2015-01-27

Digital Signal Processing Foundations

David Dorran
Technological University Dublin, david.dorran@tudublin.ie

2015-01-27

Digital Signal Processing Foundations

David Dorran

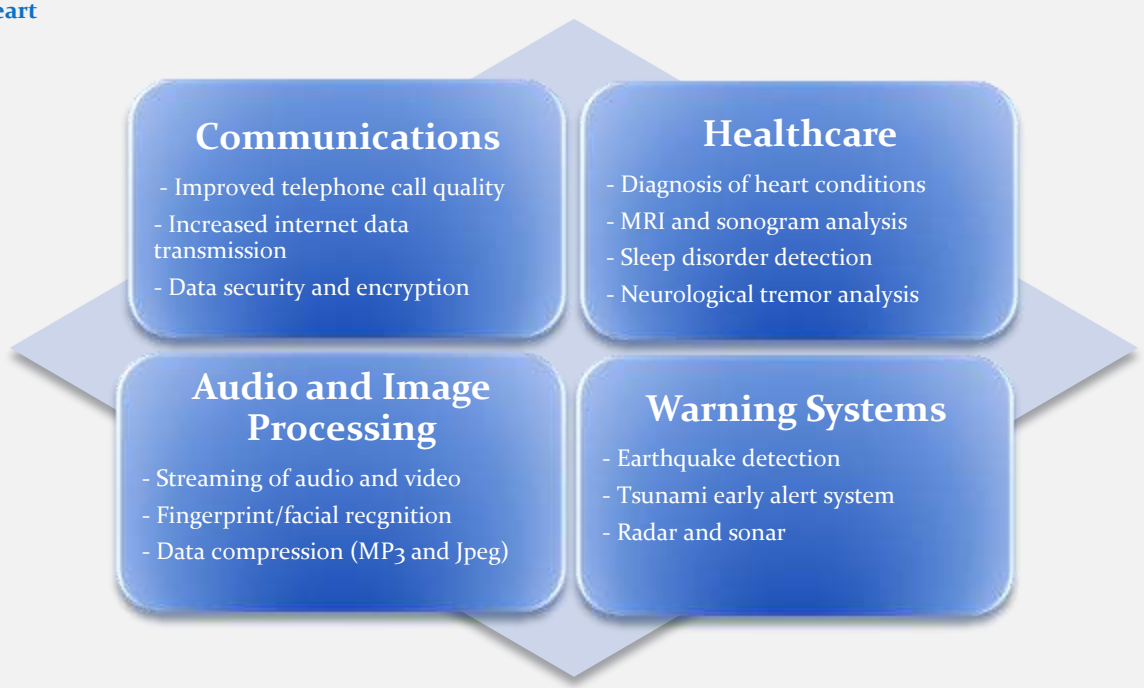
Table of Contents

Introduction

Signals are all around us and come in a wide variety of shapes and forms. When we speak we create pressure variations in the air which generate audio signals; earthquakes produce large seismic signals; healthcare professionals monitor ECG signals which capture the electrical activity of the heart; radio, internet and telephone signals are being transmitted across the world; the list of signals is endless! (see 2 minute video at pzdsp.com/vid1 for some examples)

An ECG signal showing the electrical activity of the heart

Digital signal processing (DSP) is primarily about making use of computers to help us analyse and manipulate signals in order to help us with our everyday lives. To get a flavour of where DSP is being used check out the lists below; it really is a key component in many innovative solutions and products in recent times.



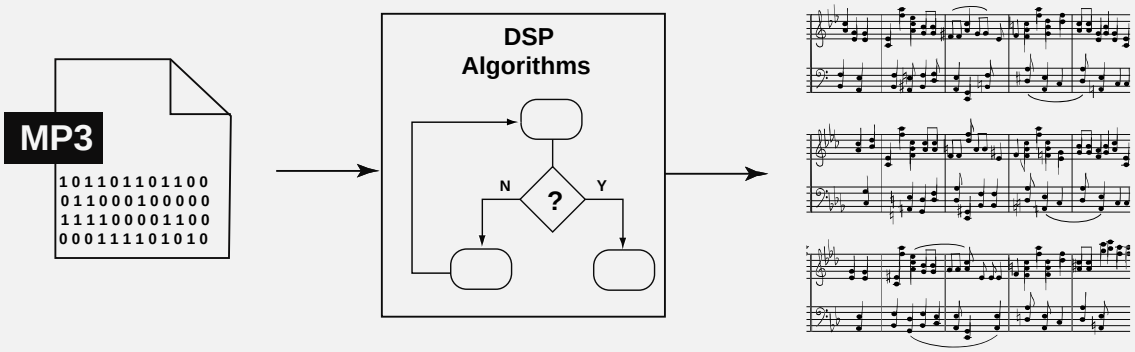
This document introduces a few of the basic concepts of digital signal processing with relatively little mathematics. You should treat it as a relatively gentle introduction to the area which will hopefully provide a route to understanding more sophisticated techniques.

Two example applications

DSP impacts many areas of our lives but it's often executing in the background and can go unnoticed by a casual user. As you become comfortable with the various signal processing techniques you will start to appreciate more and more where DSP plays its role in helping us go about our daily business. Here are just a couple of examples of where DSP is applied that most people can readily appreciate.

Music transcription

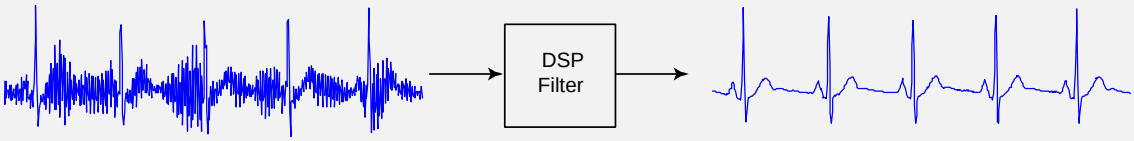
The digital age has meant that everyone has easy access to large amounts of music and it's very easy to listen to whatever appeals to you whenever and wherever you want. While DSP has made this possible through compression techniques like MP3 it can also be used to automatically convert your digital music collection into musical notation or score. The DSP techniques work by analysing the frequencies of each note using Fourier transform techniques together with statistical and music knowledge to identify which notes being played in a recording. So if you'd like to find out what chords are being played in your favourite guitar riff then DSP can come to the rescue!



My early research career was all about analysing and manipulating both music and speech signals. I was part of an audio research group who were developing DSP techniques to transcribe music, isolate instruments from a recording and to either speed up or slow down the playback rate of music. You can see some of the results of this research in a software application called Riffstation – 'The Ultimate Guitar App'! Check it out at riffstation.com!

'Cleaning' a 'noisy' ECG signal

The signals that you see on a heart monitor are taken from a patient who has electrode pads attached to his/her skin and electrical leads carry the signal from the patient to the monitor. Unfortunately any electrical lead can suffer from interference as a result of being in the vicinity of other electrical equipment - and there are lots of interesting electrical devices in hospitals! This interference, referred to as noise, would make the ECG (electrocardiogram) signal very difficult for a healthcare professional to interpret as it effects what the signal would look like on the monitor. DSP filtering techniques clean up the noisy signal so that the noise is removed making it easier for doctors to read the signal.



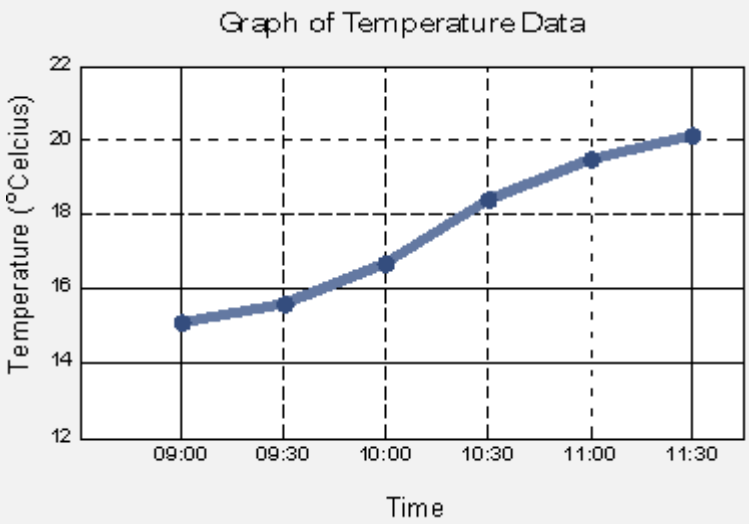
DSP doesn't have to stop with just removing noise from the ECG signal, it can go much further by detecting the heart rate and automatically identifying abnormalities within each cycle of the ECG waveform (see pzdsp.com/vid2 for an ECG heart rate analysis example). There is ongoing research into how DSP techniques can further help with the diagnosis of health conditions through the analysis of ECG signals and other physiological signals like brain waves (EEG – Electroencephalogram). This is an area of significant commercial interest with large multinationals investing heavily in research into such topics.

Digital/Discrete signals

At some stage in your early life you will have undertaken an investigation whereby you have measured something like the temperature outside your school or perhaps the amount of rain that has fallen. When you did this you would have recorded your measurements every so often and then tabulated them; and once you had finished gathering your data you probably would have plotted a graph like the one shown at the bottom of this page.

Time	Temperature (°C)
09:00	15.1
09:30	15.6
10:00	16.7
10:30	18.4
11:00	19.5
11:30	20.1

You didn't realise it at the time but you were actually capturing your first digital signal! By measuring a signal (temperature as it changes over time) at discrete time intervals (e.g. every half hour) you created a list of numbers that represented a signal. That sequence of numbers that you made a note of was a digital signal!



A digital signal is just a sequence of numbers which represents some signal of interest.
The temperature signal example is represented by the following sequence of numbers:
[*15.1 15.6 16.7 18.4 19.5 20.1*]

Note that it's easier to interpret digital signals when they are graphed!

I can't emphasise this basic fact that a digital signal is a list of sequence of numbers enough. It's a key point that if somehow goes unappreciated can cause no end of confusion. It's also the key reason why computers and DSP are linked so closely together, since computers are extremely good at processing (adding, multiplying, etc.) sequences of numbers extremely quickly and accurately. Digital signals and computers

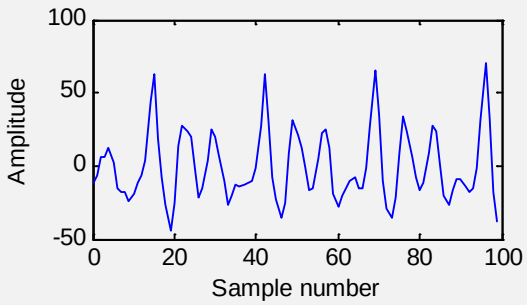
really are a perfect match and this combination has allowed engineers create some amazing technology which has become part and parcel of how we live today.

At this stage you might be wondering how signals are recorded on to a computer so that we can take advantage of their computational power. One way is to simply type the numbers into a spreadsheet or text editor like Notepad++, but there is of course a much easier way using some electronic components to do the job automatically. If you'd like to see a demo of this in action check out pzdsp.com/vid3.

While it is true that digital signals are just a sequence of numbers we normally like to visualise them graphically since generally we find it much easier to interpret a sequence of numbers when they are plotted in a graph. As an example take a look at the sequence of numbers shown below. It might not be obvious but there is a repeating pattern in that digital signal. Now take a look at the graph of the same digital signal to the right – I'm sure you'll agree that it's easier to see the repeating pattern in the graph!

This sequence of 100 numbers is a digital signal which represents a short 20ms segment of a speech signal:

-11 -6 6 6 13 2 -15 -18 -18 -24 -19 -12 -7 4 44 63 19 -8 -27 -44 -25 14 28 24 20 -2 -22 -15 3 25 20 7 -10 -27 -21 -13 -14 -13 -11 -10 -2 27 63 30 -8 -23 -35 -26 8 31 21 12 -2 -17 -15 5 22 25 12 -19 -28 -20 -15 -10 -8 -15 -15 -2 27 65 35 -10 -29 -35 -22 8 34 24 6 -8 -17 -11 8 28 24 1 -21 -27 -17 -9 -9 -14 -18 -15 -1 31 71 32 -18 -38



A plot of a quasi-periodic digital signal given by the numbers to the left. There are about 3.5 repetitions in the signal shown.

NOTE: I'd like to point out that digital signals are often referred to as discrete signals (or more correctly discrete-time signals), I often use the terms interchangeably myself. There is however a subtle difference between them which I don't explain in these notes as I feel it can be a distraction from the key point that a discrete/digital signal is simply sequence of numbers. If you would to find out more you can always make use of your favourite search engine!

Terminology used

Whenever you start doing something new you have to get used to a new 'language' and terminology in order to communicate with everyone else involved. This applies to sports, board games, work and even digital signal processing. If you don't know the

terminology you won't be able to discuss your ideas with others and work on projects within a team. Unfortunately in signal processing there is quite a bit of new terminology and also a requirement to use mathematical notation to explain some concepts. Getting used to the terminology can take some time even though the concepts behind the terminology can be pretty straightforward. So if you find yourself being confused by the terminology or mathematical notation remind yourself that the concepts are the most important aspect and being comfortable with the new terminology will come with time. The next few paragraphs introduce some of the terminology and notation that will be used throughout this document.

Q: You weren't told what the sampling rate was for the speech signal example but you could work it out from the information given! See if you can work out why it's 5000Hz!

In the temperature measurement example shown earlier the temperature was measured every 30 minutes. In this case the temperature signal is said to be *sampled* at a particular *sampling rate* or *sampling frequency*. The sampling rate in this case is 2 samples per hour, which is the same as saying that the *sampling interval* is 30 minutes. Each measurement taken is referred to as a *sample*, so there are 6 samples in the example temperature signal and 100 samples of a speech signal used in the second example.

Also note that the sampling interval is normally given in seconds and the sampling rate given in Hertz, so for the temperature example the sampling interval is 1800 seconds and the sampling rate/frequency is 0.0005555Hz. You can test your understanding of these terms by completing the quiz at pzdsp.com/quiz1 after you complete the next section on Mathematical notation.

Notation used

The table below shows that data used in the temperature measurement example with three additional columns: sample number, sample notation and sample value. You'll notice that the first value is referred to as sample number 0 rather than 1.

Time	Temperature (°C)	Sample Number (n)	Sample Notation	Sample Value
09:00	15.1	0	$x[0]$	15.1
09:30	15.6	1	$x[1]$	15.6
10:00	16.7	2	$x[2]$	16.7
10:30	18.4	3	$x[3]$	18.4
11:00	19.5	4	$x[4]$	19.5
11:30	20.1	5	$x[5]$	20.1

The 4th column of the table shows the mathematical notation used to represent each sample, so sampled number 0 is denoted by $x[0]$ has a value of 15.1; sample number 1 is denoted by $x[1]$ has a value of 15.6; and so on. In this document I'll use the notation $x[n]$ to refer to the entire sequence of numbers associated with a discrete signal and I'll show the sample values as follows:

$x[n] = [15.1 \quad 15.6 \quad 16.7 \quad 18.4 \quad 19.5 \quad 20.1]$

If I'm referring to two or more discrete signals I'll just use a different letter to represent each signal. So here are two more discrete signals $g[n]$ and $w[n]$:

$g[n] = [310\ 23\ 90\ 100\ 390\ 255\ 292\ 42\ 902\ 34\ 102\ 394\ 292\ 492\ 12\ 324\ 841\ 232]$

$w[n] = [-0.12\ 0.35\ 1.01\ -2.3\ 8.7\ 0.02\ ]$

Q: *If the sampling rate associated with g[n] is 5Hz and the sampling interval associated with w[n] is 200ms, which signal shown has a longer duration?*

Q: *Can you see why the result of multiplying g[3]by* w[1] *is 35?*

Working with Matlab/Octave

In order to fully appreciate the concepts presented in this document you should start to create and visualise signals yourself. Matlab and Octave are two applications that are used frequently by signal processing engineers and scientists to analyse and manipulate discrete signals and I would encourage you to install one of these on your computer so that you can try out the techniques that you will learn about; alternatively you can use online versions of these tools with some restrictions http://octave-online.net. Whatever route you choose make sure that you start working on problems as it really is a great way to develop your understanding!

In this document I'll provide links to video tutorials on using Matlab/Octave (I mainly use Matlab but all the tutorials should work with Octave as well) and it would be useful to try and replicate what you see in those tutorials as much as you can without having the example code used directly in front of you. It might a good idea to have the code available close by for reference if required though. The key aspect of this activity is that you write the code yourself! You might feel that there is no need to write code because you understood everything while you watched a video but you will find that that you'll gain valuable insight by trying things out and you'll also be developing the practical skills that you'd need to work in industry. You will most likely find writing your own

code to be a slow process at first but you should persevere as the commands used in the tutorials are used very frequently in practical signal analysis. You'll also see the benefits later on if you start to work on more complex problems.

Additional resources

An online quiz can be found at pzdsp.com/quiz1

Some related videos:

pzdsp.com/vid4 - Introduction to discrete signals

pzdsp.com/vid5 - Capture of discrete signals demonstration

pzdsp.com/vid6 - Choice of sampling rate/sampling interval

pzdsp.com/vid7 - Discrete versus continuous signals using plots

pzdsp.com/vid8 - Overview of process of capturing a discrete signal

pzdsp.com/vid9 - An introduction to aliasing

pzdsp.com/vid10 - Quantisation error explanation

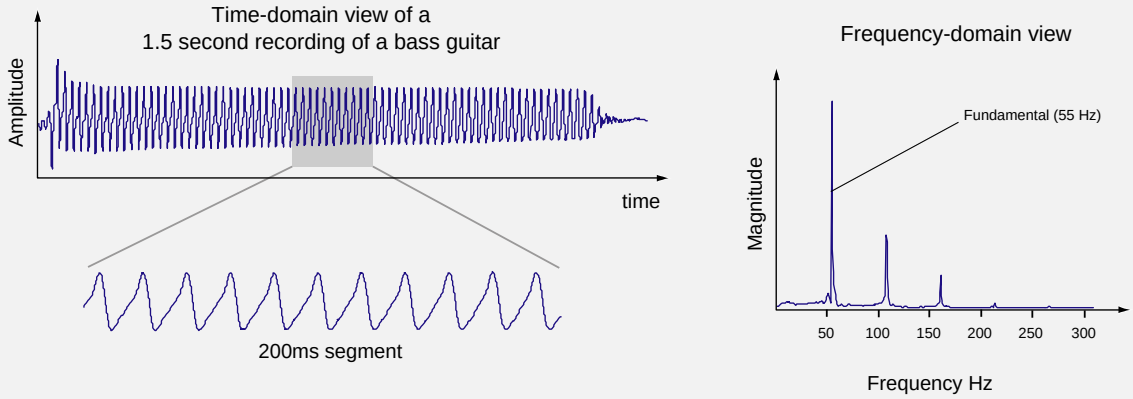
Frequency-domain representation of signals

When someone plays the guitar different sounds are created because the guitar strings vibrate or oscillate at different frequencies. A similar effect can be heard if you stretch an elastic band between your fingers and pluck it and you'd notice that changing either the length or the tension of the band would alter the frequency of the sound since this causes the band to vibrate at a different rate or frequency.

When something is oscillating a repeating pattern is being produced over time. This can be seen with a vibrating elastic band as it moves backwards and forwards through its initial position. Take a look at the following link to clearly see this effect a slow motion video of guitar strings as they oscillate: pzdsp.com/vid11.

The repeating nature associated with the movement of a guitar string can also be seen in a plot of the audio signal produced. You should note that the rate of oscillation of the string is the same as the rate of oscillation of the audio signal since it is the string vibrations that cause pressure variations in the air which we perceive as sound (The recording of the bass guitar signal shown above can be downloaded from pzdsp.com/sig1) . The change in air pressure can also be picked up by a microphone and stored on a computer as a discrete signal.

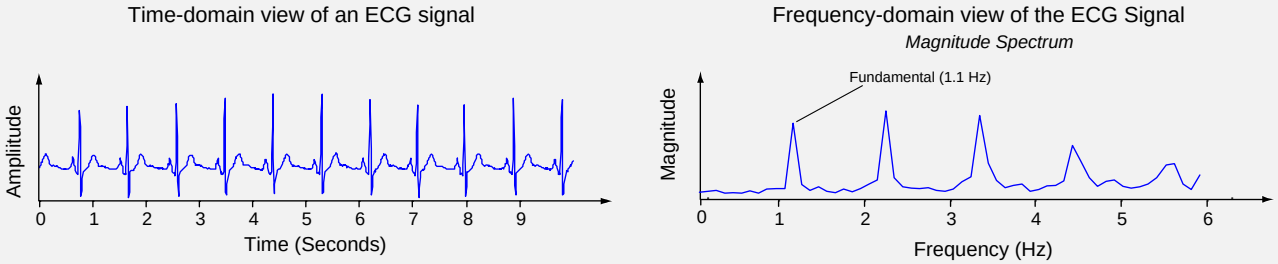
The frequency-domain representation of a signal is a convenient way of showing the oscillation rate of a signal, as explained in the following paragraph.



As can be seen in the figure the sound pressure oscillates after the initial 'attack' or transient component of the signal. This plot of pressure variation over time is referred to as a time-domain plot and by looking closely at this plot you can see that the time to complete one cycle of an oscillation is about 1.82 milliseconds (approx. 11 cycles over a

200ms segment). In other words the cycle is repeating about 55 times every second. To the right of the time-domain plot is a plot of the *magnitude spectrum* which is a frequency-domain representation that can be used to quickly determine the rate of oscillations in time-domain signals. The three relatively large 'spikes' shown in the magnitude spectrum represent the fundamental frequency (55Hz) and the first two harmonics (110Hz and 165Hz). You should notice that you can tell the rate of oscillation (55Hz) quite easily when you look at the signal in the frequency-domain; much more quickly and easily than by analysing the period of the time-domain signal.

This type of repeating pattern doesn't just happen with audio signals and it can be observed in many signals including those from our heart. Your heart will beat at particular rate, or frequency, depending on what you are doing and your heart rate will increase if you go for a run or cycle. Engineers and scientists (and musicians and doctors!) are often analysing the repeating nature of signals and the frequency-domain view of a signal shows the frequency of the repeating patterns in a convenient graph.



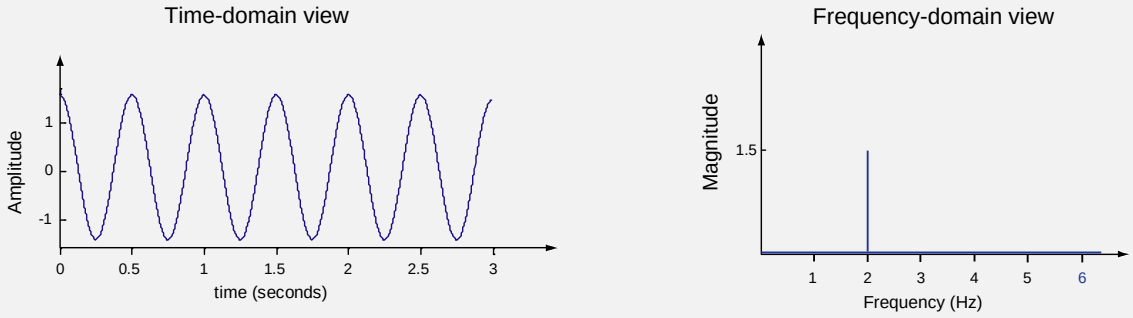
The frequency-domain view of a signal provides another way of analysing a signal which can provide valuable insight into a signals' behaviour. I find it useful to relate this to the way an architect has different drawings of a building depending on who she is dealing with: A client would find it easier to visualise what the building would look like by examining a 3-D view of the building; while a builder would require detailed plans in order to construct the building. Both sets of drawings are representations of the same building and both have their uses. It's the same with the time-domain and frequency-domain views of signals – both represent the same signal and both can be very useful when analysing signals. Here's a link to a video which demonstrates the benefit of both the time-domain view and frequency-domain view of a signal pzdsp.com/vid12.

Frequency-domain graphs of signals are very easy to create using tools like Octave and Matlab and they make use of Fourier analysis techniques to extract frequency information from a time-domain signal. A detailed description of how these techniques work is outside the scope of this document but I've provided links to videos at the end of this section if you would like to find out more. It is worth noting that the basic principle behind all of the Fourier analysis techniques is that any signal can be broken down into a set of sinusoidal signals and this concept is explored in the next couple of subsections.

What are sinusoids?

A sinusoid is a waveform that oscillates smoothly over time and is associated with many signals that occur in nature. For example, when you whistle you create pressure variations in the air which have a sinusoidal shape or of you were to allow an object attached to the end of a spring bounce up and down then the motion of the object would also be sinusoidal (see pzdsp.com/vid13). Even more interestingly it turns out that sinusoids are a fundamental building block of any signal so it's worth spending some time getting used to what they look like and how they can be represented mathematically. This fact was proven mathematically by a French mathematician called Jean Baptiste Joseph Fourier (1768-1830).

There are three features of sinusoidal waveforms that you'll need to be comfortable with: *frequency*, *amplitude* and *phase offset*.



A sinusoidal waveform of amplitude 1.5 and frequency 2 Hz

The figure above shows a time-domain plot of a cosine waveform to the left and its corresponding magnitude spectrum to the right. From the time-domain view notice that the sinusoids amplitude oscillates between 1.5 and -1.5 which means that the amplitude of the sinusoid is 1.5. You'll notice that the sinusoid is repeating every 0.5 seconds, in other words it has a period of 0.5 seconds, which means that it has a frequency of 2Hz.

I'd recommend you check out the interactive animation at pzdsp.com/sinusoids in order to get a clearer idea about these parameters.

The frequency-domain plot of the sinusoid shows a single spike at a frequency of 2Hz. Anytime you have a time-domain plot of a single sinusoid you will observe a single spike in the frequency-domain and the position of the spike on the frequency axis corresponds to the frequency of the sinusoid. The magnitude (height) of the spike is proportional to the amplitude of the sinusoid. You'll see examples of signals with more than one sinusoid present in the next section.

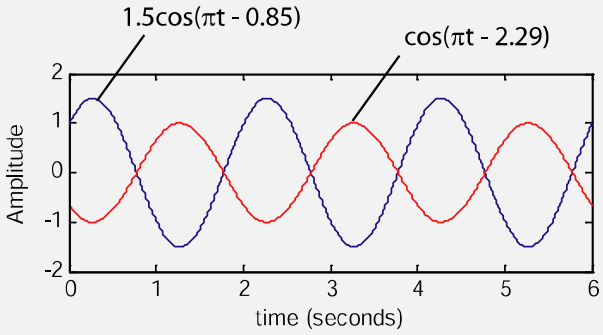
Code to create a plot of a sinusoid:

```
A = 1.5;
f = 2;
phi = 0;
duration = 1; %1 second
T = 1/f;
t=0:T/100:duration;
x = A*cos(2*pi*f*t + phi);
plot(t,x)
xlabel('time (seconds)')
ylabel('Amplitude')
```

Before we look at the phase associated with this sinusoid lets first take a look at a mathematical function often used to represent a sinusoid which is shown below:

$$x(t) = Acos(2\pi ft + \varphi)$$

The A parameter determines the amplitude of the sinusoid; f determines the frequency and φ (Greek letter phi) parameter determines the phase offset (also referred to as the initial phase or phase). The t variable represents time and the mathematical expression is evaluated for a range of values of t in order to create a time-domain signal. So if you wanted to recreate the plot of the sinusoid shown above you'd substitute A with 1.5, f with 2 and φ with 0 to give $x(t) = 1.5\mathbf{cos}(4\pi t)$, and then you could solve this for a number of values of t before finally plotting your graph of $x(t)$ against time.



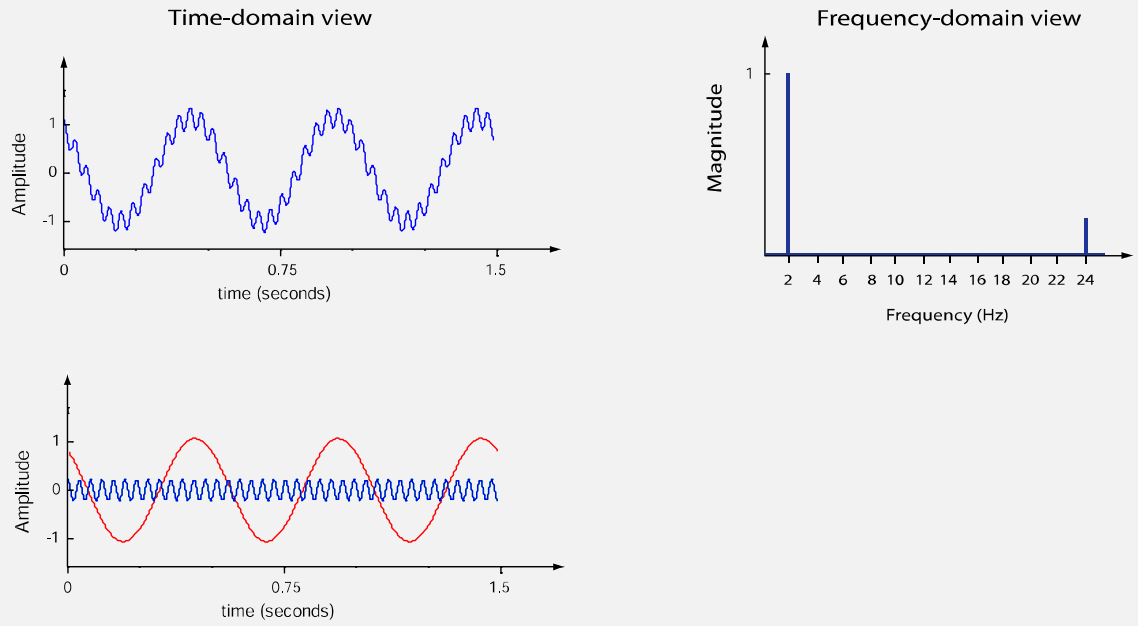
You should notice that when the phase value is zero that the waveform will be a maximum when t=0 and every period of the waveform after that. Changing the phase will change the times when the maximum of the sinusoid will occur. You should try this

out for yourself using the code above and you should also observe that adding 2π to any phase offset value you try out will produce the same waveform. For example, the waveform produced when the phase offset is set to 1.4 will be the same as the waveform produced when the phase is set to $1.4+2\pi$, or $1.4+4\pi$, or even $1.4-2\pi$ for that matter.

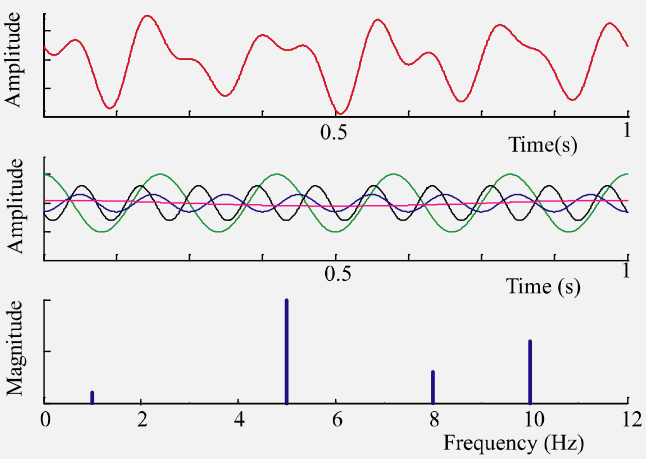
All signals can be decomposed into sinusoids

The French mathematician called Jean-Baptiste Joseph Fourier proved that any signal can be recreated by adding sinusoidal signals together. (see pzdsp.com/vid14 and pzdsp.com/vid15 for video tutorials/demonstrations on this concept).

The frequency-domain view of a signal provides a way to visualise the sinusoids that make up a signal. The magnitude spectrum shows the amplitudes of the various sinusoids which make up a signal, while the phase spectrum shows the phases of the sinusoids which make up a signal.



The figure above shows a waveform (top) which is a plot of the time-domain signal produced when the two sinusoids shown below it are added together. The frequency-domain view of this signal contains two spikes; the spike at 2 Hz is larger than the one at 24Hz because the 2Hz sinusoid is larger (5 times larger) than the 24Hz sinusoid.



The figure to the right shows the magnitude spectrum of a signal in the bottom plot; with the time-domain view of the same signal shown in the top plot. Each of the 'spikes' in the magnitude spectrum represents a sinusoid (there are 4 in total indicating the presence of 4 sinusoids in the signal; in other words the signal could be reproduced by adding four sinusoids together). Each of the four sinusoids, which when summed together produce the time-domain signal shown in the top plot, is shown in the middle plot. The green sinusoid has 5 cycles over the one second duration of the segment shown and therefore has a frequency of 5 Hz; it has the largest amplitude, as can also be seen in the corresponding magnitude spectrum plot where the 'spike' shown at 5 Hz is the largest. It can also be seen in the magnitude spectrum that the 'spike' at 8 Hz is less than half the height of the 5 Hz spike; this can also be seen in the middle plot whereby the sinusoid with 8 cycles in one second has an amplitude of less than a half the amplitude of the 5 Hz sinusoid.

The phase values for each of the sinusoids present in the signal are 0, 0, 3.14, 2.13 radians for the 1, 5, 8, and 10 Hz components. These phase values are phase shifts relative to cosine waveforms. A plot of the phase spectrum shows the phase values plotted against frequency in a similar way to the magnitude spectrum showing the magnitude values plotted against frequency.

If you would like to see a practical application of the frequency-domain then take a look at pzdsp.com/vid12.

Additional resources

An online quiz can be found at pzdsp.com/quiz2

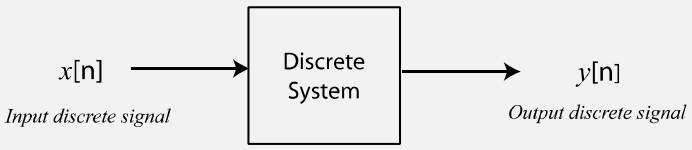
Some related videos:

pzdsp.com/vid16 – Using matlab's fft function

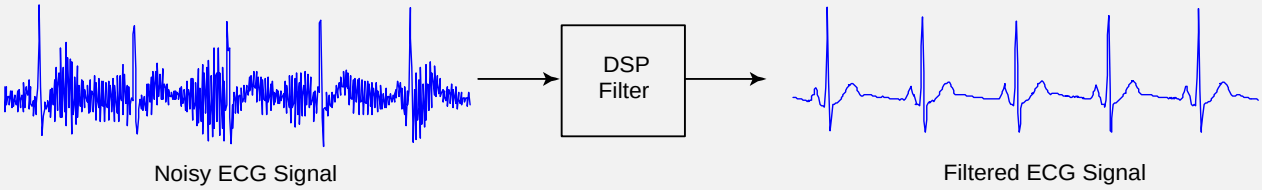
pzdsp.com/vid17– Plotting magnitude spectrum

Discrete systems

Discrete systems are used to manipulate or modify discrete signals in meaningful ways. They take a discrete signal (sequence of numbers) as an input and generate a discrete signal at their output.



One common application of discrete systems is to remove some interference or 'noise' component from a signal. It was mentioned earlier that signals such as ECG signals can be plagued with the introduction of noise from electrical components in the vicinity of the sensors used to capture the signal. This type of noise is normally present at frequencies of 50Hz or 60Hz depending on where you live and discrete systems, known as filters, can be designed to remove such artefacts.



Another common application of discrete systems is to model the behaviour of a real-world systems like a bridge or the suspension of a car. Using the car suspension as an example, imagine you wanted to try out a new suspension design for a racing car and you'd like to see how it performs on bumpy race tracks. One way would be build the suspension, hire a track out for the day, test your design, go back to your workshop to make any adjustments, and repeat as often as needs be. This might prove to be very costly if you need to test frequently! An alternative is to build a discrete system that models your suspension design; then create a signal that models the forces being applied to the car as it goes over bumps; and finally use your computer to analyse and adjust how your design works before going through the costly process of manufacture.

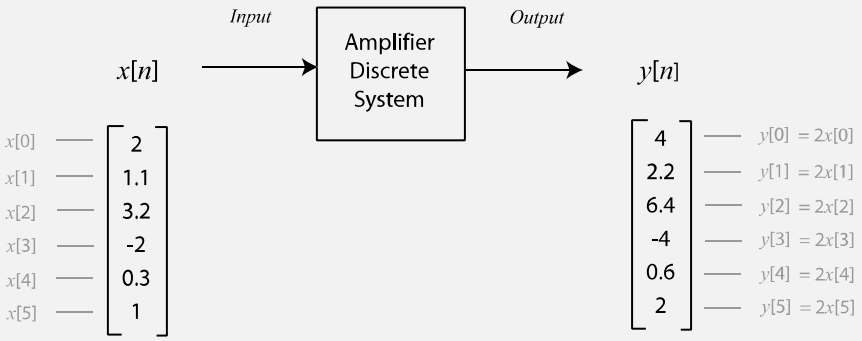
Discrete systems are capable of achieving some amazing results and they are central to Digital Signal Processing. In the next section you'll learn about the three basic components of discrete systems: adders, multipliers and delays. You'll also see how to visualise discrete systems using *signal flow diagrams* and how they relate to the mathematical description of a discrete system i.e. *difference equations*.

Difference equations and signal flow diagrams

In this section you'll see how three practical discrete systems operate and also be shown their *signal flow diagrams* and *difference equations*. After you complete this section it will be important that you are able to relate the difference equations to the signal flow diagrams for each example. You'll also be shown some example code which can be used to implement each system; if you just want to get a flavour of DSP then it's not important for you to fully understand this code but if you want to be able to apply DSP then you will.

Example 1 – An amplifier

Perhaps the most basic discrete system is an amplifier. The figure below shows a discrete signal x[n] being amplified by a factor of 2 with all the samples of the input signal being multiplied by 2. (see pzdsp.com/vid18 for a demo)



Although this is a very basic system it is nonetheless very useful. For example in audio applications such a system would have the effect of increasing the volume of the audio signal.

Discrete systems are very often described mathematically and this amplifier system can be described using the following equation (referred to as a *difference equation*):

$$y[n] = 2x[n]$$

, where $x[n]$ represents the input to the system, $y[n]$ represents the output and n represents the sample number.

By solving this difference equation for different values of n you can determine the output of the system for any input discrete signal (sequence of numbers). For example

let's work out what the output of the amplifier will be if the input to the system is given by the following sequence:

$$x[n] = [10\ \ 20\ \ 5\ \ 50\ \ 60]$$

Let's first solve for $n = 0$, therefore

$$y[0] = 2x[0]$$

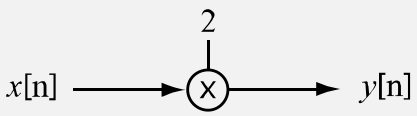
Substituting $x[0]$ for 10 gives

$$y[0] = 20$$

If you were to solve for $n = 1$ you'd find that$y[1] = 40$; $n = 2$ gives $y[2] = 10$; $y[3] = 100$; etc.

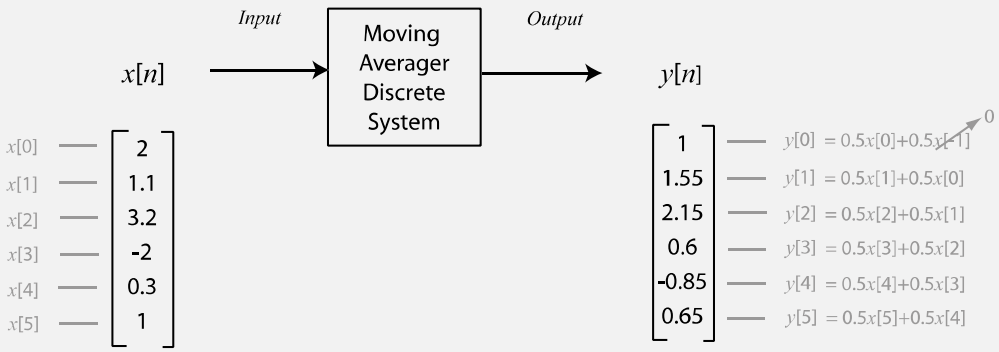
I'm sure you'll have found solving these equations a trivial task however it would become extremely tedious to determine the output of the system by hand if you were dealing with thousands or millions of input samples. This is where computers come in extremely handy since they can easily multiply millions of numbers every second. To give you an idea of how computers do this then take a look at this video which implements the following Octave/Matlab code that determines the output of the amplifier system for a large sequence of numbers.

Another way of representing a discrete system is by using what's referred to as a signal flow diagram. The signal flow diagram is a graphical representation of a discrete system which illustrates how the system behaves. The figure below shows the signal flow diagram for the amplifier system. The signal flow diagram contains a multiplier operator which multiplies any input by a factor of 2.



Example 2 – Moving average filter

Another discrete system that is relatively straightforward to appreciate is a *two-tap moving average filter*. This system takes the average of two consecutive input samples and passes the result to the output of the system.

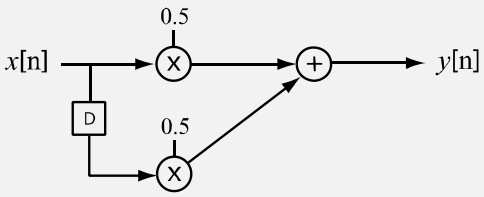


The moving average filter is very useful as it can mitigate fluctuations that may occur when data is gathered. For example consider the situation where you are measuring a river level which is affected by a local tide. These measurements will fluctuate as the water sloshes around the meter. By taking the average of two consecutive measurements the effect of the sloshing movement of the water will be reduced (take a look at the plots at the end of this subsection to see the effect of a moving average filter).

The two-tap moving average system can be represented mathematically by the following difference equation:

$$y[n] = 0.5x[n] + 0.5x[n-1]$$

Once again if you were to solve this difference equation for different values of n you would determine the output of this system for any input $x[n]$. I would encourage you to do this for yourself and notice that when you solve for $n = 0$ that you will have an $x[-1]$ term in the equation; this $x[-1]$ term is not explicitly defined so you assume that it is zero. Also, if you were to solve for $n = 6$ using the input shown in the figure above you should get a value of 0.5 for $y[6]$ since $x[6]$ is not explicitly defined. Take some time to make sure you understand the contents of this paragraph.

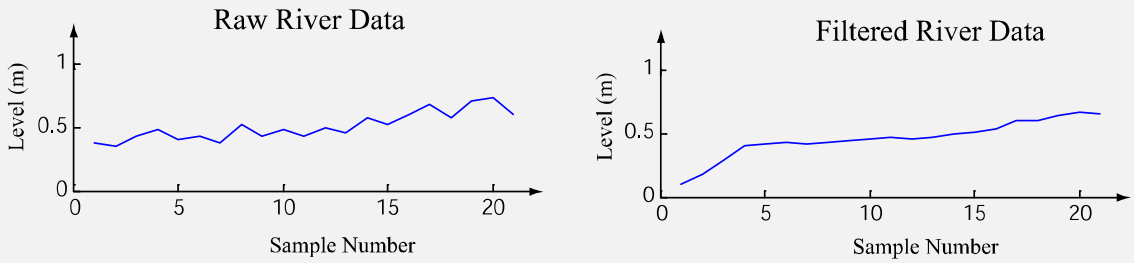


The signal flow diagram of the two-tap moving average is shown to the left and it contains two multipliers, one adder and one delay operator (the square with a D inside it). The behaviour of the delay operator is most easily explained by way of example so check out pzdsp.com/vid19.

Moving average filters can average more than just two consecutive samples and can take the average of any number of consecutive samples. As an example, a four-tap moving average filter is applied to the following data sequence which represents a river level (in meters) taken at 20 minute intervals:

[.375 .35 .425 .475 .4 .375 .525 .425 .475 .425 .5 .45 .575 .525 .6 .675 .575 .7 .725 .6]

This raw data is plotted to the left in the figure below and you can see that the readings fluctuate significantly. The plot to the right shows the result of passing the data through a four-tap moving average which takes the average over 4 consecutive samples. You see that the fluctuations in the readings have been smoothed out by the filter.

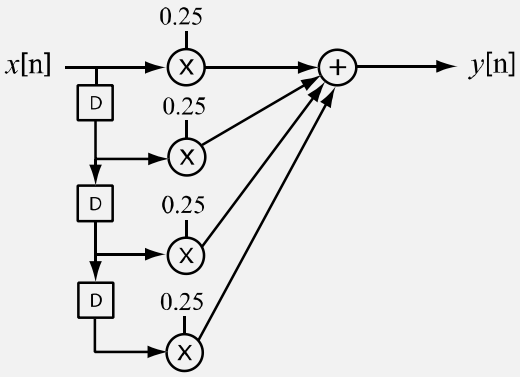


The difference equation of a four-tap moving average filter is given by:

$$y[n] = 0.25x[n] + 0.25x[n-1] + 0.25x[n-2] + 0.25x[n-3]$$

You will notice that the first four three samples of the filtered river data have a relatively steep slope. This is because the first three outputs ($y[0]$, $y[1]$, and $y[2]$) are all calculated using values of $x[n]$ that are undefined and therefore assumed to be zero. This has the affect of skewing the output of the filter and DSP practitioners need to be acutely aware of this issue when filtering data.

The signal flow diagram is shown to the right.



Example 3 – Temperature model

This example is not as straightforward as the other two examples and don't be too concerned if the application of the example doesn't make complete sense. The main reason I included this example is to show that the output of a system can be *fed back*

into a system i.e. in this system the output is dependent upon previous output samples and not just input samples.

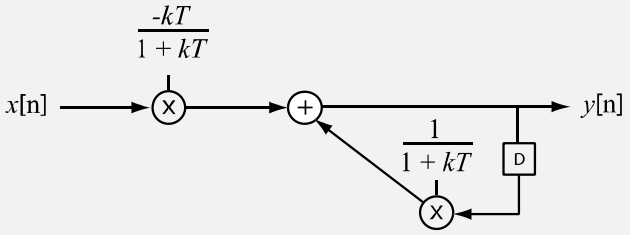
The discrete system given by the difference equation below allows you to determine (approximately) how the temperature of a glass of water would change if it was placed in a cool fridge. Hopefully you can appreciate that the temperature of the glass of water would gradually fall until it was the same temperature as the fridge.

The model takes the initial temperature difference between the water and the fridge as an input $x[n]$ and gives the change in temperature relative to the initial temperature over time as an output $y[n]$. Since the temperature of the glass of water will eventually match the temperature of the fridge then y[n] will eventually reach a near constant value, and be in *steady-state*.

$$y[n] = \left\{\frac{-kT}{1+kT}\right\}x[n] + \left\{\frac{1}{1+kT}\right\}y[n-1]$$

This model was derived using Newton's Law of Cooling and the constant k is the referred to as the cooling constant and is dependent upon the amount of water in the glass and the insulation provided by the glass containing the water. The T variable is the sampling interval used by the system and the product kT should be no more than .01 to ensure the model is reasonably accurate.

The constant k often determined through experimentation and once found you can use the model to predict how the temperature of a glass of water would change over time for any initial temperature difference and any fridge temperature (once the water remains a liquid! Also you would need to cover the glass to prevent any evaporation for high temperatures!).



As an example let's try to model how the temperature of a glass of water will change if it was initially 15 °C and then placed in a refrigerator of 4 °C. The difference is therefore 11 °C and the input will be a sequence of sample values that are all values of 11:

$$x[n] = [11\ \ 11\ 11\ 11\ 11 \ldots \ldots 11]$$

In this example we'll set the cooling constant k to being a value of 0.01. Since we are synthesising the input signal we can use any value for T that we like; the smaller the value of T the more accurate the model will be, however if T is very small we'd have to perform a lot of calculations before the output would reach a steady-state. A reasonable value for T in this case is 1.

The difference equation is therefore:

$$y[n] = -0.0099x[n] + 0.9901y[n-1]$$

To determine the output of the system you need to solve this difference equation for different values of *n*. Solving for n =0 gives:

$$y[0] = -0.0099x[0] + 0.9901y[-1]$$

Substituting *x*[0] for 11 and *y*[-1] for 0 (since *y*[-1] has not yet been defined) gives:

$$y[0] = -0.0099(11) = -0.1089$$

Solving for *n* = 1 gives

$$y[1] = -0.0099x[1] + 0.9901y[0]$$

Substituting *x*[1] for 11 and *y*[0] for -0.1089 gives

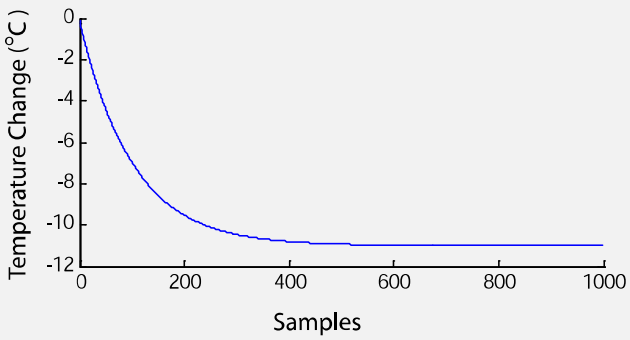
$$y[1] = -0.0099(11) + 0.9901(-0.1089) = -0.2167$$

All of the remaining output samples can be determined in a similar manner which is possible but somewhat tedious. In practice a computer would be used to determine the output samples and you can see how to use Matlab or Octave to do this in the next section.

The table to the left below shows the first 7 output samples of the discrete system model and you should make sure that you can work them out by hand to ensure you understand the process. The plot to the right shows a plot of 1000 output samples and you should notice that the output reaches what's known as *steady-state* after about 500 samples, which corresponds to 500 seconds in this example since the sampling interval is 1 second. Remember that the output represents the change from the initial input value

so the *steady-state* reading of -11 °C means that the temperature of the glass of water eventually reaches 4 °C (initial value of 15 °C less 11 °C).

n	*x*[*n*]	*y*[*n*]
0	11	-0.1089
1	11	-0.2167
2	11	-0.3235
3	11	-0.4292
4	11	-0.5339
5	11	-0.6375
6	11	-0.7401



Implementing discrete systems using Matlab/Octave

Determining the output of discrete systems (implementing a discrete system) can be a tedious task when you are dealing with lots of samples. Luckily computers can perform the calculations for us extremely quickly and accurately. Both Matlab and Octave are used very frequently for this purpose and have built-in commands/functions that make it a very straight forward process; however these functions require the discrete systems to be described in terms of *b* and *a* coefficients.

Once you develop an understanding of *b* and *a* coefficients you will be able to quickly implement and analyse discrete systems using built-in Matlab/Octave functions.

Perhaps the quickest way to understand how to determine a system's *b* and *a* coefficients is to run through a few of examples. The table below shows five discrete systems in their difference equation form in the left column and their corresponding *b* and *a* coefficient representation in the right. I'll describe the conversion process in the following paragraphs but you might find it easier to understand by looking at the following video pzdsp.com/vid20

$\boldsymbol{y[n] = 3x[n] + 2x[n-1]}$	*a* = [1] *b* = [3 2]
$\boldsymbol{y[n] = 0.5x[n] - x[n-1] + 2x[n-2]}$	*a* = [1] *b* = [0.5 -1 2]
$\boldsymbol{y[n] = -x[n] + x[n-3]}$	*a* = [1] *b* = [-1 0 0 1]
$\boldsymbol{y[n] = 2x[n] + 0.3y[n-1]}$	*a* = [1 -0.3] *b* = [2]
$\boldsymbol{y[n] = 0.1x[n] + x[n-2] - 0.2y[n-2]}$	*a* = [1 0 0.2] *b* = [0.1 0 1]

The mappings of the *b* and *a* coefficient vectors to the coefficients of the difference equation for each of the examples shown in the table above is illustrated in the paragraphs below. While referring to each of the illustrations you should note that the *a* coefficients are all associated with *y* terms in the difference equation and the *b* coefficients are all associated with *x* terms.

Also note that the first element of the *a* coefficient vector is associated with *y*[*n*]; the second element (if there is one) is associated with *y*[*n*-1]; the third element (if there is one) is associated with *y*[*n*-2]; etc.

Similarly the first element of the *b* coefficient vector is associated with *x*[*n*]; the second element (if there is one) is associated with *x*[*n*-1]; the third element (if there is one) is associated with *x*[*n*-2]; etc.

Example 1

$$y[n] = 3x[n] + 2x[n-1]$$

Rewritten as:

$$1y[n] = 3x[n] + 2x[n-1]$$

a = [1]
b = [3 2]

Example 2

$$y[n] = 0.5x[n] - x[n-1] + 2x[n-2]$$

Rewritten as:

$$1y[n] = 0.5x[n] - 1x[n-1] + 2x[n-2]$$

a = [1]
b = [0.5 -1 2]

Example 3

$$y[n] = -x[n] + x[n-3]$$

Rewritten as:

$$1y[n] = -1x[n] + 0x[n-1] + 0x[n-2] + 1x[n-3]$$

a = [1]
b = [-1 0 0 1]

Example 4

$$y[n] = 2x[n] + 0.3y[n-1]$$

Rewritten as:

$$1y[n] - 0.3y[n-1] = 2x[n]$$

a = [1 -0.3]
b = [2]

Example 5

$$y[n] = 0.1x[n] + x[n-2] - 0.2y[n-2]$$

Rewritten as:

$$1y[n] + 0y[n-1] + 0.2y[n-2] = 0.1x[n] + 0x[n-1] + 1x[n-2]$$

a = [1 0 0.2]
b = [0.1 0 1]

Implementation examples

Once you have the *b* and *a* coefficients of a system then the implementation is a trivial task using built-in Matlab/Octave functions. This section runs through the examples shown earlier (amplifier, moving average and system model) that you should try out yourself. If you don't have Matlab or Octave installed on your computer then you can use an online version http://octave-online.net

An amplifier system which amplifies by a factor of 2 is described by the following difference equation:

$$y[n] = 2x[n]$$

The *b* and *a* coefficients just have one value each i.e. a = [1] and b= [2].

Use the code below to determine the output of this system after applying an input $x[n]$ which is given by

$$x[n] = [2 \;\; 1.1 \;\; 3.2 \;\; 0.3 \; 1 \;\; 0]$$

```
x = [2  1.1  3.2  0.3 1  0];
b = 2;
a = 1;
y = filter(b, a, x)
```

A 4-tap moving-average filter is given by the following difference equation

$$y[n] = 0.25x[n] + 0.25x[n-1] + 0.25x[n-2] + 0.25x[n-3]$$

The *b* and *a* coefficients are therefore: a= [1] and b = [0.25 0.25 0.25 0.25].

Use the code below to determine and plot the output after passing the following river level signal though the system:

[.375 .35 .425 .475 .4 .375 .525 .425 .475 .425 .5 .45 .575 .525 .6 .675 .575 .7 .725 .6]

```
x = [.375 .35 .425 .475 .4 .375 .525 .425 .475 .425 .5 .45 .575 .525 .6 .675 .575 .7 .725 .6];
b = [0.25 0.25 0.25 0.25];
a = 1;
y = filter(b, a, x)
plot(y)
```

A model of the change in temperature of a glass of water placed in a fridge is given by (where the cooling constant is 0.01 and the model operates on data sampled at 1Hz):

$$y[n] = -0.0099x[n] + 0.9901y[n-1]$$

where x[n] is the initial difference in temperature between the water and the fridge and $y[n]$ is the change in the water temperature from its initial temperature.

Using the earlier example of the water temperature being initially 15 °C and then placed in a refrigerator of 4 °C then $x[n]$ will be constantly 11 °C.

Use the following code to model this setup for 1000 seconds:

```
x = ones(1, 1000)*11; %1000 seconds
b = [ -0.0099 ];
a = [ 1  -0.9901];
y = filter(b, a , x);
plot(y)
xlabel('Samples')
ylabel('Temperature Change')
figure
plot(y+15)
xlabel('Samples')
ylabel('Temperature of water')
```

Additional resources

An online quiz can be found at pzdsp.com/quiz3

Some related videos:

pzdsp.com/vid21 – System frequency response

pzdsp.com/vid22 – Magnitude response using plots

pzdsp.com/vid23 – Filtering signals using discrete systems