looks 'crowded'

and dense, then it is more difficult to scan visually; in addition, it is visually off-putting which may lead to fatigue and irritation of the reader (but not necessarily in that order). Clearly, with a document of this type (i.e. this book), limits have to be placed on the number of pages for reasons of expense, 'page space' being at a premium. However, code that is stored in a file is not space limited as with hardcopies. Many excellent CASE tools exist to aid the programmer in the task of producing quality coded products but at the end of the day, it is the programmers hand that does the writing - the moving figure writes and have writ moves on. Ultimately, there is a large amount of artistry associated with programming and software engineering in general, but it is always worth remembering that like other engineering disciplines and products, there is certain truth to the comment that *...if it looks good, it will work good*. The art of programming is based on the principle of keeping it simple, on the understanding that it is simple and is only made complicated by human beings, some of whom like to hide behind complexity for reasons of personal security. Further, some software engineers, particularly those in management positions, can be likened to geese in that they like to hiss allot. Some geese hiss even if there is nothing to hiss at; it make them feel important.

Other Useful Tips

(i) Remember to declare correctly the size of any internal workspace required (dynamically or otherwise).

(ii) Always initialize internal parameters and arrays.

(iii) Nest loops in the correct order.

(iv) Free any memory allocated during run time, i.e. clear the stack.

12.8 Summary of Important Results

Modules in C
A function specified by

```
FUNCTION_NAME(input)
```

in which the output (of type int, float or char) is specified by the function name or a void function given by

```
void FUNCTION_NAME(input,output)
```

where both the input an output data is specified by variables and/or arrays defined in the parenthesis.

Dynamic Memory Allocation
The allocation of memory during run time, typically to arrays that are internal to a module.

Module Complexity
The comprehensibility, changeability, independent development and transfer of control of a module.

Cyclometric Complexity Measure of a Module
The number of predicate or if..then..else type statements in a module plus 1.

Coupling of Modules
Describes the way in which individual modules affect each other; the most important forms of coupling being: (i) procedure calls with a small number of data parameters; (ii) passing a serial stream of data from one module to another; (iii) application of a switch to identify a specific processing path.

Cohesion of Modules
Describes the nature of the interactions within a module in which every operation in the module contributes toward the performance of a single well-defined task (high-cohesion) or where the operations in a module collaborate in modifying a piece of data (sequential cohesion).

Coupling and Cohesion of Modules
Strong coupling and weak cohesion are bad; weak coupling and strong cohesion are good.

Test Unit
A program designed specifically to test the functionality of a module in terms of the expected and actual processing of test data.

12.9 Further Reading

- Kernighan B W and Ritchie D M, *The C Programming Language*, Prentice-Hall, 1978.
- Farmer M, *The Intensive C Course*, Chartwell-Bratt, 1988.
- Johnsonbaugh R and Kalin M, *Applications Programming in ANSI C*, Macmillan, 1990.
- Jaeschke R, *The Dictionary of Standard C*, McGraw-Hill, 1991.
- Perry G, *C++ by Example*, Que Programming Series, 1992.
- Deiss W, it Turbo C++ Step by Step, Abacus, 1992.

- Hanley J R, Koffman E B and Friedman F L, *Problem Solving and Program Design in C*, Addison-Wesley, 1993.
- Weiss M A, *Data Structures and Algorithms in C++*, Benjamin/Cummings, 1994.
- Lippman S B, *C++ Primer*, Addison-Wesley, 1995.
- Bramer B and Bramer S, *C++ for Engineers*, Arnold, 1995.
- Bramer B and Bramer S, *C for Engineers*, Arnold, 1997.

12.10 Problems

12.1 Write a C void function which inputs an array of positive floating point numbers and outputs the average value, the maximum value and the minimum value of the array. Test this function by writing a test unit to input ten numbers from the keyboard and output the results to the VDU.

12.2 A method of writing a portable random number generator is to use the iteration

$$x_{n+1} = ax_n \text{mod}(P)$$

where x_0 is given (the seed). For given values of r and P, the sequence of numbers generated via this iteration are uniformly distributed (i.e. satisfies a rectangular distribution) so that the chance of any number occurring in a given range is the same as that of all the others. Write a void function to compute a sequence of n random numbers for a given seed with

$$a = 7^5 = 16807 \quad \text{and} \quad P = 2^{31} - 1 = 2147483647.$$

Use integer arithmetic to evaluate the sequence but scale the results so that the output consists of floating point numbers between 0 and 1. Test the output of the function for different seeds using an appropriate test unit.

12.3 Write a function to compute the histogram of an array of n positive integer numbers for a range of values m say. Test your program by inputting a set of numbers from the keyboard and studying the output. Use your algorithm to study the random numbers generated by the random number generator discussed in the previous question. Apply a range to the output to specify the number of bins the histogram will have. Do the random numbers approximately satisfy a uniform distribution?

12.4 The discrete Fourier transform (DFT) of series of N complex numbers $x_0, x_1, ..., x_{N-1}$ is given by

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k \exp(-i2\pi kn/N)$$

for $n = 0, 1, ..., N-1$ where y_n is complex. Write a function to return the real and imaginary parts of y_n given the real and imaginary parts of x_k of length N.

The inverse transform

$$x_k = \sum_{n=0}^{N-1} y_n \exp(i2\pi kn/N)$$

for $k = 0, 1, ..., N-1$ should convert the complex array y_n of length N containing all the results $y_0, y_1, ..., y_{N-1}$ back into the numbers x_k (with rounding and truncation errors). Write a function to do this given the complex array y_n of length N. Test these functions by taking the DFT of the array (1,1,1,1,0,0,0,0) - the real part of x_n - and then reconstruct it using the inverse transform.

12.5 An example of a technique for smoothing data to reduce the effect of random error is to apply a moving average filter of the type

$$g_i = \frac{f_{i-1} + f_i + f_{i+1}}{3}$$

Write a void function to implement this algorithm which inputs data of size n and outputs the results of applying this filter. Assume that the initial and end conditions are $f_0 = 0$ and $f_{n+1} = 0$ respectively. Design an appropriate test unit to validate the process.

12.6 An iterative technique for solving a set of linear equations $A\mathbf{x} = \mathbf{b}$ known as Jacobi's method leads to the result

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^k \right)$$

where $x_i^0 = b_i/a_{ii}$ and $a_{ii} > 0 \forall i$. Write a function to implement this method of solution which inputs the matrix A, the data $\mathbf{b}$ and its size n and the maximum number of iterations required and outputs the solution vector $\mathbf{x}$. The method is convergent in cases where the matrix is diagonally dominant, i.e.

$$a_{ii} > \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}.$$

Hence, test the function using a 3×3 example which is diagonally dominant with a known solution and study the rate of convergence of the solution.

12.7 Another iterative technique known as the Gauss-Seidel method updates the elements of the solution vector as the iteration proceeds using the result

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right)$$

where $x_i^0 = b_i/a_{ii}$ and $a_{ii} > 0 \forall i$. Write a function to implement this method of solution using the same I/O and test procedure as described in the previous question.

12.8 The Newton-Raphson method is a technique for finding an approximate root of a function $f(x)$, i.e. a value for x that makes $f(x)$ approximately zero. To use this method, one starts with an estimate x_0 of the root. Successive approximations are then generated via the formula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

where f' is the derivative of f. The 'stopping criterion' can be when the difference between the last two approximations is a small fraction of the last approximation or until the computer calculates some preset number of approximations. Using this method, write a void function which inputs the initial root approximation and the error bound required and outputs the value of the root. Note, that the void function will require two function subprograms for $f(x)$ and $f'(x)$ which must be supplied by the user. Test your module by finding a root of the equation

$$x^3 - 2x^2 - 5x + 6$$

whose roots are (3,-2,1).

12.9 Use the Newton-Raphson technique to derive a formula to compute the n^{th} root of a number and write a function to implement this algorithm which inputs the number whose root is required, its initial approximation and the error bound required. Test your algorithm by comparing the output with known roots of numbers (e.g. square root of 4, cube root of 9 etc.)

12.10 Suppose we are given a set of data points $(x_1, x_2, ..., x_n)$ at which the data values are $[f(x_1), f(x_2), ..., f_(x_n)]$. We want to fit a straight line of the form $a_0 + a_1 x_i$ with coefficients a_0 and a_1 chosen to minimize the error function

$$e = \sum_{i=1}^{n} [(a_0 + a_1 x_i) - f(x_i)]^2.$$

This error function is a minimum when

$$\frac{\partial e}{\partial a_0} = 0 \quad \text{and} \quad \frac{\partial e}{\partial a_1} = 0.$$

These conditions provide a linear least squares solution for a_1 and a_0 given by

$$a_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(f_i - \bar{f})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

and

$$a_0 = \bar{f} - a_1 \bar{x}$$

where $f_i \equiv f(x_i)$ and $\bar{x}$ and $\bar{f}$ are the average values of x_i and f_i respectively. Write a function that reads in values of x_i and f_i and outputs the coefficients a_0 and a_1. Test the module by inputting data obtained for given values of a_0 and a_1 and examining the values computed via this method.

Part IV

DSP: Methods, Algorithms and Building a Library

Chapter 13

Digital Filters and the FFT

Many DSP algorithms can be classified in terms of a (digital) filter and there are two important classes of digital filter used in DSP:

(i) convolution-type filters;

(ii) Fourier based filters.

Convolution filters are non-recursive filters. They are linear processes which operate on the data directly; they are real space filters. Fourier filters operate on data obtained by computing the Discrete Fourier Transform of a signal. This is accomplished using the Fast Fourier Transform algorithm.

13.1 Digital Filters

Real Space Filters

Real space filters are based on some form of 'moving window' principle. A sample of data about a given element of the signal is processed giving (typically) one output value. The window is then moved on to the next element of the signal and the process repeated. A common real space filter is the Finite Impulse Response or FIR filter. This is a non-recursive filter, a discrete convolution operation of the form

$$s_i = \sum_j p_{i-j} f_j$$

where f_i is the input, s_i is the output and p_i is the 'kernel' of the filter.

Fourier Space Filters

Fourier space filters are usually multiplicative operations which operate on the Discrete Fourier Transform (DFT) of the signal. If S_i, P_i and F_i are taken to denote the DFT's of s_i, p_i and f_i respectively, then, using the discrete convolution theorem, in

Fourier space,

$$s_i = \sum_j p_{i-j} f_j$$

transforms to

$$S_i = P_i F_i.$$

If p_i is composed of just a few elements, then the discrete convolution can be computed directly. However, if p_i is composed of many elements then it is numerically more efficient to use a Fast Fourier Transform (FFT) and perform the filtering operation in Fourier space. A Fourier space filter is just one type (although a fundamentally important type) of transform space filter where the transform is chosen according to the properties of the input data and the desired result of the output. Examples of such transforms are given in Chapter 5.

Inverse Problems

Many problems associated with the processing of digital signals are related to what is generally referred to as inverse problems. Consider the case when

$$s_i = \sum_j p_{i-j} f_j.$$

This discrete convolution can be written in the form (see Chapter 16)

$$\mathbf{s} = P\mathbf{f}$$

where P is a matrix whose elements are the components of the kernel p_i arranged in an appropriate order. A common inverse problem encountered in DSP is: 'given s_i and p_i compute f_i'. This inverse problem is called deconvolution. Since a convolution can be written in terms of a matrix equation, the solution to this problem can be compounded in terms of the solution to a set of linear simultaneous equations which can be undertaken using methods discussed in Part II of this book. However, if we express the same problem in Fourier space, then we have

$$S_i = P_i F_i$$

and the problem now becomes: 'given S_i and P_i find F_i'. In this case, the solution appears trivial since

$$F_i = \frac{S_i}{P_i}$$

where $1/P_i$ is known as the 'inverse filter'. However, such approaches to solving this problem lead to ill-conditioned results especially when the data (P_i and/or F_i) is noisy and methods of regularization are needed to develop robust solutions. This is the Fourier space equivalent problem of solving the system of linear equations $P\mathbf{f} = \mathbf{s}$ when they are ill-conditioned (see Chapter 8) in real space.

Another approach is to use the logarithm to convert the multiplicative process $P_i F_i$ into an additive one, i.e.

$$\log S_i = \log P_i + \log F_i$$

and attempt to solve for F_i using the result that

$$F_i = \exp[\log S_i - \log P_i]$$

but again, problems can occur in the computation of these functions when the data is noisy.

All aspects of Fourier transform based filtering require the efficient computation of a discrete Fourier transform and in the following section the Fast Fourier transform is discussed.

13.2 The Fast Fourier Transform

The Fast Fourier Transform or FFT is an algorithm for computing the Discrete Fourier Transform with less additions and multiplications. The DFT (in standard form) of an N-point vector is given by

$$F_m = \sum_n f_n \exp(-2\pi i n m/N)$$

where

$$\sum_n \equiv \sum_{n=0}^{N-1}.$$

How much computation is involved in computing the DFT of N points? If we write

$$W_N = \exp(-2\pi i/N)$$

then

$$F_m = \sum_n W_N^{nm} f_n.$$

This result is a matrix equation which can be written in the form

$$\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} W_N^{00} & W_N^{01} & \dots & W_N^{0(N-1)} \\ W_N^{10} & W_N^{11} & ... & W_N^{1(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & W_N^{(N-1)1} & \dots & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{(N-1)} \end{pmatrix}.$$

In this form, we see that the DFT is essentially computed by multiplying an N-point vector f_n by a matrix of coefficients given by a (complex) constant W_N to the power of nm. This requires $N \times N$ multiplications Thus, for example, to compute the DFT of 1000 points requires 10^6 multiplications!

13.2.1 Basic Ideas

By applying a simple but very elegant trick, a N-point DFT can be written in terms of two $\frac{N}{2}$-point DFT's. The FFT algorithm is based on repeating this trick again

and again until a single point DFT is obtained. The basic idea is compounded in the following result:

$$\sum_{n=0}^{N-1} f_n \exp(-2\pi inm/N)$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} \exp[-2\pi i(2n)m/N] + \sum_{n=0}^{(N/2)-1} f_{2n+1} \exp[-2\pi i(2n+1)m/N]$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} \exp[-2\pi inm/(N/2)] + \exp(-2\pi im/N) \sum_{n=0}^{(N/2)-1} f_{2n+1} \exp[-2\pi inm/(N/2)]$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} f_{2n+1} W_{N/2}^{nm}.$$

Fundamental Property

The result above leads to the fundamental property:

DFT of a N-point array
= DFT of even components + $W_N^m \times$ DFT of odd components.

Using the subscripts e and o to represent odd and even components respectively, we can write this result in the form

$$F_m = F_m^e + W_N^m F_m^o$$

The important thing to note here, is that the evaluation of F_m^e and F_m^o is over $N/2$ points, the $N/2$ even components and the $N/2$ odd components of the original N-point array. To compute F_m^e and F_m^o we only need half the number of multiplications that are required to compute F_m.

Repeating the Trick: The Successive Doubling Method

Because the form of the expressions for F_m^e and F_m^o are identical to the form of the original N-point DFT, we can repeat the idea and decompose F_m^e and F_m^o into even and odd parts producing a total four $\frac{N}{4}$-point DFT's as illustrated below.

$$\begin{array}{ccc} & F_m & \\ & \Downarrow & \\ F_m^e & + & W_N^m F_m^o \\ \Downarrow & \Downarrow & \Downarrow \\ F_m^{ee} + W_{N/2}^m F_m^{eo} & + & W_N^m \times (F_m^{oe} + W_{N/2}^m F_m^{oo}) \end{array}$$

We can continue subdividing the data into odd and even components until we get down to the DFT of a single point. However, because the data is subdivided into odd

and even components of equal length, we require an initial array of size $N = 2^k, \quad k = 1, 2, 3, 4, ...$ Computing the DFT in this way reduces the number of multiplications needed to the order of $N \log N$ which, for even moderate values of N is considerably smaller than N^2.

Example

Consider the 2 point FFT with data (f_0, f_1). Then

$$F_m = \sum_{n=0}^{1} f_n W_2^{nm} = W_1^0 f_0 + W_2^m W_1^0 f_1 = f_0 + \exp(i\pi m) f_1$$

so that

$$F_0 = f_0 + f_1$$

and

$$F_1 = f_0 + \exp(i\pi) f_1 = f_0 - f_1.$$

Now consider the 4 point FFT operating on the data (f_0, f_1, f_2, f_3). Here,

$$F_m = \sum_{n=0}^{3} f_n W_4^{nm} = \sum_{n=0}^{1} f_{2n} W_2^{nm} + W_4^m \sum_{n=0}^{1} f_{2n+1} W_2^{nm} = f_0 + W_2^m f_2 + W_4^m (f_1 + W_2^m f_3).$$

Thus,

$$F_0 = f_0 + f_1 + f_2 + f_3,$$
$$F_1 = f_0 + f_2 W_2 + f_1 W_4 + f_3 W_4 W_2,$$
$$F_2 = f_0 + f_2 W_2^2 + f_1 W_4^2 + f_3 W_4^2 W_2^2$$

and

$$F_3 = f_0 + f_2 W_2^3 + f_1 W_4^3 + f_3 W_4^3 W_2^3.$$

Further, certain values of W_N^m are simple, for example,

$$W_2^0 = 1, \quad W_2^1 = -1, \quad W_4^0 = 1, \quad W_4^1, \quad W_4^2 = -1, \quad W_4^3 = -i.$$

Also, if we let $k = n + N/2$, then

$$\exp\left(\frac{2\pi i k}{N}\right) = \exp\left(\frac{2\pi i (n + N/2)}{N}\right) = \exp\left(\frac{2\pi i n}{N}\right) \exp(\pi i) = -\exp\left(\frac{2\pi i n}{N}\right)$$

and thus,

$$W_N^{(n+N/2)} = -W_N^n.$$

13.2.2 Bit Reversal

Consider the 8-point array

$$f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7$$

and the decomposition of this array into even and odd components as given below.

	Even arguments	Odd arguments	
	f_0, f_2, f_4, f_6	f_1, f_3, f_5, f_7	
Even	Odd	Even	Odd
f_0, f_4	f_2, f_6	f_1, f_5	f_3, f_7

To use the FFT algorithm, the input array must first be expressed in the form

$$f_0, f_4, f_2, f_6, f_1, f_5, f_3, f_7.$$

The general procedure for re-ordering an input array of this type follows a simple bit-reversal rule where the position of an element of the original array f_i is expressed in binary form. The bits are then reversed to obtain the position of this element in the re-ordered array as illustrated below.

Original Argument	Original Array	Bit-reversed Argument	Re-ordered Array
000	f_0	000	f_0
001	f_1	100	f_4
010	f_2	010	f_2
011	f_3	110	f_6
100	f_4	001	f_1
101	f_5	101	f_5
110	f_6	011	f_3
111	f_7	111	f_7

If the FFT algorithm is applied to an array in its natural order, then the output is bit-reversed. Bit reversal can be applied either before of after the computations commence. The effect of applying this method is to reduce the number of multiplications from $O(N^2)$ to $O(N \log N)$ which for relatively small array sizes considerably reduces the time taken to perform a DFT.

The method discussed above depends on using array sizes of 2^N and is therefore a base-2 algorithm. It is natural to ask, why this method cannot be extended, i.e. instead of decomposing the original array into two arrays (based on the odd and even components of the original) why not decompose it into three or four arrays and repeat the process accordingly leading to base-3 and base-4 algorithms. The problem with this approach is that, although it can lead to slightly less operations, the reordering of the data required to establish an appropriate output is significantly more complicated than bit reversal. The extra effort that is required to establish a re-ordering algorithm tends to outweigh the reduction in the processing time from adopting a base-3 or base-4 approach.

13.2.3 The FFT in C

The following code is a C (void) function called FFT1D (FFT in one-dimension or 1D) which computes the DFT of a complex input with real and imaginary parts using

the method described above. In this case, the arrays are taken to run from 1 to n are are re-ordered on both input and output so that the DFT appears in optical forms in which the DC or zero frequency component occurs a $n/2+1$ where n is the size of the array. The algorithm performs either a forward or an inverse DFT using the switch sign. If sign=-1, then the forward DFT is computed and if sign=1, then the inverse DFT is computed.

```
#include <math.h>

void FFT1D( float a[], float b[], int n, int sign )
{
  int l,l1,l2,j,jj,i,ii,k,nh,nm;
  float *cr, *ci;
  float den,p,q;
  float ur,ui,vr,vi,wr,wi,urtemp;
  double pi,x;

/*********************************************************************/

/* FUNCTION: This function computes the DFT of a complex array whose
             real and imaginary parts are a and b respectively using
             the successive doubling method.

 NOTES:    The function returns the real (a) and imaginary (b) parts
           of the DFT.

           The size of the arrays n must be an int power of 2.

           Zero frequency occurs at 1+n/2.

           By convention, the forward Fourier transform is obtained
           when sign=-1 and the inverse Fourier transform is obtained
           when sign=1.

           If input is purely real then the imaginary part
           (i.e. array b) must be set to be a null vector.

 PARAMETERS

 Input: a - real part of signal/spectrum
        b - imaginary part of signal/spectrum
        n - size of signal

 Output: a - real part of spectrum/signal
         b - imaginary part of spectrum/signal

 EXTERNAL MODULES: None
```

```
 INTERNAL MODULES: None */

/*********************************************************************/

/* Allocate n+1 words of memory to internal arrays. */

  cr = (float *) calloc( n+1, sizeof( float ) );
  ci = (float *) calloc( n+1, sizeof( float ) );

  /* Compute scaling parameter (den) */

  if ( sign < 0 )
    den=1.0;
  else
    den=n;

  /* Setup constants required for computation. */

  pi = 4.0 * atan( 1.0 );
  p = n;
  q = log( p ) / log( 2.0 );
  k = q;
  nh = n * 0.5;
  nm = n-1;

  /* Generate switched arrays - switch positive/negative
     half spaces to negative/positive half spaces respectively. */

  for ( i=nh+1, j=1; i<=n; i++, j++ )
    {
    cr[j] = a[i];
    ci[j] = b[i];
    }

  for ( i=1; i<=nh; i++, j++ )
    {
    cr[j] = a[i];
    ci[j] = b[i];
    }

  /* Reorder data, i.e. perform 'bit-reversal'. */

  for ( i=1, j=1; i<=nm; i++ )
    {
    if ( i < j )
      {
```

```
      vr = cr[j];
      vi = ci[j];
      cr[j] = cr[i];
      ci[j] = ci[i];
      cr[i] = vr;
      ci[i] = vi;
      }

    jj = nh;
    while ( jj < j )
      {
      j -= jj;
      jj = jj * 0.5;
      }

    j += jj;
    }

  /* Do fast transform computations. */

  for ( l=1; l<=k; l++ )
    {
    l1 = ldexp( 1.0, l );
    x = (2 * pi * (double) sign) / l1;
    wr = cos( x );
    wi = sin( x );
    l2 = l1 * 0.5;
    ur = 1.0;
    ui = 0.0;

    for ( j=1; j<=l2; j++ )
      {

      for ( i=j; i<=n; i+=l1 )
        {
        ii = i + l2;
        vr = (cr[ii] * ur) - (ci[ii] * ui);
        vi = (cr[ii] * ui) + (ci[ii] * ur);
        cr[ii] = cr[i] - vr;
        ci[ii] = ci[i] - vi;
        cr[i] = cr[i] + vr;
        ci[i] = ci[i] + vi;
        }

      urtemp = ur;
      ur = (ur * wr) - (ui * wi);
      ui = (urtemp * wi) + (ui * wr);
```

```
      }
    }

 /*  Scale */

  for ( i=1; i<=n; i++ )
    {
    cr[i] = cr[i] / den;
    ci[i] = ci[i] / den;
    }

  /* Reverse half-spaces - write out data in 'optical form'. */

  for ( i=nh+1, j=1; i<=n; i++, j++ )
    {
    a[j] = cr[i];
    b[j] = ci[i];
    }

  for ( i=1; i<=nh; i++, j++ )
    {
    a[j] = cr[i];
    b[j] = ci[i];
    }

  /* Free space from work arrays. */

  free( cr );
  free( ci );
}
```

13.3 Data Windowing

Unlike the Fourier transform which is expressed in terms of an infinite integral, the DFT involves computing with a discrete array of finite extent. Thus, it is important to assess the difference between the DFT transform of a function and its theoretical Fourier transform. Computing the DFT of a digital signal with N samples is equivalent to multiplying an infinite run of sampled data by a 'window function' which is zero except during the total sampling time $N\Delta t$ and is unity during that time. Consider the following optical form of the DFT

$$F_m = \sum_{n=-N/2}^{(N/2)-1} f_n \exp(-i2\pi nm/N).$$

When an N-point DFT is computed, the data are 'windowed' by a square window function. To identify the 'effect' of computing an N-point DFT, consider

$$F_m \sim \int_{-N/2}^{N/2} f(t)\exp(-i\omega_m t)dt = \int_{-\infty}^{\infty} f(t)w(t)\exp(-i\omega_m t)dt$$

where

$$w(t) = \begin{cases} 1, & \mid t \mid \leq N/2; \\ 0, & \mid t \mid > N/2. \end{cases}$$

Using the product theorem we can write

$$F_m \sim \frac{N}{2\pi}\int_{-\infty}^{\infty} F(\omega_m - \omega)\text{sinc}(\omega N/2)d\omega$$

where

$$\text{sinc}(\omega N/2) = \frac{\sin(\omega N/2)}{\omega N/2}.$$

Thus, a sample of the discrete spectrum F_m obtained by taking the DFT of a N-point signal is not given by $F(\omega_m)$ but by $F(\omega_m)$ convolved with a 'sinc' function. Note, that

$$F_m \to F(\omega_m) \quad \text{as} \quad N \to \infty$$

since

$$\lim_{N\to\infty} \frac{N}{2\pi}\text{sinc}(\omega N/2) = \delta(\omega)$$

and

$$\int_{-\infty}^{\infty} F(\omega_m - \omega)\delta(\omega)d\omega = F(\omega_m).$$

Each sample F_m is an approximation to $F(\omega_m)$ which depends on the influence of the 'sinc' function associated with one sample 'bin' on the next sample 'bin'. The 'sinc' function 'leaks' from one bin to the next producing errors in the values of the neighbouring spectral components. The reason for this 'leakage' is that the square window (i.e. the tophat function) turns on and off so rapidly that its Fourier transform (i.e. the sinc function) has substantial components at high frequencies. To remedy this situation, we can multiply the input data f_n by a window function w_n that changes more gradually from zero to a maximum and then back to zero as n goes from $-N/2$ to $N/2$. Many windows exist for this purpose. The difference lies in trade-off's between the 'narrowness' and 'peakedness' of the 'spectral leakage function' (i.e. the amplitude spectrum) of the window function.

Some Examples of Windows

For $n = 0, 1, ..., N-1$ we have:

1. The Parzan window

$$w_n = 1 - \left| \frac{n - \frac{1}{2}N}{\frac{1}{2}N} \right|$$

2. The Welch window

$$w_n = 1 - \left(\frac{n - \frac{1}{2}N}{\frac{1}{2}N} \right)^2$$

3. The Hanning window (Cosine taper)

$$w_n = \frac{1}{2} \left[1 - \cos \left(\frac{2\pi n}{N} \right) \right]$$

4. The Hamming window

$$w_n = 0.54 - 0.46 \cos \left(\frac{2\pi n}{N} \right)$$

5. The von Hann window (Raised cosine)

$$w_n = \frac{1}{2} \left[1 + \cos \left(\frac{\pi(n - \frac{1}{2}N)}{\frac{1}{2}N} \right) \right]$$

6. The generalized von Hann window

$$w_n = b + 2a \cos \left(\frac{\pi(n - \frac{1}{2}N)}{\frac{1}{2}N} \right); \quad 2a + b = 1$$

7. The Kaiser window

$$w_n = \frac{I_0 \left[\alpha \sqrt{1 - \left(\frac{(n - \frac{1}{2}N)}{\frac{1}{2}N} \right)^2} \right]}{I_0}$$

where I_0 is the modified Bessel function of the first kind and α is a constant. Windows of this type (and many others) are of significant value when the size of the arrays are relatively small. The larger the array that is used to represent a signal, the less the spectral leakage and hence, the less significant the requirement of applying an appropriate window becomes. Data windowing was particularly important in the days when the size of an array that could be stored and processed was relatively small and hence, text books in this field that date from the 1960s and 1970s (particularly those concerned with the application of microprocessors) tend to make a 'big deal' over the 'theory of windows'.

13.4 Computing with the FFT

Computing with the FFT involves a basic principle which is that the FFT can be used to implement digital algorithms derived from some theoretical result which is itself based on Fourier theory, provided the data is adequately sampled (to avoid aliasing) and appropriate windows are used (as required) to minimize spectral leakage.

The FFT provides a complex spectrum with real and imaginary arrays a_i and b_i respectively. From the output of the FFT we can construct a number of useful functions required for spectral analysis:

(i) The discrete amplitude spectrum given by $\sqrt{a_i^2 + b_i^2}$.

(ii) The discrete power spectrum $a_i^2 + b_i^2$.

(iii) The discrete phase spectrum $\tan^{-1}(b_i/a_i) \pm 2\pi n$.

The dynamic range of the amplitude and power spectra is often very low especially at high frequencies. Analysis of the spectrum can be enhanced using the logarithmic function and generating an output given by

$$\log\sqrt{a_i^2 + b_i^2}$$

or

$$\log(a_i^2 + b_i^2).$$

It practice, it is typical to add 1 to $a_i^2 + b_i^2$ to prevent a singularity occurring if a_i and b_i are equal to zero for a particular value of i. This also ensures that the output is greater than or equal to zero. If a logarithm to base 10 or $\log_{10}$ is used, the scale of the spectrum is measured in decibels (dB) where one decibel is equal to one tenth of a logarithmic unit. This scale is often used by engineers for spectral analysis and originates from the legacy of acoustic signal analysis.

13.5 Discrete Convolution and Correlation

To convolve two discrete arrays p_i and f_i of equal length using an FFT we use the convolution theorem and consider the result

$$s_i = p_i \otimes f_i$$

$$\Downarrow$$

$$S_i = P_i F_i$$

$$\Downarrow$$

$$s_i = \text{Re}[\text{IDFT}(S_i)]$$

where S_i, P_i and F_i are the DFT's of s_i, p_i and f_i respectively computed using the FFT and IDFT denotes the inverse DFT also computed using the FFT. A typical program will involve the following steps:

1. Input the real arrays p and f.

2. Set the imaginary parts associated with p, f and s (denoted by pi, fi and si say) to zero.

3. Compute the DFT's of f and p using the FFT.

4. Do complex multiplication:

   s=p×f-pi×fi,

   si=p×fi+f×pi.

5. Compute the inverse DFT using the FFT.

6. Write out s.

Using pseudo code, the algorithm for convolution using an FFT is:

```
for i=1 to n; do:
   fr(i)=signal_1(i)
   fi(i)=0.
   pr(i)=signal_2(i)
   pi(i)=0.
enddo
     forward_fft(fr,fi)
     forward_fft(pr,pi)

 for i=1 to n; do:
    sr(i)=fr(i)*pr(i)-fi(i)*pi(i)
    si(i)=fr(i)*pi(i)+pr(i)*fi(i)
 enddo

 inverse_fft(sr,si)
```

Discrete Correlation

To correlate two real discrete arrays p_i and f_i of equal length using an FFT we use the correlation theorem and consider the result

$$s_i = p_i \odot f_i$$

$$\Downarrow$$

$$S_i = P_i^* F_i$$

$$\Downarrow$$

$$s_i = \text{Re}[\text{IDFT}(S_i)]$$

where S_i, P_i and F_i are the DFT's of s_i, p_i and f_i respectively computed using the FFT. Thus, a typical program will involve the following steps:

1. Input the real arrays p and f.

2. Set the imaginary parts associated with p, f and s (denoted by pi, fi and si say) to zero.

3. Compute the DFT's of f and p using the FFT.

4. Do complex multiplication:

   s=p×f+pi×fi,

   si=p×fi-f×pi.

5. Compute the inverse DFT using the FFT.

6. Write out s.

Using pseudo code, the algorithm for correlation using an FFT is:

```
for i=1 to n; do:
   fr(i)=signal_1(i)
   fi(i)=0.
   pr(i)=signal_2(i)
   pi(i)=0.
enddo

 forward_fft(fr,fi)
 forward_fft(pr,pi)

 for i=1 to n; do:
    sr(i)=fr(i)*pr(i)+fi(i)*pi(i)
    si(i)=fr(i)*pi(i)-pr(i)*fi(i)
 enddo

 inverse_fft(sr,si)
```

13.6 Computing the Analytic Signal

The Argand diagram representation provides us with a complex representation of the analytic signal given by

$$s(t) = A(t)\exp[i\theta(t)] = A(t)\cos\theta(t) + iA(t)\sin\theta(t)$$

where $A(t)$ are the amplitude modulations and $\theta(t)$ is the instantaneous phase. Writing $s(t)$ in the form

$$s(t) = f(t) + iq(t)$$

it is shown in Chapter 5 that $q(t)$ - the quadrature signal - is given by the Hilbert transform of $f(t)$. Being able to compute the analytic signal provides us with the

signal attributes, i.e. the amplitude modulation, the frequency modulations and the instantaneous phase (see Chapter 5). The analytic signal $s(t)$ is related to $f(t)$ by

$$s(t) = \frac{1}{\pi} \int_0^\infty F(\omega) \exp(i\omega t) d\omega$$

and $s(t)$ is characterized by a single side-band spectrum whose real and imaginary parts are $f(t)$ and $q(t)$ respectively. We can write this result as

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^\infty 2F(\omega)U(\omega) \exp(i\omega t) d\omega$$

where

$$U(\omega) = \begin{cases} 1, & \omega \geq 0; \\ 0, & \omega < 0. \end{cases}$$

Thus, to compute the Hilbert transform of a discrete function f_i say where $i = 1, 2, ..., N$ we can apply the follow procedure:

1. Take the DFT of f_i to get F_i.

2. Set F_i to zero for all negative frequencies.

3. Compute the inverse DFT of F_i.

Then, on output, the real part of the inverse DFT is f_i and the imaginary part of the inverse DFT is q_i. In practice, the DFT can be computed using a FFT. Thus, using pseudo code, the FFT algorithm for computing the Hilbert transform is (noting that the DC level occurs at $n/2 + 1$)

```
for i=1 to n; do:
 sreal(i)=signal(i)
 simaginary(i)=0.
enddo
    forward_fft(sreal,simaginary)

for i=1 to n/2; do:
    sreal(i)=0.
    sreal(i+n/2)=2.*sreal(i+n/2)
    simaginary(i)=0.
    simaginary(i+n/2)=2.*simaginary(i+n/2)
enddo

inverse_fft(sreal,simaginary)

 for i=1 to n; do:
    signal=sreal(i)
```

```
    hilbert_transform(i)=simaginary(i)
enddo
```

13.7 Summary of Important Results

Principle of the (base-2) FFT

DFT of N-point array = DFT of even components + $W_N^m \times$ DFT of odd components.

Bit Reversal
Reversal of the binary number representation of the position of an element in an array which is used to reorder the data before repeated application of the principle above.

FFT C Function

```
void FFT1D(float a[ ], float b[ ], int n, int sign)
```

where a and b are the real and imaginary parts respective, n is the array size (an integer power of 2) and sign is a switch to control the computation of the forward DFT (sign=-1) or inverse DFT (sign=1).

Approximation to $F(\omega_m)$

$$F_m \sim \frac{N}{2\pi} F(\omega_m - \omega) \otimes \mathrm{sinc}(\omega N/2),$$

$$F_m \to F(\omega_m) \quad \text{as} \quad N \to \infty.$$

Data Windows
Functions with edge tapers that reduce the spectral leakage generated by the sinc function.

13.8 Further Reading

- Brigham E O, *The Fast Fourier Transform and its Applications*, Prentice-Hall, 1988.
- Bareman A and Yates W, *Digital Signal Processing Design*, Pitman, 1988.
- Van den Enden A W M and Verhoeckx N A M, *Discrete Time Signal Processing*, Prentice-Hall, 1989.
- INMOS Limited, *Digital Signal Processing*, Prentice Hall, 1989

- Press W H, Teukolsky S A, Vetterling W T and Flannery B P, *Numerical Recipes in C*, Cambridge University Press, 1994.

13.9 Programming Problems

In the questions that follow, the functions required should be void functions written in ANSI C. They should be compiled, tested and then added to a digital signal processing object library *dsplib.lib* say. In each case, a simple I/O test procedure should be written; the I/O being studied graphically using the graphics function *gsignal* discussed in Appendix D (or other graphics facilities as appropriate) working with arrays of size 64, 128, 256 or 512. Each array should be processed using elements running from 1 to n, i.e. using vectors $\mathbf{x} \in R^n$. This requires, where necessary, n+1 words of memory to be assigned.

Each function should be self-contained within the context of the DSP algorithm to be coded. The problems given are in two sections; the first is concerned with the generation of different digital signals and the second section is concerned with using the FFT algorithm to undertake a variety of computations relating to the material covered in this chapter. In each case, n (which is of type integer) is the size of the array.

13.9.1 Digital Signal Generation

13.1 Write a function to compute two spikes of unit amplitude and arbitrary width (apart) which are at the center of the output array.

```
void SPIKES(float s[ ], int n, int w)
```

where s is the output and w is the width.

13.2 Write a function to compute a tophat function of arbitrary width.

```
void TOPHAT(float s[ ], int n, int w)
```

where s is the output and w is the width of tophat which is placed at the center of the array.

13.3 Write a function to compute a triangle function of unit amplitude and arbitrary (base) width.

```
void TRIANGLE(float s[ ], int n, int w)
```

where s is the output and w is the width of the base.

13.4 Write a function to compute a Gaussian function of arbitrary width (standard deviation).

```
void GAUSSIAN(float s[ ], int n, int w)
```

where s is the output and w is the half width of function at $1/e$.

13.5 Write a function to compute a cosine function with an arbitrary number of periods.

```
void COSINE(float s[ ], int n, int p)
```

where s is the output and p is the number of periods.

13.6 Write a function to compute a sine function with an arbitrary number of periods.

```
void SINE(float s[ ], int n, int p)
```

where s is the output and p is the number of periods.

13.9.2 Computing with the FFT

13.7 Write a program to compute the DFT of the output of the functions in questions 13.1-13.6 above using the function FFT1D provided and plot the real and imaginary parts. Remember to initialize the imaginary part - set it to zero - before using FFT1D. Also, compute the inverse DFT of the results and study the real and imaginary parts. Note that the imaginary array is not full of zeros (as theoretically it should be), but filled with relatively small values due to the numerical effects of working with arrays of finite length.

13.8 Write a function to compute the amplitude spectrum of a signal using FFT1D.

```
void AMPSPEC(float s[ ], float a[ ], int n)}
```

where s is the input signal and a is the amplitude spectrum (output).

13.9 Write a function to compute the power spectrum of a signal using FFT1D.

```
void POWSPEC(float s[ ], float p[ ], int n)
```

where s is the signal (input) and p is the power spectrum (output).

Compute the discrete amplitude and power spectra of the functions generated in questions 13.1-13.6 using functions AMPSPEC and POWSPEC respectively using a linear and then a logarithmic scale.

13.10 Write a function to scale a signal of arbitrary polarity so that its upper bound (in modulus) is a.

```
void SCALE(float s[ ], int n, float a)
```

where s is the input/output and a is the upper bound.

13.11 Write functions to compute the Parzan, Welch, Hanning and Hamming windows where w is the output:

```
void PARZEN(float w[ ], int n)

void WELCH(float w[ ], int n)

void HANNING(float w[ ],int n)

void HAMMING(float w[ ], int n)
```

Study the frequency response of these filters by computing their amplitude spectra on a logarithmic scale.

13.12 Write a function to differentiate a signal using FFT1D and the filter obtained from Fourier analysis.

```
void DIF(float s[ ], int n)
```

where s is the input/output. Test this function by computing the gradient of the tophat function using function TOPHAT. Explain the results.

13.13 Write a function to convolve two real (one-dimensional) arrays using FFT1D.

```
void CONVOLVE(float f[ ], float p[ ], float s[ ], int n)
```

where f is the input, p is the IRF and s is the output signal. Test your algorithm by convolving two spikes with a Gaussian using functions SPIKES and GAUSSIAN respectively. Observe the effect of varying the width of the Gaussian.

13.14 Write a function to cross-correlate two digital signals using FFT1D.

```
void CROSCOR(float f[ ], float p[ ], float s[ ], int n)
```

where f is an input signal, p is an input signal and s is the output. Test your algorithm by correlating two sine waves and then a sine wave with a cosine wave (with equal periodicity).

13.15 Write a function to compute the autocorrelation function of a signal using FFT1D.

```
void AUTOCOR(float f[ ], float s[ ], int n)
```

where f is the input signal and s is the output. Test your algorithm by autocorrelating a signal consisting of two spikes of unit amplitude and then a tophat function. Explain the results.

13.16 Write a function to filter a given input with an arbitrary real filter.

```
void FILTER(float s[ ], float f[ ], int n)
```

where s is the input/output and f is the filter which is real. Use this algorithm together with function TOPHAT to lowpass filter a signal consisting of a cosine wave. Study the effect in varying the bandwidth of the filter.

13.17 Write a function to generate the analytic signal.

```
void ANASIG(float f[ ], float q[ ], int n)
```

where f is the real part (input/output) and q is the imaginary or 'quadrature' component (output).

13.18 Write a function to compute the Hilbert transform of an input using function ANASIG.

```
void HILBERT(float s[ ], int n)
```

where s is the input/output. Test your algorithm by computing the Hilbert transform of a sine function. Remember that the Hilbert transform of a sine function is a cosine function. Hence, the output should be compounded in a 90^o phase shift.

13.19 Use function ANASIG to compute the amplitude envelope of a signal.

```
void AMPENV(float s[ ], float a[ ], int n)
```

where s is the input signal and a is the amplitude envelope. Use this function to compute the amplitude envelope of a sine and cosine wave. Study the effect of increasing the number of periods on the amplitude envelope. Explain the results observed.

13.20 Using FFT1D and the principle of 'sinc interpolation' (see Chapter 4), write a function to interpolate an input from 2^k to 2^l where $l > k$ and k and l are integers.

```
void SINCINT(float x[ ], int n, float y[ ], int m)
```

where x is the input signal, n is the array size of the input, y is the interpolated output signal and m is the array size of the output. Test your algorithm by interpolating a sine function from 128 to 256 data points.

Further Exercises

In addition to the exercises given above, the reader should use the software developed to investigate the properties of the Fourier transform (as discussed in Chapter 4). For example:

- the shifting property, i.e. shift the spectrum of a sine wave for example and observe the output;

- the product theorem, i.e. multiply a sine wave with itself and study its spectrum.

The purpose here, is to observe the characteristics of the Fourier transform and its properties that were derived theoretically in Chapter 4 and for the reader to satisfy him/her self that these properties are reflected in the computations performed by the FFT algorithm. However, it should be understood that the numerical output will always have features that are a product of using discrete arrays of finite length.

Chapter 14

Frequency Domain Filtering with Noise

Digital filtering in the frequency domain is the basis for an important range of filters. This mode of filtering relies on extensive use of the FFT and without the FFT, this type of filtering would be very time consuming to implement. Fourier based filters usually have a relatively simple algebraic form. They change the 'shape' of the input spectrum via a multiplicative process of the type:

$$\text{Output spectrum} = \text{Filter} \times \text{Input spectrum}$$

Filters of this type characterize the frequency response of a system to a given input. The algebraic form of the filter usually originates from the solution (with appropriate conditions and approximations) to a particular type of problem.

14.1 Highpass, Lowpass and Bandpass Filters

An operation which changes the distribution of the Fourier components of a function (via a multiplicative process) may be defined as a (Fourier) filtering operation. Thus, in an operation of the form

$$S_i = P_i F_i$$

P_i may be referred to as a filter and S_i can be considered to be a filtered version of F_i. In general, Fourier filters fall into one of three classes:

(i) lowpass filters;

(ii) highpass filters;

(iii) bandpass filters.

A lowpass filter is one which suppresses or attenuates the high frequency components of a spectrum while 'passing' the low frequencies within a specified range. A highpass filter does exactly the opposite to a lowpass filter; it attenuates the low frequency components of a spectrum while 'passing' the high frequencies within a specified

range. A bandpass filter only allows those frequencies within a certain band to pass through. In this sense, lowpass and highpass filters are just special types of bandpass filters. Many signals can be modelled in terms of some bandpass filter modifying the distribution of the complex Fourier components associated with an information source. Data processing is then required to restore the out-of-band frequencies in order to recover the complex spectrum of the source. This requires a good estimate of the original bandpass filter.

14.2 The Inverse Filter

The inverse filter is a straightforward approach to deconvolving the equation

$$s_i = p_i \otimes f_i + n_i.$$

In the absence of any useful information about the noise n_i, we may ignore it under the assumption that its total contribution to the signal s_i is small. We can then set about inverting the reduced equation

$$s_i = p_i \otimes f_i.$$

The basic approach to solving this problem is to process the data s_i in Fourier space. Using the convolution theorem, we have

$$S_i = P_i F_i.$$

Re-arranging and taking the inverse DFT, denoted by IDFT, we get

$$f_i = \text{IDFT}\left(\frac{S_i}{P_i}\right) = \text{IDFT}\left(\frac{P_i^* S_i}{|\, P_i \,|^2}\right).$$

The function $1/P_i$ is known as the inverse filter.

Criterion for the Inverse Filter

The criterion for the inverse filter is that the mean square of the noise is a minimum. In other words, f_i is chosen in such a way that the mean square error

$$e = \|n_i\|^2 = \|s_i - p_i \otimes f_i\|^2$$

is a minimum. Using the orthogonality principle (see Chapter 8), this error is a minimum when

$$[s_i - p_i \otimes f_i] \odot p_i^*(x) = 0$$

and through the correlation and convolution theorems, in Fourier space, this equation becomes

$$[S_i - P_i F_i] P_i^* = 0.$$

Solving for F_i, we obtain the same result as before, namely,

$$F_i = \frac{P_i^*}{|\, P_i \,|^2} S_i.$$

Computational problems

In principle, the inverse filter provides an exact solution to the problem when n_i approaches zero. However, in practice this solution is fraught with difficulties. First, the inverse filter is invariably a singular function due to zeros occurring in $\mid P_i \mid$. Equally bad is the fact that even if the inverse filter is not singular, it is usually ill-conditioned. This is where the magnitude of P_i goes to zero so quickly as i increases that $1/ \mid P_i \mid^2$ rapidly acquires extremely large values. The effect of this ill-conditioning is typically to amplify the noisy high frequency components of S_i. This can lead to a reconstruction for f_i which is dominated by the noise in s_i. The inverse filter can therefore only be used when: (i) the inverse filter is nonsingular; (ii) the signal to noise ratio of the data is very small. Such conditions are rare. The computational problems associated with the inverse filter can be avoided by implementing a variety of different filters whose individual properties and characteristics are suited to certain types of experimental data.

14.3 The Wiener Filter

The Wiener filter is a minimum mean square filter (i.e. it is based on application of the least squares principle). This filter is commonly used for signal and image restoration in the presence of additive noise. It is named after the American mathematician Norbert Wiener who was among the first to discuss its properties. The problem (and consequent solution) can be formulated using either continuous or discrete functions. Here, the latter approach is taken which is consistent with the analysis of digital signals. The problem is as follows: Let s_i be a digital signal consisting of N real numbers $i = 0, 1, ..., N-1$ formed by a time invariant stationary process of the type

$$s_i = \sum_j p_{i-j} f_j + n_i$$

where

$$\sum_j \equiv \sum_{j=0}^{N-1}.$$

Find an estimate for f_i of the form

$$\hat{f}_i = \sum_j q_j s_{i-j}.$$

Clearly, the problem is to find q_i. Wiener's solution to this problem is based on utilizing the 'least squares principle'.

14.3.1 The Least Squares Principle

Application of the least squares principle to this class of problem is based on finding a solution for q_i such that

$$e = \| f_i - \hat{f}_i \|_2^2 \equiv \sum_{i=0}^{N-1} (f_i - \hat{f}_i)^2$$

is a minimum. Under the condition that the noise is signal independent, i.e.

$$\sum_j n_{j-i} f_j = 0 \quad \text{and} \quad \sum_j f_{j-i} n_j = 0,$$

the DFT of q_i is given by

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \frac{|N_i|^2}{|F_i|^2}}$$

where F_i, P_i and N_i are the DFT's of f_i, p_i and n_i respectively. Here, Q_i is known as the 'Wiener Filter' and using the convolution theorem we can write the required solution as

$$\hat{f}_i = \text{IDFT}\,(Q_i S_i)$$

where S_i is the DFT of s_i, IDFT is taken to denote the inverse DFT and since the data s_i is real, only the real part of the output is taken.

14.3.2 Derivation of the Wiener Filter

The function e defines an 'error' in the sense that the closer $\hat{f}_i$ is to f_i, the smaller the error becomes. The error is a function of q_i and hence is a minimum when

$$\frac{\partial}{\partial q_k} e(q_j) = 0 \quad \forall k.$$

Differentiating, we get (as discussed in Chapter 8)

$$\sum_{i=0}^{N-1} \left(f_i - \sum_j q_j s_{i-j} \right) s_{i-k} = 0.$$

We now use the convolution and correlation theorems to write the above equation in the form

$$F_i S_i^* = Q_i S_i S_i^*$$

giving

$$Q_i = \frac{S_i^* F_i}{\mid S_i \mid^2}$$

where F_i, S_i and Q_i and the DFT's of f_i, s_i and q_i respectively. This function can be written in terms of P_i and N_i (the DFT's of p_i and n_i respectively) since

$$s_i = \sum_j p_{i-j} f_j + n_i$$

which transforms to

$$S_i = P_i F_i + N_i$$

via the convolution theorem. This gives

$$S_i^* F_i = (P_i^* F_i^* + N_i^*) F_i = P_i^* \mid F_i \mid^2 + N_i^* F_i.$$

Now,

$$\mid S_i \mid^2 = S_i S_i^* = (P_i F_i + N_i)(P_i^* F_i^* + N_i^*) = \mid P_i \mid^2 \mid F_i \mid^2 + \mid N_i \mid^2 + P_i F_i N_i^* + N_i P_i^* F_i^*$$

and

$$Q_i = \frac{P_i^* \mid F_i \mid^2 + N_i^* F_i}{\mid P_i \mid^2 \mid F_i \mid^2 + \mid N_i \mid^2 + P_i F_i N_i^* + N_i P_i^* F_i^*}.$$

14.3.3 Signal Independent Noise

If we assume that the noise is signal independent, then we can say that to a good approximation, there is no correlation between the signal (in particular, the input f_i) and the noise and visa versa. This statement is compounded mathematically by the conditions

$$\sum_j n_{j-i} f_j = 0 \quad \text{and} \quad \sum_j f_{j-i} n_j = 0.$$

Using the correlation theorem, these conditions can be written in the form

$$N_i^* F_i = 0 \quad \text{and} \quad F_i^* N_i = 0.$$

These conditions allow us to drop the cross terms in the expression for Q_i leaving us with the result

$$Q_i = \frac{P_i^* \mid F_i \mid^2}{\mid P_i \mid^2 \mid F_i \mid^2 + \mid N_i \mid^2}$$

or after rearranging

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \frac{|N_i|^2}{|F_i|^2}}.$$

14.3.4 Properties of the Wiener Filter

As the noise goes to zero (i.e. as $\mid N_i \mid^2 \rightarrow 0$) the Wiener filter reduces to the 'inverse filter' for the system, i.e.

$$\frac{P_i^*}{\mid P_i \mid^2}.$$

Hence, with minimal noise, the Wiener filter behaves like the inverse filter. As the power spectrum of the input goes to zero (i.e. as $\mid F_i \mid^2 \rightarrow 0$), the Wiener filter has zero gain. This solves problems concerning the behaviour of the filter as $\mid P_i \mid^2$ approaches zero. In other words, the filter is 'well conditioned'. Note that the quotient $\mid F_i \mid^2 / \mid N_i \mid^2$ is a measure of the Signal-to-Noise Ratio (SNR).

14.3.5 Practical Implementation

The Wiener filter is given by

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \frac{|N_i|^2}{|F_i|^2}}.$$

Clearly, the main problem with this filter is that in practice, accurate estimates of $\mid N_i \mid^2$ and $\mid F_i \mid^2$ are usually not available. The practical implementation of the Wiener filter usually involves having to make an approximation of the type

$$Q_i \sim \frac{P_i^*}{\mid P_i \mid^2 + \Gamma}$$

where Γ is a suitable constant. The value of Γ ideally reflects knowledge on the SNR of the data, i.e.

$$\Gamma \sim \frac{1}{(\mathrm{SNR})^2}.$$

In practice, it is not uncommon for a user to apply the Wiener filter over a range of different value of SNR and then choose a restoration $\hat{f}_i$ which is optimum in the sense that it is a good approximation to the users *a priori* knowledge on the expected form of the impulse response function.

14.3.6 FFT Algorithm for the Wiener Filter

Clearly, the Wiener filter has a relative simple algebraic form. The main source of CPU time is the computation of the DFT's. In practice this is done by restricting the data to be of size 2^k and using a FFT. Using pseudo code, the algorithm for the Wiener filter is:

```
snr=snr*snr
constant=1/snr

for i=1, 2, ..., n; do:
   sr(i)=signal(i)
   si(i)=0.
   pr(i)=IRF(i)
   pi(i)=0.
enddo
       forward_fft(sr,si)
       forward_fft(pr,pi)

for i=1, 2, ..., n; do:
    denominator=pr(i)*pr(i)+pi(i)*pi(i)+constant

               fr(i)=pr(i)*sr(i)+pi(i)*si(i)
               fi(i)=pr(i)*si(i)-pi(i)*sr(i)
               fr(i)=fr(i)/denominator
               fi(i)=fi(i)/denominator
enddo
inverse_fft(fr,fi)

 for i=1, 2, ..., n; do:
     hatf(i)=fr(i)
 enddo
```

The Wiener filter is one of the most robust filters for solving problems of this kind, restoring signals in the presence of additive noise. It can be used with data of single or dual polarity and for 1D or 2D signal processing problems which are the result of linear time invariant processes and non-causal. A loop can be introduced allowing the user to change the value of SNR or to sweep through a range of values of SNR on an interactive basis. This is known as 'interactive restoration'.

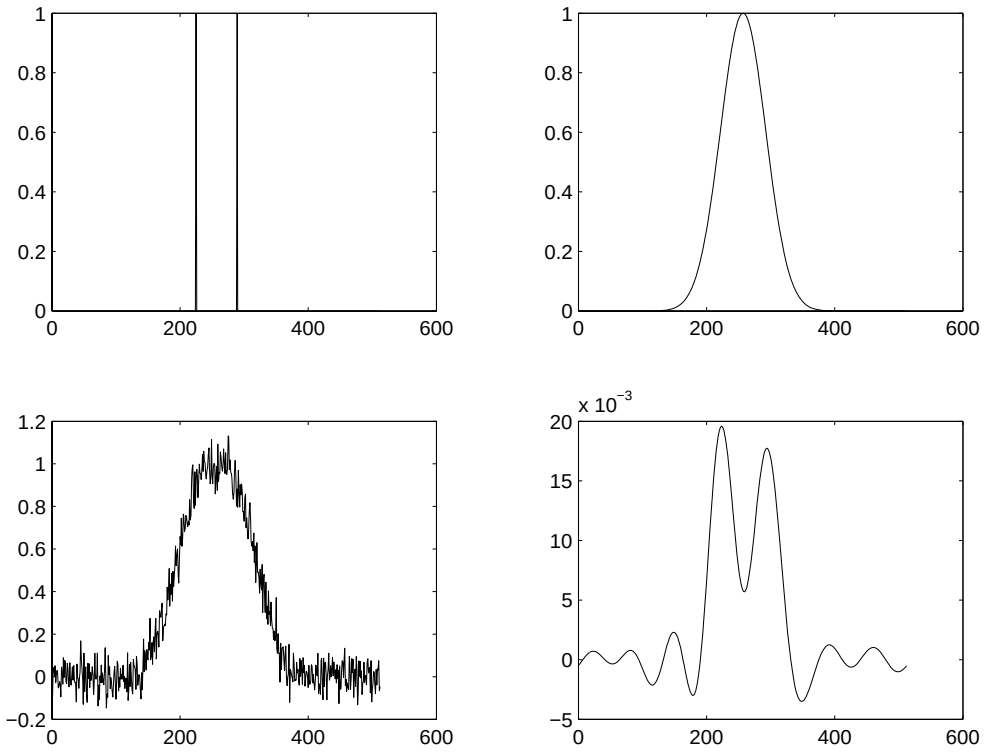


Figure 14.1: Example of a Wiener filter restoration (bottom right) of a noisy signal (bottom left) generated by the convolution of an input consisting of two spikes (top left) with a Gaussian IRF (top right). The simulation and restoration of the signal given in this example is accomplished using the MATLAB function WIENER(50,5,1).

An example of the Wiener filter in action is given in Figure 14.1 using the MATLAB code provided below. In this example, the input signal is assumed to be zero except for two components which are unit spikes or Kronecker delta's spaced apart by as user defined amount. The reason for using such a function as an input, is that it is ideal for testing the filtering operation in terms of the resolution obtained in the same way that two point sources can be used for evaluating the resolution of an imaging system for example. This type of test input can be considered in terms of a two impulses which need to be recovered from the effects of a system characterized by a known IRF which 'smears' the impulses together. Further, we can consider such

an input to be a binary stream with two non-zero elements in which the task is their optimal recovery from an output with additive (Gaussian) noise.

```
function WIENER(sigma,snr_signal,snr_filter)
%Input:
%      sigma - standard deviation of Gaussian IRF
%      snr_signal - signal-to-noise ratio of signal
%      snr_filter - signal-to-noise ratio for computing Wiener filter
%
n=512;    %Set size of array (arbitrary)
nn=1+n/2; %Set mid point of array
m=64;     %Set width of spikes

%Compute input function (two spikes of width m centered
%at the mid point of the array).
mm=m/2;
for i=1:n
   f(i)=0.0;
end
   f(nn-mm)=1.0;
   f(nn+mm)=1.0;

%Plot result
figure(1);
subplot(2,2,1), plot(f);

%Compute the IRF - a unit Gaussian distribution
for i=1:n
   x=i-nn;
   p(i)=exp(-(x.*x)/(sigma*sigma));
end

%Plot result
subplot(2,2,2), plot(p);

%Convolve f with p using the convolution theorem and normalize to unity.
f=fft(f); p=fft(p);
          f=p.*f;
          f=ifft(f); f=fftshift(f); f=real(f);
f=f./max(f); %N.B. No check on case when f=0.

%Compute random Gaussian noise field and normalize to unity.
noise=randn(1,n);
noise=noise./max(noise);

%Compute signal with signal-to-noise ratio defined by snr_signal.
s=f+noise./snr_signal;
```

```
%Plot result
subplot(2,2,3), plot(s);

%Restore signal using Wiener filter.

%Transform signal into Fourier space.
s=fft(s);

%Compute Wiener filter.
gamma=1/(snr_filter).^2;
rest=(conj(p).*s)./((abs(p).*abs(p))+gamma);
rest=ifft(rec); rest=fftshift(rest); rest=real(rest);

%Plot result
subplot(2,2,4), plot(rest);
```

Note that the signal-to-noise ratio of the input signal is not necessarily the same as that used for regularization in the Wiener filter. As the noise increases, a larger value of SNR (as used for the Wiener filter) is required but this in turn leads to more ringing in the restoration. The ideal restoration, is one that provides optimum resolution of the input signal with minimum ringing. This leads to a method of automation by searching for a restoration in which the optimum result is that for which the ratio

$$\frac{\sum_i |\ \text{grad of } \hat{f}_i\ |}{\sum_i \text{zeros in } \hat{f}_i}$$

is a maximum. This is the ratio of the cumulative gradient of the output (which is a measure of the resolution of the restoration) to the number of zero crossings (which is a measure of the amount of ringing generated by the Wiener filter).

14.3.7 Estimation of the Signal-to-Noise Power Ratio

From the algebraic form of the Wiener Filter derived above, it is clear that this particular filter depends on: (i) the functional form of the Fourier transform of the Impulse Response Function (the Transfer Function) P_i that is used; (ii) the functional form of $|\ N_i\ |^2\ /\ |\ F_i\ |^2$. The IRF of the system can usually be found by literally recording the effect a system has on a single impulse as an input which leaves us with the problem of estimating the signal-to-noise power ratio $|\ F_i\ |^2\ /\ |\ N_i\ |^2$. This problem can be solved if one has access to two successive recordings under identical conditions as shall now be shown.

Consider two digital signals denoted by s_i and s'_i of the same object function f_i recorded using the same IRF p_i (i.e. the system) but at different times and hence with different noise fields n_i and n'_i. Here, it can be assumed that the statistics of the noise fields are the same. These signals are given by

$$s_i = p_i \otimes f_i + n_i$$

and

$$s_i' = p_i \otimes f_i + n_i'$$

respectively where the noise functions are uncorrelated and signal independent, i.e. we can impose the conditions

$$n_i \odot n_i' = 0, \quad f_i \odot n_i = 0, \quad n_i \odot f_i = 0, \quad f_i \odot n_i' = 0, \quad n_i' \odot f_i = 0.$$

We now proceed to compute the autocorrelation function of s_i given by

$$c_i = s_i \odot s_i.$$

Using the correlation theorem we get

$$C_i = S_i S_i^* = (P_i F_i + N_i)(P_i F_i + N_i)^* = \mid P_i \mid^2 \mid F_i \mid^2 + \mid N_i \mid^2$$

where C_i is the DFT of c_i. Next, we correlate s_i with s_i' giving the cross-correlation function

$$c_i' = s_i \odot s_i'.$$

Using the correlation theorem again, we have

$$C_i' = \mid P_i \mid^2 \mid F_i \mid^2 + P_i F_i N_i'^* + N_i P_i^* F_i^* + N_i N_i'^* = \mid P_i \mid^2 \mid F_i \mid^2$$

The noise-to-signal ratio can now be obtained by dividing C_i by C_i' giving

$$\frac{C_i}{C_i'} = 1 + \frac{\mid N_i \mid^2}{\mid P_i \mid^2 \mid F_i \mid^2}$$

and re-arranging, we obtain the result

$$\frac{\mid N_i \mid^2}{\mid F_i \mid^2} = \left(\frac{C_i}{C_i'} - 1 \right) \mid P_i \mid^2 .$$

Note that both C_i and C_i' can be obtained from the available data s_i and s_i'. Substituting this result into the formula for Q_i, we obtain an expression for the Wiener filter in terms of C_i and C_i' given by

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2} \frac{C_i'}{C_i}.$$

The approach given above, represents one of the most common methods for computing the signal-to-noise ratio of a system.

14.4 Power Spectrum Equalization

As the name implies, the Power Spectrum Equalization (PSE) filter is based on finding an estimate $\hat{f}_i$ whose power spectrum is equal to the power spectrum of the desired function f_i. The estimate $\hat{f}_i$ is obtained by employing the criterion

$$\mid F_i \mid^2 = \mid \hat{F}_i \mid^2$$

together with the linear convolution model

$$\hat{f}_i = q_i \otimes s_i.$$

Like the Wiener filter, the PSE filter also assumes that the noise is signal independent. Since

$$\hat{F}_i = Q_i S_i = Q_i(P_i F_i + N_i)$$

and given that $N_i^* F_i = 0$ and $F_i^* N_i = 0$, we have

$$| \hat{F}_i |^2 = \hat{F}_i \hat{F}_i^* = | Q_i |^2 \, (| P_i |^2 | F_i |^2 + | N_i |^2).$$

The PSE criterion can therefore be written as

$$| F_i |^2 = | Q_i |^2 \, (| P_i |^2 | F_i |^2 + | N_i |^2).$$

Solving for $| Q_i |$, $\hat{f}_i$ is then given by

$$\hat{f}_i = \text{IDFT}(| Q_i | S_i)$$

where $| Q_i |$ is the PSE filter given by

$$| Q_i | = \left(\frac{1}{| P_i |^2 + | N_i |^2 / | F_i |^2} \right)^{1/2}.$$

Like the Wiener filter, in the absence of accurate estimates for $| F_i |^2 / | N_i |^2$, we approximate the PSE filter by

$$| Q_i | \simeq \left(\frac{1}{| P_i |^2 + \Gamma} \right)^{1/2}$$

where

$$\Gamma = \frac{1}{(\text{SNR})^2}.$$

Note that the criterion used to derive this filter can be written in the form

$$\sum_i (| F_i |^2 - | \hat{F}_i |^2) = 0$$

or using Parseval's theorem

$$\sum_i (| f_i |^2 - | \hat{f}_i |^2) = 0$$

which should be compared to that for the Wiener filter, i.e.

$$\text{minimise} \quad \sum_i | f_i - \hat{f}_i |^2 .$$

14.5 The Matched Filter

The matched filter is a result of finding a solution to the following problem: Given that

$$s_i = \sum_j p_{i-j} f_j + n_i,$$

find an estimate for the Impulse Response Function (IRF) given by

$$\hat{f}_i = \sum_j q_j s_{i-j}$$

where

$$r = \frac{\mid \sum_i Q_i P_i \mid^2}{\sum_i \mid N_i \mid^2 \mid Q_i \mid^2}$$

is a maximum. The ratio defining r is a measure of the signal-to-noise ratio. In this sense, the matched filter maximizes the signal-to-noise ratio of the output. Assuming that the noise n_i has a 'white' or uniform power spectrum, the filter Q_i which maximizes the SNR defined by r is given by

$$Q_i = P_i^*$$

and the required solution is therefore

$$\hat{f}_i = \text{IDFT}(P_i^* S_i).$$

Using the correlation theorem, we then have

$$\hat{f}_i = \sum_j p_{j-i} s_j.$$

The matched filter is therefore based on correlating the signal s_i with the IRF p_i. This filter is frequently used in systems that employ linear frequency modulated (FM) pulses - 'chirped pulses' - which will be discussed later.

14.5.1 Derivation of the Matched Filter

With the problem specified as above, the matched filter is essentially a 'by-product' of the 'Schwarz inequality', i.e.

$$\left| \sum_i Q_i P_i \right|^2 \leq \sum_i \mid Q_i \mid^2 \sum_i \mid P_i \mid^2$$

as discussed in Chapter 8. The principal trick is to write

$$Q_i P_i = \mid N_i \mid Q_i \times \frac{P_i}{\mid N_i \mid}$$

so that the above inequality becomes

$$\left|\sum_i Q_i P_i\right|^2 = \left|\sum_i \mid N_i \mid Q_i \frac{P_i}{\mid N_i \mid}\right|^2 \leq \sum_i \mid N_i \mid^2 \mid Q_i \mid^2 \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

From this result, using the definition of r given above, we see that

$$r \leq \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

Now, if r is to be a maximum, then we want

$$r = \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}$$

or

$$\left|\sum_i \mid N_i \mid Q_i \frac{P_i}{\mid N_i \mid}\right|^2 = \sum_i \mid N_i \mid^2 \mid Q_i \mid^2 \sum_i \frac{\mid P_i \mid^2}{\mid N_i \mid^2}.$$

But this is only true if

$$\mid N_i \mid Q_i = \frac{P_i^*}{\mid N_i \mid}$$

and hence, r is a maximum when

$$Q_i = \frac{P_i^*}{\mid N_i \mid^2}.$$

14.5.2 White Noise Condition

If the noise n_i is white noise, then its power spectrum $\mid N_i \mid^2$ is uniformly distributed. In particular, under the condition

$$\mid N_i \mid^2 = 1 \quad \forall i = 0, 1, ..., N-1$$

then

$$Q_i = P_i^*.$$

14.5.3 FFT Algorithm for the Matched Filter

Using pseudo code, the algorithm for the matched filter is

```
for i=1 to n; do:
   sr(i)=signal(i)
   si(i)=0.
   pr(i)=IRF(i)
   pi(i)=0.
enddo

      forward_fft(sr,si)
```

```
      forward_fft(pr,pi)

for i=1 to n; do:
   fr(i)=pr(i)*sr(i)+pi(i)*si(i)
   fi(i)=pr(i)*si(i)-pi(i)*sr(i)
enddo
      inverse_fft(fr,fi)

for i=1 to n; do:
    hatf(i)=fr(i)
enddo
```

14.5.4 Deconvolution of Frequency Modulated Signals

The matched filter is frequently used in systems that utilize linear frequency modulated (FM) pulses. IRF's of this type are known as chirped pulses. Examples of where this particular type of pulse is used include real and synthetic aperture radar, active sonar and some forms of seismic prospecting for example. Interestingly, some mammals (dolphins, whales and bats for example) use frequency modulation for communication and detection. The reason for this is the unique properties that FM IRFs provide in terms of the quality of extracting information from signals with very low signal-to-noise ratios and the simplicity of the process that is required to do this (i.e. correlation). The invention and use of FM IRFs for man made communications and imaging systems dates back to the early 1960s (the application of FM to radar for example); mother nature appears to have 'discovered' the idea some time ago.

Linear FM Pulses

The linear FM pulse is given (in complex form) by

$$p(t) = \exp(-i\alpha t^2), \quad \mid t \mid \leq T/2$$

where α is a constant and T is the length of the pulse. The phase of this pulse is αt^2 and the instantaneous frequency is given by

$$\frac{d}{dt}(\alpha t^2) = 2\alpha t$$

which varies linearly with t. Hence, the frequency modulations are linear which is why the pulse is referred to as a linear FM pulse. In this case, the signal that is recorded is given by (neglecting additive noise)

$$s(t) = \exp(-i\alpha t^2) \otimes f(t).$$

Matched filtering, we have

$$\hat{f}(t) = \exp(i\alpha t^2) \odot \exp(-i\alpha t^2) \otimes f(t).$$

Evaluating the correlation integral,

$$\exp(i\alpha t^2) \odot \exp(-i\alpha t^2) = \int_{-T/2}^{T/2} \exp[i\alpha(t+\tau)^2] \exp(-i\alpha\tau^2) d\tau$$

$$= \exp(i\alpha t^2) \int_{-T/2}^{T/2} \exp(2i\alpha\tau t) d\tau$$

and computing the integral over τ, we have

$$\exp(i\alpha t^2) \odot \exp(-i\alpha t^2) = T \exp(i\alpha t^2) \operatorname{sinc}(\alpha T t)$$

and hence

$$\hat{f}(t) = T \exp(i\alpha t^2) \operatorname{sinc}(\alpha T t) \otimes f(t).$$

In some systems, the length of the linear FM pulse is relatively long. In such cases,

$$\cos(\alpha t^2) \operatorname{sinc}(\alpha T t) \simeq \operatorname{sinc}(\alpha T t)$$

and

$$\sin(\alpha t^2) \operatorname{sinc}(\alpha T t) \simeq 0$$

and so

$$\hat{f}(t) \simeq T \operatorname{sinc}(\alpha T t) \otimes f(t).$$

Now, in Fourier space, this last equation can be written as

$$\hat{F}(\omega) = \begin{cases} \frac{\pi}{\alpha} F(\omega), & \mid \omega \mid \leq \alpha T; \\ 0, & \text{otherwise.} \end{cases}$$

The estimate $\hat{f}$ is therefore a band limited estimate of f whose bandwidth is determined by the product of the chirping parameter α with the length of the pulse T. An example of the matched filter in action is given in Figure 14.2 obtained using the MATLAB code given below. Here, two spikes have been convolved with a linear FM chirp whose width or pulse length T is significantly greater than that of the input signal. The output signal has been generated using an SNR of 1 and it is remarkable that such an excellent restoration of the input is recovered using a relatively simple operation for processing data that has been so badly distorted by additive noise. The remarkable ability for the matched filter to accurately recover information from linear FM type signals with very low SNRs leads naturally to consider its use for covert information embedding. This is the subject of the case study that follows which investigates the use of chirp coding for covertly watermarking digital signals for the purpose of signal authentication.

```
function MATCH(T,snr)

%Input:
%      T - width of chirp IRF
%      snr - signa-to-noise ratio of signal
%
n=512;    %Set size of array (arbitrary)
nn=1+n/2; %Set mid point of array

%Compute input function (two spikes of width m centered
```

```
%at the mid point of the array.
m=10; %Set width of the spikes (arbitrary)
for i=1:n
   f(i)=0.0; %Initialize input
   p(i)=0.0; %Initialize IRF
end
   f(nn-m)=1.0;
   f(nn+m)=1.0;

%Plot result
figure(1);
subplot(2,2,1), plot(f);

%Compute the (real) IRF, i.e. the linear FM chirp using a
%sine function. (N.B. Could also use a cosine function.)
m=T/2;
k=1;
for i=1:m
   p(nn-m+i)=sin(2*pi*(k-1)*(k-1)/n);
   k=k+1;
end

%Plot result
subplot(2,2,2), plot(p);

%Convolve f with p using the convolution theorem and normalize to unity.
f=fft(f); p=fft(p);
          f=p.*f;
          f=ifft(f); f=fftshift(f); f=real(f);
f=f./max(f); %N.B. No check on case when f=0.

%Compute random Gaussian noise field and normalize to unity.
noise=randn(1,n);
noise=noise./max(noise);

%Compute signal with signal-to-noise ratio defined by snr.
s=f+noise./snr;

%Plot result
subplot(2,2,3), plot(s);

%Restore signal using Matched filter.

%Transform to Fourier space.
s=fft(s);

%Compute Matched filter.
```

```
rest=conj(p).*s;
rest=ifft(rest); rest=fftshift(rest); rest=real(rest);

%Plot result
subplot(2,2,4), plot(rest);
```

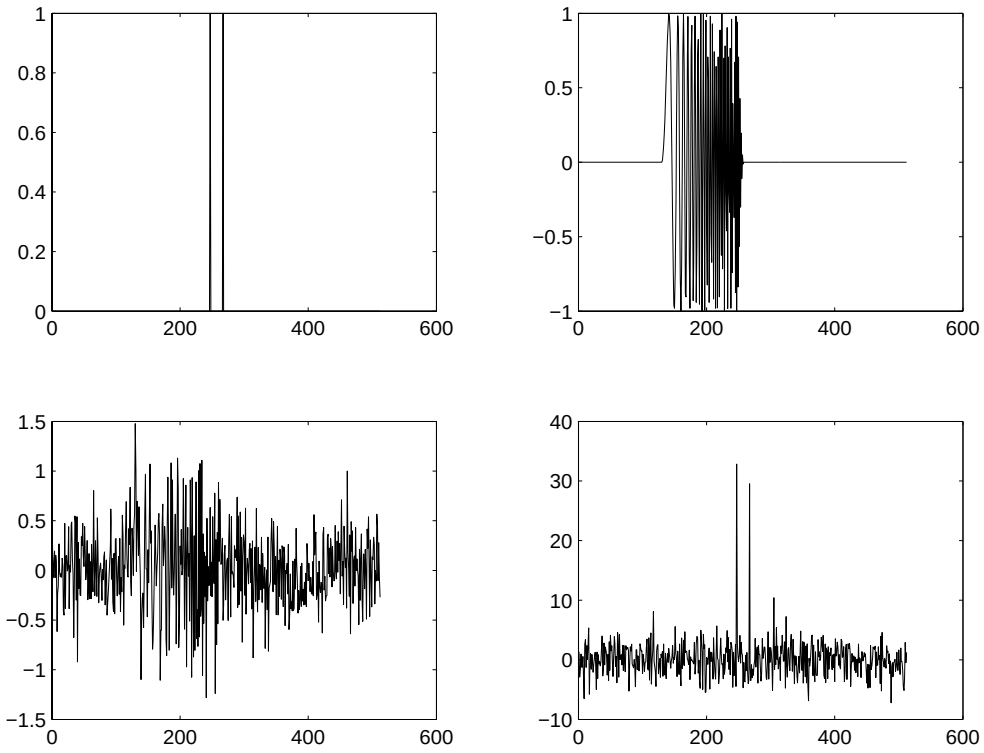


Figure 14.2: Example of a matched filter in action (bottom right) by recovering information from a noisy signal (bottom left) generated by the convolution of an input consisting of two spikes (top left) with a linear FM chirp IRF (top right). The simulation and restoration of the signal given in this example is accomplished using the MATLAB function MATCH(256,1).

14.6 Case Study: Watermarking using Chirp Coding

In this case study[1], we discusses a new approach to 'watermarking' digital signals using linear frequency modulated 'chirp coding'. The principle underlying this approach is

[1]Based on research undertaken by the author with collaboration from Dr O Farooq and Dr S Datta, Applied Signal Processing Group, Loughborough University.

based on the use of a matched filter to provide a reconstruction of a chirped code that is uniquely robust, i.e. in the case of very low signal-to-noise ratios.

Chirp coding for authenticating data is generic in the sense that it can be used for a range of data types and applications (the authentication of speech and audio signals for example). The theoretical and computational aspects of the matched filter and the properties of a chirp are briefly revisited to provide the essential background to the method. Signal code generating schemes are then addressed and details of the coding and decoding techniques considered.

14.6.1 Introduction

Methods of watermarking digital data have applications in a wide range of areas. Digital watermarking of images has been researched for many years in order to achieve methods which provide both anti-counterfeiting and authentication facilities. One of the principle equations that underpins this technology is based on the 'fundamental model' for a signal which is given by

$$s = \hat{P}f + n$$

where f is the information content for the signal (the watermark), $\hat{P}$ is some linear operator, n is the noise and s is the output signal. This equation is usually taken to describe a stationary process in which the noise n is characterized by stationary statistics (i.e. the probability density function of n is invariant of time). In the field of cryptology, the operation $\hat{P}f$ is referred to as the processes of 'diffusion' and the process of adding noise (i.e. $\hat{P}f + n$) is referred to as the process of 'confusion'. In cryptography and steganography (the process of hiding secret information in images) the principal 'art' is to develop methods in which the processes of diffusion and confusion are maximized, an important criterion being that the output s should be dominated by the noise n which in turn should ideally be characterized by a maximum entropy[2] (a uniform statistical distribution).

Digital watermarking and steganography can be considered to form part of the same field of study, namely, cryptology. Being able to recover f from s provides a way of authenticating the signal. If, in addition, it is possible to determine that a copy of s has been made leading to some form of data degradation and/or corruption that can be conveyed through an appropriate analysis of f, then a scheme can be developed that provides a check on: (i) the authenticity of the data s; (ii) its fidelity.

Formally, the recovery of f from s is based on the inverse process

$$f = \hat{P}^{-1}(s - n)$$

where $\hat{P}^{-1}$ is the inverse operator. Clearly, this requires the field n to be known *a priori*. If this field has been generated by a pseudo random number generator for example, then the seed used to generate this field must be known *a priori* in order to recover the data f. In this case, the seed represents the private key required to recover f. However, in principle, n can be any field that is considered appropriate for confusing the information $\hat{P}f$ including a pre-selected signal. Further, if the process of confusion is undertaken in which the signal-to-noise ratio is set to be very low (i.e.

[2] A measure of the lack of information on the exact state of a system - see Chapter 15

$\|n\| >> \|\hat{P}f\|$), then the watermark f can be hidden covertly in the data n provided the inverse process $\hat{P}^{-1}$ is well defined and computationally stable. In this case, it is clear that the host signal n must be known in order to recover the watermark f leading to a private watermarking scheme in which the field n represents a key. This field can of course be (lossless) compressed and encrypted as required. In addition, the operator $\hat{P}$ (and its inverse $\hat{P}^{-1}$) can be key dependent. The value of this operator key dependency relies on the nature and properties of the operator that is used and whether it is compounded in an algorithm that is required to be in the public domain for example.

Another approach is to consider the case in which the field n is unknown and to consider the problem of extracting the watermark f in the absence of this field. In this case, the reconstruction is based on the result

$$f = \hat{P}^{-1}s + m$$

where

$$m = -\hat{P}^{-1}n.$$

Now, if a process $\hat{P}$ is available in which $\|\hat{P}^{-1}s\| >> \|m\|$, then an approximate (noisy) reconstruction of f can be obtained in which the noise m is determined by the original signal-to-noise ratio of the data s and hence, the level of covertness of the diffused watermark $\hat{P}f$. In this case, it may be possible to post-process the reconstruction (de-noising for example) and recover a relatively high-fidelity version of the watermark, i.e.

$$f \sim \hat{P}^{-1}s.$$

This approach (if available) does not rely on a private key (assuming $\hat{P}$ is not key dependent). The ability to recover the watermark only requires knowledge of the operator $\hat{P}$ (and its inverse) and post-processing options as required. The problem here is to find an operator that is able to recover the watermark effectively in the presence of the field n. Ideally, we require an operator $\hat{P}$ with properties such that $\hat{P}^{-1}n \to 0$.

In this application, the operator is based on a chirp function, specifically, a linear Frequency Modulated (FM) chirp of the (complex) type $\exp(-i\alpha t^2)$ where α is the chirp parameter and t is the independent variable. This function is then convolved with f. The inverse process is undertaken by correlating with the (complex) conjugate of the chirp $\exp(i\alpha t^2)$. This provides a reconstruction for f in the presence of the field n that is accurate and robust with very low signal-to-noise ratios. Further, we consider a watermark based on a coding scheme in which the field n is the input. The watermark f is therefore n-dependent. This allows an authentication scheme to be developed in which the watermark is generated from the field in which it is to be hidden. Authentication of the watermarked data is then based on comparing the code generated from $s = \hat{P}f + n$ and that reconstructed by processing s when $\|\hat{P}f\| << \|n\|$. This is an example of a self-generated coding scheme which avoids the use, distribution and application of reference codes. Here, the coding scheme is based on the application of Daubechies wavelets. There are numerous applications of this technique in areas such as telecommunications and speech recognition where authentication is mandatory. For example, the method can readily be applied to audio data with no detectable differences in the audio quality of the data. The watermark

code is able to be recovered accurately and changes relatively significantly if the data is distorted through cropping, filtering, noise or a compression system for example. Thus, it provides a way making a signal tamper proof.

14.6.2 Matched Filter Reconstruction

Given that

$$s(t) = \exp(-i\alpha t^2) \otimes f(t) + n(t),$$

after matched filtering, we obtain the estimate

$$\hat{f}(t) \simeq T\,\mathrm{sinc}(\alpha T t) \otimes f(t) + \exp(i\alpha t^2) \odot n(t).$$

The correlation function produced by the correlation of $\exp(i\alpha t)$ with $n(t)$ will in general be relatively low in amplitude since $n(t)$ will not normally have features that match those of a chirp. Thus, it is reasonable to assume that

$$\|T\,\mathrm{sinc}(\alpha T t) \otimes f(t)\| >> \|\exp(i\alpha t^2) \odot n(t)\|$$

and that in practice, $\hat{f}$ is a band-limited reconstruction of f with high SNR. Thus, the process of using chirp signals with matched filtering for the purpose of reconstructing the input in the presence of additive noise provides a relatively simple and computationally reliable method of 'diffusing' and reconstructing information encoded in the input function f. This is the underlying principle behind the method of watermarking described here.

14.6.3 The Fresnel Transform

Ignoring scaling, we can define the Fresnel transform as

$$s(x, y) = \exp[-i\alpha(x^2 + y^2)] \otimes\otimes f(x, y).$$

This result is just a 2D version of the 'chirp transform' discussed earlier. The reconstruction of f from s follows the same principles and can be accomplished using a correlation of s with the function $\exp[i\alpha(x^2 + y^2)]$. This result leads directly to a method of digital image watermarking using the Fresnel transform to 'diffuse' the watermark f. In particular, reverting to the operator notation used previously, our Fresnel transform based watermarking model becomes

$$s(x, y) = \hat{P} f(x, y) + n(x, y)$$

where the operator $\hat{P}$ is given by

$$\hat{P} = \exp[-i\alpha(x^2 + y^2)] \otimes\otimes$$

and the inverse operator is given by

$$\hat{P}^{-1} = \exp[i\alpha(x^2 + y^2)] \odot\odot \quad .$$

Note that $\otimes\otimes$ denotes 2D convolution and $\odot\odot$ denotes 2D correlation. Also, in practice, only values ≥ 0 can be used for application to digital images so that we must consider a function of the normalized form $(1 + \exp[i\alpha(x^2 + y^2)])/2$ for example.

A covert watermarking procedure involves the addition of a (diffused) watermark to a host image with a very low watermark-to-signal ratio, i.e.

$$\|\hat{P}f(x,y)\| << \|n(x,y)\|.$$

Recovery of the watermark is then based on the result

$$f(x,y) = \hat{P}^{-1}[s(x,y) - n(x,y)].$$

14.6.4 Chirp Coding, Decoding and Watermarking

We now return to the issue of watermarking using chirp functions. The basic model for the watermarked signal (which is real) is

$$s(t) = \text{chirp}(t) \otimes f(t) + n(t)$$

where

$$\text{chirp}(t) = \sin(\alpha t^2).$$

We consider the field $n(t)$ to be some pre-defined signal to which a watermark is to be 'added' to generate $s(t)$. In principle, any watermark described by the function $f(t)$ can be used. On the other hand, for the purpose of authentication, we require two criterion: (i) $f(t)$ should represent a code which can be reconstructed accurately and robustly; (ii) the watermark code should be sensitive (and ideally ultra-sensitive) to any degradation in the field $n(t)$ due to lossy compression, cropping or highpass and lowpass filtering for example. To satisfy condition (i), it is reasonable to consider $f(t)$ to represent a bit stream, i.e. to consider the discretized version of $f(t)$ - the vector f_i - to be composed of a set of elements with values 0 or 1 and only 0 or 1. This binary code can of course be based on a key or set of keys which, when reconstructed, is compared to the key(s) for the purpose of authenticating the data. However, this requires the distribution of such keys (public and/or private). Instead, we consider the case where a binary sequence is generated from the field $n(t)$. There are a number of approaches that can be considered based on the spectral characteristics of $n(t)$ for example. These are discussed later on, in which binary sequences are produced from the application of wavelet decomposition.

Chirp Coding

Given that a binary sequence has been generated from $n(t)$, we now consider the method of chirp coding. The purpose of chirp coding is to 'diffuse' each bit over a range of compact support T. However, it is necessary to differentiate between 0 and 1 in the sequences. The simplest way to achieve this is to change the polarity of the chirp. Thus, for 1 we apply the chirp $\sin(\alpha t^2)$, $t \in T$ and for 0 we apply the chirp -$\sin(\alpha t^2)$, $t \in T$ where T is the chirp length. The chirps are then concatenated to produce a contiguous stream of data, i.e. a signal composed of $\pm$chirps. Thus, the binary sequence 010 for example is transformed to the signal

$$s(t) = \begin{cases} -\text{chirp}(t), & t \in [0,T); \\ +\text{chirp}(t), & t \in [T,2T); \\ -\text{chirp}(t), & t \in [2T,3T). \end{cases}$$

The period over which the chirp is applied depends on the length of the signal to which the watermark is to be applied and the length of the binary sequence. In the example given above, the length of the signal is taken to be $3T$. In practice, care must be taken over the chirping parameter α that is applied for a period T in order to avoid aliasing and in some cases it is of value to apply a logarithmic sweep instead of a linear sweep. The instantaneous frequency of a logarithmic chirp is given by

$$\psi(t) = \psi_0 + 10^{at}$$

where

$$a = \frac{1}{T}\log_{10}(\psi_1 - psi_0)$$

ψ_0 is the initial frequency and ψ_1 is the final frequency at time T. In this case, the final frequency should be greater than the initial frequency.

Decoding

Decoding or reconstruction of the binary sequence requires the application of a correlator using the function chirp(t), $t \in [0, T)$. This produces a correlation function that is either -1 or +1 depending upon whether $-$chirp(t) or +chirp(t) has been applied respectively. For example, after correlating the chirp coded sequence 010 given above, the correlation function $c(t)$becomes

$$c(t) = \begin{cases} -1, & t \in [0, T); \\ +1, & t \in [T, 2T); \\ -1, & t \in [2T, 3T). \end{cases}$$

from which the original sequence 010 is easily inferred, the change in sign of the correlation function identifying a bit change (from 0 to 1 or from 1 to 0). Note that in practice the correlation function may not be exactly 1 or -1 when reconstruction is undertaken and the binary sequence is effectively recovered by searching the correlation function for changes in sign. The chirp used to recover the watermark must of course have the same parameters (inclusive of its length) as those used to generate the chirp coded sequence. These parameters can be used to define part of a private key.

Watermarking

The watermarking process is based on adding the chirp coded data to the signal $n(t)$. Let the chirp coded signal be given by the function $h(t)$, then the watermarking process is described by the equation

$$s(t) = a\left[\frac{bh(t)}{\|h(t)\|_\infty} + \frac{n(t)}{\|n(t)\|_\infty}\right]$$

and the coefficients $a > 0$ and $0 < b < 1$ determine the amplitude and the SNR of s where

$$a = \|n(t)\|_\infty.$$

The coefficient a is required to provide a watermarked signal whose amplitude is compatible with the original signal n. The value of b is adjusted to provide an output that is acceptable in the application to be considered and to provide a robust reconstruction of the binary sequence by correlating $s(t)$ with $\text{chirp}(t), t \in [0, T)$. To improve the robustness of the reconstruction, the value of b can be increased, but this has to be off-set with regard to the perceptual quality of the output, i.e. the perturbation of n by h should be as small as possible.

14.6.5 Code Generation

In the previous section, the method of chirp coding a binary sequence and watermarking the signal $n(t)$ has been discussed where it is assumed that the sequence is generated from this same signal. In this section, the details of this method are presented. The problem is to convert the salient characteristics of the signal $n(t)$ into a sequence of bits that is relatively short and conveys information on the signal that is unique to its overall properties. In principle, there are a number of ways of undertaking this. For example, in practice, the digital signal n_i, which will normally be composed of an array of floating point numbers, could be expressed in binary form and each element concatenated to form a contiguous bit stream. However, the length of the code (i.e. the total number of bits in the stream) will tend to be large leading to high computational costs in terms of the application of chirp coding/decoding. What is required, is a process that yields a relatively short binary sequence (when compared with the original signal) that reflects the important properties of the signal in its entirety. Two approaches are considered here: (i) power spectral density decomposition and (ii) wavelet decomposition.

Power Spectral Density Decomposition

Let $N(\omega)$ be the Fourier transform $n(t)$ and define the Power Spectrum $P(\omega)$ as

$$P(\omega) = \mid N(\omega) \mid^2 .$$

An important property of the binary sequence is that it should describe the spectral characteristics of the signal in its entirety. Thus, if for example, the binary sequence is based on just the low frequency components of the signal, then any distortion of the high frequencies components will not affect the watermark and the signal will be authenticated. Hence, we consider the case where the power spectrum is decomposed into N components, i.e.

$$\begin{aligned} P_1(\omega) &= P(\omega), \quad \omega \in [0, \Omega_1); \\ P_2(\omega) &= P(\omega), \quad \omega \in [\Omega_1, \Omega_2); \\ &\vdots \\ P_N(\omega) &= P(\omega), \quad \omega \in [\Omega_{N-1}, \Omega_N). \end{aligned}$$

Note that it is assumed that the signal $n(t)$ is band-limited with a bandwidth of Ω_N.

The set of the functions $P_1, P_2, ..., P_N$ now reflect the complete spectral characteristics of the signal $n(t)$. Since each of these functions represents a unique part of the spectrum, we can consider a single measure as an identifier or tag. A natural

measure to consider is the energy which is given by the integral of the functions over their frequency range. In particular, we consider the energy values in terms of their contribution to the spectrum as a percentage, i.e.

$$E_1 = \frac{100}{E}\int_0^{\Omega_1} P_1(\omega)d\omega,$$

$$E_2 = \frac{100}{E}\int_{\Omega_1}^{\Omega_2} P_2(\omega)d\omega,$$

$$\vdots$$

$$E_N = \frac{100}{E}\int_{\Omega_{N-1}}^{\Omega_N} P_N(\omega)d\omega,$$

where

$$E = \int_0^{\Omega_N} P(\omega)d\omega.$$

Code generation is then based on the following steps:

(i) Rounding to the nearest integer the (floating point) values of E_i to decimal integer form:

$$e_i = \text{round}(E_i), \quad \forall i.$$

(ii) Decimal integer to binary string conversion:

$$b_i = \text{binary}(e_i).$$

(iii) Concatenation of the binary string array b_i to a binary sequence:

$$f_j = \text{cat}(b_i).$$

The watermark f_j is then chirp coded as discussed previously.

Wavelet Decomposition

The wavelet transform is discussed in Chapter 5 and is defined by

$$\hat{W}[f(t)] = F_L(t) = \int f(\tau)w_L(t,\tau)d\tau$$

where

$$w_L(t,\tau) = \frac{1}{\sqrt{\mid L \mid}} w\left(\frac{t-\tau}{L}\right).$$

The wavelet transformation is essentially a convolution transform in which $w(t)$ is the convolution kernel but with a factor L introduced. The introduction of this factor

provides dilation and translation properties into the convolution integral (which is now a function of L) that gives it the ability to analyse signals in a multi-resolution role.

The code generating method is based on computing the energies of the wavelet transformation over N levels. Thus, the signal $f(t)$ is decomposed into wavelet space to yield the following set of functions:

$$F_{L_1}(\tau), \quad F_{L_2}(\tau), \quad ... \quad F_{L_N}(\tau).$$

The (percentage) energies of these functions are then computed, i.e.

$$E_1 = \frac{100}{E} \int \mid F_{L_1}(\tau) \mid^2 d\tau,$$

$$E_2 = \frac{100}{E} \int \mid F_{L_1}(\tau) \mid^2 d\tau,$$

$$\vdots$$

$$E_N = \frac{100}{E} \int \mid F_{L_N}(\tau) \mid^2 d\tau,$$

where

$$E = \sum_{i=1}^{N} E_i.$$

The method of computing the binary sequence for chirp coding from these energy values follows that described in the method of power spectral decomposition. Clearly, whether applying the power spectral decomposition method or wavelet decomposition, the computations are undertaken in digital form using a DFT and a DWT (Discrete Wavelet Transform) respectively.

14.6.6 MATLAB Application Programs

Two prototype MATLAB programs have been developed to implement the watermarking method discussed. The *coding process* reads in a named file, applies the watermark to the data using wavelet decomposition and writes out a new file using the same file format. The *Decoding process* reads a named file (assumed to contain the watermark or otherwise), recovers the code from the watermarked data and then recovers the (same or otherwise) code from the watermark. The coding program displays the decimal integer and binary codes for analysis. The decoding program displays the decimal integer streams generated by the wavelet analysis of the input signal and the stream obtained by processing the signal to extract the watermark code or otherwise. This process also provides an error measure based on the result

$$e = \frac{\sum_i \mid x_i - y_i \mid}{\sum_i \mid x_i + y_i \mid}$$

where x_i and y_i are the decimal integer arrays obtained from the input signal and the watermark (or otherwise). In the application considered here, the watermarking

method has been applied to audio (.wav) files in order to test the method on data which requires that the watermark does not affect the fidelity of the output (i.e. audio quality). Only a specified segment of the data is extracted for watermarking which is equivalent to applying and off-set to the data. The segment can be user defined and if required, form the basis for a (private) key system. In this application, the watermarked segment has been 'hard-wired' and represents a public key. The wavelets used are Daubechies wavelets computed using the MATLAB wavelet toolbox. However, in principle, any wavelet can be used for this process and the actual wavelet used yields another feature that can form part of the private key required to extract the watermark.

Coding Process

The coding process is compounded in the following basic steps:

Step 1: Read a .wav file.

Step 2: Extract a section of a single vector of the data (note that a .wav contains stereo data, i.e. two vectors).

Step 3: Apply wavelet decomposition using Daubechies wavelets with 7 levels. Note that in addition to wavelet decomposition, the approximation coefficients for the input signal are computed to provide a measure on the global effect of introducing the watermark into the signal. Thus, 8 decomposition vectors in total are generated.

Step 4: Compute the (percentage) 'energy values'.

Step 5: Round to the nearest integer and convert to binary form.

Step 6: Concatenate both the decimal and binary integer arrays.

Step 7: Chirp code the binary sequence.

Step 8: Scale the output and add to the original input signal.

Step 9: Re-scale the watermarked signal.

Step 10: Write to a file.

In the MATLAB code that follows, the above procedure has been implemented where the parameters for segmenting and processing data of a specific size have been 'hard wired'.

```
%read .wav audio file
[au2,fs,nbit]=wavread('wavefil');

%clear screen
clc

%Set data size (arbitrary) to be watermarked (assumed to be less than
or equal to data in file).
```

```
data_size=1500150;

%Extract single set of data composed of 1500150 (arbitrary) elements.
au1=au2(1:data_size,1);

%Set scaling factor.
div_fac=270;

%Set data segment origin.
data_seg=300031;

%Extract segment of data from data_seg to data_size and
%compute the maximum value.
au=au1(data_seg:data_size,1);
au_max1=max(au1(data_seg:data_size,1));

%Apply wavelet decomposition using Daubechies (4) wavelets with 7 levels.
[ca cl]=wavedec(au(:,1),7,'db4');

%Compute the approximation coefficients at level 7.
appco=appcoef(ca,cl,'db4',7);

%Determine coefficients at each level.
detco7=detcoef(ca,cl,7);
detco6=detcoef(ca,cl,6);
detco5=detcoef(ca,cl,5);
detco4=detcoef(ca,cl,4);
detco3=detcoef(ca,cl,3);
detco2=detcoef(ca,cl,2);
detco1=detcoef(ca,cl,1);

%Compute the energy for each set of coefficients.
ene_appco=sum(appco.^2);
ene_detco7=sum(detco7.^2);
ene_detco6=sum(detco6.^2);
ene_detco5=sum(detco5.^2);
ene_detco4=sum(detco4.^2);
ene_detco3=sum(detco3.^2);
ene_detco2=sum(detco2.^2);
ene_detco1=sum(detco1.^2);

%Compute the total enegy of all the coefficients.
tot_ene=round(ene_detco7+ene_detco6+ene_detco5+ene_detco4...
       +ene_detco3+ene_detco2+ene_detco1);

%Round towards nearest integer the percentage energy of each set.
pene_hp7=round(ene_detco7*100/tot_ene);
```

```
pene_hp6=round(ene_detco6*100/tot_ene);
pene_hp5=round(ene_detco5*100/tot_ene);
pene_hp4=round(ene_detco4*100/tot_ene);
pene_hp3=round(ene_detco3*100/tot_ene);
pene_hp2=round(ene_detco2*100/tot_ene);
pene_hp1=round(ene_detco1*100/tot_ene);

%Do decimal integer to binary conversion with at least 17 bits.
tot_ene_bin=dec2bin(tot_ene,31);
f7=dec2bin(pene_hp7,17);
f6=dec2bin(pene_hp6,17);
f5=dec2bin(pene_hp5,17);
f4=dec2bin(pene_hp4,17);
f3=dec2bin(pene_hp3,17);
f2=dec2bin(pene_hp2,17);
f1=dec2bin(pene_hp1,17);

%Concatenate the arrays f1, f2, ... along dimension 2 to
%produce binary sequence - the watermark wmark.
wmark=cat(2,tot_ene_bin,f7,f6,f5,f4,f3,f2,f1);

%Concatenate decimal integer array.
per_ce=cat(2,tot_ene,pene_hp7,pene_hp6,pene_hp5,...
           pene_hp4,pene_hp3,pene_hp2,pene_hp1);


%Write out decimal integer and binary codes.
d_string=per_ce
b_string=wmark

%Assign -1 to a 0 bit and 1 for 1 bit.
for j=1:150
   if str2num(wmark(j))==0
      x(j)=-1;
   else
      x(j)=1;
   end
end

%Initialize and compute chirp function using a log sweep.
t=0:1/44100:10000/44100;
y=chirp(t,00,10000/44100,100,'log');

%Compute +chirp for 1 and -chirp for 0, scale by div_fac and concatenate.
znew=0;
for j=1:150
   z=x(j)*y/div_fac;
```

```
   znew=cat(2,znew,z);
end

%Compute length of znew and watermark signal.
znew=znew(2:length(znew));
wmark_sig=znew'+au1;

%Compute power of watermark and power of signal.
w_mark_pow=(sum(znew.^2));
sig_pow=(sum(au1.^2));

%Rescale watermarked signal.
wmark_sig1=wmark_sig*au_max1/max(wmark_sig);

%Concatenate and write to file.
wmark_sig=cat(2,wmark_sig1,au2(1:data_size,2));
wavwrite(wmark_sig,fs,nbit,'wm_wavefile');
```

Decoding process

The decoding process is as follows:

Step 1: Steps 1-6 in the coding processes are repeated.

Step 2: Correlate the data with a chirp identical to that used for chirp coding.

Step 3: Extract the binary sequence.

Step 4: Convert from binary to decimal.

Step 5: Display the original and reconstructed decimal sequence.

Step 6: Display the error.

```
%Clear variables and functions from memory.
clear

%Read watermarked file and clear screen.
[au,fs, nbit]=wavread('wm_wavefile');
clc

%Set parameters for data processing.
data_size=1500150;
data_seg=300031;

%Extract data.
au1=au(data_seg:data_size,1);
```

```
%Do wavelet decomposition.
[ca cl]=wavedec(au1,7,'db4');

%Extract wavelet coefficients.
appco=appcoef(ca,cl,'db4',7);
detco7=detcoef(ca,cl,7);
detco6=detcoef(ca,cl,6);
detco5=detcoef(ca,cl,5);
detco4=detcoef(ca,cl,4);
detco3=detcoef(ca,cl,3);
detco2=detcoef(ca,cl,2);
detco1=detcoef(ca,cl,1);

%Compute energy of wavelet coefficients.
ene_appco=sum(appco.^2);
ene_detco7=sum(detco7.^2);
ene_detco6=sum(detco6.^2);
ene_detco5=sum(detco5.^2);
ene_detco4=sum(detco4.^2);
ene_detco3=sum(detco3.^2);
ene_detco2=sum(detco2.^2);
ene_detco1=sum(detco1.^2);

%Compute total energy factor.
tot_ene=round(ene_detco7+ene_detco6+ene_detco5+ene_detco4...
   +ene_detco3+ene_detco2+ene_detco1);


%Express energy values as percentage of total.
%energy and round to nearest integer.
pene_hp7=round(ene_detco7*100/tot_ene);
pene_hp6=round(ene_detco6*100/tot_ene);
pene_hp5=round(ene_detco5*100/tot_ene);
pene_hp4=round(ene_detco4*100/tot_ene);
pene_hp3=round(ene_detco3*100/tot_ene);
pene_hp2=round(ene_detco2*100/tot_ene);
pene_hp1=round(ene_detco1*100/tot_ene);
per_ene=cat(2,tot_ene,pene_hp7,pene_hp6,pene_hp5,...
            pene_hp4,pene_hp3,pene_hp2,pene_hp1);

%Output original decimal integer code obtained from
%signal via wavelet decomposition.
original_d_string=per_ene;
original_d_string
orig=original_d_string;

%Compute chirp function.
```

```
t=0:1/44100:10000/44100;
y=chirp(t,00,10000/44100,100,'log');

%Correlate input signal with chirp and recover sign.
for i=1:150
   yzcorr=xcorr(au(10000*(i-1)+1:10000*i),y,0);

   r(i)=sign(yzcorr);
end

%Recover bit stream.
 for i=1:150
   if r(i)==-1
      recov(i)=0;
   else
      recov(i)=1;
   end
end

%Convert from number to sring.
recov=(num2str(recov,-8));

%Covert from binary to decimal and concatenate.
rec_ene_dist=cat(2,bin2dec(recov(1:31)),bin2dec(recov(32:48)),...
bin2dec(recov(49:65)),bin2dec(recov(66:82)),bin2dec(recov(83:99)),...
bin2dec(recov(100:116)),bin2dec(recov(117:133)),bin2dec(recov(134:150)));

%Write out reconstructed decimal integer stream recoverd from watermark.
reconstructed_d_string=rec_ene_dist;
reconstructed_d_string
rec=reconstructed_d_string;

%Write out error between reconsructed and original watermark code.
error=sum(abs(rec-orig))/sum(abs(rec+orig))
```

14.6.7 Discussion

In a practical application of this method for authenticating audio files, for example, a threshold can be applied to the error value. If and only if the error lies below this threshold is the data taken to be authentic.

The prototype MATLAB programs provided have been developed to explore the applications of the method for different signals and systems of interest to the user. Note that in the decoding program, the correlation process is carried out using a spatial cross-correlation scheme (using the MATLAB function *xcorr*), i.e. the watermark is recovered using the process $\text{chirp}(t) \odot s(t)$ instead of the Fourier equivalent $\text{CHIRP}^*(\omega)S(\omega)$ where CHIRP and S are the Fourier transforms of chirp and s respectively (in digital form of course). This is due to the fact that the 'length' of the

chirp function is significantly less than that of the signal. Application of a spatial correlator therefore provides greater computational efficiency.

The method of digital watermarking discussed here makes specific use of the chirp function. This function is unique in terms of its properties for reconstructing information (via application of the Matched Filter) that has been 'diffused' through the convolution process, i.e. the watermark extracted is, in theory, an exact band-limited version of the original watermark as defined in the presence of significant additive noise, in this case, the signal into which the watermark is 'embedded'. The method has a close relationship with the Fresnel transform and can be used for digital image watermarking in an entirely equivalent way. The approach considered here allows a code to be generated directly from the input signal and that same code used to watermark the signal. The code used to watermark the signal is therefore self-generating. Reconstruction of the code only requires a correlation process with the watermarked signal to be undertaken. This means that the signal can be authenticated without access to an external reference code. In other words, the method can be seen as a way of authenticating data by extracting a code (the watermark) within a code (the signal).

Audio data watermarking schemes rely on the imperfections of the human audio system. They exploit the fact that the human auditory system is insensitive to small amplitude changes, either in the time or frequency domains, as well as insertion of low amplitude time domain echo's. Spread spectrum techniques augment a low amplitude spreading sequence which can be detected via correlation techniques. Usually, embedding is performed in high amplitude portions of the signal, either in the time or frequency domains. A common pitfall for both types of watermarking systems is their intolerance to detector de-synchronization and deficiency of adequate methods to address this problem during the decoding process. Although other applications are possible, chirp coding provides a new and novel technique for fragile audio watermarking. In this case, the watermarked signal does not change the perceptual quality of the signal. In order to make the watermark inaudible, the chirp generated is of very low frequency and amplitude. Using audio files with sampling frequencies of over 1000Hz, a logarithmic chirp can be generated in the frequency band of 1-100Hz. Since the human ear has low sensitivity in this band, the embedded watermark will not be perceptible. Depending upon the band and amplitude of the chirp, the signal-to-watermark ratio can be in excess of 40dB. Various forms of attack can be applied which change the distribution of the percentage sub-band energies originally present in the signal including filtering (both low pass and high pass), cropping and lossy compression (MP3 compression) with both constant and variable bit rates. In each case, the signal and/or the watermark is distorted enough to register the fact that the data has been tampered with. Further, chirp based watermarks are difficult to remove from the signal since the initial and the final frequency is at the discretion of the user and its position in the data stream can be varied through application of an offset, all such parameters being combined to form a private key.

14.7 Constrained Deconvolution

Constrained deconvolution provides a filter which gives the user additional control over the deconvolution process. This method is based on minimising a linear operation on

the object f_i of the form $g_i \otimes f_i$ subject to some other constraint. Using the least squares approach, we find an estimate for f_i by minimizing $\|g_i \otimes f_i\|^2$ subject to the constraint

$$\|s_i - p_i \otimes f_i\|^2 = \|n_i\|^2.$$

Using this result, we can write

$$\|g_i \otimes f_i\|^2 = \|g_i \otimes f_i\|^2 + \lambda(\|s_i - p_i \otimes f_i\|^2 - \|n_i\|^2)$$

because the quantity inside the brackets on the right hand side is zero. The constant λ is called the Lagrange multiplier. Using the orthogonality principle, $\|g_i \otimes f_i\|^2$ is a minimum when

$$(g_i \otimes f_i) \odot g_i^* - \lambda(s_i - p_i \otimes f_i) \odot p_i^* = 0.$$

In Fourier space, this equation becomes

$$\mid G_i \mid^2 F_i - \lambda(S_i P_i^* - \mid P_i \mid^2 F_i) = 0$$

and solving for F_i, we get

$$F_i = \frac{S_i P_i^*}{\mid P_i \mid^2 + \gamma \mid G_i \mid^2}$$

where γ is the reciprocal of the Lagrange multiplier ($= 1/\lambda$). Hence, the constrained least squares filter is given by

$$\frac{P_i^*}{\mid P_i \mid^2 + \gamma \mid G_i \mid^2}.$$

The constrained deconvolution filter allows the user to change G to suite a particular application. This filter can be thought of as a generalization of the other filters; thus, if $\gamma = 0$ then the inverse filter is obtained; if $\gamma = 1$ and $\mid G_i \mid^2 = \mid N_i \mid^2 / \mid F_i \mid^2$ then the Wiener is obtained and if $\gamma = 1$ and $\mid G_i \mid^2 = \mid N_i \mid^2 - \mid P_i \mid^2$ then the matched filter is obtained.

14.8 Homomorphic Filtering

The homomorphic filter employs the properties of the logarithm to write the equation

$$S_i = P_i F_i$$

in the form

$$\ln S_i = \ln P_i + \ln F_i.$$

In this case, the object function f_i can be recovered using the result

$$f_i = \text{IDFT}[\exp(\ln S_i - \ln P_i)].$$

This type of operation is known as homomorphic filtering. In practice, deconvolution by homomorphic processing replaces the problems associated with computing the inverse filter $1/P_i$ with computing the logarithm of a complex function (i.e. computing

the functions $\ln S_i$ and $\ln P_i$). By writing the complex spectra S_i and P_i in terms of their amplitude and phase spectra, we get

$$S_i = A_i^S \exp(i\theta_i^S)$$

and

$$P_i = A_i^P \exp(i\theta_i^P)$$

where A_i^S and A_i^P are the amplitude spectra of S_i and P_i respectively and θ_i^S and θ_i^P are the phase spectra of S_i and P_i respectively. Using these results, we can write

$$f_i = \mathrm{Re}\{\mathrm{IDFT}[\exp(\ln A_i^S - \ln A_i^P)\cos(\theta_i^S - \theta_i^P) + i\exp(\ln A_i^S - \ln A_i^P)\sin(\theta_i^S - \theta_i^P)]\}$$

Homomorphic filtering occurs very naturally in the processing of speech signals and it is closely related to the interpretation of such signals via the Cepstrum as discussed in Chapter 5 (see Cepstral transform).

14.9 Noise

The term noise $n(t)$ has been mentioned in this and some previous chapters without a proper discussion of its physical nature or how to simulate it. This is now addressed.

Noise plays a central role in all aspects of signal processing and the simulation of noise and noisy signals has a long an important history. The simulation of noise is vitally important in testing the robustness of a digital signal processing algorithm especially when the algorithm is based on some method that has been designed to overcome the ill-conditioned nature of a 'naive approach'. The study of noise and methods of generating it are given at this point because in this chapter, and some of the chapters that follow, the addition of noise to data is required in a number of the programming exercises that are set in order to study the robustness or otherwise of a given algorithm. The noise term is usually an additive one and the noisy signal is typically modelled as

$$s(t) = p(t) \otimes f(t) + n(t).$$

Now, s is the combination of two terms, one deterministic (i.e. $p \otimes f$) and one stochastic (i.e. n). This is the basic noise model for a signal which is the most common type. However, it is worth stating that is some cases, the noise can be multiplicative instead of additive or could even be a combination of both. From here on, the noise model considered is additive alone. The noise of a signal is a combination of effects due to a whole range of unwanted disturbances and interference. In practice, noise is multifaceted and very much application dependent and models are therefore constructed which are based on the design of a suitable probability density function for the noise which is statistically compatible with the data. In general, noise accounts for all the non-ideal effects that may occur in a system. Typically, in order to measure the noise of a system (assumed to be a time invariant linear system), data can be collected on the output of the system in the case when there is no input. From the measured data, a histogram can be generated and its statistical properties measured and modelled as required. These statistical parameters can then be used as required for processing the signal generated by an input using, for example, Bayesian estimation methods which are discussed in Chapter 16.

14.10 Noise Types

In the equation

$$s(t) = p(t) \otimes f(t) + n(t),$$

the addition of the stochastic term n as representing all non-deterministic effects in an otherwise ideal system is extremely naive. Depending on the specific application, this term can also represent the error associated with the approximation of a process to the form $p \otimes f$ in addition to the inevitable electronic noise types and 'jitter' associated with the process of data acquisition. An illustration of this is discussed below.

14.10.1 Multiple Scattering Effects

In active, pulse-echo type systems, the term $p \otimes f$ is taken to represent the first order reflection or back-scattering components where it is assumed that multiple scattering, which is sometimes referred to as reverberation, is insignificant. However, in some cases, the high order components can be significant but in an additive context. To show this, consider the wave equation

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u(x,t) = -k^2 f(x) u(x,k)$$

where u is the wavefield, $f(x)$ describes the inhomogeneity associated with a layered medium and $k = \omega/c_0$ where ω is the angular frequency and c_0 is the (constant) wavespeed (see the Case Study given in Chapter 4 for an example of the derivation of this result and the Green's function solution that follows). The Green's function solution to this equation at a point x_0 along x is given by

$$u(x_0,k) = u_0(x_0,k) + k^2 \int g(x \mid x_0,k) f(x) u(x,k) dx$$

where

$$g(x \mid x_0,k) = \frac{i}{2k} \exp(ik \mid x - x_0 \mid).$$

Now, the first order solution to this equation is given by

$$u(x_0,k) = u_0(x_0,k) + k^2 \int g(x \mid x_0,k) f(x) u_0(x,k) dx$$

where u_0 is the solution to

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) u_0(x,k) = 0$$

namely, the incident field $u(x,k) = P(k)\exp(ikx)$ where P is the spectrum of an emitted pulse. This approximation, i.e. the Born approximation, leads directly to the asymptotic result in which the reflected signal generated by a pulse of $p(t)$ at normal incidence to the layer media is given by $p(t) \otimes f(t)$ where

$$f(t) = \frac{1}{2}\frac{d}{dt} f(\tau/2).$$

However, this approximation can be considered to be just the first order solution of the iterative process

$$u_{n+1}(x_0,k) = u_0(x_0,k) + k^2 \int g(x \mid x_0,k) f(x) u_n(x,k) dx.$$

If we now write this equation in the form

$$u_{n+1}(x,k) = u_0(x_0,k) + k^2 g(\mid x \mid,k) \otimes f(x) u_n(x,k)$$

then we have

$$u_{n+1} = u_0 + k^2 g \otimes f u_0 + k^4 g \otimes f(g \otimes f u_0) + k^6 g \otimes f(g \otimes f(g \otimes f u_0)) + ...$$
$$... + k^{2n} g \otimes f(g \otimes f(g \otimes f(...(g \otimes f u_0)))).$$

Physically, this equation describes the wavefield in terms of the sum of the incident field and the first order scattered field (single scattering) and then the second order scattered field (double scattering) and so on. If we now collect all the terms of order k^4 and above, then the reflected field (to n^{th} order is given by)

$$r(x,k) = u_{n+1}(x,k) = k^2 g \otimes f u_0 + n(x,k)$$

where n is given by

$$n(x,k) = k^4 g \otimes f(g \otimes f u_0) + k^6 g \otimes f(g \otimes f(g \otimes f u_0)) + ...$$
$$... + k^{2n} g \otimes f(g \otimes f(g \otimes f(...(g \otimes f u_0)))).$$

Observe that the reflected field is given by the sum of two terms. The first term provides us with the classic convolution model for a signal. The second term has an analogy with the additive noise term that has so far been introduced in a rather arbitrary or phenomenological manner. However, in this case, this 'noise term' is the result of a physical process which represents the reverberation of the wavefield due to multiple scattering events under the condition that

$$\| k^2 g \otimes f u_0 + n \| < \| u_0 \|$$

where

$$\| f(x) \| \equiv \left(\int \mid f(x) \mid^2 dx \right)^{\frac{1}{2}}$$

which provides the condition for convergence of the solution. In many physical systems, the higher order scattering events may be relatively weak or else an experiment can be made which enhances the first order effect, by combining a number of recordings of the same interaction for example and adding the data together. Nevertheless, the principle point to be emphasized here is that in defining a model of the type

$$s(t) = p(t) \otimes f(t) + n(t)$$

for a signal s obtained from some pulse-echo type system, the term $p \otimes f$ is a description of a physical process in which weak interactions occur, and that in addition to electronic noise, the term n can include physical effects due to multiple scattering.

14.10.2 Beam Profile Effects

In the previous section, we have seen that in the term $p \otimes f$, the impulse response function p evolves from considering the incidence of a pulse with spectrum $P(\omega)$. However, it is also assumed that this pulse travels along a pencil line beam, i.e. the incident field is given by $P(k)\exp(ikx)$. In practice, pencil line beams are very rare. The emission of any radiation field from a single point is invariably accompanied by a divergence of that field and a pencil line beam approximation is very inadequate. Moreover, some types of emissions are characterized by field patterns that are the result of diffraction effects leading to 'side lobes'. Thus, features may occur in a signal that are the result of the interaction of a part of the incident wavefield that has diverged due to the presence of side lobes, a physical process that is certainly not included in a one-dimensional model. In general, the application of a one-dimensional model for computing a scattered wavefield is very limited since any signal (i.e. a one dimensional time trace) recorded at some point in space is ultimately the result of fully three dimensional interactions.

Many attempts have and continue to be made in accurately modelling fully three dimensional interactions which take into account relatively complete models for the incident field and allow for multiple interactions. However, such models inevitably lead to more and more complex results which are often difficult to quantify and incompatible with the methods developed to process signals in practice. As in many aspects of mathematical physics, a point can be reached were deterministic modelling based on fundamental physics leads to the development of results that are intractable and in such cases, it is of greater practical value to investigate the problem using stochastic models. Thus, the principle associated with the model $s = p \otimes f + n$ is based on assuming that the first order effects are compounded in the term $p \otimes f$ and that all other physical effects due to geometry, field patterns, multiple interactions together with electronic noise are described by an additive noise term, i.e. the stochastic function $n(t)$. Attempts are then made to accurately model the probability distribution function of n which in turn leads to the design of specific random number generators for simulating n.

14.11 Pseudo Random Number Generation

Random number generators are not random because they do not have to be. Most simple applications, such as computer games for example, need very few random numbers. Nevertheless, use of a poor random number generator can lead to strange correlations and unpredictable results which are compounded in terms of spurious correlations. These must be avoided at all costs.

The problem is that a random number generator does not produce a random sequence. In general, random number generators do not necessarily produce anything that looks even remotely like the random sequences produced in nature. However, with some careful tuning, they can be made to approximate such sequences. Of course, it is impossible to produce something truly random on a computer. As John von Neumann states, 'Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin'. Computers are deterministic, stuff goes in at one end, completely predictable operations occur inside, and different stuff comes out

the other end, a principle that includes a notion that is fundamental to DSP and computing in general, namely, 'rubbish in given rubbish out'. Put the same data into two identical computers, and the same data comes out of both of them (most of the time!).

A computer can only be in a finite number of states (a large finite number, but a finite number nonetheless), and the data that comes out will always be a deterministic function of the data that went in and the computer's current state. This means that any random number generator on a computer (at least, on a finite-state machine) is, by definition, periodic. Anything that is periodic is, by definition, predictable and can not therefore be random. A true random number generator requires some random input; a computer can not provide this.

14.11.1 Pseudo Random Sequences

The best a computer can produce is a pseudo random sequence generator. Many attempts have been made to define a pseudo random sequence formally and in this section, a general overview is given of these attempts. A pseudo random sequence is one that looks random. The sequence's period should be long enough so that a finite sequence of reasonable length - that is, one that is actually used - is not periodic. If for example, a billion random bits is required, then a random sequence generator should not be chosen that repeats after only sixteen thousand bits. These relatively short non-periodic sequences should be as indistinguishable as possible from random sequences. For example, they should have about the same number of ones and zeros, about half the runs (sequences of the same bit) should be of length one, one quarter of length two, one eighth of length three, and so on. In addition, they should not be compressible. The distribution of run lengths for zeros and ones should be the same. These properties can be empirically measured and then compared with statistical expectations.

A sequence generator is pseudo random if it has the following properties: It looks random, which means that it passes all the statistical tests of randomness that we can find. Considerable effort has gone into producing good pseudo random sequences on a computer. Discussions of generators abound in the literature, along with various tests of randomness. All of these generators are periodic (there is no exception); but with potential periods of 2^{256} bits and higher, they can be used for the largest applications. The problem with all pseudo random sequences is the correlations that result from their inevitable periodicity. Every pseudo random sequence generator will produce them if they are use extensively. A non periodic pseudo random sequence must have the following property: it is unpredictable. It must be computationally non-feasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in the stream.

14.11.2 Real Random Sequences

Is there such a thing as randomness? What is a random sequence? How do you know if a sequence is random? Is for example '101110100' more random than '101010101'? Quantum mechanics tells us that there is honest-to-goodness randomness in the real

world but can we preserve that randomness in the deterministic world of computer chips and finite-state machines? Philosophy aside, a sequence generator is really random if it has the following additional property: It cannot be reliably reproduced. If the sequence generator is run twice with the exact same input (at least as exact as computationally possible), then the sequences are completely unrelated; their cross correlation function is effectively zero. This property is not usually possible to produce on a finite state machine and for some applications of random number sequences, is not desirable, as in cryptography for example. Thus, we refer to those processes that produce number streams which look random (and passes appropriate statistical tests) and are unpredictable as Pseudo Random Number Generators (PRNG).

14.11.3 Pseudo Random Number Generators

The performance of many DSP algorithms depends on the degree of noise present in the signal and because many types of DSP algorithms are sensitive to noise, it is important to test their behaviour in the presence of noise. This is usually done by synthesizing noise signals which is accomplished using pseudo random number generators. Random numbers are not numbers generated by a random process but are numbers generated by a completely deterministic arithmetic process. The resulting set of numbers may have various statistical properties which together are called randomness. A typical mechanism for generating random numbers is via the iterative process defined by

$$x_{n+1} = (ax_n + b) \text{mod} P, \quad n \geq 0$$

which produces an integer number stream in the range $[0, P]$ and is known as the Linear Congruential Method (LCM). Here, the modular function mod operates in such a way as to output the remainder from the division of $ax_n + b$ by P, e.g.

$$23 \text{mod} 7 = 2 \quad \text{and} \quad 6 \text{mod} 8 = 6.$$

By convention $a \text{mod} 0 = a$ and $a \text{mod} b$ has the same sign as b. The reason for using modular arithmetic is because modular based functions tend to behave more erratically than conventional functions. For example consider the function $y = 2^x$ and the function $y = 2^x \text{mod} 13$ for example. The table below illustrates the difference between the output of these two function.

x	1	2	3	4	5	6	7	8
2^x	2	4	8	16	32	64	128	256
$2^x \text{mod} 13$	2	4	8	3	6	12	11	9

This approach to creating random sequences was first introduced by D H Lehmer in 1949. The values of the parameters are constrained as follows: $0 < a < P$, $0 \leq b < P$ and $0 \leq x_0 < P$. The essential point to understand when employing this method, is that not all values of the four parameters (a, b, x_0 and P) produce sequences that pass all the tests for randomness. Further, all such generators eventually repeat themselves cyclically, the length of this cycle (the period) being at most P. When $b = 0$ the algorithm is faster and referred to as the multiplicity congruential method and many authors refer to mixed congruential methods when $b \neq 0$.

An initial value or 'seed' x_0 is repeatedly multiplied by a and added to b, each product being reduced by modulo P. The element x_0 is commonly referred to as the seed. For example, suppose we let $a = 13$, $b = 0$, $P = 100$ and $x_0 = 1$; we will then generate the following sequence of two digit numbers

1, 13, 69, 97, 61, 93, 09, 17, 21, 73, 49, 37, 81, 53, 89, 57, 41, ...

For certain choices of a and P, the resulting sequence $x_0, x_1, x_2, ...$ is fairly evenly distributed over $(0, P)$ and contains the expected number of upward and downward double runs (e.g. 13, 69, 97) and triple runs (e.g. 9,17,21,73) and agrees with other predictions of probability theory. The values of a and P can vary and good choices are required to obtain runs that are statistically acceptable and have long cycle lengths, i.e. produce a long stream of numbers before the stream is repeated. For example, suppose we choose $a = 7$, $b = 12$, $P = 30$ and $x_0 = 0$, then the following sequence is generated

$$0, \quad 12, \quad 16, \quad 4, \quad 10, \quad 22, \quad 16, \quad 4, \quad 10, \quad 22, \quad 16, \quad 4, \quad ...$$

Here, after the first three digits, the sequence repeats the digits 4, 10, 22, 16; the 'cycle length' of the number generator is very short. To improve the cycle length, the value of P should be a prime number whose 'size' is close to that of the word length of the computer. The reason for using a prime number is that it is divisible by only 1 or itself. Hence, the modulo operation will always produce an output which is distinct from one element to the next. Many prime numbers are of the form $2^n - 1$ where n is an integer (Mersenne prime numbers and not for any value of n). A typical example of a Mersenne prime number is given by $2^{31} - 1 = 2147483648$. Values of the multiplier a vary considerable from one application to the next and include values such as 7^5 or 7^7 for example.

For long periods, P must be large. The other factor to be considered in choosing P is the speed of the algorithm. Computing the next number in the sequence requires division by P and hence a convenient choice is the word size of the computer. Perhaps the most subtle reasoning involves the choice of the multiplier a such that a cycle of period of maximum length is obtained. However, a long period is not the sole criterion that must be satisfied. For example, a = b = 1, gives a sequence which has a maximum period P but is anything but random. It is always possible to obtain the maximum period but a satisfactory sequence is not always attained. When P is the product of distinct primes only $a = 1$ will produce a full period, but when P is divisible by a high power of some prime, there is considerable latitude in the choice of a.

There are a few other important rules for optimising the performance of a random number generator using the linear congruential method in terms of developing sensible choices for a, b and P, these include:

- b is relatively prime to P.
- $a - 1$ is a multiple of every prime dividing P.
- $a - 1$ is a multiple of 4 if P is a multiple of 4.

These conditions allow a linear sequence to have a period of length P.

Pseudo random number generators are often designed to produce a floating point number stream in the range $[0, 1]$. This can be achieved by normalising the integer stream after the random integer stream has been computed. A typical example of a random number generator is given below using pseudo code.

```
x(1)=seed
a=7^5
P=2^31-1

for i=1 to n-1; do:
    x(i+1)=(a*x(i))mod(P)
enddo

max=0.
    for i=1 to n-1; do:
        if x(i) > max, then max=x(i)
enddo

for i=1 to n-1; do:
    x(i)=x(i)/max
enddo
```

Here, the first loop computes the random integer stream using the LCM, the second loop computes the maximum value of the array and the third loop nomalizes it so that on output, the random number stream consists of floating point numbers (to single or double precision) between 0 and 1 inclusively. The seed is typically a relatively long integer which is determined by the user. The exact value of the seed should not change the statistics of the output, but it will change the numerical values of the output array. These values can only be reproduced using the same seed, i.e. such pseudo random number generators do not satisfy the property that their outputs cannot be reliably reproduced.

The output of such a generator is good enough for many simulation type applications. There are a few simple guidelines to follow when using such random number generators:

(i) Make sure that the program calls the generator's initialization routine before it calls the generator.

(ii) Use initial values that are 'somewhat random', i.e. have a good mixture of bits. For example 2731774 and 10293082 are 'safer' than 1 or 4096 (or some other power of two).

(iii) Note that two similar seeds (e.g. 23612 and 23613) may produce sequences that are correlated. Thus, for example, avoid initialising generators on different processors or different runs by just using the processor number or the run numbers as the seed.

A typical C function for computing uniform random noise in the range 0 to 1 is given below.

```
#include<math.h>

void UNOISE(float s[], int n, long int seed)
{
long int i,x,a,P;
float max, temp;
                a=16807; /* 7^5 */
                P=2147483647; /* Mersenne prime number 2^{31}-1 */

        x=seed; /* Assign initial value */
                /* Do computation */
                 for(i=1; i<=n; i++)
                     {
                     x=(a*x)%P;
                  s[i]=(float)x;
                     }

        /* Compute maximum number */
        max=0.0;
               for(i=1; i<=n; i++)
               {
                temp=s[i];

                    if(fabs(temp) > max)
                    {
                     max=fabs(temp);
                    }
                }

        /* Normalize output */
           for(i=1; i<=n; i++) s[i]=fabs(s[i])/max;
}
```

In addition to the standard linear congruential generator discussed so far, a number of 'variations on a theme' can be considered such as the iteration

$$x_i = (a_1 x_{i-1}^2 + a_2 x_{i-1} + a_3) \mathrm{mod} P$$

or

$$x_i = (a_1 x_{i-1}^3 + a_2 x_{i-1}^2 + a_3 x_{i-1} + a_4) \mathrm{mod} P$$

and so on where a_n are predefined (integer) numbers and P is a prime.

14.11.4 Shuffling

A relatively simple method that further randomizes the output of a PRNG is to shuffle the values with a temporary storage. We first initialize an array $x_i, \quad i = 1, 2, ..., N$ with random numbers from the random number generator given above for example.

The last integer random number computed x_N is then set to M say. To create the next random sequence y_i, we apply the following process:

```
for i=1 to N, do:
    j=1+int(N*M)
      y(i)=x(j)
      M=x(i)
enddo
```

14.12 Additive Generators

An alternative solution to random number generation which creates very long cycles of values is based on additive generators. A typical algorithm commences by initialising an array x_i with random numbers (not all of which are even) so that we can consider the initial state of the generator to be $x_1, x_2, x_3, ...$ We then apply

$$x_i = (x_{i-a} + x_{i-b} + ... + x_{i-m}) \mathrm{mod} 2^n$$

where $a, b, ..., m$ and n are assigned integers. An example of this PRNG is the 'Fish generator' given by

$$x_i = (x_{i-55} + x_{i-24}) \mathrm{mod} 2^{32}.$$

This approach to pseudo random number generation is fast as no multiplication operations (e.g. ax_i) are required. The period of the sequence of random numbers is also very large and of the order of $2^f(2^{55} - 1)$ where $0 \leq f \leq n$.

A further example is the linear feedback shift register given by

$$x_n = (c_1 x_{n-1} + c_2 x_{n-2} + c_m x_{n-m}) \mathrm{mod} 2^k$$

which, for specific values of $c_1, c_2, ...c_m$ has a cycle length of 2^k.

14.12.1 Pseudo Random Number Generators and Cryptography

In cryptography, pseudo random number generation plays a central role as does modular arithmetic in general. One of the principal goals in cryptography is to design random number generators that provide outputs (random number streams) where no element can be predicted from the preceding elements given complete knowledge of the algorithm. Another important feature is to produce generators that have long cycle lengths. A further useful feature, is to ensure that the Entropy of the random number sequence is a maximum, i.e. that the histogram of the number stream is uniform. Finally, the use of modular arithmetic in the development of encryption algorithms tends to provide functions which are not invertible. They are one-way functions that can only be used to reproduce a specific (random) sequence of numbers from the same initial condition.

The basic idea in cryptography is to covert a plaintext file to a cyphertext file using a key that is used as a seed for the PRNG. A plaintext file is converted to a stream of integer numbers using ASCII (American Standard Code for Information Interchange)

conversion. For example, suppose we wish to encrypt the authors surname Blackledge for which the ASCII[3] decimal integer stream or vector is

$$\mathbf{f} = (66, 108, 97, 99, 107, 108, 101, 100, 103, 101).$$

Suppose we now use the linear congruential PRNG defined by

$$n_{i+1} = an_i \text{mod} P$$

where $a = 13$, $P = 131$ and let the seed be 250659, i.e. $n_0 = 250659$. The output of this iteration is

$$\mathbf{n} = (73, 32, 23, 37, 88, 96, 69, 111, 2, 26).$$

If we now add the two vectors together, we can generate the cypher stream

$$\mathbf{c} = \mathbf{f} + \mathbf{n} = (139, 140, 120, 136, 195, 204, 170, 211, 105, 127).$$

Clearly, provided the recipient of this number stream has access to the same algorithm (including the values of the parameters a and P) and crucially to the same seed, the vector $\mathbf{n}$ can be regenerated and $\mathbf{f}$ obtained from $\mathbf{c}$ by subtracting $\mathbf{n}$ from $\mathbf{c}$. This process can of course be accomplished using binary streams where the binary stream representation of the plaintext $\mathbf{f}_b$ and that of the random stream $\mathbf{n}_b$ say are used to generate the cypher binary stream $\mathbf{c}_b$ via the process

$$\mathbf{c}_b = \mathbf{n}_b \oplus \mathbf{f}_b$$

where $\oplus$ denotes the XOR operation. Restoration of the plaintext is then accomplished via the operation

$$\mathbf{f}_b = \mathbf{n}_b \oplus \mathbf{c}_b = \mathbf{n}_b \oplus \mathbf{n}_b \oplus \mathbf{f}_b.$$

Clearly, the process above is just an example of digital signal processing in which the information contained in a signal $\mathbf{f}$ (i.e. the plaintext) is 'scrambled' by introducing additive noise. Here, the seed plays the part of a key that it utilized for the process of encryption and decryption; a form of encryption that is commonly known as symmetric encryption in which the key is a private key known only to the sender and recipient of the encrypted message. Given that the algorithm used to generate the random number stream is publically available (together with the parameters it uses which are typically 'hard-wired' in order to provide a random field pattern with a long cycle length), the problem is how to securely exchange the key to the recipient of the encrypted message so that decryption can take place. If the key is particular to a specific communication and is used once and once only for this communication (other communications being encrypted using other keys), then the processes is known as a one-time pad, because the key is only used once. Simple though it is, this process is not open to attack. In other words, no form of cryptanalysis will provide a way of decyphering the encrypted message. The problem is how to exchange the keys in a way that is secure and thus, solutions to the key exchange problem are paramount in symmetric encryption. A well known historical example of this problem involved

[3]Any code can be used.

the distribution of the keys used to initialize the Enigma cypher used by the German forces during the Second World War. The Enigma machine (which was named after Sir Edward Elgar's composition, the 'Enigma Variations') was essentially an electro-mechanical PRNG in which the the seed was specified using a plug board and a set of three (and later four) rotors whose initial positions could be changed. These settings were effectively equivalent to a password or a private key as used today. For a period of time and using a very simplistic and rather exaggerated explanation, the German land forces sometimes communicated the password used on a particular day (and at a set time) by radio transmission using standard Morse code. This transmission was sometimes repeated in order to give the recipient(s) multiple opportunity to receive the key(s) accurately. Worse still, in some rare but important cases, the passwords were composed of simple names (of some of the politicians at the time for example). Thus, in many cases, a simple password consisting of a well known name was transmitted a number of times sequentially leading to near perfect temporal correlation of the initial transmission. This was a phenomenally irresponsible way of using the Enigma system. In today's environment, it is like choosing a password for your personal computer which is a simple and possibly well known name (of the your boss for example) that is easily remembered, shouting it out a number of times to your colleagues in an open plan office and then wondering why everyone seems to know something about your private parts! In this sense, the ability for the British war time intelligence services to decypher the German land forces communications is self-evident. The use of Enigma by the German naval forces (in particular, the U-boat fleet) was far more secure in that the password used from one day to the next was based on a code book provided to the users prior to departure from base. Thus, no transmission of the daily passwords was required and, if not for a lucky break, in which one of these code books was recovered in tact by a British destroyer (HMS Bulldog) from a damaged U-boat, breaking the Enigma naval transmissions under their time variant code-book protocol would have been effectively impossible. Although the Enigma story has many facets to those discussed here, a careful study of this historically intriguing technology reveals that the breaking of Enigma had as much to do with German stupidity and some bad luck as it did with British intelligence coupled with some good luck. Thus is the reality of how random events (or lucky breaks to some) of the past can effect the outcome of the future!

The discussion above has been used by way of an example to highlight the problem of exchanging keys when applying a symmetric encryption scheme. It also provides an example of how, in addition to developing the technology for encryption, it is imperative to develop appropriate protocols and procedures for using it effectively with the aim of reducing inevitable human error, one of the underlying principles being the elimination of any form of temporal correlation. Another fundamental principle which has been demonstrated time and again throughout the history of cryptology is that although improvements in methods and technology are to be welcomed, information security is ultimately to do with cultivating the 'right state of mind' and that part of this state should include a healthy respect for the enemy.

In cryptography, the design of specialized random number generators with idealized properties forms the basis of many of the techniques that are applied to the area of information technology security. There are many such PRNGs available for this

purpose such as the Blum, Blum, Shub generator given by

$$x_{i+1} = x_i^2 \mathrm{mod}(pq)$$

where p and q are two prime numbers whose product forms the so called Blum integer, the Blum-Mercali generator

$$x_{i+1} = q^{x_i} \mathrm{mod} p$$

where q is a prime and p is an odd prime and the RSA (Rivest, Shamir and Adleman) generator given by

$$x_{i+1} = x_i^e \mathrm{mod}(pq)$$

where e is a relative prime of $p-1$ and $q-1$. The latter PRNG is the basis for public/private or asymmetric encryption methods and is fundamental to all PKI (Public Key Infrastructure) systems and was first developed in the early 1970s. Here, the public key is given by the number e and the product pq which are unique to a given recipient and in the public domain (like an individuals telephone number). This public key is then used to encrypt a message transformed into a decimal integer array M_i say using the one-way function

$$C_i = M_i^e \mathrm{mod}(pq).$$

The recipient is then able to decrypt the cyphetext C_i with knowledge of p and q which represent the private key. This is done by solving the equation

$$de = \mathrm{mod}[(p-1)(q-1)]$$

for d and then using the result

$$M_i = C_i^d \mathrm{mod}(pq).$$

In this way, the sender and receiver do not have to exchange a key before encryption/decryption can take place and such systems, in effect solve the key exchange problem associated with symmetric cyphers. However, a symmetric cypher is still systematically more secure as the RSA algorithm can in principle be broken by searching through different combinations of the product pq and hence $(p-1)(q-1)$ given that: (i) there is a limit to the number of primes that are available and able to be stored on a standard computer such as a 32-bit PC for example; (ii) knowledge of the fact that e (which is known) is a relative prime of $p-1$ and $q-1$ which provides a search domain.

14.12.2 Gaussian Random Number Generation

The generation of Gaussian random numbers which are taken to conform to the distribution

$$P(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right)$$

where σ is the standard deviation, is important in the analysis of real signals because many signals are characterized by additive noise that is Gaussian or normally distributed. The method is based on the Box-Muller transform which, in effect, transforms

uniformly distributed deviates into Gaussian distributed deviates. The basic idea is to first create two uniform deviates x_1 and x_2 say on $(0,1)$. Now, assume that we wish to create two values y_1 and y_2 which conform to the Gaussian probability distribution function

$$P(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)$$

which has a zero mean and a standard deviation of 1. We can then consider a relationship between x_1, x_2, y_1 and y_2 of the form

$$y_1 = \sqrt{-2\ln x_1}\cos(2\pi x_2) \quad \text{and} \quad y_2 = \sqrt{-2\ln x_1}\sin(2\pi x_2)$$

or equivalently

$$x_1 = \exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] \quad \text{and} \quad x_2 = \frac{1}{2\pi}\tan^{-1}\frac{y_2}{y_1}.$$

Further, suppose we let

$$\sin(2\pi x_2) = \frac{v_1}{R} \quad \text{and} \quad \cos(2\pi x_2) = \frac{v_2}{R}.$$

Then $R^2 = v_1^2 + v_2^2$ and if we set $x_1 = R^2$, then we obtain the result that

$$y_1 = v_1\sqrt{\frac{-2\ln r}{r}} \quad \text{and} \quad y_2 = v_2\sqrt{\frac{-2\ln r}{r}}$$

where $r = R^2$. Here, v_1 and v_2 are uniform deviates on $(0,1)$ such that $r \leq 1$.

Note that if we compute the joint probability distribution of y_1 and y_2, then

$$p(y_1, y_2)dy_1dy_2 = p(x_1, x_2)\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2}\right| dy_1dy_2$$

where the Jacobian determinant is given by

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = -\left[\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y_1^2}{2}\right)\right]\left[\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y_2^2}{2}\right)\right]$$

which shows that y_1 and y_2 are independent and that the method creates two Gaussian deviates from two uniformly random deviates as required. Thus, an algorithm for implementing this method is as follows:

repeat
v_1=RAND()
v_2=RAND()
$r = v_1^2 + v_2^2$
until $r \leq 1$

$y_1 = v_1\sqrt{\frac{-2\ln r}{r}}$
$y_2 = v_2\sqrt{\frac{-2\ln r}{r}}$

where the function RAND() is taken to output a uniform random deviate using the linear congruential method discussed earlier.

The following C code provides a function GNOISE that outputs a Gaussian random field using the method discussed above. The process generates two arrays of uniform deviates (with different seeds) using the function UNOISE and feeds these deviates as pairs into the Box-Muller transform.

```
#include<math.h>

void UNOISE( float s[], int n, long int seed );

void GNOISE( float s[], int n, long int seed )

/* FUNCTION: Generates an n size array s of Gaussian distributed   */
/*           noise with zero mean and a standard deviation of 1.   */

{
  int i, k, nn;
  float r, fac, v1, v2, *x1, *x2;

  /*Allocate internal work space.*/
  x1 = (float *) calloc( n+1, sizeof( float ) );
  x2 = (float *) calloc( n+1, sizeof( float ) );

  nn=n/2;
  UNOISE(x1,nn,seed);/*Generate uniform deviates.*/
  seed=seed+3565365; /*Add randomly chosen integer to seed.*/
  UNOISE(x2,nn,seed);/*Generate new set of uniform deviates.*/

      k=0;
      for(i=1; i<=nn; i++)
      {
      v1 = 2.0 * x1[i] - 1.0; /* -1 < v1 < 1 */
      v2 = 2.0 * x2[i] - 1.0; /* -1 < v2 < 1 */

      r  = pow( v1, 2 ) + pow( v2, 2 );
      r = r/2;/* r<=1 */

      /* Apply the Box-Muller transform */

      fac=sqrt( (double) -2.0 * log(r)/r);

      /*Write to output array.*/
      s[k]=v1*fac;
      s[k+1]=v2*fac;
      k=k+2;
      }
```

```
}
```

14.13 Chaos

Chaos is often associated with noise in that it is taken to represent a field which is unpredictable. Although this is the case, a signal generated by a chaotic system generally has more structure if analysed in an appropriate way. Moreover, this structure often exhibits features that are similar at different scales which leads to a natural connection between the behaviour of chaotic systems, the signals they produce and fractal or self-affine signals which is the subject of Chapter 17. Thus, chaotic signals are not the same as noisy signals either in terms of their behaviour or the way in which they are simulated. Chaotic signals are typically the product of a iteration of the form $x_{n+1} = f(x_n)$ where the function f is some nonlinear map which depends on a single or a set of parameters. The chaotic behaviour of x_n depends critically of the value of the parameter(s). The iteration process may not necessarily be a single nonlinear mapping but consist of a set of nonlinear coupled equations of the form

$$\begin{aligned} x_{n+1}^{(1)} &= f_1(x_n^{(1)}, x_n^{(2)}, ..., x_n^{(N)}), \\ x_{n+1}^{(2)} &= f_2(x_n^{(1)}, x_n^{(2)}, ..., x_n^{(N)}), \\ &\vdots \\ x_{n+1}^{(N)} &= f_N(x_n^{(1)}, x_n^{(2)}, ..., x_n^{(N)}) \end{aligned}$$

where the functions $f_1, f_2, ..., f_N$ may be nonlinear or otherwise. In turn, such a coupled system can be the result of many different physical models covering a wide range of applications in science and engineering.

It is opportune at this point to consider a well known example. Suppose there is a fixed population of N individuals living on an island (with no one leaving or entering) and a fatal disease (for which there is no cure) is introduced, which is spread through personal contact causing an epidemic due to a promiscuous life style for example. The rate of growth of the disease will normally be proportional to the number of carriers c say. Suppose we let $x = c/N$ be the proportion of individuals with the disease so that $100x$ is the percentage of the population with the disease. Then, the equation describing the rate of growth of the disease is

$$\frac{dx}{dt} = kx$$

whose solution is

$$x(t) = x_0 \exp(kt)$$

where x_0 is the proportion or the population carrying the disease at $t = 0$ (i.e. when the disease first 'strikes') and k is a constant of proportionality defining the growth rate. The problem with this conventional growth rate model, is that when $x = 1$, there can be no further growth of the disease because the island population no longer exists and so we must impose the condition that $0 < x(t) \leq 1, \ \ \forall t$. Alternatively, suppose we include the fact that the rate of growth must also be proportional to the

number of individuals $1 - x$ who do not become carriers, due to isolation of their activities and/or genetic disposition for example. Then, our rate equation becomes

$$\frac{dx}{dt} = kx(1 - x)$$

and if $x = 1$, the epidemic is extinguished. This equation can be used to model a range of situations similar to that introduced above associated with predator-prey type processes. (In the example given above, the prey is the human and the predator could be a virus or bacterium for example.) Finite differencing over a time interval Δt, we have

$$\frac{x_{n+1} - x_n}{\Delta t} = kx_n(1 - x_n)$$

or

$$x_{n+1} = x_n + k\Delta t x_n(1 - x_n)$$

or

$$x_{n+1} = rx_n(1 - x_n)$$

where $r = 1 + k\Delta t$. This is a simple quadratic iterator known as the logistic map and has a range of characteristics depending on the value of r. This is illustrated in Figure 14.3 which shows the output (just 30 elements) from this iterator for $r = 1$, $r = 2$, $r = 3$ and $r = 4$ for an initial value of 0.1.[4]

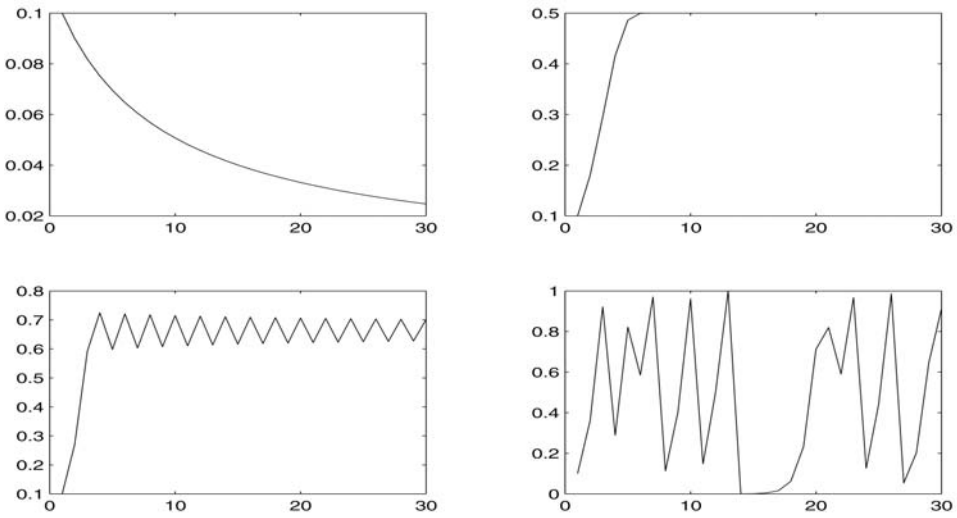


Figure 14.3: Output (30 elements) of the logistic map for values of $r = 1$ (top left), $r = 2$ (top right), $r = 3$ (bottom left) and $r = 4$ (bottom right) and an initial value of 0.1.

[4]The initial value, which is taken to be any value between 0 and 1, changes the signature of the output but not its characteristics.

For $r = 1$ and $r = 2$, convergent behaviour takes place; for $r = 3$ the output is oscillatory and for $r = 4$ the behaviour is chaotic. The transition from monotonic convergence to oscillatory behaviour is known as a bifurcation and is better illustrated using a so called Fiegenbaum map or diagram which is a plot of the output of the iterator in terms of the values produced (after iterating enough times to produce a consistent output) for different values of r. An example of this for the logistic map is given in Figure 14.4 for $0 < r \leq 4$ and shows convergent behaviour for values of r from 0 to approximately 3, bifurcations for values of r between approximately 3 and just beyond 3.5 and then a region of chaotic behaviour, achieving 'full chaos' at $r = 4$ where the output consists of values between 0 and 1.

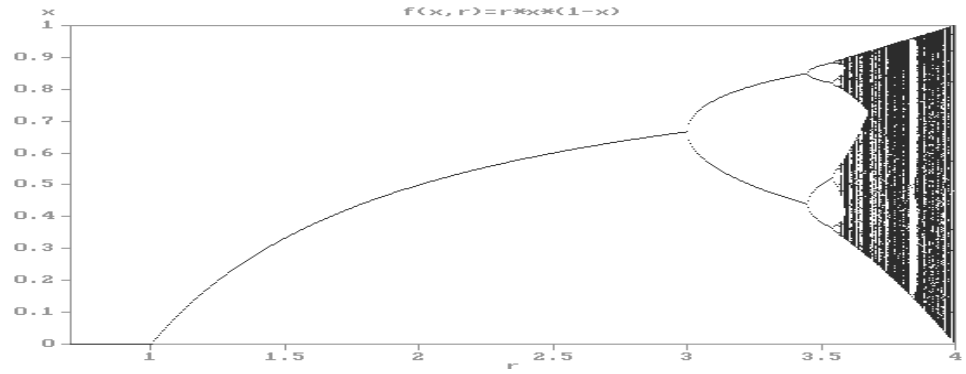


Figure 14.4: Feigenbaum diagram of the logistic map for $0 < r < 4$ and $0 < x < 1$.

However, closer inspection of this data representation reveals repeating patterns, an example being given in Figure 14.5 which gives a Fiegenbaum diagram of the output for values of r between 3.840 and 3.855 and values of x between 0.44 and 0.52.

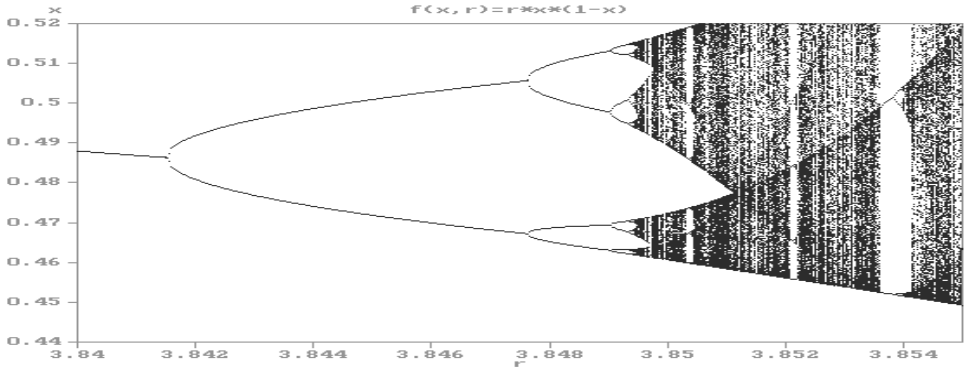


Figure 14.5: Feigenbaum diagram of the logistic map for $3.840 < r < 3.855$ and $0.44 < x < 0.52$.

As before, we observe a region of convergence, bifurcation and then chaos. Moreover, from Figure 14.5 we observe another region of this map (for values of r around 3.854) in which this same behaviour occurs. The interesting feature about this map is that the convergence→bifurcation→chaos characteristics are repeated albeit at smaller scales. In other words, there is a similarity of behaviour at smaller scales, i.e. the pattern of behaviour is self-similar or 'fractal'. Moreover, all this comes from a remarkably simple iterator, i.e. the map $x \to rx(1-x)$.

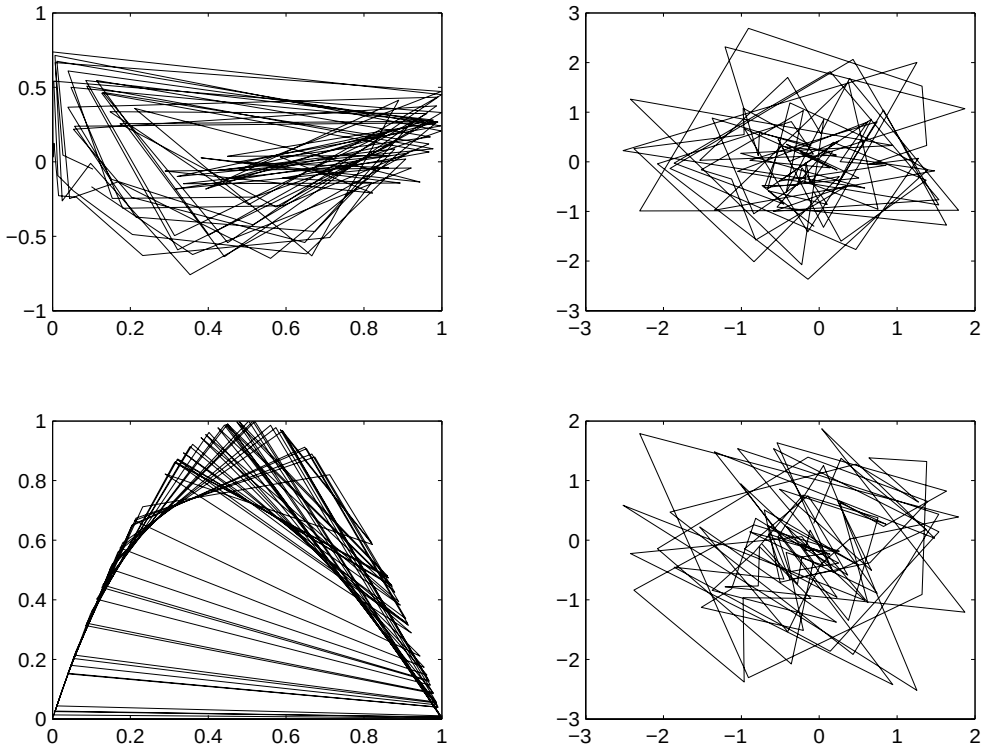


Figure 14.6: Complex plane diagram and phase portrait of the chaotic field generated by the logistic map for 100 iterations (top left and bottom left respectively) and those of a random Gaussian noise field (top right and bottom right respectively).

Another feature of a chaotic signal which can be used as an analysis tool, is based on its characterization in the complex plane (via application of the Hilbert transform) and as a 'phase portrait' (e.g. a plot of x_{n+1} against x_n) which reveals structures not associated with noise. Figure 14.6 shows a plot of these data representations for the logistic map (for $r = 4$) which should be compared to those associated with a random Gaussian noise field which is also provided. Figure 14.7 shows plots of the distribution

of the principal phases associated with the fields given in Figure 14.6.[5] The phase distributions for the chaotic field have specific characteristics showing that the phase of the chaotic field are not uniformly distributed unlike those of a Gaussian noise field which are uniformly distributed. The phase distribution of a phase map is a useful method of quantifying a chaotic signal as it typically provides an unambiguous and unique signature which is specific to a given chaotic process.

In addition to the logistic map, which has been used here to introduce chaos, there are a wide variety of other maps which yield signals that exhibit the same basic properties as the logistic map (convergence→bifurcation→chaos) with similar structures at different scales at specific regions of the Feigenbaum diagram.

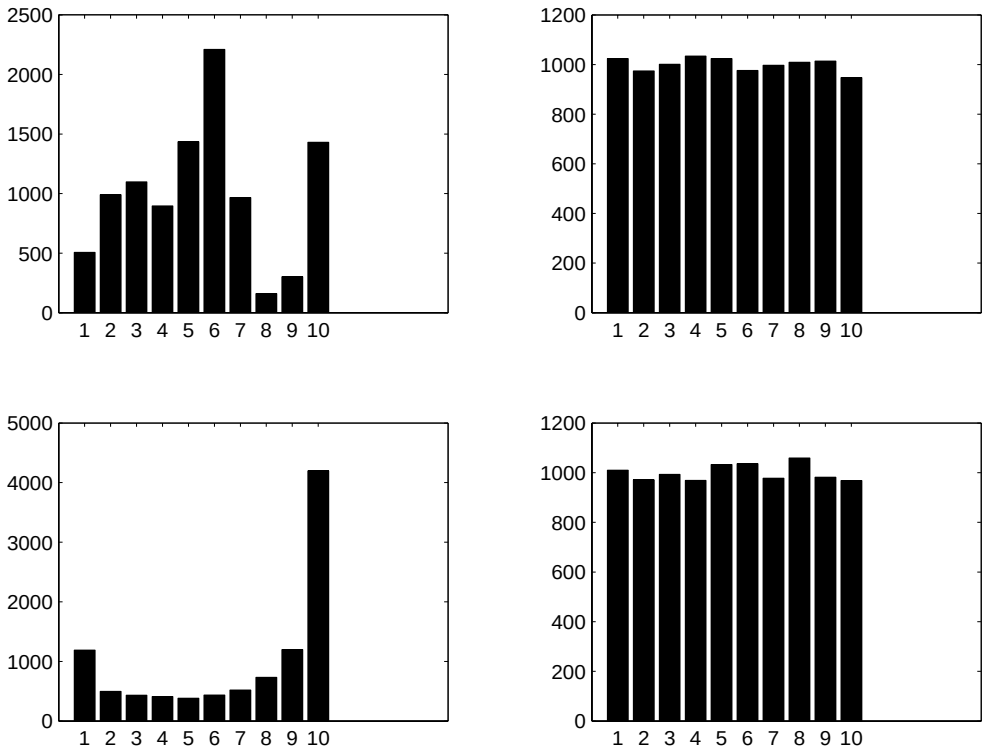


Figure 14.7: Frequency distribution (using 10 bins) of the principal phase of the fields given in Figure 14.6 for 1000 iterations

Examples, include the following:

Linear functions

The sawtooth map

$$x_{n+1} = 5x_n \text{mod} 4.$$

[5]The principle phase is given by $\tan^{-1}(\hat{\text{H}}(x_n)/x_n)$ where $\hat{H}$ denotes the Hilbert transform for the complex plane map and $\tan^{-1}(x_n/x_{n+1})$ for the phase portrait where $-\pi \leq \tan^{-1}(x_n) \leq \pi$.

The tent map

$$x_{n+1} = r(1- \mid 2x_n - 1 \mid).$$

The generalized tent map

$$x_{n+1} = r(1- \mid 2x_n - 1 \mid^m), \quad m = 1, 2, 3, ...$$

Nonlinear functions

The sin map

$$x_{n+1} = \mid \sin(\pi r x_{n+1} \mid .$$

The tangent feedback map

$$x_{n+1} = rx_n[1 - \tan(1/2x_n)].$$

The logarithmic feedback map

$$x_{n+1} = rx_n - n[1 - \log(1 + x_n)].$$

Further, there are a number of 'variations on a theme' that are of value, an example being the 'delayed logistic map'

$$x_{n+1} = rx_n(1 - x_{n-1})$$

which arises in certain problems to population dynamics. Moreover, coupled iterative maps occur from the development of physical models leading to nonlinear coupled differential equations, a famous and historically important example being the Lorenz equations given by

$$\frac{dx_1}{dt} = a(x_2 - x_1), \quad \frac{dx_2}{dt} = (b - x_3)x_1 - x_2, \quad \frac{dx_3}{dt} = x_1x_2 - cx_3$$

where a, b and c are constants. These equations were originally derived by Lorenz from the fluid equations of motion (the Navier Stokes equation, the equation for thermal conductivity and the continuity equation) used to model heat convection in the atmosphere and were studied in an attempt to explore the transition to turbulence where a fluid layer in a gravitational field is heated from below. Finite differencing, these equations become

$$\begin{aligned} x_1^{(n+1)} &= x_1^{(n)} + \Delta t a(x_2^{(n)} - x_1^{(n)}), \\ x_2^{(n+1)} &= x_2^{(n)} + \Delta t[(b - x_3^{(n)})x_1^{(n)} - x_2^{(n)}], \\ x_3^{(n+1)} &= x_3^{(n)} + \Delta t[x_1^{(n)}x_2^{(n)} - cx_3^{(n)}]. \end{aligned}$$

For specific values of a, b and c (e.g. $a = 10, b = 28$ and $c = 8/3$) and a step length Δt, the digital signals $x_1^{(n)}, x_2^{(n)}$ and $x_3^{(n)}$ exhibit chaotic behaviour which can be analysed quantitatively in the three dimension phase space (x_1, x_2, x_3) or variations on this theme, e.g. a three dimensional plot with axes $(x_1 + x_2, x_3, x_1 - x_3)$ or as a two

dimensional projection with axes $(x_1 + x_2, x_3)$ an example of which is shown in Figure 14.8. Here, we see that the path is confined to two domains which are connected. The path is attracted to one domain and then to another but this connection (the point at which the path changes form one domain to the next) occurs in quite an intricate manner - an example of a 'strange attractor'.

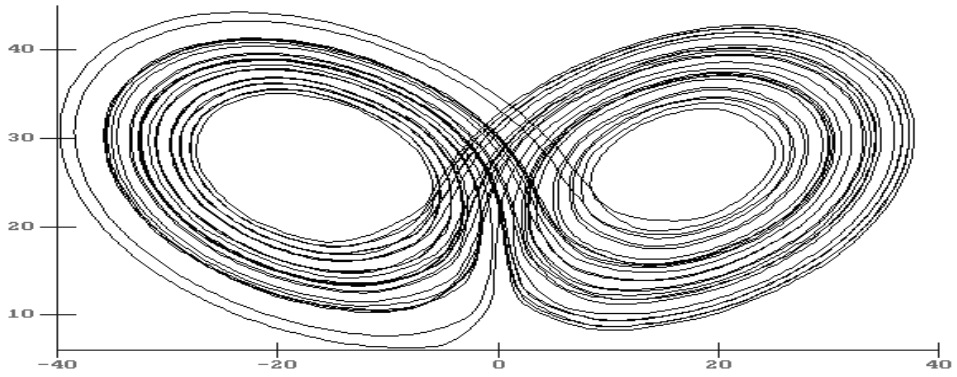


Figure 14.8: Two dimensional phase space analysis of the Lorenz equations illustrating the 'strange attractor'.

As with the simple iterative maps discussed previously, there are a number of non-linear differential equations (coupled or otherwise) that exhibit chaos whose behaviour can be quantified using an appropriate phase space. These in include:

The Rössler equations

$$\frac{dx_1}{dt} = -x_2 - x_3, \quad \frac{dx_2}{dt} = x_3 + ax_2, \quad \frac{dx_3}{dt} = b + x_3(x_1 + c).$$

The Hénon-Heiles equations

$$\frac{dx_1}{dt} = p_x, \quad \frac{dp_x}{dt} = -x - 2xy; \quad \frac{dx_2}{dt} = p_y, \quad \frac{dp_y}{dt} = -y - x^2 + y^2.$$

The Hill's equations

$$\frac{d^2}{dt^2}u(t) + \omega^2(t)u(t) = 0,$$

a special case being the Mathieu equation where

$$\omega^2(t) = \omega_0^2(1 + \lambda\cos\omega t),$$

ω_0 and λ being constants.

The Duffing Oscillator

$$\frac{dx}{dt} = v, \quad \frac{dv}{dt} = av + x + bx^3 + \cos t$$

where a and b are constants.

In each case, the chaotic nature of the output to these systems depends on the values of the constants.

14.13.1 The Lyapunov Exponent and Dimension

For iterative processes where stable convergent behaviour is expected, an output that is characterised by exponential growth can, for example, be taken to be due to unacceptable numerical instability. However, some algorithms are intrinsically unstable and do not, for example, converge to a specific value. Such algorithms include those that are based on physical models that exhibit chaos. A characteristic that can help to quantify the characteristics of such algorithms is the Lyapunov exponent which, in turn, can be taken to be a measure of the systems 'chaoticity'.

Consider the iterative system

$$f_{n+1} = F(f_n) = f + \epsilon_n$$

where ϵ_n is a perturbation to the value of f at an iterate n which is independent of the value of f_0. If the system converges to f as $n \to \infty$ then $\epsilon_n \to 0$ as $n \to \infty$ and the system is stable. If this is not the case, then the system may be divergent or chaotic. Suppose we model ϵ_n in terms of an exponential growth ($\sigma > 0$) or decay ($\sigma < 0$) so that

$$\epsilon_{n+1} = c \exp(n\sigma)$$

where c is an arbitrary constant. Then $\epsilon_1 = c$, $\epsilon_2 = \epsilon_1 \exp(\sigma)$, $\epsilon_3 = \epsilon_1 \exp(2\sigma) = \epsilon_2 \exp(\sigma)$ and thus, in general, we can write

$$\epsilon_{n+1} = \epsilon_n \exp(\sigma).$$

Noting that

$$\ln\left(\frac{\epsilon_{n+1}}{\epsilon_n}\right) = \sigma$$

we can write

$$\sum_{n=1}^{N} \ln\left(\frac{\epsilon_{n+1}}{\epsilon_n}\right) = N\sigma.$$

Thus, we can define σ as

$$\sigma = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \ln\left(\frac{\epsilon_{n+1}}{\epsilon_n}\right).$$

The constant σ is known as the Lyapunov exponent. Since we can write

$$\sigma = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} (\ln \epsilon_{n+1} - \ln \epsilon_n)$$

and noting that (using forward differencing)

$$\frac{d}{dx} \ln \epsilon \simeq \frac{\ln \epsilon_{n+1} - \ln \epsilon_n}{\delta x} = \ln \epsilon_{n+1} - \ln \epsilon_n, \quad \delta x = 1$$

we see that σ is, in effect, given by the mean value of the derivatives of the natural logarithm of ϵ. Note that, if the value of σ is negative, then the iteration is stable and will approach f since we can expect that as $N \to \infty$, $\epsilon_{n+1}/\epsilon_n < 1$ and, thus, $\ln(\epsilon_{n+1}/\epsilon_n) < 0$. If σ is positive, then the iteration will not converge to f but will diverge or, depending on the characteristics of the mapping function F, will exhibit chaotic behaviour. The Lyapunov exponent is a parameter that is a characterization of the 'chaoticity' of the signal f_n. In particular, if we compute σ_N using N elements of the signal f_n and then compute σ_M using M elements of the same signal, we can define the Lyapunov dimension as

$$D_L = \begin{cases} 1 - \frac{\sigma_N}{\sigma_M}, & \sigma_M > \sigma_N; \\ 1 - \frac{\sigma_M}{\sigma_N}, & \sigma_M < \sigma_N. \end{cases}$$

where

$$\sigma_N = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \ln \left| \frac{\epsilon_{n+1}}{\epsilon_n} \right|.$$

14.14 Case Study: Cryptography using Chaos

The use of deterministic chaos[6] for encrypting data follows the same basic approach as that discussed earlier with regard to the application of modular based pseudo random number generation. Pseudo chaotic numbers are in principle, ideal for cryptography because they produce number streams that are ultra-sensitive to the initial value (the key). However, instead of using iterative based maps using modular arithmetic with integer operations, here, we require the application of principally nonlinear maps using floating point arithmetic. Thus, the first drawback concerning the application of deterministic chaos for encryption concerns the processing speed, i.e. pseudo random number generators (PRNGs) generate integer streams using integer arithmetic where as pseudo chaotic number generators (PCNGs) produce floating point streams using floating point arithmetic. Another drawback of chaos based cryptography is that the cycle length (i.e. the period over which the number stream repeats itself) is relatively short when compared to the cycle length available using conventional PRNGs (e.g. additive generators). Thus, compared with conventional approaches, the application of deterministic chaos has (at least) two distinct disadvantages. However, providing the application of chaos in this field has some valuable advantages, the computational overheads can be enhanced through the use of appropriate real time DSP units (essentially, high performance floating point accelerators). Moreover, the lower cycle lengths can be overcome by designing block cyphers which is where an iterator produces a cypher stream only over a block of data whose length is significantly less than that of the cycle length of the iterator, each block being encrypted using a different key and/or algorithm. So are there any advantages to using deterministic chaos? One advantage is compounded in Figure 14.9 which qualitatively illustrates complexity as a function of information showing regions associated with ordered, random and chaotic fields. Imagine that an algorithm can output a number stream which can be ordered, chaotic or random. In the case of an ordered number stream (those generated from a discretized piecewise continuous functions for example), the complexity

[6]Based on the research of Dr N Ptitsyn, a former research student of the author.

of the field is clearly low. Moreover, the information and specifically the information Entropy (the lack of information we have about the exact state of the number stream) is low as is the information content that can be conveyed by such a number stream. A random number stream (taken to have a uniform distribution for example) will provide a sequence from which, under ideal circumstances, it is not possible to predict any number in the sequence from the previous values. All we can say is that the probability of any number occurring between a specified range is equally likely. In this case, the information entropy is high. However, the complexity of the field, in terms its erratic transitions from one type of localized behaviour to another, is low.

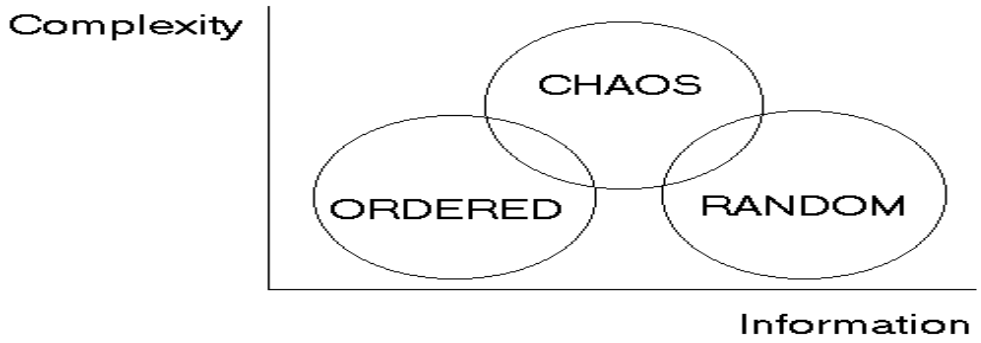


Figure 14.9: Qualitative comparison of ordered, random and chaotic fields in terms of their complexity and information content.

Thus, in comparison to a random field, a chaotic field is high in complexity but its information entropy, while naturally higher than an ordered field is lower than that of a random field, e.g. chaotic fields which exhibit uniform number distributions are rare.

From the discussion above, the application of deterministic chaos to encryption has a number of disadvantages relative to the application of PRNGs. However, the increased level of complexity can be used to provide complexity driven block cyphers. One method of approach is to use well known maps and modify them to extend the region of chaos. For example, the Matthews cypher is a modification of the logistic map to

$$x_{n+1} = (1+r)\left(1+\frac{1}{r}\right)^{r} x_n(1-x_n)^r, \quad r \in (0,4].$$

The effect of this generalization is seen in Figure 14.10 which shows the Feigenbaum diagram for values of r between 1 and 4. Compared to the conventional logistic map $x_{n+1} = rx_n(1-x_n), \quad r \in (0,4]$ which yields full chaos at $r = 4$, the chaotic behaviour of the Matthews map is clearly more extensive providing full chaos for the majority (but not all) of values of r between approximately 0.5 and 4. In the conventional case, the key is the value of x_0 (the initial condition). In addition, because there is a wide

range of chaotic behaviour for the Matthews map, the value of r itself can be used as a primary or secondary key.

The approach to using deterministic chaos for encryption has to date, been based on using conventional and other well known chaotic models of the type discussed above with modifications such as the Matthew map as required. However, in cryptography, the physical model from which a chaotic map has been derived is not important; only the fact that the map provides a cypher that is 'good' at scrambling the plaintext. This point leads to an approach which exploits two basic features of chaotic maps: (i) they increase the complexity of the cypher; (ii) there are an unlimited number of maps of the form $x_{n+1} = f(x_n)$ that can be literally 'invented' and then tested for chaoticity to produce a data base of algorithms.

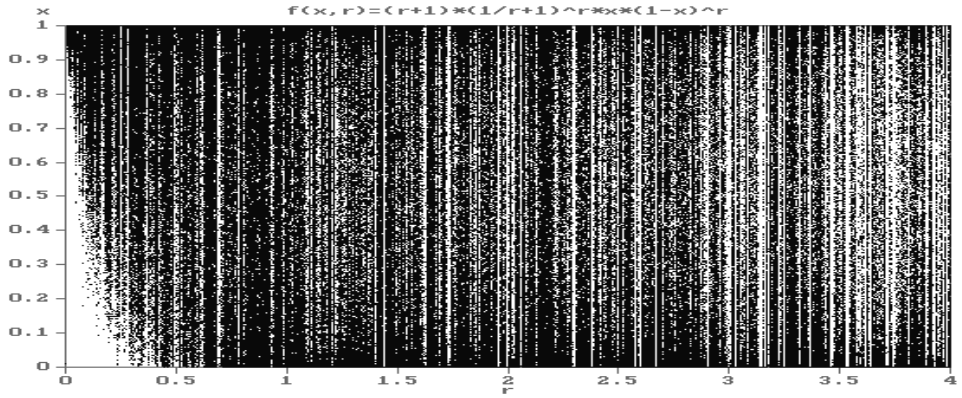


Figure 14.10: Feigenbaum map of the Matthews cypher

14.14.1 Block Cyphers using Deterministic Chaos

The low cycle lengths that are inherent in chaotic maps leads naturally to consider their application to block cyphers. However, instead of using a single algorithm to encrypt data over a series of blocks using different (block) keys, here we can use different algorithms, i.e. chaotic maps. Two maps can be used to generate the length of each block and the maps that are used to encrypt the plaintext over each block. Thus, suppose we have designed a data base consisting of 100 chaotic maps say consisting of iterative functions $f_1, f_2, f_3, ..., f_{100}$, each of which generates a floating point number steam through the operation

$$x_{n+1} = f_m(x_n, p_1, p_2, ...)$$

where the parameters $p_1, p_2, ...$ are pre-set or 'hard-wired' to produce chaos for any initial value $x_0 \in (0, 1)$ say. An 'algorithm selection key' is then introduced in which two algorithms (or the same algorithm) are chosen to 'drive' the block cypher - f_{50} and f_{29} say, the session key in this case being $(50, 29)$. Here, we shall consider the case where map f_{50} determines the algorithm selection and map f_{29} determines the block

size. Map f_{50} is then initiated with the key 0.26735625 say and map f_{29} with the key 0.65376301 say. The output from these maps (floating point number streams) are then normalized, multiplied by 100 and 1000 respectively for example and then rounded to produce integer streams with values ranging from 0 to 100 and 0 to 1000 respectively. Let us suppose that the first few values of these integer streams are $28, 58, 3, 61$ and $202, 38, 785, 426$. The block encryption starts by using map 28 to encrypt 202 elements of the plaintext using the key 0.78654876 say. The second block of 38 elements is then encrypted using map 58 (the initial value being the last floating point value produced by algorithm 28) and the third block of 785 elements is encrypted using algorithm 3 (the initial value being the last floating point value produced by algorithm 58) and so on. The process continues until the plaintext has been fully encrypted with the 'session key' (50,29,0.26735625,0.65376301,0.78654876).

14.14.2 Encrypting Processes

The encryption can be undertaken using a binary representation of the plaintext and applying an XOR operation using a binary representation of the cypher stream. This can be constructed using a variety of ways. For example, one could extract the last significant bits from the floating point format of x_n for example. Another approach, is to divide the floating point range of the cypher into two compact regions and apply a suitable threshold. For example, suppose that the output x_n from a map operating over a given block consists of floating point value between 0 and 1, then, with the application of a threshold of 0.5, we can consider generating the bit stream

$$b(x_n) = \begin{cases} 1, & x_n \in (0.5, 1]; \\ 0, & x_n \in [0, 0.5). \end{cases}$$

However, in applying such a scheme, we are assuming that the distribution of x_n is uniform an this is rarely the case with chaotic maps. Figure 14.11 shows the PDF for the logistic map $x_{n+1} = 4x_n(1 - x_n)$ which reveals a non-uniform distribution with a bias for floating point number approach 0 and 1. However, the mid range (i.e. for $x_n \in [0.3, 0.7]$) is relatively flat indicating that the probability for the occurrence of different numbers generated by the logistic map in the mid range is the same. In order to apply the threshold partitioning method discussed above in a way that provides an output that is uniformly distributed for a any chaotic map, it is necessary to introduce appropriate conditions and modify the above to the form

$$b(x_n) = \begin{cases} 1, & x_n \in [T, T + \Delta_+); \\ 0, & x_n \in [T - \Delta_-, T); \\ -1, & \text{otherwise.} \end{cases}$$

where T is the threshold and Δ_+ and Δ_- are those values which characterize (to a good approximation) a uniform distribution. For example, in the case of the logistic map $T = 0.5$ and $\Delta_+ = \Delta_- = 0.2$. This aspect of the application of deterministic chaos to cryptography, together with the search for a parameter or set of parameters that provides full chaos for an 'invented' map, determines the overall suitability of the function that has been 'invented' for this application. The 'filtering' of a chaotic field to generate a uniformly distributed output is equivalent to maximizing the entropy

of the cypher stream (i.e. generating a cypher stream with a uniform PDF) which is an essential condition in cryptography.

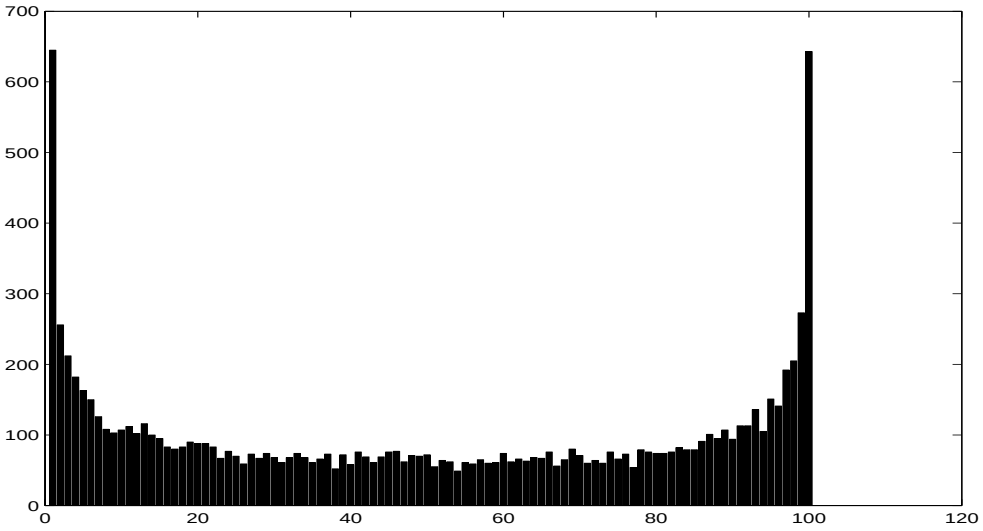


Figure 14.11: Probability density function (with 100 bins) of the output from the logistic map for 10000 iterations.

In terms of cryptanalysis and attack, the multi-algorithmic approach to designing a block cypher discussed here introduces a new 'dimension' to the attack problem. The conventional problem associated with an attack on a symmetric cypher is to search for the private key(s) given knowledge of the algorithm. Here, the problem is to search not only for the session key(s), but the algorithms they 'drive'. One overriding issue concerning cryptology in general, is that algorithm secrecy is weak. In other words, a cryptographic system should not rely of the secrecy of its algorithms and all such algorithms should be openly published.[7]. The system described here is multi-algorithmic, relying on many different chaotic maps to scramble the data. Here, publication of the algorithms can be done in the knowledge that many more maps can be invented as required (subject to appropriate conditions in terms of generating a fully chaotic field with a uniform PDF) by a programmer, or possibly with appropriate 'training' of a digital computer.

14.14.3 Key Exchange and Authentication

The process of 'scrambling' data using PCNGs or PRNGs is just one aspect of cryptography. The other major aspects are (i) key exchange; (ii) authentication. Without developing secure ways of transferring the keys from sender to receiver, there is little

[7]Except for some algorithms developed by certain federal government agencies. Perhaps they have something to hide!

virtue in developing sophisticated methods of 'scrambling'. Further, the ability for a receiver to decrypt a transmission can lead to a false sense of confidence with regard to its content and authentication of a decrypted message is often necessary, particularly when a system is being attacked through the promotion of disinformation for example by searching for a crib, i.e. forcing an encrypted communication whose plaintext in known to have certain key words, phases or quotations for example.

With regard to chaotic block cyphers, one can apply the RSA algorithm discussed earlier, not to encrypt the plaintext, but to encrypt the sessions keys and the algorithm data base. With regard to authentication of a message, one approach is to use a key that is plaintext dependent for which the chirp coding approach discussed in the previous case study can be used (with appropriate modifications). Further, application of chirp coding can be used to transfer a plaintext key in the cypher text, a key that is one of those used to encrypt/decrypt the data, but in contributing to the decryption, provides an authentication of the original plaintext. In effect, provided that appropriate protocols and procedures have been introduced, this approach, not only provides a method of authentication but does so, using a one time pad.

The history and development of encryption is a fascinating subject in its own right and this case study can only provide a brief glimpse into some of the methods employed (using deterministic chaos or otherwise). However, there are some basic concepts that are easy to grasp and sometimes tend to get lost in the detail. The first of these is that the recipient of any encrypted message must have some form of *a priori* knowledge on the method (the algorithm for example) and the operational conditions (the public and/or private keys) used to encrypt a message. Otherwise, the recipient is in no better a 'state of preparation' than the potential attacker. The idea is to keep this *a priori* information to the bare minimum but in such a way that it is super critical to the decryption process. Another important reality is that in an attack, if the information transmitted is not decyphered in good time, then it is typically redundant. Coupled with the fact that an attack usually has to focus on a particular approach (a specific algorithm for example), one way to enhance the security of a communications channel is to continually change the encryption algorithm and/or process offered by the technology currently available. This is one of the most interesting challenges for state control over the 'information society'. The point here, is that as more and more members of the primarily younger generation become increasingly IT literate, it is inevitable that a larger body of perfectly able minds will become aware of the fact that cryptology is not as difficult as some may like to make out. Indeed, the average graduate of today has the ability to write an encryption system which although relatively simple, possibly trivial and ill-informed, can, by the very nature of its non-compliance to international standards, provide surprisingly good security. A problem then occurs with the control of information when increasingly more individuals, groups, companies, agencies and nation states decide that they can 'go it alone' and do it themselves. While each home grown encryption system may be relatively weak compared to those that have had expert development over many years, have been well financed and been tested against the very best of attack strategies, the proliferation of such systems is itself a source of significant difficulty for any authority whose role is to accurately monitor communications traffic in a way that is timely and cost effective. This explains why there have been a number of recent attempts by certain western governments to control the use and exploitation

of new encryption methods in the commercial sector. It also explains why there is so much made of international encryption standards in terms of both public perception and free market exploitation. There is nothing a government and other controlling authorities like better than to preside over a situation in which everybody else is confidently reliant for their information security on products that have been created or cultivated, inspected, tested and of course broken by the very authorities that encourage their use; a use that is covertly 'diffused' into the 'information society' through various legitimate business ventures coupled with all the usual commercial sophistication and investment portfolios. The proliferation of stand alone encryption systems that are designed and used by informed individuals is not only possible but inevitable, an inevitability that is guided by the principle that if you want to know what you are eating then you should cook it yourself. Given time, security issues of this type are likely to become the single most important agenda for future government policy on information technology, especially when such systems have been 'home spun' by those who have learned to fully respect that they should, in the words of Shakespeare, 'Neither a borrower, nor a lender be'.[8]

14.15 Summary of Important Results

Given that

$$s_i = p_i \otimes f_i + n_i$$

and

$$\hat{f}_i = q_i \otimes s_i$$

we have:

Inverse Filter

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2}$$

where P_i is the DFT of the impulse response function. The criterion is that $\|n_i\|_2^2$ is a minimum.

Wiener Filter

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \frac{|F_i|^2}{|N_i|^2}}$$

where F_i and N_i are the DFT's of the input or object function and noise respectively. The criterion is that

$$\|f_i - \hat{f}_i\|_2^2$$

is a minimum and that

$$n_i \odot f_i = 0, \quad \text{and} \quad f_i \odot n_i = 0.$$

Power Spectrum Equalization Filter

$$\mid Q_i \mid = \left(\frac{1}{\mid P_i \mid^2 + \mid N_i \mid^2 / \mid F_i \mid^2} \right)^{1/2}$$

[8] From William Shakespeare's play, Hamlet

The criterion is that

$$\mid F_i \mid^2 = \mid \hat{F}_i \mid^2$$

and that

$$n_i \odot f_i = 0, \quad \text{and} \quad f_i \odot n_i = 0.$$

Matched Filter

$$Q_i = \frac{P_i^*}{\mid N_i \mid^2}$$

The criterion is that

$$\frac{\mid \sum_i Q_i P_i \mid^2}{\sum_i \mid N_i \mid^2 \mid Q_i \mid^2}$$

is a maximum.

Constrained Filter

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \gamma \mid G_i \mid^2}$$

where G_i is the constraining filter and γ is the reciprocal of the Lagrange multiplier. The criterion is that

$$\|g \otimes f\|_2^2$$

is a minimum.

Noise-to-Signal Power Spectrum Ratio

$$\frac{\mid N_i \mid^2}{\mid F_i \mid^2} = \left(\frac{C_i}{C_i'} - 1 \right) \mid P_i \mid^2$$

where C_i is the auto-correlation function of the signal

$$s_i = p_i \otimes f_i + n_i$$

and C_i' is the cross-correlation function of s_i with the signal

$$s_i' = p_i \otimes f_i + n_i'.$$

Noisy Signal Model

$$s(t) = p(t) \otimes f(t) + n(t)$$

where $n(t)$ is the noise which is taken to obey a certain probability density function f, i.e.

$$\Pr[n(t)] = f(n).$$

Basic PRNG (using the linear congruential method)

$$x_{i+1} = a x_i \mathrm{mod} P$$

where for example $a = 7^7$ and $P = 2^{31} - 1$ (a Mersenne prime number) and the output conforms to a uniform distribution over $(0, P)$.

Seed or Key The initial value x_0.

Basic Properties

(i) The output of a PRNG looks random (in the sense that it will produce an output that conforms to a known distribution) and has a cycle length determined by P.

(ii) Given complete knowledge of the PRNG algorithm (other than the seed) and all data from x_0 to x_n, the value x_{n+1} is not predictable.

Additive Generators

$$x_i = (x_{i-a} + x_{i-b} + ... + x_{i-m}) \mathrm{mod} 2^k$$

Linear Feedback Shift Register

$$x_i = (c_1 x_{i-1} + c_2 x_{i-2} + ... + c_m x_{i-m}) \mathrm{mod} 2^k$$

Pseudo Chaotic Number Generators

$$x_{n+1} = f(x_n, p_1, p_2, ...)$$

where f is typically (but not always) a nonlinear function with parameters $p_1, p_2, ...$ of specific (chaos generating) values.

14.16 Further Reading

- Jazinski A, *Stochastic Processes and Filtering Theory*, Academic Press, 1970.
- Robinon E A and Silvia M T, *Digital Signal Processing and Time Series Analysis*, Holden-Day, 1978.
- Kailath T, *Lectures on Kalman and Wiener Filtering Theory*, Springer, 1981.
- Mortensen R E, *Random Signals and Systems*, Wiley, 1987.
- Bateman A and Yates W, *Digital Signal Processing Design*, Pitman, 1988.
- Van Den Enden A W M and Verhoeckx N A M, *Discrete Time Signal Processing*, Prentice-Hall, 1989.

- Inmos Limited, *Digital Signal Processing*, Prentice-Hall, 1989.
- Brown R G and Hwang P Y C, *Introduction to Random Signals and Applied Kalman Filtering*, Wiley, 1992.
- Press W H, Teukolsky S A, Vetterling W T and Flannery, B P, *Numerical Recipes in C*, Cambridge University Press, 1994
- Korsch H J and Jodl H J, *Chaos: A Program Collection for the PC*, Springer, 1994.
- Woolfson M M and Pert G J, *An Introduction to Computer Simulation*, Oxford University Press, 1999.

14.17 Programming Problems

In the questions that follow, the functions required should be void functions written in ANSI C. They should be compiled, tested and then added to a digital signal processing object library *dsplib.lib* say. In each case, a simple I/O test procedure should be written, the I/O being studied by plotting the signal at appropriate points in the process working with arrays of size 64, 128, 256 or 512 samples. Each function should be self-contained within the context of the DSP algorithm to be coded. In each case, n (which is of type integer) is the size of the array which should be processed from 1 to n.

14.1 Write a function to filter a signal using an ideal lowpass filter.

```
function ILF(float s[ ], int n, int bw)
```

where f is the I/O and bw is the bandwidth

14.2 Write a function to filter a signal using a Gaussian lowpass filter.

```
function GLF(float s[ ], int n, int bw)
```

where s is the I/O and bw is the bandwidth

14.3 Write a function to filter a signal using a Butterworth lowpass filter.

```
function BLF(float s[ ], int n, int bw, int ord)}
```

where s is the I/O, bw is the bandwidth and ord is the order of the filter.

14.4 Write functions to filter a signal using high pass versions of the filters discussed in questions 14.1-14.3 above.

```
function IHF(float s[ ], int n, int bw)
function GHF(float s[ ], int n, int bw)
function BHF(float s[ ], int n, int bw)
```

14.5 A random fractal signal is a statistically self-affine signal which is characterized by a transfer function of the form $(i\omega)^{-q}, \omega \neq 0$ where $q \in (0,1)$ is the Fourier dimension (as discussed further in Chapter 17). Thus, a random fractal signal can be generated by filtering white noise. Using the Gaussian random number generator GNOISE, write a function to generate a random fractal signal.

```
function FRACTAL(float s[ ], int n, int seed, long float q)
```

where s is the output (the fractal signal), seed is the random number generators initial value and q is the Fourier dimension.

14.6 Write a function to process (restore) a signal using the Wiener filter.

```
function WIENER(float s[ ], float p[ ], float f[ ], int n, long float snr)
```

where s is the digital signal (input), p is the IRF (input), f is the restored signal (output) and snr is the signal-to-noise ratio.

14.7 Using the functions SPIKES, GAUSSIAN and CONVOLVE designed in Chapter 13 (programming problems) and working with array sizes of 128, convolve two spikes of width 10 with a Gaussian of arbitrary width. Adjust the width of the Gaussian to give an output where the location of the spikes is just visible. Use the function WIENER to recover the object function f (i.e. the two spikes) using a range of values for snr (e.g. $1 < \text{snr} < 100$). Repeat this process using two spikes of unit amplitude but opposite polarity.

14.8 Repeat question 14.7 above using function GNOISE to add random Gaussian noise to the output (after convolution with the Gaussian) with a variable signal-to-noise ratio defined as

$$\text{snr} = \frac{\|p_i \otimes f_i\|_\infty}{\|\text{n}_\text{i}\|_\infty}$$

One way of doing this is to first rescale $p_i \otimes f_i$ and n_i so that both arrays have an upper bound (in modulus) of 1 and then construct the signal s_i via the equation

$$s_i = (p_i \otimes f_i) \times \text{snr} + n_i$$

or alternatively

$$s_i = p_i \otimes f_i + n_i/\text{snr}; \quad \text{snr} > 0.$$

14.9 Write a function to match filter a given input using FFT1D under the assumption that the noise is white (power spectrum is unity over the bandwidth of the signal).

```
function MATFIL(float s[ ], float p[ ], float f[ ], int n)
```

where s is the input signal, p is the IRF (input) and f is the matched filtered output.

14.10 Working with 512 arrays use functions SPIKES and CONVOLVE to convolve two spikes of width 30 with a linear FM instrument function ('chirped sinusoid') of the form

$$p_i = \sin(2\pi i^2/N); \quad i = 0, 1, ...N-1$$

where N is the width of the pulse ($\leq$ 128). Use function MATFIL to recover the two spikes and study the effect of changing the width of the instrument function (i.e. changing the value of N). Investigate the application of the matched filter using additive Gaussian noise as discussed with regard to application of the Wiener filter (i.e. Question 14.8).

Chapter 15

Statistics, Entropy and Extrapolation

The processes discussed so far (i.e. in the previous chapter) have not taken into account the statistical nature of the signal and in particular, the statistics of the (additive) noise. To do this, another type of approach needs to be considered which is based on Bayesian estimation. Bayesian estimation allows digital filters to be constructed whose performance is determined by various (statistical) parameters which can be determined (approximately) from the statistics of the data. After discussing Bayesian methods, this chapter covers the principles associated with using the entropy as a criterion for extracting information from noise and finally, studies the problem of extrapolating a bandlimited spectrum which occurs when we are required to deconvolve data whose impulse response function is a sinc.

15.1 Bayes Rule

Suppose we toss a coin, observe whether we get heads or tails, and then repeat this process a number of times. As the number of trials increases, we expect that the number of times heads or tails occurs is half that of the number of trials. In other words, the probability of getting heads is $1/2$ and the probability of getting tails is also $1/2$. Similarly, if a cubic dice with six faces is thrown repeatedly, then the probability of it landing on any one particular face is $1/6$. In general, if an experiment is repeated N times and an event A occurs n times, then the probability of this event $P(A)$ is defined as

$$P(A) = \lim_{N\to\infty} \left(\frac{n}{N}\right).$$

The probability is the relative frequency of an event as the number of trials tends to infinity. However, in practice, only a finite number of trials can be conducted and we therefore define the (experimental) probability of an event A as

$$P(A) = \frac{n}{N}$$

where N is assumed to be large. Nevertheless, the results that follow are only strictly valid under the limiting condition that $N \to \infty$, just as, for example, a delta sequence $S_n(t)$ is only strictly valid under the condition that

$$\lim_{n\to\infty} \int_{-\infty}^{\infty} S_n(t) f(t) dt = \int_{-\infty}^{\infty} \delta(t) f(t) dt = f(0).$$

Suppose we have two coins which we label C_1 and C_2. We toss both coins simultaneously N times and record the number of times C_1 is heads, the number of times C_2 is heads and the number of times C_1 and C_2 are heads together. What is the probability that C_1 and C_2 are heads together ? Clearly, if m is the number of times out of N trials that heads occurs simultaneously, then the probability of such an event must be given by

$$P(C_1 \text{ heads and } C_2 \text{ heads}) = \frac{m}{N}.$$

This is known as the joint probability of C_1 being heads when C_2 is heads. In general, if two events A and B are possible and m is the number of times both events occur simultaneously, then the joint probability is given by

$$P(A \text{ and } B) = \frac{m}{N}.$$

Now suppose we setup an experiment in which two events A and B can occur. We conduct N trials and record the number of times A occurs (which is n) and the number of times A and B occur simultaneously (which is m). In this case, the joint probability may written as

$$P(A \text{ and } B) = \frac{m}{N} = \frac{m}{n} \times \frac{n}{N}$$

Now, the quotient n/N is the probability $P(A)$ that event A occurs. The quotient m/n is the probability that events A and B occur simultaneously given that event A has already occurred. The latter probability is known as the conditional probability and is written as

$$P(B \mid A) = \frac{m}{n}$$

where the symbol $B \mid A$ means B 'given' A. Hence, the joint probability can be written as

$$P(A \text{ and } B) = P(A)P(B \mid A).$$

Suppose we undertake an identical experiment but this time we record the number of times p that event B occurs and the number of times q that event B occurs simultaneously with event A. In this case, the joint probability of events B and A occurring together is given by

$$P(B \text{ and } A) = \frac{q}{N} = \frac{q}{p} \times \frac{p}{N}.$$

The quotient p/N is the probability $P(B)$ that event B occurs and the quotient q/p is the probability of getting events B and A occurring simultaneously given that event

B has already occurred. The latter probability is just the probability of getting A 'given' B, i.e.

$$P(A \mid B) = \frac{q}{p}.$$

Hence, we have

$$P(B \text{ and } A) = P(B)P(A \mid B).$$

Now, the probability of getting A and B occurring simultaneously is exactly the same as getting B and A occurring simultaneously, i.e.

$$P(A \text{ and } B) = P(B \text{ and } A).$$

By using the definition of these joint probabilities in terms of the conditional probabilities we arrive at the following formula:

$$P(A)P(B \mid A) = P(B)P(A \mid B)$$

or alternatively

$$P(B \mid A) = \frac{P(B)P(A \mid B)}{P(A)}.$$

This result is known as Bayes rule. It relates the conditional probability of 'B given A' to that of 'A given B'.

15.1.1 Bayesian Estimation

In signal analysis, Bayes rule is written in the form

$$P(f \mid s) = \frac{P(f)P(s \mid f)}{P(s)}$$

where f is the information we want to recover from the signal which is assumed (as usual) to be the result of a time invariant stationary process and given by

$$s = p \otimes f + n$$

where p is the impulse response function and n is the noise. This result is the basis for a class of filters which are known collectively as Bayesian estimators. In simple terms, Bayesian estimation attempts to recover f in such a way that the probability of getting f given s is a maximum. In practice, this is done by assuming that $P(f)$ and $P(s \mid f)$ obey certain statistical distributions which are consistent with the experiment in which s is measured. In other words, models are chosen for $P(f)$ and $P(s \mid f)$ and then f is computed at the point where $P(f \mid s)$ reaches its maximum value. This occurs when

$$\frac{\partial}{\partial f} P(f \mid s) = 0.$$

The function P is the Probability Density Functions or PDF. The PDF $P(f \mid s)$ is called the *a posteriori* PDF. Since the logarithm of a PDF varies monotonically with that PDF, the *a posteriori* PDF is also a maximum when

$$\frac{\partial}{\partial f} \ln P(f \mid s) = 0.$$

Using Bayes rule, we can write this equation as

$$\frac{\partial}{\partial f} \ln P(s \mid f) + \frac{\partial}{\partial f} \ln P(f) = 0.$$

Because the solution to this equation for f maximizes the *a posteriori* PDF, this method is known as the maximum *a posteriori* or MAP method.

15.1.2 Some Simple Examples of Bayesian Estimation

Suppose we measure a single sample s (one real number) in an experiment where it is known *a priori* that

$$s = f + n$$

where n is noise (a random number). Suppose that it is also known *a priori* that the noise is determined by a zero mean Gaussian distribution of the form

$$P(n) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp(-n^2/2\sigma_n^2)$$

where σ_n is the standard deviation of the noise. The probability of measuring s given f - the conditional probability $P(s \mid f)$ - is determined by the noise since

$$n = s - f$$

and we can therefore write

$$P(s \mid f) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp[-(s-f)^2/2\sigma_n^2].$$

To find the MAP estimate, the PDF for f must also be known. Suppose that f also has a zero-mean Gaussian distribution of the form

$$P(f) = \frac{1}{\sqrt{2\pi\sigma_f^2}} \exp(-f^2/2\sigma_f^2).$$

Then,

$$\frac{\partial}{\partial f} \ln P(s \mid f) + \frac{\partial}{\partial f} \ln P(f) = \frac{(s-f)}{\sigma_n^2} - \frac{f}{\sigma_f^2} = 0.$$

Solving this equation for f gives

$$f = \frac{s\Gamma^2}{1+\Gamma^2}$$

where Γ is the signal-to-noise ratio defined by

$$\Gamma = \frac{\sigma_f}{\sigma_n}.$$

Note that as $\sigma_n \to 0$, $f \to s$ which must be true since $s = f + n$ and n has a zero-mean Gaussian distribution. Also note that the solution we acquire for f is entirely

dependent on the prior information we have on the PDF for f. A different PDF produces an entirely different solution. To illustrate this, let us suppose that f obeys a Rayleigh distribution of the form

$$P(f) = \begin{cases} \frac{f}{\sigma_f^2} \exp(-f^2/2\sigma_f^2), & f \geq 0; \\ 0, & f < 0. \end{cases}$$

In this case,

$$\frac{\partial}{\partial f} \ln P(f) = \frac{1}{f} - \frac{f}{\sigma_f^2}$$

and we get (still assuming that the noise obeys the same zero-mean Gaussian distribution)

$$\frac{(s-f)}{\sigma_n^2} + \frac{1}{f} - \frac{f}{\sigma_f^2} = 0.$$

This equation is quadratic in f and its solution is

$$f = \frac{s\Gamma^2}{2(1+\Gamma^2)} \left[1 \pm \sqrt{1 + \frac{4\sigma_n^2}{s^2\Gamma^2}\left(1 + \frac{1}{\Gamma^2}\right)} \right].$$

The solution for f which maximizes the value of $P(f \mid s)$ can then be written in the form

$$f = \frac{s}{2a}\left(1 + \sqrt{1 + \frac{4a\sigma_n^2}{s^2}}\right)$$

where

$$a = 1 + \frac{1}{\Gamma^2}.$$

Note that if

$$\frac{2\sigma_n\sqrt{a}}{s} << 1$$

then

$$f \simeq \frac{s}{a}$$

which is identical to the MAP estimate obtained earlier where it was assumed that f was Gaussian distributed.

15.1.3 The Maximum Likelihood Estimation

From the previous example, it is clear that the MAP estimate is only as good as the prior information on the statistical behaviour of f - the object for which we seek a solution. When $P(f)$ is broadly distributed compared with $P(s \mid f)$, we can apply a further approximation. In particular, if $P(f)$ is roughly constant, then $\partial \ln P/\partial f$ is close to zero and therefore

$$\frac{\partial}{\partial f} \ln P(f \mid s) \simeq \frac{\partial}{\partial f} \ln P(s \mid f).$$

In this case, the *a posteriori* PDF is a maximum when

$$\frac{\partial}{\partial f} \ln P(s \mid f) = 0.$$

The estimate for f that is obtained by solving this equation for f is called the maximum likelihood or ML estimate. To obtain this estimate, only prior knowledge on the statistical fluctuations of the conditional probability is required. If, as in the previous example, we assume that the noise is a zero-mean Gaussian distribution, then the ML estimate is given by

$$f = s.$$

Note that this is the same as the MAP estimate when the standard deviation of the noise is zero.

ML .v. MAP Estimations

The basic difference between the MAP and ML estimates is that the ML estimate ignores prior information about the statistical fluctuations of the object f. ML estimation only requires a model for the statistical fluctuations of the noise. For this reason, the ML estimate is usually easier to compute. It is also the estimate to use in cases where there is a complete lack of knowledge about the statistical behaviour of the object. To further illustrate the difference between the MAP and ML estimate and as prelude to their use in signal analysis, consider the case where we measure N samples of a real signal s_i in the presence of additive noise n_i which is the result of transmitting a known signal f_i modified by a random amplitude factor a, the samples of the signal being given by

$$s_i = af_i + n_i; \quad i = 1, 2, ..., N.$$

The problem is to find an estimate for a. To solve problems of this type, using Bayesian estimation, we need to introduce multi-dimensional probability theory. In this case, the PDF is a function of not just one number s but the set of numbers $s_1, s_2, ..., s_N$, i.e. a vector space. To emphasize this, we use the vector notation

$$P(\mathbf{s}) \equiv P(s_1, s_2, s_3, ..., s_N).$$

The ML estimate is given by solving the equation

$$\frac{\partial}{\partial a} \ln P(\mathbf{s} \mid a) = 0$$

for a where

$$P(\mathbf{s} \mid a) \equiv P(s_1, s_2, ..., s_N \mid a).$$

Let us assume that the noise is described by a zero-mean Gaussian distribution of the form

$$P(\mathbf{n}) \equiv P(n_1, n_2, ..., n_N) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \sum_{i=1}^{N} n_i^2\right).$$

The conditional probability is then given by

$$P(\mathbf{s} \mid a) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(s_i - af_i)^2\right)$$

and

$$\frac{\partial}{\partial a}\ln P(\mathbf{s} \mid a) = \frac{1}{\sigma_n^2}\sum_{i=1}^{N}(s_i - af_i)f_i = 0.$$

Solving this last equation for a we obtain the ML estimate

$$a = \frac{\sum\limits_{i=1}^{N} s_i f_i}{\sum\limits_{i=1}^{N} f_i^2}.$$

The MAP estimate is obtained by solving the equation

$$\frac{\partial}{\partial a}\ln P(\mathbf{s} \mid a) + \frac{\partial}{\partial a}\ln P(a) = 0$$

for a. Using the same distribution for the conditional PDF, let us assume that a has a zero-mean Gaussian distribution of the form

$$P(a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp(-a^2/2\sigma_a^2)$$

where σ_a^2 is the standard deviation of a. In this case,

$$\frac{\partial}{\partial a}\ln P(a) = -\frac{a}{\sigma_a^2}$$

and hence, the MAP estimate is obtained by solving the equation

$$\frac{\partial}{\partial a}\ln P(\mathbf{s} \mid a) + \frac{\partial}{\partial a}\ln P(a) = \frac{1}{\sigma_n^2}\sum_{i=1}^{N}(s_i - af_i)f_i - \frac{a}{\sigma_a^2} = 0$$

for a. The solution to this equation is given by

$$a = \frac{\frac{\sigma_a^2}{\sigma_n^2}\sum\limits_{i=1}^{N} s_i f_i}{1 + \frac{\sigma_a^2}{\sigma_n^2}\sum\limits_{i=1}^{N} f_i^2}.$$

Note that if $\sigma_a >> \sigma_n$, then,

$$a \simeq \frac{\sum\limits_{i=1}^{N} s_i f_i}{\sum\limits_{i=1}^{N} f_i^2}$$

which is the same as the ML estimate.

15.2 The Maximum Likelihood Method

The maximum likelihood method uses the principles of Bayesian estimation discussed above to design deconvolution algorithms or filters. The problem is as follows (where all discrete functions are assumed to be real): Given the digital signal

$$s_i = \sum_j p_{i-j} f_j + n_i,$$

find an estimate for f_i when p_i is known together with the statistics for n_i. The ML estimate for f_i is determined by solving the equation

$$\frac{\partial}{\partial f_k} \ln P(s_1, s_2, .., s_N \mid f_1, f_2, ..., f_N) = 0.$$

As before, the algebraic form of the estimate depends upon the model that is chosen for the PDF. Assume that the noise has a zero-mean Gaussian distribution. In this case, the conditional PDF is given by

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \sum_i (s_i - \sum_j p_{i-j} f_j)^2 \right)$$

where σ_n is the standard deviation of the noise. Substituting this result into the previous equation and differentiating, we get (see Chapter 8 - the orthogonality principle)

$$\frac{1}{\sigma_n^2} \sum_i \left(s_i - \sum_j p_{i-j} f_j \right) p_{i-k} = 0$$

or

$$\sum_i s_i p_{i-k} = \sum_i \left(\sum_j p_{i-j} f_j \right) p_{i-k}.$$

Using the appropriate symbols, we may write this equation in the form

$$s_n \odot p_n = (p_n \otimes f_n) \odot p_n$$

where $\odot$ and $\otimes$ denote the correlation and convolution sums respectively. The ML estimate is obtained by solving the equation above for f_n. This can be done by transforming it into Fourier space. Using the correlation and convolution theorems, in Fourier space this equation becomes

$$S_m P_m^* = (P_m F_m) P_m^*$$

and thus

$$f_n = \text{IDFT}(F_m) = \text{IDFT}\left(\frac{S_m P_m^*}{\mid P_m \mid^2} \right)$$

where IDFT denotes the inverse DFT. Thus, for Gaussian statistics, the ML filter is given by

$$\text{ML filter} = \frac{P_m^*}{\mid P_m \mid^2}$$

which is identical to the inverse filter.

15.3 Maximum a Posteriori Method

This method is based on computing f_i such that

$$\frac{\partial}{\partial f_k} \ln P(s_1, s_2, ..., s_n \mid f_1, f_2, ..., f_n) + \frac{\partial}{\partial f_k} \ln P(f_1, f_2, ..., f_n) = 0.$$

Consider the following models for the PDF's:

(i) Zero-mean Gaussian statistics for the noise,

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \sum_i \mid s_i - \sum_j p_{i-j} f_j \mid^2 \right).$$

(ii) Zero-mean Gaussian distribution for the object

$$P(\mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_f^2}} \exp\left(-\frac{1}{2\sigma_f^2} \sum_i \mid f_i \mid^2 \right)$$

where for generality, we also assume that the data may be complex. By substituting these expressions for $P(\mathbf{s} \mid \mathbf{f})$ and $P(\mathbf{f})$ into the equation above, we obtain (using the orthogonality principle)

$$\frac{1}{\sigma_n^2} \sum_i \left(s_i - \sum_j p_{i-j} f_j \right) p^*_{i-k} - \frac{1}{\sigma_f^2} f_k = 0.$$

Rearranging, we may write this result in the form

$$s_n \odot p_n^* = \frac{\sigma_n^2}{\sigma_f^2} f_n + (p_n \otimes f_n) \odot p_n^*.$$

In Fourier space, this equation becomes

$$S_m P_m^* = \frac{1}{\Gamma^2} F_m + \mid P_m \mid^2 F_m.$$

The MAP filter for Gaussian statistics is therefore given by

$$\text{MAP} \quad \text{filter} = \frac{P_m^*}{\mid P_m \mid^2 + 1/\Gamma^2}$$

where

$$\Gamma = \frac{\sigma_f}{\sigma_n}$$

which defines the signal-to-noise ratio. Note that this filter is the same as the Weiner filter under the assumption that the power spectra of the noise and object are constant. Also, note that

$$\text{MAP} \quad \text{filter} \rightarrow \text{ML} \quad \text{filter} \quad \text{as} \quad \sigma_n \rightarrow 0.$$

The algebraic form of this filter is based on the assumption that the noise is Gaussian. For PDF's of other forms, the computation of the filter can become more complicated. Note that in practice, the value of σ_n can be obtained by recording the output of a system with no input. The output is then noise driven and a histogram can be computed from this output noisefield from which an estimate of σ_n can be computed using a least squares fit to the function $\exp(-n^2/2\sigma_n^2)$ for example. Bayesian estimation is a very useful approach for the extraction of 'signals from noise' when accurate statistical information on the signal and noise are available and has a wide range of applications other than those that have been focused on here. However, the method often leads to estimates that are idealized 'middle of the road' solutions and there is some truth to the observation that with Bayesian statistics, one sees a horse, thinks of a donkey and ends up with a mule!

15.4 The Maximum Entropy Method

The entropy of a system describes its disorder; it is a measure of the lack of information about the exact state of a system. Information is a measure of order, a universal measure applicable to any structure or any system. It quantifies the instructions that are needed to produce a certain organisation. There are several ways in which one can quantify information but a specially convenient one is in terms of binary choices. In general, we compute the information inherent in any given arrangement from the number of choices we must make to arrive at that particular arrangement among all possible ones. Intuitively, the more arrangements that are possible, the more information that is required to achieve a particular arrangement.

15.4.1 Information and Entropy

Consider a simple linear array such as a deck of eight cards which contains the ace of diamonds for example and where we are allowed to ask a series of sequential questions as to where in the array the card is. The first question we could ask is in which half of the array does the card occur which reduces the number of cards to four. The second question is in which half of the remaining four cards is the ace of diamonds to be found leaving just two cards and the final question is which card is it. Each successive question is the same but applied to successive subdivisions of the deck and in this way we obtain the result in three steps regardless of where the card happens to be in the deck. Each question is a binary choice and in this example, 3 is the minimum number of binary choices which represents the amount of information required to locate the card in a particular arrangement. This is the same as taking the binary logarithm of the number of possibilities, since $\log_2 8 = 3$. Another way of appreciating this result, is to consider a binary representation of the array of cards, i.e. 000,001,010,011,100,101,110,111, which requires three digits or bits to describe any one card. If the deck contained 16 cards, the information would be 4 bits and if it contained 32 cards, the information would be 5 bits and so on. Thus, in general, for any number of possibilities N, the information I for specifying a member in such a linear array, is given by

$$I = -\log_2 N = \log_2 \frac{1}{N}$$

where the negative sign is introduced to denote that information has to be acquired in order to make the correct choice, i.e. I is negative for all values of N larger than 1. We can now generalize further by considering the case where the number of choices N are subdivided into subsets of uniform size n_i. In this case, the information needed to specify the membership of a subset is given not by N but by N/n_i and hence, the information is given by

$$I_i = \log_2 p_i$$

where $p_i = n_i/N$ which is the proportion of the subsets. Finally, if we consider the most general case, where the subsets are nonuniform in size, then the information will no longer be the same for all subsets. In this case, we can consider the mean information given by

$$I = \sum_i p_i \log_2 p_i$$

which is the Shannon entropy measure established in his classic works on information theory in the 1940s. Information, as defined here, is a dimensionless quantity. However, its partner entity in physics has a dimension called 'Entropy' which was first introduced by Ludwig Boltzmann as a measure of the dispersal of energy, in a sense, a measure of disorder, just as information is a measure of order. In fact, Boltmann's entropy concept has the same mathematical roots as Shannon's information concept in terms of computing the probabilities of sorting objects into bins (a set of N into subsets of size n_i) and in statistical mechanics the entropy is defined as

$$E = -k \sum_i p_i \ln p_i$$

where k is Boltzmann's constant ($=3.2983 \times 10^{-24}$Calories/^oC). Shannon's and Boltmann's equations are similar. E and I have opposite signs, but otherwise differ only by their scaling factors and they convert to one another by $E = -(k \ln 2)I$. Thus, an entropy unit is equal to $-k \ln 2$ of a bit. In Boltzmann's equation, the probabilities p_i refer to internal energy levels. In Shannon's equations p_i are not *a priori* assigned such specific roles and the expression can be applied to any physical system to provide a measure of order. Thus, information becomes a concept equivalent to entropy and any system can be described in terms of one or the other. An increase in entropy implies a decrease of information and vise versa. This gives rise to the fundamental conservation law:

The sum of (macroscopic) information change and entropy change in a given system is zero.

In signal analysis, the entropy of a signal is a measure of the lack of information about the exact information content of the signal, i.e. the precise value of f_i for a given i. Thus, noisy signals have a larger entropy. The general definition for the entropy of a system E is

$$E = -\sum_i p_i \ln p_i$$

where p_i is the probability that the system is in a state i. The negative sign is introduced because the probability is a value between 0 and 1 and therefore, $\ln p_i$ is a value between 0 and $-\infty$, but the Entropy is by definition, a positive value.

Maximum entropy deconvolution is based on modelling the entropy of the information input or the object function f_i. A reconstruction for f_i is found such that

$$E = -\sum_i f_i \ln f_i$$

is a maximum which requires that $f_i > 0 \forall i$. Note that the function $x \ln x$ has a single local minimum value between 0 and 1 whereas the function $-x \ln x$ has a single local maximum value. It is a matter of convention as to whether a criteria of the type

$$E = \sum_i f_i \ln f_i$$

or

$$E = -\sum_i f_i \ln f_i$$

is used leading to (strictly speaking) a minimum or maximum entropy criterion respectively. In some ways, the term 'maximum entropy' is misleading because it implies that we are attempting to recover information from noise with minimum information content and the term 'minimum entropy' conveys a method that is more consistent with the philosophy of what is being attempted, i.e. to recover useful and unambiguous information from a signal whose information content has been distorted by (additive) noise. For example, suppose we input a binary stream into some time invariant linear system, where $\mathbf{f} = (...010011011011101...)$. Then, the input has an entropy of zero since $0 \ln 0 = 0$ and $1 \ln 1 = 0$. We can expect the output of such a system to generate floating point values (via the convolution process) which are then perturbed through additive noise. The output $s_i = p_i \otimes f_i + n_i$ (where it is assumed that $s_i > 0 \forall i$) will therefore have an entropy that is greater than 0. Clearly, as the magnitude of the noise increases, so, the value of the entropy increases leading to greater loss of information on the exact state of the input (in terms of f_i, for some value of i being 0 or 1). With the deconvolution process, we ideally want to recover the input without any bit-errors. In such a hypothetical case, the entropy of the restoration would be zero just as a least squares error would be. However, just as we can seek a solution in which the least squares error is a minimum, so we can attempt a solution in which the entropy is a minimum in the case when we define it as

$$E = \sum_i f_i \ln f_i$$

or a maximum in the case when we define the entropy as

$$E = -\sum_i f_i \ln f_i.$$

15.4.2 Maximum Entropy Deconvolution

Given the signal equation

$$s_i = p_i \otimes f_i + n_i$$

we find f_i such that the entropy E, defined by

$$E = -\sum_i f_i \ln f_i$$

is a maximum. Note that because the ln function enters in to this argument, the maximum entropy method must be restricted to cases where f_i is real and positive. Hence, the method can not be applied to an original (dual-polarity) signal but to its amplitude modulations for example.

From the signal equation, we can write

$$s_i - \sum_j p_{i-j} f_j = n_i$$

where we have written the (digital) convolution operation out in full. Squaring both sides and summing over i, we can write

$$\sum_i \left(s_i - \sum_j p_{i-j} f_j \right)^2 - \sum_i n_i^2 = 0.$$

Now, this equation holds for any constant λ (the Lagrange multiplier) which is a multiple of the left hand side. We can therefore write the equation for E as

$$E = -\sum_i f_i \ln f_i - \lambda \left[\sum_i \left(s_i - \sum_j p_{j-i} f_j \right)^2 - \sum_i n_i^2 \right]$$

because the second term on the right hand side is zero anyway (for all values of λ). Given this equation, our problem is to find f_i such that the entropy E is a maximum, i.e.

$$\frac{\partial E}{\partial f_k} = 0 \quad \forall k.$$

Differentiating and switching to the notation for 1D convolution $\otimes$ and 1D correlation $\odot$, we find that E is a maximum when

$$-1 - \ln f_i + 2\lambda(s_i \odot p_i - p_i \otimes f_i \odot p_i) = 0$$

or, after rearranging,

$$f_i = \exp[-1 + 2\lambda(s_i \odot p_i - p_i \otimes f_i \odot p_i)].$$

This equation is transcendental in f_i and as such, requires that f_i is evaluated iteratively, i.e.

$$f_i^{n+1} = \exp[-1 + 2\lambda(s_i \odot p_i - p_i \otimes f_i^n \odot p_i)], \quad n = 1, 2, ..., N.$$

The rate of convergence of this solution is determined by the value of the Lagrange multiplier given an initial estimate of f_i, (i.e. f_i^0) in a way that is analogous to the use of a relaxation parameter for solving the equation $\mathbf{x} = M\mathbf{x} + \mathbf{c}$ iteratively (see Chapter 9).

15.4.3 Linearization

The iterative nature of this nonlinear estimation method may be undesirable, primarily because it is time consuming and may require many iterations before a solution is achieved with a desired tolerance. The MEM can be linearized by retaining the first two terms (i.e. the linear terms) in the series representation of the exponential function leaving us with the following equation

$$f_i = 2\lambda(s_i \odot p_i - p_i \otimes f_i \odot p_i).$$

Using the convolution and correlation theorems, in Fourier space, this equation becomes

$$F_i = 2\lambda S_i P_i^* - 2\lambda \mid P_i \mid^2 F_i$$

which after rearranging gives

$$F_i = \frac{S_i P_i^*}{\mid P_i \mid^2 + \frac{1}{2\lambda}}.$$

Thus, we can define a linearized maximum entropy filter of the form

$$\frac{P_i^*}{\mid P_i \mid^2 + \frac{1}{2\lambda}}.$$

Note that this filter is very similar to the Wiener filter. The only difference is that the Wiener filter is regularized by a constant determined by the SNR of the data whereas this filter is regularized by a constant determined by the Lagrange multiplier.

15.4.4 The Cross Entropy Method

The cross entropy or Patterson entropy as it is sometimes referred to, uses a criterion in which the entropy measure

$$E = -\sum_i f_i \ln\left(\frac{f_i}{w_i}\right)$$

is maximized where w_i is some weighting function based on any available *a priori* information of the structure of f_i. If the computation described earlier is re-worked using this definition of the cross entropy, then we obtain the result

$$f_i = w_i \exp[-1 + 2\lambda(s_i \odot p_i - p_i \otimes f_i \odot p_i)].$$

The cross entropy method has a synergy with the Wilkinson test in statistics in which a discrete PDF or histogram P_i say of a stochastic field p_i is tested against a histogram Q_i representative of a stochastic field q_i. A standard test to quantify how close the stochastic behaviour of p_i is to q_i (the null-hypothesis test) is to use the Chi-squared test in which we compute

$$\chi^2 = \sum_i \left(\frac{P_i - Q_i}{Q_i}\right)^2.$$

The Wilkinson test uses the metric

$$E = -\sum_i P_i \ln\left(\frac{P_i}{Q_i}\right).$$

15.5 Spectral Extrapolation

The effect of deconvolving a signal is to recover the information it contains by compensating for the effect of the instrument function or Impulse Response Function (IRF). The resolution of the information obtained by this process is determined by the bandwidth of the data which in turn, is controlled by the finite frequency response of the system and is a characteristic of the Transfer Function (the Fourier transform of the IRF). Spectral extrapolation is a process which attempts to overcome the limited resolving power of an instrument by designing algorithms which extrapolate the complex spectrum of the information from a finite sample of data.

Bandlimited Functions

A bandlimited function is a function whose spectral bandwidth is finite and nearly all real signals of practical significance are bandlimited functions. The bandwidth determines the resolution of a signal. This leads one to consider the problem of how the bandwidth and hence the resolution of the signal, can be increased synthetically. In other words, how can we extrapolate the spectrum of a bandlimited function from an incomplete sample. Solutions to this type of problem are important in signal analysis when a resolution is required that is not a characteristic of the signal provided and is difficult or even impossible to achieve experimentally. The type of resolution obtained by extrapolating the spectrum of a bandlimited function is sometimes referred to as super resolution.

Formulation of the Problem

The basic problem is an inverse problem. In its simplest form, it is concerned with the inversion of the integral equation

$$s(t) = \int_{-\infty}^{\infty} f(\tau) \frac{\sin[\Omega(t-\tau)]}{(t-\tau)} d\tau$$

for f where Ω determines the bandwidth of the signal s and hence the resolution of f. The equation above is a convolution over the interval $[-\infty, \infty]$. Hence, we may view our problem (i.e. the super resolution problem) in terms of deconvolving s to recover the object f in the special case when the IRF is a sinc function.

Eigenfunction Solution

In practice, signals have a finite duration and so

$$f(t) = 0, \quad | \, t \, | > T.$$

In this case, we can restrict the convolution integral to the interval $[-T, T]$ and model the signal as

$$s(t) = \int_{-T}^{T} f(\tau) \frac{\sin[\Omega(t-\tau)]}{(t-\tau)} d\tau.$$

The object function $f(t)$ can be expressed in the following form

$$f(t) = \sum_{n=0}^{\infty} \lambda_n^{-1} \left[\int_{-T}^{T} s(\tau)\phi_n(\tau)d\tau \right] \phi_n(t)$$

where the eigenfunctions ϕ_n are the prolate spheroidal wave functions given by the solution to the equation

$$\int_{-T}^{T} \phi_n(\tau) \frac{\sin[\Omega(t-\tau)]}{\pi(t-\tau)} d\tau = \lambda_n \phi_n(t)$$

and λ_n are the associated eigenvalues. Like other theoretical inverse solutions, this solution is extremely sensitive to errors in measuring s (i.e. experimental noise). It is therefore often difficult to achieve a stable solution using this method with real signals.

Solution by Analytic Continuation

Using the convolution theorem, we can write our equation in Fourier space as

$$S(\omega) = H(\omega)F(\omega), \quad |\,\omega\,| \leq \infty$$

where

$$H(\omega) = \begin{cases} \frac{1}{2}, & |\,\omega\,| \leq \Omega; \\ 0, & |\,\omega\,| > \Omega. \end{cases}$$

or alternatively

$$S(\omega) = \begin{cases} \frac{1}{2}F(\omega), & |\,\omega\,| \leq \Omega; \\ 0, & \text{otherwise.} \end{cases}$$

Here, S and F are the Fourier transforms of s and f respectively. In this form, our problem is to recover $F(\omega)$ for all values of ω from $S(\omega)$. Because f has finite support, its spectrum is analytic and can therefore, in principle, be analytically continued beyond $[-\Omega, \Omega]$ to provide higher resolution. This can be done by computing the Taylor series for F, i.e.

$$F(\omega) = \sum_{n=0}^{\infty} F^{(n)}(0) \frac{\omega^n}{n!},$$

The derivatives $F^{(n)}$ of F at $\omega = 0$ can be determined from the finite segment $F(\omega)$, $|\,\omega\,| \leq \Omega$ which is equal to S. Hence, we can write

$$F(\omega) = \sum_{n=0}^{\infty} S^{(n)}(0) \frac{\omega^n}{n!}$$

This method of extrapolation is known as analytic continuation. Once again, although of theoretical interest, in practice, this method is fraught with problems. First, it not possible to evaluate $S^{(n)}(0)$ accurately when the signal is noisy. Secondly, the

truncation of the Taylor series (which is necessary in practice) yields large errors for large ω, and since knowledge of $F(\omega)$ is required for all values of ω, errors of this kind are unacceptable. Thus, in practice, analytic continuation fails even in the presence of small amounts of noise.

Re-evaluation of the Problem

There are two important features of the equation

$$s(t) = \int_{-T}^{T} f(\tau) \frac{\sin[\Omega(t-\tau)]}{(t-\tau)} d\tau$$

and therefore its inversion, which in practice are entirely unsuitable: (i) It is assumed that the signal s can be measured without any experimental error; (ii) it is assumed that all the functions are continuous. In practice, we are usually provided with a digital signal which is a discrete set of real or complex numbers. From this digital signal, we can generate discrete Fourier data (via the discrete Fourier transform). These data are related to s via the transform

$$S_n \equiv S(\omega_n) = \int_{-T}^{T} s(t) \exp(-i\omega_n t) dt, \quad \mid \omega_n \mid \leq \Omega.$$

They are a set of N numbers and define the bandlimited signal

$$s_{BL}(t) = \sum_{n=-N/2}^{N/2-1} S_n \exp(i\omega_n t).$$

This signal may be complex, real and of alternating or fixed polarity depending on the type of experiment that is conducted. In each case, the problem is to reconstruct the object f from N spectral samples in the presence of additive noise n.

Ill-posed Problems

There is no exact, unique or even correct solution to the type of problem that has been presented here. In other words, it is simply not possible to derive a solution as such for f from S_n, only an estimate for it. There are two reasons for this: (i) the exact value of the noise n at t is not known, only (at best) the probability of n having a particular value at t; (ii) even in the absence of noise, this type of problem is 'ill-posed'. A problem is well posed, if the solution:

(i) exists;

(ii) is unique;

(iii) depends continuously on the data.

If a problem violates any of these conditions, then it is ill-posed. It is condition (iii) that causes the main problem with digital signals. The finite nature of the data

means that there are many permissible solutions to the problem. As a consequence, we are faced with the problem of having to select one particular reconstruction. To overcome this inherent ambiguity, prior knowledge must be used to reduce the class of allowed solutions. For this reason, the use of prior information in the treatment of ill-posed problems of this nature is essential. In addition to prior information, the discrete nature of the data forces one to employ mathematical models for f. In principle, a wide variety of different models can be used which accounts for the range and diversity of algorithms that have been designed to cope with problems of this kind. Since all such algorithms attempt to solve the same basic problem, attention should focus on designs which are simple to implement and compute, data adaptive and reliable in the presence of noise of a varying dynamic range. Models for the object and conditions for the reconstruction should be utilized which are amenable to modification as knowledge about the object improves. This provides the opportunity to design an algorithm that is characterized by criteria and/or conditions for the reconstruction which is best suited to a particular application.

15.6 The Gerchberg-Papoulis Method

Given the equation

$$S_n \equiv S(\omega_n) = \int_{-T}^{T} s(t) \exp(-i\omega_n t) dt, \quad | \omega_n | \leq \Omega,$$

our problem is to solve for f given the data S_n. We start by considering a linear polynomial model for $f(t)$ of the following form,

$$f(t) = \sum_n A_n \exp(ik_n t)$$

where

$$\sum_n \equiv \sum_{n=-N/2}^{N/2-1}.$$

This model for f is just a complex Fourier series (with $k_n = n$). In order to compute f, the coefficients A_n (which are complex numbers) must be known. Given the model above, our problem is reduced to finding a method of computing A_n. How this is done depends on the criterion for the reconstruction that is chosen. The choice depends on a number of factors, such as the nature of the data, the complexity of the resulting algorithm and its computational cost. Here, we consider a least squares approach. The application of this approach for spectral extrapolation is known as the Gerchberg-Papoulis method. In this case, A_n are chosen in such a way that the mean square error

$$e = \int_{-T}^{T} | s(t) - f(t) |^2 \, dt$$

is a minimum. Substituting the equation

$$f(t) = \sum_n A_n \exp(i\omega_n t)$$

into the above equation and differentiating with respect to A_m we get (using the orthogonality principle)

$$\frac{\partial e}{\partial A_m} = -\int_{-T}^{T} \left(s - \sum_n A_n \exp(i\omega_n t) \right) \exp(-i\omega_m t) dt = 0.$$

Note that the above result is obtained using (see Chapter 8)

$$\frac{\partial e}{\partial \mathrm{Re}[A_m]} = 0; \quad \frac{\partial e}{\partial \mathrm{Im}[A_m]} = 0.$$

Interchanging the order of integration and summation, we obtain the following equation

$$\int_{-T}^{T} s(t) \exp(-i\omega_m t) dt = \sum_n A_n \int_{-T}^{T} \exp[-i(\omega_m - \omega_n)t] dt.$$

The left hand side of this equation is just the discrete Fourier data that is provided $S(\omega_n)$ and the integral on the right hand side of this equation gives a sinc function,

$$\int_{-T}^{T} \exp[-i(\omega_m - \omega_n)t] dt = 2T \,\mathrm{sinc}[(\omega_m - \omega_n)T]$$

By solving the equation

$$S(\omega_n) = 2T \sum_n A_n \mathrm{sinc}[T(\omega_m - \omega_n)]$$

for A_n, the object function $f(t)$ can be obtained.

15.7 Application of Weighting Functions

The solution above is a least squares approximation for $f(t)$. To compute A_n from $S(\omega_n)$, the value of T (the support of the object) needs to be known. We can therefore write $f(t)$ in the closed form,

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

where we have introduced the weighting function

$$w(t) = \begin{cases} 1, & \mid t \mid \leq T; \\ 0, & \mid t \mid > T. \end{cases}$$

This function is a simple form of *a priori* information. In this case, it is information about the finite extent of the object. The algebraic form of equation

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

suggests that the function $w(t)$ is used to incorporate more general prior knowledge about the object. Consider the case where we are given the data $S(\omega_n)$ defined by the equation

$$S_n \equiv S(\omega_n) = \int_{-T}^{T} s(t) \exp(-i\omega_n t) dt, \quad | \omega_n | \leq T$$

together with some form of prior knowledge on the structure of $f(t)$ that can be used to construct a suitable weighting function $w(t)$. The weighting function can be used to compute $f(t)$ as follows:

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t).$$

Substituting this equation into

$$e = \int_{-T}^{T} | s(t) - f(t) |^2 \, dt$$

we find that the error is a minimum when

$$\int_{-T}^{T} s(t) w(t) \exp(-i\omega_m t) dt = \sum_n A_n \int_{-T}^{T} [w(t)]^2 \exp[-i(\omega_m - \omega_n)t] dt.$$

Here, a problem occurs which is that for arbitrary functions w (which is what we must assume if different types of *a priori* information are to be incorporated), the integral on the left hand side of the above equation is not the same as the data provided $S(\omega_n)$. In other words, the equation above cannot be solved from the available data; it is not 'data consistent'. A way of overcoming this difficulty is to modify the expression for the mean square error function and introduce the following 'inverse weighted' form,

$$e = \int_{-T}^{T} | s(t) - f(t) |^2 \, \frac{1}{w(t)} dt.$$

This is a weighted Hilbert space, designed to provide data consistency. It is a minimum when

$$\int_{-T}^{T} s(t) \exp(-i\omega_m t) dt = \sum_n A_n \int_{-T}^{T} w(t) \exp[-i(\omega_m - \omega_n)t] dt.$$

Here, the data on the left hand side of the above equation is equal to $S(\omega_n)$. We therefore have a data consistent equation of the form

$$S(\omega_m) = \sum_n A_n W(\omega_m - \omega_n)$$

where

$$W(\omega_m) = \int_{-T}^{T} w(t) \exp(-i\omega_m t) dt, \quad |\, \omega_m \,| \leq \Omega.$$

This method provides a solution which allows arbitrary weighting functions $w(t)$ containing additional prior information on the structure of f to be introduced. The method can be summarized as follows:

1. Given the data $S(\omega_n)$, construct a weighting function $w(t)$ that is obtained from *a priori* knowledge on the structure of $f(t)$.

2. Compute $W(\omega_n)$ from $w(t)$.

3. Solve the equation

$$\sum_n A_n W(\omega_m - \omega_n) = S(\omega_m)$$

to obtain the coefficients A_n.

4. Compute the estimate $w(t) \sum_n A_n \exp(i\omega_n t)$.

This algorithm is based on minimizing the inverse weighted mean square error function given by

$$e = \int_{-T}^{T} |\, s(t) - f(t) \,|^2 \, \frac{1}{w(t)} dt.$$

Note that the algebraic form of this error indicates that w should be greater than zero to avoid singularities occurring in $1/w$. Also note that when

$$w(t) = 1, \quad |\, t \,| \leq T$$

the former least squares (Gerchberg-Papoulis method) estimate is obtained.

Practical Considerations

In practice, the data $S(\omega_n)$ is obtained by taking the discrete Fourier transform of some band limited signal $s_{BL}(t)$. The data $W(\omega_n)$ is obtained by computing the discrete Fourier transform of $w(t)$ and reducing the bandwidth of the spectrum so that it is the same as $S(\omega_n)$. We then solve

$$\sum_n A_n W(\omega_m - \omega_n) = S(\omega_m)$$

for A_n. This equation is just a discrete convolution in Fourier space, i.e.

$$S(\omega_n) = A(\omega_n) \otimes W(\omega_n)$$

where $\otimes$ denotes the convolution sum. Hence, using the product theorem, we can write (ignoring scaling)

$$s_{BL}(t) = a(t) w_{BL}(t)$$

where s_{BL} and w_{BL} are bandlimited estimates of $s(t)$ and $w(t)$ respectively given by

$$s_{BL}(t) = \sum_n S(\omega_n) \exp(i\omega_n t),$$

$$w_{BL}(t) = \sum_n W(\omega_n) \exp(i\omega_n t)$$

and

$$a(t) = \sum_n A(\omega_n) \exp(i\omega_n t).$$

Using the (weighted) model for $f(t)$ we have

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t) \equiv w(t) \sum_n A(\omega_n) \exp(i\omega_n t) = w(t)a(t)$$

and hence, the minimum mean square estimate of f can be written as

$$f(t) = \frac{w(t)}{w_{BL}(t)} s_{BL}(t).$$

From this equation, it is easy to compute f given s_{BL} and w. All that is required is a DFT to obtain w_{BL} from w which can be computed using an FFT.

15.8 Burg's Maximum Entropy Method

The application of a maximum entropy criterion for solving the spectral extrapolation problem is usually attributed to a research thesis by J P Burg published in 1975 at Stanford University, USA. This technique is sometimes called the 'all poles' method because of the nature of the estimation model used. It is a method which is often associated with the reconstruction of a power spectrum but can in fact be applied to any problem involving the reconstruction of signals from band limited data. The problem is, given

$$F_n \equiv F(k_n) = \int_{-T}^{T} f(t) \exp(-i\omega_n t) dt$$

where $\mid \omega_n \mid \leq \Omega$ (the bandwidth), recover $f(t)$. This problem is equivalent to extrapolating the data F_n beyond the bandwidth Ω. Suppose we consider a model for $f(t)$ given by

$$f(t) = \frac{1}{\mid \sum_n A_n \phi_n(t) \mid^2}$$

where

$$\phi_n(t) = \exp(-i\omega_n t) \quad \text{and} \quad \sum_n \equiv \sum_{n=0}^{N-1}.$$

Further, let us consider a criterion for computing A_n that is based on maximising the Entropy of the signal E defined by

$$E = \int_{-T}^{T} \ln f(t) dt$$

which requires us to impose the condition $f(t) > 0 \ \ \forall \ \ t$. Now, the Entropy measure $E(A_m)$ is a maximum when

$$\frac{\partial E}{\partial A_m} = 0 \quad \text{for} \quad m > 0.$$

Using the model for $f(t)$, the entropy can be written in the form

$$E = -\int_{-T}^{T} \ln \mid \sum_n A_n \phi_n(t) \mid^2 dt.$$

Differentiating, we get

$$\frac{\partial E}{\partial A_m} = -\frac{\partial}{\partial A_m} \int_{-T}^{T} \ln \mid \sum_n A_n \phi_n(t) \mid^2 dt$$

$$= -\int_{-T}^{T} \frac{1}{\mid \sum_n A_n \phi_n(t) \mid^2} \frac{\partial}{\partial A_m} \mid \sum_n A_n \phi_n(t) \mid^2 dt$$

$$= \int_{-T}^{T} f(t) \sum_n A_n \phi_n(t) \phi_n^*(t) dt = 0$$

or with $\phi_n(t) = \exp(-i\omega_n t)$,

$$\sum_n A_n \int_{-T}^{T} f(t) \exp[-i(\omega_n - \omega_m)t] dt = 0$$

or

$$\sum_n A_n F(\omega_n - \omega_m) = 0, \quad m > 0.$$

where $F(\omega_n)$ is the data given. This is a data consistent result if in the case when $m = 0$, we use the normalization condition:

$$\sum_n A_n F(\omega_n - \omega_m) = 1, \quad m = 0.$$

We are then required to solve the following system of equations:

$$\sum_n A_n F_{n-m} = \begin{cases} 1, & m = 0; \\ 0, & m > 0. \end{cases}$$

In matrix form, this system can be written as:

$$\begin{pmatrix} F_0 & F_1 & F_2 & \dots & F_{N-1} \\ F_{-1} & F_0 & F_1 & \dots & F_{N-2} \\ F_{-2} & F_{-1} & F_0 & \dots & F_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F_{1-N} & F_{2-N} & F_{3-N} & \dots & F_0 \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{N-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The characteristic matrix is a Toeplitz matrix and by solving this system of equations for A_n we can compute $f(t)$ using

$$f(t) = \frac{1}{\mid \sum_n A_n \exp(-i\omega_n t) \mid^2}.$$

15.9 Summary of Important Results

Bayes Rule

$$P(A)P(B \mid A) = P(B)P(A \mid B)$$

where $P(A)$ and $P(B)$ are the probabilities that events A and B will occur respectively and $P(B \mid A)$ and $P(A \mid B)$ are the conditional probability of B given A and the conditional probability of A given B respectively.

Bayesian Estimation (for a linear time invariant system with additive noise
Given that

$$\mathbf{s} = \mathbf{p} \otimes \mathbf{f} + \mathbf{n},$$

find $\mathbf{f}$ such that

$$P(\mathbf{f} \mid \mathbf{s}) = \frac{P(\mathbf{s} \mid \mathbf{f})P(\mathbf{f})}{P(\mathbf{s})}.$$

Maximum a Posteriori Estimation
Given that

$$\mathbf{s} = \mathbf{p} \otimes \mathbf{f} + \mathbf{n},$$

find $\mathbf{f}$ such that

$$\frac{\partial}{\partial \mathbf{f}} \ln P(\mathbf{s} \mid \mathbf{f}) + \frac{\partial}{\partial \mathbf{f}} \ln P(\mathbf{f}) = 0.$$

Maximum Likelihood Estimation
Given that

$$\mathbf{s} = \mathbf{p} \otimes \mathbf{f} + \mathbf{n},$$

find $\mathbf{f}$ such that

$$\frac{\partial}{\partial \mathbf{f}} \ln P(\mathbf{s} \mid \mathbf{f}) = 0.$$

Maximum Entropy Estimation
Given that

$$s_i = p_i \otimes f_i + n_i,$$

find f_i such that the entropy E, defined by

$$E = -\sum_i f_i \ln f_i$$

is a maximum where $f_i > 0, \quad \forall i$.

Spectral Extrapolation of Bandlimited Signals
Given the bandlimited signal (with a bandwidth of 2Ω)

$$s_{\mathrm{BL}}(t) = \mathrm{sinc}(\Omega t) \otimes f(t) + n(t), \quad \mid t \mid \leq T$$

where $n(t)$ is bandlimited noise (with the same bandwidth), find an estimate for $f(t)$. The problem is equivalent to, given

$$S_n \equiv S(\omega_n) = \int_{-T}^{T} s(t) \exp(-i\omega_n t) dt$$

where

$$s(t) = f(t) + n(t)$$

and

$$\mid \omega_n \mid \leq \Omega,$$

estimate $f(t)$.

Gerchberg-Papoulis Method

$$f(t) = \sum_n A_n \exp(ik_n t)$$

where A_n is given by solving the equation

$$S(\omega_n) = 2TA_n \otimes \mathrm{sinc}[T(\omega_m - \omega_n)]$$

for A_n and is based on minimizing

$$e(A_n) = \|s(t) - f(t)\|_2^2.$$

Weighted Gerchberg-Papoulis Method

$$f(t) = \frac{w(t)}{w_{\mathrm{BL}}} s_{\mathrm{BL}}(t)$$

where w (weighting function) is *a priori* information on the functional characteristics of f and w_{BL} is the bandlimited *a priori* information (bandlimited weighting function). The solution is based on a linear polynomial model of the form

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

and computing A_n by minimizing the inverse weighted mean square error

$$e(A_n) = \| \mid s(t) - f(t) \mid / \sqrt{w(t)} \|_2^2$$

where $\mathrm{Re}[w(t)] > 0, \quad \forall t.$

(Burg's) Maximum Entropy Method

$$f(t) = \frac{1}{\mid \sum_{n=0}^{N-1} A_n \exp(-i\omega_n t) \mid^2}$$

where A_n are computed subject to the maximum entropy criterion in which the Entropy is defined by

$$E = \int_{-T}^{T} \ln f(t) dt, \quad f(t) > 0 \ \ \forall \ \ t,$$

and given by solving

$$\sum_{n=0}^{N-1} A_n F_{n-m} = \begin{cases} 1, & m = 0; \\ 0, & m > 0; \end{cases}$$

where

$$F_n \equiv F(\omega_n) = \int_{-T}^{T} f(t) \exp(-i\omega_n t) dt.$$

15.10 Further Reading

- Blackman R B and Yukey J W, *The Measurement of Power Spectra*, Dover, 1958.
- Papoulis A, *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, 1965.
- Van Trees H L, P*Detection, Estimation and Modulation Theory*, Wiley, 1968.
- Burg J P, *Maximum Entropy Spectral Analysis*, PhD Thesis, Stanford CA, Stanford University, 1975.
- Papoulis A, *Signal Analysis*, McGraw-Hill, 1977.
- Erickson G J and Smith C R (Eds.), *Maximum Entropy and Bayesian Methods in Science and Engineering*, Kluwer Academic Publishers, 1988.

- Oppenheim A V (Ed.), Applications of Digital Signal Processing, Prentice-Hall, 1978.

- Buck B B and Macaulay V A (Eds.), *Maximum Entropy in Action*, Clarendon Press, 1992.

- Lee P M, *Bayesian Statistics*, Arnold, 1997.

15.11 Problems

15.1 Given the data S_n which is taken to be a discrete bandlimited spectrum of the signal

$$s(t) = f(t) + n(t)$$

where n is the noise, obtain an ML estimate for the object function f of the form

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

based on the finding the coefficients A_n subject to the condition that

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{s} \mid \mathbf{f}) = 0$$

where

$$P(\mathbf{n}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T}^{T} \frac{\mid n(t) \mid^2}{w(t)} dt \right).$$

15.2 Obtain an ML estimate based on question 15.1 above in which the noise is taken to obey a Rayleigh distribution given by

$$P(\mathbf{n}) = \frac{1}{\sigma_n^2} \int_{-T/2}^{T/2} \mid n(t) \mid dt \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T/2}^{T/2} \frac{\mid n(t) \mid^2}{w(t)} dt \right)$$

where $s(t) \geq 0$ and real.

15.3 Consider the case in which

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T}^{T} \mid n(t) \mid^2 \frac{1}{w(t)} dt \right)$$

and

$$P(\mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_f^2}} \exp\left(-\frac{1}{2\sigma_f^2} \int_{-T}^{T} \mid f(t) - \bar{f}(t) \mid^2 \frac{1}{w(t)} dt \right)$$

where $\bar{f}$ is the average value of f at a point t. Find a MAP estimate for the object function $f(t)$ given by

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

where

$$s(t) = f(t) + n(t)$$

and when the data S_n is bandlimited.

15.4 Given that $s(t) = f(t) + n(t) \geq 0 \forall t$ and is a real valued function, find a MAP estimate for $f(t)$ when

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T}^{T} \mid n(t) \mid^2 \frac{1}{w(t)} dt \right)$$

and

$$P(\mathbf{f}) = \frac{1}{\sigma_f^2} \int_{-T}^{T} f(t) dt \exp\left(-\frac{1}{2\sigma_f^2} \int_{-T}^{T} [f(t)]^2 \frac{1}{w(t)} dt \right)$$

where

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

and the data provided is a bandlimited discrete spectrum of $s(t)$ given by S_n.

15.5 Given the data S_n which is assumed to be a bandlimited spectrum obtained from the signal

$$s(t) = f(t) + n(t),$$

derive a solution for $f(t)$ where

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t)$$

by finding the coefficients A_n that maximize the entropy of the amplitude spectrum given by

$$E = \sum_n \mid A_n \mid \ln \mid A_n \mid .$$

By linearising your solution, obtain an approximate form for $f(t)$.

15.6 Suppose we consider a model for $f(t)$ of the form

$$f(t) = w(t) \exp\left[\sum_n A_n \exp(i\omega_n t) \right]$$

which is to be used to reconstruct f from the bandlimited data S_n obtained from a signal $s(t) = f(t) + n(t)$. Design a suitable Hilbert space (i.e. an expression for the error function) which yields a data consistent result in order to evaluate the coefficients A_n and hence obtain a closed form solution for $f(t)$.

Chapter 16

Digital Filtering in the Time Domain

Time domain filtering is based on processing the 'real space' data of a signal rather than its associated 'Fourier space' data. There are a wide range of filters of this type but in general they nearly all fall in to one of two classes:

(i) non-recursive filters;

(ii) recursive filters.

16.1 The FIR Filter

The finite impulse response or FIR filter is one of the most elementary but widely used filters. An impulse response function is simply the output of the filter when an impulse is applied as input as illustrated below.

$$\begin{array}{ccccc} \textbf{Input} & \longrightarrow & \textbf{System} & \longrightarrow & \textbf{Output} \\ \delta(t) & \longrightarrow & p(t) & \longrightarrow & s(t) \end{array}$$

If the system is a linear time invariant system then we have,

$$s(t) = \int p(\tau - t)\delta(\tau)d\tau = p(t)$$

and p is referred to the impulse response function. In digital form, the impulse response function is finite and given by

$$s_j = \sum_i p_{j-i}\delta_i = p_j$$

where we consider the case when

$$\sum_i \equiv \sum_{i=-N}^{N}$$

and δ_i is the Kronecker delta function. For an arbitrary input f_i, the filtering operation is

$$s_j = \sum_i p_{j-i} f_i$$

which models the response of an input to the finite impulse response function and hence the name, FIR filter. Filters of this type have at most $2N + 1$ non-zero coefficients.

The FIR Filter for Discrete Convolution

The discrete convolution operation (the convolution sum) can be written in the form (since the convolution process is commutative)

$$s_j = \sum_{i=-N}^{N} p_i f_{j-i}.$$

To illustration the nature of this process, consider the case when p_i and f_i are vectors with just 3 elements, i.e.

$$\mathbf{p} = (p_{-1}, p_0, p_1)^T,$$

$$\mathbf{f} = (f_{-1}, f_0, f_1)^T,$$

and where

$$f_{-2} = 0, \quad \text{and} \quad f_2 = 0.$$

Then,

for $j = -1$:

$$s_{-1} = \sum_{i=-1}^{1} p_i f_{-1-i} = p_{-1} f_0 + p_0 f_{-1} + p_1 f_{-2} = p_{-1} f_0 + p_0 f_{-1},$$

for $j = 0$:

$$s_0 = \sum_{i=-1}^{1} p_i f_{-i} = p_{-1} f_1 + p_0 f_0 + p_1 f_{-1},$$

for $j = 1$:

$$s_1 = \sum_{i=-1}^{1} p_i f_{1-i} = p_{-1} f_2 + p_0 f_1 + p_1 f_0 = p_0 f_1 + p_1 f_0.$$

Clearly, this result can be written in matrix form as

$$\begin{pmatrix} s_{-1} \\ s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} f_0 & f_{-1} & 0 \\ f_1 & f_0 & f_{-1} \\ 0 & f_1 & f_0 \end{pmatrix} \begin{pmatrix} p_{-1} \\ p_0 \\ p_1 \end{pmatrix}.$$

Now consider the convolution sum defined as

$$s_j = \sum_{i=-N}^{N} p_{j-i} f_i.$$

With

$$\mathbf{p} = (p_{-1}, p_0, p_1)^T, \quad p_{-2} = p_2 = 0$$

and

$$\mathbf{f} = (f_{-1}, f_0, f_1)^T$$

we have

for $j = -1$:

$$s_{-1} = \sum_{i=-1}^{1} p_{-1-i} f_i = p_0 f_{-1} + p_{-1} f_0 + p_{-2} f_1 = p_0 f_{-1} + p_{-1} f_0,$$

for $j = 0$:

$$s_0 = \sum_{i=-1}^{1} p_{-i} f_i = p_1 f_{-1} + p_0 f_0 + p_{-1} f_1,$$

for $j = 1$:

$$s_1 = \sum_{i=-1}^{1} p_{1-i} f_i = p_2 f_{-1} + p_1 f_0 + p_0 f_1 = p_1 f_0 + p_0 f_1.$$

In matrix form, this result becomes

$$\begin{pmatrix} s_{-1} \\ s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} p_0 & p_{-1} & 0 \\ p_1 & p_0 & p_{-1} \\ 0 & p_1 & p_0 \end{pmatrix} \begin{pmatrix} f_{-1} \\ f_0 \\ f_1 \end{pmatrix}$$

Note that

$$\begin{pmatrix} p_0 & p_{-1} & 0 \\ p_1 & p_0 & p_{-1} \\ 0 & p_1 & p_0 \end{pmatrix} \begin{pmatrix} f_{-1} \\ f_0 \\ f_1 \end{pmatrix} = \begin{pmatrix} f_0 & f_{-1} & 0 \\ f_1 & f_0 & f_{-1} \\ 0 & f_1 & f_0 \end{pmatrix} \begin{pmatrix} p_{-1} \\ p_0 \\ p_1 \end{pmatrix}$$

and that in general

$$\sum_{i=-N}^{N} p_i f_{j-i} = \sum_{i=-N}^{N} p_{j-i} f_i$$

which confirms that the discrete convolution sum is commutative. However, the latter definition of a convolution sum is better to work with because it ensures that the matrix is filled with elements relating to the impulse response function p_i, i.e.

$$\mathbf{s} = P\mathbf{f}.$$

Clearly, if $\mathbf{f}$ is an $(2N+1)^{\text{th}}$ order vector and $\mathbf{p}$ contains just three elements say, then the convolution sum can be written in the form

$$\begin{pmatrix} s_{-N} \\ \vdots \\ s_{-1} \\ s_0 \\ s_1 \\ \vdots \\ s_N \end{pmatrix} = \begin{pmatrix} \ddots & & & & & & \\ & \ddots & & & & & \\ & p_1 & p_0 & p_{-1} & & & \\ & & p_1 & p_0 & p_{-1} & & \\ & & & p_1 & p_0 & p_{-1} & \\ & & & & & \ddots & \\ & & & & & & \ddots \end{pmatrix} \begin{pmatrix} f_{-N} \\ \vdots \\ f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix}.$$

Here, P is a tridiagonal matrix. In general, the bandwidth of the matrix is determined by the number of elements of the impulse response function. Note that the inverse process (i.e. deconvolving $\mathbf{s}$ given P to compute $\mathbf{f}$) can be solved in this case by using an algorithm for solving tridiagonal systems of equations (see Chapter 7).

Useful Visualization of the Discrete Convolution Process

Another way of interpreting the discrete convolution process which is useful visually is in terms of the of two streams of numbers sliding along each other where at each location in the stream, the appropriate numbers are multiplied and the results added together. In terms of the matrix above we have:

$$\begin{array}{cc} \vdots & \\ f_{-4} & \\ f_{-3} & p_1 \\ f_{-2} & p_0 \quad (= s_{-2}) \\ f_{-1} & p_{-1} \\ f_0 & \\ f_1 & \\ f_2 & p_1 \\ f_3 & p_0 \quad (= s_3) \\ f_4 & p_{-1} \\ \vdots & \end{array}$$

In general, if

$$\mathbf{f} = \begin{pmatrix} f_{-N} \\ \vdots \\ f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix} \quad \text{and} \quad \mathbf{p} = \begin{pmatrix} p_{-N} \\ \vdots \\ p_{-1} \\ p_0 \\ p_1 \\ \vdots \\ p_N \end{pmatrix}$$

then

$$\begin{array}{cc} \vdots & \\ f_{-4} & \\ f_{-3} & \vdots \\ f_{-2} & p_1 \\ f_{-1} & p_0 \quad (= s_{-1}) \\ f_0 & p_{-1} \\ f_1 & \vdots \\ f_2 & \\ f_3 & \\ f_4 & \\ \vdots & \end{array}$$

Note that the order of the elements of $\mathbf{p}$ is reversed with respect to $\mathbf{f}$.

On Notation and Jargon

The vector $\mathbf{p}$ is sometimes called the Kernel, a term taken from the 'Kernel' of an integral equation of the type

$$s(t) = \int K(t,\tau) f(\tau) d\tau$$

where K is the Kernel. Visualising a discrete convolution in the form discussed above leads to $\mathbf{p}$ being referred to as a 'window' since we can think of this process in terms of looking at the data f_i through a window of coefficients p_i. As we slide the stream of coefficients p_i along the data f_i, we see the data in the form of the output s_i which is the running weighted average of the original data f_i. Because the window moves over the data it is often referred to as a 'moving window'.

16.2 The FIR Filter and Discrete Correlation

The discrete correlation operation (the correlation sum) can be written in the form

$$s_j = \sum_{i=-N}^{N} p_i f_{i-j}.$$

Compared with the convolution sum, the subscript on f is reversed (i.e. f_{j-i} becomes f_{i-j}). Consider the case, when p_i and f_i are vectors with just 3 elements:

$$\mathbf{p} = (p_{-1}, p_0, p_1)^T,$$

$$\mathbf{f} = (f_{-1}, f_0, f_1)^T, \quad f_{-2} = f_2 = 0.$$

For $j = -1$:

$$s_{-1} = \sum_{i=-1}^{1} p_i f_{i+1} = p_{-1} f_0 + p_0 f_1 + p_1 f_2 = p_{-1} f_0 + p_0 f_1,$$

for $j = 0$:

$$s_0 = \sum_{i=-1}^{1} p_i f_i = p_{-1}f_{-1} + p_0 f_0 + p_1 f_1,$$

for $j = 1$:

$$s_1 = \sum_{i=-1}^{1} p_i f_{i-1} = p_{-1}f_{-2} + p_0 f_{-1} + p_1 f_0 = p_0 f_{-1} + p_1 f_0.$$

This result can be written in matrix form as

$$\begin{pmatrix} s_1 \\ s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} f_0 & f_1 & 0 \\ f_{-1} & f_0 & f_1 \\ 0 & f_{-1} & f_0 \end{pmatrix} \begin{pmatrix} p_{-1} \\ p_0 \\ p_1 \end{pmatrix}.$$

Now consider the correlation sum defined as

$$s_j = \sum_{i=-N}^{N} p_{i-j} f_i.$$

With

$$\mathbf{p} = (p_{-1}, p_0, p_1)^T, \quad p_{-2} = p_2 = 0$$

and

$$\mathbf{f} = (f_{-1}, f_0, f_1)^T$$

we have

for $j = -1$:

$$s_{-1} = \sum_{i=-1}^{1} p_{i+1} f_i = p_0 f_{-1} + p_1 f_0 + p_2 f_1 = p_0 f_{-1} + p_1 f_0,$$

for $j = 0$:

$$s_0 = \sum_{i=-1}^{1} p_i f_i = p_{-1}f_{-1} + p_0 f_0 + p_1 f_1,$$

for $j = 1$:

$$s_1 = \sum_{i=-1}^{1} p_{i-1} f_i = p_{-2}f_{-1} + p_{-1}f_0 + p_0 f_1 = p_{-1}f_0 + p_0 f_1.$$

and in matrix form, the result becomes

$$\begin{pmatrix} s_{-1} \\ s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} p_0 & p_1 & 0 \\ p_{-1} & p_0 & p_1 \\ 0 & p_{-1} & p_0 \end{pmatrix} \begin{pmatrix} f_{-1} \\ f_0 \\ f_1 \end{pmatrix}.$$

Note that

$$\begin{pmatrix} p_0 & p_1 & 0 \\ p_{-1} & p_0 & p_1 \\ 0 & p_{-1} & p_0 \end{pmatrix} \begin{pmatrix} f_{-1} \\ f_0 \\ f_1 \end{pmatrix} \neq \begin{pmatrix} f_0 & f_1 & 0 \\ f_{-1} & f_0 & f_1 \\ 0 & f_{-1} & f_0 \end{pmatrix} \begin{pmatrix} p_{-1} \\ p_0 \\ p_1 \end{pmatrix}$$

and in general

$$\sum_{i=-N}^{N} p_i f_{i-j} \neq \sum_{i=-N}^{N} p_{i-j} f_i$$

illustrating that unlike the convolution sum, the correlation sum is not commutative. As with the discrete convolution sum, the latter definition of a correlation sum is better to work with because it ensures that the matrix is filled with elements relating to the impulse response function p_i so that we can write

$$\mathbf{s} = P\mathbf{f}.$$

If $\mathbf{f}$ is an $(2N+1)^{\text{th}}$ order vector and $\mathbf{p}$ contains just three elements say, then the correlation sum can be written in the form

$$\begin{pmatrix} s_{-N} \\ \vdots \\ s_{-1} \\ s_0 \\ s_1 \\ \vdots \\ s_N \end{pmatrix} = \begin{pmatrix} \ddots & & & & & & \\ & \ddots & & & & & \\ & p_{-1} & p_0 & p_1 & & & \\ & & p_{-1} & p_0 & p_1 & & \\ & & & p_{-1} & p_0 & p_1 & \\ & & & & & \ddots & \\ & & & & & & \ddots \end{pmatrix} \begin{pmatrix} f_{-N} \\ \vdots \\ f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix}.$$

Useful Visualization of the Discrete Correlation Process

A useful way of visualising the discrete correlation process is in terms of the of two streams of numbers sliding along each other where at each location in the stream, the appropriate numbers are multiplied and the results added together. In terms of the matrix equation above we have:

$$\begin{array}{cc} \vdots & \\ f_{-4} & \\ f_{-3} & p_{-1} \\ f_{-2} & p_0 \quad (= s_{-2}) \\ f_{-1} & p_1 \\ f_0 & \\ f_1 & \\ f_2 & p_{-1} \\ f_3 & p_0 \quad (= s_3) \\ f_4 & p_1 \\ \vdots & \end{array}$$

In general, if

$$\mathbf{f} = \begin{pmatrix} f_{-N} \\ \vdots \\ f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix} \quad \text{and} \quad \mathbf{p} = \begin{pmatrix} p_{-N} \\ \vdots \\ p_{-1} \\ p_0 \\ p_1 \\ \vdots \\ p_N \end{pmatrix}$$

then

$$\begin{array}{cc} \vdots & \\ f_{-4} & \\ f_{-3} & \\ f_{-2} & \vdots \\ f_{-1} & p_{-1} \\ f_0 & p_0 \quad (= s_0) \\ f_1 & p_1 \\ f_2 & \vdots \\ f_3 & \\ f_4 & \\ \vdots & \end{array}$$

Note that unlike convolution, the order of the elements of $\mathbf{p}$ is preserved with respect to $\mathbf{f}$. If the impulse response function is symmetric, then the convolution and correlation sums are identical (for real data). The jargon associated with the discrete convolution process is also used in the case of discrete correlation. Correlation is also sometimes used in the context of matched filtering (see Chapter 14 - the 'Matched Filter').

16.3 Computing the FIR filter

A problem arises in computing the FIR filter (convolution or correlation) at the ends of the array f_i. For example, if $\mathbf{p}$ is a 5×1 kernel, then at the end of the data stream we have

$$\begin{array}{cc} \vdots & \\ f_{N-3} & p_{-2} \\ f_{N-2} & p_{-1} \\ f_{N-1} & p_0 \\ f_N & p_1 \\ & p_2 \end{array}$$

In the computation of s_{N-1} there is no number associated with the data f_i with which to multiply p_2. Similarly, in the computation of s_N we have

$$\begin{array}{cc} \vdots & \\ f_{N-3} & \\ f_{N-2} & p_{-2} \\ f_{N-1} & p_{-1} \\ f_N & p_0 \\ & p_1 \\ & p_2 \end{array}$$

Here, there are no numbers associated with the array f_i with which to multiply p_1 and p_2. The same situation occurs at the other end of the array f_i. Hence, at both ends of the data, the moving window 'runs out' of data for computing the convolution sum. There are a number of ways of solving this problem including zero padding, endpoint extension and rapping.

Zero Padding

Zero padding assumes that the data is zero beyond the ends of the array, i.e.

$$f_{\pm N\pm 1} = f_{\pm N\pm 2} = f_{\pm N\pm 2} = ... = 0.$$

This method was applied in the previous sections to introduce the FIR filter.

Endpoint Extension

Endpoint extension assumes that the data beyond the ends of the array takes on the value of the end points of the array, i.e. the extrapolated data is equal in value to end points:

$$f_{N+1} = f_{N+2} = f_{N+3} = ... = f_N$$

and

$$f_{-N-1} = f_{-N-2} = f_{-N-3} = ... = f_{-N}.$$

This method is sometimes known as the 'constant continuation method'.

Rapping

The rapping technique assumes that the array is rapped back on itself so that

$$f_{N+1} = f_{-N}; \quad f_{N+2} = f_{-N+1}; \quad f_{N+3} = f_{-N+2}; \quad \text{etc.}$$

and

$$f_{-N-1} = f_N; \quad f_{-N-2} = f_{N-1}; \quad f_{-N-3} = f_{N-2}; \quad \text{etc.}$$

These methods are used in different circumstances but the endpoint extension technique is probably one of the most widely used.

16.3.1 Moving Window Filters

The FIR filter is just one example of a moving window filter in which the computational process is a convolution. There are a range of filters that can be designed in which various processes are repeatedly applied to the windowed data.

The Moving Average Filter

The moving average filter computes the average value of a set of samples within a predetermined window.

Example For a 3×1 window:

$$\begin{array}{ll} \vdots & \\ f_i & \\ f_{i+1} & s_{i+1} = (f_i + f_{i+1} + f_{i+2})/3 \\ f_{i+2} & s_{i+2} = (f_{i+1} + f_{i+2} + f_{i+3})/3 \\ f_{i+3} & s_{i+3} = (f_{i+2} + f_{i+3} + f_{i+4})/3 \\ f_{i+4} & \\ \vdots & \end{array}$$

As the window moves over the data, the average of the samples 'seen' within the window is computed, hence the term 'moving average filter'. In mathematical terms, we can express this type of processing in the form

$$s_i = \frac{1}{M} \sum_{j \in w(i)} f_j$$

where $w(i)$ is the window located at i over which the average of the data samples is computed and M is the total number of samples in w. Note that the moving average filter is just an FIR of the form

$$s_i = \sum_{i=-N}^{N} p_{j-i} f_i$$

with

$$\mathbf{p} = \frac{1}{M}(1, 1, 1, ..., 1)$$

so for a 3×1 kernel

$$\mathbf{p} = \frac{1}{3}(1, 1, 1)$$

and for a 5×1 kernel

$$\mathbf{p} = \frac{1}{5}(1, 1, 1, 1, 1).$$

This filter can be used to smooth a signal, a feature which can be taken to include the reduction of noise. Note that this filter is in effect, the convolution of an input with a tophat function; the spectral response is therefore a sinc function.

The Median Filter

The median filter moves a window (of arbitrary but usually odd size) over the data computing the median of the samples defined within the window at each stage. The median m of a set of numbers is such that half the numbers in the set are less than m and half are greater than m. For example, if we consider the set $(3, 4, 10, 21, 22, 48, 57)$, then $m = 21$. There are a number of ways to compute the median of an arbitrary set of numbers. One way is to reorder the numbers in ascending values.

Example (1,6,2,4,7,3,9)$\longrightarrow$(1,2,3,4,6,7,9) giving $m = 4$. The reordering of the numbers in this way can be accomplished using a 'bubble sort' where the maximum values of the array (in decreasing order) are computed and relocated in a number of successive passes.

The Moving Average .v. the Median Filter

Both filters can be used to reduce noise in a signal. Noise reduction algorithms aim to reduce noise while attempting to preserve the information content of a signal. In this sense, because the moving average filter 'smooths' the data, the median filter is a superior noise reducing filter for the removal of isolated noise spikes. Note that unlike the moving average filter, the median filter is not a convolution process and the spectral response can not be computed via the convolution theorem.

Other Statistical Filters

Having introduced the process above, it should be clear to the reader that a range of filters can be introduced using the moving window principle. The type of filter reflects the process that is being undertaken. Thus, the mean, variance and other moments can by computed where the r^{th} moment of the signal s_i whose histogram is $P_i \equiv P(x_j), \quad j = 1, 2, 3, ..., N$ formed from N bins is given by[1]

$$M_r = \sum_{j=1}^{N} (x_j - M)^r P(x_j)$$

where M is the mean. In addition, the median and mode filters can be computed using the same approach, the mode being defined as that value which occurs most often (i.e. has the greatest probability of occurring) or

$$\text{mode} = \|P_i\|_\infty.$$

Other statistical parameters include the skewness, one such measure being defined by

$$\text{Skewness} = \frac{M_3}{M^{\frac{3}{2}}}$$

and the Kurtosis based on the common measure

$$\text{Kurtosis} = \frac{M_4}{M_2^2}.$$

[1]Note the the second moment M_2 is the variance.

Moreover, if the signal has statistical and/or spectral characteristics that change, it is often informative to investigate such variations especially when the signal is stochastically non-stationary. For example, suppose that the input is a discrete stochastic signal s_i that is Gamma-distributed, i.e. ignoring scaling, its histogram is given by

$$P(x_i) = x_i^{\alpha} \exp(-\beta x_i)$$

where α and β are time variant. Then, by computing α and β on a moving window basis, the signals α_i and β_i can be used to analyse the non-stationary behaviour of the data. The computation undertaken at each position of the window along the data stream in this example can be based on a least squares fit, where α and β are computed such that

$$e(\alpha, \beta) = \| \ln D_i - \ln P_i \|_2^2$$

is a minimum where $D_i \equiv D(x_i), \quad i = 1, 2, 3, ..., N$ is the histogram of the input data formed from N bins. Clearly, the size of the window has to provide data that produces a statistically significant result. As a final example, consider a random fractal signal with variations in the Fourier dimension q and whose power spectrum is modelled by[2]

$$\hat{P}_i = \frac{c}{\mid x_i \mid^{2q}}$$

where c is a constant. By utilising the error function

$$e(q, c) = \| \ln P_i - \ln \hat{P}_i \|_2^2$$

and minimizing it with respect to q and c, an expression for q (and c) can be obtained that is then used to compute q on a moving window basis to yield the signal q_i where i is the position of the window.

16.3.2 Interpolation using the FIR Filter

Discrete convolution can be used effectively to interpolate a function. For example, suppose we want to linearly interpolate a function f_i from N data points to 2N where the computation of a point between f_{i+1} and f_i is given by

$$f_i + \frac{f_{i+1} - f_i}{2} = \frac{f_{i+1} - f_i}{2}.$$

This process is equivalent to implementing the following:

- Given the initial array $(f_1, f_2, f_3, ..., f_N)$ of size N, construct the array $g_i = (0, f_1, 0, f_2, 0, f_3, 0, ..., f_N, 0)$ which is of size $2N + 1$ and is zero padded.

- Convolve g_i with the kernel $\frac{1}{2}(1, 0, 1)$.

[2] As discussed further in Chapter 17.

16.3.3 The FIR Filter and the FFT

The implementation of the FIR filter directly is computationally advantageous when the size of the kernel is small compared to the data stream. For a kernel of size M say and a data stream of size N, the number of multiplications are $N \times M$. When M approaches N and the number of multiplication approach N^2, it is computationally more efficient to implement the FFT algorithm to perform the convolution operation (where the number of multiplications is approximately equal to $N \log_2 N$). To perform this operation, the kernel is required to be zero padded. In other words, the signal is padded out with zeros (on both sides) until the size of the array is equal to the size of the data stream to which the convolution operation is to be applied.

16.4 The IIR Filter

The FIR filter is based on the model

$$s_i = p_i \otimes f_i.$$

It represents a system in which an input f_i is modified (via the a convolution process) by some system characterized by p_i to produce an output s_i. In such a process, there is no feedback of the output to the input. Suppose that we want to model a feedback system where the output is fed back into the input. Now, let the original input f_i give an output $s_i^{(1)} = p_i \otimes f_i$ in the usual way. Feeding this output back into the input, the next input becomes $f_i + s_i^{(1)}$ giving an output

$$s_i^{(2)} = p_i \otimes (f_i + s_i^{(1)}) = p_i \otimes f_i + p_i \otimes s_i^{(1)} = p_i \otimes f_i + p_i \otimes p_i \otimes f_i.$$

Similarly, we can write

$$s_i^{(3)} = p_i \otimes (f_i + s_i^{(2)}) = p_i \otimes f_i + p_i \otimes s_i^{(2)} = p_i \otimes f_i + p_i \otimes p_i \otimes f_i + p_i \otimes p_i \otimes p_i \otimes f_i$$

so that in general, for $n = 1, 2, ...$

$$s_i^{(n)} = p_i \otimes f_i + p_i \otimes s_i^{(n-1)}$$

where $s_i^{(0)} = 0$. Now, the term $p_i \otimes f_i$ is just a FIR filter describing how the input f_i is modified by the impulse response function p_i. The second term introduces the feedback process. Suppose we consider a filter q_i say, which allows us to write the iterative process

$$s_i^{(n)} = p_i \otimes s_i^{(n-1)}$$

in terms of the recursive process

$$s_i = q_i \otimes s_i.$$

On the basis of the above, it is then valid to consider a general linear filter of the form

$$s_i = p_i \otimes f_i + q_i \otimes s_i.$$

Now, if $q_i = 0$, then the FIR filter is obtained which is non-recursive. However, if $q_i \neq 0$, then the filter is recursive and is known as an Infinite Impulse Response or IIR filter. Unlike the computation of the FIR filter, in this case, we need to reserve space for the modified values of s_i as the computation proceeds; we cannot simply over write them into s_i directly. The filter must be calculated recursively and thus, there is no way of applying the filter to a single segment of a signal and IIR filters are said to be only suitable for infinite signals (hence the name). Note that the length of the data stream associated with the computation of the first and second terms of the IIR filter does not have to be same. Also note that in Fourier space, this result becomes

$$S_i = P_i F_i + Q_i S_i$$

or

$$S_i = R_i F_i$$

where

$$R_i = \frac{P_i}{1 - Q_i}$$

which is rational requiring that $Q_i \neq 1 \quad \forall i$.

16.5 Non-Stationary Problems

The principal model for a signal, i.e.

$$s = p \otimes f + n$$

assumes that the 'system' is time invariant and that the process is stationary. The term non-stationary is used in a number of circumstances and needs to be defined carefully whenever the term is used. Here, a non-stationary process refers to the case when:

(i) the noise statistics change with time;

(ii) the impulse response function changes with time;

(iii) both (i) and (ii) above apply.

We have already briefly mentioned the use of statistical type filters coupled with the moving window principle for the analysis of signals with time variant statistics which applies to the analysis of systems conforming to point (i) above. In this section, we focus on the case when the impulse response function is time variant. The essential point to understand is that when p changes with time, the convolution process becomes

$$s(t) = \int p(t - \tau, t) f(\tau) d\tau$$

rather that just

$$s(t) = \int p(t - \tau) f(\tau) d\tau$$

for which there is no equivalent convolution theorem. Thus, the application of this theorem for transforming in and out of Fourier space in order to develop the Fourier

filters discussed in Chapters 13, 14 and 15 no longer holds; however, the convolution process itself does. By way of an illustration consider the case where we convolve a data stream $f_1, f_2, f_3, ..., f_N$ with a 3×1 kernel

$$\mathbf{p} = (1+i, 2+i, 1+i)$$

Using the moving window principle, with zero padding we have

$$\begin{array}{rcl} s_1 & = & f_1, \\ s_2 & = & 2f_1 + 3f_2 + 2f_3, \\ s_3 & = & 3f_2 + 4f_3 + 3f_4, \\ & \vdots & \end{array}$$

In terms of a matrix representation, we have

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_{N-1} \\ s_N \end{pmatrix} = \begin{pmatrix} 3 & 2 & & & & & \\ 3 & 4 & 3 & & & & \\ & 4 & 7 & 4 & & & \\ & & & \ddots & & & \\ & & & N & 1+N+1 & N & \\ & & & & 2+N & 1+N & \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-1} \\ f_N \end{pmatrix}.$$

This matrix is tridiagonal but with elements that increase monotonically from the top-left to the bottom-right of the matrix. To deconvolve the signal $\mathbf{s}$, this system of equations needs to be solved directly using the algorithms discussed in Chapter 7. The example above illustrates the case when the kernel is time variant with regard to the values of its elements but the size of the kernel remains the same (in this case, a 3×1 vector). Another case is when the size of the kernel changes. To illustrate this, consider the kernel

$$\mathbf{p} = \mathbf{1}, \quad \mathbf{p} \in R^i.$$

Here, the kernel is a unit vector which linearly increases in size (i.e. the number of elements in the vector space). With zero padding, the matrix representation of this process becomes

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ \\ s_N \end{pmatrix} = \begin{pmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & & \\ & & & & \ddots & \\ 1 & 1 & 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ \\ f_N \end{pmatrix}.$$

Here, the characteristic matrix is lower diagonal and the solution for f_i is trivial and given by

$$f_i = s_i - s_{i-1} \equiv p_i \otimes s_i$$

where

$$p_i = (1, -1).$$

Note that s_i is the discrete integral of f_i and f_i is the discrete differential (forward difference) of s_i.

There are many other simple examples that can be used to illustrate the process of non-stationary convolution, but the essential issue, is that to undertake the inverse process, appropriate methods of solving the associated matrix equations are required and the deconvolution problem must be approach is real space. Note that as the bandwidth (the size of the vector space) of the kernel changes, so does the bandwidth of the matrix. Further, for non-stationary processes, where the kernel is relatively small compared to the data, the characteristic matrix is sparse and thus, iterative techniques become appropriate for the general case (i.e. assuming that the characteristic matrix is not symmetric positive definite for example).

16.6 Summary of Important Results

The FIR Filter

For an input digital signal f_i, the output of an FIR filter s_i is given by

$$s_i = p_i \otimes f_i \equiv \sum_{j=0}^{N-1} p_{i-j} f_j$$

for the convolution FIR filter and

$$s_i = p_i \odot f_i \equiv \sum_{j=0}^{N-1} p_{j-i} f_j$$

for the correlation FIR filter where p_i is the FIR filter or kernel.

The FIR filter and the Matrix Equation

If $\mathbf{f}$ is an input vector with N elements and $\mathbf{s}$ is an output vector with N, elements then the FIR filter can be written in the form

$$\mathbf{s} = P\mathbf{f}$$

where P is an $N \times N$ matrix whose elements are composed from the kernel $\mathbf{p}$.

Zero Padding

Computing the FIR filter by padding the input data with zeros.

End Point Extension

Computing the FIR filter by padding the input data with the values of the first and last elements of the data.

FIR filtering using an FFT

Computing the FIR filter by zero padding the kernel to the array size of the input data and using the convolution theorem to employ an FFT.

Computational Efficiency

For kernels that are small compared to the input, the FIR filter is efficiently computed by direct application. For kernels whose arrays are of similar size to the input data, the FIR is computed most efficiently by application of the FFT.

Moving Window Filtering

Filtering the input data by application of some process applied to a sample of data over a window that moves along the input data one element at a time.

The IIR filter

A recursive filter of the form

$$s_i = p_i \otimes f_i + q_i \otimes s_i$$

Non-stationary or Time Variant Processes

The process whereby the statistics of the noise changes with time and/or the convolution kernel changes (in value and/or array size) as the moving window process evolves, e.g.

$$s_i = \sum_j p_{i-j}[i] f_j.$$

Note there is no general convolution theorem for this case. However, non-stationary convolution processes can still be written in matrix form

$$\mathbf{s} = P\mathbf{f}.$$

Deconvolution problems associated with this process must then be undertaken in real space using appropriate computational methods in linear algebra. Non-stationary processes in signal analysis are very important simply because signals rarely behave in the same way over time and time invariant linear systems analysis is limited in its general application. Thus, developing non-stationary models for systems and signals is becoming more and more important. This theme is developed further in the following chapter which studies random fractal signals and the development of non-stationary or multi-fractal models.

16.7 Further Reading

- Rabiner L and Gold B, *Theory and Applications of Digital Signal Processing*, Prentice-Hall, 1975.
- Tretter S, *Introduction to Discrete Time Signal Processing*, Wiley, 1976.
- Oppenheim A and Shafer R, *Digital Signal Processing*, Prentice-Hall, 1975.
- Robinson E and Silvia M, *Digital Foundations of Time Series Analysis*, Holden-Day, 1979.
- Candy J V, *Signal Processing: The Model Based Approach*, McGraw-Hill, 1986.
- Skelton R E, *Dynamic Systems Control*, Wiley, 1988.

16.8 Programming Problems

In the questions that follow, the functions required should be void functions written in ANSI C. They should be compiled, tested and then added to a digital signal processing object library *dsplib.lib*. In each case, a simple I/O test procedure should be written; the I/O being studied graphically using the graphics routine provided and discussed in Appendix B working with arrays of size 64, 128, 256 or 512 samples with array processing from 1 to n where n (which is of type int) is the size of the array. Each function should be self-contained within the context of the process. For opt = 0, apply zero padding and for opt = 1, apply end point extension.

16.1 Write a subroutine to perform a discrete convolution of two digital signals - an FIR (convolution) filter.

```
function FIRCON(float f[ ], float p[ ], float s[ ], int n, int w, int opt)
```

where f is the input signal, p is the FIR (convolution) filter, s is the output and w is the size of the kernel.

Test your function by applying a kernel of the form (1,-1) to a tophat function. The result should be two spikes of opposite polarity - equivalent to applying a forward differencing scheme to compute the digital gradient of a signal.

16.2 Write a function to perform a discrete correlation of two digital signals - an FIR (correlation) filter.

```
function FIRCOR(float f[ ], float p[ ], float s[ ], int n, int w, int opt)
```

where f is the input signal, p is the FIR filter, s is the output and w is the the size of the kernel. Test your routine by autocorrelating a tophat function - the result should be a triangle.

16.3 Write a function to filter a signal using the moving average principle.

```
function MAVFIL(float s[ ], int n, int w, int opt)
```

where s is the input/filtered output and w is the size of the window.

16.4 Write a function to filter a signal using a median filter.

```
function MEDFIL(float s[ ], int n, int w)
```

where s is the input/filtered output and w is the size of the window which is odd, i.e. w=3,5,7...

16.5 Compare the difference in performance between the moving average and median filters by adding random Gaussian noise to a sine wave for example and filtering the result with a range of window sizes. Add some isolated noise spikes of unit value (i.e. Kronecker delta functions) to the signal and compare the output produced by the two filters. The noise spikes can be produced by thresholding the Gaussian noise, i.e.

$$\text{spikes}_i = \begin{cases} 1, & g_i > T; \\ 0, & g_i \leq T. \end{cases}$$

where g_i is the Gaussian noise, post-processed so that $\|g_i\|_\infty = 1$ and $0 < T < 1$ is a user defined threshold which determines the density of the noise spikes.

Chapter 17

Random Fractal Signals

Many signals observed in nature are random fractals. Random fractals are signals that exhibit the same statistics at different scales. In other words, they are signals whose frequency distribution (Probability Distribution Function or PDF) has the same 'shape' irrespective of the scale over which they are observed. Thus, random fractal signals are (statistically) self-similar; they 'look the same' (in a stochastic sense) at different scales.

We can define this property as follows: Suppose that $s(t)$ is a statistically self-similar stochastic field with a PDF denoted by $\Pr[s(t)]$, then, if λ is a scaling parameter,

$$\lambda \Pr[s(t)] = \Pr[s(\lambda t)].$$

Here, the PDF of the signal $s(\lambda t)$ observed over a scale of λ is a scaled down version of the PDF of $s(t)$ (i.e. the PDF of $s(t)$ is multiplied by λ). More generally, random fractal signals are statistically self-affine, conforming to the property

$$\lambda^q \Pr[s(t)] = \Pr[s(\lambda t)], \quad q > 0.$$

Such signals are characterized by power spectra whose frequency distribution is proportional to $1/\omega^q$ where ω is the (angular) frequency and $q > 0$ is the 'Fourier Dimension' (a value that is simply related to the 'Fractal Dimension' and is discussed later on in this chapter). In cases where the signal is governed by a stationary process, the value of q is constant. However, when the process is non-stationary and (random) fractal, the value of q may change. Such signals are common; they are examples of multi-fractal behaviour and finding a theoretical basis for modelling and interpreting such signals is important in many areas of science and engineering.

In addition to the random fractal geometry of signals, there are a number of natural signals that can be considered to be the result of a chaotic process (time invariant or otherwise). Such systems can often be modelled in terms of deterministic chaos which is itself based on a variety of (typically iterative) models obtained from some nonlinear equation or system of equations. An important aspect of chaotic signals is that when the data as analysed in an appropriate way (in an appropriate phase space or using a so called Feigenbaum diagram - see Chapter 14), self-affine structures are observed. In this sense, there is a close connection between nonlinear (chaotic) systems[1] and

[1]Not all nonlinear systems exhibit chaos.

fractal geometry; the geometric interpretation and characteristics of chaotic signals is typically undertaken in terms of their fractal geometry in an analogous approach to the way in which the interpretation and characteristics of linear signals is undertaken in terms of Euclidean geometry (e.g. plotting a graph!).

In this chapter, the principles of fractal geometry are applied to fractal signal synthesis and analysis using an approach in which fractional calculus plays a central role. In particular, a non-stationary approach to simulating fractal signals is developed. Two case studies are provided which discuss the application of this approach to covert communications and financial time series analysis.

17.1 Introduction

Developing mathematical models to simulate and analyse noise has an important role in digital signal processing. Computer generated noise is routinely used to test the robustness of different types of algorithms (e.g. algorithms whose principal goal is to extract information from noise). Further, noise generators are used for data encryption and to enhance or amplify signals through the process of 'stochastic resonance', e.g. the correlation of noise with noise for information extraction

Accurate statistical models for noise (e.g. the PDF or the Characteristic Function, i.e. the Fourier transform of the PDF) are particularly important in signal restoration using Bayesian estimation (see Chapter 15), for signal reconstruction and in the segmentation of coherent images in which 'speckle' (arguably a special type of noise, i.e. coherent Gaussian noise, which, to a first approximation, is Gamma distributed) is a prominent feature. The noise characteristics of a given system often dictate the type of filters that are used to process and analyse the data. Noise simulation is also important in the synthesis of signals and images used in computer graphics and computer animation systems in which fractal noise has a special place.

The application of fractal geometry for modelling naturally occurring signals is well known. This is due to the fact that the 'statistics' and spectral characteristics of Random Scaling Fractals (RSFs) are consistent with many objects found in nature, a characteristic that is compounded in the term 'statistical self-affinity'. This term refers to random processes whose statistics are scale invariant. A RSF signal is one whose PDF remains the same irrespective of the scale over which the signal is sampled. Thus, as we zoom into a RSF signal, although the time signature changes, the PDF of the signal remains the same (a scaled down version of the 'original') and a concept that is aptly compounded in the Chinese proverb: *'In every way one can see the shape of the sea'*. There is another presumably Chinese proverb concerning this theme that aptly describes human beings that exhibit self-affine personalities, namely, *'He who goes to work with hole in pocket, feel cocky all day'*.

Many signals found in nature are statistically self-affine. For example, speech signals tend to exhibit the characteristics of RSFs as do other signals such as financial time series, seismic signals and music (irrespective of the culture from which it has been derived). The incredible range of vastly different systems which exhibit random fractal behaviour is leading the scientific community to consider statistical self-affinity to be a universal law, a law that is particularly evident in systems which are undergoing a phase transition.

In a stable state, the behaviour of the elements from which a system is composed depends primarily on their local neighbours and the statistics of the system is not self-affine. In a critical state, the elements become connected, propagating 'order' throughout the system in the sense that the statistical characteristics of the system are self-affine with 'system wide' correlations. This is more to do with the connectivity of the elements than the elements themselves. (Critical states can of course be stable in the dynamical sense.) Moreover, critical states appear to be governed by the power law

$$\text{System(size)} \propto \frac{1}{\text{size}^q}$$

where $q > 0$ is a non-integer. Here, the term 'System' is a generic term representative of some definable parameter that can be measured experimentally over different scales of a certain 'size'. This power law is the principal 'signature' that the system is behaving in a statistically self-affine way. There are a wide variety of examples which demonstrate this power law. For example, the growth rate of companies tends to diminishes with size, irrespective of the type of business being conducted; typical US companies are characterized by $q \in [0.7, 0.9]$. This also applies to the death rate of companies, i.e. those that are forced into liquidation. The frequency of the creation and extinction of species (as revealed through a growing number of fossil records) is starting to indicate that the pattern of fitness for survival is statistically self-affine. The distribution of base pairs in DNA is statistically self-affine, i.e. the frequency of occurrence of Adenine-Thymine and Cytosine-Guanine in a DNA molecule is the same at different scales. DNA is in effect, a self-affine bit stream.

The power law given above which governs so many of natures signals and systems is a universal law. However, to date, there is no general mechanism or deeper understanding through which this law can be derived. It is like Newton's universal law of gravitation in which two bodies exert a force upon each other which is inversely proportional to the square of the distance between them and, like Newton's law (and other universal physical laws), although complex in its action, it is beautifully simple in its pattern. Thus, in introducing this power law, it is worth reflecting on Newton's response to criticism over his theory of gravitation: '... I have told you how it works, not why'.

Conventional RSF models are based on stationary processes in which the 'statistics' of the signal are invariant of time. However, many signals exhibit non-stationary behaviour. In addition, many signals exhibit episodes which are rare but extreme (sudden changes in amplitude and/or frequency), events which are statistically inconsistent with the 'normal' behaviour of the signal. These episodes include so-called Lévy flights in cases when the statistics of a signal conform to that of a Lévy distribution; a power-law distribution of the type $1/x^{1+q}$, $0 < q < 2$ that is used to investigate the 'strange kinetics' of systems undergoing phase transitions including hot plasmas, super-fluids, super-conducting materials and economic systems.

In this chapter, a model is developed which attempts to unify these features of stochasticism using a phenomenological approach. The model is based on a modification to the stochastic diffusion equation in which a fractional temporal derivative to an order $q(t)$ is introduced. By considering a model for the PDF of $q(t)$, a solution is derived which allows a stochastic field to be computed that is fractal, non-stationary and where the likelihood of events called 'Brownian flights' (effects which are analo-

gous to Lévy-type flights) can be altered via the PDF. The following section gives a brief overview of the different aspects of stochasticism which form the background to the postulation and analysis of this model.

17.2 Stochastic Modelling Revisited

There are two principal criteria used to define the characteristics of a stochastic field:

(i) The PDF or the Characteristic Function (i.e. the Fourier transform of the PDF).

(ii) The Power Spectral Density Function (PSDF).

The PSDF is the function that describes the envelope or shape of the power spectrum of the field and is related to the autocorrelation function of a signal through the autocorrelation theorem. In this sense, the PSDF is a measure of the time correlations of a signal. For example, zero-mean white Gaussian noise is a stochastic field characterized by a PSDF that is effectively constant over all frequencies and has a PDF with a Gaussian profile whose mean is zero.

Stochastic fields can of course be characterized using transforms other than the Fourier transform (from which the PSDF is obtained). However, the conventional PDF-PSDF approach serves many purposes in stochastic signals theory.

There are two conventional approaches to simulating a stochastic field. The first of these is based on predicting the PDF (or the Characteristic Function) theoretically (if possible). A pseudo random number generator is then designed whose output provides a discrete stochastic field that is characteristic of the predicted PDF. For example, a Gaussian pseudo random number generator can be designed using the Box-Muller transformation operating on uniform deviates (see Chapter 14). The second approach is based on considering the PSDF of a signal which, like the PDF, is ideally derived theoretically. The stochastic field is then typically simulated by filtering white noise. Many stochastic fields observed in nature have two fundamental properties:

(i) the PSDF is determined by irrational power laws, i.e. $1/\mid \omega \mid^q$ noise where ω is the (angular) frequency and $q > 0$;

(ii) the field is statistical self-affine.

(iii) the PDF is Lévy-type distributed.

What is a good stochastic model?

A 'good' stochastic model is one that accurately predicts both the PDF and the PSDF of the data. It should take into account the fact that in general, stochastic processes are non-stationary. In addition, it should, if appropriate, include behaviour that is characteristic of fractal walks and be able to produce rare but extreme events in which large deviations from the norm occur - effects that might be considered analogous to, but not necessarily formally related to Lévy flights. Note that although we refer to Lévy flights and/or distributions, such references should be taken to be qualitative in nature and are being used only in terms of introducing an analogy to a strictly well

defined process or term (i.e. Lévy flight or Lévy distribution respectively) which can yield analogous or similar effects. Note there is no connection between the theoretical prediction and/or the experimental determination of the PDF and the PSDF, i.e. there is no direct relationship between the characteristics of the Fourier transform of a stochastic field and the Fourier transform of the PDF of the same field - one cannot compute directly the PDF of a stochastic field from its PSDF or the PSDF from its PDF.

Lévy Flights and Distributions

Named after the French mathematician Paul Lévy (1886-1971), Lévy flights are random walks whose distribution has infinite moments. The statistics of (conventional) physical systems are usually concerned with stochastic fields that have PDFs where (at least) the first two moments (the mean and variance) are well defined and finite. Lévy statistics is concerned with statistical systems where all the moments (starting with the mean) are infinite.

Many distributions exist where the mean and variance are finite but are not representative of the process, e.g. the tail of the distribution is significant, where rare but extreme events occur. These distributions include Lévy distributions. Lévy's original approach[2] (which was developed in the late 1930s) to deriving such distributions is based on the following question: Under what circumstances does the distribution associated with a random walk of a few steps look the same as the distribution after many steps (except for scaling)? This question is effectively the same as asking under what circumstances do we obtain a random walk that is statistically self-affine. For a 1D random walk, the characteristic function $P(k)$ of such a distribution $p(x)$ was first shown by Lévy to be given by (for symmetric distributions only)

$$P(k) = \exp(-a \mid k \mid^q), \quad 0 < q < 2$$

where a is a (positive) constant. If $q = 0$,

$$p(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(-a) \exp(ikx) dk = \exp(-a)\delta(x)$$

and the distribution is concentrated solely at the origin as described by the delta function $\delta(x)$. When $q = 1$, the Cauchy distribution

$$p(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(-a \mid k \mid) \exp(ikx) dk = \frac{1}{\pi} \frac{a}{a^2 + x^2}$$

is obtained and when $q = 2$, $p(x)$ is characterized by the Gaussian distribution

$$p(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(-ak^2) \exp(ikx) dk = \frac{1}{2\pi} \sqrt{\frac{\pi}{a}} \exp[-x^2/(4a)],$$

[2]P Lévy was the research supervisor of B Mandelbrot, the 'inventor' of 'fractal geometry'

whose first and second moments are finite. The Cauchy distribution has a relatively long tail compared with the Gaussian distribution and a stochastic field described by a Cauchy distribution is likely to have more extreme variations when compared to a Gaussian distributed field. For values of q between 0 and 2, Lévy's characteristic function corresponds to a PDF of the form

$$p(x) \sim \frac{1}{x^{1+q}}$$

for $\mid x \mid >> 1$. For $q \geq 2$, the second moment of this distribution exists and the sums of large numbers of independent trials are Gaussian. For example, if the result were a random walk with a step length distribution governed by this PDF, then the result would be normal (Gaussian) diffusion, or Brownian motion. For $q < 2$ the second moment of this PDF (the mean square), diverges and the characteristic scale of the walk is lost. This type of random walk is called a fractal walk, or a Leýy flight. Figure 17.1 shows 10,000 steps for such a walk with $q = 1.5$. The statistics of the walk conform to a Lévy distribution rather than a Gaussian. In this way, Lévy distributions are a generalization of Gaussian distributions that include infinite variance and therefore fractal scaling behaviour. Lévy distributions offer a better description of random fields with long PDF tails although infinite variance is not always observed (or the variance is very slow to converge).

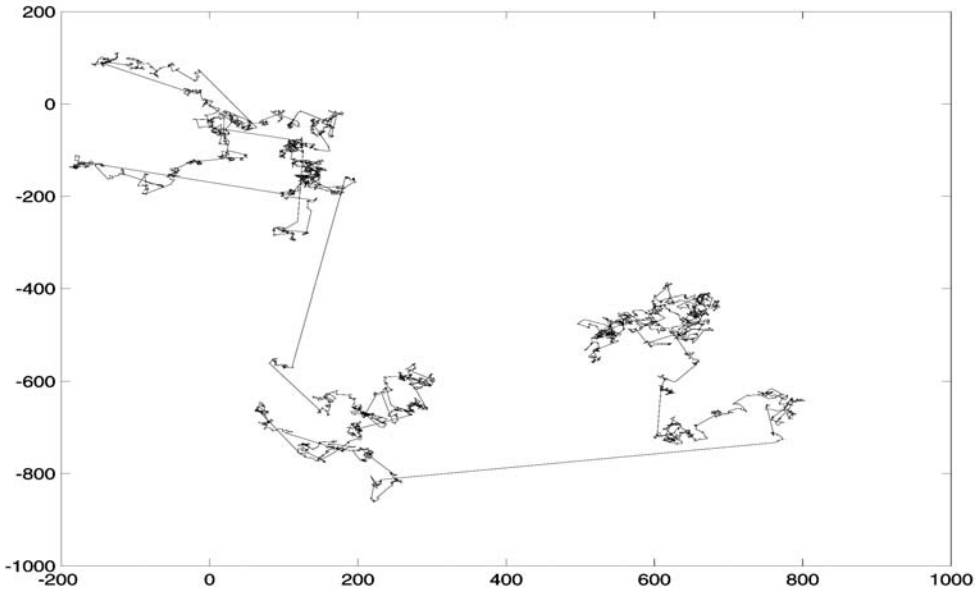


Figure 17.1: A Lévy flight with $p(x) \propto x^{-1.5}$

17.3 Fractional Calculus

In a famous letter from l'Hospital to Leibnitz written in 1695, l'Hospital asked the following question: 'Given that $d^n f/dt^n$ exists for all integer n, what if n be $\frac{1}{2}$'. The reply from Leibnitz was all the more interesting: 'It will lead to a paradox ... From this paradox, one day useful consequences will be drawn'. Fractional calculus has been studied for many years by some of the great names of mathematics since the development of (integer) calculus in the late seventeenth century. Relatively few papers and books exist on such a naturally important subject. However, a study of the works in this area of mathematics clearly show that the ideas used to define a fractional differential and a fractional integral are based on definitions which are in effect, little more than generalizations of results obtained using integer calculus. The classical fractional integral operators are the Riemann-Liouville transform

$$\hat{I}^q f(t) = \frac{1}{\Gamma(q)} \int_{-\infty}^{t} \frac{f(\tau)}{(t-\tau)^{1-q}} d\tau, \quad q > 0$$

and the Weyl transform

$$\hat{I}^q f(t) = \frac{1}{\Gamma(q)} \int_{t}^{\infty} \frac{f(\tau)}{(t-\tau)^{1-q}} d\tau, \quad q > 0$$

where

$$\Gamma(q) = \int_0^\infty t^{q-1} \exp(-t) dt.$$

For integer values of q (i.e. when $q = n$ where n is a non-negative integer), the Riemann-Liouville transform reduces to the standard Riemann integral. This transform is just a (causal) convolution of the function $f(t)$ with $t^{q-1}/\Gamma(q)$. For fractional differentiation, we can perform a fractional integration of appropriate order and then differentiate to an appropriate integer order. The reason for this is that direct fractional differentiation can lead to divergent integrals. Thus, the fractional differential operator $\hat{D}^q$ for $q > 0$ is given by

$$\hat{D}^q f(t) \equiv \frac{d^q}{dt^q} f(t) = \frac{d^n}{dt^n} [\hat{I}^{n-q} f(t)].$$

Another (conventional) approach to defining a fractional differential operator is based on using the formula for n^{th} order differentiation (obtained by considering the definitions for the first, second, third etc. differentials using backward differencing) and then generalising the formula by replacing n with q. This approach provides us with the result

$$\hat{D}^q f(t) = \lim_{N \to \infty} \left[\frac{(t/N)^{-q}}{\Gamma(-q)} \sum_{j=0}^{N-1} \frac{\Gamma(j-q)}{\Gamma(j+1)} f\left(t - j\frac{t}{N}\right) \right].$$

A review of this result shows that for $q = 1$, this is a point process but for other values it is not, i.e. the evaluation of a fractional differential operator depends on the history of the function in question. Thus, unlike an integer differential operator, a fractional differential operator has 'memory'. Although the memory of this process fades, it does not do so quickly enough to allow truncation of the series in order to retain acceptable accuracy. The concept of memory association can also be seen from the result

$$\hat{D}^q f(t) = \frac{d^n}{dt^n}[\hat{I}^{n-q} f(t)]$$

where

$$\hat{I}^{q-n} f(t) = \frac{1}{\Gamma(n-q)} \int_{-\infty}^{t} \frac{f(\tau)}{(t-\tau)^{1+q-n}} d\tau, \quad n - q > 0$$

in which the value of $\hat{I}^{q-n} f(t)$ at a point t depends on the behaviour of $f(t)$ from $-\infty$ to t via a convolution with the kernel $t^{n-q}/\Gamma(q)$. The convolution process is of course dependent on the history of the function $f(t)$ for a given kernel and thus, in this context, we can consider a fractional derivative defined via the result above to have memory.

17.3.1 The Laplace Transform and the Half Integrator

It informative at this point to consider the application of the Laplace transform to identify an ideal integrator and then a half integrator. The Laplace transform is given by (see Chapter 5)

$$\hat{L}[f(t)] \equiv F(p) = \int_0^{\infty} f(t) \exp(-pt) dt$$

and from this result we can derive the transform of a derivative given by

$$\hat{L}[f'(t)] = pF(p) - f(0)$$

and the transform of an integral given by

$$\hat{L}\left[\int_0^t f(\tau) d\tau\right] = \frac{1}{p} F(p).$$

Now, suppose we have a standard time invariant linear system whose input is $f(t)$ and whose output is given by

$$s(t) = f(t) \otimes g(t)$$

where the convolution is causal, i.e.

$$s(t) = \int_0^t f(\tau) g(t-\tau) d\tau.$$

Suppose we let

$$g(t) = H(t) = \begin{cases} 1, & t > 0; \\ 0, & t < 0. \end{cases}$$

Then, $G(p) = 1/p$ and the system becomes an ideal integrator:

$$s(t) = f(t) \otimes H(t) = \int_0^t f(t-\tau)d\tau = \int_0^t f(\tau)d\tau.$$

Now, consider the case when we have a time invariant linear system with an impulse response function by given by

$$g(t) = \frac{H(t)}{\sqrt{t}} = \begin{cases} |\ t\ |^{-1/2}, & t > 0; \\ 0, & t < 0. \end{cases}$$

The output of this system is $f \otimes g$ and the output of such a system with input $f \otimes g$ is $f \otimes g \otimes g$. Now

$$g(t) \otimes g(t) = \int_0^t \frac{d\tau}{\sqrt{\tau}\sqrt{t-\tau}} = \int_0^{\sqrt{t}} \frac{2xdx}{x\sqrt{t-x^2}} = 2\left[\sin^{-1}\left(\frac{x}{\sqrt{t}}\right)\right]_0^{\sqrt{t}} = \pi.$$

Hence,

$$\frac{H(t)}{\sqrt{\pi t}} \otimes \frac{H(t)}{\sqrt{\pi t}} = H(t)$$

and the system defined by the impulse response function $H(t)/\sqrt{\pi t}$ represents a 'half-integrator' with a Laplace transform given by

$$\hat{L}\left[\frac{H(t)}{\sqrt{\pi t}}\right] = \frac{1}{\sqrt{p}}.$$

This result provides an approach to working with fractional integrators and/or differentiators using the Laplace transform. Fractional differential and integral operators can be defined and used in a similar manner to those associated with conventional or integer order calculus and we now provide an overview of such operators.

17.3.2 Operators of Integer Order

The following operators are all well-defined, at least with respect to all test functions $u(t)$ say which are (i) infinitely differentiable and (ii) of compact support (i.e. vanish outside some finite interval).

Integral Operator:

$$\hat{I}u(t) \equiv \hat{I}^1 u(t) = \int_{-\infty}^t u(\tau)d\tau.$$

Differential Operator:

$$\hat{D}u(t) \equiv \hat{D}^1 u(t) = u'(t).$$

Identify Operator:

$$\hat{I}^0 u(t) = u(t) = \hat{D}^0 u(t).$$

Now,

$$\hat{I}[\hat{D}u](t) = \int_{-\infty}^{t} u'(\tau)d\tau = u(t)$$

and

$$\hat{D}[\hat{I}u](t) = \frac{d}{dt}\int_{-\infty}^{t} u(\tau)d\tau = u(t)$$

so that

$$\hat{I}^1\hat{D}^1 = \hat{D}^1\hat{I}^1 = \hat{I}^0.$$

For n (integer) order:

$$\hat{I}^n u(t) = \int_{-\infty}^{t} d\tau_{n-1} ... \int_{-\infty}^{\tau_2} d\tau_1 \int_{-\infty}^{\tau_1} u(\tau)d\tau,$$

$$\hat{D}^n u(t) = u^{(n)}(t)$$

and

$$\hat{I}^n[\hat{D}^n u](t) = u(t) = \hat{D}^n[\hat{I}^n u](t).$$

17.3.3 Convolution Representation

Consider the function

$$t_+^{q-1}(t) \equiv \mid t \mid^{q-1} H(t) = \begin{cases} \mid t \mid^{q-1}, & t > 0; \\ 0, & t < 0. \end{cases}$$

which, for any $q > 0$ defines a function that is locally integrable. We can then define an integral of order n in terms of a convolution as

$$\hat{I}^n u(t) = \left(u \otimes \frac{1}{(n-1)!} t_+^{n-1} \right)(t) = \frac{1}{(n-1)!}\int_{-\infty}^{t} (t-\tau)^{n-1} u(\tau)d\tau$$

$$= \frac{1}{(n-1)!}\int_{-\infty}^{t} \tau^{n-1} u(t-\tau)d\tau$$

In particular,

$$\hat{I}^1 u(t) = (u \otimes H)(t) = \int_{-\infty}^{t} u(\tau)d\tau.$$

These are classical (absolutely convergent) integrals and the identity operator admits a formal convolution representation, using the delta function, i.e.

$$\hat{I}^0 u(t) = \int_{-\infty}^{\infty} \delta(\tau) u(t-\tau) d\tau$$

where

$$\delta(t) = \hat{D} H(t).$$

Similarly,

$$\hat{D}^n u(t) \equiv \hat{I}^{-n} u(t) = \int_{-\infty}^{\infty} \delta^{(n)}(\tau) u(t-\tau) d\tau = u^{(n)}(t).$$

On the basis of the material discussed above, we can now formally extend the integral operator to fractional order and consider the operator

$$\hat{I}^q u(t) = \frac{1}{\Gamma(q)} \int_{-\infty}^{\infty} u(\tau) t_+^{q-1}(t-\tau) d\tau = \frac{1}{\Gamma(q)} \int_{-\infty}^{t} u(\tau) t_+^{q-1}(t-\tau) d\tau$$

where

$$\Gamma(q) = \int_0^{\infty} t^{q-1} \exp(-t) dt, \quad q > 0$$

with the fundamental property that

$$\Gamma(q+1) = q\Gamma(q).$$

Here, I^q is an operator representing a time invariant linear system with impulse response function $t_+^{q-1}(t)$ and transfer function $1/p^q$. For the cascade connection of I^{q_1} and I^{q_2} we have

$$\hat{I}^{q_1}[\hat{I}^{q_2} u(t)] = \hat{I}^{q_1+q_2} u(t).$$

This classical convolution integral representation holds for all real $q > 0$ (and formally for $q = 0$, with the delta function playing the role of an impulse function and with a transfer function equal to the constant 1).

17.3.4 Fractional Differentiation

For $0 < q < 1$, if we define the (Riemann-Liouville) derivative of order q as

$$\hat{D}^q u(t) \equiv \frac{d}{dt}[\hat{I}^{1-q} u](t) = \frac{1}{\Gamma(1-q)} \frac{d}{dt} \int_{-\infty}^{t} (t-\tau)^{-q} u(\tau) d\tau,$$

then,

$$\hat{D}^q u(t) = \frac{1}{\Gamma(1-q)} \int_{-\infty}^{t} (t-\tau)^{-q} u'(\tau) d\tau \equiv \hat{I}^{1-q} u'(t).$$

Hence,

$$\hat{I}^q[\hat{D}^q u] = \hat{I}^q[\hat{I}^{1-q}u'] = \hat{I}^1 u' = u$$

and $\hat{D}^q$ is the formal inverse of the operator $\hat{I}^q$. Given any $q > 0$, we can always write $\lambda = n - 1 + q$ and then define

$$\hat{D}^\lambda u(t) = \frac{1}{\Gamma(1-q)} \frac{d^n}{dt^n} \int_{-\infty}^{t} u(\tau)(t-\tau)^{-q} d\tau.$$

D^q is an operator representing a time invariant linear system consisting of a cascade combination of an ideal differentiator and a fractional integrator of order $1 - q$. For D^λ we replace the single ideal differentiator by n such that

$$\hat{D}^0 u(t) = \frac{1}{\Gamma(1)} \frac{d}{dt} \int_{-\infty}^{t} u(\tau) d\tau = u(t) \equiv \int_{-\infty}^{\infty} u(\tau)\delta(t-\tau) d\tau$$

and

$$\hat{D}^n u(t) = \frac{1}{\Gamma(1)} \frac{d^{n+1}}{dt^{n+1}} \int_{-\infty}^{t} u(\tau) d\tau = u^{(n)}(t) \equiv \int_{-\infty}^{\infty} u(\tau)\delta^{(n)}(t-\tau) d\tau.$$

In addition to the conventional and classical definitions of fractional derivatives and integrals, more general definitions have recently been developed including the Erdélyi-Kober operators, hypergeometric operators and operators involving other special functions such as the Maijer G-function and the Fox H-function. Moreover, all such operators leading to a fractional integral of the Riemann-Liouville type and the Weyl type would appear (through induction) to have the general forms

$$\hat{I}^q f(t) = t^{q-1} \int_{-\infty}^{t} \Phi\left(\frac{\tau}{t}\right) \tau^{-q} f(\tau) d\tau$$

and

$$\hat{I}^q f(t) = t^{-q} \int_{t}^{\infty} \Phi\left(\frac{t}{\tau}\right) \tau^{q-1} f(\tau) d\tau$$

respectively, where the kernel Φ is an arbitrary continuous function so that the integrals above make sense in sufficiently large functional spaces. Although there are a number of approaches that can be used to define a fractional differential/integral, there is one particular definition, which in terms of its 'ease of use' and wide ranging applications, is of significant value and is based on the Fourier transform, i.e.

$$\frac{d^q}{dt^q} f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} (i\omega)^q F(\omega) \exp(i\omega t) d\omega, \quad -\infty < q < \infty$$

where $F(\omega)$ is the Fourier transform of $f(t)$. When $q = 1, 2, 3...$, this definition reduces to a well known result that is trivial to derive in which for example, the 'filter' $i\omega$

(for the case when $q = 1$) is referred to as a 'differentiator'. When $q < 0$, we have a definition for the fractional integral where, in the case of $q = -1$ for example, the filter $(i\omega)^{-1}$ is an 'integrator'. When $q = 0$ we just have $f(t)$ expressed in terms of its Fourier transform $F(\omega)$. This Fourier based definition of a fractional derivative can be extended further to include a definition for a 'fractional Laplacean' ∇^q where for n dimensions

$$\nabla^q \equiv -\frac{1}{(2\pi)^n} \int d^n\mathbf{k} k^q \exp(i\mathbf{k} \cdot \mathbf{r}), \quad k = \mid \mathbf{k} \mid$$

are $\mathbf{r}$ is an n-dimensional vector. This is the fractional Riesz operator. It is designed to provide a result that is compatible with the case of $q = 2$ for $n > 1$, i.e. $\nabla^2 \Longleftrightarrow -k^2$ (which is the reason for introducing the negative sign). Another equally valid generalization is

$$\nabla^q \equiv \frac{1}{(2\pi)^n} \int d^n\mathbf{k} (ik)^q \exp(i\mathbf{k} \cdot \mathbf{r}), \quad k = \mid \mathbf{k} \mid$$

which introduces a q dependent phase factor of $\pi q/2$ into the operator.

In general terms, although it is possible to compute fractional integral and differential operators using the results discussed above, neither a fractional differential (or a fractional integral) operator appear to have a geometric and/or physical interpretation unlike an integer differential, which can be considered to be a map of the gradient of a piecewise continuous function (for which a picture can be drawn at any point by zooming into the function and arguing that over a small enough portion of the function, a straight line can be considered which has a definable gradient). However, when generalized functions are introduced, the concept of definable gradients requires attention and generalized functions such as the delta function $\delta(t)$ are only defined properly in terms of the role they play in certain transforms, e.g. the sampling property of the delta function. There have been some attempts to develop a geometric interpretation of a fractional differential and/or integral operator. In the few published works that consider this problem, relationships have been explored between fractional calculus and fractal geometry using a few specific examples and in some cases, it has been concluded that no direct relationship between fractional calculus and fractal geometry has yet been established. Thus, the establishment of a proper geometrical interpretation of a fractional derivative is still 'up for grabs' and we arrive at a rather fundamental and open question: Is there a geometrical representation of a fractional derivative? If not, can one prove that a graphical representation of a fractional derivative does not exist? The general consensus of opinion is that there is no simple geometrical interpretation of a derivative of fractional order and that if there is, then as Virginia Kiryakova concludes in her book on 'Generalized Fractional Calculus and Applications', '... it is likely to be found in our fractal world'. An approach to quantifying the relationship between a fractal object and fractional calculus is through a parametric approach. Consider a fractal object with a well defined Fourier dimension which is composed of an infinite set of points in the plane defined by the coordinates functions $[x(t), y(t)]$ where $t \in [0, 1]$. This parametric representation of the fractal consists of the independent signals $x(t)$ and $y(t)$. If the spectra of $x(t)$ and $y(t)$ are characterized by a PSDF of the type $(i\omega)^{-q}$ then, by inference, they are signals that can be represented in terms of the Riemann-Liouville or fractional integral transform. Thus, through the geometrical parametrization of deterministic

fractal objects we can represent such objects in terms of a signal or set of signals which have self-affine properties that can be expressed via a fractional integral. In principle, the signal obtained should have a Fourier dimension that is independent of the parametrization that is applied (uniform, chord length, arc length etc.).

17.3.5 Fractional Dynamics

Mathematical modelling using (time dependent) fractional Partial Differential Equations (PDEs) is generally known as fractional dynamics. A number of works have shown a close relationship between fractional diffusion equations of the type (where p is the space-time dependent PDF and τ is the generalized diffusivity)

$$\nabla^2 p - \tau \frac{\partial^q}{\partial t^q} p = 0, \quad 0 < q \leq 1$$

and

$$\nabla^q p - \tau \frac{\partial}{\partial t} p = 0, \quad 0 < q \leq 2$$

and continuous time random walks with either temporal or spatial scale invariance (fractal walks). Fractional diffusion equations of this type have been shown to produce a framework for the description of anomalous diffusion phenomena and Lévy-type behaviour. In addition, certain classes of fractional differential equations are known to yield Lévy-type distributions. For example, the normalized one-sided Lévy-type PDF

$$p(x) = \frac{a^q}{\Gamma(q)} \frac{\exp(-a/x)}{x^{1+q}}, \quad a > 0, \quad x > 0$$

is a solution of the fractional integral equation

$$x^{2q} p(x) = a^q \hat{I}^{-q} p(x)$$

where

$$\hat{I}^{-q} p(x) = \frac{1}{\Gamma(q)} \int_0^x \frac{p(y)}{(x-y)^{1-q}} dy, \quad q > 0.$$

Another example involves the solution to the anomalous diffusion equation

$$\nabla^q p - \tau \frac{\partial}{\partial t} p = 0, \quad 0 < q \leq 2.$$

Fourier transforming this equation and using the fractional Riesz operator defined previously, we have

$$\frac{\partial}{\partial t} P(k,t) = -\frac{1}{\tau} k^q P(k,t)$$

which has the general solution

$$P(k,t) = \exp(-t \mid k \mid^q / \tau), \quad t > 0.$$

Comparing this result with

$$P(k) = \exp(-a \mid k \mid^q), \quad 0 < q \leq 2$$

we recognise the characteristic function of a Lévy distribution with $a = t/\tau$. This analysis can be extended further by considering a fractal based generalization of the Fokker-Planck-Kolmogorov (FPK) equation

$$\frac{\partial^q}{\partial t^q} p(x,t) = \frac{\partial^\beta}{\partial x^\beta}[s(x)p(x,t)]$$

where s is an arbitrary function and $0 < q \leq 1$, $0 < \beta \leq 2$. This equation is referred to as the fractal FPK equation; the standard FPK equation is of course recovered for $q = 1$ and $\beta = 2$. The characteristic function associated with $p(x,t)$ is given by

$$P(k,t) = \exp(-ak^\beta t^q)$$

where a is a constant which again, defines a Lévy distribution. Finally, d-dimensional fractional master equations of the type (for example)

$$\frac{\partial^q}{\partial t^q} p(\mathbf{r},t) = \sum_{\mathbf{s}} w(\mathbf{r}-\mathbf{s})p(\mathbf{s},t), \quad 0 < q \leq 1$$

can be used to model non-equilibrium phase transitions where p denotes the probability of finding the diffusing entity at a position $\mathbf{r} \in R^d$ at time t (assuming that it was at the origin $\mathbf{r} = \mathbf{0}$ at time $t = 0$) and w are the fractional transition rates which measure the propensity for a displacement $\mathbf{r}$ in units of $1/(\text{time})^q$. These equations conform to the general theory of continuous time random walks and provide models for random walks of fractal time.

A study of the work on fractional dynamics reveals that the fractional PDEs proposed are being solved through application of the Fourier based definition of a fractional differential operator. One could therefore argue that such equations are being 'invented' to yield a desired result in Fourier space (e.g. the characteristic function for a Lévy distribution). Moreover, definitions of fractional derivatives are being considered in such a way that they lead to self-consistent results for the integer case (e.g. the Riesz definition of a fractional Laplacean). In this sense, taking well known PDEs and 'fractionalising' them in such a way that a Fourier transform yields the desired result might justifiably be considered as an example of phenomenology at its worst. It is clearly more desirable to derive a PDE from basic principles (based on the known physical laws) and then solve the equation using appropriate solution methods (some of which are based on certain integral transforms including the Fourier transform). On the other hand, there are a number of important PDEs whose form is based on postulation alone and cannot be derived or proved but nevertheless yield remarkably accurate models for the behaviour of a physical system informed by experiment, e.g. the Schrödinger equation and the Dirac field equations. However, the measurable quantity is not always that for which a solution can be derived. Referring to the following table,

Name	Basic operator	Solution variable	Experimental measurable
Diffusion equation	$\nabla^2 - \frac{\partial}{\partial t}$	$u(\mathbf{r},t)$	$u(\mathbf{r},t)$
Schrödinger equation	$\nabla^2 + i\frac{\partial}{\partial t}$	$u(\mathbf{r},t)$	$\mid u(\mathbf{r},t) \mid^2$
Wave equation	$\nabla^2 - \frac{\partial^2}{\partial t^2}$	$u(\mathbf{r},t)$	$u(\mathbf{r},t)$ or $\mid u(\mathbf{r},t) \mid^2$

we note that the diffusion and wave equations can be derived rigorously from a range of fundamental physical laws (Fourier's law of thermal conduction, conservation of mass, conservation of momentum, the continuity equation, Newton's laws of motion, Maxwell's equations and so on) but that Schrödinger's equation is phenomenological. Further, although we can solve Schrödinger's equation for the field variable, the probability wave u, we can only ever measure $\mid u \mid^2$ (the probability of the occur $\mathbf{r}$ and time t). This feature is analogous to the approach that follows, where the phenomenological operator[3]

$$\nabla^2 - \frac{\partial^{q(t)}}{\partial t^{q(t)}}$$

is introduced and solved for the non-stationary variable $u(\mathbf{r},t)$ where only $\Pr[u(\mathbf{r},t)]$ can be measured experimentally ($\Pr[u(\mathbf{r},t)]$ denotes the space-time PDF of u).

17.4 Non-stationary Fractional Dynamic Model

A common theme associated with the fractional dynamic models discussed in the previous section is that they describe stationary random processes in terms of solutions to a PDE. Here, we postulate a PDE whose characteristics incorporate behaviour that describes non-stationary fractal walks and Lévy-type flights in terms of a solution to the stochastic field itself rather than its PDF. Also, as will be discussed later, within the context of the solution proposed, these so called Lévy-type flights are actually the result of randomly introducing Brownian noise over a short period of time into an otherwise non-stationary fractal signal. In this sense, they have nothing to do with Lévy flights as such, but produce results that may be considered to be analogous to them. We call these effects 'Brownian transients' which have longer time correlations than fractal noise.

Suppose we consider an inhomogeneous fractal diffusion equation of the form

$$\left[\frac{\partial^2}{\partial x^2} - \tau\frac{\partial^q}{\partial t^q}\right] u(x,t) = F(x,t), \quad 0 < q \leq 1$$

where τ is a constant, F is a stochastic source term with some PDF and u is the stochastic field whose solution we require. When $q = 1$ we obtain the diffusion equation but in general, a solution to this equation will provide stationary temporal fractal walks - random walks of fractal time. One way of introducing a (first order) non-stationary process is to consider a source term of the form $F[x,t,\alpha(x),\beta(t)]$ where α and β are arbitrary functions. For example, suppose that we write F in separable form $F(x,t) = f(x)n(t)$ and that $n(t)$ is a random variable of time with a normal or

[3]First postulated by Dr S Mikhailov, a former research student of the author

Gaussian PDF given by

$$\Pr[n(t)] = \frac{1}{\sigma\sqrt{2\pi}} \exp[-(\mu - n)^2/2\sigma^2], \quad -\infty < n < \infty$$

where μ and σ are the mean and standard deviation respectively. By letting μ and/or σ be functions of t, we can introduce time variations in the mean and/or standard deviation respectively. In this case, varying the mean will cause the range of $n(t)$ to change with time and varying the standard deviation will change the variance of $n(t)$ with time. Note that in this case, the form of the distribution of the field remains the same, it is a time varying Gaussian field. A more general statement of a non-stationary stochastic process is one in which the distribution itself changes with time.

Another way of introducing non-stationarity is through q by letting it become a function of time t. Suppose that in addition to this, we extend the range of q to include the values 0 and 2 so that $0 \leq q \leq 2$. This idea immediately leads us to an interesting consequence because with q in this range, we can choose $q = 1$ to yield the (stochastic) diffusion equation but also choose $q = 2$ to obtain an entirely different equation, namely, the (stochastic) wave equation. Choosing (quite arbitrarily) q to be in this range, leads to control over the basic physical characteristics of the equation so that we can define a static mode when $q = 0$, a diffusive mode when $q = 1$ and a propagative mode when $q = 2$. In this case, non-stationarity is introduced through the use of a time varying fractional derivative whose values modify the physical 'essence' of the equation. Since the range of q has been chosen arbitrarily, we generalize further and consider the equation

$$\left[\frac{\partial^2}{\partial x^2} - \tau^{q(t)} \frac{\partial^{q(t)}}{\partial t^{q(t)}}\right] u(x,t) = F(x,t), \quad -\infty < q(t) < \infty, \quad \forall t.$$

Now, when $q = 0 \ \forall t$, the time dependent behaviour is determined by the source function alone; when $q = 1 \ \forall t$, u describes a diffusive process where τ is the 'diffusivity' (the inverse of the coefficient of diffusion); when $q = 2$ we have a propagative process where τ is the 'slowness' (the inverse of the wave speed). The latter process should be expected to 'propagate information' more rapidly than a diffusive process leading to transients or 'flights' of some type. We refer to q as the Fourier dimension which, for a fractal signal, is related to the conventional definition of the fractal (i.e. the 'Similarity', 'Minkowksi' or 'Box Counting') dimension D by

$$q = \frac{5 - 2D}{2} \quad 1 < D < 2.$$

How Should we Choose $q(t)$?

Since $q(t)$ 'drives' the non-stationary behaviour of u, the way in which we model $q(t)$ is crucial. It is arguable that the changes in the statistical characteristics of u which lead to its non-stationary behaviour should in themselves be random. Thus, suppose that we let the Fourier dimension at a time t be chosen randomly, a randomness that is determined by some PDF. In this case, the non-stationary characteristics of u will be determined by the PDF (and associated parameters) alone. Also, since q is

a dimension, we can consider our model to be based on the 'statistics of dimension'. There are a variety of PDFs that can be applied which will in turn effect the range of q. By varying the exact nature of the distribution considered, we can 'drive' the non-stationary behaviour of u in different ways. For example, suppose we consider a system which is assumed to be primarily diffusive; then a 'normal' PDF of the type

$$\Pr[q(t)] = \frac{1}{\sigma\sqrt{2\pi}} \exp[-(q-1)^2/2\sigma^2], \quad -\infty < q < \infty$$

will ensure that u is entirely diffusive when $\sigma \to 0$. However, as σ is increased in value, the likelihood of $q = 2$ (and $q = 0$) becomes larger. In other words, the standard deviation provides control over the likelihood of the process becoming propagative. If for example, we consider a Gamma distribution given by

$$\Pr[q(t)] = \frac{1}{\beta^\alpha} \frac{1}{\Gamma(\alpha)} q^{\alpha-1} \exp(-q/\beta), \quad q \geq 0$$

where $\alpha > 0$ and $\beta > 0$, then q lies in the positive half space alone with mean and variance given by

$$\mu = \alpha\beta \quad \text{and} \quad \sigma^2 = \alpha\beta^2$$

respectively. PDFs could also be considered which are of compact support such as the Beta distribution given by

$$\Pr[q(t)] = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} q^{\alpha-1}(1-q)^{\beta-1}, \quad 0 < q < 1$$

where α and β are positive constants. Here, the mean and variance are

$$\mu = \frac{\alpha}{\alpha+\beta} \quad \text{and} \quad \sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

respectively and for $\alpha > 1$ and $\beta > 1$ there is a unique mode given by

$$\text{mode} \equiv \Pr[q(t)]_{\max} = \frac{\alpha-1}{\alpha+\beta+2}.$$

Irrespective of the type of distribution that is considered, the equation

$$\left[\frac{\partial^2}{\partial x^2} - \tau^{q(t)} \frac{\partial^{q(t)}}{\partial t^{q(t)}}\right] u(x,t) = F(x,t), \quad -\infty < q(t) < \infty, \quad \forall t$$

poses a fundamental problem which is how to define and work with the term

$$\frac{\partial^{q(t)}}{\partial t^{q(t)}} u(x,t).$$

Given the result (for constant q)

$$\frac{\partial^q}{\partial t^q} u(x,t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} (i\omega)^q U(x,\omega) \exp(i\omega t) d\omega, \quad -\infty < q < \infty$$

we might generalize as follows:

$$\frac{\partial^{q(t)}}{\partial t^{q(t)}}u(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}(i\omega)^{q(t)}U(x,\omega)\exp(i\omega t)d\omega.$$

However, if we consider the case where the Fourier dimension is a relatively slowly varying function of t, then we can legitimately consider $q(t)$ to be composed of a sequence of different states $q_i = q(t_i)$. This approach allows us to develop a stationary solution for a fixed q over a fixed period of time. Non-stationary behaviour can then be introduced by using the same solution for different values or 'quanta' q_i over fixed (or varying) periods of time and concatenating the solutions for all q_i.

17.5 Green's Function Solution

We consider a Green's function solution to the equation

$$\left[\frac{\partial^2}{\partial x^2} - \tau^{q(t)}\frac{\partial^{q(t)}}{\partial t^{q(t)}}\right]u(x,t) = F(x,t), \quad -\infty < q(t) < \infty, \quad \forall t$$

for constant q when $F(x,t) = f(x)n(t)$ where $f(x)$ and $n(t)$ are both stochastic functions. Applying a separation of variables here is not strictly necessary. However, it yields a solution in which the terms affecting the temporal behaviour of $u(x,t)$ are clearly identifiable. Thus, we require a general solution to the equation

$$\left(\frac{\partial^2}{\partial x^2} - \tau^q\frac{\partial^q}{\partial t^q}\right)u(x,t) = f(x)n(t).$$

Let

$$u(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}U(x,\omega)\exp(i\omega t)d\omega, \quad \text{and} \quad n(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}N(\omega)\exp(i\omega t)d\omega.$$

Then, using the result

$$\frac{\partial^q}{\partial t^q}u(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}U(x,\omega)(i\omega)^q\exp(i\omega t)d\omega$$

this fractional PDE transforms to

$$\left(\frac{\partial^2}{\partial x^2} + \Omega_q^2\right)U(x,\omega) = f(x)N(\omega)$$

where we shall take

$$\Omega_q = i(i\omega\tau)^{\frac{q}{2}}$$

and ignore the case for $\Omega_q = -i(i\omega\tau)^{\frac{q}{2}}$. Defining the Green's function g to be the solution of

$$\left(\frac{\partial^2}{\partial x^2} + \Omega_q^2\right) g(x \mid x_0, \omega) = \delta(x - x_0)$$

where δ is the delta function, we obtain the following solution:

$$U(x_0, \omega) = N(\omega) \int_{-\infty}^{\infty} g(x \mid x_0, k) f(x) dx$$

where

$$g(X, \omega) = -\frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{\exp(iuX)}{(u + \Omega_q)(u - \Omega_q)} du, \quad X = \mid x - x_0 \mid .$$

The contour integral

$$\oint_C \frac{\exp(izX)}{(z + \Omega_q)(z - \Omega_q)} dz$$

has complex poles at $z = \pm\Omega_q$ which are q dependent (varying from $\pm i$ when $q = 0$, through to $\pm i(i\omega\tau)^{1/2}$ when $q = 1$ and on to $\mp\omega\tau$ when $q = 2$ for example). For any value of q, we can compute this contour integral using the residue theorem. By choosing the z plane for $-\infty < x < \infty$ and $i0 \leq iy < i\infty$ where $z = x + iy$ we obtain (through application of a semi-circular contour C and Cauchy's residue theorem)

$$g(x \mid x_0, k) = \frac{i}{2\Omega_q} \exp(i\Omega_q \mid x - x_0 \mid)$$

under the assumption that Ω_q is finite. This result reduces to conventional solutions for cases when $q = 1$ (diffusion equation) and $q = 2$ (wave equation) as shall now be shown.

Wave Equation Solution

When $q = 2$, the Green's function defined above provides a solution for the outgoing Green's function. Thus, with $\Omega_2 = -\omega\tau$, we have

$$U(x_0, \omega) = \frac{N(\omega)}{2i\omega\tau} \int_{-\infty}^{\infty} \exp(-i\omega\tau \mid x - x_0 \mid) f(x) dx$$

and Fourier inverting we get

$$\begin{aligned} u(x_0, t) &= \frac{1}{2\tau} \int_{-\infty}^{\infty} dx f(x) \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{N(\omega)}{i\omega} \exp(-i\omega\tau \mid x - x_0 \mid) \exp(i\omega t) d\omega \\ &= \frac{1}{2\tau} \int_{-\infty}^{\infty} dx f(x) \int_{-\infty}^{t} n(t - \tau \mid x - x_0 \mid) dt \end{aligned}$$

which describes the propagation of a wave travelling at velocity $1/\tau$ subject to variations in space and time as defined by $f(x)$ and $n(t)$ respectively. For example, when f and n are both delta functions,

$$u(x_0,t) = \frac{1}{2\tau} H(t - \tau \mid x - x_0 \mid)$$

where H is the Heaviside step function defined by

$$H(y) = \begin{cases} 1, & y > 0; \\ 0, & y < 0. \end{cases}$$

This is a d'Alembertian type solution to the wave equation where the wavefront occurs at $t = \tau \mid x - x_0 \mid$ in the causal case.

Diffusion Equation Solution

When $q = 1$ and $\Omega_1 = i\sqrt{i\omega\tau}$,

$$u(x_0,t) = \frac{1}{2} \int_{-\infty}^{\infty} dx f(x) \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{\exp(-\sqrt{i\omega\tau} \mid x - x_0 \mid)}{\sqrt{i\omega\tau}} N(\omega) \exp(i\omega t) d\omega.$$

For $p = i\omega$, we can write this result in terms of a Bromwich integral (i.e. an inverse Laplace transform) and using the convolution theorem for Laplace transforms with the result

$$\int_{c-i\infty}^{c+i\infty} \frac{\exp(-a\sqrt{p})}{\sqrt{p}} \exp(pt) dp = \frac{1}{\sqrt{\pi t}} \exp[-a^2/(4t)],$$

we obtain

$$u(x_0,t) = \frac{1}{2\sqrt{\tau}} \int_{-\infty}^{\infty} dx f(x) \int_0^t \frac{\exp[-\tau(x_0 - x)^2/(4t_0)]}{\sqrt{\pi t_0}} n(t - t_0) dt_0.$$

Now, if for example, we consider the case when n is a delta function, the result reduces to

$$u(x_0,t) = \frac{1}{2\sqrt{\pi\tau t}} \int_{-\infty}^{\infty} f(x) H(t) \exp[-\tau(x_0 - x)^2/(4t)] dx, \quad t \to \infty$$

which describes classical diffusion in terms of the convolution of an initial source $f(x)$ (introduced at time $t = 0$) with a Gaussian function.

General Series Solution

The evaluation of $u(x_0,t)$ via direct Fourier inversion for arbitrary values of q is not possible due to the irrational nature of the exponential function $\exp(i\Omega_q \mid x - x_0 \mid)$ with respect to ω. To obtain a general solution, we use the series representation of the exponential function and write

$$U(x_0,\omega) = \frac{iM_0N(\omega)}{2\Omega_q}\left[1 + \sum_{m=1}^{\infty}\frac{(i\Omega_q)^m}{m!}\frac{M_m(x_0)}{M_0}\right]$$

where

$$M_m(x_0) = \int_{-\infty}^{\infty} f(x) \mid x - x_0 \mid^m dx.$$

We can now Fourier invert term by term to develop a series solution. This requires us to consider three distinct cases.

Case 1: $q = 0$

Evaluation of $u(x_0,t)$ in this case is trivial since

$$U(x_0,\omega) = \frac{M(x_0)}{2}N(\omega) \quad \text{or} \quad u(x_0,t) = \frac{M(x_0)}{2}n(t)$$

where

$$M(x_0) = \int_{-\infty}^{\infty} \exp(- \mid x - x_0 \mid)f(x)dx.$$

Case 2: $q > 0$

Fourier inverting, the first term in this series becomes

$$\begin{aligned}\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{iN(\omega)M_0}{2\Omega_q}\exp(i\omega t)d\omega &= \frac{M_0}{2\tau^{\frac{q}{2}}}\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{N(\omega)}{(i\omega)^{\frac{q}{2}}}\exp(i\omega t)d\omega \\ &= \frac{M_0}{2\tau^{\frac{q}{2}}}\frac{1}{\Gamma(q/2)}\int_0^t\frac{n(\xi)}{(t-\xi)^{1-(q/2)}}d\xi, \quad \mathrm{Re}[q] > 0.\end{aligned}$$

The second term is

$$-\frac{M_1}{2}\frac{1}{2\pi}\int_{-\infty}^{\infty}N(\omega)\exp(i\omega t)d\omega = -\frac{M_1}{2}n(t).$$

The third term is

$$-\frac{iM_2}{2.2!}\frac{1}{2\pi}\int_{-\infty}^{\infty}N(\omega)i(i\omega\tau)^{\frac{q}{2}}\exp(i\omega t)d\omega = \frac{M_2\tau^{\frac{q}{2}}}{2.2!}\frac{d^{\frac{q}{2}}}{dt^{\frac{q}{2}}}n(t)$$

and the fourth and fifth terms become

$$\frac{M_3}{2.3!}\frac{1}{2\pi}\int_{-\infty}^{\infty} N(\omega)i^2(i\omega\tau)^q \exp(i\omega t)d\omega = -\frac{M_3\tau^q}{2.3!}\frac{d^q}{dt^q}n(t)$$

and

$$i\frac{M_4}{2.4!}\frac{1}{2\pi}\int_{-\infty}^{\infty} N(\omega)i^3(i\omega\tau)^{\frac{3q}{2}} \exp(i\omega t)d\omega = \frac{M_4\tau^{\frac{3q}{2}}}{2.4!}\frac{d^{\frac{3q}{2}}}{dt^{\frac{3q}{2}}}n(t)$$

respectively with similar results for all other terms. Thus, through induction, we can write $u(x_0,t)$ as a series of the form

$$\begin{aligned} u(x_0,t) &= \frac{M_0(x_0)}{2\tau^{q/2}}\frac{1}{\Gamma(q/2)}\int_0^t \frac{n(\xi)}{(t-\xi)^{1-(q/2)}}d\xi - \frac{M_1(x_0)}{2}n(t) + \\ &\quad \frac{1}{2}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{(k+1)!}M_{k+1}(x_0)\tau^{kq/2}\frac{d^{kq/2}}{dt^{kq/2}}n(t). \end{aligned}$$

Observe that the first term involves a fractional integral, the second term is composed of the source function $n(t)$ alone (apart from scaling) and the third term is an infinite series composed of fractional differentials of increasing order $kq/2$. Also note that the first term is scaled by a factor involving $\tau^{-q/2}$ whereas the third term is scaled by a factor that includes $\tau^{kq/2}$.

Case 3: $q < 0$

In this case, the first term becomes

$$\begin{aligned} \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{iN(\omega)M_0}{2\Omega_q}\exp(i\omega t)d\omega &= \frac{M_0}{2}\tau^{\frac{q}{2}}\frac{1}{2\pi}\int_{-\infty}^{\infty} N(\omega)(i\omega)^{\frac{q}{2}}\exp(i\omega t)d\omega \\ &= \frac{M_0}{2}\tau^{\frac{q}{2}}\frac{d^{\frac{q}{2}}}{dt^{\frac{q}{2}}}n(t). \end{aligned}$$

The second term is the same is in the previous case (for $q > 0$) and the third term is

$$-\frac{iM_2}{2.2!}\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{N(\omega)i}{(i\omega\tau)^{\frac{q}{2}}}\exp(i\omega t)d\omega = \frac{M_2}{2.2!}\frac{1}{\tau^{q/2}}\frac{1}{\Gamma(q/2)}\int_0^t\frac{n(\xi)}{(t-\xi)^{1-(q/2)}}d\xi.$$

Evaluating the other terms, by induction we obtain

$$\begin{aligned} u(x_0,t) &= \frac{M_0(x_0)\tau^{q/2}}{2}\frac{d^{q/2}}{dt^{q/2}}n(t) - \frac{M_1(x_0)}{2}n(t) + \\ &\quad \frac{1}{2}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{(k+1)!}\frac{M_{k+1}(x_0)}{\tau^{kq/2}}\frac{1}{\Gamma(kq/2)}\int_0^t\frac{n(\xi)}{(t-\xi)^{1-(kq/2)}}d\xi \end{aligned}$$

where $q \equiv \mid q \mid$, $q < 0$. Here, the solution is composed of three terms: a fractional differential, the source term and an infinite series of fractional integrals of order $kq/2$. Thus, the roles of fractional differentiation and fractional integration are reversed as q changes from being greater than to less than zero. N.B. all fractional differential operators associated with the equations above and hence forth should be considered in terms of the definition for a fractional differential given by

$$\hat{D}^q f(t) = \frac{d^n}{dt^n}[\hat{I}^{n-q} f(t)], \quad n - q > 0.$$

Asymptotic Forms for $f(x) = \delta(x)$

We consider a special case in which the source function $f(x)$ is an impulse so that

$$M_m(x_0) = \int_{-\infty}^{\infty} \delta(x) \mid x - x_0 \mid^m dx = \mid x_0 \mid^m .$$

This result immediately suggests a study of the asymptotic solution

$$u(t) = \lim_{x_0 \to 0} u(x_0, t) = \begin{cases} \frac{1}{2\tau^{q/2}} \frac{1}{\Gamma(q/2)} \int_0^t \frac{n(\xi)}{(t-\xi)^{1-(q/2)}} d\xi, & q > 0; \\ \frac{n(t)}{2}, & q = 0; \\ \frac{\tau^{q/2}}{2} \frac{d^{q/2}}{dt^{q/2}} n(t), & q < 0. \end{cases}$$

The solution for the time variations of the stochastic field u for $q > 0$ are then given by a fractional integral alone and for $q < 0$ by a fractional differential alone. In particular, for $q > 0$, we see that the solution is based on a causal convolution. Thus in t-space

$$u(t) = \frac{1}{2\tau^{q/2}\Gamma(q/2)} \frac{1}{t^{1-q/2}} \otimes n(t), \quad q > 0$$

where $\otimes$ denotes (causal) convolution and in ω-space

$$U(\omega) = \frac{N(\omega)}{2\tau^{q/2}(i\omega)^{q/2}}.$$

This result is the conventional fractal noise model. The table below quantifies the results for different values of q with conventional name associations. Note that u has the following fundamental property (for $q > 0$):

$$\lambda^{q/2} \Pr[u(t)] = \Pr[u(\lambda t)].$$

This property describes the statistical self-affinity of u. Thus, the asymptotic solution considered here, yields a result that describes a RSF signal characterized by a PSDF of the form $1/ \mid \omega \mid^q$ which is a measure of the time correlations in the signal.

q-value	t-space	ω-space (PSDF)	Name
$q=0$	$\frac{1}{2}n(t)$	$\frac{1}{4}$	White noise
$q=1$	$\frac{1}{2\sqrt{\tau}\Gamma(1/2)}\frac{1}{\sqrt{t}}\otimes n(t)$	$\frac{1}{4\tau\lvert\omega\rvert}$	Pink noise
$q=2$	$\frac{1}{2\tau\Gamma(1)}\int_0^t n(t)dt$	$\frac{1}{4\tau^2\omega^2}$	Brown noise
$q>2$	$\frac{1}{2\tau^{q/2}\Gamma(q/2)}t^{(q/2)-1}\otimes n(t)$	$\frac{1}{4\tau^q\lvert\omega\rvert^q}$	Black noise

Table 17.1: Noise characteristics for different values of q. (Note that $\Gamma(1/2)=\sqrt{\pi}$ and $\Gamma(1)=1$.)

Other Asymptotic Forms

Another interesting asymptotic form is

$$u(x_0,t)=\frac{M_0(x_0)}{2\tau^{q/2}}\frac{1}{\Gamma(q/2)}\int_0^t\frac{n(\xi)}{(t-\xi)^{1-(q/2)}}d\xi-\frac{M_1(x_0)}{2}n(t),\quad\tau\to 0.$$

Here, the solution is the sum of fractal noise and white noise. By relaxing the condition $\tau\to 0$ we can consider the approximation

$$\begin{aligned}u(x_0,t)\quad\simeq\quad&\frac{M_0(x_0)}{2\tau^{q/2}}\frac{1}{\Gamma(q/2)}\int_0^t\frac{n(\xi)}{(t-\xi)^{1-(q/2)}}d\xi-\\&\frac{M_1(x_0)}{2}n(t)+\frac{M_2(x_0)}{2.2!}\tau^{q/2}\frac{d^{q/2}}{dt^{q/2}}n(t),\quad\tau<<1\end{aligned}$$

in which the solution is expressed in terms of the sum of fractal noise, white noise and the fractional differentiation of white noise.

17.6 Digital Algorithms

There are two principal algorithms that are required to investigate the results given in the previous section using a digital computer. The first of these concerns the computation of discrete fractal noise u_j given q which is as follows:

(i) Compute a pseudo random zero mean (Gaussian) distributed array $n_j,\ \ j=0,1,...,N-1$.

(ii) Compute the Discrete Fourier Transform (DFT) of n_j giving N_j using a Fast Fourier Transform (FFT).

(iii) Filter N_j with $1/(i\omega_j)^{q/2}$.

(iv) Inverse DFT the result using a FFT to give u_j (real part).

The second algorithm is an inversion algorithm. Given the digital algorithm described above, the inverse problem can be defined as given u_j compute q. A suitable approach to solving this problem, which is consistent with the algorithm given above is to estimate q from the power spectrum of u_j whose expected form (considering the positive half space only and excluding the DC component which is singular) is

$$\hat{P}_j = \frac{A}{\omega_j^q}; \quad j = 1, 2, ..., (N/2) - 1$$

where A is a constant. Here, we assume that the FFT provides data in 'standard form' and that the DC or zero frequency component occurs at $j = 0$. If we now consider the error function

$$e(A, q) = \| \ln P_j - \ln \hat{P}_j \|_2^2$$

where P_j is the power spectrum of u_j, then the solution of the equations (least squares method)

$$\frac{\partial e}{\partial q} = 0; \quad \frac{\partial e}{\partial A} = 0$$

gives

$$q = \frac{\left(\frac{N}{2} - 1\right) \sum_{j=1}^{(N/2)-1} [(\ln P_j)(\ln \omega_j)] - \left(\sum_{j=1}^{(N/2)-1} \ln \omega_j \right) \left(\sum_{j=1}^{(N/2)-1} \ln P_j \right)}{\left(\sum_{j=1}^{(N/2)-1} \ln \omega_j \right)^2 - \left(\frac{N}{2} - 1\right) \sum_{j=1}^{(N/2)-1} (\ln \omega_j)^2}$$

and

$$A = \exp \left(\frac{\sum_{j=1}^{(N/2)-1} \ln P_j + q \sum_{j=1}^{(N/2)-1} \ln \omega_j}{\frac{N}{2} - 1} \right).$$

The algorithm required to implement this inverse solution is as follows:

(i) Compute the power spectrum P_j of the fractal noise u_j using an FFT.

(ii) Extract the positive half space data (excluding the DC term).

(iii) Compute q

This algorithm (which is commonly known as the Power Spectrum Method) provides a reconstruction for q which is (on average) accurate to 2 decimal places for $N \geq 64$.

17.7 Non-stationary Algorithm

The results considered so far have been based on the assumption that the Fourier dimension is constant. We have considered the case of $q(t)$ having discrete states

$q_i = q(t_i)$ assuming that $q(t)$ is a slowly varying function. In this case, the solutions and algorithms discussed so far are valid for any $q_i, \quad i = 0, 1, 2, ..., M-1$ over a window of time Δt_i say, which is taken to be composed of $N-1$ elements. A non-stationary signal can therefore be generated by computing $M-1$ signals each of size $N-1$ and concatenating the results to form a contiguous stream of data of size $(M-1) \times (N-1)$ to give u_j. A further generalization can be made by choosing Δt_i randomly. An interesting approach to this is based on letting Δt_i be a fractal signal so that the stochastic behaviour of u_j is governed by fractal time.

Following the ideas discussed in the section on 'how should we choose $q(t)$?', q_i can be taken to be a discrete random variable which is chosen to conform to a discrete PDF or histogram. For example, we can consider the 'normal' distribution given by

$$\Pr[q(t)] = \frac{1}{\sigma\sqrt{2\pi}} \exp[-(q-1)^2/2\sigma^2], \quad -\infty < q < \infty \quad \forall \quad t.$$

Here, the distribution is symmetric about $q = 1$; pink noise characterising a diffusive process.

Continuity Condition

The short time integration of white noise that occurs when $q = 2$ can lead to a significant change in the range of values of u_j. This type of behaviour is called a Brownian transient. A time series generated for different values of q may be composed of Brownian transients that 'look like' spikes. These spikes may be of arbitrary polarity given that $n(t)$ is zero-mean white Gaussian noise. An example of this is given in Figure 17.2 where a fractal signal has been simulated using two values of q, namely, 1 (for points 1 to 500 and 516 to 1000) and 2 (for points 501 to 515). The complex plane map obtained through application of the Hilbert transform is also given for comparison illustrating the concentration of the stochastic field over one center of this plane. Suppose we wish to simulate a non-stationary signal whose mean value changes from time to time. One way of doing this is to let the mean value vary according to the amplitude obtained at the end of a transient (where q changes from one value to the next). This provides continuity in the mean value from one transient to another. An example of this is given in Figure 17.3 for the same stochastic field illustrated in Figure 17.2. The effect of the continuity condition is to simulate a multi-fractal signal with a much broader distribution where Brownian transients in particular, change the mean value over a short period of time. This is compounded in the complex plane representation of the signal which now shows two centers of activity with a transitory path (Brownian transient) between them.

The effect of introducing this is further illustrated in Figures 17.4 to 17.8. Here q is chosen from a Gaussian distribution with a mean of 1 and the standard deviation increased from 0 to 4 in steps of 1 providing multi-random-fractal non-stationary signals with increasingly transient behaviour and consequently, broader distributions. The positions at which these transients occur together with their directions and amplitudes are entirely arbitrary and 'driven' by the Gaussian pseudo random number generators used to compute n_j and q_i which in turn depends on the seeds used to initiate these generators. Moreover, the likelihood of generating a Brownian transient is determined by the standard deviation of q_i alone. This provides a method of

modelling non-stationary fractal signals as found in a wide variety of areas such as in economic time series, biomedical signal analysis etc.

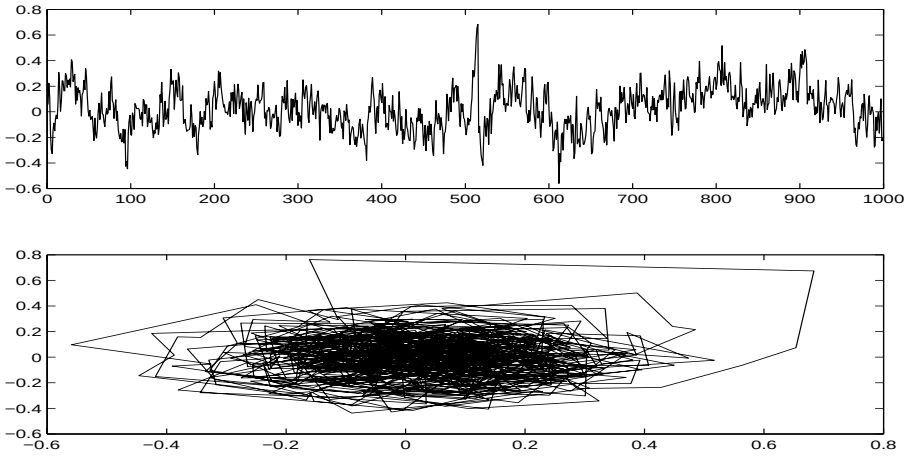


Figure 17.2: Non-stationary fractal signal (top) and complex plane map (bottom) for $q = 1$ and $q = 2$.

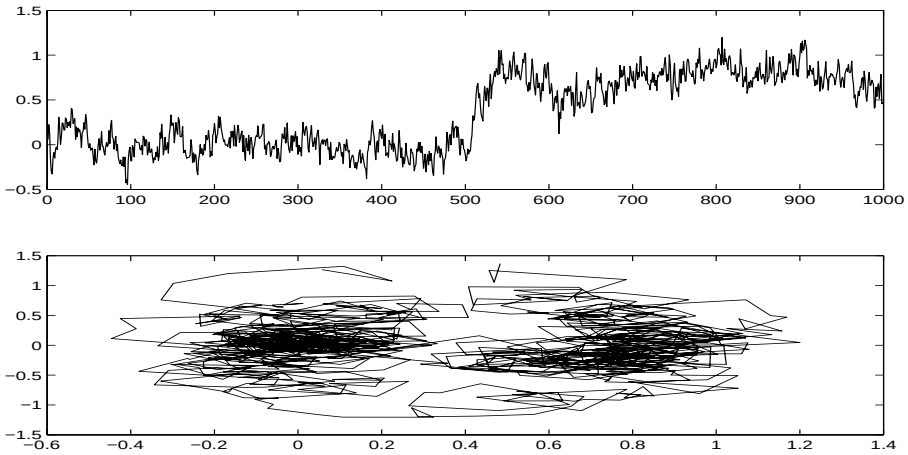


Figure 17.3: Non-stationary fractal signal (top) and complex plane map (bottom) for $q = 1$ and $q = 2$ with application of the continuity condition.

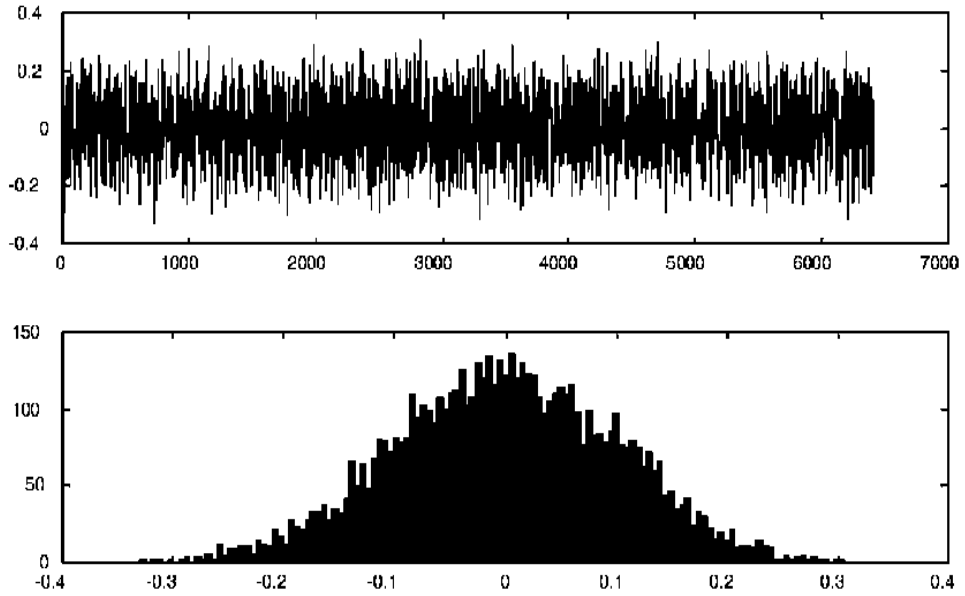


Figure 17.4: Mono-fractal signal (top) for $\sigma = 0$ (entirely diffusive with $q_i = 1, \quad \forall i$) and histogram (bottom).

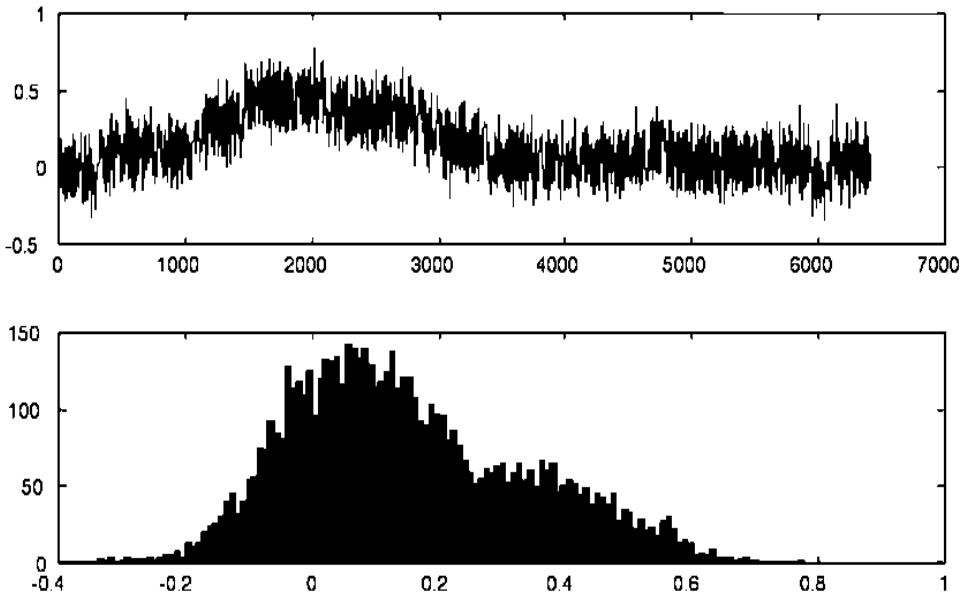


Figure 17.5: Multi-fractal signal (top) for $\sigma = 1$ and histogram (bottom).

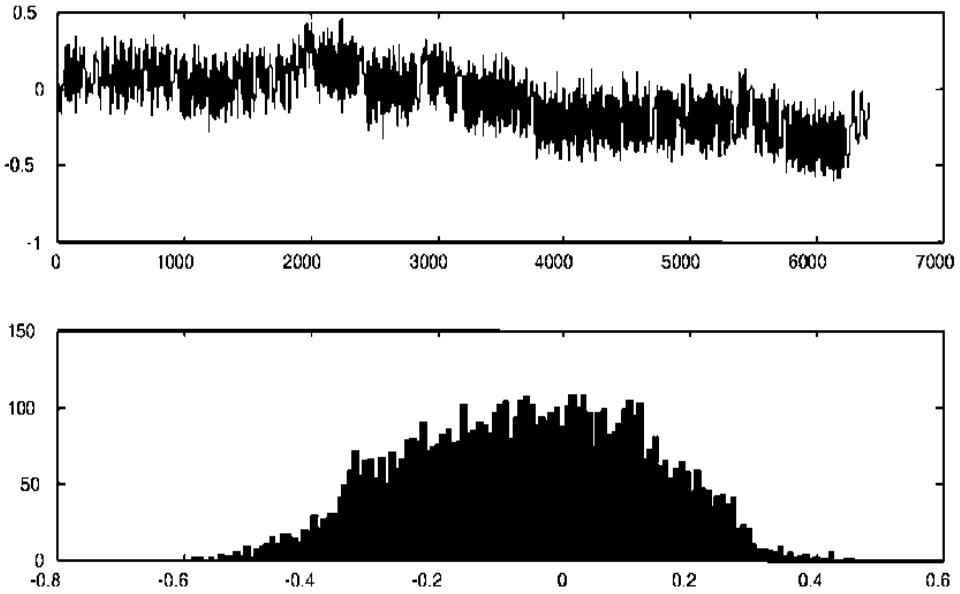


Figure 17.6: Multi-fractal signal (top) for $\sigma = 2$ and histogram (bottom).

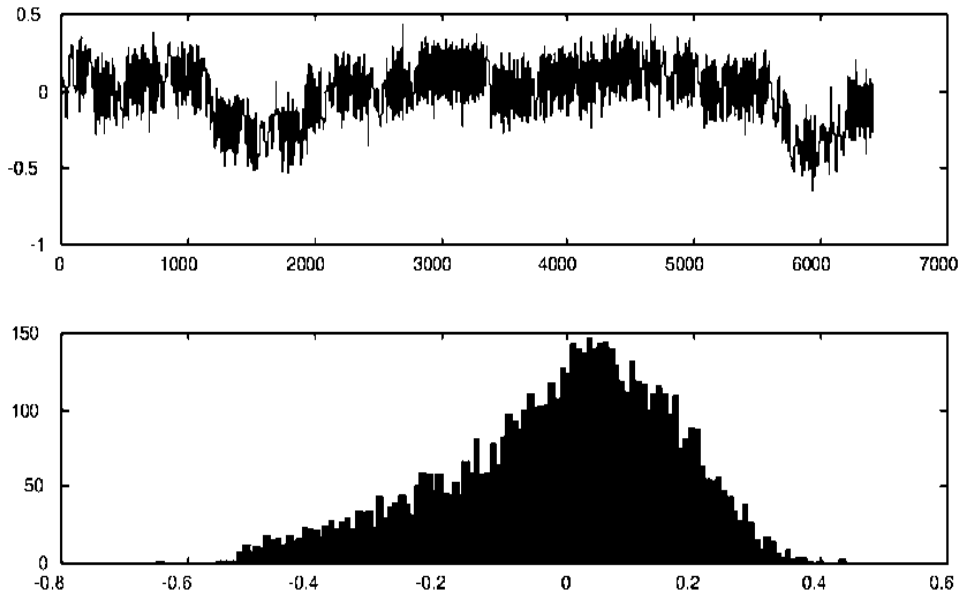


Figure 17.7: Multi-fractal signal (top) for $\sigma = 3$ and histogram (bottom).

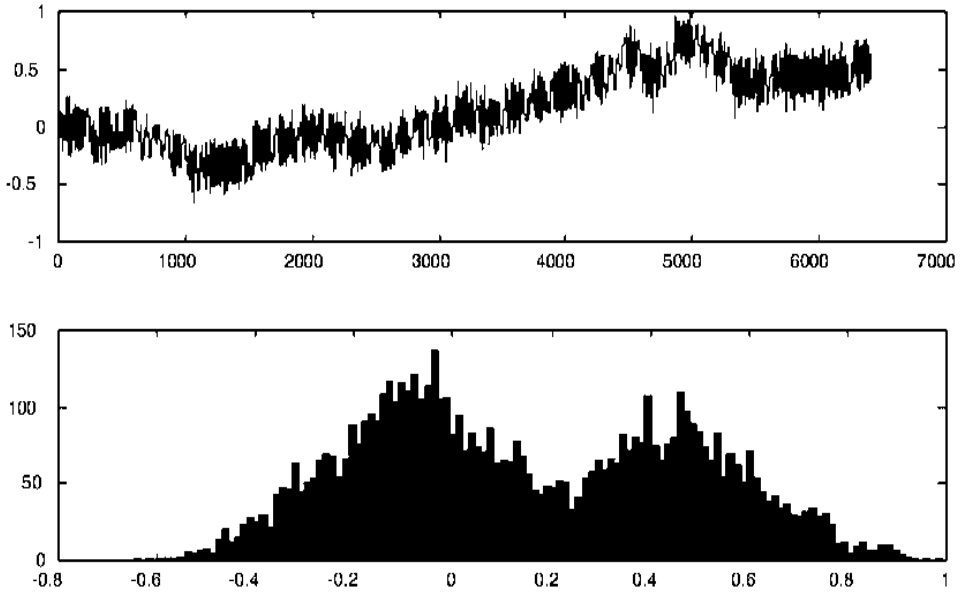


Figure 17.8: Multi-fractal signal (top) for $\sigma = 4$ and histogram (bottom).

Inverse Solution

The inverse solution is based on computing the variable Fourier dimension and then an appropriate time variant (or otherwise) statistic from it. For example, in the case of a time variant Gaussian distributed Fourier dimension, we compute the standard deviation using a moving window. In this case, for a given input signal s_i say, the Fourier dimension function q_i is computed using a moving window in which the power spectrum method is applied to the windowed data to produce a single value of the Fourier dimension for the window at position i. Having computed q_i, the moving window principle is again invoked in which the windowed data is used to compute the standard deviation of the Fourier dimension under the assumption that the distribution of these values is Gaussian. In each case, the data are used to construct a histogram of q_i and a least squares fit to a Gaussian function obtained from which the standard deviation is then computed. This produces the variable standard deviation function σ_i which provides a measure of the non-stationary nature of the original signal in terms of its propensity for propagative behaviour (i.e. Brownian transients).

Applications

There are a wide variety of applications. For example, in medicine, there are a number of biomedical signals that can and should be analysed using multi-Fourier dimensional

analysis. An interesting example concerns the behaviour of the heart rate prior to the on set of a heart attack. It is well known that the heart rate (measured in terms of the number of beats per minute f_i where i counts the minutes) is not constant but varies from one minute to the next. In other words, the heart rate signal f_i does not have a constant amplitude and if it is generated over a long enough period of time, starts to exhibit multi-fractal characteristics.[4] For a healthy condition, the computation of q_i from f_i yields a signal with a narrow (Gaussian) distribution and a well defined mean value and variance (of the type given in Figure 17.4). However, the behaviour of q_i becomes significantly more erratic (leading to a broadening of a typically non-Gaussian distribution similar to the type given in Figure 17.8 for example) prior to the onset of a heart attack. More importantly, this characteristic is observable up to 24 hours before the heart attack starts!

Another example includes the use of multi-fractal statistics for developing a financial volatility index. It tuns out, that although the distribution of financial time series are not Gaussian, the Fourier dimensions of such time series are Gaussian distributed. Moreover, these Gaussian statistics are not time invariant, i.e. the standard deviation is a variable. Now, if the standard deviation increases then, based on the model considered here, there is an increased likelihood for either propagative behaviour (Brownian transients when $q = 2$) or white noise behaviour (when $q = 0$) indicative of the possibility of greater volatility of the time series. Thus, by monitoring the standard deviation of the Gaussian distributed Fourier dimension associated with a macro-economic time series, a measure can be introduced that attempts to predict market volatility. Finally, a method of covert communications can be consider based on the application of a process called fractal modulation which is discussed in further detail in the following case study.

17.8 General Stochastic Model

For a time invariant linear system, the power law $\mid \omega \mid^{-q}$ is consistent with statistically self-affine noise which is the basic model used so far. Many signals do have a high frequency decay for which the fractal model is appropriate but the overall spectrum may have characteristics for which a $\mid \omega \mid^{-q}$ power law is not appropriate. In such cases, it is often possible to apply a spectral partitioning algorithm which attempts to extract the most appropriate part of the spectrum for which this power law applies.

There are of course a range of PSDF models that can be considered. One process of interest is the Bermann process where

$$P(\omega) = \frac{A \mid \omega \mid^{2g}}{(\omega_0^2 + \omega^2)}$$

which is a generalization of the Ornstein-Uhlenbeck process where

$$P(\omega) = \frac{A\omega}{(\omega_0^2 + \omega^2)}$$

and stems from an attempt to overcome the difficulties of the non-differentiability associated with a Wiener process. Here, A is a constant, $g > 0$ and ω_0 is a constant

[4]Principal source: American Journal of Cardiology, from 1999 onwards.

frequency. Clearly, we can consider a further generalization and consider the PSDF

$$P(\omega) = \frac{A \mid \omega \mid^{2g}}{(\omega_0^2 + \omega^2)^q}.$$

This PSDF represents a more general fractal-type process and is less restrictive particularly with regard to the low frequency characteristics.

17.9 Case Study: Fractal Modulation

Embedding information in data whose properties and characteristics resemble those of the background noise of a transmission system is of particular interest in covert digital communications. In this case study[5], we explore a method of coding bit streams by modulating the fractal dimension of a fractal noise generator. Techniques for reconstructing the bit streams (i.e. solving the inverse problem) in the presence of additive noise (assumed to be introduced during the transmission phase) are then considered. This form of 'embedding information in noise' is of value in the transmission of information in situations when a communications link requires an extra level of security or when an increase in communications traffic needs to be hidden in a covert sense by coupling an increase in the background noise of a given area with appropriate disinformation (e.g. increased sun spot activity). Alternatively, the method can be considered to be just another layer of covert technology used for military communications in general. In principle, the method we develop here can be applied to any communications system and in the following section, a brief overview of existing techniques is discussed.

17.9.1 Secure Digital Communications

A digital communications systems is one that is based on transmitting and receiving bit streams. The basic processes involved are as follows: (i) a digital signal is obtained from sampling an analogue signal derived from some speech and/or video system; (ii) this signal (floating point stream) is converted into a binary signal consisting of 0s and 1s (bit stream); (iii) the bit stream is then modulated and transmitted; (iv) at reception, the transmitted signal is demodulated to recover the transmitted bit stream; (v) the (floating point) digital signal is reconstructed. Digital to analogue conversion may then be required depending on the type of technology being used.

In the case of sensitive information, an additional step is required between stages (ii) and (iii) above where the bit stream is coded according to some classified algorithm. Appropriate decoding is then introduced between stages (iv) and (v) with suitable pre-processing to reduce the effects of transmission noise for example which introduces bit errors. The bit stream coding algorithm is typically based on a pseudo random number generator or nonlinear maps in chaotic regions of their phase spaces (chaotic number generation). The modulation technique is typically either Frequency Modulation or Phase Modulation. Frequency modulation involves assigning a specific frequency to each 0 in the bit stream and another higher (or lower) frequency to each 1 in the stream. The difference between the two frequencies is minimized to provide

[5]Based on the research of Dr S Mikhailov, a former research student of the author

space for other channels within the available bandwidth. Phase modulation involves assigning a phase value $(0, \quad \pi/2, \quad \pi, \quad 3\pi/2)$ to one of four possible combinations that occur in a bit stream (i.e. 00, 11, 01 or 10).

Scrambling methods can be introduced before binarization. A conventional approach to this is to distort the digital signal by adding random numbers to the out-of-band components of its spectrum. The original signal is then recovered by lowpass filtering. This approach requires an enhanced bandwidth but is effective in the sense that the signal can be recovered from data with a relatively low signal-to-noise ratio. 'Spread spectrum' or 'frequency hopping' is used to spread the transmitted (e.g. frequency modulated) information over many different frequencies. Although spread spectrum communications use more bandwidth than necessary, by doing so, each communications system avoids interference with another because the transmissions are at such minimal power, with only spurts of data at any one frequency. The emitted signals are so weak that they are almost imperceptible above background noise. This feature results in an added benefit of spread spectrum which is that eaves-dropping on a transmission is very difficult and in general, only the intended receiver may ever known that a transmission is taking place, the frequency hopping sequence being known only to the intended party. Direct sequencing, in which the transmitted information is mixed with a coded signal, is based on transmitting each bit of data at several different frequencies simultaneously, with both the transmitter and receiver synchronized to the same coded sequence. More sophisticated spread spectrum techniques include hybrid ones that leverage the best features of frequency hopping and direct sequencing as well as other ways to code data. These methods are particularly resistant to jamming, noise and multipath anomalies, a frequency dependent effect in which the signal is reflected from objects in urban and/or rural environments and from different atmospheric layers, introducing delays in the transmission that can confuse any unauthorized reception of the transmission.

The purpose of Fractal Modulation is to try and make a bit stream 'look like' transmission noise (assumed to be fractal). The technique considered here focuses on the design of algorithms which encode a bit stream in terms of two fractal dimensions that can be combined to produce a fractal signal characteristic of transmission noise. Ultimately, fractal modulation can be considered to be an alternative to frequency modulation although requiring a significantly greater bandwidth for its operation. However, fractal modulation could relatively easily be used as an additional preprocessing security measure before transmission. The fractal modulated signal would then be binarized and the new bit stream fed into a conventional frequency modulated digital communications system albeit with a considerably reduced information throughput for a given bit rate. The problem is as follows: given an arbitrary binary code, convert it into a non-stationary fractal signal by modulating the fractal dimension in such a way that the original binary code can be recovered in the presence of additive noise with minimal bit errors.

In terms of the theory discussed earlier, we consider a model of the type

$$\left[\frac{\partial^2}{\partial x^2} - \tau^{q(t)}\frac{\partial^{q(t)}}{\partial t^{q(t)}}\right] u(x,t) = -\delta(x)n(t), \quad q > 0, \quad x \to 0$$

where $q(t)$ is assigned two states, namely q_1 and q_2 (which correspond to 0 and 1 in a bit stream respectively) for a fixed period of time. The forward problem (fractal

modulation) is then defined as: given $q(t)$ compute $u(t) \equiv u(0,t)$. The inverse problem (fractal demodulation) is defined as: given $u(t)$ compute $q(t)$.

17.9.2 Fractal Modulation and Demodulation

Instead of working in terms of the Fourier dimension q, we shall consider the fractal dimension given by

$$D = \frac{5-2q}{2}$$

where $D \in (1,2)$. The technique is outlined below:

(i) For a given bit stream, allocate $D_{\min}$ to bit=0 and $D_{\max}$ to bit=1.

(ii) Compute a fractal signal of length N for each bit in the stream.

(iii) Concatenate the results to produce a contiguous stream of fractal noise.

The total number of samples can be increased through N (the number of samples per fractal) and/or increasing the number of fractals per bit. This results in improved estimates of the fractal dimensions leading to a more accurate reconstruction. Fractal demodulation is achieved by computing the fractal dimensions via the power spectrum method using a conventional moving window to provide the fractal dimension signature $D_i, \quad i = 0, 1, 2, ...$ The bit stream is then obtained from the following algorithm:

If $D_i \leq \Delta$ then bit =0;

If $D_i > \Delta$ then bit =1;

where

$$\Delta = D_{\min} + \frac{1}{2}(D_{\max} - D_{\min}).$$

The principal criteria for the optimization of this modulation/demodulation technique is to minimize $(D_{\max} - D_{\min})$ subject to accurate reconstructions for D_i in the presence of (real) transmission noise with options on:

(i) fractal size - the number of samples used to compute a fractal signal;

(ii) fractals per bit - the number of fractal signals used to represent one bit;

(iii) $D_{\min}$ (the fractal dimension for bit=0) and $D_{\max}$ (the fractal dimension for bit=1);

(iv) addition of Gaussian noise before reconstruction for a given SNR.

Option (iv) is based on the result compounded in the asymptotic model where the signal is taken to be the sum of fractal noise and white Gaussian noise.

An example of a fractal modulated signal is given in Figure 17.9 in which the binary code 0....1....0.... has been considered in order to illustrate the basic principle.

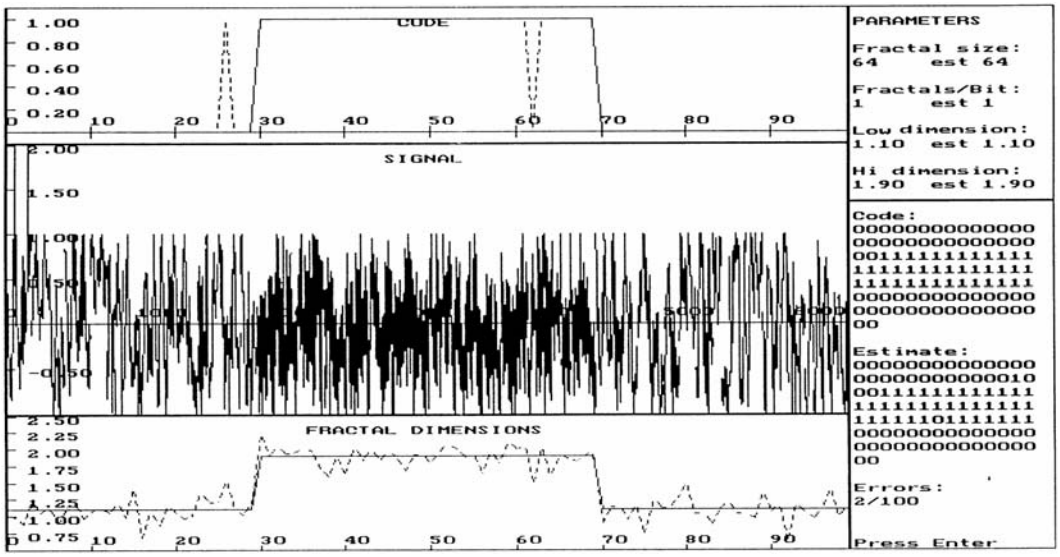


Figure 17.9: Fractal modulation of the code 0...1...0... for one fractal per bit.

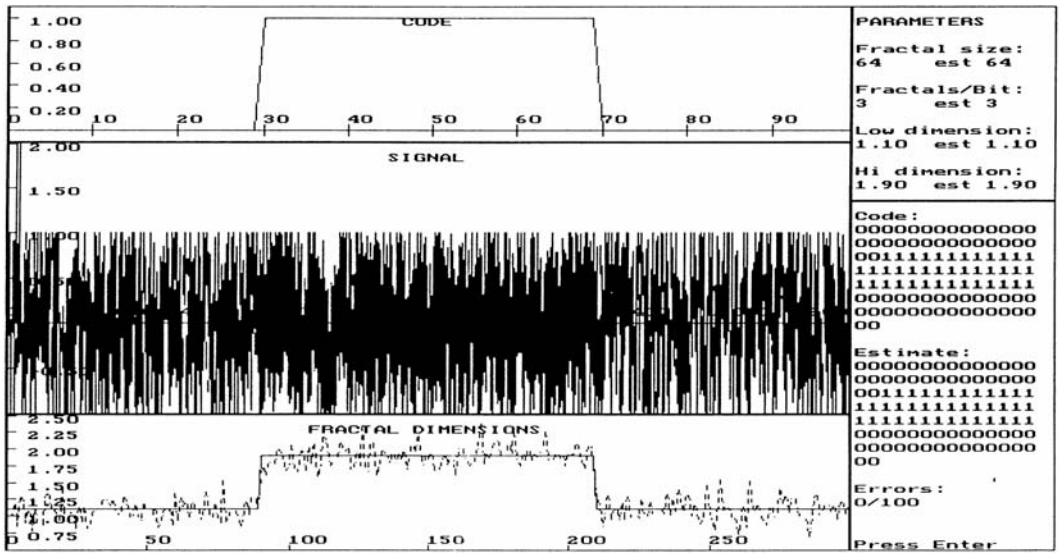


Figure 17.10: Fractal modulation of the code 0...1...0... for three fractals per bit.

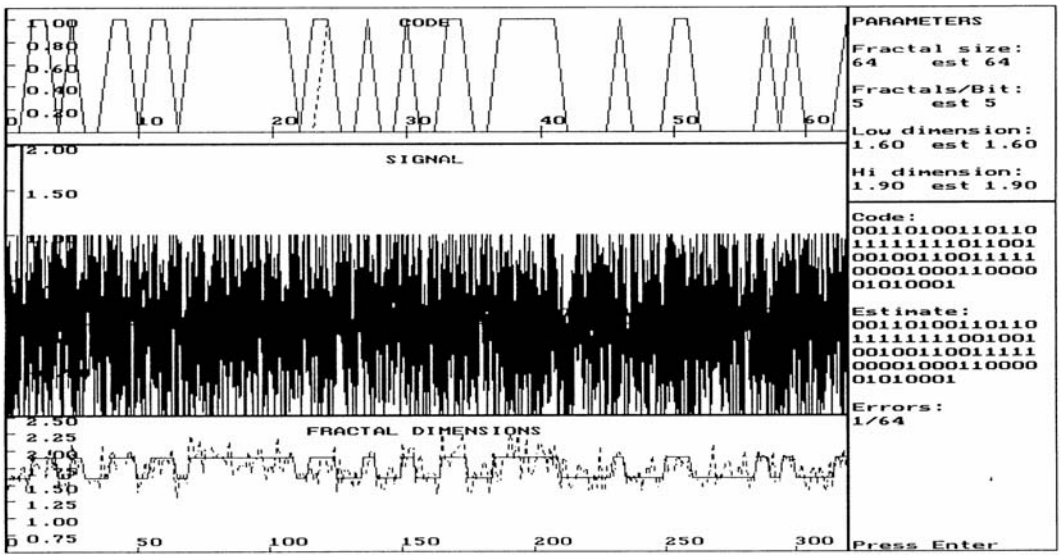


Figure 17.11: Fractal modulation of a random bit stream without additive noise.

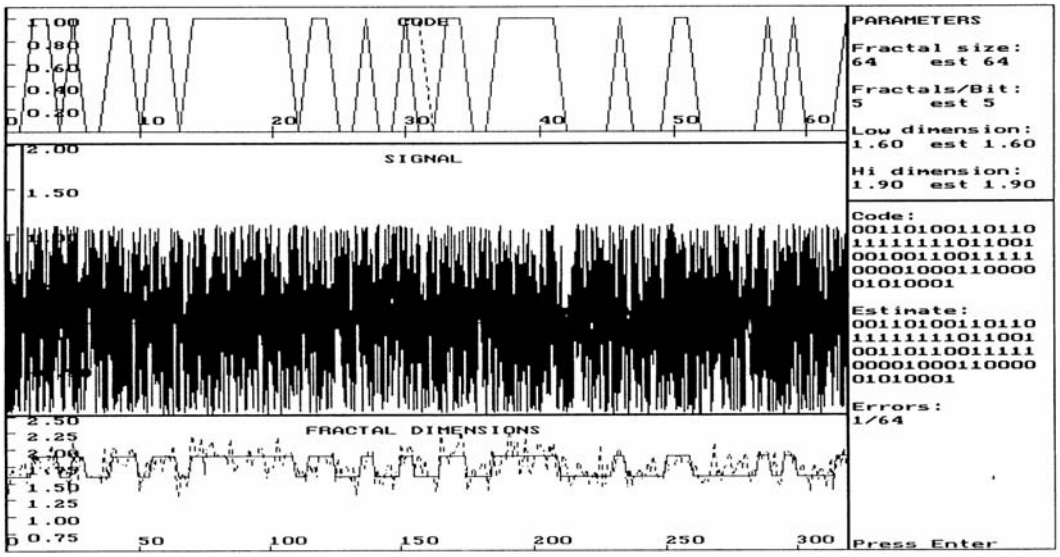


Figure 17.12: Fractal modulation of a random bit stream with 10% additive noise.

This figure shows the original binary code (top window) the (clipped) fractal signal (middle window) and the fractal dimension signature D_i (lower window - dotted line) using 1 fractal per bit, 64 samples per fractal for a 'Low dimension' ($D_{\min}$) and a 'Hi dimension' ($D_{\max}$) of 1.1 and 1.9 respectively. The reconstructed code is superimposed on the original code (top window - dotted line) and the original and estimated codes are displayed on the right hand side. In this example, there is a 2% bit error. By increasing the number of fractals per bit so that the bit stream is represented by an increased number of samples, greater accuracy can be achieved. This is shown in Figure 17.10 where for 3 fractals/bit there are no bit errors. In this example, each bit is represented by concatenating 3 fractal signals each with 64 samples. The reconstruction is based on a moving window of size 64. In Figures 17.9 and 17.10, the change in signal texture from 0 to 1 and from 1 to 0 is relatively clear because $(D_{\min}, D_{\max}) = (1.1, 1.9)$. By reducing the difference in fractal dimension, the textural changes across the signal can be reduced. This is shown in Figure 17.11 for $(D_{\min}, D_{\max}) = (1.6, 1.9)$ and a random 64 bit pattern where there is 1 bit in error. Figure 17.12 shows the same result but with 10% white Gaussian noise added to the fractal modulated signal before demodulation. Note that the bit errors have not been increased as a result of adding 10% noise.

Discussion

Fractal modulation is a technique which attempts to embed a bit stream in fractal noise by modulating the fractal dimension. The errors associated with recovering a bit stream are critically dependent on the SNR. The reconstruction algorithm provides relatively low bit error rates with a relatively high level of noise, provided the difference in fractal dimension is not too small and that many fractals per bit are used. In any application, the parameter settings would have to optimized with respect to a given transmission environment.

17.10 Case Study: Financial Signal Processing

In addition to the use of digital signal processing for a wide variety of applications ranging from medicine to mobile communications, DSP has started to play an increasing role in Computer Aided Financial Engineering[6]. In this extended final case study, we explore the background to financial signal processing. By discussing many of the conventional approaches to market analysis and the reasoning behind these approaches, we develop a Multi-fractal Market Hypothesis based on the application of the non-stationary fractional dynamical model discussed earlier in this chapter. After a short introduction, we investigate market analysis based on the 'Efficient Market Hypothesis' leading to Black-Scholes analysis. The 'Fractal Market Hypothesis' is then considered using a form of data analysis based on the work of Hurst. Finally, a multi-fractal market hypothesis' is introduced together with some example results.

[6]Grateful acknowledgement is due to Dr M London, a former research student of the author and whose research forms the basis for this case study.

17.11 Introduction

The application of statistical techniques for analysing time series and the financial markets in general is a well established practice. These practices include time series analysis and prediction using a wide range of stochastic modelling methods and the use of certain partial differential equations for describing financial systems (e.g. the Black-Scholes equation for financial derivatives). A seemingly inexhaustible range of statistical parameters are being developed to provide an increasingly complex portfolio of financial measures. The financial markets are a rich source of complex data. Changing every second or less, their behaviour is difficult to predict often leaving economists and politicians alike, baffled at events such as the crash of 15th October 1987 which lead to the recession of the late 1980s and early 1990s. This event is shown in Figure 17.13 which is the time series for the New York Average from 1980 to 1989.

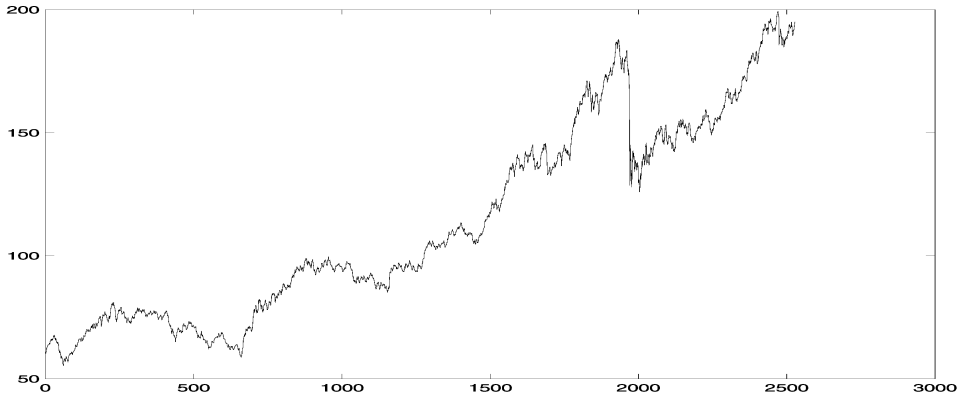


Figure 17.13: The New York Average or NYA from 1980-89, showing the crash of late 1987, the NYA being given as a function of the number of days with the crash occurring around the 2000 mark.

Attempts to develop stochastic models for financial time series (which are just digital signals or 'tick data') can be traced back to the late eighteenth century when Louis Bachelier, who studies under Henri Poincaré (one of the first mathematicians to discuss the principles of chaotic dynamics) in Paris, proposed that fluctuations in the prices of stocks and shares could be viewed in terms of random walks. Bachelier's model (like many of those that have been developed since) failed to predict extreme behaviour in financial signals because of its assumption that such signals conform to Gaussian processes.

A good stochastic financial model should ideally consider all the observable behaviour of the financial system it is attempting to model. It should therefore be able to provide some predictions on the immediate future behaviour of the system within an appropriate confidence level. Predicting the markets has become (for obvious reasons) one of the most important problems in financial engineering. Although

in principle, it might be possible to model the behaviour of each individual agent operating in a financial market, the uncertainty principle should always be respected in that one can never be sure of obtaining all the necessary information required on the agents themselves and their modus operandi. This principle plays an increasingly important role as the scale of the financial system for which a model is required increases. Thus, while quasi-deterministic models can be of value in the understanding of micro-economic systems (with fixed or fuzzy 'operational conditions' for example), in an ever increasing global economy (in which the operational conditions associated with the fiscal policies of a given nation state are increasingly open), we can take advantage of the scale of the system to describe its behaviour in terms of functions of random variables. However, many economists, who like to believe that agents are rational and try to optimize their utility function (essentially, a trade off between profit and risk), are reluctant to accept a stochastic approach to modelling the markets claiming that it is 'an insult to the intelligence of the market(s)'. In trying to develop a stochastic financial model based on a macro-economy, it is still important to respect the fact that the so called global economy is actually currently controlled by three major trading centres (i.e. Tokyo, London and New York, the largest of which, in terms of the full complement of financial services offered is London). Politicians are also generally reluctant to accept a stochastic approach to financial modelling and forecasting. The idea that statistical self-affinity may by a universal law, and that the evolution of an economy and society in general could be the result of one enormous and continuing phase transition does little to add confidence to the worth and 'power spectrum' of politics (until politics itself is considered in the same light!). Nevertheless, since the early 1990s, fractal market analysis has been developed and used to study a wide range of financial time series. This includes different fractal measures such as the Hurst dimension, the Lyapunov exponent, the correlation dimension and multi-fractals. Ultimately, an economy depends on the confidence of those from which it is composed and although governments (and/or central or federal banks) have a variety of control parameters (interest rates being amongst the most important) with which to adjust the pace of economic growth or otherwise, there is no consensus on how to control and accurately define the term confidence.

17.12 The Efficient Market Hypothesis

The unpredictable (stochastic) nature of financial time series or signals is well known and the values of the major indices such as the FTSE (Financial Times Stock Exchange) in the UK, the Dow Jones in the US and the Nikkei Dow in Japan are frequently quoted. The most interesting and important aspects which can affect the global economy in a dramatic and short term way are apparently unpredictable and sudden markets crashes such as the one that took place on the 15th October 1987. In this section, we provide an extensive case study which builds up to the application of the fractional dynamic model discussed earlier in this chapter for multi-fractal financial analysis.

A principal aim of investors is to attempt to obtain information that can provide some confidence in the immediate future of the stock markets based on patterns of the past. One of the principle components of this aim is based on the observation that there are 'waves within waves' and events with events that appear to permeate

financial signals when studied with sufficient detail and imagination. It is these repeating patterns that occupy both the financial investor and the systems modeller alike and it is clear that although economies have undergone many changes in the last one hundred years, the dynamics of market data have not changed considerably. For example, Figure 17.14 shows the build up to two different crashes, the one of 1987 given previously and that of 1997. The similarity in behaviour of these two signals is remarkable and is indicative of the quest to understand economic signals in terms of some universal phenomenon from which appropriate (macro) economic models can be generated.

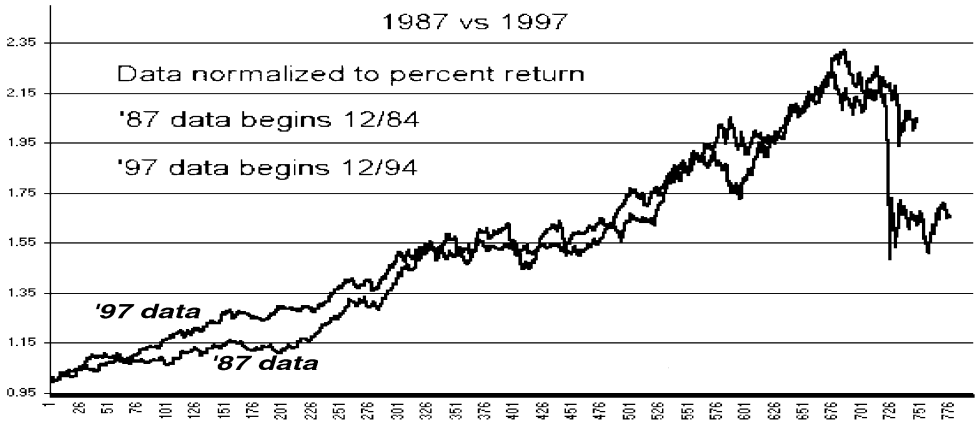


Figure 17.14: The build up to the financial crashes of 1987 and 1997.

Random walk or Martingale models, which embody the notions of the so called Efficient Market Hypothesis (EMH) and successful arbitraging, have been the most popular models of financial time series since the work of Bachelier in 1900. The Black-Scholes model (for which Scholes won a Nobel prize in economics) for valuing options is based on this approach and although Black-Scholes analysis is deterministic (one of the first approaches to achieve a determinism result), it is still based on the EMH. However, it has long been known that financial time series do not follow random walks. The shortcomings of the EMH model include: failure of the independence and Gaussian distribution of increments assumption, clustering, apparent non-stationarity and failure to explain momentous financial events such as crashes. These limitations have prompted a new class of methods for investigating time series obtained from a range of disciplines. For example, Re-scaled Range Analysis (RSRA), originally inspired by the work of Hurst on the Nile river discharges has turned out to be a powerful tool for revealing some well disguised properties of stochastic time series such as persistence, characterized by non-periodic cycles. Non-periodic cycles correspond to trends that persist for irregular periods but with some statistical regularity often caused by non-linear dynamical systems and chaos. RSRA is particularly valuable

because of its robustness in the presence of noise. Ralph Elliott first reported the fractal properties of financial data in 1938. Elliott was the first to observed that segments of financial time series data of different sizes could be scaled in such a way, that they were statistically the same producing so called Elliot waves. Many different random fractal type models for price variation have been developed over the past decade based on iterated function/dynamic systems. These models capture many properties of the time series but are not based on any underlying causal theory.

In economies, the distribution of stock returns and anomalies like market crashes emerge as a result of considerable complex interaction. In the analysis of financial time series, it is inevitable that assumptions need to be made to make the derivation of a model possible. This is the most vulnerable stage of the process. Over simplifying assumptions lead to unrealistic models. There are two main approaches to financial modelling: The first approach is to look at the statistics of market data and to derive a model based on an educated guess of the mechanics of market dynamics. The model can then be tested using real data. The idea is that this process of trial and error helps to develop the right theory of market dynamics. The alternative is to 'reduce' the problem and try to formulate a microscopic model such that the desired behaviour 'emerges', again, by guessing agents' strategic rules. This offers a natural framework for interpretation; the problem is that this knowledge may not help to make statements about the future unless some methods for describing the behaviour can be derived from it. Although individual elements of a system cannot be modelled with any certainty, global behaviour can sometimes be modelled in a statistical sense provided the system is complex enough in terms of its network of interconnection and interacting components. In complex systems, the elements adapt to the world -the aggregate pattern- they co-create. As the components react, the aggregate changes, as the aggregate changes the components react anew. Barring the reaching of some asymptotic state or equilibrium, complex systems keep evolving, producing seemingly stochastic or chaotic behaviour. Such systems arise naturally in the economy. Economic agents, be they banks, firms, or investors, continually adjust their market strategies to the macroscopic economy which their collective market strategies create. It is important to appreciate at this point, that there is an added layer of complexity within the economic community: Unlike many physical systems, economic elements (human agents) react with strategy and foresight (some of the time) by considering the implications of their actions although it is not certain whether this fact changes the resulting behaviour.

What do complex systems have to do with fractals? The property of feedback is the very essence of both economic/complex systems and chaotic dynamical systems that produce fractal structures. The link between dynamical systems, chaos and the economy is an important one because it is dynamical systems that illustrate that local randomness and global determinism can co-exist. Complex systems can be split into two categories: equilibrium and non-equilibrium. Equilibrium complex systems, undergoing a phase transition, can lead to 'critical states' that often exhibit fractal structures. A simple example is the heating of ferromagnets. As the temperature rises, the atomic spins in the magnetic field gain energy and begin to change in direction. When the temperature reaches some critical value, the spins form a random vector field with mean zero and a phase transition occurs in which the magnetic field averages to zero. The statistics of this random field are scale invariant or fractal. Non-

equilibrium complex systems or 'driven' systems give rise to 'self organised critical states', an example is the growing of sand piles. If sand is continuously supplied from above, the sand starts to pile up. Eventually, little avalanches will occur as the sand pile inevitably spreads outwards under the force of gravity. The temporal and spatial statistics of these avalanches are scale invariant.

Financial markets can be considered to be non-equilibrium systems because they are constantly driven by transactions that occur as the result of new fundamental information about firms and businesses. They are complex systems because the market also responds to itself in a highly non-linear fashion, and would carry on doing so (at least for some time) in the absence of new information. This is where confusion commonly arises. The 'price change field' is highly non-linear and very sensitive to exogenous shocks and it is highly probable that all shocks have a long term affect. Market transactions generally occur globally at the rate of hundreds of thousands per second. It is the frequency and nature of these transactions that dictate stock market indices, just as it is the frequency and nature of the sand particles that dictates the statistics of the avalanches in a sand pile. These are all examples of random scaling fractals. Further, they are example of systems that often conform to Lévy distributions discussed earlier in this chapter.

17.13 Market Analysis

In 1900, Louis Bachelier concluded that the price of a commodity today is the best estimate of its price in the future. The random behaviour of commodity prices was again noted by Working in 1934 in an analysis of time series data. Kendall (1953) attempted to find periodic cycles in indices of security and commodity prices but did not find any. Prices appeared to be yesterday's price plus some random change and he suggested that price changes were independent and that prices apparently followed random walks. The majority of financial research seemed to agree; asset price changes are random and independent, so prices follow random walks. Thus, the first model of price variation was just the sum of independent random numbers often referred to as Brownian motion (BM) after Robert Brown (1828) who studied the erratic movement of a small particle suspended in a fluid.

Some time later, it was noticed that the size of price movements depends on the size of the price itself. The model was revised to include this proportionality effect, and so a new model was developed that stated that the log price changes should be Gaussian distributed. This behaviour can be described mathematically by a model of the form

$$\frac{dS}{S} = \sigma dX + \mu dt$$

where S is the price at time t, μ is a drift term which reflects the average rate of growth of the asset, σ is the volatility and dX is a sample from a normal distribution. In other words, the relative price change of an asset is equal to some random element plus some underlying trend component. More precisely, this model is a lognormal random walk. The Brownian motion model has the following important properties:

1. Statistical stationarity of price increments. Samples of Brownian motion taken

over equal time increments can be superimposed onto each other in a statistical sense.

2. Scaling of price. Samples of Brownian motion corresponding to different time increments can be suitably re-scaled such that they too, can be superimposed onto each other in a statistical sense.

3. Independence of price changes.

Why should prices follow Gaussian random walks? It is often stated that asset prices should follow random walks because of the Efficient Market Hypothesis (EMH). The EMH states that the current price of an asset fully reflects all available information relevant to it and that new information is immediately incorporated into the price. Thus, in an efficient market, the modelling of asset prices is really about modelling the arrival of new information. New information must be independent and random, otherwise it would been anticipated and would not be new. The EMH implies independent price increments but why should they be Gaussian distributed? The Gaussian PDF is chosen because most price movements are presumed to be an aggregation of smaller ones, and sums of independent random contributions have a Gaussian PDF.

The arrival of new information actually sends 'shocks' through the market as people react to it and then to each other's reactions. The EMH assumes that there is a rational (sensible) and unique way to use the available information and that all agents possess this knowledge. Moreover, the EMH assumes that this chain reaction happens instantaneously. In an efficient market, only the revelation of some dramatic information can cause a crash, yet post-mortem analysis of crashes typically fail to (convincingly) tell us what this information must have been.

In order to understand the nature of an economy, as with any other signal, one needs to be clear about what assumptions are being made in order to develop suitable models and have some way to test them. We need to consider what is happening at the microscopic level as well as the macroscopic level on which we observe financial time series, which are often averages of composites of many fundamental economic variables. It is therefore necessary to introduce some of the approaches and issues associated with financial engineering which is given in the following sections.

17.13.1 Risk .v. Return: Arbitrage

For many years, investment advisers focused on returns with the occasional caveat 'subject to risk'. Modern Portfolio Theory (MPT) teaches that there is a trade off between risk and return. Nearly all finance theory assumes the existence of risk-free investment, e.g. the return from depositing money in a sound financial institute or investing in equities. In order to gain more profit, the investor must accept greater risk. Why should this be so? Suppose the opportunity exists to make a guaranteed return greater than that from a conventional bank deposit say; then, no (rational) investor would invest any money with the bank. Furthermore, if he/she could also borrow money at less than the return on the alternative investment, then the investor would borrow as much money as possible to invest in the higher yielding opportunity. In response to the pressure of supply and demand, the banks would raise their interest rates. This would attract money for investment with the bank and reduce the profit

made by investors who have money borrowed from the bank. (Of course, if such opportunities did arise, the banks would probably be the first to invest our savings in them.) There is elasticity in the argument because of various friction factors such as transaction costs, differences in borrowing and lending rates, liquidity laws etc., but on the whole, the principle is sound because the market is saturated with arbitrageurs whose purpose is to seek out and exploit irregularities or miss-pricing.

The concept of successful arbitraging is of great importance in finance. Often loosely stated as, 'there's no such thing as a free lunch', it means that, in practice, one cannot ever make an instantaneously risk-free profit. More precisely, such opportunities cannot exist for a significant length of time before prices move to eliminate them.

17.13.2 Financial Derivatives

As markets have grown and evolved, new trading contracts have emerged which use various tricks to manipulate risk. Derivatives are deals, the value of which is derived from (although not the same as) some underlying asset or interest rate. There are many kinds of derivatives traded on the markets today. These special deals really just increase the number of moves that players of the economy have available to ensure that the better players have more chance of winning. For example, anyone who has played naughts and crosses a few times will know that once a certain level has been reached every game should be a draw. Chess, on the other hand, is a different matter. To illustrate some of the implications of the introduction of derivatives to the financial markets we consider the most simple and common derivative, namely, the option.

Options

An option is the right (but not the obligation) to buy (call) or sell (put) a financial instrument (such as a stock or currency, known as the 'underlying') at an agreed date in the future and at an agreed price, called the strike price. For example, consider an investor who 'speculates' that the value of a stock, XYZ, will rise. The investor could buy shares in XYZ, and if appropriate, sell them later at a higher price to make money. Alternatively, the investor might buy a call option, the right to buy an XYZ share at a later date. If the asset is worth more than the strike price on expiry, the holder will be content to exercise the option, immediately sell the stock at the higher price and generate an automatic profit from the difference. The catch is, that if the price is less, the holder must accept the loss of the premium paid for the option (which must be paid for at the opening of the contract). Denoting C to be the value of a call option, S the asset price and E to be the strike price, the option is worth

$$C(S,t) = \max(S - E, 0).$$

Conversely, suppose the investor speculates that XYZ shares are going to fall, then the investor can sell shares or buy puts. If the investor speculates by selling shares that he/she does not own (which in certain circumstances is perfectly legal in many markets), then he/she is selling 'short' and will profit from a fall in XYZ shares. (The opposite of a short position is a 'long' position.) The principal question is then, how much should one pay for an option? Clearly, if the value of the asset

rises so does the value of a call option and vice versa for put options. But how do we quantify exactly how much this gamble is worth? In previous times (prior to the Black-Scholes model which is discussed later) options were bought and sold for the value that individual traders thought they ought to have. The strike prices of these options were usually the 'forward price', which is just the current price adjusted for interest-rate effects. The value of options rises in active or volatile markets because options are more likely to pay out large amounts of money when they expire if market moves have been large. That is, potential gains are higher, but loss is always limited to the cost of the premium. This gain through successful 'speculation' is not the only role options play.

Hedging

Suppose an investor already owns XYZ shares as a long-term investment, then he/she may wish to insure against a temporary fall in the share price by buying puts as well. Clearly, the investor would not want to liquidate the XYZ holdings only to buy them back again later, possibly at a higher price if the estimate of the share price is wrong, and certainly having incurred some transaction costs on the two deals. If a temporary fall occurs, the investor has the right to sell his/her holdings for a higher than market price. The investor can then immediately buy them back for less, in this way generating a profit and long-term investment then resumes. If the investor is wrong and a temporary fall does not occur, then the premium is lost for the option but at least the stock is retained, which has continued to rise in value. Furthermore, since the value of a put option rises when the underlying asset value falls, what happens to a portfolio containing both assets and puts? The answer depends on the ratio. There must exist a ratio at which a small unpredictable movement in the asset does not result in any unpredictable movement in the portfolio. This ratio is instantaneously risk free. The reduction of risk by taking advantage of correlations between the option price and the underlying price is called 'hedging'. If a market maker can sell an option and hedge away all the risk for the rest of the options life, then a risk free profit is guaranteed.

Why write options? Options are usually sold by banks to companies to protect themselves against adverse movements in the underlying price, in the same way as holders do. In fact, writers of options are no different to holders, they expect to make a profit by taking a view of the market. The writers of calls are effectively taking a short position in the underlying behaviour of the markets. Known as 'bears', these agents believe the price will fall and are therefore also potential customers for puts. The agents taking the opposite view are called 'bulls'. There is a near balance of bears and bulls because if everyone expected the value of a particular asset to do the same thing, then its market price would stabilise (if a reasonable price were agreed on) or diverge (if everyone thought it would rise). Clearly, the psychology and dynamics (which must go hand in hand) of the bear/bull cycle play an important role in financial analysis.

The risk associated with individual securities can be hedged through diversification and/or various other ways of taking advantage of correlations between different derivatives of the same underlying asset. However, not all risk can be removed by diversification. To some extent, the fortunes of all companies move with the economy.

Changes in the money supply, interest rates, exchange rates, taxation, the prices of commodities, government spending and overseas economies tend to affect all companies in one way or another. This remaining risk is generally referred to as market risk.

17.13.3 Black-Scholes Analysis

The value of an option can be thought of as a function of the underlying asset price S (a random variable) and time t denoted by $V(S,t)$. Here, V can denote a call or a put; indeed, V can be the value of a whole portfolio or different options although for simplicity we can think of it as a simple call or put. Any derivative security whose value depends only on the current value S at time t and which is paid for up front, satisfies the Black-Sholes equation given by

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

where σ is the volatility and r is the risk. As with other partial differential equations, an equation of this form may have many solutions. The value of an option should be unique; otherwise, again, arbitrage possibilities would arise. Therefore, to identify the appropriate solution, appropriate initial, final and boundary conditions need to be imposed. Take for example, a call; here the final condition comes from the arbitrage argument. At $t = T$

$$C(S,t) = \max(S - E, 0).$$

The spatial or asset-price boundary conditions, applied at $S = 0$ and $S \to \infty$ come from the following reasoning: If S is ever zero then dS is zero and will therefore never change. Thus, we have

$$C(0,t) = 0.$$

As the asset price increases it becomes more and more likely that the option will be exercised, thus we have

$$C(S,t) \propto S, \qquad S \to \infty.$$

Observe, that the Black-Sholes equation has a similarity to the diffusion equation but with extra terms. An appropriate way to solve this equation is to transform it into the diffusion equation for which the solution is well known and with appropriate transformations gives the Black-Scholes formula

$$C(S,t) = SN(d_1) - Ee^{r(T-t)}N(d_2)$$

where

$$d_1 = \frac{\log(S/E) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$

$$d_2 = \frac{\log(S/E) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

and N is the cumulative normal distribution defined by

$$N(d_1) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{d_1} e^{\frac{1}{2}s^2} ds.$$

The conceptual leap of the Black-Scholes model is to say that traders are not estimating the future price, but are guessing about how volatile the market may be in the future. The volatility is mathematically defined and it is assumed that the distribution of these moves is lognormal. The model therefore allows banks to define a fair value of an option, because it assumes that the forward price is the mean of the distribution of future market prices and that the volatility is known. Figure 17.15 illustrates an example of the Black-Sholes analysis of the price of a call option as a function of S, at 4 time intervals approaching expiry.

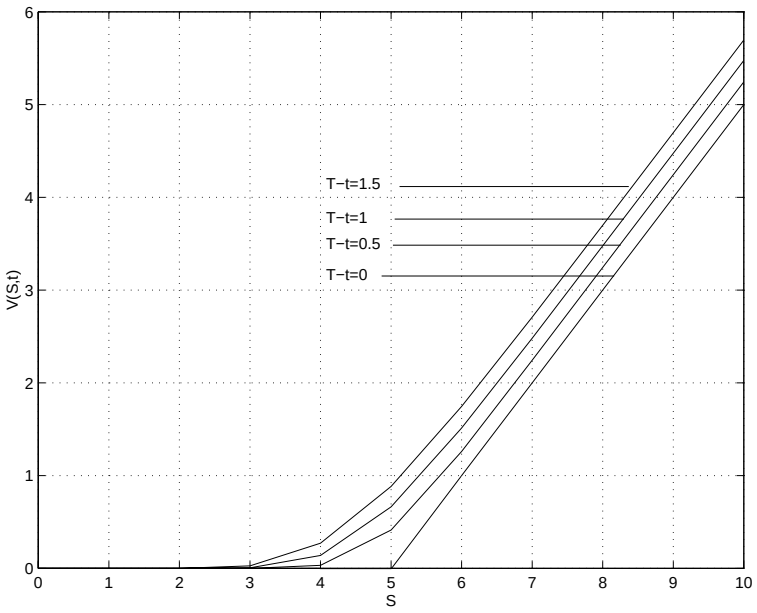


Figure 17.15: The Black-Scholes value of a call option as a function of S at 4 successive time intervals $T - t$ approaching expiry $E = 5$.

The simple and robust way of valuing options using Black-Scholes analysis has rapidly gained in popularity and has universal applications. The Black-Scholes volatility and the price of an option are now so closely linked into the markets that the price of an option is usually quoted in option volatilities or 'vols', which are displayed on traders' screens across the world. Nevertheless, the majority of methods (particularly Black-Scholes analysis) used by economic agents are based on random walk models that assume independent and Gaussian distributed price changes.

A theory of modern portfolio management, like any theory, is only valuable if we can be sure that it truly reflects reality for which tests are required. The following section investigates empirically the properties of real financial data as observed macroscopically in the form of financial time indices.

17.13.4 Macro-Economic Models

The modern portfolio rationalization of the random walk hypothesis introduced in the previous section is ultimately based on the EMH. Figure 17.16 shows the de-trended log price increments[6] of the New York Average (NYA) from 1960-89 (top) and the EMH's proposed random walk Gaussian increments (below). Both sets have been normalized and the EMH signal plotted using an offset of -5 for comparison. It is clear, that there is a significant difference between the two stochastic signals. This simple comparison indicates a failure of the statistical independence assumption using for the EMH.

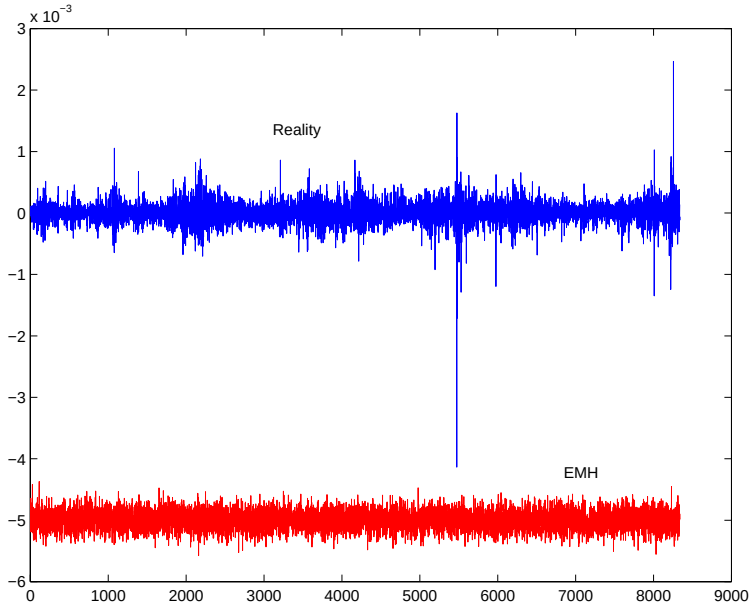


Figure 17.16: Real data (top) versus the random walk hypothesis (bottom) with mean displaced to -5 for comparative display purposes.

In general, when faced with a multidimensional process of unknown origin, it is usual to select an independent process such as Brownian motion as a working hypothesis. If analysis shows that a prediction is difficult, the hypothesis is accepted. Brownian motion is elegant because statistics and probabilities can be estimated with great accuracy. However, using traditional statistics to model the markets assumes that they are games of chance. For this reason, investment in securities is often equated with gambling. In most games of chance, many degrees of freedom are employed to ensure that outcomes are random. Take a simple dice, a coin, a roulette wheel etc. No matter how hard you may try, it is physically impossible to master your roll or throw such that you can control outcomes. There are too many non-repeatable elements (speeds, angles and so on) and non-linearly compounding errors involved.

[6]Removal of the long term exponential growth trend by taking the logarithm and then differentiating the data (using finite differencing).

Although these systems have a limited number of degrees of freedom, each outcome is independent of the previous one. However, there are some games of chance that involve memory. Take Blackjack for example, also known as '21'. Two cards are dealt to each player and the object is to get as close as possible to 21 by twisting (taking another card) or sticking. If you go 'bust' (over 21) you lose; the winner is the player that sticks closest to 21. Here, memory is introduced because the cards are not replaced once they are taken. By keeping track of the cards used, one can assess the shifting probabilities as play progresses. This game illustrates that not all gambling is governed by Gaussian statistics. There are processes that have long-term memory, even though they are probabilistic in the short term. This leads directly to the question, does the economy have memory? A system has memory if what happens today will affect what happens in the future. We can test for memory effects by testing for correlations in the data. If the system today has no affect on the system at any future time, then the data produced by the system will be independently distributed and there will be no correlations. Now, the Auto-Correlation Function (ACF), which describes the expected correlations between time periods t apart, is defined by (see Chapter 4)

$$A(t) = \langle X(\tau)X(\tau - t)\rangle \equiv \int_{-\infty}^{\infty} X(\tau)X(\tau - t)d\tau.$$

Since prices themselves, like Brownian motion, are a non-stationary process, there is no ACF as such. However, if we calculate the ACF of the price increments, which are Gaussian by null hypothesis, then we can observe how much of what happens today is correlated with what happens in the future. This can be undertaken using the Power Spectral Density Function (PSDF), the Fourier transform of the autocorrelation function, i.e.

$$P(\omega) = \hat{F}_1\langle X(\tau)X(\tau - t)\rangle \equiv \int_{-\infty}^{\infty} \langle X(\tau)X(\tau - t)\rangle \exp(-i\omega t)dt$$

where $\hat{F}_1$ denotes the Fourier Transform operator. The PSDF tells us the amplitude distribution of the correlation function from which we can estimate the time span of the memory effects. This also offers a convenient way to calculate the correlation function (by taking the inverse Fourier transform of the PSDF). If the PSDF has more power at low frequencies, then there are long time correlations and therefore long-term memory effects. Inversely, if there is greater power at the high frequency end of the spectrum, then there are short-term time correlations and evidence of short-term memory. According to traditional theory, the economy has no memory and there will therefore be no correlations, except for today with itself. We should therefore expect a PSDF that is effectively constant as would result from a delta function autocorrelation function i.e. if

$$\langle X(\tau)X(\tau - t)\rangle = \delta(t)$$

then

$$P(\omega) = \int_{-\infty}^{\infty} \delta(t)\exp(-i\omega t)dt = 1.$$

The PSDFs and ACFs of log price changes and the absolute log price changes of the NYA are shown in Figure 17.17.

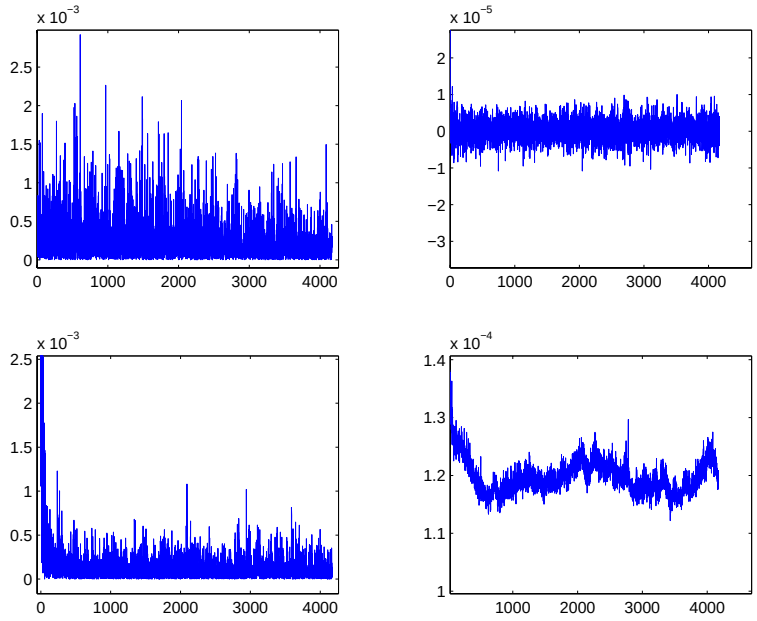


Figure 17.17: PSDF (left) and ACF (right) of log price movements (top) and absolute log price movements (bottom) as a function of days.

The PSDF of the data is not constant; there is evidence of a power law at the low frequency end and rogue spikes (or groups of spikes) at the higher frequency end of the spectrum. This power law is even more pronounced in the PSDF of the absolute log price increments, indicating that there is additional correlation in the signs of the data. If we denote q to be the parameter of this power law then we can consider a simple power law of the form

$$P(\omega) \propto 1/\omega^q$$

to describe the power law feature which is the principal signature for a random fractal signal. The ACF of the Log Price Increments (LPI) appears featureless, indicating that the excess of low frequency power within the signal (as described by PSDF) has a fairly subtle affect on the correlation function. The ACF of the absolute LPI, however, contains some interesting features. It shows that there are high short range correlations followed by a rapid decline up to approximately 100 days, followed by a steadier decline up to about 600 days when the correlations start to develop again, peaking at about 2225 days. The system governing the magnitudes of the log price movements seems to have a better long-term memory than it should. The data used in this analysis contains 8337 price movements. Thus, a spike in the PSDF at frequency ω is based on the values of $8337/\omega$ data points, and since most of this power is at the low frequency end (more points), it is improbable that these results have occurred by coincidence.

To investigate short-term correlations we can carry out a simple experiment that involves evaluating the average length of a run of either larger than or smaller than average price increments. The question we are asking is: are below average (negative)

and above average (positive) log price changes distributed randomly or do they come in 'bursts'. Given that there is a fixed probability of continuing a run of either above or below average increments, the probability of a run of length r of successive occurrences of p is, $\Pr(r) = p^r(1-p)$ and the expected run length is $E(r) = p/(1-p)$. If the data were independently distributed then we expect $p = 0.5$ and $E(r) = 1$. If the data were negatively correlated, $E(r) < 1$ and if the data were positively correlated $E(r) > 1$. In the data set for NYA $\bar{r} = 1.2695$ for LPI and $\bar{r} = 1.3027$ for the absolute LPI, where $\bar{r}$ denotes the average run length. Assuming that the bias for large and small movements is the same, this makes the probability of a run continuing equal to $p = 0.565$ and $p = 0.5657$ respectively. This can be tested for significance under the independence null hypothesis and the assumption of Gaussian error in measuring the means. The probability of observing $\bar{r} = 1.3$ is $p < 1 \times 10^{-16}$. Therefore it is highly improbable that this data originates from an independent distribution and we can conclude that the correlations are significant. Modelling 'trends' in this way assumes that the PDF for trend 'innings' in the real data follows the same distribution as $\Pr(r)$, as it does.

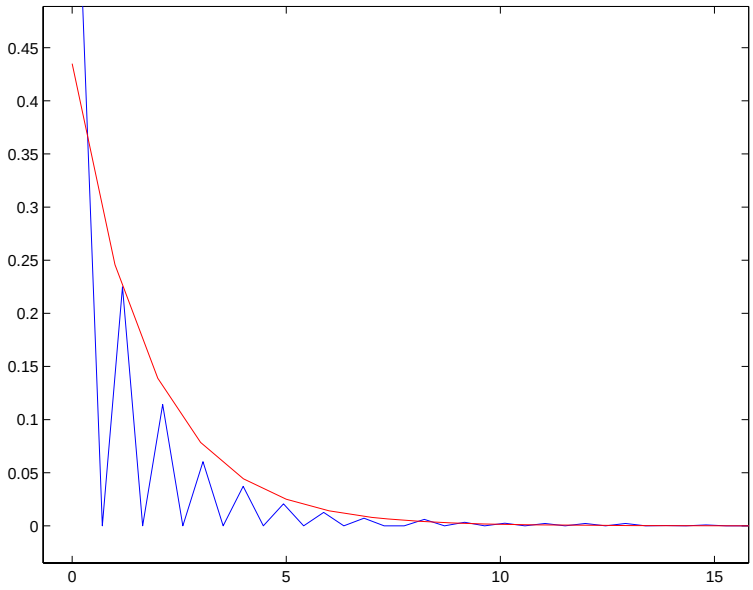


Figure 17.18: Distribution of trend innings of the NYA compared with $\Pr(r) = p^r(1-p)$ (smooth line) for $p = 0.565$.

Figure 17.18 shows the distribution of 'runs' for the NYA along with the theoretical distribution, characterized by $\bar{r} = 1.3$. Correlations in (log) price changes invalidates the independence assumption of the EMH. It does not necessarily invalidate the Gaussian distribution of price changes. Figure 17.19 shows the frequency distribution of daily and 5-daily New York Average (NYA) returns from 1960 to 1999. The two return distributions are virtually the same, implying self-affine behaviour. They are also uni-modal. There are two main features in which the distribution of returns differs from the Gaussian: A high peak at the mean and longer tails. Ninety

nine percent of a normal distribution is contained within the main six standard deviations. The distribution of returns takes a lot longer to converge to zero. Even at four standard deviations from the mean, there are as many observations as there are at two.

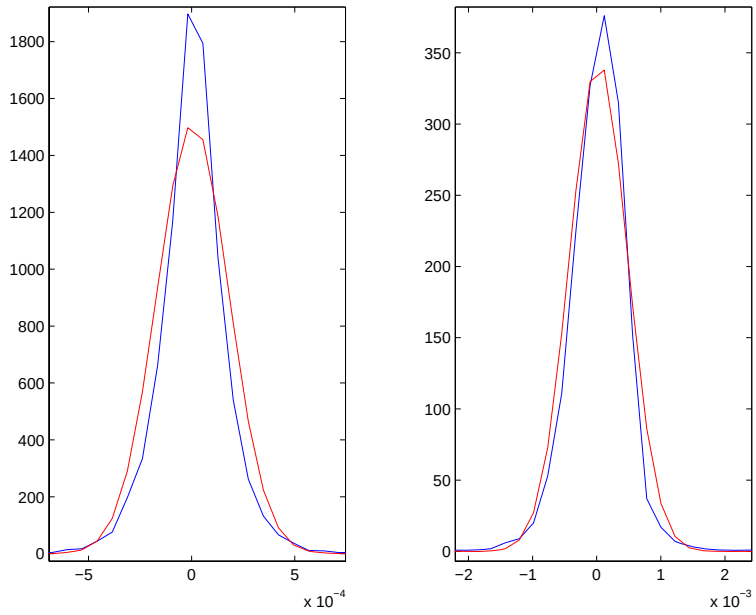


Figure 17.19: PDF of daily (left) and 5-daily (right) returns of the NYA stock market index along with the Gaussian distribution (dashed-line) for comparison.

The long tails are an important feature of this distribution. It is from the distribution of returns that we measure risk. Risk is often referred to as the, 'one in six' rule which refers to the Gaussian property that approximately 4/6 of all outcomes lie within one standard deviation of the expected outcome and there is therefore a one in six chance of a very good (upside potential) or very bad (downside risk) outcome. In reality, the risk of large events occurring is much higher than the normal distribution implies. As we measure still larger events, the gap between theory and reality becomes even more pronounced. This risk is virtually identical in all the investment horizons investigated. We can quantify the tails deviation from a normal distribution using kurtosis - long tails being referred to as excess kurtosis (see Chapter 16) since the kurtosis is zero for the normal distribution and typically between 2 and 50 for daily returns and even higher for intraday data. A number of statistical models have been suggested to account for excess kurtosis including Mandelbrot's Stable Paretian hypothesis, the mixture of distributions hypothesis and models based on conditional heteroskedasticity which refers to the condition where residual variance is non-constant. In general, market data exhibits generalized autoregressive heteroskedasticity which means that there are periods of persistent high volatility followed randomly by periods of persistent low volatility.

Another basic assumption that comes with the normal distribution involves the

scaling of volatility known as the term structure of volatility. Typically, we use the standard deviation of returns to measure volatility. For independent or 'white noise' processes, standard deviations scale according to the square root of time. For example, we can 'annualize' the standard deviation of monthly returns by multiplying it by the square root of 12. Figure 17.20 shows the NYA volatility term structure (1989-1990) along with the theoretical scaling line. Volatility grows linearly at a faster rate than the square root of time up until about 1000 days or 4 years, it then slows down dramatically. If we use volatility as a measure of risk, investors incur more risk than is implied by the normal distribution for investment horizons of less than 4 years. However, investors incur increasingly less risk for investment horizons of greater than 4 years. This verifies what is known intuitively, namely, that long-term investors incur less risk than short-term investors.

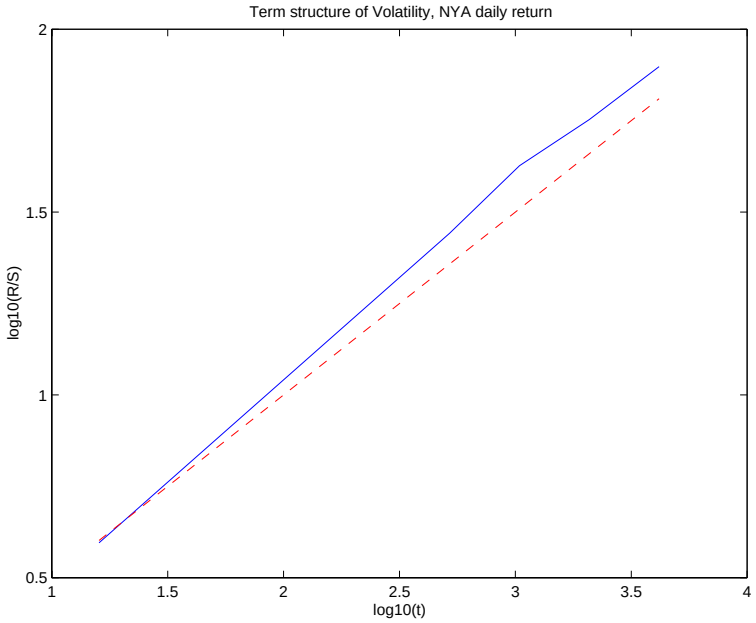


Figure 17.20: Term structure of volatility of the NYA, along with theoretical scaling line (dashed line).

If a financial walk (in the random walk sense) scales according to the power law,

$$\left\langle \frac{d}{dt} \log u(t) \right\rangle \propto t^H$$

where $u(t)$ denotes price movements for time t, then the walk is self-affine because the distribution of steps must look the same at different scales. For a non-constant PSDF, this shows that the phases of the different frequency components that make up the PSDF are randomized. The fact that data scales with a consistent power law up to 1000 days reveals that there is self-affinity in market return data, but only over a certain range.

17.13.5 Fractal Time Series and Rescaled Range Analysis

A time series is fractal if the data exhibits statistical self-affinity and has no characteristic scale and the results of the previous section have verified that there is self-affinity in market data. The data has no characteristic scale if it has a PDF with an infinite second moment. The data may have an infinite first moment as well; in this case, the data would have no stable mean either. One way to test the financial data for the existence of these moments is to plot them sequentially over increasing time periods to see if they converge. Figure 17.21 shows that the first moment, the mean, is stable, but that the second moment, the mean square, is not settled. It converges and then suddenly jumps, as a Lévy flight with an infinite mean square would do. It is interesting to observe that although the variance is not stable, the jumps occur with some statistical regularity.

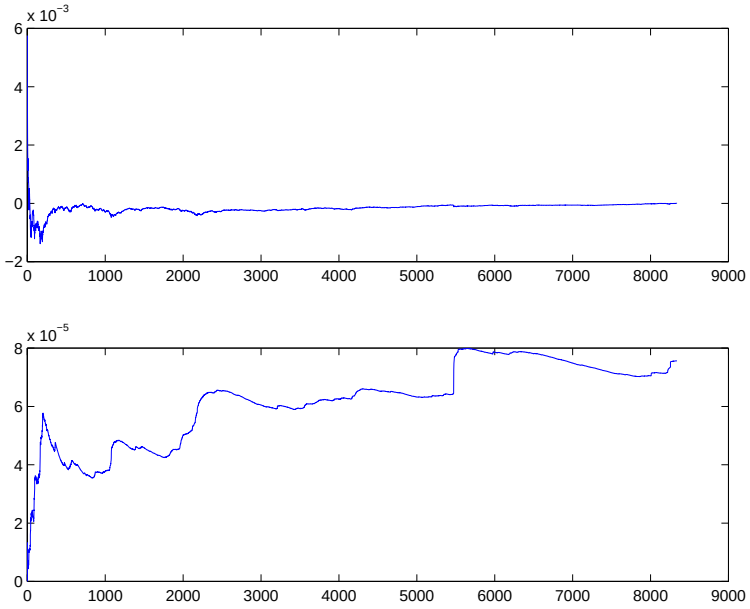


Figure 17.21: The first and second moments (top and bottom) of the NYA plotted sequentially.

Rescaled Range Analysis: Measuring Memory

H E Hurst (1900-1978) was an English civil engineer who built dams and worked on the Nile river dam project. He studied the Nile so extensively that some Egyptians reportedly nicknamed him 'the father of the Nile.' The Nile river posed an interesting problem for Hurst as a hydrologist. When designing a dam, hydrologists need to estimate the necessary storage capacity of the resulting reservoir. An influx of water occurs through various natural sources (rainfall, river overflows etc.) and a regulated amount needed to be released for primarily agricultural purposes. The storage capacity of a reservoir is based on the net water flow. Hydrologists usually begin by

assuming that the water influx is random, a perfectly reasonable assumption when dealing with a complex ecosystem. Hurst, however, had studied the 847-year record that the Egyptians had kept of the Nile river overflows, from 622 to 1469. Hurst noticed that large overflows tended to be followed by large overflows until abruptly, the system would then change to low overflows, which also tended to be followed by low overflows. There seemed to be cycles, but with no predictable period. Standard statistical analysis revealed no significant correlations between observations, so Hurst developed his own methodology. Hurst was aware of Einstein's (1905) work on Brownian motion (the erratic path followed by a particle suspended in a fluid) who observed that the distance the particle covers increased with the square root of time, i.e.

$$R = \sqrt{t}$$

where R is the range covered, and t is time. This is the same scaling property as discussed earlier in the context of volatility. It results, again, from the fact that increments are identically and independently distributed random variables. Hurst's idea was to use this property to test the Nile River's overflows for randomness. In short, his method was as follows: Begin with a time series x_i (with $i = 1, 2, ..., n$) which in Hurst's case was annual discharges of the Nile River. (For markets it might be the daily changes in the price of a stock index.) Next, create the adjusted series, $y_i = x_i - \bar{x}$ (where $\bar{x}$ is the mean of x_i). Cumulate this time series to give

$$Y_i = \sum_{j=1}^{i} y_j$$

such that the start and end of the series are both zero and there is some curve in between. (The final value, Y_n has to be zero because the mean is zero.) Then, define the range to be the maximum minus the minimum value of this time series,

$$R_n = \max(Y) - \min(Y).$$

This adjusted range, R_n is the distance the systems travels for the time index n, i.e. the distance covered by a random walker if the data set y_i were the set of steps. If we set $n = t$ we can apply Einstein's equation provided that the time series x_i is independent for increasing values of n. However, Einstein's equation only applies to series that are in Brownian motion. Hurst's contribution was to generalize this equation to

$$(R/S)_n = cn^H$$

where S is the standard deviation for the same n observations and c is a constant. We define a Hurst process to be a process with a (fairly) constant H value and the R/S is referred to as the 'rescaled range' because it has zero mean and is expressed in terms of local standard deviations. In general, the R/S value increases according to a power law value equal to H known as the Hurst exponent. This scaling law behaviour is the first connection between Hurst processes and fractal geometry.

Rescaling the adjusted range was a major innovation. Hurst originally performed this operation to enable him to compare diverse phenomenon. Rescaling, fortunately, also allows us to compare time periods many years apart in financial time series. As discussed previously, it is the relative price change and not the change itself that is

of interest. Due to inflationary growth, prices themselves are a significantly higher today than in the past, and although relative price changes may be similar, actual price changes and therefore volatility (standard deviation of returns) are significantly higher. Measuring in standard deviations (units of volatility) allows us to minimize this problem. Rescaled range analysis can also describe time series that have no characteristic scale, another characteristic of fractals. By considering the logarithmic version of Hurst's equation, i.e.

$$\log(R/S)_n = \log(c) + H\log(n)$$

it is clear that the Hurst exponent can be estimated by plotting $\log(R/S)$ against the $\log(n)$ and solving for the gradient with a least squares fit. If the system were independently distributed, then $H = 0.5$. Hurst found that the exponent for the Nile River was $H = 0.91$, i.e. the rescaled range increases at a faster rate than the square root of time. This meant that the system was covering more distance than a random process would, and therefore the annual discharges of the Nile had to be correlated.

It is important to appreciate that this method makes no prior assumptions about any underlying distributions, it simply tells us how the system is scaling with respect to time. So how do we interpret the Hurst exponent? We know that $H = 0.5$ is consistent with an independently distributed system. The range $0.5 < H \leq 1$, implies a persistent time series, and a persistent time series is characterized by positive correlations. Theoretically, what happens today will ultimately have a lasting effect on the future. The range $0 < H \leq 0.5$ indicates anti-persistence which means that the time series covers less ground than a random process. In other words, there are negative correlations. For a system to cover less distance, it must reverse itself more often than a random process.

The Joker Effect

After this discovery, Hurst analysed all the data he could including rainfall, sunspots, mud sediments, tree rings and others. In all cases, Hurst found H to be greater than 0.5. He was intrigued that H often took a value of about 0.7 and Hurst suspected that some universal phenomenon was taking place. He carried out some experiments using numbered cards. The values of the cards were chosen to simulate a PDF with finite moments, i.e. $0, \pm 1, \pm 3, \pm 5, \pm 7 \text{and} \pm 9$. He first verified that the time series generated by summing the shuffled cards gave $H = 0.5$. To simulate a bias random walk, he carried out the following steps.

1. Shuffle the deck and cut it once, noting the number, say n.
2. Replace the card and re-shuffle the deck.
3. Deal out 2 hands of 26 cards, A and B.
4. Replace the lowest n cards of deck B with the highest n cards of deck A, thus biasing deck B to the level n.
5. Place a joker in deck B and shuffle.
6. Use deck B as a time series generator until the joker is cut, then create a new biased hand.

Hurst did 1000 trials of 100 hands and calculated $H = 0.72$. We can think of the process as follows: we first bias each hand, which is determined by a random cut of the pack; then, we generate the time series itself, which is another series of random cuts; then, the joker appears, which again occurs at random. Despite all of these random events $H = 0.72$ would always appear. This is called the 'joker effect'. The joker effect, as illustrated above, describes a tendency for data of a certain magnitude to be followed by more data of approximately the same magnitude, but only for a fixed and random length of time. A natural example of this phenomenon is in weather systems. Good weather and bad weather tend to come in waves or cycles (as in a heat wave for example). This does not mean that weather is periodic, which it is clearly not. We use the term 'non-periodic cycle' to describe cycles of this kind (with no fixed period). Thus, if markets are Hurst processes, they exhibit trends that persist until the economic equivalent of the joker comes along to change that bias in magnitude and/or direction. In other words rescaled range analysis can, along with the PDF and PSDF, help to describe a stochastic time series that contains within it, many different short-lived trends or biases (both in size and direction). The process continues in this way giving a constant Hurst exponent, sometimes with flat episodes that correspond to the average periods of the non-periodic cycles, depending on the distribution of actual periods. Figure 17.22 shows RSRA performed on a synthetic data set characterized by an expected length of a trend of 100 days, or $p(x_i > \bar{x} \mid x_{i-1} > \bar{x}) = 100/101 = 0.9901$. In this case, the RSRA results give no visual indication of what the average run length is. The Hurst exponent, however is a direct representation of the amount of persistence in the data, which is related to p and $E(r)$.

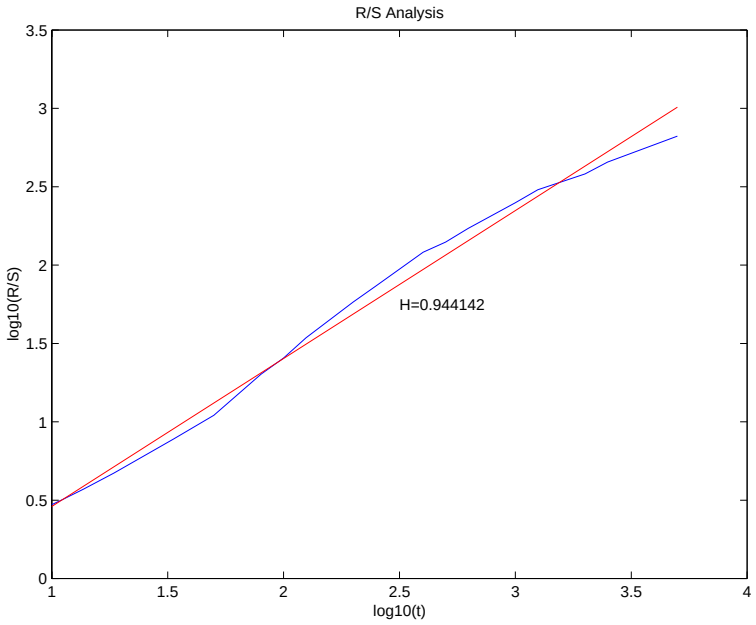


Figure 17.22: Rescaled range analysis on synthesized data with $E(r) = 100$

The following is a step by step methodology for applying R/S analysis to stock market data. Note that the AR(1) notation used below stands for auto regressive process with single daily linear dependence. Thus, taking AR(1) residuals of a signal is equivalent to plotting the signals one day out of phase and taking the day to day linear dependence out of the data.

1. Prepare the data P_t. Take AR(1) residuals of log ratios. The log ratios account for the fact that price changes are relative, i.e. depend on price. The AR(1) residuals remove any linear dependence, serial correlation, or short-term memory which can bias the analysis.

$$V_t = \log(P_t/P_{t-1})$$

$$X_t = V_t - (c + mV_{t-1})$$

   The AR(1) residuals are taken to eliminate any linear dependency.

2. Divide this time series (of length N) up into A sub-periods, such that the first and last value of time series are included i.e. $A \times n = N$. Label each sub-period I_a with $a = 1, 2, 3, ..., A$. Label each element in I_a with $X_{k,a}$ where $k = 1, 2, 3, ..., n$. For each I of length n, calculate the mean

$$e_a = (1/n) \sum_{i=1}^{k} N_{k,a}$$

3. Calculate the time series of accumulated departures from the mean for each sub interval.

$$Y_{k,a} = \sum_{i=1}^{k} (N_{i,a} - e_a)$$

4. Define the range as

$$R_{I_a} = \max(Y_{k,a}) - \min(Y_{k,a})$$

   where $1 \leq k \leq n$.

5. Define the sample standard deviation for each sub-period as

$$S_{I_a} = \sqrt{\frac{1}{n} \sum_{k=1}^{n} (N_{k,a} - e_a)^2}$$

6. Each range, R_{I_a} is now normalized by dividing by its corresponding S_{I_a}. Therefore the re-scaled range for each I_a is equal to R_{I_a}/S_{I_a}. From step 2 above, we have A contiguous sub-periods of length n. Therefore the average R/S value for each length n is defined as

$$(R/S)_n = \frac{1}{A} \sum_{a=1}^{A} (R_{I_a}/S_{I_a})$$

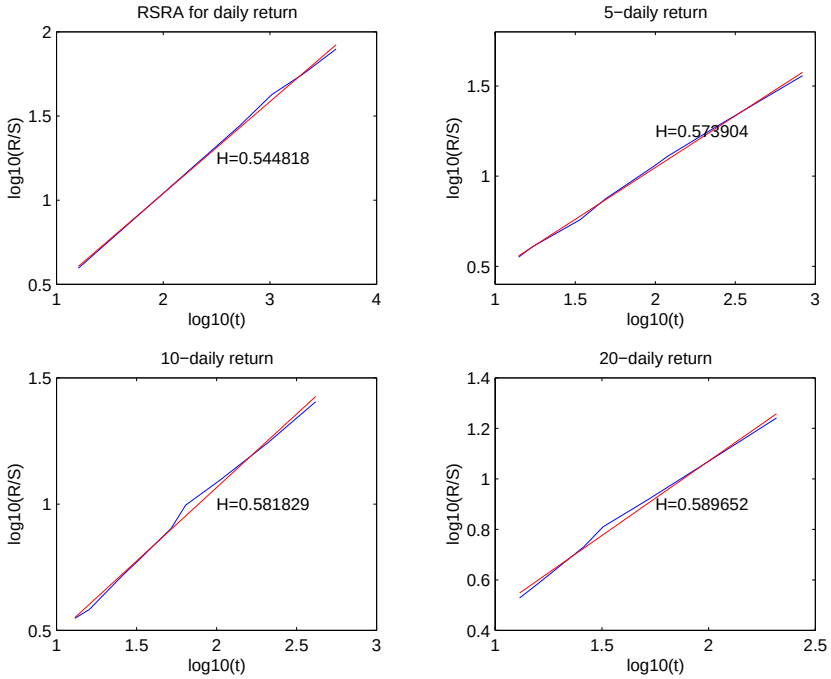


Figure 17.23: Rescaled Range Analysis results for the NYA 1960-89.

7. The length n is then increased until there are only two sub-periods, i.e. $n = N/2$. We then perform a least squares regression on $\log(n)$ as the independent variable and $\log(R/S)$ as the dependent variable. The slope of the equation is the estimate of the Hurst exponent, H.

The R/S analysis results for the NYA (1960-1998) for daily, 5-daily, 10-daily and 20-daily returns are shown in Figure 17.23. The Hurst exponent is $0.54 \leq H \leq 0.59$, from daily to 20-daily returns indicating that the data set is persistent, at least up to 1000 trading days. At this point the scaling behaviour appears to slow down. The $(R/S)_n$ values show a systematic deviation from the line of best fit which is plotted in the Figures. From the daily return results this appears to be at about 1000 days. The 5-daily, 10-day and 20-day return results appear to agree a value of about 630 days. This is also where the correlation function starts to increase. This deviation is more pronounced the lower the frequency with which the data is sampled. The results show that there are certain non-periodic cycles that persist for up to 1000 days which contribute to the persistence in the data, and after these are used up, the data (the walker) slows down. These observations can be summarized as follows: The market reacts to information, and the way it reacts is not very different from the way it reacts previously, even though the information is different. Therefore the underlying dynamics and the statistics of the market have not changed. This is especially true of fractal statistics. (The 'fractal statistics' referred to are the fractal dimension and the Hurst exponent.) The results clearly imply that there is an inconsistency between the

behaviour of real financial data and the EMH lognormal random walk model which is compounded in the following points:

1. The PSDF of log price changes is not constant. Therefore price changes are not independent.

2. The PDF of log price changes are not Gaussian, they have a sharper peak at the mean and longer tails.

In addition, the following properties are evident:

1. Asset price movements are self-affine, at least up to 1000 days.

2. The first moment of the PDF is finite but the second moment is infinite (or at least very slow to converge).

3. If stock market data is viewed as a random walk then the walk scales faster than the square root of the time up until approximately 1000 days and then slows down.

4. Large price movements tend to be followed by large movements and vice versa, although signs of increments are uncorrelated. Thus volatility comes in bursts. These cycles are referred to as non-periodic as they have randomly distributed periods.

Hurst devised a method for analysing data which does not require a Gaussian assumption about the underlying distribution and in particular, does not require the distribution to have finite moments. This method can reveal subtle scaling properties including non-periodic cycles in the data that spectral analysis alone cannot find.

17.14 Modelling Financial Data

In general, the unveiling of a new phenomenon either results from a strong theoretical reasoning or from compelling experimental evidence. In econometrics, the process that creates our time series has many component parts, or degrees of freedom, and the interaction of those components is so complex that a deterministic description is simply not possible. As in all complex systems theory, we restrict ourselves to modelling the statistics of data rather than the data itself. This means that we model the data with stochastic models.

When creating models of complex systems there is a trade off between simplifying and deriving the statistics we want to compare with reality, and simulating the behaviour and letting the statistics emerge, as they do in real life. It may be hard to learn from the latter approach (i.e. creating an artificial market) for the same reasons that it is hard to learn by looking at real markets. On the other hand, there is a danger of making incorrect assumptions and thus failing to capture certain emergent traits when adopting the former method. We need both approaches; at least for the time being. We need the simulation approach to investigate the affect of various traders' behavioural rules on the global statistics of the market; this approach has

the added benefit of a natural interpretation. We also need the derivation approach to understand exactly how the amalgamation of certain rules leads to these statistics.

So what makes a good stochastic model? A good stochastic model is one that accurately predicts the statistics we observe in reality, and one that is based upon some sensible rationale. Thus, the model should not only describe the data, but also help us to explain and understand the system. It can still, however, be advantageous to have an accurate and/or predictive model without understanding it's origins, i.e. a phenomenological model. Economists often feel insulted by a stochastic approach to financial modelling; they believe all of their decisions are rational and well founded. However, the Black-Scholes model, which has to date played such an important role (because it has no random variable), assumes a stochastic model of financial time series anyway, namely, the lognormal random walk.

One cause of correlations in market price changes (and volatility) is mimetic behaviour, known as herding. In general, market crashes happen when large numbers of agents place sell orders simultaneously creating an imbalance to the extent that market makers are unable to absorb the other side without lowering prices substantially. What is interesting is that most of these agents do not communicate with each other, nor do they take orders from a leader. In fact, most of the time they are in disagreement, and submit roughly the same amount of buy and sell orders. This is a healthy non-crash situation; it is a diffusive process. The key question is thus: by what mechanism do they suddenly manage to organise a coordinated sell-off? When constructing a comprehensive model of financial time series we need to make the following distinction regarding the nature of a crash:

1. Are crashes exceptions to the model? Do we need one model to describe stable behaviour and another for when the things become unstable? In which case, when should we switch? There should be some critical point; indeed, we would not be able to distinguish a crash from a negative price increment if there were not.

2. Can crashes be represented by a model with a (one) price increment PDF with an infinite second moment? This Lévy style method allows seemingly disproportionate large price movements to occur with (statistically) controllable frequency.

3. Are they critical points analogous to phase transitions in nature? If so, are there any warning signs prior to a crash?

4. Are they model-able at all? That is, do crashes occur because of unforeseeable circumstances like political news and news related to actual firms and businesses?

One explanation for crashes involves a replacement for the efficient market hypothesis, by the Fractal Market Hypothesis (FMH). The FMH proposes the following:

1. The market is stable when it consists of investors covering a large number of investment horizons. This ensures that there is ample liquidity for traders.

2. Information is more related to market sentiment and technical factors in the short term than in the long term. As investment horizons increase, longer term fundamental information dominates.

3. If an event occurs that puts the validity of fundamental information in question, long-term investors either withdraw completely or invest on shorter terms. When the overall investment horizon of the market shrinks to a uniform level, the market becomes unstable.

4. Prices reflect a combination of short-term technical and long-term fundamental valuation. Thus, short-term price movements are likely to be more volatile than long-term trades. Short-term trends are more likely to be the result of crowd behaviour.

5. If a security has no tie to the economic cycle, then there will be no long-term trend and short-term technical information will dominate.

Unlike the EMH, the FMH says that information is valued according to the investment horizon of the investor. Because the different investment horizons value information differently, the diffusion of information will also be uneven. Unlike most complex physical systems, the agents of the economy, and perhaps to some extent the economy itself, have an extra ingredient, an extra degree of complexity. This ingredient is consciousness.

17.14.1 Psychology and the Bear/Bull Cycle

An economy is ultimately driven by people (economic agents) and the principal component of this drive, is their expectation of others. When economic agent's expectations induce actions that aggregatively create a world that validates them as predictions, they are in equilibrium, and are called 'rational expectations'. Rational expectations are useful in demonstrating logical equilibrium outcomes and analysing their consequences. In the real world, however, they break down very easily. If some agents lack the resources to arrive at a posited outcome or if they logically arrive at different conclusions (as they might in a pattern recognition problem) or if there is more than one rational expectations equilibrium with no means to decide on which is preferable, then some agents may deviate in their expectations. Moreover, if some deviate, the world that is created may change so that others should logically predict something different and deviate too.

There have been various games made up to illustrate this problem. A well known example is the 'El Farol bar' problem: One hundred people must decide independently whether to show up at a particular bar. If a person expects that more than say 60 will attend, then they will stay at home to avoid the crowd; otherwise that person will attend. If all believe that few will go, then all will go; if all believe many will go, then no one will go. By simulating this game allowing agents access to historical data and by giving them a simple genetic algorithm to decide how to use this data, one can illustrate how stochastic/chaotic time series can emerge.

Trading crowds are made up of bulls and bears. A bullish crowd is one that will try to force prices up and a bearish crowd is one that will try to push prices down. The size of each crowd must remain roughly the same, but the members are continually swapping. When the price is going up, the bullish crowd are dominating and when the price is going down, the bearish crowd are dominating. Within a full cycle, the

bullish crowd will dominate, then there will be a period of co-existence, and finally the bearish crowd will dominate. This means that between any two given points in time there will be a successful crowd and an unsuccessful one. Members of the successful crowd will be motivated by greed and will feel pleasure from their success; they will feel integrated with like-minded investors. On the other hand, members of the unsuccessful crowd will feel fear and displeasure, and feel divorced from the other members of the same crowd. Eventually, the members of the unsuccessful crowd desert to the successful crowd and thus the cycle continues.

17.14.2 The Multi-Fractal Market Hypothesis

Crashes are the result of price corrections due to trend chasing strategies and/or external news. Markets are efficient but the EMH fails, mostly due to the break down of rational expectations and although psychology undoubtedly plays a role in market analysis, its extent is undetermined. The multi-fractal model is based on the fact that econometric data has fractal properties and is concerned with the variable *diffusion* of information. This is related to the fractional dynamic equation introduced earlier in this chapter, i.e.

$$\left(\frac{\partial^2}{\partial x^2} - \tau^{q(t)}\frac{\partial^{q(t)}}{\partial t^{q(t)}}\right) u(x,t) = F(x,t)$$

and its asymptotic solution $u(0,t)$ which is compounded in the following points:

1. The PDF of log market movements has an infinite or very slow-to-converge second moment.

2. The economy can be thought of as a non-equilibrium self-organised system that is normally and healthily in a critical state, and that crashes occur when somehow this criticality is temporarily disturbed.

3. Prices oscillate around fundamentals (due to the bear/bull cycle) which are themselves dictated by people's speculation on what they should be.

4. These oscillations propagate through supply/information chains.

5. The level of diffusion/propagation through the information chain varies with time.

Point (5) is the fundamental basis for the model above. By introducing a Fourier dimension that is a function of time $q(t)$ which is itself taken to be a stochastic function, we are able to model multi-fractal signals (signals with variations in q) in terms of both a quasi-diffusive and quasi-propagative behaviour.

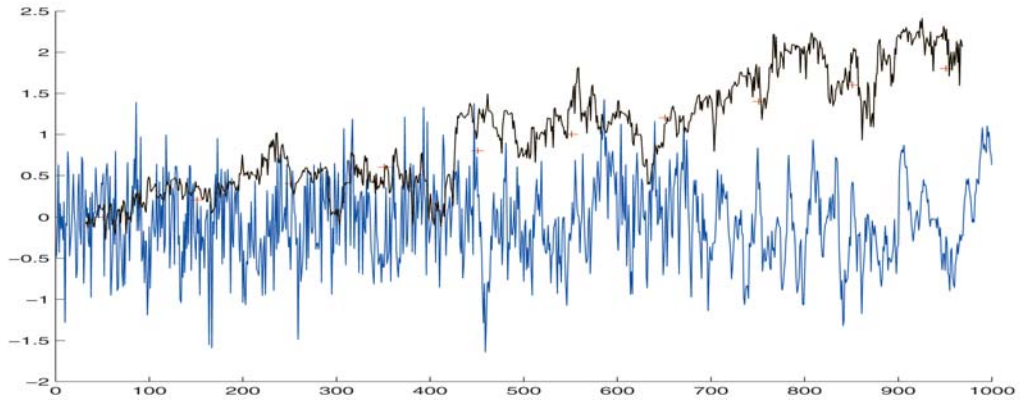


Figure 17.24: A realisation of u_i, with 10 different q values (indicated by $+$) in $q_i \in (0, 2)$ and reconstruction of q_i using the power spectrum method (darker line).

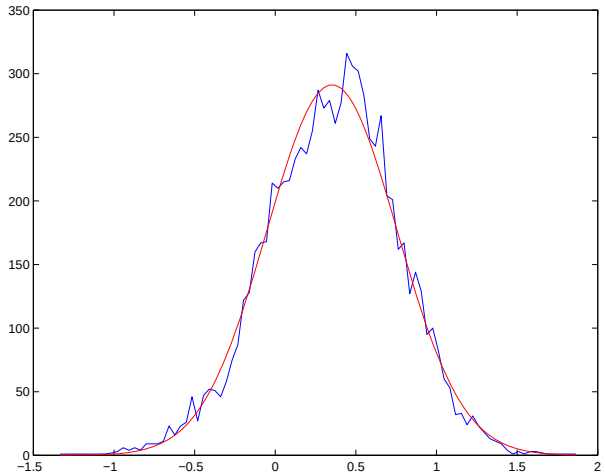


Figure 17.25: A Gaussian distribution (least squares) fit to the histogram of q values of the NYA.

Figure 17.24 shows how the value of q affects the characteristics of the signal. The signal is a concatenation of 10 smaller signals, each of 100 elements and a unique Fourier dimension, $q_i \in (0, 2)$. Each slice of the signal is scaled by dividing through by the sample variance of the signal. Without this scaling, signals with a high q value would have significantly less magnitude. The rationale here is that signals with the same Black-Scholes volatility can behave very differently - the sums of squares can

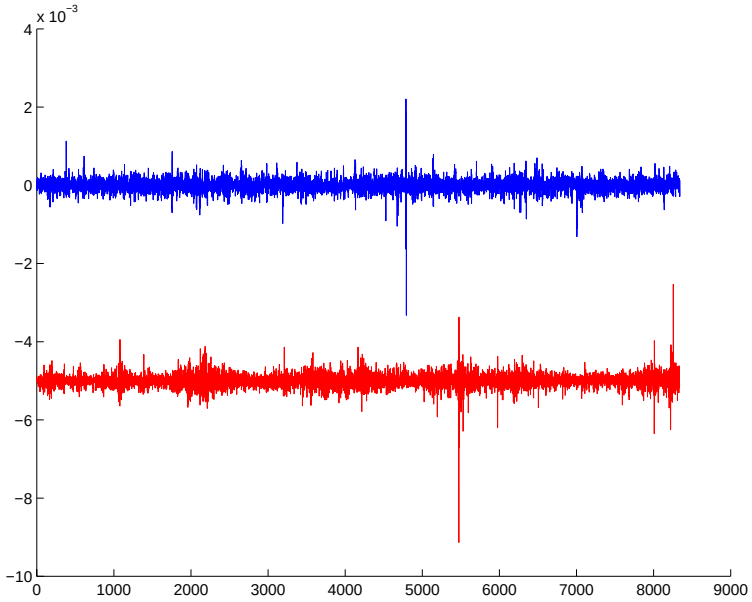


Figure 17.26: Synthetic stock market price moves (top) .v. the NYA (bottom) real data with mean displaced to -5 for comparative display purposes.

add up in many different ways. The idea is that the model will capture this different kind of volatility. The inversion algorithm (based on the power spectrum method discussed earlier in this chapter) estimates the q-values relatively accurately.

Given this model together with the simulation presented, the principal question is how does $q(t)$ behave for real data? There are two ways to approach this: (i) compute the Fourier dimension from the data u_i directly; (ii) compute the dimension from the de-trended log price increments, i.e. $\ln(u_{i+1}/u_i)$. Taking the latter case, q_i values have been evaluated over a sample of real data (NYA) using a window of size 30 and revealing the Fourier dimensions to be normally distributed with $\mu = 0.35$ and $\sigma = 0.4$, as shown in Figure 17.25. This is an interesting result; it shows that the deviations from the random walk model, as measured by the Fourier dimension, are governed by a Gaussian PDF. It also shows that on average there is much less high frequency content in the log price movements than the random walk model. It is opportune at this point to employ the 'joker effect' discussed earlier. Suppose we define trends with particular values of q. A problem here is that due to the necessity of a moving window method for computing q_i, we obtain a very smeared picture of how q is behaving on a day to day basis - we cannot measure run lengths of q values. Thus, we ideally need higher frequency data to successfully calculate the day-to-day behaviour of q. We do know, however, that there exists some probability, $p(x_i > \bar{x} \mid x_{i-1} > \bar{x}) > 0.5$ that accounts for the persistence in the data. Since q controls these correlations we can postulate a distribution for the length of stay for a particular value of q based on $\Pr(r) = p^r(1-p)$ as described earlier.

Figure 17.26 shows synthesized stock market price movements with normally distributed Fourier dimensions that persist for random periods defined by the aforementioned binomial style distribution with $\bar{R} = 1.3$ and $p = 0.565$. The market movements of the NYA are shown for comparison. The average run length of the synthetic signal is approximately the same as the NYA daily returns, as is the Hurst exponent and the distribution of Fourier dimensions. The PDF and the power law property of the PSDF as shown in Figure 17.27. Comparing these results with Figure 17.17, the PSDF and ACF are similar, although the ACF of the synthesized data does not capture the increase in correlations observed in the real absolute log price movements.

Clearly, compared with the EMH, these results - based on a Multi-fractal Market Hypothesis - are encouraging and lead naturally to consider the non-stationary behaviour of the Fourier dimension for a financial index and a possible association of a particular range of q_i with the behaviour of the index. Figure 17.28 shows the behaviour of the Fourier dimension (the q-signature) for the NYA at the time of the crash of October 1987. Here, the q-signature has been superimposed on the log price increments or price moves. Observe, that q is almost zero over the period of the crash with relative small deviations, indicating that during periods of exceptionally high volatility the price moves behave more like white noise than when the volatility is low; the value of q_i is below the mean. Also, the variation of the q-signature is minimal over this period. Most important of all, is that it has a downward trend prior to the event.

Discussion

The material discussed has been based on the asymptotic solution to the fractional dynamic model proposed for price variation. In accordance with the definitions of the model, this is equivalent to observing market price movements very close (in terms of connectivity) to the source of information affecting them. The general solution suggests that prices further 'down the line' are of the same form but with additional amounts of higher frequency noise. Thus, although the economy is totally connected (to maintain consistency), noise can induce price movements that are far apart in a chain that is virtually unconnected. The proposed model for price variation accurately mimics the PDF, the power law property of the PSDF, the Hurst exponent and the 'runs' statistic observed in real financial markets. A new market measure, q-signature, describing the 'diffusivity' of new information is the core of the model. Crashes happen when price increments become uncorrelated. The parameter q-signature can be estimated from market data with a 'least squares fit' thus providing a new risk analysis tool.

There are some properties of the financial time series, however, that the model does not capture. The first over-sight of the model is the long-range time correlations that occur over approximately 1000 days, or 4 years. As Figure 17.27 illustrates, the model gets the right power law for the PSDF, but does not capture the strange increase in the correlation function at about 600 days in the real data as shown in Figure 17.17. Further, the model does not capture the fact that market price movements only scale persistently up to 1000 days, the synthetic data scales persistently indefinitely. An interesting extension of the model would be to apply spatial reflecting bounds to the

field $u(x,t)$ and numerically evaluate the solution to the model. In this way we would not need to drive the field with noise at each time interval, the movements would continue indefinitely. However, the model does provide clarity with regard to the following important points:

1. Market data is not normally distributed but its q-signature or variable Fourier dimension is.

2. By filtering the Fourier spectrum of white noise (as described) over finite intervals (thus causing trends) and allowing them to persist for random periods of time (joker effect), one can create a synthesized financial time series possessing many of the right statistics.

3. Price movements are better described by the conventional (Gaussian) random walk models when the volatility is very high.

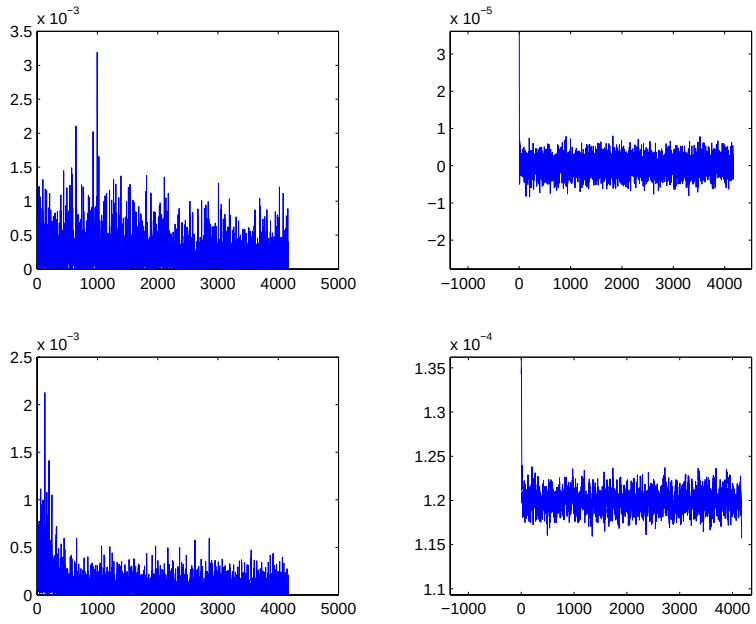


Figure 17.27: PSDF (left) and ACF (right) of synthetic price increments (top) and absolute price increments (bottom).

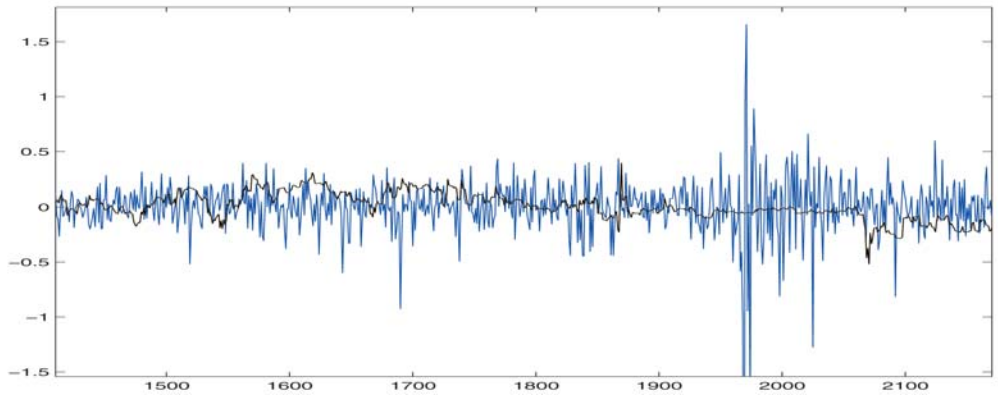


Figure 17.28: q-signature (darker line) of NYA price movements (lighter line) around the time of the 1987 crash.

17.15 Summary of Important Results

Universal Power Law

$$\text{System(size)} \propto \frac{1}{\text{size}^q}, \quad q > 0$$

Random Fractal Signal Model

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{N(\omega)}{(i\omega)^q}\exp(i\omega t)d\omega = \frac{1}{\Gamma(q)}\int_0^t \frac{n(\tau)}{(t-\tau)^{1-q}}d\tau, q > 0$$

where $n(t)$ is white noise whose Fourier transform is $N(\omega)$.

Power Spectrum of a Random Fractal

$$P(\omega) = \frac{A}{\mid \omega \mid^{2q}}$$

where A is a constant.

Fractal Dimension D of a Random Fractal Signal

$$D = \frac{5-2q}{2}, \quad 1 < D < 2$$

Lévy Distributions

Distributions with a characteristic function given by (symmetric case)

$$P(k) = \exp(-a \mid k \mid^q), \quad 0 < q < 2$$

and a PDF given by

$$p(x) \sim \frac{1}{x^{1+q}}, \quad \mid x \mid >> 1$$

Non-stationary Fractal Model

$$\left(\frac{\partial^2}{\partial x^2} - \tau^{q(t)} \frac{\partial^{q(t)}}{\partial t^{q(t)}} \right) u(x,t) = F(x,t)$$

where $F(x,t)$ is a stochastic source function and $-\infty < q(t) < \infty$ is a random variable, the Fourier dimension.

General Solution

For $F(x,t) = f(x)n(t)$,

$$\begin{aligned} u(x_0,t) &= \frac{M_0(x_0)}{2\tau^{q/2}} \frac{1}{\Gamma(q/2)} \int_0^t \frac{n(\xi)}{(t-\xi)^{1-(q/2)}} d\xi - \frac{M_1(x_0)}{2} n(t) + \\ & \frac{1}{2} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{(k+1)!} M_{k+1}(x_0) \tau^{kq/2} \frac{d^{kq/2}}{dt^{kq/2}} n(t), \quad q > 0; \end{aligned}$$

$$u(x_0,t) = \frac{M(x_0)}{2} n(t), \quad q = 0;$$

$$\begin{aligned} u(x_0,t) &= \frac{M_0(x_0)\tau^{q/2}}{2} \frac{d^{q/2}}{dt^{q/2}} n(t) - \frac{M_1(x_0)}{2} n(t) + \\ & \frac{1}{2} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{(k+1)!} \frac{M_{k+1}(x_0)}{\tau^{kq/2}} \frac{1}{\Gamma(kq/2)} \int_0^t \frac{n(\xi)}{(t-\xi)^{1-(kq/2)}} d\xi \quad , q < 0; \end{aligned}$$

where

$$M_m(x_0) = \int_{-\infty}^{\infty} f(x) \mid x - x_0 \mid^m dx.$$

Asymptotic Solution

For $f(x) = \delta(x)$,

$$u(t) = \lim_{x_0 \to 0} u(x_0,t) = \begin{cases} \frac{1}{2\tau^{q/2}} \frac{1}{\Gamma(q/2)} \int_0^t \frac{n(\xi)}{(t-\xi)^{1-(q/2)}} d\xi, & q > 0; \\ \frac{n(t)}{2}, & q = 0; \\ \frac{\tau^{q/2}}{2} \frac{d^{q/2}}{dt^{q/2}} n(t), & q < 0. \end{cases}$$

17.16 Further Reading

- Oldham K B and Spanier J, *The Fractional Calculus*, Academic Press, 1974.
- Mandelbrot B B, *The Fractal Geometry of Nature*, Freeman, 1983.
- Barnsley M F and Demko S G (Eds.), *Chaotic Dynamics and Fractals*, Academic Press, 1986.
- E R Pike and L A Lugiato (Eds.), *Chaos, Noise and Fractals*, IoP Publishing (Malvern Physics Series), 1987.
- Peitgen H O, Jürgens H and Saupe D, *Chaos and Fractals: New Frontiers of Science*, Springer, 1992.
- Miller K S and Ross B, *An Introduction to the Fractional Calculus and Fractional Differential Equations*, Wiley, 1993.
- Kiryakova V, *Generalized Fractional Calculus and Applications*, Longman, 1994.
- Peters E E, *Fractal Market Analysis*, Wiley, 1994.
- Turner M J, Blackledge J M and Andrews P A, *Fractals Geometry in Digital Imaging*, Academic Press, 1998.
- Blackledge J M, Evans A K and Turner M J (Eds.), *Fractal Geometry: Mathematical Methods, Algorithms and Applications*, Horwood Publishing Series in Mathematics and Applications, 2002.
- Barnsley M F, Saupe D and Vrscay E R, *Fractals in Multimedia*, Springer (The IMA Volumes in Mathematics and its Applications), 2002.

17.17 Problems

17.1 Design a void function in ANSI C to synthesize a random fractal signal s with an array size of n and fractal dimension $1 < D < 2$:

```
void RFS(float s [ ], int n, float D)
```

Use uniform random deviates to compute the Fourier phase and Gaussian random deviate to compute the Fourier amplitude of the fractal using the functions discussed in Chapter 14.

17.2 Design a function using ANSI C that returns the fractal dimension of a signal s of size n under the assumption that the signal is a time invariant random fractal using the power spectrum method:

```
function FD(float s[ ], int n)
```

17.3 Using the result that

$$\int_{\alpha}^{\beta} \exp(ikx)(x-\alpha)^{\lambda-1}\phi(x)dx = -A_N(k)$$

where ϕ is N times differentiable and

$$A_N(k) = \sum_{n=0}^{N-1} \frac{\Gamma(n+\lambda)}{n!} e^{i\pi(n+\lambda-2)/2}\phi^{(n)}(\alpha)k^{-n-\lambda}\exp(ik\alpha)$$

show that

$$\int_{-\infty}^{\infty} \exp(ikx)\exp(-\mid k \mid^q)dk \sim \frac{1}{x^{1+q}}$$

for $\mid x \mid >> 1$ by taking the case when $0 < q < 1$ and $1 < q < 2$ and using integration by parts.

17.4 Show that the maximum value of the PSDF given by

$$P(\omega) = \frac{\mid \omega \mid^g}{(\omega_0^2 + \omega^2)^q}$$

is

$$P_{\max} \equiv P(\omega_{\max}) = \omega_0^{2(g-q)}\frac{g^g}{q^q}(q-g)^{q-g}.$$

Derive an expression for the output signal $s(t)$ obtained when the input is white noise when a system whose PSDF is given by the above is linear and time invariant. What is the scaling relationship for this case?

Summary

This book has been designed to provide the reader with a comprehensive account of the mathematical models, computational techniques and a programming approach to Digital Signal Processing. Many of the numerical methods required for DSP form part of the 'art' of numerical computing in general. However, there are specific techniques which focus on DSP alone. Many of these relate to the discretization of certain integral transforms and the design of fast algorithms to transform a digital signal into some transformation space (where useful operations and analysis can be performed) and back again.

A central and reoccurring theme is the signal model equation

$$s(t) = p(t) \otimes f(t) + n(t)$$

and the following inverse problem: given digital data on $s(t)$, $p(t)$ and $\Pr[n(t)]$, find an estimate for $f(t)$. A number of different approaches to solving this problem have been presented ranging from the application of the least squares method (the Wiener filter) for example to the use of Bayesian statistics which provides methods for incorporating knowledge on the probability density functions of $f(t)$ and $n(t)$, i.e. $\Pr[f(t)]$ and $\Pr[n(t)]$ respectively.

The convolution equation given above is a fundamental model. It can be used to model both stationary and non-stationary systems, causal and non-causal problems, an important and fundamental difference being, that in the latter case (the non-stationary case), there is no equivalent convolution theorem. In both cases, and in digital form, where the functions $f(t), p(t)$ and $s(t)$ are the vectors $\mathbf{f}, \mathbf{p}$ and $\mathbf{s}$ respectively, we can write the operation $s = p \otimes f$ as

$$P\mathbf{f} = \mathbf{s}$$

where P is the characteristic matrix formed from $\mathbf{p}$. This yields a linear system of algebraic equations whose solution is required in the case when the system is inhomogeneous, i.e.

$$\mathbf{s} = P\mathbf{f} + \mathbf{n}.$$

There are many ways to derive this convolution model depending on the 'physics' of the signal generating system. The convolution operation emerges in a number of diverse areas: in physics, biology, statistics, information theory, etc. In information theory for example, the underlying principle is that any source of information (man-made or otherwise) tends to become 'diffused' through the act of measuring it with some instrument, an instrument that is inevitably subject to distortion by noise,

the process of 'diffusion' being compounded in the convolution of the information input with the instrument function (the impulse response function). In encryption for example, the goal is to maximize the level of diffusion and distortion or complexity (i.e. to maximize the entropy) of the information input in such a way that only the recipient of a transmitted message can recover the input.

Convolution is the basic operation generated by using a Green's function to solve problems that are concerned with the 'physics' of the propagation and interaction of wavefields with matter. This is usually specified by some inhomogeneous wave equation whose most general solution involves a convolution with the appropriate Green's function. There are a wide range of signal generating systems that are based on detecting wavefields using an appropriate instrument in both passive and active modes. All such systems are related, in one form or another, to the 'physics' of waves and vibrations and through the Green's function, to the convolution integral and convolution-type operations. This is how we are able to connect the physics of a system to a model that many electrical engineers for example may tend to take for granted.

Another feature that has been a dominant theme of this book is the modelling of noise $n(t)$. The physical processes by which noise is generated are variable and wide ranging. Noise can sometimes be interpreted in terms of the multiple interactions that occur in pulse-echo type imaging systems, effects that can be minimized by averaging over many recordings to give a good approximation to the primary field. Electronic noise is present in all types of instrumentation and detectors and can be a dominant feature in highly sensitive instrumentation. A special type of noise known as 'speckle' is a dominant feature of imaging systems that record coherent signals such as radar. Noise can also feature in terms of spurious results that are the result of the geometry of a system being inconsistent with a one-dimensional model.

The statistical modelling of noise and the analysis of signals using these 'statistics' is fundamental to DSP. In developing an interpretation of a signal, it is often useful to attempt to design some deterministic model if possible. However, as a system becomes more and more complex, determinism often has to be sacrificed in place of the development of appropriate stochastic models from fundamental (i.e. random walk type) principles. An emergent feature of such models involves the application of fractal geometry for interpreting random signals that exhibit self-affine statistics, a feature that is common in an increasingly wide range signals. Statistical signal analysis is based on the interpretation of a signal in terms of a set of statistical parameters using the moving window principle discussed in Chapter 16. The fractal dimension discussed in Chapter 17 can be considered to form part of such a set.

In addition to the theoretical aspects of DSP, this book has attempted to help the reader design appropriate software and also to become acquainted with the numerical methods that from the basis for the design of such software. There is often a large 'gap' between the theoretical aspects of a subject and the engineering solutions. Filling this 'gap' is often the most difficult part of the 'learning curve' and requires the engineer to come to terms with a diverse range of subjects. DSP requires a unique blend of physics, mathematics, statistics, computing and electronics and a book of this type can only ever provide a brief glimpse of the way in which these subjects combine to solve problems in DSP. Thus, many important new approaches, transformation methods, filters, etc., have been omitted or not covered in sufficient detail, in particular, the

applications of important subjects such as fuzzy logic and artificial neural networks to DSP. However, it is hoped that the reader will be able to approach new aspects of this subject and develop new and original ideas with a greater degree of understanding and confidence as a result of studying this book. If so, then its composition and publication have been worth while. Further, the principles of signal processing can be extended to the analysis and processing of digital images. This is the subject of a companion volume entitled *Imaging and Image Processing* by the author, which covers aspects of electromagnetic and acoustic field theory for modelling optical and acoustic images and investigates conventional and novel methods of image processing and pattern recognition.

Appendix A

Solutions to Problems

A.1 Part I

A.1.1 Solutions to Problems Given in Chapter 1

1.1

(i)

$$\frac{1+i}{2-i} = \left(\frac{1+i}{2-i}\right)\left(\frac{2+i}{2+i}\right) = \frac{1}{5}(1+3i) = \frac{1}{5} + \frac{3}{5}i.$$

(ii)

$$\frac{1}{i^5} = \frac{1}{i^5}\frac{i}{i} = \frac{i}{i^6} = \frac{i}{(\sqrt{-1})^6} = \frac{i}{(-1)^3} = -i.$$

(iii)

$$(2-3i)^2 = -5 - 12i.$$

Thus,

$$\frac{4-5i}{(2-3i)^2} = \frac{5i-4}{5+12i} = \left(\frac{5i-4}{5+12i}\right)\left(\frac{5-12i}{5-12i}\right) = \frac{40}{160} + i\frac{73}{169}.$$

(iv)

$$\frac{(1+2i)(1+3i)(3+i)}{1-3i} = \frac{(1+2i)(1+3i)(3+i)(1+3i)}{(1-3i)(1+3i)} = \frac{(5i-5)}{10}10i = -5 - 5i.$$

(v)

$$\left(-\frac{1}{2} + i\frac{\sqrt{3}}{2}\right)\left(-\frac{1}{2} + i\frac{\sqrt{3}}{2}\right) = \frac{1}{4} - i\frac{\sqrt{3}}{4} - i\frac{\sqrt{3}}{4} - \frac{3}{4} = -\frac{1}{2} - i\frac{\sqrt{3}}{2}.$$

1.2 In each case, consider the real part to be a number along the real (horizontal) axis and the imaginary part to along the imaginary (vertical axis). The complex

number forms a point with coordinates (x, iy) in the same way that a value of a two dimensional function forms a point on an (x, y) plane.

1.3

$$(z_1 + z_2)^* = [(x_1 + iy_1) + (x_2 + iy_2)]^*$$
$$= [(x_1 - iy_1) + (x_2 - iy_2)] = [(x_1 + iy_1)^* + (x_2 + iy_2)^*] = z_1^* + z_2^*.$$

$$(z_1 z_2)^* = [(x_1 + iy_1)(x_2 + iy_2)]^*$$
$$= [(x_1 - iy_1)(x_2 - iy_2)] = [(x_1 + iy_1)^*(x_2 + iy_2)^*] = z_1^* z_2^*.$$

Note, that in general

$$\left(\sum_i z_i\right)^* = \sum_i z_i^*; \quad \left(\prod_i z_i\right)^* = \prod_i z_i^*.$$

1.4 Note, that $z = a + ib = Ae^{i\theta}$ where $A = \sqrt{a^2 + b^2} \equiv \text{mod}z$ or $\mid z \mid$, $\theta = \tan^{-1}(b/a) \equiv \arg z$.

(i) $z = 1 - i3; \mid z \mid = 4, \arg z = \tan^{-1}(-\sqrt{3}/2) = -\pi/3$.

(ii) $e^{i\pi/2} + \sqrt{2}e^{i\pi/4} = \cos(\pi/2) + i\sin(\pi/2) + \sqrt{2}\cos(\pi/4) + i\sqrt{2}\sin(\pi/4) = i + \sqrt{2}\frac{1}{\sqrt{2}} + i\frac{\sqrt{2}}{\sqrt{2}} = i + 1 + i = 1 + 2i$. Hence $\mid z \mid = \sqrt{5}$ and $\arg z = \tan^{-1} 2$.

(iii) Noting that $e^{i\pi/4} = \cos(\pi/4) + i\sin(\pi/4) = 1/\sqrt{2} + i/\sqrt{2}$, we can write $1 + i = \sqrt{2}e^{i\pi/4}$. Hence, $(1+i)e^{i\pi/6} = \sqrt{2}e^{i\pi/4}\mathbf{e}^{i\pi/6} = \sqrt{2}e^{i5\pi/12}$ and $\mid z \mid = \sqrt{2}, \arg z = 5\pi/12$.

(iv) $z_1 z_2 = 2e^{i\pi/5}3e^{i\pi/3} = 6e^{i8\pi/15}$. Hence $\mid z_1 z_2 \mid = 6, \quad \arg(z_1 z_2) = 8\pi/15$.

(v) $z_1/z_2 = (2e^{i\pi/5})/(3e^{i\pi/3)} = (2/3)e^{i\pi/5}e^{-\pi/3} = (2/3)e^{-i2\pi/15}$. Thus, $\mid z_1/z_2 \mid = 2/3, \arg(z_1/z_2) = -2\pi/15$.

1.5 (i) Noting that, from De Moivre's theorem,

$$\exp(i\pi) = \cos(\pi) + i\sin(\pi) = -1$$

the result (Eulers equation) follows. Note that the equation $\exp(i\pi) + 1 = 0$ is like no other equation in mathematics because it provides a link between the most important numbers and constants in all mathematics, i.e. 0, 1, e and π!

(ii) Noting that

$$\exp(i\pi/2) = \cos(\pi/2) + i\sin(\pi/2) = i$$

the result follows.

(iii)

$$i^i = [\exp(i\pi/2)]^i = \exp(i^2\pi/2) = \exp(-\pi/2) = 1/\sqrt{e^\pi}$$

(iv)

$$i^{1/i} = [\exp(i\pi/2)]^{1/i} = \exp(\pi/2) = \sqrt{e^\pi}$$

(v) The result follows from (iii) and (iv) above.

1.6

$$C + iS = \int e^{ax}\cos(bx)dx + i\int e^{ax}\sin(bx)dx = \int e^{ax}[\cos(bx) + i\sin(bx)]dx$$

$$= \int e^{ax}e^{ibx}dx = \int e^{(a+ib)x}dx = \frac{e^{(a+ib)x}}{a+ib}$$

ignoring the constant of integration. We now write the result in terms of real and imaginary parts, thus:

$$C + iS = \frac{a-ib}{a^2+b^2}e^{ax}[\cos(bx) + i\sin(bx)]$$

$$= \frac{e^{ax}}{a^2+b^2}[a\cos(bx) + b\sin(bx)] + i\frac{e^{ax}}{a^2+b^2}[a\sin(bx) - b\cos(bx)].$$

Hence,

$$C = \int e^{ax}\cos(bx)dx = \frac{a^{ax}}{a^2+b^2}[a\cos(bx) + b\sin(bx)]$$

and

$$S = \int e^{ax}\sin(bx)dx = \frac{a^{ax}}{a^2+b^2}[a\sin(bx) - b\cos(bx)].$$

1.7 Equating real and imaginary parts of the equation

$$r\exp(i\theta) - 1 = R\exp(i\alpha),$$

we have

$$r\cos\theta - 1 = R\cos\alpha$$

and

$$r\sin\theta = R\sin\alpha.$$

Thus,

$$\sin\alpha = \frac{r\sin\theta}{R} \Longrightarrow \cos\alpha = \frac{\sqrt{R^2 - r^2\sin^2\theta}}{R}$$

and therefore

$$r\cos\theta - 1 = \frac{\sqrt{R^2 - r^2\sin^2\theta}}{R} \quad .$$

Taking squares,

$$r^2\cos^2\theta - 2r\cos\theta + 1 = R^2 - r^2\sin^2\theta$$

and solving for R gives

$$R = \sqrt{1 + r^2 - 2r\cos\theta} \quad .$$

Hence,

$$\mathrm{Re}[\ln(z-1)] = \ln R = \frac{1}{2}\ln(1 - r^2 - 2r\cos\theta).$$

1.8 Consider a rectangle with a path composed of four elements: Γ_1 along the real axis from $x = -R$ to $x = R$, Γ_2 from $x = R$ to $R + ia/2$, Γ_3 from $R + ia/2$ through $ia/2$ to $-R + ia/2$ and finally Γ_4 from $-R + ia/2$ to $x = -R$. From Cauchy's theorem

$$\oint_C e^{-z^2} dz = \int_{\Gamma_1} e^{-z^2} dz + \int_{\Gamma_2} e^{-z^2} dz + \int_{\Gamma_3} e^{z^2} dz + \int_{\Gamma_4} e^{z^2} dz = 0.$$

Hence,

$$\int_{-R}^{R} e^{-x^2} dx + \int_{0}^{a/2} e^{-(R+iy)^2} i dy + \int_{R}^{-R} e^{-(x+ia/2)^2} dx + \int_{a/2}^{0} e^{-(-R+iy)^2} i dy = 0$$

or

$$\int_{-R}^{R} e^{-x^2} dx + \int_{0}^{a/2} e^{-R^2} e^{2iyR} e^{y^2} i dy - \int_{-R}^{R} e^{-x^2} e^{-iax} e^{a^2/4} dx - \int_{0}^{a/2} e^{-R^2} e^{2iRy} e^{y^2} i dy = 0.$$

Now, as $R \to \infty$, $e^{-R^2} \to 0$. Thus,

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \int_{-\infty}^{\infty} e^{-x^2} e^{-iax} a^{a^2/4} dx = e^{a^2/4} \left(\int_{-\infty}^{\infty} e^{-x^2} \cos(ax) dx - i \int_{-\infty}^{\infty} e^{-x^2} \sin(ax) dx \right).$$

Equating real and imaginary parts, we get

$$e^{a^2/4} \int_{-\infty}^{\infty} e^{-x^2} \cos(ax) dx = \int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

and

$$e^{a^2/4} \int_{-\infty}^{\infty} e^{-x^2} \sin(ax) dx = 0.$$

Hence,

$$\int_{0}^{\infty} e^{-x^2} \cos(ax) dx = \frac{\sqrt{\pi}}{2} e^{-a^2/4}.$$

1.9 (i) For a path described by $x^2 + 2y^2 = 4$, the contour C is an ellipse which pases through the point $(-2, 2)$ on the real axis and $(-i\sqrt{2}, i\sqrt{2})$ on the imaginary axis. Now,

$$\oint_C \frac{z^3 - z + 2}{z - 1} = \oint_C \left[\frac{z(z^2 - 1)}{z - 1} + \frac{2}{z - 1} \right] dz = \oint_C \left[\frac{z(z - 1)(z + 1)}{z - 1} + \frac{2}{z - 1} \right] dz$$

$$= \oint_C z^2 dz + \oint_C z dz + 2 \oint_C \frac{1}{z - 1} dz = 0 + 0 + 2(2\pi i) = 4\pi i$$

since the last integral has a simple pole at 1 which is inside the contour. For the case when the contour C is a circle described by the equation $x^2+y^2=1/\sqrt{2}$, the pole is outside the contour and therefore the contour integral is zero.

(ii)

$$\oint_C \frac{3z^2-2z+1}{(z^2+1)(z-1)}dz = \oint_C \left[\frac{1}{z+i}+\frac{1}{z-i}+\frac{1}{z-1}\right]dz = \oint_C \frac{dz}{z+i}+\oint_C \frac{dz}{z-i}+\oint_C \frac{dz}{z-1}.$$

The contour described by the circle $x^2+y^2=4$ has points $(-2,2)$ on the real axis and $(-2i,2i)$ on the imaginary axis. Now the contour integral above has simple poles at $z=-i, z=i$ and $z=1$ which are all within the contour C. Thus, each of the three contour integrals above is given by $2\pi i$ and the contour integral given is therefore given by $6\pi i$.

1.10 Consider a semi-circle made-up of two elements, a path C_1 along the real axis from $-R$ to R and a circumference C_2 with a radius R going through the point iR on the imaginary axis. Then

$$\oint_{C_1+C_2} \frac{dz}{z^2+2z+2} = \int_{C_1} \frac{dz}{z^2+2z+2}+\int_{C_2} \frac{dz}{z^2+2z+2} = \int_{-R}^{R} \frac{dx}{x^2+2x+2}+\int_{C_2} \frac{dz}{z^2+2z+2}.$$

Now

$$\int_{C_2} \frac{dz}{z^2+2z+2} = \int_0^{\pi} \frac{Rie^{i\theta}d\theta}{R^2e^{2i\theta}+2Re^{i\theta}+2} \to 0 \quad \text{as} \quad R\to\infty$$

and therefore

$$\int_{-\infty}^{\infty} \frac{dx}{x^2+2x+2} = \oint_C \frac{dz}{z^2+2z+2}$$

where C is a large semi-circle, and has simple poles at $z=-1\pm i$. One pole at $z=-1+i$ exists inside the contour described above and has a residue given by

$$\lim_{z\to -1+i}\left[\frac{1}{z-(-1-i)}\right] = \frac{1}{2i}.$$

Thus, by Cauchy's residue theorem

$$\oint = 2\pi i \times \frac{1}{2i} = \pi$$

and

$$\int_{-\infty}^{\infty} \frac{dx}{x^2+2x+2} = \pi.$$

1.11 Let $z=e^{i\theta}$, then $dz = ie^{i\theta}d\theta = izd\theta$ and $d\theta = dz/iz$. Also

$$\sin\theta = \frac{e^{i\theta}-e^{-\theta}}{2i} = \frac{1}{2i}\left(z-\frac{1}{z}\right)$$

and hence,

$$\int_0^{2\pi} \frac{d\theta}{2+\sin\theta} = \oint_C \frac{dz/iz}{2+(z-1/z)/2i} = \oint \frac{2dz}{z^2+4iz-1}.$$

Simple poles exist at $i(-2\pm\sqrt{3})$ but only the pole at $-2i+i\sqrt{3}$ exist inside a contour which can be taken to be a circle in the complex plane with a radius of 1. The residue at $z = i(-2+\sqrt{3})$ is given by

$$\lim_{z\to -2i+i\sqrt{3}} \left(\frac{1}{z+2i+i\sqrt{3}} \right) = \frac{1}{i2\sqrt{3}}.$$

Hence, by Cauchy's theorem, the integral is

$$2\times 2\pi i \times \frac{1}{i2\sqrt{3}} = \frac{2\pi}{\sqrt{3}}.$$

1.12 Consider the function $e^{iz}/(1+z^2)$ and a contour composed of a semi-circle of radius R with an element C_1 from $-R$ to R along the real axis and a circumference C_2 passing through the point iR on the imaginary axis. Then,

$$\oint_C \frac{e^{iz}}{1+z^2}dz = \int_{C_1} \frac{e^{iz}}{1+z^2}dz + \int_{C2} \frac{e^{iz}}{1+z^2}dz$$

$$= \int_{-R}^{R} \frac{e^{ix}}{1+x^2}dx + \int_0^{\pi} \frac{\exp(iRe^{i\theta})}{1+R^2e^{2i\theta}} Rie^{i\theta}d\theta.$$

Now, the second (complex) integral on the RHS above goes to zero as $R\to\infty$ and thus, we can write

$$\int_{-\infty}^{\infty} \frac{e^{ix}}{1+x^2}dx = \oint_C \frac{e^{iz}}{1+z^2}dz$$

where simple poles exist at $z=\pm i$. Only the pole at $z=i$ is inside the contour described above and the residue at $z=i$ is given by

$$\lim_{z\to i} \left(\frac{e^{iz}}{z+i} \right) = \frac{e^{-1}}{2i}.$$

Hence, by Cauchy's residue theorem, the integral is

$$2\pi i \times \frac{e^{-1}}{2i} = \frac{\pi}{e} = \int_{-\infty}^{\infty} \frac{\cos x + i\sin x}{1+x^2}dx$$

whose imaginary component is zero, leaving the result that

$$\int_{-\infty}^{\infty} \frac{\cos x dx}{1+x^2} = \frac{\pi}{e}.$$

A.1.2 Solutions to Problems Given in Chapter 2

General remark: In proving relationships that involve the δ-function, it must be understood that the relationships only have a proper 'meaning' when expressed as an integrand in the basic sampling property of the δ-function, i.e.

$$\int_{-\infty}^{\infty} f(t)\delta(t-a)dt = f(a).$$

2.1

$$\int_{-\infty}^{\infty} f(t)\delta(t-a)dt = f(a)$$

and

$$\int_{-\infty}^{\infty} f(a)\delta(t-a)dt = f(a)\int_{-\infty}^{\infty}\delta(t-a)dt = f(a)$$

by the normalization condition

$$\int_{-\infty}^{\infty}\delta(t-a) = 1.$$

Hence,

$$f(t)\delta(t-a) = f(a)\delta(t-a).$$

2.2

$$\int_{-\infty}^{\infty} t\delta(t)f(t)dt = [tf(t)]_{t=0} = 0 \equiv \int_{-\infty}^{\infty} 0f(t)dt$$

$$\therefore\ t\delta(t) = 0.$$

2.3

$$\int_{-\infty}^{\infty}\delta(a-t)f(t)dt = \int_{\infty}^{-\infty}\delta[\tau-(-a)]f(-\tau)(-d\tau)$$

where $\tau = -t$. Hence,

$$\int_{-\infty}^{\infty}\delta(a-t)dt = \int_{-\infty}^{\infty}\delta[\tau-(-a)]f(-\tau)d\tau = f[-(-a)] = \int_{-\infty}^{\infty}\delta(t-a)f(t)dt.$$

$$\therefore\ \delta(a-t) = \delta(t-a).$$

From the above result, we note that with $a = 0$, we get $\delta(-t) = \delta(t)$ and so the δ-function behaves as an even function.

2.4 Let

$$I = \int_{-\infty}^{\infty} \delta(at) f(t) dt.$$

Consider the case when $a > 0$; $y = \mid a \mid t$, $d\tau = \mid a \mid dt$ and

$$I = \frac{1}{\mid a \mid} \int_{-\infty}^{\infty} \delta(\tau) f\left(\frac{\tau}{\mid a \mid}\right) d\tau = \frac{1}{\mid a \mid} f(0) = \int_{-\infty}^{\infty} \left[\frac{1}{\mid a \mid} \delta(t)\right] f(t) dt.$$

$$\therefore \quad \delta(at) = \frac{1}{\mid a \mid} \delta(t), \quad a > 0, \quad a \neq 0$$

Now consider the case when $a < 0; \tau = - \mid a \mid t, d\tau = - \mid a \mid dt$ so that

$$I = \frac{1}{\mid a \mid} \int_{\infty}^{-\infty} \delta(\tau) f\left(-\frac{\tau}{\mid a \mid}\right) (-d\tau) = \frac{1}{\mid a \mid} \int_{-\infty}^{\infty} \delta(\tau) f\left(-\frac{\tau}{\mid a \mid}\right) d\tau$$

$$= \frac{1}{\mid a \mid} f(0) = \int_{-\infty}^{\infty} \left[\frac{1}{\mid a \mid} \delta(\tau)\right] f(\tau) d\tau.$$

$$\therefore \quad \delta(at) = \frac{1}{\mid a \mid} \delta(t); \quad a < 0, \quad a \neq 0.$$

2.5 Using the result $(f\delta)' = f'\delta + f\delta'$ we have

$$\int_{-\infty}^{\infty} f(t)\delta'(t) dt = \int_{-\infty}^{\infty} (f\delta)' dt - \int_{-\infty}^{\infty} \delta f' dt = [f\delta]_{-\infty}^{\infty} - f'(0) = -f'(0)$$

since $\delta(t) = 0 \forall t$ except at $t = 0$.

2.6 Observe that

$$\delta(t^2 - a^2) = \delta[(t - a)(t + a)].$$

Since $\delta(t) = 0$ unless $t = 0$, it follows that $\delta(t^2 - a^2) = 0$ except at the point where $t = \pm a$. Hence, we can write

$$\int_{-\infty}^{\infty} \delta(t^2 - a^2) f(t) dt = \int_{-a-\epsilon}^{-a+\epsilon} \delta[(t + a)(t - a)] f(t) dt + \int_{a-\epsilon}^{a+\epsilon} \delta[(t + a)(t - a)] f(t) dt, \quad a > 0$$

where $0 < \epsilon < 2a$ and ϵ is arbitrarily small. Now, in the neighborhood of $t = -a$, the factor $t - a$ may be replaced by $-2a$. Then,

$$\int_{-a-\epsilon}^{-a+\epsilon} \delta[(t + a)(t - a)] f(t) dt = \int_{-a-\epsilon}^{-a+\epsilon} \delta[(-2a)(t + a)] f(t) dt = \int_{-\infty}^{\infty} \frac{1}{2a} \delta(t + a) f(t) dt$$

since $\delta(-t) = \delta(t)$ and $\delta(at) = \delta(t)/a, \quad a > 0$. N.B. The infinite limits can be used again because $\delta(t + a) = 0$ except at $t = -a$. In a similar manner,

$$\int_{a-\epsilon}^{a+\epsilon} \delta[(t+a)(t-a)]f(t)dt = \int_{-\infty}^{\infty} \frac{1}{2a}\delta(t-a)f(t)dt, \quad a > 0.$$

Hence,

$$\delta(t^2 - a^2) = \frac{1}{2a}[\delta(t-a) + \delta(t+a)].$$

Note, that this result breaks down for $a = 0$ and there is apparently no way of interpreting the expression $\delta(t^2)$.

2.7 $\delta(\sin t) = 0$ except at the points where $\sin t = 0$ which occurs when ever $t = n\pi$ where $n = 0, \pm 1, \pm 2, \pm 3, ..., \pm\infty$.

$$\therefore \quad \int_{-\infty}^{\infty} \delta(\sin t)f(t)dt = \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(t - n\pi)f(t)dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - n\pi)f(t)dt$$

and

$$\delta(\sin t) = \sum_{n=-\infty}^{\infty} \delta(t - n\pi).$$

Similarly,

$$\delta(\cos t) = \sum_{n=-\infty}^{\infty} \delta\left(t - \frac{n\pi}{2}\right).$$

2.8

$$\int_{-\infty}^{\infty} \delta(t)e^{-i\omega t}dt = e^{-i\omega 0} = 1$$

and by inversion

$$\delta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t}d\omega.$$

Now

$$\cos t = \frac{1}{2}(e^{it} + e^{-it}),$$

$$\therefore \quad \int_{-\infty}^{\infty} e^{-i\omega t} \cos t dt = \frac{1}{2}\left[\int_{-\infty}^{\infty} e^{it(1-\omega)}dt + \int_{-\infty}^{\infty} e^{-it(1+\omega)}dt\right]$$

$$= \pi\delta(1+\omega) + \delta[-(1+\omega)] = \pi[\delta(1-\omega) + \delta(1+\omega)]$$

Similarly,

$$\sin t = \frac{1}{2i}(e^{it} - e^{-it}),$$

$$\therefore \int_{-\infty}^{\infty} e^{-i\omega t}\sin t dt = \frac{1}{2i}\left[\int_{-\infty}^{\infty} e^{it(1-\omega)}dt - \int_{-\infty}^{\infty} e^{-it(1+\omega)}dt\right]$$
$$= -i\pi\delta(1-\omega) - \delta(1+\omega) = i\pi[\delta(1+\omega) - \delta(1-\omega)].$$

2.9 Substituting and differentiating, we have

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}(-u^2+k^2)g(u,k)\exp(iuX)du = -\frac{1}{2\pi}\int_{-\infty}^{\infty}\exp(iuX)du$$

from which it follows that

$$g(u,k) = \frac{1}{u^2-k^2}.$$

Hence

$$g(X,k) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{\exp(iuX)}{u^2-k^2}du = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{\exp(iuX)}{(u-k)(u+k)}du$$

and the problem is reduced to that of evaluating the above integral and in particular, the contour integral

$$I = \oint_C \frac{e^{izX}}{(z-k)(z+k)}dz.$$

This integral has two real roots $z = \pm k$. Thus, we can consider a contour composed of a semicircle of radius R so that

$$I = \int_{-R}^{R}\frac{e^{iuX}}{(u-k)(u+k)}du + \int_S \frac{e^{izX}}{(z-k)(z+k)}dz$$

where S is the path defining the circumference of the semicircle. Now, the pole at $z = k$ is given by

$$\lim_{z\to k}\left[\frac{(z-k)e^{izX}}{(z+k)(z-k)}\right] = \frac{1}{2k}e^{ikX}$$

and the pole at $z = -k$ is given by

$$\lim_{z\to -k}\left[\frac{(z+k)e^{izX}}{(z+k)(z-k)}\right] = -\frac{1}{2k}e^{-ikX}.$$

Hence, from Cauchy's residue theorem,

$$I = \oint_C \frac{e^{izX}}{(z-k)(z+k)}dz = 2\pi i\left(\frac{1}{2k}e^{ikX} - \frac{1}{2k}e^{-ikX}\right) = -2\pi\frac{\sin(kX)}{k}.$$

With $z = Re^{i\theta}$, the integral over the path S is

$$\int_S \frac{e^{iXR\cos\theta}e^{-XR\sin\theta}}{(Re^{i\theta}+k)(Re^{i\theta}-k)}e^{i\theta}(dR + iRd\theta) \longrightarrow 0 \quad \text{as} \quad R\longrightarrow\infty.$$

Thus we can write

$$\int_{-\infty}^{\infty} \frac{e^{iuX}}{(u-k)(u+k)} du = -2\pi \frac{\sin(kX)}{k}$$

and we obtain the result,

$$g(x \mid x_0, k) = -\frac{\sin(k \mid x - x_0 \mid)}{k}.$$

This expression for the Green's function characterizes the propagation of both left and right travelling waves (associated with the poles at $u = k$ and $u = -k$ respectively). Thus, if we want the Green's function for left travelling waves alone (equivalent to computing the integral for the pole at $u = k$ alone) for example, then we obtain

$$g(x \mid x_0, k) = \frac{i}{2k} \exp(ik \mid x - x_0 \mid).$$

2.10 The Green's function solution to this equation is given by (as shown in Chapter 2)

$$u(x_0, k) = \int_{-\infty}^{\infty} f(x) g(x \mid x_0, k) dx$$

where, from question 2.9, the Green's function is given by

$$g(x \mid x_0, k) = -\frac{\sin(k \mid x - x_0 \mid)}{k} = \frac{i \exp(ik \mid x - x_0 \mid)}{2k} - \frac{i \exp(-ik \mid x - x_0 \mid)}{2k}.$$

The two exponential terms above represent left and right travalling wave respectively. If we consider the case for left travelling waves, then the Green's function solution becomes

$$u(x_0, k) = \frac{i}{2k} \int_{-\infty}^{\infty} f(x) e^{ik|x - x_0|} dx.$$

Now, as $x_0 \longrightarrow \infty$, the asymptotic solution is

$$u(x_0, k) = \frac{i}{2k} e^{ikx_0} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx$$

which is characterized by the Fourier transform of $f(x)$. In obtaining this result, and at the risk of being pedantic, note that

$$\mid x - x_0 \mid = \sqrt{(x - x_0)} = \sqrt{x^2 - 2xx_0 + x_0^2} = x_0 \left(1 - \frac{2x}{x_0} + \frac{x^2}{x_0^2}\right)^{\frac{1}{2}}$$

$$= x_0 \left(1 - \frac{x}{x_0} + \frac{x^2}{2x_0^2} + ...\right)$$

via a binomial expansion. Now if $x_0 \longrightarrow \infty$ then $x/x_0 \longrightarrow 0 \forall x \in (-\infty, \infty)$ and hence

$$\mid x - x_0 \mid \longrightarrow x_0 - x, \quad x_0 \longrightarrow \infty.$$

A.1.3 Solutions to Problems Given in Chapter 3

3.1 (i)

$$\int_{-\pi}^{\pi} \cos(nt)\sin(nt)dt = \frac{1}{2}[\sin(k+n)+\sin(k-n)t]dt$$

$$= \frac{1}{2}\left(\left[-\frac{\cos(k+n)t}{k+n}\right]_{-\pi}^{\pi} + \left[-\frac{\cos(k-n)}{k-n}\right]_{-\pi}^{\pi}\right) = \frac{1}{2}\left[-\frac{1}{k+n} - \left(-\frac{1}{k+n}\right)\right] = 0 \forall n,k.$$

(ii)

$$\int_{-\pi}^{\pi} \sin(nt)\sin(kt)dt = \frac{1}{2}[\cos(n-k)-\cos(n+k)t]dt$$

$$= \frac{1}{2}\left(\left[\frac{\sin(n-k)t}{n-k}\right]_{-\pi}^{\pi} - \left[-\frac{\sin(n+k)}{n+k}\right]_{-\pi}^{\pi}\right) = 0, n \neq k.$$

If $n = k$, then

$$\int_{-\pi}^{\pi} \sin(nt)\sin(kt)dt = \frac{1}{2}\left(\int_{-\pi}^{\pi}[1-\cos(2nt)]dt\right) = \frac{1}{2}\left([t]_{-\pi}^{\pi} - \left[\frac{1}{2n}\sin(2nt)\right]_{-\pi}^{\pi}\right) = \pi.$$

$$\therefore \int_{-\pi}^{\pi} \sin(nt)\sin(kt)dt = \begin{cases} 0, & n \neq k; \\ \pi, & n = k. \end{cases}$$

3.2 Sketches are left to the reader.

3.3 The Fourier series representation of this (periodic) function is

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}[a_n\cos(nt) + b_n\sin(nt)]$$

where

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)dt = \frac{1}{\pi}\int_{-\pi}^{0} 0dt + \frac{1}{\pi}\int_{0}^{\pi} tdt = \frac{1}{\pi}\left[\frac{t^2}{2}\right]_0^{\pi} = \frac{\pi}{2}$$

$$a_n = \frac{}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt = \frac{1}{\pi}\int_{0}^{\pi} t\cos(nt)dt$$

$$= \frac{1}{\pi}\left[\frac{t\sin(nt)}{n} + \frac{\cos(nt)}{n^2}\right]_0^{\pi} = \frac{1}{\pi}\left[\frac{\cos(n\pi}{n^2 - \frac{1}{n^2}}\right] = \frac{1}{\pi}\frac{1}{n^2}[(-1)^n - 1]$$

and

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(nt)dt = \frac{1}{\pi}\int_{0}^{\pi} t\sin(nt)dt = \frac{1}{\pi}\left[-\frac{t\cos(nt)}{n} + \frac{\sin(nt)}{n^2}\right]_0^{\pi}$$

$$= -\frac{\cos(n\pi)}{n} = -\frac{(-1)^n}{n}.$$

$$\therefore \quad f(t) = \frac{\pi}{4} + \frac{1}{\pi}\sum_{n=1}^{\infty}\frac{1}{n^2}[(-1)^n - 1]\cos(nt) - \sum_{n=1}^{\infty}\frac{(-1)^n}{n}\sin(nt)$$

$$= \frac{\pi}{4} - \frac{2}{\pi}\left(\frac{\cos t}{1^2} + \frac{\cos(3t)}{3^2} + ...\right) + \left(\frac{\sin t}{1} - \frac{\sin(2t)}{2} + \frac{\sin(3t)}{3} - ...\right).$$

Finally, since $f(0) = 0$, we have

$$0 = \frac{\pi}{4} - \frac{2}{\pi}\left(1 + \frac{1}{3^2} + \frac{1}{5^2} + ...\right).$$

Rearranging,

$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + ...$$

3.4 The Fourier sine series for the function $\pi - t, \quad 0 \le t \le \pi$ is

$$\sum_{n=1}^{\infty} b_n \sin(nt)$$

where

$$b_n = \frac{2}{\pi}\int_0^{\pi}(\pi - t)\sin(nt)dt = \frac{2}{\pi}\left[-\frac{\pi\cos(nt)}{n} + \frac{t\cos(nt)}{n} - \frac{\sin(nt)}{n^2}\right]_0^{\pi} = \frac{2}{\pi}\frac{\pi}{n} = \frac{2}{n}$$

$$\therefore \quad \pi - t = \sum_{n=1}^{\infty}\frac{2}{n}\sin(nt), 0 \le t \le \pi.$$

The Fourier cosine series for this function is

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nt)$$

where

$$a_0 = \frac{2}{\pi}\int_0^{\pi}(\pi - t)dt = \frac{2}{\pi}\left[\pi t - \frac{x^2}{2}\right]_0^{\pi} = \pi$$

and

$$a_n = \frac{2}{\pi}\int_0^{\pi}(\pi - t)\cos(nt)dt$$

$$= \frac{2}{\pi}\left[\frac{\pi\sin(nt)}{n} - \frac{t\sin(nt)}{n} - \frac{\cos(nt)}{n^2}\right]_0^{\pi} = \frac{2}{\pi}\frac{1}{n^2}[1 - (-1)^n].$$

$$\therefore \quad \pi - t = \frac{\pi}{2} + \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{[1 - (-1)^n]}{n^2}\cos(nt), 0 \le t \le \pi.$$

3.5 The Fourier sine series for $\cos t, 0 \le t \le \pi$, is

$$\sum_{n=1}^{\infty} b_n \sin(nt)$$

where

$$b_n = \frac{2}{\pi}\int_0^{\pi} \cos t \sin(nt)dt = \frac{2}{\pi}\int_0^{\pi} \frac{1}{2}[\sin(1+n)t + \sin(n-1)t]dt$$

$$= \frac{1}{\pi}\left[-\frac{\cos(1+n)t}{1+n} - \frac{\cos(n-1)t}{n-1}\right]_0^{\pi}$$

$$= \frac{1}{\pi}\left(\left[-\frac{(-1)^{n+1}}{1+n} - \frac{(-1)^{n+1}}{n-1}\right] - \left[-\frac{1}{1+n} - \frac{1}{n-1}\right]\right)$$

$$= \frac{1}{\pi}\left[\frac{1+(-1)^n}{n+1} + \frac{1+(-1)^n}{n-1}\right] = \frac{1}{\pi}\frac{2n[1+(-1)^n]}{n^2-1}.$$

Hence, we obtain the result

$$\cos t = \frac{2}{\pi}\sum_{n=1}^{\infty}[1+(-1)^n]\frac{n}{n^2-1}\sin(nt).$$

3.6

$$x = \sum_{n=1}^{\infty} b_n \sin(nt)$$

since $f(t) = t$ is an odd function [i.e. $f(-t) = -f(t)$] and

$$b_n = \frac{2}{\pi}\int_0^{\pi} t\sin(nt)dt = \frac{2}{\pi}\left(\left[-\frac{t}{n}\cos(nt)\right]_0^{\pi} + \int_0^{\pi}\frac{\cos(nt)}{n}dt\right)$$

$$= -\frac{2}{n}\cos(n\pi) = \frac{2}{n}(-1)^{n+1}.$$

$$\therefore \quad t = \sum_{n=1}^{\infty}\frac{2}{n}(-1)^{n+1}\sin(nt)$$

and

$$F(\omega) = \int_{-\infty}^{\infty} te^{-i\omega t}dt = \sum_{n=1}^{\infty}\frac{2}{n}(-1)^{n+1}\int_{-\infty}^{\infty}\sin(nt)e^{-i\omega t}dt$$

$$2\pi i\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n}[\delta(k+n) - \delta(k-n)].$$

3.7 $f(t) = \mid t \mid$ is an even function and

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nt),$$

$$a_n = \frac{2}{\pi}\int_0^{\pi} t\cos(nt)dt = \begin{cases} \frac{2}{\pi}\frac{1}{n^2}[(-1)^n - 1], & n \neq 0; \\ \pi, & n = 0. \end{cases}$$

$$\therefore \quad f(t) = \frac{\pi}{2} + \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{1}{n^2}[(-1)^n - 1]\cos(nt).$$

Now,

$$F(\omega) = \frac{\pi}{2}\int_{-\infty}^{\infty} e^{-i\omega t}dt + \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{1}{n^2}[(-1)^n - 1]\int_{-\infty}^{\infty}\cos(nt)e^{-i\omega t}dt$$

$$= \pi^2\delta(\omega) + 2\sum_{n=1}^{\infty}\frac{1}{n^2}[(-1)^n - 1][\delta(k-n) + \delta(k+n)].$$

3.8 $f(t) = t^2$ is even and so

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n\cos(nt)$$

where

$$a_n = \frac{2}{\pi}\int_0^{\pi} t^2\cos(nt)dt = \begin{cases} \frac{4}{n^2}(-1)^n, & n \neq 0; \\ \frac{2\pi^2}{3}, & n = 0. \end{cases}$$

Thus,

$$f(t) = \frac{\pi^2}{3} + 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}\cos(nt)$$

and

$$F(\omega) = \frac{\pi^3}{3}\int_{-\infty}^{\infty} e^{-i\omega t}dt + 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}\int_{-\infty}^{\infty}\cos(nt)e^{-i\omega t}dt$$

$$= \frac{2\pi^3}{3}(\omega) + 4\pi\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}[\delta(k-n) + \delta(k+n)].$$

Also, since $f(0) = 0$, we have

$$0 = \frac{\pi^2}{3} + 4\left(-1 + \frac{1}{2^2} - \frac{1}{3^3} + \frac{1}{4^2} - ...\right)$$

or after rearranging

$$\frac{\pi^2}{12} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + ...$$

A.1.4 Solutions to Problems Given in Chapter 4

4.1 For $\omega \neq 0$,

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a} te^{-i\omega t}dt = i\frac{d}{d\omega}\int_{-a}^{a} e^{-i\omega t}dt = i\frac{d}{d\omega}\left[-\frac{1}{i\omega}e^{-i\omega t}\right]_{-a}^{a}$$

$$= 2i\frac{d}{d\omega}\left[\frac{\sin(\omega a)}{\omega}\right] = 2i\left[a\frac{\cos(\omega a)}{\omega} - \frac{\sin(\omega a)}{\omega^2}\right] = \frac{2ia}{\omega}\left[\cos(\omega a) - \frac{\sin(\omega a)}{\omega a}\right].$$

For $\omega = 0$,

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a} tdt = \left[\frac{t^2}{2}\right]_{-a}^{a} = 0.$$

4.2

$$\int_{-\infty}^{\infty} \text{comb}(t)e^{-i\omega t}dt = \int_{-\infty}^{\infty}\sum_{j=1}^{n}\delta(t-t_j)e^{-i\omega t}dt = \sum_{j=1}^{n}\int_{-\infty}^{\infty}\delta(t-t_j)e^{-i\omega t}dt = \sum_{j=1}^{n}e^{-i\omega t_j}$$

using the sampling property of the δ-function.

4.3

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a}\left(1-\frac{|\,t\,|}{a}\right)e^{-i\omega t}dt$$

$$= \int_{0}^{a}\left(1-\frac{|\,t\,|}{a}\right)e^{-i\omega t}dt + \int_{-a}^{0}\left(1-\frac{|\,t\,|}{a}\right)e^{-i\omega t}dt$$

$$= \int_{0}^{a}\left(1-\frac{t}{a}\right)e^{-i\omega t}dt - \int_{0}^{a}\left(1-\frac{|\,t\,|}{a}\right)e^{-i\omega t}dt$$

$$= \int_{0}^{a}\left(1-\frac{x}{a}\right)e^{-i\omega t}dt + \int_{0}^{a}\left(1-\frac{t}{a}\right)e^{i\omega t}dt$$

$$= 2\int_{0}^{a}\left(1-\frac{t}{a}\right)\cos(\omega t)dt = 2\int_{0}^{a}\cos(\omega t)dt - \frac{2}{a}\int_{0}^{a}t\cos(\omega t)dt$$

$$= \frac{2}{\omega}\left[\sin(\omega t)\right]_0^a - \frac{2}{a}\left[t\frac{\sin(\omega t)}{\omega} + \frac{\cos(kt)}{\omega^2}\right]_0^a$$

$$= \frac{2}{\omega}\sin(\omega a - \frac{2}{a}\left[a\frac{\sin(\omega a)}{\omega} + \frac{\cos(\omega a)}{\omega^2} - \frac{1}{\omega^2}\right]$$

$$= \frac{2}{\omega^2 a}[1-\cos(\omega a)] = \frac{4a}{\omega^2 a^2}\sin^2(\omega a/2) = a\text{sinc}^2(\omega a/2).$$

If $\omega = 0$, then

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = 2\int_{0}^{a}(1-t/a)dt = a.$$

$$\therefore\ \hat{F}_1 f(t) = \begin{cases} a\text{sinc}^2(\omega a/2), & \omega \neq 0; \\ a, & \omega = 0 \end{cases}$$

Inverse Fourier transforming, we have

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} a\frac{\sin^2(\omega a/2)}{\omega a/2}e^{i\omega t}d\omega = 1 - \frac{\mid t \mid}{a}, \mid t \mid \leq a.$$

Let $\omega a/2 = y$, then $d\omega = 2dy/a$ and

$$1 - \frac{\mid t \mid}{a} = \frac{1}{2\pi}\int_{-\infty}^{\infty} a\frac{\sin^2 y}{y^2}e^{i\omega t}\frac{2}{a}dy = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{\sin^2 y}{y^2}e^{i\omega t}dy.$$

Now, with $t = 0$,

$$1 = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{\sin^2 y}{y^2}dy$$

or

$$\int_{-\infty}^{\infty}\frac{\sin^2 t}{t^2}dt = \pi.$$

4.4

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a}\frac{1}{2a}e^{-i\omega t}dt$$

$$= \frac{1}{2a}\frac{1}{-i\omega}\left[e^{-i\omega t}\right]_{-a}^{a} = \frac{1}{\omega a}\frac{e^{i\omega a} - e^{-i\omega a}}{2} = \frac{\sin(\omega a)}{\omega a}.$$

For $\omega = 0$,

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a}\frac{1}{2a}dt = \frac{1}{2a}[a - (-a)] = 1$$

$$\therefore\ F(\omega) = \begin{cases} \frac{\sin(\omega a)}{\omega a}, & \omega \neq 0; \\ 1, & \omega = 0. \end{cases}$$

from which it follows that

$$\lim_{a\to 0}[F(\omega)] = 1,$$

i.e.

$$1 = \lim_{a\to 0}\int_{-a}^{a}\frac{1}{2a}e^{-i\omega t}dt = \int_{-\infty}^{\infty}\delta(t)e^{-i\omega t}dt.$$

Hence,

$$\int_{-\infty}^{\infty} \delta(t) e^{-i\omega t} dt = 1$$

and by inversion

$$\delta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} d\omega.$$

4.5

$$\int_{-\infty}^{\infty} e^{-|t|} e^{-i\omega t} dt = \lim_{a\to\infty} \int_{-a}^{a} e^{-|t|} e^{-i\omega t} dt$$

$$= \lim_{a\to\infty} \left(\int_{0}^{a} e^{-|t|} e^{-i\omega t} dt + \int_{-a}^{0} e^{-|t|} e^{-i\omega t} dt \right)$$

$$= \lim_{a\to\infty} \left(\int_{0}^{a} e^{-t} e^{-i\omega t} dt + \int_{0}^{a} e^{-|t|} e^{-i\omega t} dt \right)$$

$$= \lim_{a\to\infty} \left(\int_{0}^{a} e^{-t} e^{-i\omega t} dt + \int_{0}^{a} e^{-t} e^{i\omega t} dt \right)$$

$$= \lim_{a\to\infty} \left(\int_{0}^{a} e^{-t(1+i\omega)} dt + \int_{0}^{a} e^{-t(1-i\omega)} dt \right)$$

$$= \lim_{a\to\infty} \left[\frac{-1}{1+i\omega} (e^{-a} e^{-i\omega a} - 1) - \frac{1}{1-i\omega} (e^{-a} e^{i\omega a} - 1) \right]$$

$$= \frac{1}{1+i\omega} + \frac{1}{1-i\omega} = \frac{2}{1+\omega^2}.$$

Hence,

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{2}{1+\omega^2} e^{i\omega t} d\omega = e^{-|t|}$$

and with $t = 1$, we get

$$\int_{-\infty}^{\infty} \frac{e^{i\omega}}{1+\omega^2} d\omega = \frac{\pi}{e}.$$

Equating real and imaginary parts, the imaginary part is zero the the real part is

$$\int_{-\infty}^{\infty} \frac{\cos t}{1+t^2} dt = \frac{\pi}{e}.$$

4.6

$$\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-a}^{a} \frac{1}{-i\omega}\left[e^{-i\omega t}\right]_{-a}^{a} = 2\frac{\sin(\omega a)}{\omega}.$$

From the product theorem

$$f(t)f(t) \Longleftrightarrow \frac{1}{2\pi}F(\omega) \otimes F(\omega).$$

Hence, we can write

$$\int_{-a}^{a} e^{-ixt}dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{2\sin(x-\omega)a}{\omega}\frac{2\sin(\omega a)}{\omega}d\omega.$$

With $x = 0$, we then get

$$\frac{2}{\pi}\int_{-\infty}^{\infty} \frac{\sin(-\omega a)\sin(\omega a)}{\omega^2}d\omega = \int_{-a}^{a} dt = 2a.$$

Writing $t = \omega a$, we obtain

$$\int_{-\infty}^{\infty} \frac{\sin(-t)\sin t}{t^2}dt = \pi$$

but $\sin(-t)\sin t$ is an even function,

$$\int_{-\infty}^{\infty} \frac{\sin^2 t}{t^2} = \pi$$

or

$$\int_{0}^{\infty} \frac{\sin^2 t}{t^2}dt = \frac{\pi}{2}.$$

4.7

$$f(t) \otimes e^{i\omega t} = \int_{-\infty}^{\infty} f(t-\tau)e^{i\omega\tau}d\tau = \int_{-\infty}^{\infty} f(\tau)e^{i\omega(t-\tau)d\tau}$$

$$= e^{i\omega t}\int_{-\infty}^{\infty} f(\tau)e^{-i\omega\tau}d\tau = e^{i\omega t}F(\omega).$$

Suppose that $f(t) = g(t) \otimes h(t)$, then,

$$F(\omega)e^{i\omega t} = f(t) \otimes e^{i\omega t}$$

$$= g(t) \otimes h(t) \otimes e^{i\omega t} = g(t) \otimes H(\omega)e^{i\omega t} = G(\omega)H(\omega)e^{i\omega t}.$$

Hence, $F(\omega) = G(\omega)H(\omega)$ or

$$g(t) \otimes h(t) \Longleftrightarrow G(\omega)H(\omega).$$

4.8

$$f(t) \otimes e^{i\alpha t^2} = \int_{-\infty}^{\infty} f(\tau) e^{i\alpha(t-\tau)^2} d\tau$$

$$= \int_{-\infty}^{\infty} f(\tau) e^{i\alpha t^2} e^{-2i\alpha\tau t} e^{i\alpha\tau^2} d\tau = e^{i\alpha t^2} \int_{-\infty}^{\infty} f(\tau) e^{i\alpha\tau^2} e^{-2i\alpha\tau t} d\tau.$$

Now, $e^{i\alpha\tau^2} \simeq 1 + i\alpha\tau^2, \quad \alpha\tau^2 << 1$. Hence, provided α is small enough, we can write

$$f(t) \otimes e^{i\alpha t^2} - e^{i\alpha t^2} \left[\int_{-\infty}^{\infty} f(\tau) e^{-2i\alpha t\tau} d\tau + \int_{-\infty}^{\infty} f(\tau) i\alpha\tau^2 e^{-2i\alpha t\tau} d\tau \right]$$

$$= e^{i\alpha t^2} [F(2\alpha t) - i\alpha F''(2\alpha t)]$$

since

$$F''(u) \Longleftrightarrow -y^2 f(y)$$

where $u = 2\alpha\tau$.

4.9

$$f'(t) \Longleftrightarrow i\omega F(\omega).$$

$$\therefore \quad f'(t) \otimes \frac{1}{\pi t} \Longleftrightarrow i\omega F(\omega)[-i\text{sgn}(\omega)] = \omega\text{sgn}(\omega)F(\omega).$$

But $\omega\text{sgn}(\omega) = \mid \omega \mid$,

$$\therefore \quad f'(t) \otimes \frac{1}{\pi t} \Longleftrightarrow \mid \omega \mid F(\omega).$$

Also,

$$-\frac{1}{\pi t^2} = \frac{d}{dt}\frac{1}{\pi t}$$

and

$$\frac{d}{dt}\frac{1}{\pi t} \Longleftrightarrow i\omega[-i\text{sgn}(\omega)] = \omega\text{sgn}(\omega),$$

$$\therefore \quad -\frac{1}{\pi t^2} \Longleftrightarrow \mid \omega \mid$$

Further,

$$-\frac{1}{\pi t} \otimes \frac{1}{\pi t} \Longleftrightarrow -[-i\text{sgn}(\omega)][-i\text{sgn}(\omega)] = [\text{sgn}(\omega)]^2 = 1 \forall \omega.$$

Now,

$$\hat{F}_1^{-1}[\text{sgn}(\omega)]^2 = \hat{F}_1^{-1}(1) = \delta(t)$$

and hence,

$$-\frac{1}{\pi t} \otimes \frac{1}{\pi t} = \delta(t).$$

4.10 Let

$$f_n = \frac{1}{N}\sum_m F_m \exp(2\pi inm/N)$$

and

$$g_n = \frac{1}{N}\sum_m G_m \exp(2\pi inm/N).$$

Then,

$$\sum_n f_n g_{n+m} = \frac{1}{N^2}\sum_n\sum_k F_k \exp(2\pi ikn/N)\sum_\ell G_\ell \exp[2\pi i\ell(n+m)/N]$$

$$= \frac{1}{N}\sum_k F_k \sum_\ell G_\ell \exp(2\pi i\ell m/N)\frac{1}{N}\sum_n \exp[2\pi in(k+\ell)/N]$$

$$= \frac{1}{N}\sum_k F_k \sum_\ell G_\ell \exp(2\pi i\ell m/N)\delta_{k(-\ell)} = \frac{1}{N}\sum_k F_k G_k \exp(-2\pi ikm/N)$$

$$= \frac{1}{N}\sum_k F_k G_{-k} \exp(2\pi ikm/N) = \frac{1}{N}\sum_k F_k G_k^* \exp(-2\pi ikm/N)$$

Also,

$$\sum_n f_n g_n \exp(-2/N)$$

$$= \sum_n \exp(-2\pi inm/N)\frac{1}{N}\sum_k G_k \exp(2\pi ikn/N)\frac{1}{N}\sum_\ell F_\ell \exp(2\pi i\ell n/N)$$

$$= \frac{1}{N}\sum_k\sum_\ell F_\ell G_k \frac{1}{N}\sum_n \exp(2\pi in(k+\ell-m)/N)$$

$$= \frac{1}{N}\sum_k\sum_\ell F_\ell G_k \frac{1}{N}\sum_n \exp(2\pi in[k-(m-\ell)]$$

$$= \frac{1}{N}\sum_k\sum_\ell F_\ell G_k \delta_{k(m-\ell)} = \frac{1}{N}\sum_\ell F_\ell G_{m-\ell}.$$

4.11 The Green's function solution to this equation is

$$u(x,k) = u_0 + k^2 g(\mid x \mid, k) \otimes f(x)u(x,k)$$

where g is the solution to

$$\left(\frac{\partial^2}{\partial x^2} + k^2\right) g(\mid x - x_0 \mid, k) = \delta(x - x_0).$$

Now,

$$q \otimes (u - u_0) = k^2 q \otimes g \otimes fu$$

and if we let

$$q \otimes g = s$$

where s is the 'ramp function' as defined, then

$$\frac{\partial^2}{\partial x^2}(q \otimes g) = q \otimes \frac{\partial^2}{\partial x^2} g = \delta(x - x_0).$$

But

$$\frac{\partial^2}{\partial x^2} g = \delta(x - x_0) - k^2 g$$

and therefore

$$q = \delta(x - x_0) + k^2 s.$$

Hence,

$$\frac{\partial^2}{\partial x^2}\left[q \otimes (u - u_0)\right] = k^2 \frac{\partial^2}{\partial x^2}(q \otimes g) \otimes fu = k^2 fu$$

or

$$\frac{\partial^2}{\partial x^2}\left[(u - u_0) + k^2 s \otimes (u - u_0)\right] = k^2 fu$$

giving

$$f(x) = \frac{1}{u(x,k)}\left(\frac{\partial^2}{\partial x^2}(s(x) \otimes [u(x,k) - u_0(x,k)] + \frac{1}{k^2}[u(x,k) - u_0(x,k)])\right)$$

provided $\mid u(x,k) \mid > 0$.

A.1.5 Solutions to Problems Given in Chapter 5

5.1

$$\hat{L}\cosh(at) = \hat{L}\frac{1}{2}(e^{at} + e^{-at})$$

$$= \frac{1}{2}\hat{L}e^{at} + \frac{1}{2}\hat{L}e^{-at} = \frac{1}{2}\left(\frac{1}{p-a} + \frac{1}{p+a}\right) = \frac{p}{p^2 - a^2}, \quad p > \mid a \mid .$$

$$\hat{L}te^{at} = \int_0^\infty e^{-pt} e^{at} t dt = \int_0^\infty e^{-(p-a)t} t dt = \frac{1}{(p-a)^2}, \quad p > a.$$

$$\hat{L}x^{-1/2} = \int_0 e^{-pt} t^{-1/2} dt.$$

Let $pt = u^2$, then $dt = 2udu/p$ and

$$\hat{L}x^{-1/2} = \int_0^\infty e^{-u^2} \sqrt{\frac{p}{u^2}} \frac{2u}{p} du = \frac{2}{\sqrt{p}} \int_0^\infty e^{-u^2} du = \frac{2}{p}\frac{\sqrt{\pi}}{2} = \sqrt{\frac{\pi}{p}}, \quad p > 0.$$

$$\hat{L}\delta(t-a) = \int_0^\infty e^{-pt}\delta(t-a)dt = e^{-at}, \quad a > 0.$$

5.2

$$J_0(at) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2} \left(\frac{at}{2}\right)^{2n} = 1 - \frac{a^2t^2}{2^2} + \frac{a^4t^4}{2^2 4^2} - \frac{a^6t^6}{6^2 4^2 2^2} + ...$$

$$\therefore \quad \hat{L}J_0(at) = \hat{L}1 - \hat{L}\frac{a^2x^2}{2^2} + \hat{L}\frac{a^4t^4}{2^2 4^2} - \hat{L}\frac{a^6t^6}{6^2 4^2 2^2} + ...$$

$$\frac{1}{p} - \frac{a^2}{2^2}\frac{2!}{p^3} + \frac{a^4}{2^2 4^2}\frac{4!}{p^5} - \frac{1}{2^2 4^2 6^2}\frac{6!}{p^7} + ...$$

$$= \frac{1}{p}\left(1 - \frac{a^2}{2}\frac{1}{p^2} + a^2\frac{1\times 3}{2\times 4}\frac{1}{p^4} - a^2\frac{1\times 3\times 5}{2\times 4\times 6}\frac{1}{p^6} + ...\right)$$

$$= \frac{1}{p}\left(1 + \frac{a^2}{p^2}\right)^{-1/2} = \frac{1}{\sqrt{p^2 + a^2}}.$$

5.3 Taking the Laplace transform and using the convolution theorem, we obtain

$$\hat{y}(x) = \hat{f}(x) + \hat{L}\int_0^x g(x-u)y(u)du = \hat{L}f(x) + \hat{L}g(x)\hat{L}y(x).$$

Rearranging,

$$[1 - \hat{L}g(x)]\hat{L}y(x) = \hat{L}f(x).$$

$$\therefore \quad \hat{L}y(x) = \frac{\hat{L}f(x)}{1 - \hat{L}g(x)}$$

Now, if

$$y(x) = \sin(3x) + \int_0^x \sin(x-u)y(u)du$$

then

$$\hat{L}y(x) = \frac{\hat{L}\sin(3x)}{1 - \hat{L}\sin x} = \frac{\frac{3}{p^2+9}}{1 - \frac{1}{p^2+1}} = \frac{3}{p^2+9} + \frac{3}{p^2(p^2+9)}$$

and

$$y(x) = \hat{L}^{-1}\left[\frac{3}{p^2+3^2}\right] + \hat{L}^{-1}\left[\frac{1}{3p^2} - \frac{1}{3(p^2+3^2)}\right]$$

$$= \sin(3x) + \frac{x}{3} - \frac{1}{9}\sin(3x) = \frac{x}{3} + \frac{8}{9}\sin(3x)$$

5.4 Taking Laplace transforms, we obtain

$$p^2Y(p) - y'(0) - py(0) + 4Y(p) = F(p)$$

and with $y'(0) = 1$ and $y(0) = 1$, we get

$$(p^2+4)Y(p) = 1 + p + F(p)$$

or

$$Y(p) = \frac{2}{2(p^2+2^2} + \frac{p}{p^2+2^2} + \frac{1}{2}\frac{2}{p^2+2^2}F(p).$$

Hence,

$$y(x) = \hat{L}^{-1}Y(p) = \frac{1}{2}\sin(2x) + \cos(2x) + \frac{1}{2}\int_0^x f(x-y)\sin(2y)dy.$$

5.5 This integral equation is characterised by a causal convolution. Thus, taking Laplace transforms, we get

$$\Phi(p) = \hat{L}[x] - \hat{L}\left[\int_0^x (t-x)\phi(t)dt\right].$$

$$\hat{L}[x] = \int_0^\infty xe^{-px}dx = \left[-\frac{x}{p}e^{-px}\right]_0^\infty + \int_0^\infty \frac{1}{p}e^{-px}dx = \left[-\frac{1}{p^2}e^{-px}\right]_0^\infty = \frac{1}{p^2}.$$

Also,

$$\hat{L}\left[\int_0^x (t-x)\phi(t)dt\right] = \hat{L}[x]\hat{L}[\phi(x)] = \frac{1}{p^2}\Phi(p).$$

$$\therefore\ \ \Phi(p) + \frac{1}{p^2}\Phi(p) = \frac{1}{p^2}$$

or

$$\Phi(p) = \frac{1}{p^2+1}.$$

Inverting, we have $\phi(x) = \sin x$.

5.6 Taking the sine transform, i.e.

$$F(k) = \frac{2}{\pi}\int_0^\infty f(x)\sin(kx)dx$$

we obtain

$$-k^2U(k,t) + \frac{2}{\pi}ku(0,t) = \frac{\partial}{\partial t}U(k,t).$$

The solution to this equation is

$$U(k,t) = Ae^{-k^2t} + \frac{2}{\pi k}$$

where A is a constant of integration. Now,

$$U(k,0) = \int_0^\infty u(x,0)\sin(kx)dx = 0$$

and putting $t = 0$, we find that $A = -2/(\pi k)$. Hence,

$$U(k,t) = \frac{2}{\pi k}\left(1 - e^{-k^2 t}\right)$$

and therefore (taking the inverse sine transform),

$$u(x,t) = \frac{2}{\pi}\int_0^\infty \frac{1}{k}\left(1 - e^{-k^2 t}\right)\sin(kx)dk$$

5.7 The z-transform is

$$\hat{Z}[\sin(k\omega T)] = F(z) = \sum_{k=0}^{\infty}[\sin(k\omega T)]z^{-k} = \sum_{k=0}^{\infty}\left(\frac{e^{ik\omega T} - e^{-ik\omega T}}{2i}\right)z^{-k}.$$

Now, using the result

$$\hat{Z}[(e^{i\omega T})^k] = \frac{z}{z - e^{i\omega T}}, \quad |\, z \,| > 1$$

we have

$$\hat{Z}[\sin(k\omega T)] = \frac{z\sin(\omega T)}{(z - e^{i\omega T})(z - e^{-i\omega T})} = \frac{z\sin(\omega T)}{z^2 - 2z\cos(\omega T) + 1}, \quad |\, z \,| > 1.$$

Similarly,

$$\hat{Z}[\cos(\omega kT)] = \hat{Z}\left(\frac{e^{i\omega kT} + e^{-i\omega kT}}{2}\right) = \frac{1}{2}\left(\frac{z}{z - e^{i\omega T}} + \frac{z}{z - e^{-i\omega T}}\right)$$

$$= \frac{z[z - \cos(\omega T)]}{z^2 - 2z\cos(\omega T) + 1}, \quad |\, z \,| > 1.$$

5.8 Taking the z-transform and noting that

$$\hat{Z}[f(k+n) = z^n F(z) - \sum_{j=0}^{n-1} f(j)z^{n-j}, \quad k \geq -n$$

we get

$$(z^2 + a_1 z + a_2)C(z) - c(0)z^2 - c(1)z - a_1 c(0)z$$
$$= (b_0 z^2 + b_1 z + b_2)R(z) - b_0 r(0)z^2 - b_0 r(1)z - b_1 r(0)z$$

or

$$C(z) = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2}R(z) + \frac{z^2[c(0) - b_0 r(0)] + z[c(1) + a_1 c(0) - b_1 r(0) - b_0 r(1)]}{z^2 + a_1 z + a_2}$$

Also,

$$\begin{array}{ll} \text{for} \quad k = -2: & c(0) = b_0 r(0); \\ \text{for} \quad k = -1: & c(1) + a_1 c(0) = b_0 r(1) + b_1 r(0) \end{array}$$

and the result reduces to

$$C(z) = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2} R(z)$$

or

$$H(z) = \frac{C(z)}{R(z)} = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2}.$$

For the general case, taking the z-transform of each term and using the shifting theorem we have

$$C(z) + a_1 z^{-1} C(z) + ... + a_n z^{-n} C(z) = b_0 R(z) + b_1 z^{-1} R(z) + ... + b_n z^{-n} R(z)$$

so that

$$H(z) = \frac{C(z)}{R(z)} = \frac{b_0 + b_1 z^{-1} + ... + b_n z^{-n}}{1 + a_1 z^{-1} + ... + a_n z^{-n}}.$$

5.9 The proof of this result is based on the application of the convolution theorem. The Fourier transform of $f(t)$ is given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt = \int_{-1/2}^{1/2} \exp(-i\omega t) dt = \text{sinc}(\omega/2).$$

Thus,

$$f(t) \leftrightarrow \text{sinc}(\omega/2)$$

and from the convolution theorem in follows that

$$g(t) = \prod_{n=1}^{N} \!\!\!\!\!\!\otimes \; f_n(t) \leftrightarrow \text{sinc}^N(\omega/2).$$

Using the series expansion of the sine function for an arbitrary constant α,

$$\text{sinc}(\alpha\omega) = \frac{1}{\alpha\omega}\left(\alpha\omega - \frac{1}{3!}(\alpha\omega)^3 + \frac{1}{5!}(\alpha\omega)^5 - \frac{1}{7!}(\alpha\omega)^7 + \ldots\right)$$

$$= 1 - \frac{1}{3!}(\alpha\omega)^2 + \frac{1}{5!}(\alpha\omega)^4 - \frac{1}{7!}(\alpha\omega)^6 + \ldots$$

The N^{th} power of $\text{sinc}(\alpha\omega)$ can be written in terms of a binomial expansion giving

$$\text{sinc}^N(\alpha\omega) = \left(1 - \frac{1}{3!}(\alpha\omega)^2 + \frac{1}{5!}(\alpha\omega)^4 - \frac{1}{7!}(\alpha\omega)^6 + \ldots\right)^N$$

$$= 1 - N\left(\frac{1}{3!}(\alpha\omega)^2 - \frac{1}{5!}(\alpha\omega)^4 + \frac{1}{7!}(\alpha\omega)^6 - \ldots\right)$$

$$+ \frac{N(N-1)}{2!}\left(\frac{1}{3!}(\alpha\omega)^2 - \frac{1}{5!}(\alpha\omega)^4 + \frac{1}{7!}(\alpha\omega)^6 - \ldots\right)^2$$

$$-\frac{N(N-1)(N-2)}{3!}\left(\frac{1}{3!}(\alpha\omega)^2-\frac{1}{5!}(\alpha\omega)^4+\frac{1}{7!}(\alpha\omega)^6-\ldots\right)^3+\ldots$$

$$=1-N\frac{\alpha^2\omega^2}{3!}+N\frac{\alpha^4\omega^4}{5!}-k\frac{\alpha^6\omega^6}{7!}-\ldots+\frac{N(N-1)}{2!}\left(\frac{\alpha^4\omega^4}{(3!)^2}-2\frac{\alpha^6\omega^6}{3!5!}+\ldots\right)$$

$$-\frac{N(N-1)(N-2)}{3!}\left(\frac{\alpha^6\omega^6}{(3!)^3}+\ldots\right)$$

$$=1-\frac{N}{3!}\alpha^2\omega^2+\left(\frac{N}{5!}\alpha^4+\frac{N(N-1)}{2!(3!)^2}\alpha^4\right)\omega^4$$

$$-\left(\frac{N}{7!}\alpha^6+\frac{N(N-1)}{3!5!}\alpha^6+\frac{N(N-1)(N-2)}{3!(3!)^3}\alpha^6\right)\omega^6+\ldots$$

Now the series representation of the exponential (for an arbitrary positive constant c) is

$$\exp(-c\omega^2)=1-c\omega^2+\frac{1}{2!}c^2\omega^4-\frac{1}{3!}c^3\omega^6+\ldots$$

Equating terms involving ω^2, ω^4 and ω^6 it is clear that (evaluating the factorials),

$$c=\frac{1}{6}N\alpha^2,$$

$$\frac{1}{2}c^2=\left(\frac{1}{120}N+\frac{1}{72}N(N-1)\right)\alpha^4$$

or

$$c^2=\left(\frac{1}{36}N^2-\frac{1}{90}N\right)\alpha^4,$$

and

$$\frac{1}{6}c^3=\left(\frac{1}{5040}N+\frac{1}{720}N(N-1)+\frac{1}{1296}N(N-1)(N-2)\right)\alpha^6$$

or

$$c^3=\left(\frac{1}{216}N^3-\frac{1}{1080}N^2+\frac{1}{2835}N\right)\alpha^6.$$

Thus, by deduction, we can conclude that

$$C^n=\left(\frac{1}{6}N\right)^n\alpha^{2n}+O(N^{n-1}\alpha^{2n}).$$

Now, for large N, the first terms in the equation above dominates to give the following approximation for the constant c,

$$c\simeq\frac{1}{6}k\alpha^2.$$

We have therefore shown that the N^{th} power of the $\text{sinc}(\alpha)$ function approximates to a Gaussian function (for large N), i.e.

$$\text{sinc}^N(\alpha)\simeq\exp\left(\frac{1}{6}N\alpha^2\omega^2\right).$$

Thus, if $\alpha = \frac{1}{2}$, then

$$g(t) \leftrightarrow \exp\left(-\frac{1}{24}N\omega^2\right)$$

approximately. The final part of the proof is therefore to Fourier invert the function $\exp(-N\omega^2/24)$, i.e. to compute the integral

$$I = \frac{1}{2\pi}\int_{-\infty}^{\infty} \exp\left(-\frac{1}{24}N\omega^2\right)\exp(i\omega t)d\omega.$$

Now,

$$I = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{e}^{-\left[\left(\sqrt{\frac{N}{24}}\omega - \sqrt{\frac{24}{N}}\frac{it}{2}\right)^2 - \frac{6t^2}{N}\right]}d\omega = \frac{1}{\pi}\sqrt{\frac{6}{N}}\mathrm{e}^{-\frac{6t^2}{N}}\int_{-\infty+it\sqrt{\frac{6}{N}}}^{\infty+it\sqrt{\frac{6}{N}}} \mathrm{e}^{-z^2}dz$$

after making the substitution

$$z = \sqrt{\frac{N}{6}}\frac{\omega}{2} - it\sqrt{\frac{6}{N}}.$$

By Cauchy's theorem

$$I = \frac{1}{\pi}\sqrt{\frac{6}{N}}\mathrm{e}^{-\frac{6t^2}{N}}\int_{-\infty}^{\infty} \mathrm{e}^{-z^2}dz = \sqrt{\frac{6}{\pi N}}\mathrm{e}^{-\frac{6t^2}{N}}$$

where we have use the result

$$\int_{-\infty}^{\infty} \exp(-z^2)dz = \sqrt{\pi}.$$

Thus, we can write

$$g(t) = \prod_{n=1}^{N}{}^{\otimes} f_n(t) \simeq \sqrt{\frac{6}{\pi N}}\exp(-6t^2/\omega)$$

for large k which completes the proof.

5.10 (i) The Fourier transform is given by

$$W(\omega) = \int_{-\infty}^{\infty} w(t)\exp(-i\omega t)dt$$

$$= \frac{1}{\pi^{\frac{1}{4}}}\int_{-\infty}^{\infty} \exp[-i(\omega-\omega_0)t]\exp(-t^2/2)dt = \frac{1}{\pi^{\frac{1}{4}}}\sqrt{\frac{\pi}{1/2}}\exp[-(\omega-\omega_0)^2/(4(1/2))]$$

$$= \pi^{\frac{1}{4}}\sqrt{2}\exp[-(\omega-\omega_0)^2/2].$$

(ii) The power spectrum is given by

$$\mid W(\omega) \mid^2 = 2\pi^{\frac{1}{2}} \exp[-(\omega-\omega_0)^2]$$

and the central frequency ω_c can be defined as

$$\omega_c = \mid W \mid^2_{\max}$$

giving

$$\omega_c = \omega_0,$$

i.e. the central frequency is given by the characteristics frequency of the Morlet wavelet.

(iii) To rationalise how the Morlet wavelet picks out such a discontinuity, we follow the wavelet of arbitrary scale as it traverses the discontinuity, the effect of the wavelet location on the transform being discussed for different locations on the wavelet function.

Location A (away from the discontinuity). The wavelet and the constant signal of amplitude a combine to give near-zero values of the integral for the wavelet transform

$$F_L(t) = \frac{1}{\sqrt{L}} \int_{-\infty}^{\infty} f(t) w\left(\frac{\tau - t}{L}\right) d\tau$$

with

$$w(t) = \pi^{-\frac{1}{4}} \exp(i\omega_0 t - t^2/2)$$

The wavelet function must be zero mean (the admissibility condition) and, as it is a localised function, the wavelet becomes approximately zero at relatively short distances from its centre. Hence the wavelet transform effectively becomes a convolution of the wavelet with a constant valued signal producing a zero value.

Location B (the signal discontinuity coincides with the right zero value of the main wavelet lobe). The left-hand lobes of the wavelet produce a contribution to the wavelet transform; the right-hand lobes also produce a contribution to the wavelet transform. These contributions have the same modulus but different signs and thus, the main lobe of the wavelet produces a significant positive value for the transform at this location.

Location C (the signal discontinuity coincides with the wavelet centre). The right and left halves of the wavelet contribute to a zero value of the wavelet transform.

Location D (the signal discontinuity coincides with the left zero value of the main wavelet lobe; this location is similar to B). As the wavelet further traverses the discontinuity, the left hand lobes of the signal produce a contribution to the wavelet transform; the right hand lobes of the signal produces a contribution to the wavelet transform. These contributions have the same modulus, but different signs. This

time, however, the main lobe of the wavelet coincides with the negative constant amplitude signal and hence the wavelet transform produces a significant negative value at this location.

Location E (much later than the discontinuity; this location is similar to location A. The wavelet and signal combine to give near-zero values of the wavelet transform.

As the Morlet wavelet traverses the discontinuity, there are first positive and then negative values returned by the wavelet transform. However, these values are localized in the vicinity of the discontinuity.

5.11 (i) Consider the Wigner-Ville transform of just two complex sinusoids which can be written in the form

$$F_{12}(\omega,t) = \int_{-\infty}^{\infty} \{a_1 \exp[-i\omega_1(t-\tau/2)] + a_2 \exp[-i\omega_2(t-\tau/2)]\}$$

$$.\{a_1 \exp[i\omega_1(t+\tau/2) + a_2 \exp[i\omega_2(t+\tau/2)]\} \exp(-i\omega\tau)d\tau$$

$$= a_1^2 \int_{-\infty}^{\infty} \exp[-i\omega_1(t-\tau/2)] \exp[i\omega_1(t+\tau/2)] \exp(-i\omega\tau)d\tau$$

$$+a_2^2 \int_{-\infty}^{\infty} \exp[-i\omega_2(t-\tau/2)] \exp[i\omega_2(t+\tau/2)] \exp(-i\omega\tau)d\tau$$

$$2a_1a_2 \cos(\omega_2-\omega_1)t \int_{-\infty}^{\infty} \exp[i(\omega_1+\omega_2)\tau/2] \exp(-i\omega\tau)d\tau.$$

The first and second terms of the above equation are the auto terms, i.e. the Wigner-Ville transform of the first and second sinusoids respectively; the third term is the cross-term. Using the integral representation of a delta function we can write

$$F_{12}(\omega,t) = a_1^2\delta(\omega-\omega_1) + a_2^2\delta(\omega-\omega_2) + 2a_1a_2 \cos[(\omega_2-\omega_1)t]\delta[\omega-(\omega_1-\omega_2)]$$

and, by induction, for the four sinusoidal components, the Wigner-Ville transform becomes

$$F_{1234}(\omega,t) = \sum_{n=1}^{4} a_n\delta(\omega-\omega_n) + 2\sum_{n,m} a_na_m \cos[(\omega_m-\omega_n)t]\delta[\omega-(\omega_n+\omega_m)]$$

where $n,m = 1,2,3,4; m \neq n$. Each cross-term is located at the frequency $(\omega_n+\omega_m)$ and oscillates at frequency $(\omega_m-\omega_n)$.

(ii) the Wigner-Ville transform can be interpreted as the Fourier transform of an instantaneous auto-correlation function $c(t,\tau)$ of a (complex) signal, i.e.

$$c(t,\tau) = s^*(t-\tau/2)s(t+\tau/2)$$

where $s(t)$ is the analytic signal (time history) and τ is the time delay.

A.1.6 Supplementary Problems to Part I

I.1 (i) By considering the limit of a periodic train of rectangular pulses, or otherwise, show that

$$\hat{F}_1\left[\sum_{n=-\infty}^{\infty}\delta(t-nT)\right]=\frac{1}{T}\sum_{n=-\infty}^{\infty}\exp(in\omega_0 t)$$

where $T>0$ and $\omega_0\equiv 2\pi/T$.

(ii) Let $f(t)$ be a bounded continuous function which is absolutely integrable over $(-\infty,\infty)$. If $\tilde{f}(\omega)$ denotes the Fourier transform of $f(t)$ show that,

$$\hat{F}_1\left[\sum_{n=-\infty}^{\infty}f(nT)\delta(t-nT)\right]=\frac{1}{T}\sum_{n=-\infty}^{\infty}\tilde{f}(\omega-n\omega_0)$$

and hence derive the sampling theorem for the case when $\tilde{f}(\omega)$ vanishes identically outside $(-\omega_s,\omega_s)$, where $\omega_s<\omega_0/2$.

(iii) Show that the Sampling Expansion remains valid if the Fourier transform $\tilde{f}(\omega)$ of part (ii) above contains delta functions, but fails if $\tilde{f}(\omega)$ contains the derivative of a delta function.

I.2 (i) The classical Gamma Function Γ can be defined by

$$\Gamma(\lambda)=\int_0^{\infty}x^{\lambda-1}\exp(-x)dx$$

where $\lambda>0$. Confirm that the integral defining Γ does converge for all such values of λ, and show that $\Gamma(\lambda+1)=\lambda\Gamma(\lambda)$. Find $\Gamma(1/2)$ and $\Gamma(3/2)$.

(ii) Find the Laplace transform of the function $u(t)t^{-\alpha}$, where $\alpha>-1$, and use your result to show that

$$\hat{L}[u(t)\log\mid t\mid]=-\frac{1}{s}(\gamma+\log s)$$

where

$$\gamma=-\Gamma'(1)=-\int_0^{\infty}\exp(-x)\log(x)dx$$

(iii) Solve the integral equation

$$\int_0^t\frac{y(\tau)}{t-\tau}d\tau=\frac{1}{4}t^2(2\log t-3);\quad t>0$$

Note: you may assume the result $\int_{-\infty}^{\infty}\exp(-x^2)dx=\sqrt{\pi}$

I.3 (i) Let $k(x)$ be a non-negative integrable function such that

$$\int_{-\infty}^{\infty} k(x)dx = K$$

Prove that, if $f(x)$ is any function bounded and continuous on $(-\infty, \infty)$, then

$$\lim_{n\to\infty} \int_{-\infty}^{\infty} f(x)nk(nx)dx = Kf(0)$$

(ii) Find the limits, in the distributional sense, of each of the following:

(a)

$$\lim_{n\to\infty} \left(\frac{\sin^2(nx)}{nx^2} \right)$$

(b)

$$\lim_{n\to\infty} [\sin(n!x)]$$

I.4 (i) If $\phi(t)$ is a continuously differentiable function then formally we can write

$$\delta[\phi(t)] = \frac{1}{\phi'(t)} \left(\frac{d}{dt} u[\phi(t)] \right)$$

where u denotes the Heaviside unit step function and δ is the Dirac delta function. Show that this is meaningful provided that $\phi(t)$ has only simple zeros.

Illustrate the use of this formula by expressing $\delta(\sin t)$ as a periodic infinite sum of translated delta functions.

(ii) If $\phi(t) = \sin \mid t \mid$ show that the formula of part (i) can still be used to derive an equivalent expression for $\delta(\sin \mid t \mid)$ as an infinite sum of translated delta functions, and find that expression.

(iii) Find the (one-sided) Laplace transform of each of the following: $\delta(\sin t)$, $\delta(\sin \mid t \mid)$ and $\mid \sin t \mid$

I.5 (i) Let

$$f(t) = \frac{1}{(a^2 + t^2)}$$

where $a > 0$. By direct evaluation of the Fourier integral, show that

$$\widetilde{f}(\omega) = \frac{\pi}{a} \exp(-a \mid \omega \mid).$$

(ii) If a function $f(t)$ and its (classical) Fourier transform $\widetilde{f}(\omega)$ are both absolutely integrable functions over $(-\infty, +\infty)$ then the Poisson summation formula holds:

$$\sum_{-\infty}^{\infty} f(m) = \sum_{n=-\infty}^{\infty} \widetilde{f}(2\pi n).$$

Use this formula to show that

$$\sum_{n=-\infty}^{\infty} \frac{1}{a^2+n^2} = \frac{\pi}{a}\coth(\pi a).$$

(iii) Let

$$\phi(t) = \frac{1}{d}[u(t+d/2) - u(t-d/2)]$$

where $u(t)$ is the Heaviside unit step function, and $0 < d < T$. By expanding the periodic extension

$$\phi_T(t) = \sum_{n=-\infty}^{\infty} \phi(t-nT)$$

as a Fourier series and allowing $d \to 0$, show that

$$\sum_{n=-\infty}^{\infty} \delta(t-nT) \simeq \frac{1}{T}\sum_{n=-\infty}^{\infty} \exp(in2\pi t/T).$$

(iv) Use the result of (iii) above to find the generalized Fourier Transform of

$$\sum_{n=-\infty}^{\infty} \delta(t-nT)$$

and confirm that this satisfies the distributional definition of the transform of a tempered distribution by applying the Poisson summation formula.

I.6 (i) The Bessel Function $J_0(t)$ can be defined as the inverse Fourier transform of the function

$$2u(1-\omega^2)/\sqrt{1-\omega^2}$$

where u denotes the Heaviside unit step function. Confirm that, with this definition, $J_0(t)$ does satisfy Bessel's equation of order 0,

$$y''(x) + \frac{1}{x}y' + y(x) = 0.$$

(ii) Given that

$$J_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(k!)(n+k)!}\left(\frac{x}{2}\right)^{n+2k}$$

for each positive integer n, and that $J_{-n}(x) = (-1)^n J_n(x)$, show that

$$\exp\left[\frac{x}{2}\left(t-\frac{1}{t}\right)\right] = \sum_{n=-\infty}^{\infty} t^n J_n(x).$$

(iii) Use the result of part (ii) to obtain the following (Fourier series) expansion

$$\cos(x\sin\theta) = J_0(x) + 2\sum_{n=1}^{\infty} J_{2n}(x)\cos(2n\theta).$$

Find a corresponding Fourier series expansion for $\sin(x\sin\theta)$.

I.7 (a) By considering a path parallel to the x-axis and then a path parallel to the y-axis in the complex z-plane of the function $f(z) = u + iv$, show that if $f(z)$ is assumed to be differentiable at $z = x + iy$ then $f(z)$ satisfies the Cauchy-Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \text{and} \quad \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}.$$

Explain the meaning of the term 'analytic function' and determine whether or not the functions z^2 and $\exp(-z^2)$ are analytic.

(b) Green's theorem in the plane states that if S is a closed region in the xy plane bounded by a simple closed curve C and if P and Q are continuous functions of x and y having continuous derivatives in S, then

$$\oint_C (Pdx + Qdy) = \iint_S \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy.$$

Use this theorem together with the Cauchy-Riemann equations defined above to show that if $f(z)$ is an analytic function, then

$$\oint_C f(z)dz = 0 \qquad \text{(Cauchy's Theorem)}.$$

(c) By integrating $\exp(-z^2)$ around a rectangle whose sides are $x = R, x = -R, y = 0$ and $y = b/2\sqrt{a}$ and then letting $R \to \infty$ use Cauchy's theorem to show that

$$\int_{-\infty}^{\infty} \exp(-ax^2)\exp(-ibx)dx = \sqrt{\frac{\pi}{a}}\exp(-b^2/4a),$$

noting that

$$\int_{-\infty}^{\infty} \exp(-x^2)dx = \sqrt{\pi}.$$

I.8 (a) Sketch the periodic function

$$f(x) = \mid x \mid, \quad -\pi \le x \le \pi; \quad f(x + 2\pi) = f(x)$$

and derive its Fourier series representation. Use the result to show that

$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + ...$$

(b) State and prove the convolution theorem for Fourier transforms

Using this theorem or otherwise, obtain a formal solution of the Fredholm equation of the second kind

$$g(x) = f(x) + \lambda \int_{-\infty}^{\infty} h(x-y)g(y)dy$$

where g is the unknown function, specifying the condition on the function h for the solution to be valid.

(c) Prove the result

$$\hat{L}\left[\frac{d}{dx}f(x)\right] = pF(p) - f(0)$$

where $\hat{L}$ is the Laplace transform operator and $F(p)$ is the Laplace transform of $f(x)$.

Use this result to solve the equation

$$\frac{df}{dx} + f = \exp(-ax); \quad f(0) = 1.$$

I.9 (a) If $f(z)$ is analytic in a simply connected region R and C is a contour that lies within R and encloses a point z_0, show that

$$\oint_C \frac{f(z)}{z - z_0} dz = 2\pi i f(z_0).$$

(b) By expanding $f(z)$ about a point z_0 as

$$f(z) = a_0 + a_1(z - z_0) + a_2(z - z_0)^2 + ... + a_n(z - z_0)^n$$

$$= \frac{b_1}{z - z_0} + \frac{b_2}{(z - z_0)} + ... + \frac{b_n}{(z - z_0)^n}$$

using Cauchy's theorem to show that

$$\oint f(z)dz = 2\pi i b_1.$$

Also, prove by induction, that for an n-order pole, b_1 is given by

$$b_1 = \frac{1}{(n-1)!} \lim_{z \to z_0} \frac{d^{(n-1)}}{dz^{(n-1)}} [(z - z_0)^n f(z)].$$

(c) By integrating around a suitable semicircle, use Cauchy's residue theorem to show that

$$\int_0^{\infty} \frac{\cos(mx)}{x^2 + 1} dx = \frac{\pi}{2} e^{-m}; \quad m > 0.$$

I.10 (a) Sketch the periodic wave form described by the function

$$f(x) = \begin{cases} 0, & -1 < x < 0; \\ 1, & 0 < x < 1; \end{cases} \qquad f(x+2) = f(x).$$

By computing the Fourier coefficients of this function, derive its Fourier series representation

Prove Parseval's identity for the Fourier series of a function $f(x)$ over $[-\ell, \ell]$ given by

$$\frac{1}{\ell}\int_{-\ell}^{\ell}[f(x)]^2 dx = \frac{a_0^2}{2} + \sum_{n=1}^{\infty}(a_n^2 + b_n^2)$$

where the integral is taken to exist and a_0, a_n and b_n are the Fourier coefficients.

(c) The Gamma function is defined for all $\alpha > 0$ by

$$\Gamma(\alpha) = \int_0^{\infty} e^{-x}x^{\alpha-1}dx.$$

(i) Prove that $\Gamma(\alpha+1) = \alpha\Gamma(\alpha)$ and that $\Gamma(n+1) = n!$ for any positive integer n.

(ii) Show that

$$\hat{L}[x^\alpha] = \frac{\Gamma(\alpha+1)}{p^{\alpha+1}} \quad \forall \alpha > 0$$

and hence derive the value of $\Gamma\left(\frac{1}{2}\right)$, noting that

$$\int_0^{\infty} e^{-y^2}dy = \frac{\sqrt{\pi}}{2}.$$

I.11 (a) Explain the meaning of the phrase 'the function $f(z)$ is analytic in the region R of the complex plane'.

Show that, if $z = x + iy$, then

$$\mathrm{Im}\left(\frac{1}{z}\right) = -\frac{y}{x^2+y^2}$$

and find $\mathrm{Re}[z^{-1}]$. Hence or otherwise, show that z^{-1} is analytic in every region R which does not contain the origin in its interior or on its boundary.

(b) If C is a circle with centre z_0 and radius r on the complex plane, then we can define an integral I_n as

$$I_n = \oint_C \frac{1}{(z-z_0)^n}dz$$

where n is an integer.

(i) Prove that $I_1 = 2\pi i$ and state the value of I_n for $n \neq 1$.

(ii) If $g(z)$ has a singularity at z_0 but at no other point on or inside C, deduce from the above result that

$$\oint_C g(z)dz = 2\pi i b_1$$

where b_1 is the residue of $g(z)$ at the point $z = z_0$.

(c) The function $f(z)$ is defined as

$$f(z) = \frac{\exp(i\pi z/6)}{z^2 + 4z + 5}.$$

(i) Find the poles of $f(z)$ and show their positions in the complex plane.

(ii) Use Cauchy's residue theorem to integrate $f(z)$ around a semi-circular contour of radius R and use the result to evaluate the indefinite integrals

$$\int_{-\infty}^{\infty} \frac{\cos(\pi x/6)}{x^2 + 4x + 5}dx$$

and

$$\int_{-\infty}^{\infty} \frac{\sin(\pi x/6)}{x^2 + 4x + 5}dx.$$

I.12 Let the functions $f_1(t), f_2(t), f_3(t)$ be all periodic with period 2π which are defined, for $-\pi < t < \pi$ as follows:

$$f_1(t) = t; \quad f_2(t) = t^2; \quad f_3(t) = \frac{1}{3}(t^3 - \pi^2 t).$$

(a) Sketch the graph of $y = f_1(t)$ and its periodic extension and show that $f_1(t)$ can be represented by the Fourier series

$$f_3(t) = 2\left(\sin t - \frac{\sin(2t)}{2} + \frac{\sin(3t)}{3} - \frac{\sin(4t)}{4} + ...\right)$$

(b) Sketch the graph of $y = f_2(t)$, showing its periodic extension, and use the series for $f_1(t)$ (or otherwise) to show that the Fourier series for the function $f_2(t)$ is

$$f_3(t) = \frac{\pi^2}{3} - 4\left(\cos t - \frac{\cos(2t)}{2^2} + \frac{\cos(3t)}{3^2} - \frac{\cos(4t)}{4^2} + ...\right)$$

(c) For any function $f(t)$, defined on the interval $(-\pi, \pi)$, which can be expressed as a Fourier series

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nt) + \sum_{n=1}^{\infty} b_n \sin(nt)$$

Parseval's identity states that

$$\frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n^2 + b_n^2) = \frac{1}{\pi}\int_{-\pi}^{\pi} |\, f(t)\,|^2\, dt$$

Use Parseval's identity on the Fourier series for $f_2(t)$ to show that

$$\frac{\pi^4}{90} = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \frac{1}{5^4} + ...$$

(d) (i) Use the series for $f_2(t)$ to obtain the Fourier series for $f_3(t)$. Why can we expect the constant term in this series to be zero?

(ii) Choose an appropriate value for t at which to evaluate $f_3(t)$ and its Fourier series in order to show that

$$\frac{\pi^3}{32} = 1 - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + ...$$

I.13 (a) The z-transform of the causal sequence $\{x_n\}, n = 0, 1, 2, 3, ...$ is defined as

$$\hat{Z}\{x_n\} = X(z) = \sum_{n=0}^{\infty}\frac{x_n}{z^n}$$

Use this definition to show that

$$\hat{Z}\{a^n\} = \frac{z}{z-a}$$

where a is a constant and state the region of convergence for this transform. Hence find $\hat{Z}\{2^n \exp(i\pi n/2)\}$, giving the region of convergence and deduce that, for real values of z,

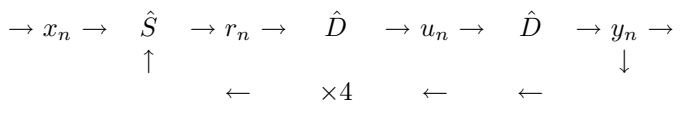
$$\hat{Z}\left\{2^n \cos\left(\frac{n\pi}{2}\right)\right\} = \frac{z^2}{z^2+4}, \quad \hat{Z}\left\{2^n \sin\left(\frac{n\pi}{2}\right)\right\} = \frac{2z}{z^2+4}$$

(b) Using the definition given above for the z-transform of a causal sequence, show that

$$\hat{Z}\{x_{n-m}\} = \frac{X(z)}{z^m}$$

where m is a positive integer.

(c) The circuit diagram given below represents a system into which signal pulses $\{x_n\}$ are fed at unit time intervals, so that $t = n$, and the resulting output is the sequence $\{y_n\}$. The processes denoted by $\hat{D}$ each cause a delay of one time unit and the 'feedback gain' multiplies the pulse passing through it by 4. The process $\hat{S}$ represents the summation of the signals meeting at this junction, being subtracted.



(i) Show that y_n satisfies the difference equation

$$y_{n+2} + 4y_n = x_n.$$

(ii) If the input x_n consists of a single impulse of size 2 at time $t = 0$ so that the sequence $\{x_n\} = \{2, 0, 0, 0, 0, ...\}$, write down the value of $\hat{Z}\{x_n\}$. If it is also known that $y_0 = 1$ and $y_1 = 4$, use the z-transform to solve the difference equation.

(iii) Check that the solution to (ii) above gives correct values of y_0 and y_1 and use your solution to find the value of y_7.

I.14 (a) A bandlimited function $f(t)$ has Fourier transform $F(\omega)$ given by

$$F(\omega) = \begin{cases} 2 + \omega, & -2 \leq \omega \leq -1; \\ 1, & -1 \leq \omega \leq 1; \\ 2 - \omega, & 1 \leq \omega \leq 2; \\ 0 & \text{otherwise.} \end{cases}$$

Using Fourier inversion, show that $f(t)$ can be expressed as

$$f(t) = \frac{1}{\pi}\left(\int_0^1 \cos(\omega t)d\omega + \int_1^2 (2-\omega)\cos(\omega t)d\omega\right)$$

and hence show that

$$f(t) = \frac{1}{\pi t^2}[\cos t - \cos(2t)].$$

(b) Using the definition of the Fourier transform, show that

$$\hat{F}_1[\cos(\omega_0 t)] = \pi[\delta(\omega - \omega_0) + \delta(\omega - \omega_0)].$$

(c) If $f(t)$ is the function whose spectrum $F(\omega)$ is defined in part (a) above, fined the effect on $F(\omega)$ of modulating $f(t)$ with $\cos(8t)$ and sketch the resulting graph of $\hat{F}_1[f(t)\cos(8t)]$. Briefly outline the process of demodulation designed to recover the original spectrum $F(\omega)$ and hence $f(t)$.

I.15 (a) (i) By considering the effect of the delta function $\delta(-t)$ on a function $f(t)$ in the sampling integral, show that $\delta(t)$ behaves as an even function. State the corresponding result for $\delta'(t)$.

(ii) Evaluate the integral

$$\int_{-\infty}^{\infty} \frac{e^t}{t}[\delta(3-t) - \delta'(3-t)]dt$$

(b)

(i) State the relationship between the unit step function $U(t)$ and $\delta(t)$.

(ii) Sketch the graph of the function $U(t-t^2)$ and express the function as the sum or difference of two unit step functions of the form $U(at+b)$. Hence show that

$$\delta(t-t^2) = \delta(t) + \delta(t-1)$$

and use the result to evaluate the integral

$$\int_{-\infty}^{\infty} \cos\left(\frac{\pi t}{3}\right) \delta(t-t^2)dt$$

(c) The comb function $\text{comb}_T(t)$ consists of an infinite series of delta functions at equally spaced intervals T of t and it can therefore be expressed as

$$\text{comb}_T(t) = \sum_{n=-\infty}^{\infty} \delta(t-nT)$$

Its Fourier transform is given by

$$\hat{F}_1[\text{comb}_T(t)] = \frac{2\pi}{T} \sum_{n=-\infty}^{\infty} \delta\left(\omega - \frac{2n\pi}{T}\right)$$

(i) A comb function is used to sample the function $g(t) = \cos t$ at intervals of $\pi/6$, the sampled function being denoted by $g_s(t)$. Explain how this is done and show the sampled points on a sketched graph of $\cos t$ in the range $0 \leq t \leq 2\pi$.

(ii) Use the product theorem to show that $\hat{F}_1[g_s(t)]$ is a periodic function in ω-space consisting of pairs of delta functions and state the period of the transform. Illustrate this transform by a sketched graph.

I.16 (a) Explain the meaning of the statement, 'the function $f(z)$ is analytic ate the point z_0 in the complex plane'.

Use the Cauchy-Riemann equations to show that the function $f(z) = z^2 + iz$ is analytic at all points in the complex plane.

(b) State Cauchy's theorem and use it to prove that if C is a circle, with centre z_0, entirely contained within a simple closed contour Γ, and $f(z)$ is a function which is analytic in the region between C and Γ and on the boundaries, then

$$\oint_{\Gamma} f(z)dz = \oint_{C} f(z)dz.$$

In this case, state the value of the integrals if C is a circle, with centre 2, and

$$f(z) = \frac{1}{(z-2)^3} + \frac{1}{z-2}$$

(c) Write down the general form of the principal part of the Laurent expansion of a function $g(z)$ about the point z_0 when z_0 is: (i) a simple pole; (ii) a double pole.

The function

$$g(z) = \frac{1}{(z^2+1)^2(z^2+4)}$$

has a double pole and a simple pole in the upper half of the complex plane. Find each of these poles, and show that the residue of $g(z)$ at this double pole is $-i/36$. Find the residue of $g(z)$ at the simple pole.

By integrating the function $g(z)$ around an appropriate large semi-circular contour, show that

$$\int_{-\infty}^{\infty} \frac{dx}{(x^2+1)^2(x^2+4)} = \frac{\pi}{9}$$

I.17 The periodic function $f_1(t)$ is defined on the interval $-\pi < t < \pi$ as

$$f_1(t) = \begin{cases} 0, & -\pi < t < 0; \\ t, & 0 \le t < \pi \end{cases}$$

and

$$f_1(t + 2\pi) = f_1(t)$$

(i) Sketch the graph of $y = f_1(t)$ over the interval $-3\pi < t < 3\pi$ and show that the Fourier series which represents $f_1(t)$ can be written as

$$f_1(t) = \frac{\pi}{4} - \frac{2}{\pi}\left(\cos t + \frac{\cos(3t)}{3^2} + \frac{\cos(5t)}{5^2} + ...\right)$$
$$+ \left(\sin t - \frac{\sin(2t)}{2} + \frac{\sin(3t)}{3} - \frac{\sin(4t)}{4} + ...\right)$$

(ii) By evaluating $f_1(t)$ and its Fourier series at a suitable value of t, show that

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ...$$

(iii) The function $f_2(t)$ is given by

$$f_2(t) = \begin{cases} t, & -\pi < t < 0 \\ 0 & 0 \le t < \pi \end{cases}$$

and

$$f_1(t + 2\pi) = f_1(t)$$

Use the Fourier series for $f_1(t)$ to show that

$$f_1(t) = \frac{\pi}{4} + \frac{2}{\pi}\left(\cos t + \frac{\cos(3t)}{3^2} + \frac{\cos(5t)}{5^2} + ...\right)$$

$$-\left(\sin t + \frac{\sin(2t)}{2} + \frac{\sin(3t)}{3} + \frac{\sin(4t)}{4} + ...\right)$$

(iv) A third period function $f_3(t)$ is defined as $f_3(t) = t$ for $-\pi < t < \pi$, the period again being 2π. Using part of the calculation already carried out in part (i) above, as appropriate, obtain the Fourier series for $f_3(t)$.

(v) For any function $f(t)$, defined on the interval $(-\pi, \pi)$, which can be expressed as a Fourier series with the usual notation, Parseval's identity states that

$$\frac{a_0^2}{2} + \sum_{n=1}^{\infty}(a_n^2 + b_n^2) = \frac{1}{\pi}\int_{-\pi}^{\pi} |\, f(t)\,|^2\, dt$$

Use Parseval's identity on each of the series representing $f_3(t)$ and $f_1(t)$ to show that

$$\frac{\pi^4}{96} = 1 + \frac{1}{3^4} + \frac{1}{5^4} + \frac{1}{7^4} + ...$$

[N.B. $\cos[n(t-\pi)] = (-1)^n \cos(nt)$ and $\sin[n(t-\pi)] = (-1)^n \sin(nt)$]

I.18 (a) The z-transform of the causal sequence $\{x_n\}, n = 0, 1, 2, 3, ...$ is defined as

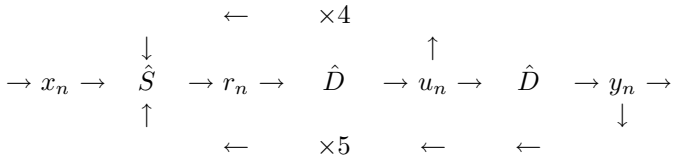
$$\hat{Z}\{x_n\} = X(z) = \sum_{n=0}^{\infty} \frac{x_n}{z^n}$$

(i) Using this definition, show that the z-transform of the sequence $\{1, -1, 1, -1, 1, ...\}$ is $z/(z+1)$ and state the region of convergence.

(ii) Find the z-transform of the sequence $\{0, -1, -3, 4, -5, ...\}$ and deduce that

$$\hat{Z}\{1, -2, 3, -4, 5, -6, ...\} = \frac{z^2}{(z+1)^2}$$

(b) The circuit diagram below represents a system into which signal pulses $\{x_n\}$ are fed at unit time intervals, so that $t = n$, and the resulting output is the sequence $\{y_n\}$. The $\hat{D}$ processes cause a delay of one time unit; the 'feedback' processes multiply the pulse passing through it by 4 and 5 respectively. The process $\hat{S}$ represents the summation of the signals meeting at this junction.



(i) Show that y_n satisfies the difference equation

$$y_{n+2} - 4y_{n+1} - 5y_n = x_n.$$

Use z-transforms to solve the equation for y_n when $\{x_n\} = \{6(-1)^n\}$, given that $y_0 = 0$ and $y_1 = 2$ and use your solution to check the given initial values y_0 and y_1.

(c) The output sequence $\{v_n\}$ of a linear system is given by the convolution of the input sequence $\{u_n\}$ and the impulse response sequence $\{h_n\}$, so that

$$u_n \otimes h_n = v_n$$

If the output $\{v_n\}$ is found to be $\{(-1)^n\}$ and $\{h_n\}$ is known to be $\{1, 1, 0, 0, 0, 0, ...\}$, use the convolution theorem for the z-transform to find $U(z)$ where $U(z) = \hat{Z}\{u_n\}$ and hence find the input sequence $\{u_n\}$.

(d) For a time invariant linear filter to be stable, the poles of its transfer function must lie in a unit circle. Use this result together with the z-transform to determine for what values of a the filter, governed by the difference equation below, is stable.

$$g_n = f_n + 2f_{n-1} + ag_{n-1} + 0.75g_{n-2}$$

I.19 (a) (i) Assuming that the sampling property of the delta function $\delta(t - t_0)$ is known, use formal integration by parts on the integral

$$\int_{-T}^{T} f(t)\delta'(t - t_0)dt$$

where T and t_0 are constant, $T > 0$ and $-T < t_0 < T$, to obtain the sampling property for the derived delta function $\delta'(t - t_0)$, i.e.

$$\int_{-\infty}^{\infty} f(t)\delta'(t - t_0)dt = -f'(t_0)$$

If $\delta'(t_0 - t)$ replaces $\delta'(t - t_0)$ in this integral, what is the effect on the result and why?

(ii) Evaluate the integral

$$\int_{-\infty}^{\infty} t^2 \sin\left(\frac{\pi t}{2}\right)[\delta(t - 3) + \delta'(4 - t)]dt$$

(b) (i) State the relationship between $\delta(t - t_0)$ and $U(t - t_0)$, where $U(t)$ is the unit step function.

(ii) Show that $U(t^3 - t)$ can be expressed as a linear combination of the three step functions $U(t + 1)$, $U(t)$, $U(t - 1)$ and hence show that

$$\delta(t^3 - t) = \frac{\delta(t + 1)}{2} + \delta(t) + \frac{\delta(t - 1)}{2}$$

(iii) Use this result to show that $\hat{F}_1[\delta(t^3 - t)] = 1 + \cos\omega$, where $\hat{F}_1$ is the Fourier transform operator.

(iv) If $H(\omega)$ is the top-hat function

$$H(\omega) = \begin{cases} 1, & -\pi < \omega < \pi; \\ 0, & \text{otherwise} \end{cases}$$

sketch the graph of the product

$$P(\omega) = (1 + \cos\omega)H(\omega)$$

and use the convolution theorem to show that

$$\hat{F}_1^{-1}[P(\omega)] = \sin(\pi t)\left(\frac{2}{t} - \frac{1}{t-1} - \frac{1}{t+1}\right)$$

[N.B. $\hat{F}_1^{-1}[H(\omega)] = 2\sin(\pi t)/t$]

I.20 (a) The Fourier transform of $Y(\omega)$ of a function $y(t)$ is known to be

$$Y(\omega) = \exp(-2 \mid \omega \mid) = \begin{cases} \exp(2\omega), & \omega < 0; \\ \exp(-2\omega), & \omega > 0. \end{cases}$$

(i) Invert this transform to show that

$$y(t) = \frac{2}{\pi(t^2 + 4)}$$

(ii) By taking the Fourier transform of $y(t)$, deduce that

$$\int_{-\infty}^{\infty} \frac{dt}{t^2 + 4} = \frac{\pi}{4}$$

and evaluate the integral

$$\int_{-\infty}^{\infty} \frac{\cos(3t)}{t^2 + 4} dt$$

(b) The comb function is defined as

$$\text{comb}_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

(i) Given an analogue signal $f(t)$, explain how the comb function is used to obtain a sampled version $g(t)$ of the signal, sampling being taken at equally spaced intervals T.

(ii) Given that the comb function can be expressed as a complex Fourier series as

$$\frac{1}{T}\sum_{n=-\infty}^{\infty}\exp\left(\frac{i2\pi nt}{T}\right),$$

show that the Fourier transform of comb_T can be expressed as

$$\hat{F}_1[\text{comb}_T] = \frac{2\pi}{T}\sum_{n=-\infty}^{\infty}\delta\left(\omega - \frac{2\pi n}{T}\right)$$

(iii) If $F(\omega)$ and $G(\omega)$ are respectively the Fourier transforms of $f(t)$ and $g(t)$, use the product theorem to show that

$$G(\omega) = \frac{1}{T}\sum_{n=-\infty}^{\infty}F\left(\omega - \frac{2\pi n}{T}\right)$$

(iv) $f(t)$ is a bandlimited function whose spectrum $F(\omega)$ is defined by

$$F(\omega) = \begin{cases} \frac{1}{|\omega|}, & |\,\omega\,| \leq \Omega; \\ 0, & \text{otherwise.} \end{cases}$$

Sketch the graph of $G(\omega)$ to illustrate the case in which T has been chosen at the ideal (Nyquist) frequency and use your graph to obtain the relationship between T and Ω.

I.21 (a) Consider the z-transform

$$X(z) = 4z^2 + 2 + 3^{-1}, 0 <| z |< \infty$$

Determine the inverse transform of $X(z)$.

(b) Prove that

$$x_n \otimes u_{n-m} = \sum_{k=-\infty}^{n-m} x_k$$

where

$$u_n = \begin{cases} 1, & n \geq 0; \\ 0, & n < 0 \end{cases}$$

and that if

$$x_n \leftrightarrow X(z)$$

then

$$nx_n \leftrightarrow -z\frac{d}{dz}X(z)$$

(c) Compute $y_n = x_n \otimes h_n$ where

$$x_n = a^n u_n \quad \text{and} \quad h_n = b^n u_n$$

noting that a and b could have the same value. Hint: use the relationship

$$\left(\frac{1}{1-az^{-1}}\right)^2 = \frac{z^2}{a}\frac{d}{dz}\left(\frac{1}{1-az^{-1}}\right)$$

I.22 Consider a linear time invariant system that is characterised by the difference equation

$$y_n - \frac{3}{4}y_{n-1} + \frac{1}{8}y_{n-2} = 2x_n$$

(a) Using the DFT, determine the frequency response and impulse response of the system.

(b) Determine:

(i) whether the system is causal;

(ii) whether the system is stable.

(c) Consider an input to the system given by

$$x_n = \left(\frac{1}{4}\right)^n u_n$$

where

$$u_n = \begin{cases} 1, & n \geq 0; \\ 0, & n < 0. \end{cases}$$

Use the DFT to determine the output of the system.

A.2 Part II

A.2.1 Solutions to Problems Given in Chapter 6

6.1 (i) The solution is $\mathbf{x} = (\text{adj}A\mathbf{b})/\det A$ and the equations are linearly dependent if $(\text{adj})\mathbf{b} = 0$ and $\det A = 0$. Using Cramers rule:

$$x = \frac{1}{|\,A\,|}\begin{vmatrix} 1 & 1 & 1 \\ 2 & -2 & 1 \\ -3 & 1 & -2 \end{vmatrix} = 0, \quad y = \frac{1}{|\,A\,|}\begin{vmatrix} -2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & -3 & -2 \end{vmatrix} = 0,$$

$$z = \frac{1}{|\,A\,|}\begin{vmatrix} 1 & 1 & 1 \\ 2 & -2 & 1 \\ -3 & 1 & -2 \end{vmatrix} = 0.$$

Also,

$$|\,A\,| = \begin{vmatrix} -2 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 1 & -2 \end{vmatrix} = 0$$

Therefore, the equations are linearly dependent. Note, that the sum of the first and second equations are equal to the negation of the third equation so that in reality, there are only two equations for three unknowns, namely

$$x - 2y + z = 2, \quad x + y - 2z = -3.$$

Subtracting the second from the first of these equations, we have $-3y+3z = 5, \quad \therefore \quad y = z - 5/3$ and $x = z - 4/3$. Hence, if we let $z = \lambda$ where λ is any constant, then the infinity of solutions is given by

$$x = \lambda - \frac{4}{3}, \quad y = \lambda - \frac{5}{3}, \quad z = \lambda.$$

(ii) A non-trivial solution to a homogeneous system of linear equations exists when $\det A = 0$, i.e. when

$$\begin{vmatrix} 1 & 5 & 3 \\ 5 & 1 & -k \\ 1 & 2 & k \end{vmatrix} = 27(1-k) = 0, \quad \therefore \quad k = 1.$$

The equations are linearly dependent since

$$\frac{8}{3} \times \text{Eq. (3)} - \frac{1}{3} \times \text{Eq. (2)} = \text{Eq. (1)}.$$

Therefore there are only two equations for three unknowns. Taking the second and third equations with $k = 1$, we get $y = -2x, z = 3x$. Let $x = -\lambda$ where λ is any constant parameter, then the infinity of solutions becomes $x = -\lambda, \quad y = 2\lambda, \quad z = -3\lambda$.

6.2 (i) The equations given can be written in the form

$$\begin{aligned} (1-\lambda)x_1 + 2x_2 + x_3 &= 0, \\ 2x_1 + (1-\lambda)x_2 + x_3 &= 0, \\ x_1 + x_2 + (2-\lambda)x_3 &= 0. \end{aligned}$$

The set of homogeneous equations will have non-trivial solutions provided that,

$$\begin{vmatrix} 1-\lambda & 2 & 1 \\ 2 & 1-\lambda & 1 \\ 1 & 1 & 2-\lambda \end{vmatrix} = \begin{vmatrix} 4-\lambda & 4-\lambda & 4-\lambda \\ 2 & 1-\lambda & 1 \\ 1 & 1 & 2-\lambda \end{vmatrix} = 0,$$

i.e.

$$(4-\lambda)\begin{vmatrix} 1 & 1 & 1 \\ 2 & 1-\lambda & 1 \\ 1 & 1 & 2-\lambda \end{vmatrix} = (4-\lambda)(\lambda-1)(\lambda+1) = 0.$$

Hence, the only possible values of λ are 4, 1 and -1. With $\lambda = 1$ the equations become

$$\begin{aligned} 2x_2 + x_3 &= 0, \\ 2x_1 + x_3 &= 0, \\ x_1 + x_2 + x_3 &= 0, \end{aligned}$$

which has a general solution given by

$$x_1 = x_2 = -\frac{1}{2}x_3.$$

(ii) The set of equations can be written in the form

$$\begin{aligned} x_1 - x_2 + 2x_3 - x_4 &= 1, \\ x_2 - x_3 - 2x_4 &= 0, \\ 3x_3 + 2x_4 &= 0. \end{aligned}$$

Thus, if $x_4 = \lambda$ (an arbitrary constant) then $x_3 - 2\lambda/3$, $x_2 = 4\lambda/3$ and $x_1 = 1 + 11\lambda/3$.

(iii) The system of equations given is equivalent to the system

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &= 0, \\ -5x_2 + 8x_3 &= 0, \\ 5x_2 - 8x_3 &= 0. \end{aligned}$$

If $x_3 = \lambda$ then $x_2 = 8\lambda/5$ and $x_1 = -\lambda/5$. Hence, $x_1^2+x_2^2+x_3^2 = \lambda^2(1/25+64/25+1) = 18\lambda^2/5$. A solution which also satisfies $x_1^2 + x_2^2 + x_3^2 = 1$ will be obtained if we put $\lambda = \pm\frac{1}{3}\sqrt{5/2}$.

6.3

$$\begin{array}{lll} \delta = -0.02; & x = -151, & y = 100; \\ \delta = -0.01; & \text{No solution}; & \\ \delta = -0; & x = 153, & y = -100; \\ \delta = -0.01; & x = 77, & y = -50; \\ \delta = -0.02; & x = 51\frac{2}{3}, & y = -33\frac{1}{3}. \end{array}$$

6.4 For over-determined systems $A\mathbf{x} = \mathbf{b}$, the least squares method is based on computing a solution vector $\mathbf{x}$ such that $\|A\mathbf{x} - \mathbf{b}\|_2^2$ is a minimum. In this case,

$$e = \|A\mathbf{x} - \mathbf{b}\|_2^2 = \sum_{i=1}^{3}(a_{i1}x_1 + a_{i2}x_2 - b_i)^2.$$

Here, e is a function of $x_i; i = 1, 2$ and is therefore a minimum when

$$\frac{\partial e}{\partial x_i} = 0, \quad i = 1, 2.$$

Thus,

$$\frac{\partial}{\partial x_1}e(x_1, x_2) = 2\sum_{i=1}^{3} a_{i1}(a_{i1} + a_{i2} - b_i) = 0,$$

$$\frac{\partial}{\partial x_2}e(x_1, x_2) = 2\sum_{i=1}^{3} a_{i2}(a_{i1}x_1 + a_{i2}x_2 - b_i) = 0.$$

From these equations, we get

$$a_{11}(a_{11}x_1 + a_{12}x_2 - b_1) + a_{21}(a_{21}x_1a_{22}x_2 - b_2) + a_{31}(a_{31}x_1 + a_{32}x_2 - b_3) = 0$$

and

$$a_{12}(a_{11}x_1 + a_{12}x_2 - b_1) + a_{22}(a_{21}x_1a_{22}x_2 - b_2) + a_{32}(a_{31}x_1 + a_{32}x_2 - b_3) = 0.$$

From the first set of equation given, we have

$$\begin{array}{rl} 2(2x + 3y - 8) + 3(3x - y - 1) + (x + y - 4) & = 0, \\ 3(2x + 3y - 8) - (3x - y - 1) + (x + y - 4) & = 0, \end{array}$$

or, after collecting terms,

$$\begin{array}{rcl} 14x + 4y & = & 23, \\ 4x + 11y & = & 27. \end{array}$$

Using Cramers rule:

$$\frac{\begin{vmatrix} 23 & 4 \\ 24 & 11 \end{vmatrix}}{\begin{vmatrix} 23 & 4 \\ 24 & 11 \end{vmatrix}} = \frac{145}{138}; \quad \frac{\begin{vmatrix} 14 & 23 \\ 4 & 27 \end{vmatrix}}{\begin{vmatrix} 14 & 4 \\ 4 & 11 \end{vmatrix}} = \frac{286}{138}.$$

From the second set of equations, we get

$$\begin{array}{rl} (x - y - 2) + (x + y - 4) + 2(2x + y - 8) & = 0, \\ -(x - y - 2) + (x + y - 4) - (2x + y - 8) & = 0, \end{array}$$

or

$$\begin{array}{rcl} 3x + y & = & 11, \\ 2x + 3y & = & 10, \end{array}$$

and using Cramers rule:

$$\frac{\begin{vmatrix} 11 & 1 \\ 10 & 3 \end{vmatrix}}{\begin{vmatrix} 3 & 1 \\ 2 & 3 \end{vmatrix}} = \frac{23}{7}; \quad \frac{\begin{vmatrix} 3 & 11 \\ 2 & 10 \end{vmatrix}}{\begin{vmatrix} 3 & 1 \\ 2 & 3 \end{vmatrix}} = \frac{8}{7}.$$

In the first set of equations, the 'residuals' are given by

$$r_1 = 2 \times \frac{145}{138} + 3 \times \frac{286}{138} - 8 = \frac{44}{138}; \quad r_2 = 3 \times \frac{145}{138} - \frac{286}{138} - 1 = \frac{11}{138}$$

and

$$r_3 = \frac{145}{138} + \frac{286}{138} - 4 = -\frac{121}{138}.$$

Now,

$$\text{RMS} \quad \text{error} = \left(\frac{r_1^2 + r_2^2 + r_3^2}{3}\right)^{\frac{1}{2}} = \left(\frac{16698}{57132}\right)^{\frac{1}{2}} = \left(\frac{2783}{9522}\right)^{\frac{1}{2}}.$$

In the second system of equations, the residuals are:

$$r_1 = \frac{23}{7} - \frac{8}{7} - 2 = \frac{1}{7}; \quad r_2 = \frac{23}{7} + \frac{8}{7} - 4 = \frac{3}{7}$$

and

$$r_3 = 2 \times \frac{23}{7} + \frac{8}{7} - 8 = -\frac{2}{7}.$$

$$\therefore \text{ RMS error} = \left(\frac{14}{147}\right)^{\frac{1}{2}} = \sqrt{\frac{2}{21}}.$$

where RMS stands for Root Mean Square.

A.2.2 Solutions to Problems Given in Chapter 7

7.1 Using the appropriate augmented matrix, we have

$$\left(\begin{array}{cccc|c} 1 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 1 & 0 & -1 & 0 & 2 \\ 1 & 1 & 1 & 1 & 1 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_3 - R_1 \\ R_4 - R_1 \end{array} \left(\begin{array}{cccc|c} 1 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & 2 & 1 & 1 & 0 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_3 - R_2 \\ R_4 - 2R_2 \end{array} \left(\begin{array}{cccc|c} 1 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & 2 \\ 0 & 0 & 1 & -1 & 2 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ \\ \\ R_4 + R_3 \end{array} \left(\begin{array}{cccc|c} 1 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & 2 \\ 0 & 0 & 0 & -2 & 4 \end{array}\right)$$

Back-substituting,

$$\begin{array}{rcrcrcl} -2x_4 & = & 4, & \Longrightarrow & x_4 & = & -2; \\ -x_3 - x_4 & = & 2, & \Longrightarrow & x_3 & = & 0; \\ x_2 + x_4 & = & -1, & \Longrightarrow & x_2 & = & 1; \\ x_1 - x_2 & = & 1, & \Longrightarrow & x_1 & = & 2. \end{array}$$

7.2 (i) With natural pivoting:

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & -3 \\ 3 & 7 & 2 & 1 \\ 4 & -2 & 1 & -2 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_2 - 3R_1 \\ R_3 - 4R_1 \end{array} \left(\begin{array}{ccc|c} 1 & 2 & -1 & -3 \\ 0 & 1 & 5 & 10 \\ 0 & -10 & 5 & 10 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ \\ R_3 + 10R_2 \end{array} \left(\begin{array}{ccc|c} 1 & 2 & -1 & -3 \\ 0 & 1 & 5 & 10 \\ 0 & 0 & 55 & 110 \end{array}\right).$$

Back-substituting:

$$\begin{array}{rclcrcl} 55x_3 & = & 110, & \Longrightarrow & x_3 & = & 2; \\ x_2 - 10 & = & 10, & \Longrightarrow & x_2 & = & 0; \\ x_1 - 2 & = & -3, & \Longrightarrow & x_1 & = & -1. \end{array}$$

(ii) Partial pivoting:

$$\left(\begin{array}{cccc|c} 1 & 2 & -1 & | & -3 \\ 3 & 7 & 2 & | & 1 \\ 4 & -2 & 1 & | & -2 \end{array}\right) \longrightarrow \begin{array}{c}\text{Interchange} \\ R_3 \& R_1\end{array} \left(\begin{array}{cccc|c} 4 & -2 & 1 & | & -2 \\ 3 & 7 & 2 & | & 1 \\ 1 & 2 & -1 & | & -3 \end{array}\right) \longrightarrow$$

$$\begin{array}{l} R_2 - \frac{3}{4}R_1, \times 4 \\ R_3 - \frac{1}{4}R_1, \times 4 \end{array} \left(\begin{array}{cccc|c} 4 & -2 & 1 & | & -2 \\ 0 & 34 & 5 & | & 10 \\ 0 & 10 & -5 & | & -10 \end{array}\right) \longrightarrow$$

$$R_3 - \frac{10}{34}R_2, \times 7 \left(\begin{array}{cccc|c} 4 & -2 & 1 & | & -2 \\ 0 & 34 & 5 & | & 10 \\ 0 & 0 & -110 & | & -220 \end{array}\right).$$

Back-substituting:

$$\begin{array}{rclcrcl} -110x_3 & = & -220, & \Longrightarrow & x_3 & = & 2; \\ 34x_2 + 10 & = & 10, & \Longrightarrow & x_2 & = & 0; \\ 4x_1 + 2 & = & -2, & \Longrightarrow & x_1 & = & -1. \end{array}$$

7.3 (i) With natural pivoting

$$\left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & | & 10 \\ -2 & 1 & 4 & 1 & | & 11 \\ 1 & -2 & -3 & 2 & | & -9 \\ 3 & -3 & -5 & -2 & | & -17 \end{array}\right) \longrightarrow \begin{array}{c} R_2 + 2R_1 \\ R_3 - R_1 \\ R_3 - 3R_1 \end{array} \left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & | & 10 \\ 0 & 7 & 8 & -7 & | & 31 \\ 0 & -5 & -5 & 6 & | & -19 \\ 0 & -12 & -11 & 10 & | & -47 \end{array}\right) \longrightarrow$$

$$\begin{array}{l} R_3 + \frac{5}{7}R_1, \times 7 \\ R_4 + \frac{12}{7}R_2, \times 7 \end{array} \left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & | & 10 \\ 0 & 7 & 8 & -7 & | & 31 \\ 0 & 0 & 5 & 7 & | & 22 \\ 0 & 0 & 19 & -14 & | & 43 \end{array}\right) \longrightarrow$$

$$R_4 - \frac{19}{5}R_3, \times 5 \left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & | & 10 \\ 0 & 7 & 8 & -7 & | & 31 \\ 0 & 0 & 5 & 7 & | & 22 \\ ) & 0 & 0 & -203 & | & -203 \end{array}\right).$$

Back-substitution gives the solution vector $(2, 2, 3, 1)^T$.

(ii) Using partial pivoting (pivot given in bold face)

$$\left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & | & 10 \\ -2 & 1 & 4 & 1 & | & 11 \\ 1 & -2 & -3 & 2 & | & -9 \\ \mathbf{3} & -3 & -5 & -2 & | & -17 \end{array}\right) \longrightarrow$$

$$\begin{matrix} R_1 \\ \uparrow \\ \downarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ -2 & 1 & 4 & 1 & 11 \\ 1 & -2 & -3 & 2 & -9 \\ 1 & 3 & 2 & -4 & 10 \end{array} \right) \longrightarrow$$

$$\begin{matrix} \\ R_2 + \frac{2}{3}R_1, \times 3 \\ R_3 - \frac{1}{3}R_1, \times 3 \\ R_4 - \frac{1}{3}R_1, \times 3 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 0 & -3 & 2 & -1 & -1 \\ 0 & -3 & -4 & 8 & -10 \\ 0 & \mathbf{12} & 11 & -10 & 47 \end{array} \right) \longrightarrow$$

$$\begin{matrix} R_2 \\ \updownarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 0 & 12 & 11 & -10 & 47 \\ 0 & -3 & -4 & 8 & \vert -10 \\ 0 & -3 & 2 & -1 & -1 \end{array} \right) \longrightarrow$$

$$\begin{matrix} \\ R_3 + \frac{3}{12}R_2, \times 12 \\ R_4 + \frac{3}{12}R_2, \times 12 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 0 & 12 & 11 & -10 & 47 \\ 0 & 0 & -15 & 66 & 21 \\ 0 & 0 & \mathbf{57} & -42 & 129 \end{array} \right) \longrightarrow$$

$$\begin{matrix} R_3 \\ \updownarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 0 & 12 & 11 & -10 & 47 \\ 0 & 0 & 57 & -42 & 129 \\ 0 & 0 & -15 & 66 & 21 \end{array} \right) \longrightarrow$$

$$\begin{matrix} \\ \\ \\ R_4 + \frac{15}{57}R_3, \times 57 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 0 & 12 & 11 & -10 & 47 \\ 0 & 0 & 57 & -42 & 129 \\ 0 & 0 & 0 & 3132 & 3132 \end{array} \right).$$

Back substitution gives the solution vector $(2, 2, 3, 1)^T$.

(iii) With full pivoting (pivot given in bold face):

$$\left(\begin{array}{cccc|c} 1 & 3 & 2 & -4 & 10 \\ -2 & 1 & 4 & 1 & 11 \\ 1 & -2 & -3 & 2 & -9 \\ 3 & -3 & \mathbf{-5} & -2 & -17 \end{array} \right) \longrightarrow$$

$$\begin{matrix} R_1 \\ \uparrow \\ \downarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ -2 & 1 & 4 & 1 & 11 \\ 1 & -2 & -3 & 2 & -9 \\ 1 & 3 & 2 & -4 & 10 \end{array} \right) \longrightarrow$$

$$\begin{matrix} \\ R_2 + \frac{4}{5}R_1, \times 5 \\ R_3 - \frac{3}{5}r_1, \times 5 \\ R_4 + \frac{2}{5}R_1, \times 5 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 2 & -7 & 0 & -3 & -13 \\ -4 & -1 & 0 & 16 & 6 \\ 11 & 9 & 0 & \mathbf{-24} & 16 \end{array} \right) \longrightarrow$$

$$\begin{matrix} \\ R_2 \\ \updownarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 11 & 9 & 0 & -24 & 16 \\ -4 & -1 & 0 & 16 & 6 \\ 2 & -7 & 0 & -3 & -13 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ R_3 + \frac{16}{24}R_2, \times 24 \\ R_4 - \frac{3}{24}R_2, \times 24 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 11 & 9 & 0 & -24 & 16 \\ 80 & 120 & 0 & 0 & 400 \\ 15 & \mathbf{-195} & 0 & 0 & -360 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ R_3 \\ \updownarrow \\ R_4 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 11 & 9 & 0 & -24 & 16 \\ 15 & -195 & 0 & 0 & -360 \\ 80 & 120 & 0 & 0 & 400 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ \\ \\ R_4 + \frac{120}{195}R_3 \end{matrix} \left(\begin{array}{cccc|c} 3 & -3 & -5 & -2 & -17 \\ 11 & 9 & 0 & -24 & 16 \\ 15 & -195 & 0 & 0 & -360 \\ 17400 & 0 & 0 & 0 & 34800 \end{array}\right).$$

Back-substituting yields the solution vector $(2, 2, 1, 3)^T$.

7.4 (i)

$$\left(\begin{array}{ccc|ccc} 3 & 2 & -1 & 1 & 0 & 0 \\ 1 & -1 & 2 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ R_2 - \frac{1}{3}R_1, \times 3 \\ R_3 - \frac{2}{3}R_1, \times 3 \end{matrix} \left(\begin{array}{ccc|ccc} 3 & 2 & -1 & 1 & 0 & 0 \\ 0 & -5 & 7 & -1 & 3 & 0 \\ 0 & -1 & 5 & -2 & 0 & 3 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ R_3 - \frac{1}{5}R_2, \times 5 \end{matrix} \left(\begin{array}{ccc|ccc} 3 & 2 & -1 & 1 & 0 & 0 \\ 0 & -5 & 7 & -1 & 3 & 0 \\ 0 & 0 & 18 & -9 & -3 & 15 \end{array}\right) \longrightarrow$$

$$\begin{matrix} \\ R_1 + \frac{2}{5}R_2, \times 5 \end{matrix} \left(\begin{array}{ccc|ccc} 15 & 0 & 9 & 3 & 6 & 0 \\ 0 & -5 & 7 & -1 & 3 & 0 \\ 0 & 0 & 18 & -9 & -3 & 15 \end{array}\right) \longrightarrow$$

$$\begin{matrix} R_1 - \frac{9}{18}R_3, \times 2 \\ R_2 - \frac{7}{18}R_3, \times 6 \\ \\ \end{matrix} \left(\begin{array}{ccc|ccc} 30 & 0 & 0 & 15 & 15 & -15 \\ 0 & -30 & 0 & 15 & 25 & -35 \\ 0 & 0 & 18 & -9 & -3 & 15 \end{array}\right) \longrightarrow$$

$$\begin{matrix} {}_1/30 \\ -R_2/30 \\ R_3/30 \end{matrix} \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/2 & 1/2 & -1/2 \\ 0 & 1 & 0 & -1/2 & -5/6 & 7/6 \\ 0 & 0 & 1 & -1/2 & -1/6 & 5/6 \end{array}\right).$$

(ii)

$$\left(\begin{array}{ccc|ccc} 1 & 1/2 & 1/3 & 1 & 0 & 0 \\ 1/2 & 1/3 & 1/4 & 0 & 1 & 0 \\ 1/3 & 1/4 & 1/5 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} R_2 - \frac{1}{2}R_1, \times 12 \\ R_3 - \frac{1}{3}R_1, \times 12 \end{array} \left(\begin{array}{ccc|ccc} 1 & 1/2 & 1/3 & 1 & 0 & 0 \\ 0 & 1 & 1 & -6 & 12 & 0 \\ 0 & 1 & 48/45 & -4 & 0 & 12 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} \\ R_3 - R_2, \times \frac{1}{12} \end{array} \left(\begin{array}{ccc|ccc} 1 & 1/2 & 1/3 & 1 & 0 & 0 \\ 0 & 1 & 1 & -6 & 12 & 0 \\ 0 & 0 & 1/180 & 1/6 & -1 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} R_1 + \frac{180}{3}R_3 \\ R_2 - 180R_3 \\ \\ \end{array} \left(\begin{array}{ccc|ccc} 1 & 1/2 & 0 & -9 & 60 & -60 \\ 0 & 1 & 0 & -36 & 192 & -180 \\ 0 & 0 & 1/180 & 1/6 & -1 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} R_1 - \frac{1}{2}R_2 \\ \\ R_3 \times 100 \end{array} \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 9 & -36 & 30 \\ 0 & 1 & 0 & -36 & 192 & -180 \\ 0 & 0 & 18 & 30 & -180 & 180 \end{array}\right).$$

$$\therefore \quad A^{-1} = \left(\begin{array}{ccc} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{array}\right).$$

Observe that the inverse matrix is symmetric as is the original matrix. Also, observe that relative small values in A start to produce relatively large numbers in A^{-1}, a characteristic of Hilbert matrices.

7.5

$$\left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 4 & -1 & 5 & 0 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} \\ R_2 + R_1 \\ R_3 - 2R_1 \\ R_4 - R_1 \end{array} \left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & -2 & -2 & 0 & 1 & 0 \\ 0 & 2 & -1 & -4 & -1 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} \\ R_3 + R_2 \\ R_4 - 2R_2 \end{array} \left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 \\ 0 & 0 & -3 & 2 & -3 & -2 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} \\ \\ R_4 + 3R_2, \times(-1) \end{array} \left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 6 & -1 & -3 & -1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} R_1 - R_4 \\ R_2 - R_4 \\ R_3 + R_4 \\ \\ \end{array} \left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 1 & -5 & 1 & 3 & 1 \\ 0 & 1 & 1 & 0 & -5 & 2 & 3 & 1 \\ 0 & 0 & 1 & 1 & 5 & 0 & -2 & -1 \\ 0 & 0 & 0 & 1 & 6 & -1 & -3 & -1 \end{array}\right) \longrightarrow$$

$$\begin{array}{r} R_2 - R_3 \\ \\ \\ \\ \end{array} \left(\begin{array}{cccc|cccc} 1 & 2 & 0 & 0 & -5 & 1 & 3 & 1 \\ 0 & 1 & 0 & 0 & -10 & 2 & 5 & 2 \\ 0 & 0 & 1 & 0 & 5 & 0 & -2 & -1 \\ 0 & 0 & 0 & 1 & 6 & -1 & -3 & -1 \end{array}\right) \longrightarrow$$

$$R_1 - 2R_2 \quad \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 15 & -3 & -7 & -3 \\ 0 & 1 & 0 & 0 & -10 & 2 & 5 & 2 \\ 0 & 0 & 1 & 0 & 5 & 0 & -2 & -1 \\ 0 & 0 & 0 & 1 & 6 & -1 & -3 & -1 \end{array}\right) = [I|A^{-1}].$$

Check that $AA^{-1} = I$, i.e.

$$\left(\begin{array}{cccc} 1 & 2 & 0 & 1 \\ -1 & -1 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 4 & -1 & 5 \end{array}\right)\left(\begin{array}{cccc} 15 & -3 & -7 & -3 \\ -10 & 2 & 5 & 2 \\ 5 & 0 & -2 & -1 \\ 6 & -1 & -3 & -1 \end{array}\right) = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right).$$

The solution to the linear equations given is

$$\left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right) = \left(\begin{array}{cccc} 15 & -3 & -7 & -3 \\ -10 & 2 & 5 & 2 \\ 5 & 0 & -2 & -1 \\ 6 & -1 & -3 & -1 \end{array}\right)\left(\begin{array}{c} 2 \\ 1 \\ 5 \\ 0 \end{array}\right).$$

In cases when $A\mathbf{x}_i = b_i, \quad i = 1, 2, ..., n$ it becomes convenient (i.e. computationally efficient) to compute A^{-1} using Jordan's method. A^{-1} can then be used repeatedly to obtain the solutions $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, i.e.

$$\mathbf{x}_i = A^{-1}\mathbf{b}, \quad i = 1, 2, ..., n.$$

7.6

$$A = LU_1 = \left(\begin{array}{ccc} 2 & 1 & -3 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{array}\right) = \left(\begin{array}{ccc} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell 22 & 0 \\ \ell_[31 & \ell_{32} & \ell_{33} \end{array}\right)\left(\begin{array}{ccc} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{array}\right)$$

$$= \left(\begin{array}{ccc} 2 & 0 & 0 \\ 1 & -5/2 & 0 \\ 1 & 1/2 & 13/5 \end{array}\right)\left(\begin{array}{ccc} 1 & 1/2 & -3/2 \\ 0 & 1 & -1/5 \\ 0 & 0 & 1 \end{array}\right).$$

Solve $L\mathbf{y} = \mathbf{b}$ by forward substitution:

$$\left(\begin{array}{ccc} 2 & 0 & 0 \\ 1 & -5/2 & 0 \\ 1 & 1/2 & 13/5 \end{array}\right)\left(\begin{array}{c} y_1 \\ y_2 \\ y_3 \end{array}\right) = \left(\begin{array}{c} -5 \\ -6 \\ 6 \end{array}\right) \Longrightarrow \mathbf{y} = \left(\begin{array}{c} -5/2 \\ 7/2 \\ 3 \end{array}\right).$$

Solving $U_1\mathbf{x} = \mathbf{y}$ by back-substitution,

$$\left(\begin{array}{ccc} 1 & 1/2 & -3/2 \\ 0 & 1 & -1/5 \\ 0 & 0 & 1 \end{array}\right)\left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right) = \left(\begin{array}{c} -5/2 \\ 7/5 \\ 3 \end{array}\right) \Longrightarrow \mathbf{y} = \left(\begin{array}{c} 1 \\ 2 \\ 3 \end{array}\right).$$

7.7

$$A = CC^T = \left(\begin{array}{ccc} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{array}\right)\left(\begin{array}{ccc} c_{11} & c_{21} & c_{31} \\ 0 & c_{22} & c_{32} \\ 0 & 0 & c_{33} \end{array}\right)$$

$$= \begin{pmatrix} \sqrt{2} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & 0 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \sqrt{\frac{3}{2}} & -\sqrt{\frac{2}{3}} \\ 0 & 0 & \sqrt{\frac{4}{3}} \end{pmatrix}$$

using exact arithmetic. Solving $C\mathbf{y} = \mathbf{b}$ by forward substitution,

$$\begin{pmatrix} \sqrt{2} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & 0 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \Longrightarrow \mathbf{y} = \begin{pmatrix} 1/\sqrt{2} \\ \frac{1}{2}\sqrt{2/3} \\ \frac{4}{3}\sqrt{3/4} \end{pmatrix}.$$

Solving $C^T\mathbf{x} = \mathbf{y}$ by back-substitution,

$$\begin{pmatrix} \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \sqrt{\frac{3}{2}} & -\sqrt{\frac{2}{3}} \\ 0 & 0 & \sqrt{\frac{4}{3}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} \\ \frac{1}{2}\sqrt{2/3} \\ \frac{4}{3}\sqrt{3/4} \end{pmatrix} \Longrightarrow \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

7.8

$$A = L_1 U$$

where

$$L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & -\frac{2}{3} & 1 & 0 \\ 0 & 0 & -\frac{3}{4} & 1 \end{pmatrix}, \text{ and } U = \begin{pmatrix} 2 & -1 & 0 & 0 \\ 0 & \frac{3}{2} & -1 & 0 \\ 0 & 0 & \frac{4}{3} & -1 \\ 0 & 0 & 0 & \frac{5}{4} \end{pmatrix}.$$

Solving $L_1\mathbf{y} = \mathbf{b}$ we have $\mathbf{y} = (1, 5/2, 14/3, 15/2)^T$ and solving $U\mathbf{x} = \mathbf{y}$ we get $\mathbf{x} = (4, 7, 8, 6)^T$.

7.9

$$A = \begin{pmatrix} \sqrt{2} & 0 & 0 & 0 \\ -1/\sqrt{2} & \sqrt{3/2} & 0 & 0 \\ 0 & -\sqrt{2/3} & \sqrt{4/3} & 0 \\ 0 & 0 & -\sqrt{3/4} & \sqrt{5/4} \end{pmatrix} \begin{pmatrix} \sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & \sqrt{3/2} & -\sqrt{2/3} & 0 \\ 0 & 0 & \sqrt{4/3} & -\sqrt{3/4} \\ 0 & 0 & 0 & \sqrt{5/4} \end{pmatrix}.$$

Solving $C\mathbf{y} = \mathbf{b}$ gives

$$\mathbf{y} = \left(\frac{1}{\sqrt{2}}, \frac{5}{2}\sqrt{\frac{2}{3}}, \frac{14}{3}\sqrt{\frac{3}{4}}, \frac{30}{4}\sqrt{\frac{4}{3}}\right)^T.$$

Solving for $C^T\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ gives $\mathbf{x} = (4, 7, 8, 6)^T$.

7.10 Either the Crout or Doolittle method may be used. Using the latter, we get

$$A = L_1 U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ 2 & -1 & \frac{3}{4} & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & -3 & 1 \\ 0 & 1 & 2 & 4 \\ 0 & 0 & 4 & 20 \\ 0 & 0 & 0 & 2 \end{pmatrix}.$$

Solving $L_1\mathbf{y} = (-4, -1, 53, -8)^T$ gives $\mathbf{y} = (-4, 7, 28, 2)^T$ and solving $U\mathbf{x} = \mathbf{y}$ gives $\mathbf{x} = (3, -1, 2, 1)^T$.

A.2.3 Solutions to Problems Given in Chapter 8

8.1 Using Jordan's method for matrix inversion,

$$\left(\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 2 & 5 & -3 & 0 & 1 & 0 \\ -1 & -1 & 8 & 0 & 0 & 1 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_2 - 2R_1 \\ R_3 + R_1 \end{array} \left(\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & -1 & -5 & -2 & 1 & 0 \\ 0 & 2 & 9 & 1 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_3 + 2R_2 \end{array} \left(\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & -1 & -5 & -2 & 1 & 0 \\ 0 & 0 & -1 & -3 & 2 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_2 - 5R_3 \\ R_1 + R_3 \end{array} \left(\begin{array}{ccc|ccc} 1 & 3 & 0 & -2 & 2 & 1 \\ 0 & -1 & 0 & 13 & -9 & -5 \\ 0 & 0 & -1 & -3 & 2 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1 + 3R_2 \\ R_2 \times (-1) \\ R_3 \times (-1) \end{array} \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 37 & -25 & -14 \\ 0 & 1 & 0 & -13 & 9 & 5 \\ 0 & 0 & 1 & 3 & -2 & -1 \end{array}\right).$$

Now

$$\|A\|_1 = |\ 1\ | + |\ -3\ | + |\ 8\ | = 12$$

and

$$\|A\|_\infty = |\ -1\ | + |\ -1\ | + 8(= |\ 2\ | + |\ 5\ | + |\ -3\ |) = 10.$$

Similarly, $\|A^{-1}\|_1 = 53$ and $\|A^{-1}\|_\infty = 76$. Hence, for the ℓ_1 norm $\text{cond}(A) = 12 \times 53 = 636$ and for the ℓ_∞ norm $\text{cond}(A) = 10 \times 76 = 760$. Solving $A\mathbf{x} = (1, -3, 8)^T$ by Gaussian elimination, we have

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 1 \\ 2 & 5 & -3 & -3 \\ -1 & -1 & 8 & 8 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_2 - 2R_1 \\ R_3 - R_1 \end{array} \left(\begin{array}{ccc|c} 1 & 3 & 1 & 1 \\ 0 & -1 & -5 & -5 \\ 0 & 2 & 9 & 9 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_3 + 2R_2 \end{array} \left(\begin{array}{ccc|c} 1 & 3 & 1 & 1 \\ 0 & -1 & -5 & -5 \\ 0 & 0 & -1 & -1 \end{array}\right).$$

Back-substitution gives $\mathbf{x} = (0, 0, 1)^T$. For $\mathbf{b} \to \mathbf{b} + \delta\mathbf{b}$ we have

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 1.01 \\ 2 & 5 & -3 & -3 \\ -1 & -1 & 8 & 8 \end{array}\right) \begin{array}{c} \text{GE} \\ \to \end{array} \left(\begin{array}{ccc|c} 1 & 3 & 1 & 1.01 \\ 0 & 1 & 5 & 5.02 \\ 0 & 0 & 1 & 1.03 \end{array}\right)$$

$$\begin{array}{c} \text{BS} \\ \to \end{array} \mathbf{x} = \begin{pmatrix} 0.37 \\ -0.13 \\ 1.03 \end{pmatrix}.$$

From the previous results: $\|\delta\mathbf{x}\| = 0.53, \|\delta\mathbf{b}\| = 0.01, \|\mathbf{x}\| = 1$ and $\|\mathbf{b}\| = 12$ for the ℓ_1 norm. Hence,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} = 0.53, \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = \frac{0.01}{12}$$

and

$$\text{cond}(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = 636 \times \frac{0.01}{12} = 0.53.$$

$$\therefore \quad \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} = \text{cond}(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

for the ℓ_1 norm.

8.2 Working to 2 decimal places only, L_1U factorisation gives

$$\begin{pmatrix} 3.1 & 1.3 & 2.4 \\ 2.0 & 4.1 & 1.2 \\ 1.1 & -2.8 & 1.1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0.65 & 1 & 0 \\ 0.35 & -0.99 & 1 \end{pmatrix} \begin{pmatrix} 3.10 & 1.30 & 2.40 \\ 0 & 3.2 & -0.36 \\ 0 & 0 & -0.14 \end{pmatrix}.$$

Solving $L_1\mathbf{y} = (4.4, 6.1, -1.7)^T$ gives $\mathbf{y} = (4.40, 3.24, -0.03)^T$. Solving $U\mathbf{x}_0 = \mathbf{y}$ gives $\mathbf{x}_0 = (0.84, 1.01, 0.21)^T$. N.B. Compared to the exact solution $(1, 1, 0)^T$ sizable errors are present. The residuals are given by $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$,

$$\therefore \quad \mathbf{r}_0 = \begin{pmatrix} 4.4 \\ 6.1 \\ -1.7 \end{pmatrix} - \begin{pmatrix} 3.1 & 1.3 & 2.4 \\ 2.0 & 4.1 & 1.2 \\ 1.1 & -2.8 & 1.1 \end{pmatrix} \begin{pmatrix} 0.84 \\ 1.01 \\ 0.21 \end{pmatrix} = \begin{pmatrix} -0.0210 \\ 0.0270 \\ -0.0270 \end{pmatrix}$$

working to 4 decimal places. We then solve $L_1U\mathbf{z}_0 = \mathbf{r}_0$. Solving $L_1\mathbf{y} = \mathbf{r}_0$ gives $\mathbf{y} = (-0.0210, 0.0407, 0.0207)^T$. Solving $U\mathbf{z}_0 = \mathbf{y}$ gives $\mathbf{z}_0 = (0.1093, -0.0038, -0.1479)^T$. The improved solution is then given by

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0 = \begin{pmatrix} 0.9493 \\ 1.0062 \\ 0.0621 \end{pmatrix}.$$

8.3 The residual vector $\mathbf{r}$ associated with the approximate solution $\hat{\mathbf{x}}$ to $A\mathbf{x} = \mathbf{b}$ is given by $\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}$. Also $0 = \mathbf{b} - A\mathbf{x}$, hence $\mathbf{r} = A(\mathbf{x} - \hat{\mathbf{x}})$ or $(\mathbf{x} - \hat{x}) = A^{-1}\mathbf{r}$. Taking norms,

$$\|\mathbf{x} - \hat{\mathbf{x}}\| = \|A^{-1}\mathbf{r}\| \leq \|A^{-1}\|\|\mathbf{r}\|.$$

Dividing through by $\|\mathbf{x}\|$ gives

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\|A^{-1}\|}{\|\mathbf{x}\|}\|\mathbf{r}\| \leq \|A^{-1}\|\|A\|\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

since $\|\mathbf{x}\| \geq \|\mathbf{b}\|/\|A\|$. Hence,

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \text{cond}(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

where $\text{cond}(A) = \|A^{-1}\|\|A\|$.

8.4 Using the identity $B^{-1} - A^{-1} = A^{-1}(A - B)B^{-1}$ and taking norms, we have

$$\|B^{-1} - A^{-1}\| = \|A^{-1}(A - B)B^{-1}\| \leq \|A^{-1}\|\|A - B\|\|B^{-1}\|.$$

$$\therefore \quad \|A\|\|B^{-1} - A^{-1}\| \leq \|A\|\|A^{-1}\|\|A - B\|\|B^{-1}\|$$

or

$$\text{cond}(A) \geq \frac{\|B^{-1} - A^{-1}\|\|A\|}{\|A - B\|\|B^{-1}\|}.$$

8.5 Using the identity $(I + A)^{-1}(I + A) = I$, we have

$$(I + A)^{-1}I + (I + A)^{-1}A = I$$

or

$$(I + A)^{-1}I = I - (I + A)^{-1}A.$$

Taking norms,

$$\|(I + A)^{-1}\| = \|I - (I + A)^{-1}A\| \leq \|I\| + \|(I + A)^{-1}\|\|A\|.$$

Thus, provided $\|A\| < 1$,

$$\|(I + A)^{-1}\| \leq \frac{1}{1 - \|A\|}.$$

8.6 Let

$$e(a_j) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \sum_{i=1}^{n} \left(xi - \sum_{j=1}^{n} y_{i-j} a_j \right)^2 .$$

Now, $e(a_j)$ is a minimum when

$$\frac{\partial e}{\partial a_k} = 0 \quad \forall \quad 1 \leq k \leq n.$$

Differentiating, we have

$$\frac{\partial e}{\partial a_k} = \frac{\partial}{\partial a_k} \sum_{i=1}^{n} \left(x_i - \sum_{j=1}^{n} y_{i-j} a_j \right)^2 = 2 \sum_{i=1}^{n} \left(x_i - \sum_{j=1}^{n} y_{i-j} a_j \right) y_{i-k} = 0.$$

Hence

$$\sum_{i=1}^{n} (x_i - \hat{x}_i)\, y_{i-k} = 0.$$

8.7 Find an initial solution to $L_1 U \mathbf{x} = \mathbf{b}$ working to 6 digit accuracy:

$$L\mathbf{y} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.25 & 1 & 7 & \\ 0.50 & 0.46 & 1 & 0 \\ -0.50 & 0.77 & 0.33 & 1 \end{pmatrix} \mathbf{y} = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} \longrightarrow \mathbf{y} = \begin{pmatrix} 5.0 \\ 4.75 \\ 2.315 \\ 6.07855 \end{pmatrix}.$$

$$U\mathbf{x}_0 = \begin{pmatrix} 4 & 3 & 1 & 2 \\ 0 & 3.3 & 2.7 & 0.50 \\ 0 & 0 & 4.2 & -0.25 \\ 0 & 0 & 0 & 8.7 \end{pmatrix} \mathbf{x}_0 = \begin{pmatrix} 5.0 \\ 4.75 \\ 2.315 \\ 6.07855 \end{pmatrix} \longrightarrow \mathbf{x}_0 = \begin{pmatrix} 0.11607 \\ 0.84853 \\ 0.59278 \\ 0.69868 \end{pmatrix}.$$

Compute the residual vector $\mathbf{r}_0$:

$$\mathbf{r}_0 = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} - \begin{pmatrix} 4 & 3 & 1 & 2 \\ 1 & 4 & 3 & 1 \\ 2 & 3 & 6 & 1 \\ -2 & 1 & 3 & 8 \end{pmatrix} \begin{pmatrix} 0.11607 \\ 0.84853 \\ 0.59278 \\ 0.69868 \end{pmatrix} = \begin{pmatrix} -0.00001 \\ 0.01279 \\ -0.03309 \\ 0.01583 \end{pmatrix}.$$

Solving $L_1 U \mathbf{z}_0 = \mathbf{r}_0$ gives

$$\mathbf{z}_0 = \begin{pmatrix} -0.00707 \\ 0.01103 \\ -0.000915 \\ 0.00217 \end{pmatrix}.$$

$$\therefore \quad \mathbf{x}_1 = \mathbf{z}_0 + \mathbf{x}_0 = \begin{pmatrix} 0.10900 \\ 0.85956 \\ 0.58363 \\ 0.70085 \end{pmatrix}.$$

Compute the residual vector $\mathbf{r}_1 = \mathbf{b} - A\mathbf{x}_1$, which gives

$$\mathbf{r}_1 = \begin{pmatrix} -0.00001 \\ 0.00102 \\ 0.00069 \\ 0.00075 \end{pmatrix}.$$

Solving $L_1 U z_1$, we have

$$\mathbf{z}_1 = \begin{pmatrix} 0.00026 \\ 0.00024 \\ 0.00006 \\ 0.00013 \end{pmatrix}.$$

$$\therefore \quad \mathbf{x}_2 = \mathbf{z}_1 + \mathbf{x}_1 = \begin{pmatrix} 0.10926 \\ 0.85980 \\ 0.58369 \\ 0.70098 \end{pmatrix}.$$

Comparing $\mathbf{x}_2$ with $\mathbf{x}_1$ we see that $\mathbf{x}_1$ has 4 digit accuracy (i.e. the first 4 digits of $\mathbf{x}_1$ are the same as $\mathbf{x}_2$). Hence, $\mathbf{x}_2$ should have at least 5 digit ac Rounding off, we obtain the solution

$$\mathbf{x} = \begin{pmatrix} 0.109 \\ 0.860 \\ 0.584 \\ 0.701 \end{pmatrix}.$$

8.8 For Cholesky factorization $A = CC^T$,

$$\begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ \frac{1}{2} & \sqrt{\frac{15}{4}} & 0 & 0 \\ 0 & \sqrt{\frac{4}{15}} & \sqrt{\frac{56}{15}} & 0 \\ 0 & 0 & \sqrt{\frac{15}{56}} & \sqrt{\frac{97}{56}} \end{pmatrix} \begin{pmatrix} 2 & \frac{1}{2} & 0 & 0 \\ 0 & \sqrt{\frac{15}{4}} & \sqrt{\frac{4}{15}} & 0 \\ 0 & 0 & \sqrt{\frac{56}{15}} & \sqrt{\frac{15}{56}} \\ 0 & 0 & 0 & \sqrt{\frac{97}{56}} \end{pmatrix}$$

$$= \begin{pmatrix} 2.00 & 0.00 & 0.00 & 0.00 \\ 0.50 & 1.94 & 0.00 & 0.00 \\ 0.00 & 0.52 & 1.93 & 0.00 \\ 0.00 & 0.00 & 0.52 & 1.32 \end{pmatrix} \begin{pmatrix} 2.00 & 0.50 & 0.00 & 0.00 \\ 0.00 & 1.94 & 0.52 & 0.00 \\ 0.00 & 0.00 & 1.93 & 0.52 \\ 0.00 & 0.00 & 0.00 & 1.32 \end{pmatrix}$$

working to 2 decimal places only. Solving (working with six digits)

$$CC^T\mathbf{x}_0 = (5, 6, 7, 8)^T$$

gives

$$\mathbf{x}_0 = (0.97060, 1.11762, 0.53898, 3.70583)^T.$$

The residual is

$$\mathbf{r}_0 = (5, 6, 7, 8)^T - A\mathbf{x}_0 = (-0.00002, 0.01994, 0.02063, 0.04936)^T, \quad \|\mathbf{r}_0\|_\infty = 0.04936.$$

Solving

$$CC^T\mathbf{z}_0 = \mathbf{r}_0$$

gives

$$\mathbf{z}_0 = (-0.00152, 0.00607, -0.00289, 0.02596)^T,$$
$$\mathbf{x}_1 = (0.96908, 1.12369, 0.53609, 3.73179)^T$$

and

$$\mathbf{r}_1 = (-0.00001, 0.00007, 0.00016, 0.00033)^T, \quad \|\mathbf{r}_1\|_\infty = 0.00033.$$

Solving

$$CC^T\mathbf{z}_1 = \mathbf{r}_1$$

gives

$$\mathbf{z}_1 = (-8.0 \times 10^{-6}, 0.00002, -7.0 \times 10^{-6}, 0.00016)^T,$$
$$\mathbf{x}_2 = (0.96907, 1.12371, 0.53608, 3.73195)^T.$$

Now

$$\|\mathbf{r}_2\|_\infty < 10^{-4}$$

and hence, the solution, correct to four decimal places is

$$\mathbf{x} = (0.691, 1.1237, 0.5361, 3.7320)^T.$$

8.9 Doolittle factorization gives

$$A = \begin{pmatrix} 10 & 5 & 3.3 & 0 \\ 5 & 3.3 & 2.5 & 2 \\ 3.3 & 2.5 & 4 & 1.7 \\ 0 & 2 & 1.7 & 6.5 \end{pmatrix} \simeq CC^T$$

where

$$C = \begin{pmatrix} 3.16 & 1.58 & 1.04 & 0 \\ 0 & 0.9 & 0.95 & 2.22 \\ 0 & 0 & 1.42 & -0.29 \\ 0 & 0 & 0 & 1.22 \end{pmatrix}.$$

The initial solution to $A\mathbf{x} = (10, 8, 6, 4)^T$ is $\mathbf{x}_0 = (-3.8910.24 - 0.71 - 2.35)^T$ with $\mathbf{r}_0 = (0.043, 0.122, 0.072, 0.002)^T$ and $\|\mathbf{r}_0\|_\infty = 0.133$. Subsequent iterations give:

$$\begin{array}{rclrcl} \mathbf{x}_1 &=& (-4.2050, 10.9230, -0.7788, -2.5410)^T, & \|\mathbf{r}_1\|_\infty &=& 0.0081; \\ \mathbf{x}_2 &=& (-4.2255, 10.9679, -0.7834, -2.5545)^T, & \|\mathbf{r}_2\|_\infty &=& 0.00083; \\ \mathbf{x}_3 &=& (-4.2268, 10.9709, -0.7837, -2.5553)^T, & \|\mathbf{r}_3\|_\infty &=& 0.00013; \\ \mathbf{x}_4 &=& (-4.2269, 10.9711, -0.7837, -2.5554)^T, & \|\mathbf{r}_4\|_\infty &<& 10^{-4}. \end{array}$$

A.2.4 Solutions to Problems Given in Chapter 9

9.1 (i) The iteration equations are:

$$\begin{array}{rcl} x_1^{(k+1)} &=& \frac{1}{2}(1 + x_2^{(k)}), \\ x_2^{(k+1)} &=& \frac{1}{2}(x_1^{(k+1)} + x_3^{(k)}), \\ x_3^{(k+1)} &=& \frac{1}{2}(x_2^{(k+1)} + x_4^{(k)}, \\ x_4^{(k+1)} &=& \frac{1}{2}x_3^{(k)}. \end{array}$$

With $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = x_4^{(0)} = 0$, after 11 iterations $x_1^{(11)} = 0.79$, $x_2^{(11)} = 0.59$, $x_3^{(11)} = 0.39$, $x_4^{(11)} = 0.20$. The exact solution to this system is $\mathbf{x} = (0.8, 0.6, 0.4, 0.2)^T$.

(ii) $\mathbf{x} = (0.29, 0.85, 0.33, 3.84)^T$ correct to two decimal places. Jacobi iteration takes 13 iterations and Gauss-Seidel iteration takes 8 iterations to produce this result.

9.2 The characteristic matrix

$$A = \begin{pmatrix} 5 & 2 & -1 \\ 1 & 6 & -3 \\ 2 & 1 & 4 \end{pmatrix}$$

is diagonally dominant since

$$\begin{array}{ll} R_1: & |\,5\,| > |\,2\,| + |\,-1\,|, \\ R_2: & |\,6\,| > |\,1\,| + |\,-3\,|, \\ R_3: & |\,4\,| > |\,2\,| + |\,1\,|\,. \end{array}$$

The Jacobi iteration matrix is given by $M_J = -D^{-1}(L + U)$. Now,

$$A = L + D + U = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 4 \end{pmatrix} + \begin{pmatrix} 0 & 2 & -1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix}.$$

$$\therefore\ M_J = -\begin{pmatrix} 1/5 & 0 & 0 \\ 0 & 1/6 & 0 \\ 0 & 0 & 1/4 \end{pmatrix}\begin{pmatrix} 0 & 2 & -1 \\ 1 & 0 & -3 \\ 2 & 1 & 0 \end{pmatrix} = -\begin{pmatrix} 0 & 2/5 & -1/5 \\ 1/6 & 0 & -1/2 \\ 1/2 & 1/4 & 0 \end{pmatrix}.$$

The Gauss-Seidel iteration matrix is given by $G_G = -(D + L)^{-1}U$. Now

$$(D + L) = \begin{pmatrix} 5 & 0 & 0 \\ 1 & 6 & 0 \\ 2 & 1 & 4 \end{pmatrix}$$

and using Jordan's method,

$$\left(\begin{array}{ccc|ccc} 5 & 0 & 0 & 1 & 0 & 0 \\ 1 & 6 & 0 & 0 & 1 & 0 \\ 2 & 1 & 4 & 0 & 0 & 1 \end{array}\right) \longrightarrow \begin{array}{c} \\ R_2 - \frac{1}{5}R_1 \\ R_3 - \frac{2}{5}R_1 \end{array} \left(\begin{array}{ccc|ccc} 5 & 0 & 0 & 1 & 0 & 0 \\ 1 & 6 & 0 & -1/5 & 1 & 0 \\ 0 & 1 & 4 & -2/5 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} \\ R_3 - \frac{1}{6}R_2 \end{array} \left(\begin{array}{ccc|ccc} 5 & 0 & 0 & 1 & 0 & 0 \\ 0 & 6 & 0 & -1/5 & 1 & 0 \\ 0 & 0 & 4 & -11/30 & -1/6 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{c} R_1/5 \\ R_2/6 \\ R_3/4 \end{array} \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/5 & 0 & 0 \\ 0 & 1 & 0 & -130 & 1/6 & 0 \\ 0 & 0 & 1 & -11/120 & -1/24 & 1/4 \end{array}\right).$$

Checking the result to see if $AA^{-1} = I$,

$$\left(\begin{array}{ccc} 5 & 0 & 0 \\ 1 & 6 & 0 \\ 2 & 1 & 4 \end{array}\right)\left(\begin{array}{ccc} 1/5 & 0 & 0 \\ -130 & 1/6 & 0 \\ -11/120 & -1/24 & 1/4 \end{array}\right) = \left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right)$$

and

$$M_G = -\left(\begin{array}{ccc} 1/5 & 0 & 0 \\ -130 & 1/6 & 0 \\ -11/120 & -1/24 & 1/4 \end{array}\right)\left(\begin{array}{ccc} 0 & 2 & -1 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{array}\right) = -\left(\begin{array}{ccc} 0 & 2/5 & -/51 \\ 0 & -1/15 & -7/5 \\ 0 & -11/60 & 13/60 \end{array}\right).$$

Using Jacobi iteration:

$$\begin{array}{rcl} x_1^{(k+1)} & = & \frac{1}{5}(6 - 2x_2^{(k)} + x_3^{(k)}), \\ x_2^{(k+1)} & = & \frac{1}{6}(4 - x_1^{(k)} + 3x_3^{(k)}), \\ x_3^{(k+1)} & = & \frac{1}{4}(7 - 2x_1^{(k)} + x_2^{(k)}). \end{array}$$

For Gauss-Seidel iteration:

$$\begin{array}{rcl} x_1^{(k+1)} & = & \frac{1}{5}(6 - 2x_2^{(k)} + x_3^{(k)}), \\ x_2^{(k+1)} & = & \frac{1}{6}(4 - x_1^{(k+1)} + 3x_3^{(k)}), \\ x_3^{(k+1)} & = & \frac{1}{4}(7 - 2x_1^{(k+1)} + x_2^{(k+1)}). \end{array}$$

Jacobi iteration converges after 10 iterations giving

$$\mathbf{x} = (0.997626, 0.997577, 0.998863)^T.$$

Gauss-Seidel iteration converges after 6 iterations giving

$$\mathbf{x} = (1.003129, 1.001435, 0.998077)^T.$$

Observation of the iterative behaviour of the solution using either Jacobi or Gauss-Seidel shows that the solution oscillates and hence, under relaxation should be used. The set of iterative equations becomes

$$\begin{array}{rcl} x_1^{(k+1)} & = & x_1^{(k)} + \frac{\omega}{5}(6 - 2x_2^{(k)} + x_3^{(k)} - 5x_1^{(k)}), \\ x_2^{(k+1)} & = & x_2^{(k)} + \frac{\omega}{6}(4 - x_1^{(k+1)} + 3x_3^{(k)} - 6x_2^{(k)}), \\ x_3^{(k+1)} & = & x_3^{(k)} + \frac{\omega}{4}(7 - 2x_1^{(k+1)} + x_2^{(k+1)} - 4x_3^{(k)}). \end{array}$$

With a relaxation parameter set to 0.9, we obtain

$$\mathbf{x} = (1.005832, 0.994650, 0.997903)^T$$

after 5 iterations. The rate of convergence using SUR compared to the Gauss-Seidel method is not significantly improved.

9.3 Gauss-Seidel iteration converges in 48 iterations and is monotonic. SOR converges in 14 iterations. The characteristic matrix A of this system of equations is not diagonally dominant. Therefore, Gauss-Seidel iteration tends to converge slowly. In cases of this type, considerable improvement in the rate of convergence can be achieved using SOR. When A is highly diagonally dominant, the difference in the convergence rates between Gauss-Seidel and SOR is not significant as demonstrated in question 9.2.

9.4

System	Jacobi	Gauss-Seidel
(a)	Converges in 8 iterations	Converges in 6 iterations
(b)	Diverges	Converges in 9 iterations
(c)	Converges in 4 iterations	Diverges

9.5 (a) For Jacobi iteration,

$$D = \begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix}, \quad D^{-1} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 1/4 \end{pmatrix},$$

$$L+U = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \therefore \quad M_J = -D^{-1}(L+U) = \begin{pmatrix} 0 & 1/4 & 0 & 0 \\ 1/4 & 0 & 1/4 & 0 \\ 0 & 1/4 & 0 & 1/4 \\ 0 & 0 & 1/4 & 0 \end{pmatrix}.$$

The characteristic equation $|M_J - I\lambda| = 0$ is then

$$\begin{vmatrix} -\lambda & 1/4 & 0 & 0 \\ 1/4 & -\lambda & 1/4 & 0 \\ 0 & 1/4 & -\lambda & 1/4 \\ 0 & 0 & 1/4 & -\lambda \end{vmatrix} = -\lambda \begin{vmatrix} -\lambda & 1/4 & 0 \\ 1/4 & -\lambda & 1/4 \\ 0 & 1/4 & -\lambda \end{vmatrix} - \frac{1}{4} \begin{vmatrix} 1/4 & 1/4 & 0 \\ 0 & -\lambda & 1/4 \\ 0 & 1/4 & -\lambda \end{vmatrix} = 0$$

or

$$-\lambda \left[-\lambda \left(\lambda^2 - \frac{1}{16} \right) + \frac{\lambda}{16} \right] - \frac{1}{4} \left[\frac{1}{4} \left(\lambda^2 - \frac{1}{16} \right) \right] = 256\lambda^4 - 48\lambda^2 + 1 = 0$$

or with $x = \lambda^2$

$$256x^2 - 48x + 1 = 0$$

whose solution is

$$x = \frac{48 \pm \sqrt{48^2 - 4 \times 256}}{2 \times 256} = 0.1636, \quad 0.0239.$$

$$\therefore \quad \lambda = \pm 0.4045, \pm 0.1546$$

and

$$\rho(M_J) = \max \mid \lambda \mid = 0.4045.$$

For Gauss-Seidel iteration

$$L + D = \begin{pmatrix} -4 & 0 & 0 & 0 \\ 1 & -4 & 0 & 0 \\ 0 & 1 & -4 & 0 \\ 0 & 0 & 1 & -4 \end{pmatrix}$$

and

$$\left(\begin{array}{cccc|cccc} -4 & 0 & 0 & 0 & 71 & 0 & 0 & 0 \\ 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{l} R_2 + R_1/4 \\ R_3 + R_2/4 \\ R_4 + R_3/4 \end{array} \left(\begin{array}{cccc|cccc} -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 1/4 & 1 & 0 & 0 \\ 0 & 0 & -4 & 0 & 1/16 & 1/4 & 1 & 0 \\ 0 & 0 & 0 & -4 & 1/64 & 1/16 & 1/4 & 1 \end{array}\right) \longrightarrow$$

$$\begin{array}{l} R_i \times \left(-\frac{1}{4}\right) \\ i = 1, 2, 3, 4 \end{array} \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & -1/4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1/16 & -1/4 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1/64 & -1/16 & -1/4 & 0 \\ 0 & 0 & 0 & 1 & -1/256 & -1/64 & -1/16 & -1/4 \end{array}\right)$$

using Jordan's method and

$$-(D + L)^{-1}U = \begin{pmatrix} 0 & 1/4 & 0 & 0 \\ 0 & 1/16 & 1/4 & 0 \\ 0 & 1/64 & 1/16 & 1/4 \\ 0 & 1/256 & 1/64 & 1/16 \end{pmatrix}.$$

The characteristic equation $|M_G - I\lambda| = 0$ is

$$\begin{vmatrix} -\lambda & 1/4 & 0 & 0 \\ 0 & 1/16 - \lambda & 1/4 & 0 \\ 0 & 1/64 & 1/16 - \lambda & 1/4 \\ 0 & 1/256 & 1/64 & 1/16 - \lambda \end{vmatrix} = -\lambda \begin{vmatrix} 1/16 - \lambda & 1/4 & 0 \\ 1/64 & 1/16 - \lambda & 1/4 \\ 1/256 & 1/64 & 1/16 - \lambda \end{vmatrix} = 0$$

or

$$256\lambda^4 - 48^3 + \lambda^2 = 0.$$

Now, two eigenvalues are 0 and the other two eigenvalues are obtained by solving

$$256\lambda^2 - 48\lambda + 1 = 0$$

giving $\lambda = 0.1636, \quad 0.0239.$

$$\therefore \quad \rho(M_G) = \max \mid \lambda \mid = 0.1636.$$

$\rho(M_J)$ and $\rho(M_G)$ are both less than 1. Hence system (a) will converge for both Jacobi and Gauss-Seidel iteration. Also, note that $\rho(M_G) = [\rho(M_J)]^2$ which implies that Gauss-Seidel iteration will converge approximately twice as fast as Jacobi iteration.

(b) For the Jacobi method,

$$D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad \therefore \quad D^{-1} = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{pmatrix}.$$

$$L + U = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}. \quad \therefore \quad M_J = -D^{-1}(L+U) = \begin{pmatrix} 0 & -1/2 & -1/2 \\ -1/2 & 0 & -1/2 \\ -1/2 & -1/2 & 0 \end{pmatrix}.$$

Hence,

$$|M_J - I\lambda| = \begin{vmatrix} -\lambda & -1/2 & -1/2 \\ -1/2 & -\lambda & -1/2 \\ -1/2 & -1/2 & -\lambda \end{vmatrix} = 0$$

or

$$-\lambda\left(\lambda^2 - \frac{1}{4}\right) + \frac{1}{2}\left(\frac{\lambda}{2} - \frac{1}{4}\right) - \frac{1}{2}\left(\frac{1}{4} - \frac{\lambda}{2}\right) = -\lambda\left(\lambda^2 - \frac{1}{4}\right) + \left(\frac{\lambda}{2} - \frac{1}{4}\right)$$

$$= \lambda\left[\left(\lambda - \frac{1}{2}\right)\left(\lambda + \frac{1}{2}\right)\right] - \frac{1}{2}\left(\lambda - \frac{1}{2}\right) = \left(\lambda - \frac{1}{2}\right)\left[\lambda\left(\lambda + \frac{1}{2}\right) - \frac{1}{2}\right] = 0.$$

$$\therefore \quad \lambda = \frac{1}{2} \quad \text{or} \quad 2\lambda^2 + \lambda - 1 = 0,$$

giving

$$\lambda = \frac{-1 \pm \sqrt{1+8}}{4} = \frac{1}{2}, \quad -1.$$

Hence,

$$\rho(M_J) = 1.$$

For Gauss-Seidel iteration,

$$D + L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 2 \end{pmatrix}, \quad (D+L)^{-1} = \begin{pmatrix} 1/2 & 0 & 0 \\ -1/4 & 1/2 & 0 \\ -1/8 & -1/4 & 1/2 \end{pmatrix},$$

$$U = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad -(D+L)^{-1}U = \begin{pmatrix} 0 & -1/2 & -1/2 \\ 0 & 1/4 & -1/4 \\ 0 & 1/8 & -1/8 \end{pmatrix}.$$

$$| M_G - I\lambda | = -\lambda \begin{vmatrix} 1/4 - \lambda & -1/4 \\ 1/8 & -1/8 - \lambda \end{vmatrix} = -\lambda(\lambda^2 - \lambda/8) = 0.$$

$$\therefore \quad \lambda = 0 \quad \text{or} \quad \frac{1}{8} \simeq 0.125.$$

Hence,

$$\rho(M_G) = 0.125.$$

These results illustrate that the Jacobi method applied to system (b) will diverge but that the Gauss-Seidel method will converge.

(c)

$$D^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad U + L = \begin{pmatrix} 0 & 2 & -2 \\ 1 & 0 & 1 \\ 2 & 2 & 0 \end{pmatrix}.$$

$$\therefore \quad -D^{-1}(L+U) = -\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 2 & -2 \\ 1 & 0 & 1 \\ 2 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -2 & 2 \\ -1 & 0 & -1 \\ -2 & -2 & 0 \end{pmatrix}.$$

Now,

$$| M_J - I\lambda | = \begin{vmatrix} -\lambda & -2 & 2 \\ -1 & -\lambda & -1 \\ -2 & -2 & -\lambda \end{vmatrix} = -\lambda^3 + 4\lambda - 4\lambda = 0.$$

$$\therefore \quad \rho(M_J) = 0.$$

Further,

$$D + L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 2 & 1 \end{pmatrix}, \quad (D+L)^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}.$$

Hence,

$$M_G = -(D+L)^{-1}U = \begin{pmatrix} 0 & -2 & 2 \\ 0 & 2 & -3 \\ 0 & 0 & 2 \end{pmatrix}.$$

Now,

$$| M_G - I\lambda | = \begin{vmatrix} -\lambda & -2 & 2 \\ 0 & 2-\lambda & -3 \\ 0 & 0 & 2-\lambda \end{vmatrix} = 0.$$

$$\Longrightarrow \lambda(2-\lambda)^2 = 0.$$

Thus, $\lambda = 0, 2$ and

$$\rho(M_G) = 2.$$

System (c) will therefore converge using Jacobi's method but diverges with Gauss-Seidel iteration.

9.6 Only system (a) is diagonally dominant because

$$| a_{ii} | > \sum_{\substack{j=1 \\ j \neq i}}^{4} | a_{ij} | .$$

In system (b),

$$\mid a_{ii} \mid = \sum_{\substack{j=1 \\ j \neq i}}^{3} \mid a_{ij} \mid$$

and in system (c),

$$\mid a_{ii} \mid < \sum_{\substack{j=1 \\ j \neq i}}^{3} \mid a_{ij} \mid .$$

Diagonal dominance is not a necessary condition for convergence because from the results of question 9.4, we observe that convergence occurs when Jacobi iteration is applied to system (c) and when Gauss-Seidel iteration is applied to system (b). Diagonal dominance is a sufficient not a necessary condition. The necessary condition for convergence is that the spectral radius of the iteration matrix is less than 1.

9.7 Let λ_i be the eigenvalues obtained by solving $\mid A - I\lambda_i \mid = 0$ and $\mathbf{v}_i$ be the associated eigenvectors obtained by solving $(A - I\lambda_i)\mathbf{v}_i = 0$. Taking norms, we have

$$\|A\mathbf{v}_i\| = \|\lambda_i \mathbf{v}_i\| = \mid \lambda_i \mid \|\mathbf{v}_i\|.$$

$$\therefore \quad \mid \lambda_i \mid = \frac{\|A\mathbf{v}_i\|}{\|\mathbf{v}_i\|} \leq \|A\| \quad \forall i.$$

Now, the spectral radius is given by

$$\rho(A) = \max_i \mid \lambda_i \mid$$

and hence

$$\rho \leq \|A\|.$$

9.8 The conjugate directions are $\mathbf{v}_1 = (1,0,0)^T$, $\mathbf{v}_2 = (0,1,1)^T$, $\mathbf{v}_3 = (7,-12,9)^T$ obtained by taking an arbitrary form for $\mathbf{v}_1$ and then obtaining $\mathbf{v}_2$ and $\mathbf{v}_3$ from use of the condition

$$\mathbf{v}_i^T A \mathbf{v}_j = 0, \quad i \neq j.$$

To solve $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b} = (4,6,4)^T$, we use

$$\mathbf{x} = \sum_{i=1}^{3} \left(\frac{\mathbf{v}_i^T \mathbf{b}}{\mathbf{v}_i^T A \mathbf{v}_i} \right) \mathbf{v}_i.$$

Now,

$$A\mathbf{v}_1 = (3,1,-1)^T, \quad A\mathbf{v}_2 = (0,3,4)^T, \quad A\mathbf{v}_3 = (0,-8,8)^T.$$

$$\therefore \quad \mathbf{x} = \frac{4}{3} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{10}{7} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} - \frac{8}{168} \begin{pmatrix} 7 \\ -12 \\ 9 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}.$$

9.9

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix}, \quad \mathbf{x}_0 = \mathbf{0}.$$

$$\begin{aligned}
\mathbf{r}_0 &= \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} = \mathbf{v}_0, \\
A\mathbf{v}_0 &= \begin{pmatrix} 12 \\ 12 \\ 4 \end{pmatrix}, \\
\alpha_0 &= \tfrac{\mathbf{v}_0^T \mathbf{r}_0}{\mathbf{v}_0^T A \mathbf{v}_0} = \tfrac{48}{112} = \tfrac{3}{7}, \\
\mathbf{x}_1 &= \mathbf{x}_0 + \alpha_0 \mathbf{v}_0 = \tfrac{1}{7} \begin{pmatrix} 12 \\ 12 \\ 12 \end{pmatrix}, \\
\mathbf{r}_1 &= \mathbf{r}_0 - \alpha_0 A \mathbf{v}_0 = \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} - \tfrac{1}{7} \begin{pmatrix} 36 \\ 36 \\ 12 \end{pmatrix} = \tfrac{8}{7} \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix}, \\
\beta_0 &= \tfrac{\mathbf{r}_1^T \mathbf{r}_1}{\mathbf{r}_0^T \mathbf{r}_0} = \tfrac{8}{7^2}, \\
\mathbf{v}_1 &= \mathbf{r}_1 + \beta_0 \mathbf{v}_0 = \tfrac{8}{7} \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix} + \tfrac{8}{7^2} \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} = \tfrac{8}{7^2} \begin{pmatrix} -3 \\ -3 \\ 18 \end{pmatrix}, \\
A\mathbf{v}_1 &= \tfrac{8}{7^2} \begin{pmatrix} -9 \\ -30 \\ 39 \end{pmatrix}, \\
\alpha_1 &= \tfrac{\mathbf{v}_1^T \mathbf{r}_1}{\mathbf{v}_1^T \mathbf{v}_1} = \tfrac{14}{39}, \\
\mathbf{x}_2 &= \mathbf{x}_1 + \alpha_1 \mathbf{v}_1 = \tfrac{1}{7} \begin{pmatrix} 12 \\ 12 \\ 12 \end{pmatrix} + \tfrac{14}{39} \times \tfrac{8}{7^2} \begin{pmatrix} -3 \\ -3 \\ 18 \end{pmatrix} = \tfrac{1}{13} \begin{pmatrix} 20 \\ 20 \\ 36 \end{pmatrix}, \\
\mathbf{r}_2 &= \mathbf{r}_1 - \alpha_1 A \mathbf{v}_1 = \tfrac{8}{7} \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix} - \tfrac{14}{39} \times \tfrac{8}{7^2} \begin{pmatrix} -9 \\ -30 \\ 39 \end{pmatrix} = \tfrac{8}{13} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \\
\beta_1 &= \tfrac{\mathbf{r}_2^T \mathbf{r}_2}{\mathbf{r}_1^T \mathbf{r}_1} = \tfrac{7}{3 \times 13^2}, \\
\mathbf{v}_2 &= \mathbf{r}_2 + \beta_1 \mathbf{v}_1 = \tfrac{8}{13} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} + \tfrac{7^2}{3 \times 13^2} \times \tfrac{8}{7^2} \begin{pmatrix} -3 \\ -3 \\ 18 \end{pmatrix} = \tfrac{8}{13^2} \begin{pmatrix} -14 \\ 12 \\ 6 \end{pmatrix}, \\
A\mathbf{v}_2 &= \tfrac{8}{13^2} \begin{pmatrix} -16 \\ 16 \\ 0 \end{pmatrix}, \\
\alpha_2 &= \tfrac{\mathbf{v}_2^T \mathbf{r}_2}{\mathbf{v}_2^T A \mathbf{v}_2} = \tfrac{13}{16}, \\
\mathbf{x}_3 &= \mathbf{x}_2 + \alpha_2 \mathbf{v}_2 = \tfrac{1}{13} \begin{pmatrix} 20 \\ 20 \\ 36 \end{pmatrix} + \tfrac{8}{13^2} \times \tfrac{13}{16} \begin{pmatrix} -14 \\ 12 \\ 6 \end{pmatrix} = \tfrac{1}{13} \begin{pmatrix} 13 \\ 26 \\ 39 \end{pmatrix}, \\
\mathbf{r}_3 &= \mathbf{r}_2 - \alpha_2 A \mathbf{v}_2 = \tfrac{8}{13} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} - \tfrac{13}{16} \times \tfrac{8}{13^2} \begin{pmatrix} -16 \\ 16 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.
\end{aligned}$$

A.2.5 Solutions to Problems Given in Chapter 10

10.1 With $\mathbf{x}_0 = (1,1)^T$ we have

$$\begin{array}{ll} & \text{Scaling} \\ A\mathbf{x}_0 = A(1,1)^T = (3,6)^T, & (0.5,1)^T; \\ A^2\mathbf{x}_0 = A(0.5,1)^T = (0.5,7)^T, & (0.0714,1)^T; \\ A^3\mathbf{x}_0 = A(0.0714,1)^T = (-1.6430,7.8572)^T, & (-0.2091,1)^T; \\ A^4\mathbf{x}_0 = A(-0.2091,1)^T = (-3.0455,8.4182)^T, & (-0.3618,1)^T; \\ A^5\mathbf{x}_0 = A(-0.3018,1)^T = (-3.8090,8.7236)^T, & (-0.4366,1)^T; \\ A^6\mathbf{x}_0 = A(-0.4366,1)^T = (-4.1830,8.8732)^T, & (-0.4714,1)^T; \\ A^7\mathbf{x}_0 = A(-0.4714,1)^T = (-4.3570,8.9428)^T, & (-0.4872,1)^T. \end{array}$$

Taking $\mathbf{x}_1 \simeq (-0.49,1)$ we get

$$\lambda_1 = \frac{(A\mathbf{x}_1)^T\mathbf{x}_1}{\mathbf{x}_1^T\mathbf{x}_1} = \frac{11.1605}{1.2401} = 8.99.$$

10.2

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad P^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}.$$

$$\therefore \quad B = P^{-1}AP = \begin{pmatrix} 4 & -2 & -3 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}.$$

The deflated matrix is

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

whose eigenvalues are trivial to find and given by 1, 2 and 3 with associated eigenvectors $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$ respectively. We now compute the first element β_1 of the eigenvectors of B using the formula

$$\beta_1 = \frac{\sum_{j=2}^{4} b_{ij}\beta_j}{\mu - 4}.$$

Thus,

$$\begin{array}{lll} \mu = 1: & \beta_1 = \frac{(-2)\times 1}{1-4} & = \frac{2}{3}; \\ \mu = 2: & \beta_1 = \frac{(-3)\times 1}{2-4} & = \frac{3}{2}; \\ \mu = 3: & \beta_1 = \frac{1\times 1}{3-4} & = -1. \end{array}$$

Hence, the eigenvectors of B are $(2/3,1,0,0)^T$ $,(3/2,0,1,0)^T$, $(-1,0,0,1)^T$. Now, $\beta_i = P^{-1}\mathbf{x}_i$ where β_i and $\mathbf{x}_i$ are eigenvectors of B and A respectively. Hence, if

$\mathbf{x}_1 = (1,1,1,1)^T$, then

$$\begin{aligned}\mathbf{x}_2 = P\beta_2 &= (2/3, 5/3, 2/3, 2/3)^T;\\ \mathbf{x}_3 = P\beta_3 &= (3/2, 3/2, 5/2, 3/2)^T;\\ \mathbf{x}_4 = P\beta_4 &= (-1,-1,-1,0)^T.\end{aligned}$$

Thus, the eigenvalues of A are 4, 1, 2 and 3 with associated eigenvectors $(1,1,1,1)^T$, $(2,5,2,2)^T$, $(3,3,5,2)^T$ and $(1,1,1,0)^T$ respectively.

10.3 With $s \equiv \sin\theta$ and $c \equiv \cos\theta$,

$$P = \begin{pmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{pmatrix}, \quad P^T = \begin{pmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{pmatrix},$$

$$AP = \begin{pmatrix} 5 & 0 & 1 \\ 0 & -3 & 0.1 \\ 1 & 0.1 & 2 \end{pmatrix}\begin{pmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{pmatrix} = \begin{pmatrix} 5c-s & 0 & 5s+c \\ -0.1s & -3 & 0.1c \\ c-2s & 0.1 & s+2c \end{pmatrix},$$

$$P^TAP = \begin{pmatrix} 5c^2+2s^2-2sc & -0.1s & c^2-s^2+3sc \\ -0.1s & -3 & 0.1c \\ c^2-s^2+3sc & 0.1c & 5s^2+2c^2+2sc \end{pmatrix}.$$

For $(1,3)$ and $(3,1)$ entries to be zero, we require that $c^2 - s^2 = -3sc$, i.e.

$$\cos^2\theta - \sin^2\theta = -3\sin\theta\cos\theta$$

or

$$\cos(2\theta) = -\frac{3}{2}\sin(2\theta)$$

giving

$$\theta = -\frac{1}{}\tan^{-1}\left(\frac{2}{3}\right).$$

The approximate eigenvalues are then 5.3028, 1.6972 and -3 with approximate eigenvectors $(0.9571, 0, 0.2898)^T$, $(-0.2898, 0, 0.9571)^T$ and $(0,1,0)^T$.

10.4

$$P^TAP = \begin{pmatrix} 2c^2-4sc+6s^2 & 0.1(c-s) & -4sc+2(c^2-s^2) & 0.1c-0.05s \\ 0.1(c-s) & 3 & 0.1(c+s) & 0.2 \\ -4sc+2(c^2-s^2) & 0.1(c+s) & 2s^2+4sc+6c^2 & 0.1s+0.05c \\ 0.1c-0.05s & 0.2 & 0.1s+0.05c & 1 \end{pmatrix}.$$

Using the formula

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{qq}-a_{pp}}\right),$$

which creates zeros in the pq and qp positions, we have

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\times 2}{6-2}\right) = \frac{1}{2}\tan^{-1}1 = \frac{\pi}{8}$$

and

$$-4\sin\theta\cos\theta + 2(\cos^2\theta - \sin^2\theta) = 0.$$

Working to two decimal places, this value of θ gives

$$P^TAP = \begin{pmatrix} 1.17 & 0.05 & 0 & 0.07 \\ 0.05 & 3 & 0.13 & 0.20 \\ 0 & 0.13 & 6.83 & 0.08 \\ 0.07 & 0.20 & 0.08 & 1.00 \end{pmatrix}.$$

Now, using Gerschgorin's first theorem, the intervals in which the four eigenvalues of P^TAP lie are $[1.05, 1.29]$, $[2.62, 3.38]$, $[6.62, 7.04]$ and $[0.65, 1.35]$. All eigenvalues are therefore positive. The determinant of a matrix is equal to the product of its eigenvalues. Thus, if the eigenvalues are all positive, then $\mid A \mid \neq 0$ and A is therefore nonsingular.

10.5 The eigenvalue equation is

$$A\mathbf{x} = \lambda\mathbf{x}$$

where

$$\mathbf{x} = (x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)^T.$$

Suppose that x_i is the largest element (in magnitude) of the eigenvector $\mathbf{x}$. We can the re-scale this eigenvector and write it as

$$\mathbf{x} = (x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)^T.$$

Equating the i^{th} component of $A\mathbf{x}$ and $\lambda\mathbf{x}$, we have

$$\sum_{j=1}^{n} a_{ij}x_j = \lambda x_i = \lambda$$

since $x_i = 1$. Hence,

$$\mid \lambda - a_{ii} \mid = \left| \sum_{j=1}^{n} a_{ij}x_j - a_{ii} \right| = \sum_{\substack{j=1 \\ j\neq i}}^{n} \mid a_{ij} \mid \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} \mid a_{ij} \mid\mid x_j \mid \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} \mid a_{ij} \mid$$

using the triangle inequality and the fact that $\mid x_j \mid \leq 1$. The graphic interpretation of this result is based on considering a_{ii} to be a value on the real axis of the complex plane which is the origin of a circle with radius

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} \mid a_{ii} \mid .$$

The eigenvalue then lies somewhere in the complex plane that lies inside or on the boundary of this circle.

10.6 From Gerschgorin's (first) theorem, the intervals in which the eigenvalues lie are $[-1, 3]$, $[-6, 4]$, $[-1, 13]$ and $[-7, 1]$. The Sturm sequence is

$$f_{n+1}(\lambda = (d_{n+1} - \lambda)f_n(\lambda - c_n^2 f_{n-1}(\lambda), \quad f_0(\lambda) = 1 \forall \lambda$$

where d_n are the diagonal elements and c_n are the of-diagonal elements. To find the number of eigenvalues which lie in the interval $[1, 2]$, we compute the Sturm sequence and find the number of agreements in sign for the case when $\lambda = 1$ and when $\lambda = 2$. Thus,

for $\lambda = 1$:

$$\begin{aligned} f_0(1) &= 1, \\ f_1(1) &= d_1 - 1 = 1 - 1 = 0, \\ f_2(1) &= (d_2 - 1)f_1(1) - c_1^2 f_0(1), \\ &= (-1 - 1) \times 0 - 2^2 \times 1 = -4, \\ f_3(1) &= (d_3 - 1)f_2(1) - c_2^2 f_1(1) \\ &= (6 - 1) \times (-4) - 3^2 \times 0 = -20, \\ f_4(1) &= (d_4 - 1)f_3(1) - c_3^2 f_2(1) \\ &= (-3 - 1) \times (-20) - 4^2 \times (-4) \\ &= 80 + 64 = 144. \end{aligned}$$

The signs of the terms are $+ \quad - \quad - \quad - \quad +$. There are two agreements in sign and therefore two eigenvalues > 1.

For $\lambda = 2$:

$$\begin{aligned} f_0(2) &= 1, \\ f_1(2) &= d_1 - 2 = -1, \\ f_2(2) &= (d_2 - 1)f_1(2) - c_1^2 f_0(2) \\ &= (-1 - 2) \times (-1) - 2^2 \times 1 = -1, \\ f_3(2) &= (d_3 - 2)f_2(2) - c_2^2 f_1(2) \\ &= (6 - 2) \times (-1) - 3^2 \times (-1) = 5, \\ f_4(2) &= (d_4 - 2)f_3(2) - c_3^2 f_2(2) \\ &= (-3 - 2) \times 5 - 4^2 \times (-1) \\ &= -25 + 16 = -9. \end{aligned}$$

The signs of the terms are $+ \quad - \quad - \quad + \quad -$. There is 1 agreement in sign and therefore one eigenvalue > 1. Now, since there are four eigenvalues all together, 2 eigenvalues > 1 and 1 eigenvalue > 2, then only 1 eigenvalue lies in the interval $[1, 2]$.

10.7 From Gerschgorin's theorem, the eigenvalues of

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & -1 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

lie in the intervals $[1, 3]$, $[-3, 1]$ and $[1, 3]$. Hence, the range in which all the eigenvalues lie is $[-3, 3]$, i.e. $[-\|A\|_\infty, \|A\|_\infty]$. The Sturm sequence for A is

$$f_0(\lambda) = 1, \quad f_1(\lambda) = 2-\lambda, \quad f_2(\lambda) = (-1-\lambda)f_1(\lambda) - f_0(\lambda), \quad f_3(\lambda) = (2-\lambda)f_2(\lambda) - f_1(\lambda).$$

We now find the eigenvalues in the interval $[1, 3]$.

Bisecting, $\lambda = 2$ and

$$f_0(2) = 1, \quad f_1(2) = 0, \quad f_2(2) = -1, \quad f_3(2) = 0; s(2) = 1 \Longrightarrow \lambda_3 \in [2, 3].$$

Now, $f_3(2) = 0$; hence $\lambda_2 = 2$(exactly) where $s(\lambda)$ denotes the number of agreements in sign.

Bisecting, $\lambda = 2.5$ and

$f_0(2.5) = 1, \quad f_1(2.5) = -0.5, \quad f_2(2.5) = 0.75, \quad f_3(2.5) = 0.125,$

$s(2.5) = 1, \Longrightarrow \lambda_3 \in [2.5, 3] \Longrightarrow \lambda_3 = 2.75 \pm 0.25.$

Bisecting, $\lambda = 2.75$ and

$f_0(2.75) = 1, \quad f_1(2.75) = -0.75, \quad f_2(2.75) = 1.81, \quad f_3(2.75) = -0.61,$

$s(2.75) = 0 \Longrightarrow \lambda_3 \notin [2.75, 3] \Longrightarrow \lambda_3 \in [2.5, 2.75]$ or $\lambda_3 = 2.625 \pm 0.125.$

We now find the remaining eigenvalue in the interval $[-3, 1]$

Bisecting, $\lambda = -1$ and

$f_0(-1) = 1, \quad f_1(-1) = 3, \quad f_2(-1) = -1, \quad f_3(-1) = -6,$

$s(-1) = 2 \Longrightarrow \lambda_1 \in [-3, -1].$

Bisecting, $\lambda = -2$ and

$f_0(-2) = 1, \quad f_1(-2) = 4, \quad f_2(-2) = 3, \quad f_3(-2) = 8,$

$s(-2) = 3 \Longrightarrow \lambda_1 \in [-2, -1].$

Bisecting, $\lambda = -1.5$ and

$f_0(-1.5) = 1, \quad f_1(-1.5) = 3.5, \quad f_2(-1.5) = 0.75, \quad f_3(-1.5) = -0.88,$

$s(-1.5) = 2 \Longrightarrow \lambda_1 \in [-2, -1.5].$

Bisecting, $\lambda = -1.75$ and

$f_0(-1.75) = 1, \quad f_1(-1.75) = 3.75, \quad f_2(-1.75) = 1.81, \quad f_3(-1.75) = 3.04,$

$s(\lambda) = 3 \Longrightarrow \lambda_1 \in [-2, -1.75].$

Hence,

$$\lambda_1 = -1.875 \pm 0.125.$$

10.8 $Q = I - 2\mathbf{w}\mathbf{w}^T$. Taking the transpose of this equation, we have

$$\begin{aligned} Q^T &= [I - 2(\mathbf{w}\mathbf{w}^T)]^T \\ &= I^T - 2(\mathbf{w}\mathbf{w}^T)^T \\ &= I - 2(\mathbf{w}^T)^T\mathbf{w}^T \\ &= I - 2\mathbf{w}\mathbf{w}^T = Q \end{aligned}$$

where we have used the matrix properties,

$$(A + B)^T = A^T + B^T,$$

$$(AB)^T$$

and

$$(A^T)^T = A.$$

Hence, Q is symmetric. Also

$$\begin{aligned} Q^TQ = Q^2 &= (I - 2\mathbf{w}\mathbf{w}^T)(I - 2\mathbf{w}\mathbf{w}^T) \\ &= I - 4\mathbf{w}\mathbf{w}^T + 4\mathbf{w}(\mathbf{w}^T\mathbf{w})\mathbf{w}^T \\ &= I - 4\mathbf{w}\mathbf{w}^T + 4\mathbf{w}\mathbf{w}^T \quad (\text{since} \quad \mathbf{w}^T\mathbf{w} = 1 \\ &= I. \end{aligned}$$

Thus, Q is orthogonal. Since Householder's matrix is given by $Q = I - 2\mathbf{w}\mathbf{w}^T$ we can write $Q\mathbf{x} = k\mathbf{e}$ as

$$(I - 2\mathbf{w}\mathbf{w}^T)\mathbf{x} = k\mathbf{e}$$

or

$$\mathbf{x} - 2(\mathbf{w}\mathbf{w}^T)\mathbf{x} = k\mathbf{e}.$$

Rearranging,

$$2\mathbf{w}(\mathbf{w}^T\mathbf{x}) = \mathbf{x} - k\mathbf{e}.$$

Let $\mathbf{u} = \mathbf{x} - k\mathbf{e}$, then

$$2\mathbf{e}(\mathbf{w}^T\mathbf{x}) = \mathbf{u}$$

or $\mathbf{w} = \alpha\mathbf{u}$ where

$$\alpha = \frac{1}{2(\mathbf{w}^T\mathbf{x})}.$$

Now,

$$\mathbf{w}^T\mathbf{w} = \alpha^2\mathbf{u}^T\mathbf{u} = 1.$$

$$\therefore \quad \alpha^2 = \frac{1}{\mathbf{u}^T\mathbf{u}}.$$

Hence,

$$\mathbf{w} = \frac{1}{\mathbf{u}^T\mathbf{u}}\mathbf{u}$$

and

$$Q = I - 2\mathbf{w}\mathbf{w}^T = I - \frac{2\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}$$

where

$$\mathbf{u} = \mathbf{x} - k\mathbf{e}.$$

Note, to find k, we first take the transpose of $Q\mathbf{x} = k\mathbf{e}$ giving

$$\mathbf{x}^TQ^T = k\mathbf{e}^T.$$

Then

$$\mathbf{x}^TQ^TQ\mathbf{x} = k^2\mathbf{e}^T\mathbf{e}$$

but since $Q^TQ = I$ and $\mathbf{e}^T\mathbf{e} = 1$,

$$\mathbf{x}^T\mathbf{x} = k^2.$$

$$\therefore \quad k = \pm(\mathbf{x}^T\mathbf{x})^{\frac{1}{2}}$$

10.9 Householder's matrix is given by

$$Q = I - \frac{2\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}$$

where $\mathbf{u} = \mathbf{x} - k\mathbf{e}, k = \pm(\mathbf{x}^T\mathbf{x})^{\frac{1}{2}}$ and $\mathbf{e}$ is given by the first column of I. This matrix satisfies the equation (see Question 10.8)

$$Q\mathbf{x} = k\mathbf{e}.$$

The Householder matrix Q_1 which satisfies

$$Q_1\mathbf{x}_1 = k_1\mathbf{e}_1$$

where $\mathbf{x}_1 = (2,1,2)^T$ and $\mathbf{e}_1 = (1,0,0)^T$ can be computed thus:

(i) Compute k_1;

$$k_1 = \pm(2^2 + 1^2 + 2^2)^{\frac{1}{2}} = \pm 3 = -3,$$

since the first term of $\mathbf{x}_1$ is positive. (Note, the sign of k is chosen to be opposite to the sign of the first element of $\mathbf{x}$).

(ii) Compute $\mathbf{u}_1$;

$$u_1 = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix} - (-3)\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}.$$

(iii) Compute $\mathbf{u}_1^T\mathbf{u}$;

$$\mathbf{u}_1^T\mathbf{u} = (5,1,2)\begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix} = 30$$

Hence,

$$Q_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \frac{2}{30}\begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}(5,1,2) = \frac{1}{15}\begin{pmatrix} -10 & -5 & -10 \\ -5 & 14 & -2 \\ -10 & -2 & 11 \end{pmatrix}.$$

Similarly, the Householder matrix Q_2 which satisfies

$$Q_2\mathbf{x}_2 = k_2\mathbf{e}_2$$

where $\mathbf{x}_2 = (-3,-1)^T$ and $k_2 = (1,0)^T$ is computed thus:

(i)

$$k_2 = [(-3)^2 + (-1)^2]^{\frac{1}{2}} = \pm\sqrt{10} = \sqrt{10}$$

since the first element of $\mathbf{x}_2$ is negative.

(ii)

$$\mathbf{u}_2 = \begin{pmatrix} -3 \\ -1 \end{pmatrix} - \sqrt{10}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -3-\sqrt{10} \\ -1 \end{pmatrix}.$$

(iii)

$$\mathbf{u}_2^T\mathbf{u}_2 = (-3-\sqrt{10},-1)\begin{pmatrix} -3-\sqrt{10} \\ -1 \end{pmatrix} = 20+6\sqrt{10}.$$

Hence,

$$Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{10+3\sqrt{10}}\begin{pmatrix} -3-\sqrt{10} \\ -1 \end{pmatrix}(-3-\sqrt{10},-1)$$

$$= \frac{1}{10+2\sqrt{10}}\begin{pmatrix} -(9+3\sqrt{10}) & -(3+\sqrt{10}) \\ -(3+\sqrt{10}) & (9+3\sqrt{10}) \end{pmatrix}.$$

Working to 4 decimal places, reduction to tridiagonal form gives

$$\begin{pmatrix} 3.0000 & -3.0000 & 0.0000 & 0.0000 \\ -3.0000 & 4.0000 & 3.1623 & 0.0000 \\ 0.0000 & 3.1623 & 2.0000 & 1.0000 \\ 0.0000 & 0.0000 & 1.0000 & -2.0000 \end{pmatrix}.$$

10.10

$$A = \begin{pmatrix} 1 & 4 & 2 \\ -1 & 2 & 0 \\ 1 & 3 & -1 \end{pmatrix}$$

Step 1: Create a zero in the $(2,1)$ entry using

$$P_1 = \begin{pmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

with

$$\theta = -\tan^{-1}\left(\frac{a_{21}^{(1)}}{a_{11}^{(1)}}\right) = -\tan^{-1}\left(-\frac{1}{1}\right)$$

where $c \equiv \cos\theta$ and $s \equiv \sin\theta$ giving

$$P_1^T A = \begin{pmatrix} 1.4142 & 1.4142 & 1.4142 \\ 0 & 4.2436 & 1.4142 \\ 1.0000 & 3.0000 & -1.0000 \end{pmatrix}$$

working to four decimal place accuracy.

Step 2: Create a zero in the $(3,1)$ position using

$$P_2 = \begin{pmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{pmatrix}$$

with

$$\theta = -\tan^{-1}\left(\frac{1.0000}{1.4142}\right)$$

giving

$$P_2^T(P_1^T A) = \begin{pmatrix} 1.7321 & 2.8868 & 0.5774 \\ 0 & 4.2426 & 1.4142 \\ 0 & 1.6330 & -1.6330 \end{pmatrix}.$$

Step 3: Create a zero in the $(3,2)$ position using

$$P_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix}$$

with

$$\theta = -\tan^{-1}\left(\frac{1.6330}{4.2426}\right)$$

giving

$$R = P_3^T(P_2^T P_1^T A) = \begin{pmatrix} 1.7321 & 2.8868 & 0.5774 \\ 0 & 4.5461 & 0.7332 \\ 0 & 0 & -2.0320 \end{pmatrix}.$$

Then,

$$Q = P_1 P_2 P_3$$

$$= \begin{pmatrix} 0.7071 & 0.7071 & 0 \\ -0.7071 & 0.7071 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.8165 & 0 & -0.5774 \\ 0 & 1 & 0 \\ 0.5774 & 0 & 0.8165 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.9332 & -0.3592 \\ 0 & 0.3592 & 0.9332 \end{pmatrix}$$

$$= \begin{pmatrix} 0.5774 & 0.5133 & -0.6350 \\ -0.5774 & 0.8066 & 0.1270 \\ 0.5774 & 0.2933 & 0.7620 \end{pmatrix}$$

The orthogonal (QR) decomposition of the matrix is therefore given by

$$\begin{pmatrix} 0.5774 & 0.5133 & -0.6350 \\ -0.5774 & 0.8066 & 0.1270 \\ 0.5774 & 0.2933 & 0.7620 \end{pmatrix} \begin{pmatrix} 1.7321 & 2.8868 & 0.5774 \\ 0 & 4.5461 & 0.7332 \\ 0 & 0 & -2.0320 \end{pmatrix}.$$

A.2.6 Supplementary Problems to Part II

II.1 (i) Consider the following over-determined system

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \quad m > n.$$

Show that the solution to this system which minimizes $\|A\mathbf{x} - \mathbf{b}\|_{\mathbf{2}}^{\mathbf{2}}$ is obtained by solving the equations

$$\sum_{i=1}^{m} a_{ik}\left(\sum_{j=1}^{n} a_{ij}x_j - b_i\right) = 0, \quad k = 1, 2, ..., n$$

for x_j. Using this result, show that the optimum solution to the equations

$$x_1 + x_2 + x_3 = 1$$

$$x_1 + 2x_2 + 3x_3 = 2$$

$$3x_1 + 2x_2 - 2x_3 = 3$$

$$x_1 + x_2 + 2x_3 = 4$$

is obtained by solving the system

$$\begin{pmatrix} 6 & 5 & 0 \\ 2 & 2 & 1 \\ 0 & 5 & 18 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \\ 9 \end{pmatrix}.$$

(ii) Use Jordan's method with natural pivoting and exact arithmetic to obtain the inverse of the characteristic matrix of the system above. Hence, compute the condition numbers of this system for the ℓ_1 norm and the ℓ_∞ norm and find exact solutions for x_1, x_2 and x_3.

II.2 (i) Prove that:

(a) The eigenvalues of a matrix are preserved under a similarity transform $A \rightarrow P^{-1}AP$.

(b) If an $n \times n$ matrix A has n linearly independent eigenvectors $\mathbf{x}_i; i = 1, 2, ..., n$ and n distinct eigenvalues $\lambda_i; i = 1, 2, ..., n$, then

$$\text{diag}(\lambda_1 \lambda_2 ... \lambda_n) = X^{-1}AX$$

where

$$X = (\mathbf{x}_1 \mathbf{x}_2 ... \mathbf{x}_n).$$

(ii) (a) Derive the Sturm sequence for the matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 8 & 1 \\ 0 & 0 & 1 & 12 \end{pmatrix}.$$

(b) Use Gerschgorin's theorem to find the intervals in which the eigenvalues lie and hence compute the largest eigenvalue of this matrix with an error less than 0.25.

II.3. (a) Show that, for non-singular A and any vector/matrix norm (denoted by $\| \ \|$):

(i) if $Q = A^{-1}PA$, then

$$\frac{1}{k(A)} \|Q^i\| \leq \|P^i\| \leq k(A) \|Q^i\|$$

for any positive integer i;

(ii) if $A\mathbf{x} = \mathbf{b}$ and $A(\mathbf{x} + \mathbf{e}) = \mathbf{b} + \mathbf{r}$, then

$$\frac{1}{k(A)}\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \leq \frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq k(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

where $k(A)$ is the condition number.

(b) Explain Crout's method for solving a linear system of equations. Given that

$$A = \begin{pmatrix} 6 & 2 & 0 \\ 2 & 5 & 1 \\ 0 & 5 & 18 \end{pmatrix} \simeq \begin{pmatrix} 6.0 & 0.0 & 0.0 \\ 2.0 & 4.4 & 0.0 \\ 0.0 & 5.0 & 17.0 \end{pmatrix} \begin{pmatrix} 1.0 & 0.3 & 0.0 \\ 0.0 & 1.0 & 0.2 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

use Crout's method together with iterative improvement to find a solution to

$$A\mathbf{x} = \mathbf{b}$$

when $\mathbf{b} = (8.0, 9.0, 10.0)^T$ correct to three decimal places.

II.4 (a) If $A\mathbf{x} = \lambda\mathbf{x}$ where $\|\mathbf{x}\|_\infty$=1, show that

$$| \lambda - a_{ii} | \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} | a_{ij} | \qquad \text{(Gerschgorin's Theorem)}$$

Use this result to find the intervals containing the eigenvalues of the matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 5 & 1 & 0 \\ 0 & 1 & 10 & 1 \\ 0 & 0 & 1 & 15 \end{pmatrix}$$

(b) Explain Jacobi's method for computing the eigenvalues and eigenvectors of a symmetric matrix. Working to four decimal places only, use Jacobi's method to compute the eigenpairs of the matrix

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

II.5 (a) Suggest appropriate methods for solving the system of linear equations $A\mathbf{x} = \mathbf{b}$ in each of the following cases:

(i) A is a dense symmetric positive definite $n \times n$ real matrix.

(ii) A is a real $n \times n$ symmetric matrix which is known to be indefinite.

(iii) A is a dense real $n \times n$ matrix and the equations are possibly ill-conditioned.

(iv) A is a large sparse symmetric positive definite matrix.

Give reasons for your choice and explain any one method in detail.

(b) If $\|\mathbf{x}\|$ is a norm on R^n, define the induced norm $\|A\|$ and condition number $k(A)$ of a non-singular $n \times n$ matrix A.

Prove that

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} \mid a_{ij} \mid .$$

If $A\mathbf{x} = \mathbf{b}$ and $A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$ prove that

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq k(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

where $k(A)$ is the condition number. If

$$A = \begin{pmatrix} 1 & 3 & 1 \\ 2 & 7 & 1 \\ -1 & -2 & -1 \end{pmatrix}$$

find values of $\mathbf{b}$ and $\delta\mathbf{b}$ such that the above inequality attains its upper bound using $\|\mathbf{x}\|_\infty$.

II.6 (a) A is an $n \times n$ matrix and $\|A\|$ is any matrix norm. If A has real eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$, prove that

$$\frac{1}{\|A^{-1}\|} \leq \mid \lambda_i \mid \leq \|A\| \quad \text{for} \quad 1 \leq i \leq n.$$

(b) The equations

$$\begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

are to be solved by the Gauss-Seidel iteration method. Find the iteration matrix M which occurs in this process and show that M has a spectral radius of less than 0.33. Use this result to estimate the number of iterations required to find a solution to the set of equations above to an accuracy of 3 decimal places. Use the Gauss-Seidel method to solve these equations to 3 decimal place accuracy.

II.7 (a) Define the condition number of a (square) matrix A and explain how it can be used to assess the stability of the solution to a system of the type $A\mathbf{x} = \mathbf{b}$.

(b)

(i) Using the result

$$B^{-1} - A^{-1} = A^{-1}(A - B)B^{-1}$$

prove that

$$\chi(A) \geq \frac{\|B^{-1} - A^{-1}\| \ \ \|A\|}{\|A - B\| \ \ \|B^{-1}\|}$$

where A and B are matrices of the same order and $\chi(A)$ is the condition number of the matrix A.

(ii) Using the result

$$(I + A)^{-1}(I + A) = I$$

prove that

$$\|(I + A)^{-1}\| \leq \frac{1}{1 - \|A\|}, \quad \|A\| < 1.$$

(c) State and prove the Cayley-Hamilton theorem and then use it to compute the inverse of the matrix

$$\begin{pmatrix} 1 & 2 & -1 \\ 1 & 0 & 1 \\ 1 & 2 & 2 \end{pmatrix}$$

What are the computational problems associated with the applications of this method for computing the inverses of large matrices.

II.8 An iteration formula for solving the linear system

$$A\mathbf{x} = \mathbf{b}$$

has the form

$$\mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{c}$$

where A is a real square matrix, M is the associated iteration matrix and k it the iteration number.

(a) If $\mathbf{e}^{(k)}$ is the error vector associated with the k^{th} iteration, show that for global convergence, i.e.

$$\lim_{k \to \infty} \mathbf{e}^{(k)} = 0$$

then

$$\|M\| < 1.$$

(b) Consider the following linear system

$$\begin{pmatrix} 10 & 1 & 0 \\ 1 & 10 & 1 \\ 0 & 1 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 24 \\ 32 \end{pmatrix}.$$

(i) Find an expression for the Gauss-Seidel iteration matrix of this system.

(ii) Use the power method to compute the spectral radius of this iteration matrix.

(c) Write down the Gauss-Seidel iteration formulae for the system of equations given above and hence obtain solutions for x_1, x_2 and x_3 correct to 2 decimal places.

II.9 The modes of vibration setup on a string of length L fixed at both ends can be described by the homogeneous wave equation

$$\frac{d^2f}{dx^2} + k^2 f = 0$$

subject to the end conditions $f(0) = 0; f(L) = 0$. Here, f is the displacement (wave amplitude) and k defines the spatial frequency of the modes.

(a) Derive an analytical solution to this problem in terms of the eigenvalues and eigenfunctions and explain its physical significance.

(b) Using a centre differencing scheme, in which the second derivative is replaced with $[f(x+\Delta x) - 2f(x) + f(x-\Delta x)]/(\Delta x)^2$, show that if the problem is discretized, then the wave equation given above can be written in the form

$$A\mathbf{f} = \lambda \mathbf{f}.$$

(c)

(i) Write down the characteristic matrix A in the case when the string is divided up into 4 elements and then derive the Sturm sequence for this matrix.

(ii) State Gerschgorin's theorem and use it to find the intervals in which the eigenvalues of this matrix lie. Hence or otherwise, use the Sturm sequence derived above to compute the largest eigenvalue of the matrix with an error of less than 0.25.

II.10 (a) By eliminating the subdiagonal component using Gaussian elimination, solve the tridiagonal system of equations given by

$$A\mathbf{x} = (1 \quad 2 \quad 3 \quad 2 \quad 1)^T$$

where

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{pmatrix}.$$

(b) Consider the following tridiagonal system of equations

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1, \\ a_1x_1 + a_2x_2 + c_2x_3 &= b_2, \\ a_2x_2 + d_3x_3 + c_3x_4 &= b_3, \\ &\vdots \\ a_{n-2}x_2 + d_{n-1}x_{n-1} + c_{n-1}x_n &= b_{n-1}, \\ a_{n-1}x_{n-1} + d_nx_n &= b_n. \end{aligned}$$

By choosing the elements (d_i) as the pivots, derive an algorithm to compute the solution x_i to this system of equations.

II.11 A set of data points (x_i, y_i); $(i = 1, 2, ..., n)$ is given to which a polynomial fit of degree $m < n$ is required. Consider the weighted least square error

$$e(a_0, a_1, a_2, ..., a_m) = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the polynomial given by

$$\hat{y}_i = a_0 + a_1 x_i + a_2 x_i^2 + ... + a_m x_i^2$$

and $w_i, i = 1, 2, ..., n$ are given weights.

(a) By minimizing the error e given above, show that the coefficients a_i, $i = 1, 2, ..., m$ can be found by solving the system of equations given by

$$\begin{aligned}
a_0 \sum_{i=1}^{n} w_i + a_1 \sum_{i=1}^{n} w_i x_i + ... + a_m \sum_{i=1}^{n} w_i x_i^m &= \sum_{i=1}^{n} w_i y_i, \\
a_0 \sum_{i=1}^{n} w_i x_i + a_1 \sum_{i=1}^{n} w_i x_i^2 + ... + a_m \sum_{i=1}^{n} w_i x_i^{m+1} &= \sum_{i=1}^{n} w_i x_i y_i, \\
&\vdots \\
a_0 \sum_{i=1}^{n} w_i x_i^m + a_1 \sum_{i=1}^{n} w_i x_i^{m+1} + ... + a_m \sum_{i=1}^{n} w_i x_i^{2m} &= \sum_{i=1}^{n} w_i x_i^m y_i.
\end{aligned}$$

(b) Consider the following data,

x_i :	-5	-3	1	3	4	6	8
y_i :	18	7	0	7	16	50	67
w_i :	1	1	1	1	20	1	1

Write down the normal system of equations for a quadratic weighted least squares solution to this data. By evaluating the summations, show that the coefficients (a_i) associated with this method of approximation are given by the solution to a 3×3 system of linear equations which are positive definite symmetric.

(c) Use an appropriate decomposition method to compute the required solution and produce a sketch comparing the data with the polynomial fit.

II.12 (a) Prove that if the matrix $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ is constructed where $\mathbf{x}_i$ are linearly independent eigenvectors of the eigensystem $Ax_i = \lambda_i x_i$ then

$$X^{-1} A X = D \equiv \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n).$$

(b) Show that if the characteristic matrix A of the eigensystem is perturbed by δA, then

$$\|Q^{-1} X^{-1} \delta A X\| > 1$$

where $Q = D - \lambda' I$; λ' being an eigenvalue of $A + \delta A$. Use this result to show that

$$| \lambda' - \lambda_i | < K \|\delta A\|$$

where

$$K = \min(\|X^{-1}\| \quad \|X\|).$$

Comment on the significance of the number K.

(c) Use the power method to find one of the eigenvectors and associated eigenvalue of the matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix}.$$

II.13 (a) By writing the real square matrix A as

$$A = L + D + U$$

where L, D and U are strictly lower triangular, diagonal and upper triangular matrices respectively, show that the successive-over-relaxation iteration formula for solving the linear system $A\mathbf{x} = \mathbf{b}$ is given by

$$\mathbf{x}^{(n+1)} = M_\omega \mathbf{x}^{(n)} + \mathbf{c}$$

where

$$M_\omega = (D + \omega L)^{-1}[(1 - \omega)D - \omega U],$$
$$\mathbf{c} = (D + \omega L)^{-1}\omega \mathbf{b}$$

and ω is the relaxation parameter.

Using the power method, compute the spectral radius of M_1 when

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

and state whether or not convergence will occur in this case.

(b) By centre differencing Poisson's equation

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u(x, y) = f(x, y)$$

where

$$\frac{\partial^2}{\partial x^2} u(x, y) \to \frac{u(x + \Delta x, y) - 2u(x, y) + u(x - \Delta x, y)}{(\Delta x)^2}$$

derive a SOR iteration formula which could be used directly (with appropriate boundary conditions), to solve this equation for u in two dimensions on a Cartesian mesh $(x = i\Delta, y = j\Delta)$ where Δ is the step length.

II.14 (a) If λ_i and $\mathbf{v}_i$ are respectively an eigenvalue and the corresponding eigenvector of a $n \times n$ matrix A, write down the relationship between A, λ_i and $\mathbf{v}_i$.

Show that $\mathbf{v}_i$ is also an eigenvector of the matrix $A + kI$, where k is constant, and that the corresponding eigenvalue is $\lambda_i + k$.

(b) The eigenvectors of the matrix A, given as

$$A = \begin{pmatrix} 20 & -5 & -15 \\ -3 & -2 & 3 \\ 26 & -6 & -21 \end{pmatrix}$$

are known to be $\mathbf{v}_1 = (1, 0, 1)^T$, $\mathbf{v}_2 = (0, 3, -1)^T$, $\mathbf{v}_3 = (-1, 1, -2)^T$.

Use this information to find the eigenvalues of A and check your values against the trace of the matrix.

(c) A system of first order linear differential equations is given as

$$\frac{dx_1}{dt} = 20x_1 - 5x_2 - 15x_3$$

$$\frac{dx_2}{dt} = -3x_1 - 2x_2 + 3x_3$$

$$\frac{dx_3}{dt} = 26x_1 - 6x_2 - 21x_3$$

or

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x}$$

where $\mathbf{x} = (x_1, x_2, x_3)^T$.

If X is the matrix whose columns are the eigenvectors of A, show that the substitution $\mathbf{x} = X\mathbf{y}$ results in a simple system of differential equations and hence, find the general solution of the given system.

(d) (i) Explain briefly why the Power Method for the numerical computation of eigenvalues will not converge to a correct solution for the matrix A defined in part (b) above.

(ii) Perform three iterations of the Power Method with scaling on the matrix $(A + 4I)$, taking $(1, 1, 1)^T$ as the initial vector $\mathbf{x}_0$ to find an approximate eigenvalue and associated eigenvector of $(A + 4I)$.

(iii) Explain with reference to part (a) above how this approach can lead to finding an eigenpair of A. Comment of your result as compared with the exact values already known from part (b).

II.15 (i) Define the characteristic equation of a square matrix A and state the Caley-Hamilton Theorem for A.

If

$$A = \begin{pmatrix} 2 & 0 & 3 \\ 1 & 2 & -4 \\ 4 & 1 & 3 \end{pmatrix}$$

show that the characteristic equation for A is

$$-1 - 8\lambda + 7\lambda^2 - \lambda^3 = 0$$

Write down the equation given by the Caley-Hamilton Theorem for the matrix A and use this equation to find A^{-1}.

What are the practical disadvantages of using this method to find an inverse matrix?

(ii) Use Jordan's method to show that if M is the $n \times n$ matrix

$$M = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & m_3 & 1 & \dots & \dots & 0 \\ 0 & m_4 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \\ 0 & m_n & 0 & \dots & \dots & 1 \end{pmatrix}$$

then

$$M^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & -m_3 & 1 & \dots & \dots & 0 \\ 0 & -m_4 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \\ 0 & -m_n & 0 & \dots & \dots & 1 \end{pmatrix}$$

(iii) The product $P^{-1}BP$, where P is any non-singular matrix, is a similarity transform of the matrix B. If the eigenpairs of B are denoted by $(\mathbf{v}_i, \lambda_i)$, by considering the product $(P^{-1}BP)(P^{-1}\mathbf{v}_i)$, show that $P^{-1}BP$ has the same eigenvalues as B with corresponding eigenvectors $P^{-1}\mathbf{v}_i$.

Show that if

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}$$

is used as a transformation matrix P for a similarity transform of the matrix A given in part (i) above, the resulting transformed matrix $A_1 = M^{-1}AM$ has only one non-zero subdiagonal row, i.e. is upper Hessenberg form.

(iv) Outline briefly the way in which similarity transforms are often used in numerical methods to obtain the eigenvalues of a matrix. State why an orthogonal matrix is particularly useful to use as a transformation matrix giving an example of one such orthogonal matrix.

II.16 (i) If X is a non-singular $n \times n$ matrix whose columns are the eigenvectors of the $n \times n$ matrix A, prove that

$$X^{-1}AX = D$$

where D is a diagonal matrix. What are the diagonal elements of D and what condition must the eigenvectors satisfy for X to be non-singular?

(ii) The Power method is used to find the dominant eigenvalue and associated eigenvector of a matrix. Describe the method and discuss cases in which it may not be successful. On what does the rate of convergence chiefly depend?

The matrix A is given as

$$A = \begin{pmatrix} 49 & 42 & 6 \\ -24 & -17 & -6 \\ -24 & -21 & -2 \end{pmatrix}$$

Using $\mathbf{x}_0 = (1, 1, 1)^T$ as the starting vector, perform two iterations of the Power Method, with scaling, to obtain an approximation to the dominant eigenvalue of A and the associated eigenvector giving all values involved correct to one decimal place.

(iii) A system of second order linear differential equations is given as

$$\frac{d^2x_1}{dt^2} = -49x_1 - 42x_2 - 6x_3$$

$$\frac{d^2x_2}{dt^2} = 24x_1 + 17x_2 + 6x_3$$

$$\frac{d^2x_3}{dt^2} = 24x_1 + 21x_2 + 2x_3$$

or

$$\frac{d^2\mathbf{x}}{dt^2} = -A\mathbf{x}$$

where $\mathbf{x} = (x_1, x_2, x_3)^T$.

If the other two eigenvalues and eigenvectors of A are respectively 1, 4 and $(-1, 1, 1)^T$, $(2, -2, -1)^T$, show how the transformation $\mathbf{x} = X\mathbf{y}$, where X is the matrix composed of the eigenvectors of A, makes use of the result derived in (i) above to decouple the equations. Hence find the general solution of the differential equations.

A.3 Part III

A.3.1 Solution to Problems Given in Chapter 11

11.1 $8 = 2^3$ and $0.25 = 2^{-2}$. Hence 8.25 in binary for is 1000.01.

Decimal	**Binary**
8	1000
2	0010
5	0101

In BCD form 8.25 is 1000.00100101.

11.2 Fixed point storage of binary numbers involves defining the binary point of a word at a fixed location. For example, if binary numbers are stored using a 16 bit word where the binary point is taken to occur between the 8th and 9th entries, then only 8 bits can be allocated to the binary number before and after the binary point. Example:

Decimal	**Binary**	**Fixed point storage**
8.25	1000.01	0000100001000000
5.75	101.11	0000010111000000

Floating point representation uses a Mantissa/Exponent technique to normalize the binary point thus:

Decimal	**Binary**	**Normalised Floating Point**
8.25	1000.01	0.100001×10^{01}
5.75	101.11	0.10111×10^{0011}

In a 16 bit word in which the Mantissa and Exponent are distinguished between the 12th and 13th entries respectively, the Normalized Floating Point representations above are

1000010000000100

and

1011100000000011

respectively.

With fixed point storage, a problem occurs if the binary number becomes so large that the number of bits required to represent it exceeds the word length - giving an arithmetic 'overflow'. Floating point representation of the type described above allows much larger numbers to be stored using the same size word length. Also because the first column is always 1, this can be ignored in the representation providing further storage space - the principal of VAX/VMS systems.

11.3 The three basic constructs of a structured program are:

- Sequences - program statements executed in the order in which they appear.
- Selections - actions to be taken according to the conditions that exist at particular stages in execution (e.g. if-then-else statements).
- Repetitions - the repeated execution of the same basic statement (e.g. do-loops and for-loops).

Structured programs are programs designed using the three constructs listed above. Rigorous application of this principle leads to programs with a logical well structured

form in which the goto statement is obsolete. Goto statements allow the unconditional transfer of control from one point in a program to another. Unconstrained use of goto statements can therefore lead to a program having a complex network of data flow which is difficult for the reader to follow.

11.4

Arguments against the goto statement:

- They are unnecessary - any program written using a goto statement can be transformed into an equivalent program that uses only the structured constructs.
- Structured programs (without goto statements) can be understood by another programmer more rapidly than a non-structured program.
- Goto statements do not convey the meaning of a block of code as clearly as a structured construct - the goto statement has a lack of 'expressive power'.
- Goto statements may require that a program is read 'backwards'.
- The goto statement leads to difficulty in proving the correctness of a program.

Arguments for the goto statement:

- Goto statements have use in exceptional circumstances.
- They are sometimes necessary to make a program perform well.
- It is sometimes 'natural' to use goto statements.

The main goal of structured programming is to yield programs with optimum clarity. In this sense, the arguments against the goto statement outweigh the arguments for the goto statement.

11.5

```
      i:=start
 loop:
      IF array(i)=x then
         k=1
      endIF

      IF i=end then
         k=0
      endIF
      i:=i+1

 IF k=0 then
```

```
     write 'not Found'
     action1
endIF

IF k=1 then
     write 'Found'
     action2
endIF
end
```

The program above is structured - it follows logically from one line to the next. However, it has more conditional selections (i.e. if..then statements) giving it a higher cyclometric complexity measure than the first program; the measure for the non-structured program is 3 whereas the measure for the structured program if 5.

11.6 System design is crucially dependent on the so called system life cycle whose principal steps are as follows:

Specification $\Longrightarrow$ **Design** $\Longrightarrow$ **Testing** $\Longrightarrow$ **Software Maintenance**

Within the context of the above, the stages of this life-cycle are discussed below.

Specification

Specification is concerned with the formulation of software requirements in terms of:

- functions;
- operational constraints;
- external system behaviour;
- support environment;
- hardware on which software is to perform.

Design

Design deals with the realisation of code on the target system. This activity is dependent upon individual skill, attention to detail, knowledge of how best to use available tools and management organisation.

Testing

Testing involves:

- exercising the program using data which is similar to the real thing;
- observing the outputs;
- inferring program errors or inadequacies from anomalies in the output.

This can only be achieved through the establishment of a suitable design strategy which:

(i) tests to see if the individual components meet their requirements;

(ii) ensures that the integrated system functions perform correctly.

In practice, testing a module is done using a set of carefully selected 'test data'. Testing may be conducted by executing the program on the computer or by simulating its execution by a manual paper exercise called a 'dry run'. There are two basic types of testing:

- Functional testing or black box testing which is based upon typical, extreme and invalid data values that are representative of those covered by the specification.
- Logical testing or white box testing which is based upon examining the internal structure of the program and selecting data which gives rise to the alternative cases of control flow, e.g. both paths through an if..then..else.

Functional testing is used at the final stage of programming as a basis for accepting or rejecting the system.

Software Maintenance and Defect Amplification

Upon completion of the implementation stage, the software is typically transferred to operations staff where it must be maintained. Problems associated with software maintenance can invariably be traced to deficiencies in the way the software was designed and developed. A lack of control and discipline in the early stages of the software life-cycle nearly always translates into problems in the last stage. This leads to 'defect amplification' which refers to the following phenomenon: *During any phase of software development, errors may be generated. Errors that are not removed will be passed through to the next phase. Some of the errors that are passed through will have more significant ramifications on the next and/or subsequent phases.*

11.7 An important aspect of the software life-cycle is that it is dynamic. In other words, as the software is designed, coded, tested etc., any or all of the requirements (from specification to software maintenance) will invariably change and cycle back to one or all of the previous stages. This feature of software engineering is known as 'Bersoff's law of system engineering' which states:

'No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle'.

11.8 There are a variety of acceptable solutions to questions of this type but the solution provided should at least cover the following elements.

Functional Description of System

Functions (top level):

1. Input to system from sensors and buttons.

2. Processing of input data.

3. Output to activate responses or to update control data.

Tasks:

There are two types of tasks (at the highest level);

1. Button tasks

2. Activation tasks.

Button tasks; uses Button Sequence

Task	**Input**	**Output**
# 1 Adjust sensor levels	Adjusted sensor level	updated sensor levels
# 2 Select response list	Selected response list	updated response list
# 3 Set mode	[ON/OFF]	mode (on,off)
# 4 Inactivate	Inactivate	reset (to 'off')

Button Tasks are handled by a Button Task Manager subsystem that:

- Receives a Button Sequence
- Updates Control Data for detailed Button Tasks

Activation tasks

Task	**Input**	**Output**
Activate	Environmental conditions	Between 0 (or 1) and three responses activation mode = 'ON'

Activation Tasks are handled by an activation task manager subsystem using the following logic:

```
while (system mode = 'ON') do:

if((a given environmental condition)
     >=(respective sensor level)
```

```
     and (sensor not yet activated))
then

    BEGIN
      activation mode := 'ON'
      execute activation response for response list data;
    END
```

A.3.2 Solutions to Problems Given in Chapter 12

The C code provided here is not optimum. For example, it is not optimally structured and the commenting is rather sparse in order to minimize the number lines of code and reduce the length of the book. The code provided here aims to give the reader a 'first taste' of the software solutions to the problems specified which can be improved further and adapted to the 'tastes' of the programmer. Also, note that the .h libraries such as math.h and stdio.h have not been included and most lines of code defining a given function should include

```
#include<stdio.h>
#include<math.h>
```

for example. The name of each function is Qn where n is the number of the question. Where appropriate, the module is followed directly by a test unit for validating (or otherwise) the output of the module.

12.1

```
void Q1(float x[], float *xav, float *xmax, float *xmin, int n)
{
float sum,max,min;
int i;
/* Compute average value */
        sum=0.0;
           for(i=0; i<n; i++) sum=sum+x[i];
       *xav=sum/(float)n;
/* Compute maximum value */
max=0.;
for(i=0; i<n; i++)
        {
          if(x[i] > max)max=x[i];
}
       *xmax=max;
/* Compute minimum value */
min=max;
        for(i=0; i<n; i++)
        {
        if(x[i] < min)min=x[i];
```

```
}
       *xmin=min;
}
```

Test unit

```
main()
{
float x[10],xav,xmax,xmin;
int i,n;
        printf("How many numbers (<=10)?\n");
scanf("%d",&n);
        puts("Input numbers");

for(i=0; i<n; i++)  scanf("%f",&x[i]);

        Q1(x,&xav,&xmax,&xmin,n);

        printf("Average value = %f\n",xav);
        printf("Maximum value = %f\n",xmax);
        printf("Minimum value = %f\n",xmin);
}
```

12.2

```
void Q2(float x[], int n, int seed)
{
long int i,ix,r,p;
double max;
   /* Set parameter values */
         r=16807;
        p=2147483647;
   /* Start computation */
        ix=seed;
          for(i=0; i<n; i++)
             {
              ix=r*ix;
              ix=ix-(ix/p)*p; /* Equivalent to ix=ix%p in C */
              if(ix < 0) ix=p+ix;
              x[i]=ix;
              }
   /* Normalize output */
        max=0.0;
            for(i=0; i<n; i++) if(x[i] > max) max=x[i];
            for(i=0; i<n; i++) x[i]=x[i]/max;
}
```

Test unit

```
main()
{
float x[100];
int n=100,i,seed;
        puts("Input seed");
        scanf("%d",&seed);

        Q2(x,n,seed);

for(i=0; i<n; i++) printf("%f\n",x[i]);
}
```

12.3

```
void Q3(int x[], int h[], int n, int m)
{
int i,k;
        for(i=0; i<m; i++) h[i]=0; /* Initialize histogram */
           k=0;
           /* Compute histogram */
           for(i=0; i<n; i++)
           {
            k=x[i];
            h[k]=h[k]+1;
           }
}
```

Test unit

```
main()
{
int x[100],h[100],r,seed;
float y[100];
int i,n;

        puts("Input number of values required (<100)");
scanf("%d",&n);

        puts("Input range of values");
        scanf("%d",&r);
            printf("Input %d numbers between 0 and %d\n",n,r);
            for(i=0; i<n; i++) scanf("%d",&x[i]);
```

```
        Q3(x,h,n,r);

            puts("Histogram is");
            for(i=0; i<r; i++)printf("%d\n",h[i]);

        puts("Input number of random values required (<100)");
scanf("%d",&n);

        puts("Input range of values");
scanf("%d",&r);
             puts(" ");

        seed=12345;
        Q2(y,n,seed);

        for(i=0; i<n; i++)x[i]=(int)((float)r*y[i]);
for(i=0; i<n; i++)printf("%d\n",x[i]);

        Q3(x,h,n,r);

        puts("Histogram of random numbers is");
for(i=0; i<r; i++)printf("%d\n",h[i]);
}
```

12.4

```
void Q4(float xr[], float xi[], float yr[], float yi[], int n, int sign)
{
float pi,con;
float sumr,sumi,fac;
double atan(),cos(),sin();
int i,j;
   /* Compute constants */
pi=4.0*atan((double) 1.0);
con=2.0*pi/(float)n;
   /* Start computation */
for(i=0; i<n; i++)
        {
   /* Initialize */
sumr=0.0;
sumi=0.0;
for(j=0; j<n; j++)
                {
fac=con*(float)i*(float)j;
sumr=sumr+xr[j]*cos(fac)-isgn*xi[j]*sin(fac);
sumi=sumi+xi[j]*cos(fac)+isgn*xr[j]*sin(fac);
```

```
                }
    /* Normalize output according to whether a forward (sign=-1)
       or and inverse transform (sign=+1) is required */
        if(sign < 0)
        {
yr[i]=sumr/(float)n;
yi[i]=sumi/(float)n;
}
        if(sign > 0)
        {
yr[i]=sumr;
yi[i]=sumi;
}
    }
}
```

Test unit

```
#define m 10
main()
{
float xr[m],xi[m],yr[m],yi[m];
int i,n,isgn;

        puts("Input size of array (<=10)");
scanf("%d",&n);

        puts("Input number (real part,imaginary part)");
        for(i=0; i<n; i++) scanf("%f %f",&xr[i],&xi[i]);

isgn=-1;

        Q4(xr,xi,yr,yi,n,isgn);

        puts("Complex DFT is");
            for(i=0; i<n; i++)
               {
               printf("%f %f\n",yr[i],yi[i]);
               xr[i]=yr[i];
               xi[i]=yi[i];
               }

isgn=1;

        Q4(xr,xi,yr,yi,n,isgn);
```

```
        puts("Complex inverse DFT is");
        for(i=0; i<n; i++)printf("%f %f\n",yr[i],yi[i]);
}
```

12.5

```
void Q5(float f[],float g[],int n)
{
int i;
     /* Start computation */
        for(i=2; i<=n-1; i++) g[i]=(f[i-1]+f[i]+f[i+1])/3.0;
     /* Set end conditions */
g[1]=(f[1]+f[2])/3.0;
g[n]=(f[n]+f[n-1])/3.0;
}
```

Test unit

```
#define n 3
main()
{
float f[n],g[n];
int i;

        puts("Input data");
        for(i=1; i<=n; i++) scanf("%f",&f[i]);

        Q5(f,g,n);

        puts("Filtered data is");
for(i=1; i<=n; i++) printf("%f\n",g[i]);
exit(0);
}
```

12.6

```
void Q6(float *a[ ], float b[ ], float x[ ], int n, int maxit)
{
int i,j,k;
float sum;
/* Compute the initial solution. */
        for(i=0; i<n; i++) x[i]=b[i]/a[i][i];
/* Start the iteration process. */
for(k=1; k<=maxit; k++)
```

```
                {
         for(i=0; i<n; i++)
                        {
                         sum=b[i];
                               for(j=0; j<n; j++)
                               {
                                if(j != i)
                                sum=sum-a[i][j]*x[j];
                               }
                        x[i]=sum/a[i][i];
                        }
                 }
}
```

Test unit

```
#define n 3
main()
{
float a[n][n],b[n],x[n];
float *aa[n];
int i,maxit;

        puts("Input characteristic matrix of system");
puts(" ");

            puts("First row");
            for(i=0; i<n; i++) scanf("%f",&a[0][i]);

            puts("Second row");
            for(i=0; i<n; i++) scanf("%f",&a[1][i]);

            puts("Third row");
            for(i=0; i<n; i++){scanf("%f",&a[2][i]);aa[i]=a[i];}

               puts(" ");
               puts("Input data");
               for(i=0; i<n; i++) scanf("%f",&b[i]);

                   puts(" ");
                   puts("Input number of iterations");
                   scanf("%d",&maxit);

            puts(" ");

        Q6(aa,b,x,n,maxit);
```

```
            puts(" ");
            puts("Solution is");
                for(i=0; i<n; i++) printf("%f ",x[i]);
}
```

12.7

```
void Q7(float *a[ ], float b[ ], float x[ ], int n, int maxit)
{
float sum;
int i,j,k;
/* Compute the initial solution. */
for(i=0; i<n; i++)x[i]=b[i]/a[i][i];
/* Start the iteration process. */
for(k=1; k<=maxit; k++)
           {
            for(i=0; i<n; i++)
              {
               sum=b[i];
                   for(j=0; j<i; j++)sum=sum-a[i][j]*x[j];
                   for(j=i+1; j<n; j++)sum=sum-a[i][j]*x[j];
               x[i]=sum/a[i][i];
              }
           }
}
```

Test unit

As in Question 12.7.

12.8

```
void Q8(float a, float *root, int n)
int n;
{
float fnum(),fden();
float val;
x = (float *) calloc(n, sizeof(float)); /* Internal workspace */
int i;
        x[0]=a; /* Initialize using first approximation to root */
        /* Do computation */
        for(i=0; i<n; i++)
           {
             val=x[i];
```

```
                x[i+1]=x[i]-fnum(val)/fden(val);
                       *root=x[i];
          }
}

/* Internal functions for polynomial x^3-2x^2-5x+6 */

float fnum(x)
float x;
{
float y;
     /* Define polynomial using parenthesis (not powers) */
        y=6.0-x*(5.0+x*(2.0-x));
return(y);
}

float fden(x)
float x;
{
float y;
        y=-5.0-x*(4.0-3.0*x);
return(y);
}
```

Test unit

```
main()
{
float a,x;
int n;
        puts("Input initial approximation to root");
scanf("%f",&a);
        puts("Input number of iterations required (<=100)");
scanf("%d",&n);

        Q8(a,&x,n);

        puts(" ");
printf("Approximate value of root is %f",x);
}
```

12.9

```
void Q9(float approx, float a, float p, float *root, int n)
{
```

```
float x;
int i;
x = (float *) calloc(n, sizeof(float)); /* Internal workspace */
double pow();
     /* Take first approximation */
        x[0]=approx;
     /* Do computations */
        for(i=0; i<n; i++)
            {
             x[i+1]=(1/p)*((p-1)*x[i]+a/(pow((double)x[i],(double) p-1)));
            }
      /* Output result */
*root=x[i];
}
```

Test unit

```
main()
{
float a,x,p,ap;
int n;

        puts("Input number whose nth root is required");
scanf("%f",&a);
        puts("Input n-th root required");
scanf("%f",&p);
        puts("Input initial approximation to nth root");
scanf("%f",&ap);
        puts("Input number of iterations required (<=100)");
scanf("%d",&n);

        Q9(ap,a,p,&x,n);

        printf("Root %f of %f is %f",p,a,x);
}
```

12.10

```
void Q10(float x[ ], float f[ ], int n, float a0, float a1)
{
float sumx,sumf,sumt,sumb;
float avx,avf;
int i;
/* Compute average of x and f. */
sumx=0.0;
```

```
sumf=0.0;
        for(i=0; i<n; i++)
                {
sumx=sumx+x[i];
sumf=sumf+f[i];
                }
avx=sumx/(float)n;
avf=sumf/(float)n;
/* Compute a1. */
sumt=0.0;
sumb=0.0;
        for(i=0; i<n; i++)
                {
sumt=sumt+(x[i]-avx)*(f[i]-avf);
sumb=sumb+(x[i]-avx)*(x[i]-avx);
                }
/* Compute a1 and a0. */
*a1=sumt/sumb;
*a0=avf-*a1*avx;
}
```

Test unit

```
#define n 10
main()
{
float x[n],f[n];
float a0,a1,b0,b1;
int i;

        puts("Input a0 and a1");
scanf("%f %f",&a0,&a1);

        puts("Values of x(i) and f(i) are");
            for(i=1; i<=n; i++)
            {
            x[i]=(float)i/(float)n;
            f[i]=a0+a1*x[i];
                 printf("%f %f\n",x[i],f[i]);
            }

        Q10(x,f,n,&b0,&b1);

        put(" ");
        puts("Input parameters a0 and a1 are");
        printf("%f %f",a0,a1);
```

```
        puts(" ");
        puts("Calculated values of a0 and a1 are");
        printf("%f %f",b0,b1);
}
```

A.3.3 Supplementary Problems to Part III

III.1 Discuss the use of the following storage facilities:

- Main memory.
- Paging memory.
- Data base storage.
- Archival storage.

III.2 A serial stream of 1000 integer numbers ranging from 1 to 100 inclusive is to be analysed by counting the number times a particular integer occurs in the stream. The output of this analysis is to be an array containing a record of the number of times each integer has occurred.

- Write a logic flow diagram for this process.
- Write down an appropriate Pseudo code for the program.
- Write an appropriate subprogram which inputs the data and then outputs the result using ANSI C or else a programming language of your choice maximizing were possible algorithm efficiency.

III.3 Discuss some of the factors important in choosing a programming language.

III.4 The following pseudo-coded program has been designed to find the largest element and average value of a 3×3 matrix (2D array) of real positive numbers input from the keyboard and to write out the results on the screen during run time.

```
 * Input data. *

        write 'Input first row of data'
              read a(1,1),a(1,2),a(1,3)
        write 'Input second row of data'
              read a(2,1),a(2,2),a(2,3)
        write 'Input third row of data'
              read a(3,1),a(3,2),a(3,3)

 * Find the maximum value of the data. *
```

```
         max_val=0.0

          for j=1 to 3, do
           for i=1 to 3, do
            if a(j,i) > max_val then max_val=a(j,i)
           enddo
          enddo

 * Find the average value of the data. *

          for j=1 to 3, do
           for i=1 to 3, do
               sum=sum+a(j,i)
           enddo
          enddo

          av=sum/9

 * Write out results. *

          print: 'Maximum value of matrix is' max_val
          print: 'Average value of matrix is' av
```

The program is to be executed on a VMS machine. Although this code will compile, run and for certain data types even provide the correct answers, there are a number of features which collectively constitute (very) poor programming. What are they?

Rewrite the program using similar pseudo coding correctly in modular form using appropriate subprograms as appropriate to:

- Minimize the number of page faults.
- Provide the user with the option of repeating the program during run time.
- Provide the user with the option of inputting the data from a named disc file.

Convert the pseudo code into ANSI C code maximizing the efficiency of the code where possible.

Compute the McCabe cyclometric complexity of your program and compare it with the program given above. Comment on your result.

III.5 (a) Discuss the principles of modular programming; include statements on the following:

(i) Module size.

(ii) Complexity.

(iii) Coupling.

(iv) Cohesion.

(b) The following pseudo-coded function computes the factorial of a positive integer number n.

```
function factorial(input: n)
    if n = 0 then factorial=1
       i=1
          for j=1 to n, do
          i=j*i
factorial=i
```

(i) Write a literal translation of this function in C.

(ii) Write a C function to compute the factorial of a number using recursion.

(c) The void function *strcpy(s,t)* is to copy a character string s to string t. Write two versions of this C function to execute this process using

(i) Arrays.

(ii) Pointers.

III.6 Write a structured function in C to search an integer array a[i] for a integer number num. If the number is found the function should write out the sentence 'num found' and return a value of 1; if the number is not found, the function should write out the sentence 'num not found' and return a value of 0. The inputs to the function should include the integer array, the length of the array and the integer number being searched for.

III.7 Chebyshev's method for computing the solution to a set of linear simultaneous equations of the form

$$\sum_{j=1}^{n} a_{ij}x_j = b_i$$

is compounded in the iteration

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} - a_{ii}x_i^{(k)}\right)$$

where ω is the relaxation parameter and $x_i^{(1)} = b_i/a_{ii}$.

An independent and self consistent module is required to apply this method of solution. The module should input the following:

- the characteristic matrix of the system a_{ij};

- the data b_i;
- the size of the system to be solved n;
- the number of iterations required;
- the value of ω;

The module should output the solution x_i at the end of the process.

Error checks on the computations and checks on the validity of the input are not required but the module is to be written in structured form.

(i) Write an outline and then a detailed program flow chart for the module.

(ii) Write down appropriate pseudo code for the module.

(iv) Write an appropriate (void) function for the module using C provide suitably well commented code with optimum algorithm efficiency.

III.8 (a) Briefly describe how a software requirement specification can be developed into a good structured design, paying particular attention to the desirable attributes of the modules.

(b) The following pseudo code described a program designed to count positive and negative numbers, and calculate the mean of the positive inputs. It stops if the input value is zero, or the positive sum exceeds 1000.

```
       BEGIN
             plus=0
             minus=0
             total=0
loop:        READ(number)
                  IF number=0 THEN GOTO output
                  IF number>0 THEN GOTO positive
negative:    minus=minus+1
             GOTO loop
positive:    plus=plus+1
             total=total+number
             IF total<=1000 THEN GOTO loop
output:      mean=total/plus
             WRITE(plus,minus,mean)
       END
```

(i) Draw a control flow diagram for the pseudo code above.

(ii) Describe why it is non-structured and produce a structured version of the program in a control flow diagram and pseudo code form.

III.9 (a) Discuss some of the principal features (other than syntax) of C; include statements on the following:

(i) Layout.

(ii) Header files.

(iii) Structure.

(iv) Libraries.

(v) Pointers.

(b) The following pseudo-coded module has been designed to smooth data to reduce the effect of random error using a simple 3-element moving average filter of the type

$$Y_i = \frac{1}{3}(X_{i-1} + X_i + X_{i+1}); \quad i = 1, 2, ..., n$$

subject to the end conditions $X_0 = X_1$ and $X_{n+1} = X_n$.

```
function movav(x,n)
float x(100) *Input array with 100 words of memory*
float y(100) !Internal workspace
integer n !size of I/O array (<=100)

 * Compute filter *
       for i=2 to n-1 do
             y(i)=(x(i-1)+x(i)+x(i+1))/3.0
       enddo

 * Impose end conditions *
             y(1)=(x(1)+x(1)+x(2))/3.0
             y(n)=(x(n)+x(n)+x(n-1))/3.0

 * Return result *
       for i=1 to n, do
             x(i)=y(i)
       enddo
```

The module has been designed to return the output by overwriting the input array. In order to achieve this, internal workspace is required which has been given a maximum array size of 100 words.

(i) What is the basic problem with the general purpose application of this module.

(ii) Write a literal translation of this module in C using static arrays but employing pointers where appropriate.

(iii) Re-work your C translation to incorporate dynamic memory allocation where appropriate. Comment on the differences between your translation and the original

pseudo code given above with regard to the function being one of many modules forming an object library.

III.10 A simple but effective method of performing numerical integration i to use the trapezoidal rule for finding the area under a curve which is compounded in the formula

$$\int_{x_0}^{x_n} f(x)dx \simeq h\left(\frac{f(x_0)}{2} + f(x_1) + f(x_2) + ... + f(x_{n-1}) + \frac{f(x_n)}{2}\right)$$

were for N trapezoids of equal length,

$$h = \frac{x_n - x_0}{N}.$$

This method is to be used to evaluate the Fresnel integral

$$I(x) = \int_0^x \cos\left(\frac{\pi y^2}{2}\right) dy.$$

In independent and self consistent function is required to evaluate this integral using the trapezoidal rule which should input the following:

- The value of x.
- The number of trapezoids N.

The function should return the value of the integral I.

Error checks on the computations and checks on the validity of the I/O data are not required but the function is to be written in a structured form.

(i) Write an outline and then a detailed flow chart for this function.

(ii) Write appropriate pseudo code for the function.

(iii) Write appropriate C code for the function which is well commented with optimized algorithm efficiency.

(iv) Explain why the choice of N for a given value of x can critically affect the accuracy of the result.

III.10 (i) Explain what 'defect amplification' in software projects is and how software reviews can reduce both the problem and development costs.

(ii) Briefly explain the concept upon which Halstead's methods of determining software complexity are based and list the direct and calculated metrics.

(iii) List and describe in detail the different types of black box testing methods.

III.11 (i) In your own words, explain Bersoff's 'Law of System Engineering'. Why is it true? How does it affect software engineering paradigms, in particular, software configuration management.

(ii) Explain and compare 'configuration management audit' and 'formal technical review' for software projects. Would it be reasonable to combine them? Explain why.

(iii) List and discuss in detail the major types of software maintenance activities.

III.12 The bubble sort is a method of sorting an array of numbers into increasing values. The basic algorithm (for an array of size n and type float) is as follows:

```
START:
     for i:=1 to n-1; do:
         for j:=1 to n-i; do:

* Check if x(j) is greater than x(j+1);
  if true then exchange positions, i.e.
                 x(j) becomes x(j+1)
                 and
                 x(j+1) becomes x(j) *

 if x(j) > x(j+1)
 then
         temp=x(j)
         x(j)=x(j+1)
         x(j+1)=temp
 endif
 enddo
 enddo
```

(i) Design a C void function to implement this algorithm

```
void SORT(float x[], int n)
```

where x is the array and n is the array size.

(ii) Design an appropriate test unit to evaluate the function SORT.

(iii) Use the bubble sort algorithm to design a void function for computing the median of an array of numbers of odd size (i.e. an array of size 3, 5, 7, ...)

```
void MEDIAN(float x[], int n, int xmed}
```

where x is the input array of size n and xmed is the median value of the array that is output by this function.

Note that the median m of a set of numbers is such that half the numbers in the set are less than m and half of the numbers are greater than m, e.g. given the set $1, 6, 2, 4, 7, 3, 9$, then

$$(1, 6, 2, 4, 7, 3, 9) \rightarrow \text{bubble sort} \rightarrow (1, 2, 3, 4, 6, 7, 9)$$

and $m = 4$.

(iv) Design an appropriate test unit to evaluate the function MEDIAN.

III.13 For relatively small systems of linear equations, a method of solution is to use Cramers rule. In particular for a 3×3 system of equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_2 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_3 + a_{32}x_2 + a_{33}x_3 = b_3$$

the solution is

$$x_1 = \frac{1}{\mid A \mid}\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{23} & a_{33} \end{vmatrix}, \quad x_2 = \frac{1}{\mid A \mid}\begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}, \quad x_3 = \frac{1}{\mid A \mid}\begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}$$

where

$$\mid A \mid = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21}\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} a_{31}\begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}$$

and

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

(i) Design a module (void function) to compute the solution vector $x_1, x_2, x_3)$ using Cramers rule for a 3×3 system of linear equations.

(ii) Design a test unit to validate the module checking the output with an appropriate 'dry run'.

(iii) Design a module (void function) to interpolate three points (x_i, p_i); $i = 1, 2, 3$ using a quadratic polynomial

$$p(x) = a_0 + a_1x + a_2x^2$$

The function should input the values of the points (x_i, p_i); $i = 1, 2, 3$ and output a user defined number of values x_i and p_i for values of x_i between user defined lower and upper limits, the solutions for a_0, a_1 and a_3 being obtained using Cramers rule.

(iv) Check the output of the interpolation module by designing an appropriate test unit and implementing a suitable 'dry run' and then compute the trajectory of a projectile which is known to pass through the following points:

Position (m)	Height (m)
3030	2418
5835	3365
8293	2872

III.14 The Matthews cypher (Matthews, R, 1989, *On the Derivation of a Chaotic Encryption Algorithm*, Cryptologia No. 13, pp. 29-42) is a modification of the logistic mapping

$$x_{n+1} = rx_n(1 - x_n), \quad r \in (0, 4]$$

to

$$x_{n+1} = (1 + r)\left(1 + \frac{1}{r}\right) x_n(1 - x_n)^r, \quad r \in (0, 4]$$

which produces chaotic behaviour for a greater range of values of r. The output x_n is critically determined by the exact values of the initial condition $x_0 \in (0, 1)$ and the value of $r \in (1, 4]$. These values can be used as two keys to design a chaotic encryption engine based on a substitution cypher of the type

$$cypher = information + confusion$$

where the *confusion* field is given by x_n. In practice, this can be implemented using binary versions of the data and the XOR operation, i.e.

$$C_i = I_i \oplus X_i$$

where, in binary form, C_i is the 'cyphertext', I_i is the 'plaintext' and X_i is the 'cypher stream'.

(i) Design a module (void function) to generate a Matthews cypher for $r = 4$ which inputs the size of the array n required and the key x_0 and outputs a stream of numbers of type float:

```
void MATTHEWS(float x[], int n, float key)
```

Inspect the output of this function and compute the Lyapunov dimension (see Chapter 14) for different keys $x_0 \in (0, 1)$ using an appropriate function.

(ii) Design a main program that reads a plaintext file (in binary) specified during run-time and outputs the cyphertext to a file (in binary) specified during run time:

```
void ENCRYPT(float key)
```

(iii) Design a main program that reads a cyphertext file (in binary) specified during run-time and outputs the plaintext to a file (in binary) specified during run time:

```
void DECRYPT(float key)
```

(iv) Re-engineer the software to produce a single program that can both encrypt and decrypt using an appropriate switch:

```
void ED(float key, char option)
```

III.15 (a) Pearson's coefficient is a statistic that measures the correlation between two data sets $x_i, i = 1, 2, ..., N$ and $y_i, i = 1, 2, ..., N$. If the data x_i change as the data y_i change, then the two are correlated. Pearson's correlation coefficient r is given by the formula

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

where $\bar{x}$ is the mean (average) of the data x_i and $\bar{y}$ is the mean of the data y_i.

(i) Write a function to compute the average of an array of N values according to the following prototype:

```
double average(double a[], int N);
```

(ii) Write a function to input the data set from the keyboard for calculating Pearson's coefficient according to the following prototype:

```
void InputData(double x[], double y[], int N);
```

(iii) Using the function in (i) above, write a function to calculate Pearson's coefficient according to the prototype:

```
double pearson(double x[], double y[], int N);
```

(iv) Using the functions (i), (ii) and (iii) above, write a main function that calculates Pearson's coefficient from a set of 10 data values, $(x_i)_{i=1}^{10}$ and $(y_i)_{i=1}^{10}$ and outputs the result on the screen.

(b) (i) Write a second version of the function `InputData` that reads the data from a file according to the following prototype:

```
void InputData(double x[], double y[], FILE* f, int N);
```

where it is assumed that the file has been opened in the calling function.

(ii) Using the fact that `stdin` is a `FILE` pointer for the keyboard, modify you main program in Part (a)-(iv) to use the new `InputData` function. The main function should prompt the user as to whether they would like to input the data from the keyboard or from a file and if a file is selected, it should accept the name and open it for reading before calling the `InputData` function.

(iii) Indicate on another version of main what modifications you would make to the version of `main` from (ii) above of the memory for the array x_i and y_i is to be allocated dynamically with the size determined by the user.

III.16 (i) Describe the following derived types in C:

```
double* b[5];
int (*a)[3];
void (* f)(double);
int* (* g[5])(double);
```

(ii) Explain what is meant by the following terms:

- pass by value;
- pass by reference.

(iii) The following function computes the equation of the straight line $y = mx + c$ joining the two points $(x_1, y_1), (x_2, y_2)$:

```
int line(float x1, float y1, float x2, float y2, float m, float c)
{
     if (fabs(x1-x2)<1.0E-6) return -1;
     else {
           m=(y2-y1)/(x2-x1);
     return 0;
     }
}
```

The corresponding test unit for this function is:

```
void main()
{
    float X1, X2, Y1, Y2;
    float M, C;

    printf("input X1, Y1, X2, Y2\n");
    scanf("%f %f %f %f",&X1, &Y1, &X2, &Y2);
    if (line(X1,X2,Y1,Y2,M,C) == -1){
        printf("Cannot compute equation, exiting...\n");
        exit(1);
    }else {
        printf("Equation is Y=%fX + %f\n",M,C);
        exit(0);
    }
}
```

Explain why this program will *not* return the correct results for `m` and `c` to the `main` function.

(iv) Rewrite the function `line`and make the necessary modification to the `main` function so that the program executes correctly.

A.4 Part IV

The software solutions provided here are not ideal and have been condensed to reduce space. Where appropriate, commentary is provided mainly to explain the next line or lines of code that occur. Each module should have an appropriate header of the type discussed toward the end of Chapter 12 which explains the function of the module, I/O data and parameters, internal variables, the origin of the algorithm (as required) with appropriate references etc. Also, the style of the coding can be significantly improved upon by spacing it out, thereby making it easier to read and comprehend. The reader should consider these points if he/she is interested in reproducing the software provided here. The improvement of the code provided here is given as an exercise to the reader who wishes to make use of it.

A.4.1 Solutions to Problems Given in Chapter 13

13.1

```
void SPIKES( float s[], int n, int w)
{
  int nn, mm;

/*Determine mid point of array and position of spikes. */
  nn=1+(n/2);
  mm=1+(w/2);

/* Initialize signal array s. */
  for(i=1; i<=n; i++)s[i]=0.0;

/* Compute two spike signal nw units appart. */
    s[nn-mm]  =1.0;
    s[nn-mm+w]=1.0;
}
```

13.2

```
void TOPHAT(float s[], int n, int w)
{
  int nn,mm,i;

/* Determine mid point of array and position of sides of tophat.*/
  nn=1+(n/2);
  mm=1+(w/2);

/* Initialize signal array s and generate tophat signal of width w.*/
    for(i=1; i<=n; i++)s[i]=0.0;
```

```
    for(i=nn-mm+1; i<nn+mm; i++)s[i]=1.0;
}
```

13.3

```
void TRIANGLE(float s[], int n, int w)
{
  int nn,mm,i;
  float l1,l2;

/*Determine mid point of array and position of base.*/
  nn=1+(n/2);
  mm=1+(w/2);

/*Initialize signal array s.*/
  for(i=1; i<=n; i++)s[i]=0.0;

/* Generate left side (size l=mm) of triangle signal.*/
  l2=1.0/(l1=mm);
  for (i=1; i<=l1-1; i++)
    {
    s[nn-i]=1.0-i*l2;
    }

/*Generate center of triangle signal. */
  s[nn]=1.0;

/*Generate right side (size l=nw-mm) of triangle signal. */
  l2=1.0/(l1=nw-mm);
  for (i=1; i<=l1-1; i++)
    {
    s[nn+i]=1.0-i*l2;
    }
}
```

13.4

```
#include<math.h>

void GAUSSIAN(float s[], int n, int w)
{
  int nn, i;
  float x, sigma;

/*Determine mid point of array.* /
```

```
  nn=1+(n/2);

/*Generate Gaussian signal.*/
  sigma=(float)w;
  for(i=1; i<=n; i++)
    {
    x=(float)(i-nn);
    s[i]=exp(-(x*x)/(sigma*sigma));
    }
}
```

13.5

```
#include<math.h>

void COSINE(float s[], int n, int p)
{
  int nn, i;
  float pi, scale;

/*Determine mid point of array, value of pi, and scale factor.*/
  nn=1+(n/2);
  pi=4.0*atan(1.0);
  scale=p*pi/(nn-1);

/*Create p periods of cosine.*/
  for(i=1; i<=n; i++)s[i]=cos(scale*(i-nn));
}
```

13.6

```
#include<math.h>

void SINE( float s[], int n, int p )
{
  int nn, i;
  float pi, scale;

/*Determine mid point of array, value of pi, and scale factor.*/
  nn = 1+(n/2);
  pi = 4.0*atan(1.0);
  scale=p*pi/(nn-1);

/*Create p periods of sine wave.*/
  for(i=1; i<=0; i++) s[i]=sin(scale*(i-nn));
}
```

13.7 The following program is an example of the code that can be used to investigate the application of FFT1D using the cosine wave generator (i.e. Question 13.6) for example.

```
#include <stdio.h>
#include <math.h>
#define n 128

int main(void)
    {
    char ans;
    int i, p, sgn;
    float sr[n+1],si[n+1];

start: printf("input no. of periods\n");
scanf("%d",&p);

              COSINE(sr,n,p);
              gopen();
              gsignal(sr,n,1);

              for(i=1; i<=n; i++) si[i]=0.0;

sgn=-1;/*To compute forward FFT*/
              FFT1D(sr,si,n,sgn);
              /*Good practice to use this form rather than
                FFT1D(sr,si,n,-1) especially in a test unit*/

              gsignal(sr,n,1);
              gsignal(si,n,1);

sgn=1;/*To compute inverse FFT*/
              FFT1D(sr,si,n,sgn);
              gsignal(sr,n,1);
              gsignal(si,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
gclose();
return 0;
}
```

13.8

```
#include <math.h>

void AMPSPEC(float s[], float a[], int n)
{
  int i;
  float *si;

/*Allocate space for work arrays.*/
  si = (float *) calloc( n+1, sizeof( float ) );

/*Initialize real and imaginary working arrays for the signal.*/
  for (i=0; i<n; i++)
    {
    a[i]=s[i];
    si[i]=0.0;
    }

/*Compute the DFT of signal s (=a).*/
  FFT1D(a, si, n, -1);

/*Calculate the amplitude spectrum of the signal.*/
  for (i=1; i<=n; i++)
    {
    a[i] = sqrt(a[i]*a[i]+si[i]*si[i]);
    }

/*Free space from work arrays.*/
  free(si);
}
```

13.9

```
#include <math.h>

void POWSPEC(float s[], float p[], int n)
{
  int i;
  float *si;

/*Allocate space for work arrays.*/
  si = (float *) calloc(n+1, sizeof(float ));

/*Initialize real and imaginary working arrays for the signal.*/
  for (i=1; i<=n; i++)
    {
    p[i]  = s[i];
```

```
    si[i] = 0.0;
    }

/*Compute the DFT of s.*/
  FFT1D(p, si, n, -1);

/*Calculate the power spectrum of the signal.*/
  for (i=1; i<=n; i++)
    {
    p[i] = p[i]*p[i]+si[i]*si[i];
    }

/*Free space from work arrays.*/
  free(si);
}
```

The following code provides the power spectrum of a tophat function using the module TOPHAT for an array of size 256. Similar code can be used to investigate other signals as required.

```
#include <math.h>
#define n 256

int main(void)
{
char ans;
int w;
float f[n+1], s[n+1];

start: printf("Input width of tophat function\n");
scanf("%d",&w);

TOPHAT(f,n,w);
gopen();
gsignal(f,n,1);

POWSPEC(s,p,n);
gsignal(s,n,1);

for(i=1; i<=n; i++)s[i]=log(1+s[i]);
gsignal(s,n,1)

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
gclose();
```

```
return 0;
}
```

13.10

```
#include <math.h>

void SCALE(float s[], int n, long float a)
{
    int i;
    float max, temp;

/*Search the array for the largest entry */
  max=0.0;
  for(i=1; i<=n; i++)
     {
      temp=fabs(s[i]);
      if(temp > max)
        {
         max=temp;
        }
      }

/*Scale the array by max - equivalent to using a uniform norm */
if(max != 0)
  {
   max =1.0/max;
   for(i=1; i<=n; i++)
      {
       s[i]=s[i]*max*a;
      }
   }
}
```

13.11

```
#include <math.h>

static float VALUE(int i);
static int nn;

void PARZEN( float w[], int n )
{
  int i;
```

```
/*Determine mid point of array.*/
  nn=1+(n/2);

/*Generate Parzen Window taking into account the symmetry.*/

  w[1]=VALUE(1);
  for (i=2; i<=nn-1; i++)
    {
    w[i]    =VALUE(i);
    w[n-i+2]=w[i];
    }

  w[nn]=VALUE(nn);
}

/*Internal function to determine function value.*/

static float VALUE(int i)
{
  return(1 - fabs((i-nn)/(nn-1.0)));
}

/************************************************/

#include <math.h>

static float VALUE(int i);
static int nn;

void WELCH(float w[], int n)
{
  int i;

/*Determine mid point of array.*/
  nn=1+(n/2);

/*Generate Welch Window taking into account the symmetry.*/
  w[1]=VALUE(1);
  for (i=2; i<=nn-1; i++ )
    {
    w[i]    =VALUE(i);
    w[n-i+2]=w[i];
    }

  w[nn]=VALUE(nn);
}
```

```
/*Internal function to determine value*/

static float VALUE(int i)
{
  return(1.0 - pow((i-nn)/(nn-1), 2));
}

/*************************************************/

void HANNING(float w[], int n)
{
  int i;

/* Generate one period of a cosine signal.*/
   COSINE(w,n,1);

/*Generate Hanning Window using the cosine signal from above.*/

  for(i=1; i<=n; i++)
    {
    w[i]=0.5+0.5*w[i];
    }
}

/*************************************************/

void HAMMING(float w[], int n)
{
  int i;

/*Generate one period of a cosine signal.*/
  COSINE(w,n,1);

/*Generate Hamming Window using the cosine signal from above.*/

  for(i=1; i<=n; i++)
     {
     w[i]=0.54+0.46*w[i];
     }
}
```

13.12

```
void DIF(float s[], int n)
{
  int nn, i;
```

```
  float *si, temp;

/*Allocate space for work arrays.*/
  si = (float *) calloc(n+1, sizeof(float));

/*Determine mid point of array.*/
  nn=1+(n/2);

/*Initialize imaginary working arrays for the signal.*/
  for(i=1; i<=n; i++)si[i]=0.0;

/*Compute the Discrete Fourier Transform of signal*/
  FFT1D(s,si,n,-1);

/*Multiply spectrum by Fourier filter for a differential*/
  for(i=1; i<=n; i++)
     {
     temp  = s[i]*(i-nn);
     s[i]  = -1.0*si[i]*(i-nn);
     si[i] = temp;
     }

/*Compute the Inverse DFT signal s.*/
  FFT1D(s,si,n,1);

/*Free space from work arrays.*/
  free( si );
}
```

The output from this module produces ringing - the Gibbs effect. This is due to the discontinuity (at high frequency values) associated with the frequency 'ramp' that is applied to compute the differential.

13.13

```
void CONVOLVE( float f[], float p[], float s[], int n )
{
  int i;
  float *fr, *fi, *pr, *pi, *si;

/*Allocate space for internal work arrays.*/

  fr = (float *) calloc( n+1, sizeof( float ) );
  fi = (float *) calloc( n+1, sizeof( float ) );

  pr = (float *) calloc( n+1, sizeof( float ) );
  pi = (float *) calloc( n+1, sizeof( float ) );
```

```
  si = (float *) calloc( n+1, sizeof( float ) );

/*Initialize real and imaginary working arrays*/

  for(i=1; i<=n; i++)
    {
    fr[i] = f[i];
    fi[i] = 0.0;

    pr[i] = p[i];
    pi[i] = 0.0;
    }

/*Compute the DFT of signals f and p.*/
    FFT1D(fr,fi,n,-1);
    FFT1D(pr,pi,n,-1);

/*Compute the product of the complex Fourier transforms*/
  for(i=1; i<=n; i++)
    {
    s[i]  = (fr[i] * pr[i]) - (fi[i] * pi[i]);
    si[i] = (fr[i] * pi[i]) + (pr[i] * fi[i]);
    }

/*Compute the inverse DFT*/
  FFT1D(s,si,n,1);

/*Free internal memory assigned to internal arrays.*/
  free( fr );
  free( fi );

  free( pr );
  free( pi );

  free( si );
}
```

The following test unit prompts the user to input the standard deviation of the Gaussian function and convolves it with two spikes 32 elements apart. Each signal is displayed using gsignal for analysis.

```
#include <stdio.h>
#define n 512

void main(void)
```

```
{

char ans;
int i,w;

float s1[n+1],s2[n+1],s3[n+1];

start: printf("input width (standard deviation) of Gaussian\n");
       scanf("%d",&w);

       GAUSSIAN(s1,n,w);
       gopen();
       gsignal(s1,n,1);

       SPIKES(s2,n,32);
       gsignal(s2,n,1);

       CONVOLVE(s1,s2,s3,n);
       gsignal(s3,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

13.14

```
void CROSCOR( float f[], float p[], float s[], int n )
{
  int i;
  float *fr, *fi, *pr, *pi, *si;

/*Allocate internal work space*/
  fr = (float *) calloc(n+1, sizeof(float));
  fi = (float *) calloc(n+1, sizeof(float));

  pr = (float *) calloc(n+1, sizeof(float));
  pi = (float *) calloc(n+1, sizeof(float));

  si = (float *) calloc(n+1, sizeof(float));

/*Initialize real and imaginary arrays.*/

  for(i=1; i<=n; i++)
```

```
     {
     fr[i] = f[i];
     fi[i] = 0.0;

     pr[i] = p[i];
     pi[i] = 0.0;
     }

/*Compute the DFT of f and p.*/
  FFT1D(fr,fi,n,-1);
  FFT1D(pr,pi,n,-1);

/*Calculate the product of the complex Fourier transform of F and
the complex conjugate of P*/
  for(i=1; i<=n; i++)
     {
     s[i]  = (pr[i] * fr[i]) + (pi[i] * fi[i]);
     si[i] = (pi[i] * fr[i]) - (pi[i] * fr[i]);
     }

/*Compute the Inverse DFT*/
     FFT1D(s,si,n,1);

/*Free space from work arrays.*/
  free( fr );
  free( fi );

  free( pr );
  free( pi );

  free( si );
}
```

The test unit for this module is the same as that provided in the previous question. Cross correlation of a sine wave with the same sine wave produces a spike. The correlation of a sine wave with a cosine wave does not produce a spike. In general, the correlation of a signal f with a matching signal or template p produces a peak or spike in the output - the correlation function. This is one of the fundamental methods of pattern recognition for example.

13.15

```
#include <math.h>

void AUTOCOR(float f[], float s[], int n)
{
  int i;
```

```
  float *si;

/*Allocate space for work arrays.*/
  si = (float *) calloc( n+1, sizeof( float ) );

/*Initialize real and imaginary working arrays for the signal.*/

  for(i=1; i<=n; i++)
     {
     s[i]  = f[i];
     si[i] = 0.0;
     }

/*Compute the DFT*/
     FFT1D( s, si, n, -1 );

/*Compute the product of the complex Fourier transforms F and its
complex conjugate*/
   for( i=1; i<=n; i++ )
      {
       s[i]  = s[i]*s[i]+si[i]*si[i];
       si[i] = 0.0;
       }

/*Compute the inverse DFT*/
       FFT1D( s, si, n, 1 );

/*Free internal memory*/

  free( si );
}
```

An example test unit is given below.

```
#include <stdio.h>
#define n 512

void main(void)
{

char ans;
int i,w;

float s1[n+1],s2[n+1],s3[n+1];

start: printf("input width of Tophat function\n");
```

```
       scanf("%d",&w);

       TOPHAT(s1,n,w);
       gopen();
       gsignal(s1,n,1);

       AUTOCOR(s1,s2,n);
       gsignal(s2,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

The output is a triangle.

13.16

```
void FILTER( float s[], float f[], int n )
{
  float *si;

/*Allocate space for work arrays.*/
  si = (float *) calloc( n+1, sizeof( float ) );

/*Initialize imaginary array.*/
  for(i=1; i<=n; i++)si[i]=0.0;

/*Compute the DFT.*/
  FFT1D( s, si, n, -1 );

/*Compute the product of the complex Fourier transform
with the input filter.*/
 for(i=1; i<=n; i++)
    {
    s[i]=f[i]*s[i];
    si[i]=f[i]*si[i];
    }

/*Compute the inverse DFT.*/
  FFT1D( s, si, n, 1 );

/*Free work space.*/
  free( si );
```

```
}

/************************************************/

#include <stdio.h>
#define n 512

void main(void)
{

char ans;
int i,w,p;

float f[n+1],s[n+1];

start: printf("input width of tophat function\n");
       scanf("%d",&w);

       TOPHAT(f,n,w);
       gopen();
       gsignal(f,n,1);

       p=10;
       COSINE(s,n,p);
       gsignal(s,n,1);

       FILTER(s,f,n);
       gsignal(s,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

When the bandwidth of the Tophat filter is greater than that of the cosine wave whose Fourier transform is a delta function (in the negative and positive half spaces), the wave is reproduced. However, when the bandwidth is less than that of the cosine wave so that the delta functions are effectively set to zero, the cosine wave 'vanishes'.

13.17

```
void ANASIG( float f[], float q[], int n )
{
  int nn, i;
```

```
/*Determine mid point of array.*/
  nn = 1 + (n/2);

/*Initialize imaginary working array for the signal.*/
  for(i=1; i<=n; i++) q[i]=0.0;

/*Compute the DFT.*/
  FFT1D( f, q, n, -1 );

/*Set the negative frequencies of the DFT to zero.*/
  for ( i=1; i<=nn-1; i++ )
    {
    f[i] = 0.0;
    q[i] = 0.0;
    }

/*Scale the positive frequencies and DC of the DFT by 2.*/
  for ( i=nn; i<=n; i++ )
    {
    f[i] = 2.0 * f[i];
    q[i] = 2.0 * q[i];
    }

/*Compute the inverse DFT.*/
  FFT1D( f, q, n, 1 );
}
```

13.18

```
void HILBERT( float s[], int n )
{
  float *si;

/*Allocate space for work arrays.*/
  q = (float *) calloc( n+1, sizeof( float ) );

/*Compute the analytic signal.*/
  ANASIG( s, q, n );

/*Write quadrature component to output signal s.*/
  for(i=1; i<=n; i++)s[i]=q[i];

/*Free space from work arrays.*/
  free( q );
}
```

Test unit

```
#include <stdio.h>
#define n 512

void main(void)
{

char ans;
int i,p;

float s[n+1];

start: printf("input number of periods of sine wave\n");
       scanf("%d",&p);

       SINE(s,n,p);
       gopen();
       gsignal(s,n,1);


       HILBERT(s,n,p);
       gsignal(s,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

13.19

```
void AMPENV( float s[], float a[], int n )
{
  int i;
  float *q;

/*Allocate internal memory.*/
  q = (float *) calloc( n+1, sizeof( float ) );

/*Copy array s to array a*/
  for(i=1; i<=n; i++) a[i]=s[i];

/*Compute the analytic signal.*/
  ANASIG( a, q, n );
```

```
/*Compute amplitude envelope of the analytic signal.*/

  for ( i=1; i<=n; i++ )
    {
    a[i] = a[i]*a[i]+q[i]*q[i]
    }

/*Free space from work arrays.*/
  free( q );
}
```

Test unit.

```
#include <stdio.h>
#define n 512

void main(void)
{

char ans;
int i,p;

float s[n+1], a[n+1];

start: printf("input number of periods of sine wave\n");
       scanf("%d",&p);

       SINE(s,n,p);
       gopen();
       gsignal(s,n,1);


       AMPENV(s,a,n);
       gsignal(a,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

13.20

```
void SINCINT( float x[], int n, float y[], int m )
```

```
{
  int nn, mm, i;
  float *xr, *xi, *yi, scale;

/*Allocate internal memory.*/

  xr = (float *) calloc( n+1, sizeof( float ) );
  xi = (float *) calloc( n+1, sizeof( float ) );

  yi = (float *) calloc( m+1, sizeof( float ) );

/*Determine mid point of array.*/

  nn = 1 + (n/2);
  mm = 1 + (m/2);

/*Initialize working arrays.*/

  for( i=1; i<=n; i++ )
    {
    xr[i] = x[i];
    xi[i] = 0.0;
    }

  for ( i=1; i<=m; i++ )
    {
    y[i]  = 0.0;
    yi[i] = 0.0;
    }

/*Compute the DFT.*/

  FFT1D( xr, xi, n, -1 );

/*Zero-pad n-point spectrum to give m-point spectrum.*/
  for (i=1; i<=n; i++)
      {
      y[mm-nn+i]=xr[i];
      yi[mm-nn+i]=xi[i];
      }

/*Compute the inverse DFT.*/
  FFT1D( y, yi, m, 1 );

/*Scale sinc-interpolated signal by (m/n).*/
  scale = m / n;
  for ( i=1; i<=m; i++ )
```

```
    {
    y[i] = y[i] * scale;
    }

/*Free internal memory.*/
  free( xr );
  free( xi );

  free( yi );
}
```

Test unit.

```
#include <stdio.h>
#define n 128
#define m 256

void main(void)
{

char ans;
int i,p;

float s[n+1], ss[n+1];

start: printf("input number of periods of sine wave\n");
       scanf("%d",&p);

       SINE(s,n,p);
       gopen();
       gsignal(s,n,1);


       SINCINT(s,n,ss,m);
       gsignal(ss,m,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

A.4.2 Solutions to Problems Given in Chapter 14

14.1

```
void ILF( float s[], int n, int bw )
{
  float *fx;

/*Allocate space for work arrays.*/
  fx = (float *) calloc( n+1, sizeof( float ) );

/*Create Tophat filter of bandwidth bw.*/
  TOPHAT( fx, n, wb );

/*Filter signal with Tophat filter.*/
  FILTER( s, fx, n );

/*Free space from work arrays.*/
  free( fx );
}
```

14.2

```
void GLF( float s[], int n, int bw )
{
  float *fx;

/*Allocate internal memory.*/
  fx = (float *) calloc( n+1, sizeof( float ) );

/*Create Gaussian filter of bandwidth bw.*/
  GAUSSIAN( fx, n, bw);

/*Filter signal with Gaussian filter.*/
  FILTER( s, fx, n );

/*Free space from work arrays.*/
  free( fx );
}
```

14.3

```
#include<math.h>

void BLF( float s[], int n, int bw, int ord )
{
  int nn;
  float *fx, div;
```

```
  nn=1+(n/2);

/*Allocate space for internal array.*/
  p = (float *) calloc( n+1, sizeof( float ) );

  ord=2*ord;
/*Create Butterworth filter of bandwidth bw and order ord.*/
  for(i=1; i<=n; i++)
  {
  div=(float) (i-nn)/bw;
  p[i] = 1.0 / (1.0 + pow(div,ord));
  }

/*Filter signal with Butterworth filter.*/
  FILTER( s, p, n );

/*Free space from work arrays.*/
  free( p );
}
```

14.4

```
void IHF( float s[], int n, int bw )
{
  float *p;

/*Allocate space for work array.*/
  p = (float *) calloc( n+1, sizeof( float ) );

/*Create ideal highpass filter of bandwidth wb.*/
  nn=1 + (n/2);  /*Set position of DC level.*/
  mm=2*bw;
  for ( i=1; i<=n; i++ ) p[i]=1.0;  /*Initialize.*/

        for ( i=nn-bw+1; nn-bw+mm; i++ ) p[i]=0.

/*Filter signal.*/
  FILTER( s, p, n );

/*Free internal memory.*/
  free( p );
}

/************************************************/
```

```
void GHF( float s[], int n, int bw )
{
  int i, nn;
  float *g, x, sigma;

/*Allocate internal memory.*/
  g = (float *) calloc( n+1, sizeof( float ) );

/*Create Gaussian highpass filter of bandwidth bw.*/

/*Determine mid point of array.* /
  nn=1+(n/2);

/*Generate Gaussian signal.*/
  sigma=(float) bw;
  for(i=1; i<=n; i++)
    {
    x=(float)(i-nn);
    g[i]=exp((x*x)/(sigma*sigma)) - 1;
    }

/*Filter signal*/
  FILTER( s, g, n );

/*Free internal memory.*/
  free( g );
}

/************************************************/

\begin{verbatim}
#include<math.h>

void BHF( float s[], int n, int bw, int ord )
{
  float *fx, div, x;

/*Allocate space for internal array.*/
  p = (float *) calloc( n+1, sizeof( float ) );

  ord=2*ord;
/*Create Butterworth highpass filter of bandwidth bw and order ord.*/
  for(i=1; i<n+1; i++)
  {
     if((i-nn) != 0)
     {
     div=(float) bw/(i-nn);
```

```
     p[i] = 1.0 / (1.0 + pow(div,ord));
     }
  }
     p[nn]=0.0;/*Set midpoint value to zero.*/

/*Filter signal.*/
  FILTER( s, p, n );

/*Free space from work arrays.*/
  free( p );
}
```

14.5

```
#include <math.h>
long float pow(long float x, long float y);

void FRAC_FIL( float s[], int n, int seed, long float q )
{
  int i,nn=1+n/2;
  float pi, S, C;
  float *sr, *si;
  long float denom;
  long float omega;

  pi=4.0*atan(1.0);

/*Allocate memory to internal work space.*/
  si = (float *) calloc( n+1, sizeof( float ) );

/*Compute Gaussian noise field.*/
  GNOISE(s,n,seed);

/*Compute the DFT.*/
  FFT1D( s, si, n, -1 );

/*Compute constants associated with filter.*/
  C=cos(pi*q/2);
  S=sin(pi*q/2);

/*Apply the Fractal Filter.*/
  for ( i=1; i<=nn-1; i++ )
    {
     omega=(long float) (nn-i);
     denom=pow(omega, q);
     s[i]  = (C*s[i]-S*si[i])/denom;
```

```
     si[i] = (S*s[i]+C*si[i])/denom;
    }

  for ( i=nn; i<=n; i++)
    {
     omega=(long float) (i-nn);
     denom=pow(omega, q);
     s[i]  = (C*s[i]-S*si[i])/denom;
     si[i] = (S*s[i]+C*si[i])/denom;
    }

/*Note: DC components (where the filter is singular) remain that same.*/

/*Compute the inverse DFT.*/
  FFT1D( s, si, n, 1 );

/*Free space from work array.*/
  free( si );

}
```

14.6

```
#include<math.h>

void WIENER( float s[], float p[], float f[], int n, long float snr )
{
  int i;
  float *sr, *si, *pr, *pi, *fi, gamma, denom;

/*Allocate memory for internal arrays.*/

  sr = (float *) calloc( n+1, sizeof( float ) );
  si = (float *) calloc( n+1, sizeof( float ) );

  pr = (float *) calloc( n+1, sizeof( float ) );
  pi = (float *) calloc( n+1, sizeof( float ) );

  fi = (float *) calloc( n+1, sizeof( float ) );

/*Compute gamma from SNR.*/
  gamma = 1.0 / pow( snr, 2 );

/*Initialize real and imaginary arrays.*/
  for ( i=1; i<=n; i++ )
    {
```

```
    sr[i] = s[i];
    si[i] = 0.0;

    pr[i] = p[i];
    pi[i] = 0.0;
    }

/* Compute the DFTs of signals.*/
   FFT1D( sr, si, n, -1 );
   FFT1D( pr, pi, n, -1 );

/*Apply Wiener Filter.*/
  for ( i=1; i<=n; i++ )
      {
       denom = 1.0 / (pow( pr[i], 2 ) + pow( pi[i], 2 ) + gamma);
       f[i]  = (sr[i]*pr[i] + si[i]*pi[i]) * denom;
       fi[i] = (si[i]*pr[i] - sr[i]*pi[i]) * denom;
      }

/*Compute inverse DFT.*/
  FFT1D( f, fi, n, 1 );

/*Free internal memory.*/
  free( sr );
  free( si );

  free( pr );
  free( pi );

  free( fi );
}
```

14.7 The test unit given below is designed to investigate the application of the Wiener filter in a noise free environment. It illustrates the effect of convolution and deconvolution using a Fourier filtering approach. The purpose of using two spikes is to provide a test which illustrates the restoration of information in terms of the 'resolution' of a known input, i.e. the distance between the spikes.

```
#include <stdio.h>
#define n 128

void main(void)
{

char ans;
int i, w, snr;
```

```
float f[n+1], p[n+1], s[n+1];

       SPIKES(f,n,10)
       gopen();
       gsignal(f,n,1);

start: printf("input width of Gaussian\n");
       scanf("%d",&w);

       GAUSSIAN(p,n,w);
       gsignal(p,n,1);

       CONVOLVE(f,p,s,n);
       gsignal(s,n,1);

printf("input snr\n");
scanf("%f",&snr);

       WIENER(f,p,s,n,snr);
       gsignal(s,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

14.8 Note that the values of the SNR used in the restoration of a signal with the Wiener filter does not correlate directly to the snr of the signal.

```
#include <stdio.h>
#include <math.h>
#define n 128

void main(void)
{

char ans;
int i, w;
long float snr, SNR;
long float a=1;

float p[n], f[n+1], s[n+1], Noise[n+1];
```

```
       SPIKES(f,n,10);
       gopen();
       gsignal(f);

start: printf("input standard deviation of Gaussian\n");
       scanf("%d",&w);

       GAUSSIAN(p,n,w);
       gsignal(p,n,1);

       CONVOLVE(f,p,s,n);
       gsignal(s,n,1);

       GNOISE(Noise,n,123);
       gsignal(Noise,n,1);

       SCALE(s,n,a);
       SCALE(Noise,n,a);

printf("input snr\n");
scanf("%f",&snr);

for(i=0; i<n; i++)
{
s[i]=snr*s[i]+Noise[i];
}
gsignal(s,n,1);

printf("input SNR for Wiener restoration of signal\n");
scanf("%f",&SNR);

       WIENER(s,p,f,n,SNR);
       gsignal(f,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}
```

14.9

```
void MATFIL( float s[], float p[], float f[], int n )
{
  int i;
```

```
  float *sr, *si, *pr, *pi, *fi;

/*Allocate internal memory.*/

  sr = (float *) calloc( n+1, sizeof( float ) );
  si = (float *) calloc( n+1, sizeof( float ) );

  pr = (float *) calloc( n+1, sizeof( float ) );
  pi = (float *) calloc( n+1, sizeof( float ) );

  fi = (float *) calloc( n+1, sizeof( float ) );

/*Initialize real and imaginary arrays.*/
  for( i=1; i<=n; i++ )
     {
     sr[i] = s[i];
     si[i] = 0.0;

     pr[i] = p[i];
     pi[i] = 0.0;
     }

/*Compute the DFT of signal s and p.*/
  FFT1D( sr, si, n, -1 );
  FFT1D( pr, pi, n, -1 );

/*Apply the Matched Filter.*/
  for ( i=1; i<=n; i++ )
    {
    f[i]  = ( sr[i]*pr[i] + si[i]*pi[i] );
    fi[i] = ( si[i]*pr[i] - sr[i]*pi[i] );
    }

/*Compute the inverse DFT.*/
  FFT1D( f, fi, n, 1 );

/*Free internal memory.*/
  free( sr );
  free( si );

  free( pr );
  free( pi );

  free( fi );
}
```

14.10 The main program given below, provides a test unit for investigating the restora-

tion of a signal (two spikes) when it has been convolved with a linear FM chirp. The restoration of the signal in the presence of additive noise is remarkably robust and provides an excellent reconstruction when the snr (as defined here) is very low, i.e. snr=1 or less. As in question 14.8, the use of two spikes, provides an estimation of the resolving power of the method in terms of the extraction of a signal from noise.

```
#include <stdio.h>
#include <math.h>
#define n 512

void main(void)
{

char ans;
int i, w, snr;
long float a=1;

float f[n+1], p[n+1], s[n+1], Noise[n+1];

       SPIKES(f,n,30);

start: printf("input width of chirp\n");
       scanf("%d",&w);


       CHIRP(p,n,w);
       gopen();
       gsignal(p,n,1);
       gsignal(f,n,1);

       CONVOLVE(f,p,s,n);
       gsignal(s,n,1);

       GNOISE(Noise,n,4526);
       gsignal(Noise,n,1);
       SCALE(s,n,a);
       SCALE(Noise,n,a);

printf("input snr\n");
scanf("%f",&snr);

for(i=1; i<=n; i++)
{
s[i]=snr*s[i]+Noise[i];
}
gsignal(s,n,1);
```

```
       MATFIL(s,p,f,n);
       gsignal(f,n,1);

printf("again?<y>\n");
scanf("%s",&ans);
if(ans=='y')goto start;
else
exit(0);
}

/************************************************/
#include <math.h>

void CHIRP(float s[], int n, int nw)

{
int nn, mm, i, j, jj;
float pi;

   pi=4.0*atan(1.0);

/*Determine mid points of the array.*/

   nn=1+(n/2);
   mm=1+(nw/2);

/*Initialize array s[] just in case.*/

   for(i=1; i<=n; i++)s[i]=0.0;

/* Compute chirp. */
j=0;
    for(i=nn-mm+1; i<=nn+mm; i++)
       {
       jj=j*j;
       s[i]=sin(2*pi*(float)jj/n);
       j=j+1;
       }
}
```

A.4.3 Solutions to Problems Given in Chapter 15

15.1 The ML estimate is obtained by finding the coefficients A_n which satisfy the condition

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{s} \mid \mathbf{f}) = 0.$$

The probability of measuring s given f is determined by the noise since

$$n(t) = s(t) - f(t).$$

If the noise is zero-mean Gaussian with standard deviation σ_n^2, then,

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T}^{T} \mid s(t) - f(t) \mid^2 dt \right)$$

and the ML estimate is given by solving the equation

$$-\frac{1}{2\sigma_n^2} \frac{\partial}{\partial A_m} \int_{-T}^{T} \mid s(t) - w(t) \mid^2 \frac{1}{w(t)} dt$$

which yields

$$S(\omega_m) = \sum_n A_n W(\omega_m - \omega_n).$$

Using the convolution theorem, we have

$$s_{BL}(t) = a(t) w_{BL}(t)$$

and hence, the ML estimate is

$$f_{ML}(t) = \frac{w(t)}{w_{BL}(t)} s_{BL}(t).$$

Note, that this result is identical to the least mean square approximation.

15.2 Note, that the Rayleigh distribution is a very common and generally accurate model for the random amplitude fluctuations a of a signal where

$$P(a) = \frac{1}{\sigma_a^2} a \exp\left(-\frac{a^2}{2\sigma_a^2} \right); a \geq 0.$$

It is an accurate representation of the statistical fluctuations of the amplitude modulations of many radar, sonar, ultrasonic and seismic signals for example. If the signal s is taken to represent amplitude modulations then the functions f and n are necessarily real and non-negative. Hence, in this case we can utilize an inverse weighted PDF of the form

$$P(\mathbf{s} \mid \mathbf{f}) = \frac{1}{\sigma_n^2} \int_{-T}^{T} [s(t) - f(t)] dt \exp\left(-\frac{1}{2\sigma_n^2} \int_{-T}^{T} [s(t) - f(t)]^2 \frac{1}{w(t)} t \right).$$

The ML estimate is then given by finding the coefficients A_n which satisfy the equation

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{s} \mid \mathbf{f}) = -\frac{W(\omega_m)}{N(0)} + \frac{1}{\sigma_n^2} \left(S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n) \right) = 0$$

where $N(0)$ is the DC level of the noise, i.e.

$$N(0) = \int_{-T}^{T} [s(t) - f(t)]dt = \int_{-T}^{T} n(t)dt.$$

From this equation we get (using the convolution theorem)

$$-\frac{\sigma_n^2}{N(0)} w_{BL}(t) + s_{BL}(t) = a(t) w_{BL}(t)$$

and hence, the ML estimate for a Rayleigh distribution becomes

$$f_{ML}(t) = w(t) \left(\frac{s_{BL}(t)}{w_{BL}(t)} - \frac{\sigma_n^2}{N(0)} \right).$$

Observe, that when $\sigma_n = 0$, the ML estimate for Rayleigh statistics reduces to the ML estimate for Gaussian statistics.

15.3 The MAP estimate is obtained by finding the coefficients A_m which satisfies the equation

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{s} \mid \mathbf{f}) + \frac{\partial}{\partial A_m} \ln P(\mathbf{f}) = 0$$

where

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t).$$

Using the distributions given,

$$-\frac{1}{2\sigma_n^2} \frac{\partial}{\partial A_m} \int_{-T}^{T} \mid s(t) - w(t) \sum_n A_n \exp(i\omega_n t) \mid^2 \frac{1}{w(t)} dt$$

$$-\frac{1}{2\sigma_f^2} \frac{\partial}{\partial A_m} \int_{-T}^{T} \mid w(t) \sum_n A_n \exp(i\omega_n t) - \bar{f}(t) \mid^2 \frac{1}{w(t)} dt = 0.$$

Differentiating, we get

$$\frac{1}{\sigma_n^2} \int_{-T}^{T} [s(t) - w(t) \sum_n A_n \exp(i\omega_n t)] \exp(-i\omega_m t) dt$$

$$-\frac{1}{\sigma_f^2} \int_{-T}^{T} [w(t) \sum_n A_n \exp(i\omega_n t) - \bar{f}(t)] \exp(-i\omega_m t) dt = 0$$

or

$$\frac{1}{\sigma_n^2} S(\omega_m) - \frac{1}{\sigma_n^2} \sum_n A_n W(\omega_m - \omega_n) - \frac{1}{\sigma_f^2} \sum_n A_n W(\omega_m - \omega_n) + \frac{1}{\sigma_f^2} \bar{F}(\omega_m) = 0$$

where

$$\bar{F}(\omega_m) = \int_{-T}^{T} \bar{f}(t) \exp(-i\omega_m t) dt.$$

Using the convolution theorem, we then obtain

$$\left(\frac{1}{\sigma_n^2} + \frac{1}{\sigma_f^2} \right) a(t) w_{BL}(t) = \frac{1}{\sigma_n^2} s_{BL}(t) + \frac{1}{\sigma_f^2} \bar{f}(t).$$

Rearranging, the MAP estimate for Gaussian statistics is given by

$$f_{MAP}(t) = \frac{1}{1+\Gamma^2} \frac{w(t)}{w_{BL}(t)} \left(\bar{f}(t) + \Gamma^2 s_{BL}(t) \right)$$

where Γ is the signal to noise ratio defined by

$$\Gamma = \frac{\sigma_f}{\sigma_n}.$$

15.4 In this case

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{f}) = \frac{W(\omega_m)}{F(0)} - \frac{1}{\sigma_f^2} \sum_n A_n W(\omega_m - \omega_n)$$

where $F(0)$ is the DC level of $F(\omega_m)$,

$$F(0) = \int_{-T}^{T} f(t) dt$$

and

$$\frac{\partial}{\partial A_m} \ln P(\mathbf{s} \mid \mathbf{f}) = -\frac{W(\omega_m)}{N(0)} + \frac{1}{\sigma_n^2} \left(S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n) \right)$$

where $N(0)$ is the DC level of $N(\omega_m)$,

$$N(0) = \int_{-T}^{T} n(t) dt.$$

It is then easy to show, that the MAP estimate for Rayleigh statistics is given by

$$f_{MAP}(t) = \frac{1}{1+\Gamma^2} \frac{w(t)}{w_{BL}(t)} \left[\Gamma^2 s_{BL}(t) + \sigma_f^2 w_{BL}(t) \left(\frac{1}{F(0)} - \frac{1}{N(0)} \right) \right].$$

15.5 Maximum entropy estimation is usually based on modelling the object itself. A reconstruction is found for f such that

$$E = -\int_{-T}^{T} f(t) \ln f(t) dt$$

is a maximum. If we use a model for f of the form

$$f(t) = w(t) \sum_n A_n \exp(i\omega_n t),$$

then the problem is reduced to finding the coefficients A_n which maximizes E. This requires that f is both real and non-negative. Another way in which the object can be reconstructed is by choosing the coefficients A_n is such a way that the entropy of the amplitude spectrum $\mid A_n \mid$ is a maximum. The entropy of this spectrum is given by

$$E = -\sum_n \mid A_n \mid \ln \mid A_n \mid .$$

Now, because $s = f + n$, we can write

$$\lambda \left[\int_{-T}^{T} \mid s(t) - f(t) \mid^2 \frac{1}{w(t)} dt - \int_{-T}^{T} \mid n(t) \mid^2 \frac{1}{w(t)} dt \right] = 0$$

where λ is the Lagrange multiplier. Thus, we may write the entropy of $\mid A_n \mid$ as

$$E = -\sum_n \mid A_n \mid \ln \mid A_n \mid -\lambda \left[\int_{-T}^{T} \frac{\mid s(t) - f(t) \mid^2}{w(t)} dt - \int_{-T}^{T} \frac{\mid n(t) \mid^2}{w(t)} dt \right).$$

This entropy is a function of the coefficients A_m that we want to compute and is therefore a maximum when

$$\frac{\partial E}{\partial A_m} = 0.$$

Substituting our model for $f(t)$ into the equation for E and differentiating with respect to A_m, we get

$$(1 + \ln \mid A_m \mid) \frac{A_m^*}{2 \mid A_m \mid}$$

$$-2\lambda \left(\int_{-T}^{T} [s(t) - w(t) \sum_n A_n \exp(i\omega_n t)] \exp(-i\omega_m t) dt \right) = 0.$$

Noting that A_m can be written as

$$A_m = \mid A_m \mid \exp(i\phi_m)$$

where $\mid A_m \mid$ and ϕ_m are the amplitude and phase spectra of A_m respectively, we have

$$(1 + \ln \mid A_m \mid) \exp(i\phi_m) = -2\lambda \left(S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n) \right)$$

or

$$\mid A_m \mid = \exp \left[-1 + 2\lambda \exp(-i\phi_m) \left(S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n) \right) \right].$$

This equation is transcendental in A_m and as such requires that A_m is evaluated iteratively, i.e.

$$\mid A_m^{k+1} \mid = \exp\left[-1 + 2\lambda \exp(-i\phi_m^k)\left(S(\omega_m) - \sum_n A_n^k W(\omega_m - \omega_n)\right)\right]$$

where

$$\phi_m^k = \mathrm{Im}[\ln A_m^k].$$

A useful approximation to this maximum entropy estimate can be obtained by linearizing the above equation. This is obtained by expanding the exponential function and retaining the linear terms giving

$$\mid A_m \mid \simeq 2\lambda \exp(-i\phi_m)\left(S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n)\right)$$

or

$$\frac{A_m}{2\lambda} \simeq S(\omega_m) - \sum_n A_n W(\omega_m - \omega_n).$$

From the last equation, it follows that (taking the inverse Fourier transform and using the convolution theorem)

$$a(t) \simeq \frac{s_{BL}(t)}{w_{BL}(t) + 1/2\lambda}$$

and hence

$$f(t) = \frac{w(t) s_{BL}(t)}{w_{BL}(t) + 1/2\lambda}.$$

15.6 For the model given, the error function required is

$$e = \int_{-T}^{T} \left| \frac{s(t)}{w(t)} - \ln\left(\frac{f(t)}{w(t)}\right)\right|^2 w(t)dt = \int_{-T}^{T} \left| \frac{s(t)}{w(t)} - \sum_n A_n \exp(i\omega_n t)\right|^2 w(t)dt$$

where $w > 0$ and s and f are non-negative functions. Differentiating with respect to A_m, this error is a minimum when

$$S_n - A_n \otimes W_n = 0$$

or in real space, when

$$a(t) w_{\mathrm{BL}}(t) = s_{\mathrm{BL}}.$$

Thus, we obtain the solution

$$f(t) = w(t) \exp\left[\frac{s_{\mathrm{BL}}(t)}{w_{\mathrm{BL}}(t)}\right].$$

A.4.4 Solutions to Problems Given in Chapter 16

16.1

```
void FIRCON( float f[], float p[], float s[], int n, int w, int opt)
{
  int edge, mm, m, nn, i, j;
  float *fw, sum;

/*Allocate space for work arrays.*/
  edge = w * 0.5;
  m = n + 2 * edge;
  fw = (float *) calloc( m+1, sizeof( float ) );

/*Determine mid point of arrays.*/
  nn = 1 + (n * 0.5);
  mm = 1 + (m * 0.5);

/*Initialize working array for the signal.*/
  for(i=1; i<=m; i++) fw[i]=0.0;
  for(i=1; i<=n; i++) fw[mm-nn+i]=f[i]

/*Apply end point extension if indicated.*/
  if (opt != 0)
    {
    for ( i=1; i<=edge; i++ )
      {
      fw[i]     = f[1];
      fw[m-i+1] = f[n];
      }
    }

/*Apply convolution to work array.*/
  for ( i=1; i<=n; i++ )
    {
    sum = 0.0;

    for ( j=0; j<w; j++ )
      {
      sum += fw[i+j] * p[w-j];
      }

    s[i] = sum;
    }

/*Free internal work space.*/
```

```
  free( fw );
}

/**********************TEST UNIT******************/

#define n 128

void main(void)
{

float f[n+1], s[n+1], p[3];


       TOPHAT(f,n,20);
       gopen();
       gsignal(f,n,1);

       p[1]=1;
       p[2]=-1;

       FIRCON(f,p,s,n,2,1);
       gsignal(s,n,1);
exit(0);
}
```

16.2

```
void FIRCOR( float f[], float p[], float s[], int n, int w, int opt )
{
  int edge, mm, m, nn, i, j;
  float *fw, sum;

/*Allocate internal memory.*/
  edge = w * 0.5;
  m = n + 2 * edge;
  fw = (float *) calloc( m+1, sizeof( float ) );

/*Determine mid point of arrays.*/
  nn = 1 + (n * 0.5);
  mm = 1 + (m * 0.5);

/*Initialize internal array.*/
  for(i=1; i<=m; i++) fw[i]=0.0;
  for(i=1; i<=n; i++) fw[mm-nn+i]=f[i]
```

```
/*Apply end point extension if indicated.*/
  if (opt != 0)
    {
    for ( i=1; i<=edge; i++ )
      {
      fw[i]     = f[1];
      fw[m-i+1] = f[n];
      }
    }

/*Apply correlation process.*/
  for ( i=1; i<=n; i++ )
    {
    sum = 0.0;

    for ( j=0; j<w; j++ )
      {
      sum += fw[i+j] * p[j+1];
      }

    s[i] = sum;
    }

/*Free memory from stack.*/
  free( fw );
}

/**********************TEST UNIT*******************/

#define n 32

void main(void)
{

float f[n+1], p[n+1], s[n+1];


       TOPHAT(f,n,20);
       TOPHAT(p,n,20);
       gopen();
       gsignal(f,n,1);

       FIRCON(f,p,s,n,n,1);
       gsignal(s,n,1);
exit(0);
}
```

16.3

```
void MAVFIL( float s[], int n, int w, int opt )
{
  int edge, mm, m, nn, i;
  float *sw, sum, scale;

/*Allocate parameters and internal work space.*/
  edge = w * 0.5;
  m = n + 2 * edge;
  *sw = (float *) calloc( m+1, sizeof( float ) );

/*Determine mid point of arrays.*/
  nn = 1 + (n * 0.5);
  mm = 1 + (m * 0.5);
  scale = 1.0 / w;

/*Initialize internal working array.*/
  for(i=1; i<=m; i++) fw[i]=0.0;
  for(i=1; i<=n; i++) fw[mm-nn+i]=f[i]

/*Apply end point extension if indicated.*/
  if (opt != 0)
    {
    for ( i=1; i<=edge; i++ )
      {
      sw[i]     = s[1];
      sw[m-i+1] = s[n];
      }
    }

/*Compute the first average value.*/
  sum = 0.0;
  for ( i=1; i<=w; i++ )
    {
    sum = sum + sw[i];
    }
  s[1] = sum * scale;

/*Calculate the remaining average values.*/
  for ( i=2; i<=n; i++ )
    {
    sum  = sum - sw[i-1] + sw[i+w-1];
    s[i] = sum * scale;
    }

/*Free space from work array.*/
```

```
  free( sw );
}
```

16.4

```
    void MEDIAN(float x[], int w, int xmed) /*Prototyping*/

    void MEDFIL(float s[], int n, int w)
    {
    int i, k, m, frame;
    float *x,*y;
    float med;

/*Declare internal workspace.*/
    x = (float *) calloc(n+1, sizeof(float));
    y = (float *) calloc(n+1, sizeof(float));

/*Zero pad s with an array of zeros (w-1)/2  samples wide.
/*The zero padded signal is stored in array x.*/

        m=n+w-1;
        for (i=1; i<=m; i++) x[i]=0.0;

        frame=(w-1)/2;
        for(i=1; i<=n; i++)
        x[i+frame]=s[i];

/*Start the process.*/
        for(i=1; i<=n; i++)
        {
/*Select the size of the neighbourhood of samples
         and store the result in the array y.*/
         for(k=1; k<=w; k++) y[k]=x[i-1+k];

/*Compute the median value of y.*/
        MEDIAN(y,w,med);

/*Store the median value in array s - output.*/
        s[i]=med;

/*Repeat the process, i.e. move on to the next point.*/
        }
}

/*****************INTERNAL FUNCTION***************/
```

```
        void SORT(float x[], int n) /*Prototyping*/

        void MEDIAN(float x[], int n, int xmed)
        {
        int nn, mid;

/*Computes the median of a set of n numbers
      where n is an odd number; 3, 5, 7,...*/

/*Sort numbers in order of increasing value.*/

        SORT(x,n);

/*Compute median xmed  -  where half the numbers
  in x[i] > xmed and half the numbers in x[i] < xmed.
  The value of xmed is just the mid value of array x(i).*/

        nn=(n-1)/2;
        mid=nn+1;    /*Mid point of array.*/
        xmed=x[mid]; /*Median value.*/
        }

/*******************INTERNAL FUNCTION***************/

        void SORT(float x[], int n)
        {
        int i,j;
        float temp;

/*Sorts n numbers in order of increasing value using a
 'bubble sort' with dynamic looping.*/

/* Start process. */

        for(i=1; i<=n-1; i++)
           {
            for(j=1; j<=n-i; j++)/* Dynamic upper limit on nested loop. */
               {
/*Check if x[j] is greater than x[j+1]. If so,
  then exchange positions: x[j] becomes x[j+1]
                           x[j+1] becomes x[j]. */

        if(x[j] > x[j+1])
           {
           temp=x[j];
            x[j]=x[j+1];
             x[j+1]=temp;
```

```
            }

/* Repeat process. */
               }
            }
         }
```

16.5 The moving average filter is useful for removing high frequency noise of relative low amplitude, but very poor when it comes to dealing with high amplitude noise spikes. In the latter case, the median filter replaces a large amplitude spike which is out of context with its nearest neighbours with the median value of these neighbours rather than the average value which is dominated by the 'out-of-context' neighbour. Thus, for noise spikes, the median filter is ideal.

A.4.5 Solutions to Problems given in Chapter 17

17.1

```
#include <math.h>
/* Prototype functions */
void FFT1D(float a[], float b[], int n, int opt)
void UNOISE(float s[], int n, int seed)
void GNOISE(float s[], int n, int seed)


void RFS(float *s, int n, float D)
{
   int i, k, m, opt;
   float *si, *amp, *phase, q, pi;

/* FUNCTION: Generates a fractal signal s, of size n, with fractal
             dimension D, using the Fourier synthesis method.

 AUTHOR: J M Blackledge

 PARAMETERS
    Input:  n - signal size.
            D - fractal dimension of signal.
    Output: s - fractal signal.

 INTERNAL VARIABLES
    i - array counter.
    k - absolute frequency.
    m - midpoint of signal.
    opt - inverse FFT option.
    si - imaginary part of signal.
```

```
    amp - amplitude of signal in Fourier space.
    phase - phase of signal in Fourier space.
    q - fractal parameter of signal.
    pi - value of pi.

 EXTERNAL FUNCTIONS:
    GNOISE - generates noise signal with Gaussian distribution.
    NOISE - generates noise signal with uniform distribution.
    FFT1D - performs FFT in 1D.

 INTERNAL FUNCTIONS:
    None */


/* Calculate midpoint, q and pi values.               */
   m=n/2+1;
   q=5.0-2.0*D;
   pi=4.*atan(1.);

/* Allocate memory for arrays.                        */
   si = (float *) calloc( n+1, sizeof(float) );
   amp = (float *) calloc( n+1, sizeof(float) );
   phase = (float *) calloc( n+1, sizeof(float) );

/* Generate random values for amplitude
   and phase of the fractal.                          */
   GNOISE(amp,n,123456);
   UNOISE(phase,n,654321);

/* Calculate real and imaginary parts of signal
   in Fourier space.                                  */
   for (i=1;i<=n;i++) {

/* Absolute value of frequency at i.                  */
      k=i-m;

/* Perform lowpass filter on the signal.              */
      if (k!=0)
         amp[i]/=pow(abs(k),q/2.0);

/* Calculate phase of the signal.                     */
      phase[i]*=2.0*pi;

/* Calculate real and imaginary parts of the signal.  */
      s[i]=amp[i]*cos(phase[i]);
      si[i]=amp[i]*sin(phase[i]);
   }
```

```
/* Perform inverse FFT to get signal in real space.    */
   opt=1;
   FFT1D(s,si,n,opt);

/* Free memory.                                        */
   free(si);
   free(amp);
   free(phase);
}
```

17.2

```
#include <math.h>
#include <stdlib.h>
/* Prototype functions */
void POWSPEC(float s[], float p[], int n)


float FD(float *s,int n)
{
   int i, k, m, nk;
   float *p, lnp, lnk, lnplnk, lnk2, q;

/* FUNCTION: Calculates the fractal dimension of signal s
             using the spectral power method.

 AUTHOR: J M Blackledge

 PARAMETERS
    Input:  s - signal.
            n - signal size.
    Output: FD - fractal dimension of signal s.

 INTERNAL VARIABLES
    i - counter for signal.
    k - absolute frequency.
    m - midpoint of signal.
    nk - number of frequencies used in calculation.
    p - power spectrum of signal s.
    lnp, lnk, lnplnk, lnk2 - sum of logs of the values k and p.
    q - estimate of the q value for signal s.

 EXTERNAL FUNCTIONS:
    POWSPEC - calculates the power spectrum of the signal.
```

```
 INTERNAL FUNCTIONS:
    None                                                */

/* Allocate memory for power spectrum.                  */
   p = (float *) calloc( n+1, sizeof(float) );

/* Calculate midpoint of signal.                        */
   m=n/2+1;

/* Calculate power spectrum.                            */
   POWSPEC(s,p,n);

/* Sum the values of ln(p[i]), ln(p[i]).ln(k), ln(k)
   and (ln(k))^2.                                       */
   lnp=0.0;
   lnplnk=0.0;
   lnk=0.0;
   lnk2=0.0;
   nk=n;
   for (i=1;i<=n;i++) {
      k=abs(i-m);
      if ((p[i]!=0.0) && (k!=0)) {
         lnp+=log(p[i]);
         lnplnk+=log(p[i])*log(k);
         lnk+=log(k);
         lnk2+=log(k)*log(k);
      } else {
         nk--;
      }
   }

/* Calculate value of q.                                */
   q=(nk*lnplnk-lnk*lnp)/(pow(lnk,2)-nk*lnk2);

/* Free memory.                                         */
   free(p);

/* Return fractal dimension value.                      */
   return ((5.0-q)/2.0);
}
```

17.3[1] For $0 < q < 1$ and since the characteristic function is symmetric, we have

$$p(x) = \mathrm{Re}[f(x)]$$

[1]Grateful thanks to Dr Keith Hopcraft, Nottingham University, for suggesting this approach.

where

$$f(x) = \frac{1}{\pi}\int_0^\infty e^{ikx}e^{-k^q}dk = \frac{1}{\pi}\left(\left[\frac{1}{ix}e^{ikx}e^{-k^q}\right]_{k=0}^\infty - \frac{1}{ix}\int_0^\infty e^{ikx}(-qk^{q-1}e^{-k^q})dk\right)$$

$$= \frac{q}{i\pi x}\int_0^\infty dkk^{q-1}e^{-k^q}e^{ikx}dk = \frac{q}{i\pi x}\int_0^\infty e^{ikx}k^{q-1}\phi(k)dk$$

where

$$\phi(k) = \exp(-k^q).$$

For $0 < q < 1$, $f(x)$ is singular at $k = 0$ and is once differentiable. Using the result given, for $0 < q < 1$,

$$A_N = \Gamma(q)e^{-i\pi(1-q/2)}e^{-k^q} \mid_{k=0} \frac{e^{i0}}{x^q} = -\frac{\Gamma(q)e^{i\pi q/2}}{x^q}$$

$$\therefore \;\; f(x) = -\frac{q}{i\pi x}\left(-\frac{\Gamma(q)e^{i\pi q/2}}{x^q}\right)$$

and

$$p(x) = \frac{2\Gamma(1+q)\sin(\pi q/2)}{\pi x^{1+q}}, \quad \mid x \mid >> 0.$$

For $1 < q < 2$, we can integrate by parts twice to obtain

$$f(x) = \frac{q}{i\pi x}\int_0^\infty dkk^{q-1}e^{-k^q}e^{ikx}$$

$$= \frac{q}{i\pi x}\left[\frac{1}{ix}k^{q-1}e^{-k^q}e^{ikx}\right]_{k=0}^\infty + \frac{q}{\pi x^2}\int_0^\infty dke^{ikx}[(q-1)k^{q-2}e^{-k^q} - q(k^{q-1})^2e^{-k^q}].$$

The first term of this result is singular and therefore provides the greatest contribution and thus we can write,

$$f(x) = \frac{q(q-1)}{\pi x^2}\int_0^\infty e^{ikx}(u^{q-2}e^{-k^q})dk.$$

Using the result given, writing $q = \lambda - 1$ and noting that for $1 < q < 2$, ϕ is twice differentiable, we get

$$f(x) = -\frac{q(q-1)}{\pi x^2}[\Gamma(q-1)e^{i\pi(q-3)/2}e^{-k^q}x^{-(\lambda-1)}e^{ix0}$$

$$+\Gamma(\lambda)e^{i(1+q-3)\pi/2}(e^{-k^q})' \mid_{k=0} e^{ix0}].$$

Thus,

$$p(x) = -\frac{\Gamma(1+q)\cos[\pi(q-3)/2]}{\pi x^2 x^{q-1}} = -\frac{\Gamma(1+q)cos[\pi(q-3)/2]}{\pi x^{1+q}}, \quad \mid x \mid >> 1$$

which maps smoothly onto the previous asymptotic as $q \to 1$ from above.

17.4 For $\omega > 0$, the function $P(\omega)$ has a maximum when

$$\frac{d}{d\omega} \ln P(\omega) = \frac{2g}{\omega} - \frac{2\omega q}{\omega_0^2 + \omega^2} = 0.$$

Since $P(\omega) \neq 0 \forall \omega$, this implies that a maximum value of $P(\omega)$ occurs at a value of $\omega = \omega_{\max}$ given by

$$\omega_{\max} = \omega_0 \sqrt{\frac{g}{q-g}}, \quad q > g.$$

The value of $P(\omega)$ at this point is therefore given by

$$P_{\max} \equiv P(\omega_{\max}) = \frac{\omega_{\max}^{2g}}{(\omega_0^2 + \omega_{\max}^2)^q} = \omega_0^{2(g-q)} \frac{g^g}{q^q} (q-g)^{q-g}.$$

For a time invariant linear system and with n denoting a white noise input, we have

$$S(\omega) = \frac{(i\omega)^g}{(i\omega - \omega_0)^q} N(\omega).$$

Inverse Fourier transforming,

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} (i\omega - \omega_0)^q S(\omega) \exp(i\omega t) d\omega = \frac{1}{2\pi} \int_{-\infty}^{\infty} (i\omega)^g N(\omega) \exp(i\omega t) d\omega = \frac{d^g}{dt^g} n(t).$$

The integral on the left hand side can be written as

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} (iu)^q S(u) \exp(\omega_0 t) \exp(iut) du = \exp(\omega_0 t) \frac{d^q}{dt^q} s(t).$$

Thus,

$$\frac{d^q}{dt^q} s(t) = \exp(-\omega_0 t) \frac{d^g}{dt^g} n(t)$$

and on inversion,

$$s(t) = \frac{1}{\Gamma(q)} \int_0^t \frac{\exp(-\omega_0 \tau)}{(t-\tau)^{1-q}} \frac{d^g}{d\tau^g} n(\tau) d\tau.$$

To investigate the scaling law, consider the integral

$$\frac{1}{\Gamma(q)} \int_0^t \frac{\exp(-\omega_0 \tau)}{(t-\tau)^{1-q}} \frac{d^g}{d\tau^g} n(\lambda\tau) d\tau = \frac{\lambda^g}{\lambda^q} \frac{1}{\Gamma(q)} \int_0^t \frac{\exp(-\omega_0 \xi/\lambda)}{(\lambda t - \xi)^{1-q}} \frac{d^g}{d\xi^g} n(\xi) d\xi.$$

Hence,

$$\Pr[s(t, \omega_0)] = \frac{\lambda^g}{\lambda^q} \Pr[s(\lambda t, \omega_0/\lambda)].$$

Here, as we scale t by λ, the characteristic frequency ω_0 is scaled by $1/\lambda$; this is a result that is consistent with the scaling properties of the Fourier transform, i.e.

$$f(\lambda t) \Longleftrightarrow \frac{1}{\lambda} F\left(\frac{\omega}{\lambda}\right)$$

Thus, as we zoom into the signal $s(t)$ by decreasing λ, the distribution of amplitudes - the PDF of the signal - remains the same (subject to scaling by λ^g/λ^q) and the characteristic frequency of the signal increases by a factor of $1/\lambda$.

A.4.6 Supplementary Problems to Part IV

IV.1 (a) Given that a signal $f(t)$ with period $2T$ can be expressed as a complex Fourier series of the form

$$f(t) = \sum_{n=-\infty}^{\infty} F_n \exp(-in\pi t/T)$$

show that the coefficients F_n are given by

$$F_n = \frac{1}{2T} \int_{-T}^{T} f(t) \exp(-in\pi t/T) dx$$

Use this result to obtain a generating formula for the discrete Fourier coefficients F_n in the case when the signal is a tophat function of width $2T$ centered at $t = 0$.

(b) Defining the Fourier transform of a signal $f(t)$ as

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt$$

find $F(\omega)$ when

$$f(t) = \exp(-\lambda \mid t \mid)$$

where λ is a real constant. Using your result together with the convolution theorem, show that a solution to the equation

$$y(t) = x(t) + \lambda \int_{-\infty}^{\infty} \exp(- \mid t - \tau \mid) y(\tau) d\tau$$

is

$$y(t) = x(t) + \psi(\lambda) \int_{-\infty}^{\infty} \exp(-\sqrt{1 - 2\lambda} \mid t - \tau \mid) x(\tau) d\tau$$

stating both the functional form of ψ and the condition on λ for which this solution is valid.

IV.2 (a) Find expressions for the N-length DFT of each of the following digital signals:

(i) the N-component unit sample (impulse) signal (the Kronecker delta function)

$$\delta_n = \begin{cases} 1, & n = 0; \\ 0, & n \neq 0; \end{cases}$$

(ii) the N-component signal defined as follows

$$f_n = \begin{cases} 1, & n = m; \\ 0, & \text{otherwise} \end{cases}$$

where $0 \leq m \leq N-1$;

(iii) the N-component unit ramp signal given by

$$f_n = \begin{cases} n, & n \geq 0; o, \\ n < 0. \end{cases}$$

List all the DFT values of each signal for $N = 4$.

(b) Compute the DFT of the signal $f_i = (1, 1, 0, 0)$ and verify Parseval's theorem.

(c) Prove the following properties of the DFT:

(i) the periodicity property for the DFT F_m and the IDFT f_n;

(ii) the linearity property.

(d) Describe the use of the DFT and the inverse DFT in obtaining the linear discrete convolution for FIR systems, i.e. when a finite duration sequence f_n of length N is input to system with a with a FIR function g_n of length M.

IV.3 (a) Consider the discrete convolution equation

$$s_i = \sum_j p_{j-i} f_n.$$

Write this equation in matrix form

$$\mathbf{s} = P\mathbf{f}$$

computing the elements of P in the case when

$$\mathbf{f} = (0, 0, 0, 10, 0, 0, 10, 0, 0, 0)^T$$

and

$$\mathbf{p} = (10, 11, 12, 11, 10)^T.$$

Consider the case when additive noise is present. Compute the result of applying a 3×1 median filter to the signal s_i when the noise array is $(1, 3, 2, 3, 1, 5, 2, 3, 1, 1)^T$. What is the effect of applying this process?

(b) By minimizing the quantity $\|n_i\|_2^2$ with respect to f_i, and employing the discrete convolution and correlation theorems, show that the inverse filter is given by

$$\frac{P_i^*}{|\ P_i\ |^2}$$

where P_i is the discrete Fourier transform of p_i and P_i^* is the complex conjugate of P_i. Discuss the problems that can occur with the application of this type of digital filter.

IV.4 An experiment is carried out where M samples of a signal s_i are measured in the presence of additive Gaussian noise. The signal is known to have a constant amplitude f over the duration of the experiment. The Probability Distribution Function (PDF) of the noise is Gaussian with a standard deviation of σ_n and the PDF of the signal itself is Gaussian with a standard deviation of σ_f.

Starting from Bayes rule, show that:

(i) the Maximum Likelihood (ML) estimate for f is given by

$$\hat{f}_{ML} = \frac{1}{M}\sum_{i=1}^{M} s_i.$$

(ii) Derive an expression for the Maximum *a Posteriori* estimate $\hat{f}_{MAP}$ and hence, show that

$$\hat{f}_{MAP} \simeq \hat{f}_{ML};\quad \frac{\sigma_n}{\sigma_f} << 1.$$

IV.5 (a) Given that

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)dt$$

and

$$\int_{-\infty}^{\infty} \frac{1}{t}\exp(-i\omega t)dt = -i\pi\mathrm{sgn}(\omega)$$

show that the 'Analytic Signal' $s(t)$ defined as

$$s(t) = \frac{1}{\pi}\int_{0}^{\infty} F(\omega)\exp(i\omega t)d\omega$$

can be written in the form

$$s(t) = f(t) + iq(t)$$

where $q(t)$ is the Hilbert transform of $f(t)$ given by

$$q(t) = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{f(\tau)}{t-\tau}d\tau$$

By writing $\sin(\omega_0 t)$ and $\cos(\omega_0 t)$ in terms of complex exponentials, compute the Fourier transforms of these functions. Hence, or otherwise, find the Hilbert transform of $\cos(\omega_0 t)$.

(b) The module

```
FFT(c,n,sign)
```

is a Fast Fourier Transform (void) function where: c is the complex I/O array, n is the array size, sign=-1 gives the forward transform; sign=1 gives the inverse transform. This subroutine computes a complex spectrum where the DC level is placed at n/2 + 1. Using this FFT and including comment lines to explain the code, write a C void function

```
HILBERT(s,n)
```

to compute the discrete Hilbert transform of a given input where s is the real I/O array and n is the array size.

IV.6 The complex Fourier series representation of a piece-wise continuous function $f(t)$ with period T is given by

$$f(t) = \sum_{n=-\infty}^{\infty} a_n \exp(i2\pi nt/T)$$

where

$$a_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) \exp(-i2\pi nt/T) dt$$

(i) Derive a complex Fourier series representation of the delta sequence

$$\sum_{n=-\infty}^{\infty} \delta(t - nT)$$

Hence or otherwise, show that the Fourier transform of this sequence is given by

$$\frac{2\pi}{T} \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n/T)$$

where ω is the angular frequency.

Use the results above to show that the digitization of a bandlimited analogue signal leads to a replication of its spectrum and in this context, explain the meaning of the term 'Aliasing'.

(ii) An ultrasonic signal with a bandwidth of 1MHz is recorded in analogue form. What is its Nyquist frequency and what is the maximum sampling interval at which it should be sampled to avoid Aliasing?

(iii) Explain the basis for 'sinc interpolation' and discuss how it can be implemented in practice using a Fast Fourier Transform.

The void function

```
FFT1D(a,b,n,sign)
```

is a Fast Fourier Transform where a is the real I/O array, b is the imaginary I/O array n is the array size, sign=-1 gives the forward transform; and sign=1 gives the inverse transform

This subroutine computes a complex spectrum (with real and imaginary parts given by a and b respectively) where the DC level is placed at n/2 + 1.

Using this FFT and including comment lines to explain the code, write a C void function

```
SINC_INTERP(x,n,y,m)
```

to sinc interpolate a signal from n data samples to m data samples where m>n and both m and n are integer powers of 2 where x and y are the input and output arrays respectively and are of type float.

IV.7 (i) Compute the result of applying the FIR kernel $(0, 1, 2, 1, 0)^T$ to the digital signal

$$(0, 0, 1, 0, 0, 2, 0, 0, 3, 0, 2, 0, 1, 0, 1, 0, 0)^T$$

using zero padding.

Explain the computational differences between applying an FIR filter directly and using an FFT commenting in each case on:

- numerical efficiency
- accuracy

(ii) The digital signal

$$(1, 0, -1, 0)^T$$

is recorded by a system whose impulse response function is known to be $(1, 2, 1)^T$.

Use zero padding to construct an appropriate matrix equation which relates this digital signal to the input function of the system. Use the result to deconvolve the signal using an appropriate algorithm with exact arithmetic.

Write a C void function to implement your method of deconvolution for an arbitrary 3×1 impulse response function and an arbitrary digital signal consisting of n samples:

```
DECON(s,p,f,n)
```

where s is the input signal (real input of type float), p is the impulse response function (real input of type float), f is the deconvolved signal (real output of type float) and n is the array size (real input of type int).

IV.8 (a) Explain the term 'Gibbs' phenomenon and discuss situations when it is most likely to occur in the processing of digital signals. What techniques can be employed to remove this effect? - illustrate your answer by considering the application of a lowpass filter.

(b)

(i) Write down the Discrete Fourier Transform of a (complex) array f_n of size N and the associated inverse transform of a (complex) array F_n of the same size.

(ii) Compute the real and imaginary parts of the DFT of the array $(0, 1, 1, 0)$. If the sampling interval between the elements of this array is 1 second what is the frequency interval between the elements of its DFT ?

(c)

(i) Assume a FFT void function is available called

```
FFT(a,b,n,sign)
```

where a and b and real and imaginary I/O array respectively, n is the array size and sign=$\pm$1 for computing the forward (-1) and inverse (+1) transforms respectively. Write a void function in C to compute the power spectrum of a signal using a switch to provide the user with the option to output the result using a logarithmic scale:

```
POWSPEC(s,p,n,opt)
```

where s is the input, p is the output, n is the array size, opt=0 outputs the power spectrum on a linear scale and opt=1 output the power spectrum using a logarithmic scale

IV.9 (a) A digital signal s_i can be described by the stationary process

$$s_i = \sum_j p_{i-j} f_j + n_i$$

where p_i is the impulse response function, f_i is the object function and n_i is the noise function. Assuming that the noise is signal independent show that if we construct an estimate for f_i of the form

$$\hat{f}_i = \sum_j q_j s_{i-j}$$

then the DFT of q_i (the Wiener Filter) which minimizes the error

$$e = \|f_i - \hat{f}_i\|_2^2$$

is given by

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \frac{|N_i|^2}{|F_i|^2}}$$

where P_i, F_i and N_i are the DFT's of p_i, f_i and n_i respectively.

(b) Explain some of the practical difficulties associated with the application of this filter for digital signal processing and discuss techniques through which estimates of the signal-to-noise ratio can be estimated.

(c) The Wiener filter can be approximated by

$$Q_i = \frac{P_i^*}{\mid P_i \mid^2 + \Gamma}$$

where

$$\Gamma = \frac{1}{\mathrm{SNR}^2},$$

SNR denoting the signal-to-noise ratio of the input signal. A subprogram is to be written to compute $\hat{f}_i$ for $i = 1, 2, ..., N$ given the signal s_i, the impulse response function p_i and value of SNR.

(iii) Write a function in C to compute $\hat{f}_i$.

```
void WIENER(float s[ ], float p[ ], float f[ ], int n, float snr)
```

where it is assumed that you have an FFT subprogram called fft(a,b,n,sign) is available where a is the real I/O array, b is the imaginary I/O array, n is the array size, sign=-1 gives the forward FFT and sign=1 gives the inverse FFT.

IV.10 (a) Compute the output obtained by applying a FIR filter to the signal

$$(0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)$$

using the kernel $\frac{1}{4}(1, 2, 1)$ and the zero padding method.

(b) Compute the moving average and median filters of the signal given in part (a) using a window of size 3 and zero padding. Comment on the results?

(c)

(i) Write a subprogram in C to compute the median filter of a digital signal using zero padding and a bubble sort for windows of arbitrary (odd) size: subroutine medfil(s,n,size) where s is the I/O (type float), n is the I/O array size (type int) and size is the size of the window (=3,5,7,...) which is of type int.

IV.11 Random scaling fractal signals are stochastic signals which are statistically self-affine. There discrete power spectrum can be modelled as

$$\hat{P}(k_i) = \frac{c}{k_i^{\beta}} \quad \forall k_i > 0; \quad i = 1, 2, ..., N$$

where c is a constant and β is related to the fractal dimension D $(1 < D < 2)$ by

$$\beta = 5 - 2D$$

(a) By considering the error function

$$e(c, \beta) = \| \ln P(k_i) - \ln \hat{P}(k_i) \|_2^2$$

where $P(k_i)$ is the power spectrum of a given signal, show that the values of β and c which for which e is a minimum are given by

$$\beta = \frac{\left(\sum_i \ln k_i\right)\left(\sum_i \ln P_i\right) - N\sum_i (\ln P_i)(\ln k_i)}{N\sum_i (\ln k_i)^2 - \left(\sum_i \ln k_i\right)^2}$$

and

$$c = \exp\left[\frac{1}{N}\left(\sum_i \ln P_i + \beta \sum_i \ln k_i\right)\right]$$

where

$$P_i \equiv P(k_i) \quad \text{and} \quad \sum_i \equiv \sum_{i=1}^{N}.$$

What is the physical significance of the parameter c.

(b) Consider the following table of data

$$k_i : 1 \quad 2 \quad 3 \quad 4$$

$$P_i : 1 \quad \frac{1}{4} \quad \frac{1}{9} \quad \frac{1}{16}$$

What is the fractal dimension of the signal whose power spectrum is P_i.

(c) Working in either K & R or ANSI C, write a void function which computes the fractal dimension of signal given the following I/O:

- Inputs - P_i, k_i and N,
- Output - D.

Appendix B

Graphics Utility

There are a wide range of graphics facilities available for plotting signals. Many of these facilities form part of an integrated windows based package such as MATLAB which can of course be used for this purpose. A detailed discussion of the many graphics facilities currently available is beyond the scope of this book and it is left to the reader to apply the facilities best suited to his/her application and available software systems. However, for completeness, this appendix provides the C code developed to display a signal using the graphics functions available with the Borland Turbo C++ compiler. Version 3.00 (published in 1992 by Borland International) of this compiler was originally utilized to develop a DSP library using a DOS environment by students undertaking an MSc programme in 'Digital Signal Processing' established by the author in the early 1990s and it may still be of value to those with limited access to a more advanced graphics system. This low level graphics facility is based on the function *gsignal* and is utilized as follows:

```
gopen();

...

option=1
gsignal(s,n,option);

...

gclose();
```

Compilation requires a link to *graphics.lib* and execution assumes that the file *egavga.bgi* resides in the current directory. Here, s in an array of length n which is taken to be a stream of floating point numbers. The plot is automatically windowed and scaled but the size of the arrays that can be plotted are limited to 512 elements. There are two options available; option=1 (recommended) 'holds' the plot until the user 'hits' a key where upon the process continues; option=0 continues processing as soon as the plot has been completed. The function can be used in multiples to plot the progress of a process and for diagnostic purposes. Further, standard I/O can be used

while the graphics are 'open', the output being an overlay in standard format. The module assumes that arrays of size n are input which have been processed using the elements s[0], s[1],..., s[n-1] or s[1], s[2],..., s[n] inclusively. In the latter case, which is the basis for the processors discussed in Part IV, the 0^{th} element of the array should be initialized (i.e. set s[0]=0) to prevent spurious values from corrupting the output.

```
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>

#define XPOS0 8
#define XPOS1 3
#define YPOS0 5
#define YPOS1 12

void  clear_text( void );
void  clear_enter( void );
void  press_enter( void );
void  gopen( void );
void  gclose( void );
void  set_gray( void );
void  set_color( void );

extern int win_xmax, win_ymax;

static float step_x_len, step_y_len;
static float ymax, ymin, xmin, xmax;

void  axis( );
void  plot( float y[], int n );
int   calc_y_pixel( float y );

void gsignal( float s[], int n, int iopt )
{
  int i;

/*************************************************************/
/*                                                           */
/* FUNCTION: Displays a digital signal held in array s[] using */
/*           Borland C++ V3 graphics functions.              */
/*                                                           */
/* NOTE: The program automatically scales the axis so that the */
/*       display lies within the bounds of the x and y axis.   */
/*                                                           */
```

```
/*                                                                  */
/* PARAMETERS                                                       */
/*                                                                  */
/* Input: s - digital signal                                        */
/*        n - no. of samples (<=512)                                */
/*                                                                  */
/*        iopt - continuous or static mode                          */
/*                                                                  */
/*        iopt = 0 gives continuous mode.                           */
/*        iopt = 1 holds display until user 'Hits' a key            */
/*                                                                  */
/* Output: None                                                     */
/*                                                                  */
/********************************************************************/

/* Compute largest (ymax) and smallest (ymin) value of y.           */

    ymin = 0.0;
    ymax = 0.0;
    for ( i=0; i<=n; i++ )
      {
      if(s[i] > ymax)
        ymax = s[i];
      if(s[i] < ymin)
        ymin = s[i];
      }


    if (ymax == ymin)
      {
      ymax = 1.0;
      ymin = -1.0;
      }

/* Define the minumum and maximum values of the x-axis.             */

    xmin = 0.0;
    xmax = (float) n-1;

/* Plot axis.                                                       */

    axis( 0.7f, 0.03f );


/* Plot signal.                                                     */

  plot( s, n );
```

```
/* Provide user with display mode options, i.e.
/* continuous or held display.                                    */

  if (iopt == 1)
    press_enter();
}

void axis( )
{
  int x_axis, y_axis, x, y, temp_x;
  float temp_y, i;
  char c[20];

/*****************************************************************/
/*                                                               */
/*  FUNCTION: Plots the axes.                                    */
/*                                                               */
/*****************************************************************/

  step_x_len = (win_xmax-2.0) / (xmax-xmin);
  step_y_len = (win_ymax-2.0) / (ymax-ymin);

  setviewport( 0, 0, win_xmax, win_ymax, 1 );
  clearviewport();

  set_color();

  setcolor( BLUE );
  setlinestyle( SOLID_LINE, 1, NORM_WIDTH );
  rectangle( 0, 0, win_xmax, win_ymax );

  setcolor( DARKGRAY );
  setlinestyle( SOLID_LINE, 1, NORM_WIDTH );

  x_axis = 1 + (xmax * 0.5);
  x_axis = 1 + x_axis * step_x_len;
  y_axis = calc_y_pixel( 0.0 );

  line( x_axis, 1, x_axis, win_ymax-2 );
  line( 1, y_axis, win_xmax-2, y_axis );

  settextstyle( SMALL_FONT, HORIZ_DIR, 1 );

  temp_x = 1 + (xmax * 0.5);
  for ( i=0; i<=temp_x; i+=temp_x * 0.25 )
    {
```

```
    x = (temp_x+i) * step_x_len;
    line( x, y_axis-5, x, y_axis+5 );
    sprintf( c, "%-3.0f", temp_x+i );
    outtextxy( x, y_axis-13, c );
    x = (temp_x-i) * step_x_len;
    line( x, y_axis-5, x, y_axis+5 );
    sprintf( c, "%-3.0f", temp_x-i );
    outtextxy( x, y_axis-13, c );
    }

  temp_y = fabs( ymin );
  if (temp_y < ymax)
    temp_y = ymax;
  for ( i=temp_y * 0.2; i<=temp_y; i+=temp_y * 0.2 )
    {
    y = calc_y_pixel( i );
    line( x_axis-5, y, x_axis+5, y );
    sprintf( c, "%-g", i );
    outtextxy( x_axis+13, y, c );
    y = calc_y_pixel( -i );
    line( x_axis-5, y, x_axis+5, y );
    sprintf( c, "%-g", -i );
    outtextxy( x_axis+13, y, c );
    }
}

void plot( float y[], int n )
{
  int scr_x,scr_y;
  int i;

/*****************************************************************/
/*                                                               */
/*  FUNCTION: Plots the signal.                                  */
/*                                                               */
/*****************************************************************/

  setviewport( 0, 0, win_xmax, win_ymax, 1 );

  setcolor( RED );
  setlinestyle( SOLID_LINE, 1, NORM_WIDTH );

  scr_x = 1;
  scr_y = calc_y_pixel( y[1] );

  moveto( scr_x, scr_y );
```

```
  for ( i=2; i<=n; i++ )
    {
    scr_x = 1 + ((i-1)*step_x_len);
    scr_y = calc_y_pixel( y[i] );

    lineto ( scr_x, scr_y );
    }
}

int calc_y_pixel( float y )
{
  int i;

  i = 1 + (ymax-y) * step_y_len;

  return( i );
}

/*****************************************************/
/*                                                   */
/* FUNCTION: Provides functions for display options. */
/*                                                   */
/*****************************************************/

extern int scr_xmax, scr_ymax;

static int win_xmax, win_ymax;
int cur_x, cur_y;

void clear_text( void )
{
  win_xmax = scr_xmax - 513;
  win_ymax = scr_ymax;

  setviewport( 514, scr_ymax-13, scr_xmax, scr_ymax, 1 );
  clear_enter();

  setviewport( 514, 0, scr_xmax, scr_ymax-13, 1 );
  clearviewport();

  moveto( 0, win_ymax-13 );
  lineto( 0, 0 );
  lineto( win_xmax, 0 );
  lineto( win_xmax, win_ymax-13 );

  setcolor( LIGHTGRAY );
  settextstyle( SMALL_FONT, HORIZ_DIR, 1 );
```

```
  cur_x = XPOS1;
  cur_y = YPOS0;
}

void clear_enter( void )
{
  setcolor( LIGHTBLUE );
  setlinestyle( SOLID_LINE, 1, THICK_WIDTH );

  clearviewport();

  moveto( 0, 0 );
  lineto( 0, 13 );
  lineto( scr_xmax-513, 13 );
  lineto( scr_xmax-513, 0 );
}

void press_enter( void )
{
  setviewport( 514, scr_ymax-13, scr_xmax, scr_ymax, 1 );
  setcolor( LIGHTGRAY );

  outtextxy( XPOS1, 5, "Press Enter" );
  getch();
  clear_enter();

  setviewport( 514, 0, scr_xmax, scr_ymax-13, 1 );
  setcolor( LIGHTGRAY );
}

int get_int( void )
{
  char cx, tempstr[2], nx[10];
  int i, color, bkcolor;

  cur_x = XPOS1;
  cur_y += YPOS1;

  for ( i=0; ((cx=getch())>='0' && cx <= '9') ||
              cx == '-' || cx == 8; i++ )
    {
    if (cx == 8)
      {
      color = getcolor();
      bkcolor = getbkcolor();
      setcolor( bkcolor );
      sprintf( tempstr, "%c", nx[--i] );
```

```
outtextxy( cur_x, cur_y, tempstr );
      nx[i--] = ' ';
      cur_x -= XPOS0;
      setcolor( color );
      }
    else
      {
      nx[i] = cx;
      sprintf( tempstr, "%c", cx );
      outtextxy( cur_x += XPOS0, cur_y, tempstr );
      }
    }
  nx[i] = '\0';

  return( atoi( nx ) );
}

float get_float( void )
{
  char cx, tempstr[2], nx[10];
  int i, color, bkcolor;

  cur_x = XPOS1;
  cur_y += YPOS1;

  for ( i=0; ((cx=getch())>='0' && cx <= '9') ||
              cx == '-' || cx == '.' || cx == 8; i++ )
    {
    if (cx == 8)
      {
      color = getcolor();
      bkcolor = getbkcolor();
      setcolor( bkcolor );
      sprintf( tempstr, "%c", nx[--i] );
outtextxy( cur_x, cur_y, tempstr );
      nx[i--] = ' ';
      cur_x -= XPOS0;
      setcolor( color );
      }
    else
      {
      nx[i] = cx;
      sprintf( tempstr, "%c", cx );
      outtextxy( cur_x += XPOS0, cur_y, tempstr );
      }
    }
  nx[i] = '\0';
```

```
  return( atof( nx ) );
}

char get_char( void )
{
  char cx;

  cx = getch();
  cx = toupper( cx );

  return( cx );
}

void get_string( char string[] )
{
  char cx, tempstr[2];
  int i, color, bkcolor;

  cur_x = XPOS1;
  cur_y += YPOS1;
  for ( i=0; ((cx=getch()) != 13) && i<12; i++ )
    {
    if (cx == 8)
      {
      color = getcolor();
      bkcolor = getbkcolor();
      setcolor( bkcolor );
      sprintf( tempstr, "%c", string[--i] );
outtextxy( cur_x, cur_y, tempstr );
      string[i--] = ' ';
      cur_x -= XPOS0;
      setcolor( color );
      }
    else
      {
      string[i] = cx;
      sprintf( tempstr, "%c", cx );
      outtextxy( cur_x += XPOS0, cur_y, tempstr );
      }
    }

  string[i] = '\0';
}

void put_string( int x, int y, char string[] )
{
```

```
  outtextxy( x, cur_y+=y, string );
}

void put_ij( int i, int j )
{
  char string[20];

  setviewport( 514, scr_ymax-13, scr_xmax, scr_ymax, 1 );
  setcolor( WHITE );

  setlinestyle( SOLID_LINE, 1, THICK_WIDTH );

  clearviewport();

  moveto( 0, 0 );
  lineto( 0, 13 );
  lineto( scr_xmax-513, 13 );
  lineto( scr_xmax-513, 0 );

  sprintf( string, "i=%2d, j=%2d", i, j );
  outtextxy( XPOS1, 5, string );

  setviewport( 514, 0, scr_xmax, scr_ymax-13, 1 );
}

/*****************************************************/
/*                                                   */
/* FUNCTION: Provides functions for displaying data. */
/*                                                   */
/*****************************************************/

int scr_xmax, scr_ymax;
int win_xmax, win_ymax;

static struct palettetype color_pal;
static struct palettetype gray_pal;
static int maxcolor;

void gopen( void )
{
  int gdriver=DETECT, gmode, errorcode;
  int i;

  closegraph();
  initgraph (&gdriver,&gmode,"");

  if ( (errorcode = graphresult() ) != grOk )
```

```
    {
    closegraph();
    printf( "Graphics error: %s\n", grapherrormsg(errorcode) );
    printf( "Press any key to halt:" );
    getch();
    exit( 1 );
    }

  maxcolor = getmaxcolor( );

  getpalette( &color_pal );

  getpalette( &gray_pal );

  for ( i=0; i<=maxcolor; i++ )
    gray_pal.colors[i] = 39-i;

  for ( i=0; i<gray_pal.size; i++ )
    setrgbpalette( gray_pal.colors[i], i*4, i*4, i*4 );

  set_color();

  scr_xmax = getmaxx();
  scr_ymax = getmaxy();

  win_xmax = scr_ymax;
  win_ymax = scr_ymax;

  setviewport( 0, 0, win_xmax, win_ymax, 1 );
  clearviewport();

  setcolor( BLUE );
  setlinestyle( SOLID_LINE, 1, THICK_WIDTH );
  rectangle( 0, 0, win_xmax, win_ymax );
}

void gclose( void )
{
  closegraph();
}

void set_gray( void )
{
  int i;
  for ( i=0; i<=maxcolor; i++ )
    setpalette( i, gray_pal.colors[i] );
}
```

```
void set_color( void )
{
  int i;
  for ( i=0; i<=maxcolor; i++ )
    setpalette( i, color_pal.colors[i] );
}
```

Index

absolute error, 311
absolutely integrable, 49
accumulation error, 313
acoustic signals, 416
acoustic wave speed, 105
acoustic waves, 104
acoustics, 51
addition of matrices, 163
addition theorem, 86
additive generators, 468
additive noise, 494, 505
adjoint of a matrix, 172, 288
admissibility condition, 153
ALGOL, 344
algorithm selection key, 484
algorithmic error, 313, 315
aliasing, 98, 416
all poles method, 515
allocation, 329
amplitude envelope, 129
amplitude modulated imaging, 136
amplitude modulation, 418, 506
amplitude modulations, 134
amplitude spectrum, 77, 80, 128
analogue signal, 98
analogue signals, 93
analogue-to-digital, 96
analytic, 16
analytic continuation, 509
analytic function, 18
analytic functions, 18
analytic part, 29
analytic signal, 9, 129, 132, 418
angular frequency, 76, 105, 108
anomalous diffusion equation, 554
APL, 345
arbitrage, 584
Argand diagram, 11, 77
argument, 11
arithmetic expressions, 369
arithmetic operations, 318
array processing, 378
arrays, 367
assignment operators, 368
associative law for matrices, 165
associativity, 49, 92
asymptotic forms, 565
asymptotic Green's function solution, 107
asymptotic solution, 564
attack, 486
augmented matrix, 185, 258
authentication, 440, 443, 446, 486
autoconvolution, 89
autoconvolution theorem, 92
autocorrelation, 90
autocorrelation function, 92, 435, 590, 607
autocorrelation theorem, 92, 544
automatic code generation, 325

back-substitution, 185, 193, 218
backward differencing, 547
Banach lemma, 234
banded matrices, 238
banded matrix, 199
banded systems of equations, 199
bandlimited, 93
bandlimited function, 79, 96
bandlimited functions, 508
bandpass filter, 426
bandpass filters, 99
bandwidth, 508, 525
baseband signal, 131
baseband spectrum, 130
Basic, 343
batch adding, 317
batch systems, 338

Bayes rule, 494, 496
Bayesian estimation, 459, 494, 496
beam profile effects, 462
bench marking, 332
Bermann process, 572
Bersoff's law, 329
Bessel function, 415
bifurcation, 476, 477
bin, 414
binary arithmetic, 307
binary coded decimal, 306
binary number system, 306
binary sequence, 447
binary string conversion, 449
binary to decimal conversion, 308
binomial expansion, 106
bisection, 277
bit error, 578
bit error rate, 578
bit errors, 573
bit rate, 574
bit reversal, 408
bit stream, 573
bits, 305
bitwise logical operators, 368
black box testing, 328
black noise, 564
Black-Scholes analysis, 588
Black-Scholes equation, 587
Black-Scholes formula, 587
Black-Sholes analysis, 578
bloatware, 336
block cyphers, 484
Blum integer, 471
Blum-Mercali generator, 471
blurring, 89
Boltzmann entropy, 504
Boolean quantities, 369
Born approximation, 106, 110, 460
bottom-up design, 332, 386
bottom-up testing, 332
boundary conditions, 256
box counting dimension, 557
Box-Muller transform, 471
Bromwich integral, 121, 561
brown noise, 564
Brownian flights, 544
Brownian motion, 546
Brownian transients, 556, 567
bull/bear cycle, 586, 603
Burg entropy, 515
Burg's maximum entropy method, 515
Butterworth highpass filter, 100
Butterworth lowpass filter, 99

C programming, 305
C programming language, 346, 364
C++ programming, 347
CASE tools, 326, 335
casting, 367
Cauchy distribution, 545
Cauchy's integral formula, 122
Cauchy's residue theorem, 28, 35
Cauchy's theorem, 24, 26, 33
Cauchy-Riemann equations, 16, 17, 23
Cauchy-Schwarz inequality, 211
causal convolution, 121, 149, 548
Caylay-Hamilton theorem, 263
Cayley-Hamilton theorem - applications, 264
center differencing, 103, 255
central limit theorem, 115
cepstrum, 127
changeability, 383
chaos, 474, 477
chaotic signals, 474
char, 366
characteristic equation, 258
characteristic function, 542, 544
characteristic matrix, 256, 517, 537
characteristic polynomial, 269
characters, 318
Chebyshev method, 239
chirp coding, 442, 446, 487
chirp decoding, 447
chirp functions, 446
chirp transform, 445
chirped pulse, 439
chirping parameter, 440
Cholesky's method, 194
circuit theory, 63
clarity of structure, 340
classes, 347
classical analysis, 41

classical derivative, 46
classical Fourier transform, 57
classical fractional integrals, 149
closed contour, 31
closed path, 21
COBOL, 344
code generation, 336, 448
code portability, 334
code-book protocol, 470
coding process, 451
cofactors, 172–174
coherent Gaussian noise, 542
cohesion, 330, 385
column matrix, 286
column vector, 286
comb function, 43, 93
commenting, 324
common depth point stack, 138
communications engineering, 364
commutativity, 49, 92
compiling, 392
complex analogue signals, 231
complex analysis, 9
complex cepstrum, 127
complex conjugate, 86
complex conjugate of a matrix, 289
complex digital signals, 231
complex form, 13, 57
complex Fourier series, 67, 69, 70, 103
complex function, 16, 18
complex functions, 13
complex integration, 20
complex number, 9
complex plane, 15, 18, 25
complex plane analysis, 129
complex planes, 13
complex roots, 9
complex spectrum, 77
complex variable, 13
complex zeros, 136
complexity, 330, 383
complexity measure, 384
comprehensibility, 383
compressibility, 105
compression space, 69
computational error, 312, 314
computer storage requirements, 202
concatenation, 449
condition number, 219, 222
conditional branching, 370
conditional heteroskedacity, 593
conditional probability, 495
conditioning, 317
conditioning of linear systems, 218
confusion, 443
conjugate gradient method, 183, 238
constrained deconvolution, 457
constraints, 329
continuity condition, 567
continuous derivatives, 47, 50
continuous wavefield, 105
continuously differentiable, 47
control flow, 328
control statements, 302, 369
control structures, 317, 319
convergence, 237
convergence - necessary condition, 243
convergence - sufficient condition, 243
conversion error, 312, 314
convolution, 49, 87
convolution equation, 53
convolution integral, 41, 51, 89
convolution operation, 107
convolution process, 41, 525
convolution representation, 550
convolution sum, 51, 162, 524
convolution theorem, 90, 97, 108, 416, 561
convolution theorem for Laplace transforms, 121
correlation, 87
correlation integral, 89, 153
correlation process, 528
correlation sum, 528
correlation theorem, 91, 435
cosine function, 83
cosine taper, 415
cosine transform, 124
cosinusoidal wave functions, 125
coupled equations, 256
coupling, 330
covert digital communications, 573
Cramers rule, 169, 173
critical states, 582
cross entropy, 507

cross-correlation, 88
Crout's method, 194
cryptanalysis, 486
cryptography, 443, 468, 486
cryptology, 443
cut-off frequency, 99
cycle length, 464, 465, 469, 482
cyclometric complexity, 384
cypher stream, 486
cyphertext, 468

d'Alembertian solution, 561
data acquisition, 460
data analysis, 331
data compression, 68
data consistency, 513
data declarations, 317
data error, 312
data flow design, 331
data flow diagrams, 331
data processing, 311
data type statements, 366
data windowing, 413
database design, 334
dataflow diagram, 340
David Hilbert, 129
DC component, 71, 566
De Moivre's theorem, 13
de-noising, 444
de-trended log price, 589
decibel scale, 416
decimal integer, 366
decimal system, 305
decimal to binary conversion, 307, 308
decision tables, 323
declarations, 367
decoding process, 454
defect amplification, 329
deflation, 268, 270
deflation method, 271
deflation method - non-symmetric matrix, 271
deflation method - symmetric matrix, 272
delta function, 78
delta-sequence, 44
demodulation, 129, 130
dense matrix, 289
density, 105
design, 328
detailed design, 334
detailed flowcharts, 323
determinants, 168
deterministic arithmetic processes, 464
deterministic chaos, 482, 541
DFT, 406
diagonal dominance, 238, 243
diagonal matrix, 242, 287
diagonalization, 190
diagonalization of matrices, 260
dielectric, 104
difference equations, 256
differential operator, 549
differentiation and the Fourier transform, 81
differentiator, 64
diffraction, 81
diffused watermark, 444
diffusion, 443
diffusion equation, 75, 556
diffusion equation solution, 561
diffusion of information, 604
diffusivity, 557
digital filters, 102
digital gradient, 102
digital signals, 93
digital watermarking, 443
digital-to-analogue conversion, 97
dilation, 153
Dirac delta function, 41
Direct Current, 71, 77
direct methods of solution, 183
discontinuity, 43
discrete amplitude spectrum, 416
discrete convolution, 103, 416, 523
discrete correlation, 417, 526
Discrete Cosine Transform, 68
Discrete Fourier Transform, 68
discrete Fourier transform, 70, 101, 256, 406
discrete phase spectrum, 416
discrete power spectrum, 416
discretization, 71, 103
distributive law for matrices, 165
distributivity, 92

do loop, 372
do-loop, 320
documentation, 365
dominant eigenpair, 268
dominant eigenvalue, 267
dominant eigenvector, 268
Doolittle factorization, 282
Doolittle's method, 194
DOS, 338
double, 366, 368
double pole, 29
double precision, 224
double sided Laplace transform, 114
dry run, 328
Duffing oscillator, 480
dynamic memory allocation, 342, 379, 380
dynamic memory management, 341

ease of extension, 340
efficiency, 341
efficient market hypothesis, 578, 580, 584, 589
eigenfunctions, 255, 508
eigenvalue problem, 256
eigenvalues, 255
eigenvalues - formal methods of solution, 257
eigenvector, 256
El Farol bar problem, 603
electromagnetic wave speed, 105
electromagnetism, 51
Elliot waves, 582
else-if constructs, 370
embedding information in noise, 573
encryption, 485
endpoint extension, 530
energy theorem, 87
Enigma cypher, 470
entropy, 443, 468, 503
entropy conservation law, 504
error analysis, 311
error growth, 313
error reduction, 316
Euclidean geometry, 542
Euclidean norm, 209–211
exact arithmetic, 218
exponent, 305
exponential error growth, 313
exponential format, 366
exponential function, 12
external support, 341

Fast Fourier Transform, 71
fast Fourier transform, 405, 406, 534, 565
feedback process, 534
Feigenbaum diagram, 476, 483
FFT, 406, 417
FFT - C function, 410
field theory, 51
financial derivatives, 585
finite band, 80
finite difference analysis, 238
finite differencing, 479
finite element analysis, 238
Finite Impulse Response filter, 53
finite impulse response filter, 522, 526, 534
first order solution, 461
fixed point representation, 305, 314
fixed point storage, 309
float, 366, 368
floating point error, 314
floating point representation, 305
floating-point number, 366
flow diagrams, 334
flowcharts, 323
FM imaging, 136
Fokker-Plank-Kolmogorov equation, 555
for loop, 371
for-loop, 320
formal methods of solution, 171
formatted input control, 366
formatted output, 365
fortran, 341
forward differencing, 102
forward-substitution, 193
Fourier amplitude, 81
Fourier analysis, 42
Fourier coefficients, 57, 68
Fourier cosine series, 60
Fourier dimension, 533, 541, 557, 608
Fourier filters, 404
Fourier phase, 81
Fourier series, 57, 63
Fourier sine series, 61

Fourier space, 79, 90, 405, 427
Fourier theory, 67
Fourier transform, 9, 75, 77, 90, 103, 223, 413
Fourier transform pair, 69, 70, 78, 79
Fourier transforms, 92
fractal geometry, 542
fractal demodulation, 575
fractal diffusion equation, 556
fractal dimension, 541, 573
fractal dimension signature, 575
fractal geometry, 545, 553, 596
fractal market hypothesis, 578, 602
fractal modulation, 573–575
fractal noise, 564
fractal noise generation, 573
fractal signals, 152
fractal time, 567
fractal time series, 595
fractals, 477
fractals/bit, 578
fractional calculus, 115, 547, 553
fractional differential, 82
fractional differential operator, 547
fractional differentiation, 547, 551
fractional diffusion equations, 554
fractional dynamics, 554
fractional integrals, 547
fractional Laplacean, 555
fractional operators, 115
frequency distribution, 77, 541
frequency hopping, 574
frequency modulated signals, 439
frequency modulation, 132, 573
frequency of oscillation, 52
Fresnel transform, 445
full pivoting, 189
function, 329, 365
functional specification, 333
functional testing, 328
functionality, 346
functions, 320, 327
Fynman diagram, 51

Gabor transform, 139
Gamma distribution, 533, 542
Gamma function, 116, 547
gamma function, 149
Gauss' method, 184
Gauss-Jordan method, 193
Gauss-Seidel method, 238, 239
Gaussian distributed deviates, 472
Gaussian distribution, 115, 545
Gaussian elimination, 184, 218, 258
Gaussian function, 83
Gaussian highpass filter, 99
Gaussian lowpass filter, 99
Gaussian noise, 477
Gaussian probability density function, 557
Gaussian random number generator, 471
geese, 397
general stochastic models, 572
generalized derivative, 46
generalized Fourier transform, 57
generalized Hann window, 415
George Green, 51
George Green Memorial Committee, 51
Gerchberg-Papoulis method, 511
Gerschgorin's theorem, 277
Gibbs' phenomenon, 68
Givens' method, 278
GOTO statement, 342
goto statement, 387
Green's function, 41, 51, 52, 460, 559
Green's function solution, 53
group delay, 145

Hénon-Heiles equations, 480
half range series, 62
half-integration, 548
Hamming window, 415
Hann window, 415
Hanning window, 415
harmonic components, 60
harmonics, 60
Heaviside expansion theorem, 115
Heaviside step function, 561
hedging, 586
Helmholtz equation, 107
Hermitian conjugate of a matrix, 289
Hermitian matrix, 289
Hessenberg form, 283
Hessenberg matrix, 290
highpass filter, 64, 65, 426

highpass filters, 99
highpass system, 63
Hilbert matrices, 223
Hilbert matrix, 219, 289
Hilbert space, 228, 513
Hilbert transform, 129, 418, 477
Hill's equation, 480
histogram, 468
Holder's inequality, 211, 212
homogeneity condition, 209
homogeneous linear systems, 171
homogeneous systems, 175, 255
homogeneous wave equation, 255
homomorphic filter, 458
homomorphic process, 458
homomorphic systems, 128
Householder matrix, 281
Householder's method, 279
Hurst exponent, 596
Hurst processes, 596
Hurst's equation, 597
hybrid programming, 332

ideal bandpass filter, 100
ideal highpass filter, 99
ideal lowpass filter, 99
idempotent matrix, 288
identifier, 318
identifiers, 324
identify operator, 550
identity matrix, 287
if-else constructs, 370
if-then-else statement, 319
if-then-else statements, 328
ill-conditioned, 428
ill-conditioned systems, 176, 218
ill-conditioning, 222
ill-posed problems, 510
image compression, 68
imaginary part, 9, 91
important integral transforms, 75
improper function, 41
impulse, 51
Impulse Response Function, 89
impulse response function, 51, 150, 238, 434, 437, 496, 524, 549
independent development, 383
indirect methods of solution, 237
infinite impulse response filter, 534
infinite integral, 49
infinite moments, 545
infinite variance, 546
infinity norm, 209
infinity of solution, 175
information and entropy, 503
information entropy, 483, 504
inheritance, 347
inhomogeneous linear systems, 170
inhomogeneous systems, 175, 255
inhomogeneous wave equation, 52
initial condition, 468
input, 318
instantaneous frequency, 439
instantaneous phase, 134, 418
int, 366
integers, 318
integral equation, 51
integral operator, 549
integral representation, 45
integral transforms, 223
integration and the Fourier transform, 82
interactive restoration, 432
internal functions, 375
interpolation, 97, 533
inverse DFT, 417, 418
inverse discrete Fourier transform, 565
inverse filter, 100, 427, 501
inverse Fourier operator, 78
inverse Fourier transform, 78
inverse iteration, 285
inverse Laplace transform, 115, 120, 223
inverse matrix, 167, 171, 288
inverse problems, 405
inverse solution, 571
inverse weighted mean square error, 514
iteration, 371
iteration matrix, 237, 238
iteration number, 237
iteration process, 239
iterative improvement, 223, 225
iterative methods, 237, 240
iterative solutions, 210
iterative techniques, 537

Jackson method, 330

Jacobi's method, 238, 239
Jacobi's method - symmetric matrices, 272
Java, 347
jitter, 138, 460
job control language, 337
John Bromwich, 115
Joint Photographic Experts Group, 68
joint probability, 472, 495
Joker effect, 597
Jordan's methods, 190
Joseph Fourier, 75, 115
jump discontinuity, 46

Kaiser window, 415
kernel, 526
key, 468, 469
key exchange, 469, 486
Kronecker delta, 166
kronecker delta, 432
Kronecker delta function, 101
kurtosis, 532, 593

Lévy distributions, 545, 554
Lévy flights, 544, 545, 556
Lagrange multiplier, 458, 506
languages, 336
Laplace, 51
Laplace space, 115
Laplace transform, 114, 223, 548, 561
Laplace transform of a derivative, 119
Laplace transform of a periodic function, 117
Laplace transform of an integral, 119
laplacean, 553
least squares method, 229
least squares principle, 226, 428
libraries, 375
linear congruential generator, 467
linear congruential method, 464
linear convolution models, 229
linear differential operator, 52
linear discrete system, 151
linear eigenvalue problem, 162
linear error growth, 313
linear feedback shift register, 468
linear FM pulses, 439
linear frequency modulation, 439
linear polynomial models, 227
linear simultaneous equations, 162
linear sweep, 447
linearly dependent eigenvectors, 259
linearly independent eigenvectors, 260
linking, 392
Linux, 340
log price increments, 591
logarithmic feedback map, 479
logarithmic sweep, 447
logarithmic transform, 128
logical expressions, 369
logical relational operations, 318
logical testing, 328
logistic map, 475, 478
lognormal random walk, 583
lognormal random walk model, 601
long, 366
looping, 371
Lord Kelvin, 76
Lorenz equations, 479
low-level language, 364
low-level specifications, 323
lower triangular matrix, 193, 242, 288
lowpass filter, 67, 131, 426
lowpass filtering, 68
lowpass filters, 99
lowpass system, 65
LR method, 282
LU factorization, 193, 202
Lyapunov Exponent, 481

macro-economic models, 589
maintaining accuracy, 316
Mandelbrot stable paretian hypothesis, 593
mantissa, 305
mapping, 13
market analysis, 583
matched filter, 437, 445
Mathieu equation, 480
matrices, 163
matrix algebra, 163
matrix equations, 162
matrix inversion by Jordan's method, 191
matrix inversion by LU factorization, 198
matrix norm, 208
Matrix norms, 215
matrix norms - basic definition, 215

Matthews cypher, 483
Max Born, 106
maximum entropy, 81, 516
maximum entropy deconvolution, 505, 506
maximum entropy method, 503
maximum error, 315
maximum likelihood filter, 501
maximum likelihood method, 498, 501
maximum *a posteriori* method, 497, 502
McCabe cyclometric complexity, 330
mean, 532
mean square error, 227, 230
median filter, 532
medical ultrasonic imaging, 104
Mellin transform, 115
memory, 391, 548
memory management, 377
Mersenne prime numbers, 465
metric spaces, 210
Minkowski dimension, 557
Minkowski inequality, 212, 214
minors of a matrix, 172
modern portfolio theory, 584
modification history, 395
modular based functions, 464
modular design, 364
modular programming, 302, 332, 390, 394
modular programming in C, 377
modularity, 329, 381
modulation, 129, 130
module complexity, 383
module coupling, 384
module size, 330, 382
modulus, 11
moments, 532
monotonic convergence, 243, 476
Morse code, 470
moving average filter, 531
moving window, 103, 526, 575
moving window filters, 531
moving window principle, 536, 571
multi-access, 338
multi-fractal characteristics, 572
multi-fractal financial analysis, 580
multi-fractal market hypothesis, 578, 604, 607
multi-fractal statistics, 572
multi-fractals, 541
multi-random fractals, 567
multi-resolution, 152
multi-way decision, 370
multiple inheritance, 347, 348
multiple scattering, 109, 460
multiplexing, 126
multiplication of a matrix by a scalar, 164
multiplication of matrices, 164
multiplicative model, 128

narrowband signal, 131
natural pivoting, 218
naturalness of application, 340
necessary condition for convergence, 245
negative frequency, 132
nesting, 316
nilpotent matrix, 288
noise, 459
non-destructive evaluation, 104
non-differentiable, 21
non-equilibrium phase transitions, 555
non-periodic function, 76
non-recursive filters, 404, 522
non-singular matrix, 167, 271, 287
non-stationarity, 152
non-stationary algorithm, 566
non-stationary convolution, 537
non-stationary fractal signals, 568
non-stationary fractional dynamics, 556
non-stationary processes, 544
non-stationary signals, 535
non-trivial solutions, 175
non-uniform distribution, 485
nonlinear coupled equations, 474
nonlinear maps, 474
normal distribution, 83
normalization property, 43
normalized eigenvectors, 259
normalized floating point, 305
normalized floating point binary, 309
null matrix, 286
numerical analysis, 210
numerical error, 311
Nyquist frequency, 93, 97

Oak, 347
object library, 392, 394

object oriented design, 329
off-diagonal elements, 273
off-diagonal positions, 274
Oliver Heaviside, 115
one-dimensional Fourier operator, 76
one-way functions, 471
operating systems, 336
operating systems development, 339
operational constraints, 327
operations on characters, 318
operations on integers, 318
operations on reals, 318
operators, 367
operators of integer order, 549
optical form, 71
optimization, 208
option volatilities, 588
options, 585
Ornstein-Uhlenbeck process, 572
orthogonal basis functions, 126
orthogonal matrix, 288
orthogonal properties of eigenvector, 259
orthogonality, 59, 280
orthogonality principle, 228, 427, 458, 512
oscillatory convergence, 243
out-of-band frequencies, 100
out-of-band noise, 137
outline flowcharts, 323
output, 318
over-determined systems, 177
over-writing, 379
overflow, 267, 309, 314

p-norm, 209
page fault, 391
parameters, 321
parametrization, 554
Parseval's theorem, 86
partial differential equations, 41, 51
partial pivoting, 188
partitioning, 279
Parzan window, 415
Pascal, 343
passing variables, 374
Patterson entropy, 507
Paul Dirac, 41
pencil line beams, 462
periodic function, 18
periodic replication, 96
permeability, 105
permittivity, 105
perturbation of data, 219
perturbation of data and matrices, 221
perturbation of matrices, 220
phase distribution, 478
phase modulation, 573
phase portrait, 477
phase space, 480
phase spectrum, 77, 80, 128
phase transitions, 542
phase unwrapping, 128, 135
piecewize continuous function, 50, 67
pink noise, 564
pipes, 339
pivot, 185
pivotal strategies, 188
pivots and pivoting, 187
PL/1, 345
plaintext, 468, 484
pointers, 348, 374
pointwize behaviour, 44
Poisson, 51
pole, 29
portability, 341
post solution modification, 223
power method, 265
power spectra, 541
power spectral density decomposition, 448
power spectral density function, 77, 544, 590, 607
power spectrum, 77, 81, 435
power spectrum equalization, 435
predator-prey processes, 475
presentation, 395
prime numbers, 465, 471
principal part, 29
principal phase, 11, 134, 478
private key, 444, 469, 471, 486
probability, 494
probability density function, 443, 496, 544
probable error, 313
procedures, 321
process logic, 323
process specification, 322

product theorem, 90, 96
program design, 324
program development time, 326
program documentation, 326
program efficiency, 325
program reliability, 325
program specification, 324
program structures, 317
programming, 334
prolate spheroidal wave function, 509
propagation of information, 604
propagator, 51
proper function, 44
properties of a determinant, 169
prototyping, 335, 375
pseudo chaotic number generation, 482
pseudo chaotic number generators, 482
pseudo code, 322, 417, 418
pseudo random number generation, 443
pseudo random number generator, 466, 469
pseudo random number generators, 462, 573
pseudo random sequences, 463
pseudo-code, 186
pseudocode, 326
public key, 471
public key infrastructure, 471
pulse-echo systems, 110, 461
pulsed wavefield, 105
pure imaginary, 9
pure real, 9

q-signature, 607
q-value, 606
QR method, 282
quadratic iterators, 475
quadratic-Gaussian bandpass filter, 100
quadrature component, 129
quadrature detection, 131
quadrature filters, 131
quadrature signal, 418
quality control, 329
quantum electrodynamics, 51
quantum field theory, 51
quantum Green's functions, 51
quantum mechanics, 51
quantum scattering theory, 106

Rössler equations, 480
radar, 104
random amplitude, 499
random fractal signals, 541
random number streams, 464
random scaling fractals, 542
rapping, 530
Rayleigh quotient, 266
Rayleigh's theorem, 87
reading data, 388
real analogue signals, 230
real digital signals, 230
real part, 9, 91
real random sequences, 463
real space, 90
real zero conversion, 136
real zeros, 136
real-time systems, 338
reals, 318
recursive filters, 522
reduced matrix, 270
reference code, 444
refractive index, 105
regularization, 405
relative error, 311
relatively prime number, 471
relatively prime numbers, 465
relaxation iteration, 242
relaxation method, 239
repeat-loop, 320
repetitions, 319, 386
replicated spectra, 96
requirement specifications, 329
rescaled range analysis, 581, 595, 598
residual vector, 224
residue, 29
residue theorem, 29
reverberation, 460
Richard Fynman, 51
Riemann integral, 149, 547
Riemann-Liouville transform, 149, 547
Riesz operator, 553, 554
ringing, 68
risk analysis tool, 607
Rivest, Shamir and Adleman generator, 471

root mean square, 109
rotation matrix, 273, 274
round-off-errors, 268
rounding down, 312, 314
rounding error, 312
rounding off, 314
rounding up, 312, 314
row matrix, 285
row vector, 285
run length, 463
running weighted average, 526

sampled function, 94
sampling interval, 70
sampling operation, 44
sampling property, 41, 43, 78
sampling theorem, 42, 93
sawtooth map, 478
scale invariance, 542
scaling, 267
scaling or matrices, 203
scatter generating parameters, 107
scattering from layered media, 104
scheduling, 329
Schrödinger equation, 141, 555, 556
Schwarz inequality, 437
scrambling, 469, 484
second order derivative, 103
second order scattering, 461
secure digital communications, 573
seed, 465, 466, 468
segmented programs, 325
seismic imaging, 138
seismology, 104
selections, 319, 386
self-affinity, 541
self-similarity, 477, 541
semantics, 317
semi iterative techniques, 238
sequences, 319
serial method, 274
series solution, 561
session key, 484
Shannon entropy, 504
shared modules, 385
shift theorem, 86
short, 366
shuffling, 467
side lobes, 462
sideband spectrum, 130
sign function, 84
signal analysis, 41
signal attributes, 134
signal independent noise, 430
signal-to-noise ratio, 434, 437, 443
signum function, 42
similarity dimension, 557
similarity theorem, 86
similarity transform, 269, 283
simple closed curve, 23
simple pole, 29
simply connected region, 23
sinc function, 97, 414, 508
sinc interpolation, 97
sine function, 83
sine map, 479
sine transform, 123
single character, 366
single program systems, 338
single scattering, 461
single side-band spectrum, 419
single sideband spectrum, 133
singular matrix, 287
singularities, 21
singularity, 514
skew Hermitian matrix, 289
skewness, 532
slowness, 557
smearing, 89
smoothing, 68
software life cycle, 327
software maintenance, 328
solid state physics, 51
solution to complex systems or equations, 203
solution vector, 241
sparse matrices, 238, 537
sparse matrix, 289
spatial frequency, 76
special types of matrices, 285
specification, 327
speckle, 542
spectral analysis, 416
spectral extrapolation, 508

spectral leakage, 414, 415
spectral radius, 246
spectrogram, 140
spectrum, 71, 77
speech processing, 128
speed, 391
spread-spectrum, 574
square integrable, 76
square wave, 59
square wave signal, 60
stability of linear systems, 208
standard deviation, 497, 567
standard form, 71
standards, 329
statement layout, 365
stationary, 89
stationary process, 443
statistical filters, 532
statistical self-affinity, 543
statistics of dimension, 558
steady-state, 63
stenography, 443
step length, 102
STFT, 140
Stieltjes-Riemann integral, 129
stochastic resonance, 542
stochastic term, 460
strange attraction, 480
string (null terminated), 366
string and system testing, 335
string control, 366
structure block diagrams, 323
structure charts, 334
structured code, 323
structured programming, 302, 364, 386
structured programming in C, 377
stubs, 332
Sturm sequence, 275
Sturm sequence iteration, 256, 275
style, 395
style and presentation, 324
sub-processes, 393
subprogram, 365
subprograms, 317, 320, 393
subroutines, 393
successive doubling method, 407
successive improvement, 223
successive-over-relaxation method, 238, 239
successive-under-relaxation, 239
sufficient condition for convergence, 244
switch constructs, 370
symbolic constants, 367
symbolic definition, 44
symmetric cypher, 471
symmetric encryption, 469
symmetric functions, 92
symmetric matrix, 287
symmetry of the Fourier transform, 115
syntax, 317
system design, 327
system lifecycle, 381

tangent feedback map, 479
Taylor series, 57, 509
technical specification, 333
temporal correlation, 470
temporal frequency, 76
tent map, 479
test data, 328
test-beds, 332
testing, 328, 332
testing procedures, 302, 325
thermal conduction, 75
thermodynamics, 75
threshold partitioning, 485
tick data, 579
time history, 104, 138
time invariant linear systems, 89, 548
time-sharing, 338
Toeplitz matrix, 290, 517
top-down design, 330
top-down planning, 364
tophat function, 42, 82, 97
trace of a matrix, 287
transcription error, 312
transfer function, 434, 508
transformation matrix, 273
translation, 153
transmission line, 53
transpose matrix, 287
transpose of a matrix, 166
triangle inequality, 211
triangularization, 185
tridiagonal matrix, 289, 525, 536

tridiagonal systems, 199
trigonometric functions, 59
trigonometrical Fourier series, 58, 68
trigonometrical series, 57
trivial solution, 175
truncation error, 126, 311, 312, 314
two dimensional arrays, 374
two-dimensional Fourier transform, 81
two-way travel time, 108

unconditional branching, 369
under-determined systems, 177
underflow, 314
unformatted output, 365
uniform distribution, 438, 443, 485
uniform PDF, 486
uniformly distributed deviates, 472
unit circle, 36
unit matrix, 166, 287
unit step function, 42, 85
unit testing, 335
UNIX, 339, 346
UNIX filters, 340
unsigned, 366
unsigned decimal integer, 366
unwrapped phase, 136
upper Hessenberg form, 283
upper Hessenberg matrix, 283
upper triangular matrix, 184, 193, 242, 288
user defined functions, 372
user requirement, 333

van der Monde matrices, 223
variable names, 367
variance, 532
VAX/VMS, 342
vector norm, 208, 209
virtual address extension, 310
virtual memory, 391
virtual memory system, 391
void functions, 373
volatility, 593, 608
volatility term structure, 594
vols, 588

Walsh transform, 125
Warnier diagrams, 323
watermarking, 440, 447
wave equation, 105, 556
wave equation solution, 560
wavefield, 105
wavelet decomposition, 448, 449
wavelet function, 152
wavelet space, 154
wavelet transform, 152, 449
wavenumber, 105
weak scattering, 106, 109
weighting functions, 512
Welch window, 415
well-conditioned, 430
Weyl transform, 149
while loop, 371
while-loop, 320
white box testing, 328
white noise, 438, 564
white spectrum, 437
Wiener filter, 428, 434
Wiener process, 572
Wiener-Hopf equation, 121
Wigner transform, 141
Wigner-Ville transform, 142
Wilkinson test, 507
window, 526
window function, 413
windowing function, 140
word length, 309, 313
World Wide Web, 347
writing data, 388
Wyle transform, 547

X-ray crystallography, 81

z-transfer function, 151
z-transform, 150
zero frequency, 71
zero padding, 530, 536
zero-mean Gaussian distribution, 497
zero-mean white Gaussian noise, 544