An introduction to the principles and practise of Digital Signal Processing

By Dr. Chris Bore
This book is for use by the purchaser only

Bores Signal Processing, Fordwater, Pond Road, Woking, Surrey, GU22 OJZ
tel:01483 740138 Fax:01483 740136 Email:bores@bores.com http://www.bores.com
Introduction

This monograph covers the essentials of Digital Signal Processing, including filters and frequency analysis using the Fourier Transform. We emphasise practical considerations, making this particularly useful for engineers who have to work with DSP rather than study it as an academic specialisation. A graphical approach to explanation helps to develop intuitive understanding of the operations involved and their effects; and helps in getting a feel for what is reasonable. Some mathematical equations are presented, but there is no derivation or proof and advanced maths is not required to follow and understand the text. The notes are intended for engineers wishing for an introduction to, or refresher course in, the principles of DSP and the practical ways it can be used, including its limitations.

Our aims are that by working through these notes you should be able to:

- understand how sampled data systems work and their limitations
- use and understand correlation and convolution
- design and use digital filters
- apply the Fourier Transform and interpret the results

Bores Signal Processing retain copyright in this material, which may not be copied for any purpose. You may print one copy for your personal use.

This monograph is on of a series on DSP which include:

- Introduction to DSP
- Introduction to FIR filters
- Introduction to IIR filters
- Introduction to the Philips TriMedia
- Introduction to the Motorola DSP5630x
- Introduction to DSP processors
Basics

Digital Signal Processing (DSP) is used in a wide variety of application, and it is hard to find a good definition that is general.

We can start with dictionary definitions of the words:

- **Digital**: operation by the use of discrete signals to represent data in the form of numbers
- **Signal**: a variable parameter by which information is conveyed
- **Processing**: to perform operations on data according to programmed instructions

Which leads us to a simple definition of Digital Signal Processing as

‘changing or analysing information which is measured as discrete sequences of numbers’

Note two unique features of Digital Signal Processing as opposed to plain old ordinary digital processing:

1. Signals come from the real world - this intimate connection with the real worlds leads to many unique needs such as the need to react in real time and a need to measure signals and convert them to digital numbers
2. Signals are discrete - which means the information in between discrete samples is lost.
The advantages of DSP are common to many digital systems and include:

**Versatility:**
- digital systems can be reprogrammed for other applications
- digital systems can be ported to different hardware

**Repeatability:**
- digital systems can be easily duplicated
- digital systems do not depend on strict component tolerances
- digital system responses do not drift with temperature

**Simplicity:**
- some things can be done more easily digitally than with analogue systems

DSP is used in a very wide variety of applications but most share some common features:

- they use a lot of maths (multiplying and adding signals)
- they deal with signals that come from the real world
- they require a response in a certain time

Where general purpose DSP processors are concerned, most applications deal with signal frequencies that in the audio range.
DSP Markets

The market for DSP processors has grown rapidly over many years. It is usually the fastest growing sector of the electronics market, and has shown consistent growth averaging at least 20% annually for the past six years.

The main market for DSP processors is in telecoms: principally in mobile telephones, but also in the telecoms infrastructure. Consumer applications including audio, are increasing, with many vendors hoping that video will be a large market quite soon. The range of other applications is very broad, but the actual use is rather small.
As more applications are pushed to become digital, it is expected that the DSP market will continue to grow strongly. This is quite likely to happen whether digital is a good idea in itself or not: for any DSP offers the potential to reduce bandwidth requirement and so to allow governments to sell more channels and take the resulting increased licence fees.

A very important area for DSP is data compression. Compression methods for audio and video tend to use filters and FFTs, which are classic DSP operations and are very well suited by DSP processors. Because so many DSP applications condense down to filters and FFTs, most DSP processors are designed to do these two specialised jobs very efficiently. A good understanding of basic DSP operations, and of the ways in which DSP processors are designed to perform these operations efficiently, is essential to make good use of the technology.
Converting analogue signals

Most DSP applications deal with analogue signals. The analogue signal has to be converted to digital form. The analogue signal - a continuous variable defined with infinite precision - is converted to a discrete sequence of measured values which are represented digitally.

Information is lost in converting from analogue to digital, due to:

- inaccuracies in the measurement
- uncertainty in timing
- limits on the duration of the measurement

These effects are called quantisation errors.
The continuous analogue signal has to be held before it can be sampled, otherwise the signal would be changing during the measurement.

Only after it has been held can the signal be measured, and the measurement converted to a digital value.

The sampling results in a discrete set of digital numbers that represent measurements of the signal - usually taken at equal intervals of time.

Note: The sampling takes place after the hold. This means that we can sometimes use a slower Analogue to Digital Converter (ADC) than might seem required at first sight. The hold circuit must act fast - fast enough that the signal is not changing during the time the circularity is acquiring the signal value - but the ADC has all the time that the signal is held to make its conversion.
We don't know what we don't measure

In the process of measuring the signal, some information is lost.

In general, we can make no assumptions about what the signal did when we were not measuring it. So, for example, we do not know what the signal did in between the times when we actually held and sampled it: nor do we know what it did before we started to sample is, or after we finished. In addition, there will be measurement inaccuracies due to the analogue circuitry and limitations in the number of bits to represent the measurement.

The effects of the lost information fall into three categories:

- aliasing
- frequency resolution
- quantisation error

Sometimes we may have a prior knowledge of the signal, or be able to make some assumptions that will let us reconstruct the lost information: but we need to be aware that we are making these assumptions. Sometimes the assumptions which seem intuitively most reasonable, are not in fact valid.
Aliasing

We only sample the signal at intervals, and we don’t know what happened between the samples. A
crude example is to consider a ‘glitch’ that happened to fall between adjacent samples. Since we don’t
measure it, we have no way of knowing the glitch was there at all.

In a less obvious case, we might have signal components that are varying rapidly in between samples. Again,
we could not track these rapid inter-sample variations.

We must sample fast enough to see the most rapid changes in the signal. Sometimes we may have some prior
knowledge of the signal, or be able to make some assumptions about how the signal behaves in between
samples.

If we do not sample fast enough, we cannot track completely the most rapid changes in the signal. Some
higher frequencies can be incorrectly interpreted as lower ones.
In the diagram, the high frequency signal is sampled just under twice every cycle. The result is that each sample is taken at a slightly later part of the cycle. If we draw a smooth connecting line between the samples, the resulting curve looks like a lower frequency. This is called ‘aliasing’ because one frequency looks like another.

Note: The problem of aliasing is that we cannot tell what frequency we have - a high frequency looks like a low one so we cannot tell the two apart. But sometimes we may have some prior knowledge of the signal, or be able to make some assumptions about how the signal behaves in between samples, that will allow us to tell unambiguously what we have.
In the diagram, the high frequency signal is sampled twice every cycle. If we draw a smooth connecting line in between the samples, the resulting curve looks like the original signal. But if the samples happen to fall at the zero crossings, we would see no signal at all - this is why the sampling theorem demands we sample faster than twice the highest signal frequency. This avoids aliasing.

The highest signal frequency allowed for a given sample rate is called the Nyquist frequency. Actually, Nyquist says that we have to sample faster than the signal bandwidth, not the highest frequency, but this leads us into multirate signal processing which is a more advanced subject.
Anti aliasing

Nyquist showed that to distinguish unambiguously between all signal frequency component we must sample at least twice the frequency of the highest frequency component. To avoid aliasing, we simply filter out all the highest frequency component before sampling.

\[ \text{High frequency components} \]

\[ \text{through a low pass filter} \]

\[ \text{are removed} \]

This simple brute force method avoids the problem of aliasing, but it does remove information. If the signal had high frequency components, we cannot know anything about them.

Note: Antialias filters must be analogue - it is too late once you have done the sampling.

Although Nyquist showed that provided we sample at least at the highest signal frequency, we have all the information needed to reconstruct the signal, the sampling theorem does not say the samples will look like the signal.
A high frequency signal may still look wrong but can be reconstructed.

The diagram shows a high frequency sine wave that is nevertheless sampled fast enough according to Nyquist’s sampling theorem - just more than twice per cycle. When straight lines are drawn between the samples, the signal’s frequency is indeed evident - but it looks as though the signal is amplitude modulated. This effect arises because each sample is taken at a slightly earlier part of the cycle. Unlike aliasing, the effect does not change the apparent signal frequency. The answer lies in the fact that the sampling theorem says there is enough information to reconstruct the signal - and the correct reconstruction is not just to draw straight lines between samples.
Signal reconstruction

The signal is properly reconstructed from the samples by low pass filtering: the low pass filter should be the same as the original antialias filter.

A sampled signal

must be low pass filtered

to reconstruct the original

The construction filter interpolates between the samples to make a smoothly varying analogue signal. The explanation lies in the shape of the reconstruction filter’s impulse response

the filter's impulse response has a \( \sin(x)/x \) shape

The impulse response of the reconstruction filter has a classic ‘\( \sin(x)/x \)’ shape. The stimulus fed to this filter is the series of discrete impulses which are the samples. Every time an impulse hits the filter, we get ‘ringing’ - and it is the superposition of all these peaking rings that reconstructs the proper signal. If the signal contains frequency components that are close to the Nyquist, then the reconstruction filter has to be very sharp indeed. This means it will have a very long impulse response - and so the long ‘memory’ needed to fill the signal even in region of the low amplitude samples.
Frequency resolution

We only sample the signal for a certain time. We cannot see slow changes in the signal if we don’t wait long enough.

In fact we must sample for long enough to detect not only low frequencies in the signal, but also small differences between frequencies. The length of time for which we are prepared to sample the signal determines our ability to resolve adjacent frequencies - the frequency resolution.

We must sample for at least one complete cycle of the lowest frequency we want to resolve.

We can see that we face a forced compromise. We must sample fast to avoid aliasing and for a long time to achieve a good frequency resolution. But sampling fast, for a long time, means that we will have a lot of samples - and lots of samples means lots of computation, for which we don’t generally have time. We have to compromise between resolving frequency components of the signal and being able to see high frequencies.
Quantisation

When the signal is converted to digital form, the precision is limited by the number of bits available.

The diagram shows an analogue signal which is then converted to a digital representation - in this case, with 8 bit precision. The smoothly varying analogue signal can only be represented as a 'stepped' waveform, due to the limited precision. Sadly, the errors introduced by digitisation are both non linear and signal dependent. Non linear means we cannot calculate their effects using normal maths. Signal dependent means the errors are coherent and so cannot be reproduced by simple means.

This is a common problem in DSP. The errors due to limited precision (i.e., word length) are non linear (hence incalculable) and signal dependent (hence coherent). Both are bad news and mean that we cannot really calculate how a DSP algorithm will perform in limited precision - the only reliable way is to implement it and test it against signals of the type expected. The non linearity can also lead to instability with IIR filters.
Uncertainty in the clock timing leads to errors in the sampled signal.

Timing error leads to value error

an accurate clock

leads to accurate values

an error in the clock

translates to error in the values

The diagram shows an analogue signal which is held on the rising edge of a clock signal. If the clock edge occurs at a different time than expected, the signal will be held at the wrong value. Sadly, the errors introduced by timing error are both non linear and signal dependent.
A real DSP system suffers from three sources of error due to limited word length in the measurement and processing of the signal: limited precision due to word length when the analogue signal is converted to digital form; errors in arithmetic due to limited precision within the processor itself; and limited precision due to word length when the digital samples are converted back to analogue form. The word length of hardware used for DSP processing determines the available precision and dynamic range.

These errors are often called 'quantisation error'. The effects of quantisation error are in fact both non linear and signal dependent. Non linear means we cannot calculate their effects using normal maths. Signal dependent means that even if we could calculate their effect, we would have to do so separately for every type of signal we expect. A simple way to get an idea of the effects of limited word length is to model each of the sources of quantisation error as it were a source of random noise.
The model of quantisation as injections of random noise is helpful in gaining an idea of the effects, but it is not actually accurate, especially for systems with feedback like IIR filters.

---

The effect of quantisation error is often similar to an injection of random noise. The diagram shows the spectrum calculated from a pure tone. The top plot shows the spectrum with high precision (double precision floating point): the bottom plot shows the spectrum when the sine wave is quantised to 16 bits - the effect looks very like low level random noise. The signal to noise ratio is affected by the number of bits in the data format, and by whether the data is fixed point or floating point.
Correlation

A simple moving average is just the sum of the last few values:

\[ y[n] = \sum_{k=0}^{k=N-1} x[n-k] \]

Correlation is a weighted moving average:

\[ y[n] = \sum_{k=0}^{k=N-1} c[k] \cdot x[n-k] \]

With correlation, one signal weights the previous values. For example if we were tracking the stock exchange average over the past five trading days, we might give more weight to the previous day’s figures than to those from four days ago:

The diagram shows how a single point of the correlation function is calculated:

- one signal is shifted with respect to the other
- the amount of the shift is the position of the correlation function point to be calculated
- each element of the one signal is multiplied by the corresponding element of the other
- the area under the resulting curve is integrated

This process yields one point of the resulting function: it is repeated with the shift increased by one each time, to calculate successive points.

Correlation requires a lot of calculations. If one signal is of length M and the other is of length N, then we need (M*N) multiplications to calculate the whole correlation function.

Note: Really, we want to multiply and then accumulate the result - this is typical of DSP operations and is called a 'multiply/accumulate' operation. It is the reason that DSP processors can do multiplications and additions in parallel - called a MAC (for Multiply/ACcumulate).
Correlation as a measure of similarity

Correlation is a maximum when two signals are similar in shape and are in phase (or ‘unshifted’ with respect to each other).

Correlation is a measure of the similarity between two signals as a function of time shift between them.

- If two signals are similar and unshifted... their product is all positive.
- But as the shift increases... parts of it become negative...
- and the correlation shows where the signals are similar and unshifted

The diagram shows two similar signals. When two signals are similar in shape and unshifted with respect to each other, their product is all positive. This is like constructive interference between waves, where the peaks add and the troughs subtract to emphasise each other. The area under this curve gives the value of the correlation function at point zero, and this is a large value.

As one signal is shifted with respect to the other, the signals go out of phase - the peaks no longer coincide, so the product can have negative going parts. This is like destructive interference, where the troughs cancel the peaks. The area under this curve gives the value of the correlation function at the value of the shift. The negative going parts of the curve now cancel some of the positive going parts, so the correlation function is smaller.

The largest value of the correlation function shows when the two signals were similar in shape and unshifted with respect to each other (or ‘in phase’). The breadth of the correlation function - where it has a significant value - shows for how long the signals remain similar.
Auto correlation to distinguish between signals

Correlating a signal with itself is called auto correlation. Different sorts of signals have distinctly different auto correlation functions. We can use these differences to tell signals apart.

The diagram shows three different types of signal:

1. Random noise is defined to be uncorrelated - this means it is only similar to itself with no shift at all. Even a shift of one sample either way means there is no correlation at all, so the correlation function of random noise with itself is a single spike at shift zero.

2. Periodic signals go in and out of phase as one is shifted with respect to the other. So they will show strong correlation at any shift where the peaks coincide. The auto correlation function of a periodic signal is itself a periodic signal, with a period the same as that of the original signal.

3. Short signals can only be similar to themselves for small values of shift, so their auto correlation functions are short.

The three types of signals have easily recognizable auto correlation functions, which allow us to identify and distinguish these signals using auto correlation.

Note: Exploiting the differences in auto correlation functions to distinguish signals is an example of a common trick in DSP - using some signal transformation whose results are very different from signals of different type. If the signals can be more easily distinguished after the transformation, then obviously that transformation can be used to better separate those signals.
Auto correlation to extract a signal from noise

Auto correlation (correlating a signal with itself) can be used to extract a signal from noise.

The diagram shows how the signal can be extracted from the noise.

<table>
<thead>
<tr>
<th>Random noise has a 'spike' auto correlation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A sine wave has a periodic auto correlation function</td>
</tr>
</tbody>
</table>

- Random noise has a distinctive 'spike' auto correlation function
- A sine wave has a periodic auto correlation function

So the auto correlation function of a noisy sine wave is a periodic function with a single spike which contains all the noise power.

The separation of a signal from noise using auto correlation works because the auto correlation function of the noise is easily distinguished from that of the signal.

**Note:** Auto correlation used like this is an example of common DSP operations which rely on the noise 'collapsing' to some narrow region - leaving the signal to stand out clearly. In this case the noise collapses to a single spike. The opposite behavior can also be used - for example the frequency spectrum of random noise is spread out evenly whereas the frequency spectrum of a periodic signal is a spike or a series of spikes - but again the signal can be separated easily from the noise by its different behavior after transformation.
Cross correlation to detect a known signal

Cross correlation (correlating a signal with another) can be used to detect and locate a known reference signal in noise.

- A radar or sonar 'chirp' signal...
- Reflection may be buried in noise...
- But correlating with the 'chirp' reference clearly reveals when the echo comes

The diagram shows how the signal can be located within the noise.

- A copy of the known reference signal is correlated with the unknown signal
- The correlation will be high when the reference is similar to the unknown signal
- The peak value indicates when the reference signal occurs
- The value shows the degree of confidence that the reference signal is detected

Note: Typical applications for cross correlation used to detect a known reference signal occur in radar and sonar. A short, easily identified, signal is sent out towards a target: the echo contains the emitted signal and the range of the target can be determined by the time for the reflection to return - which is indicated by the delay of the peak in the cross correlation. Looking for a known reference helps to avoid spurious detections.
Cross correlation to identify a signal

Cross correlation (correlating a signal with another) can be used to identify a signal by comparison with a library of known reference signals. The diagram shows how the unknown signal can be identified.

- The chirp of a nightingale...
- correlates strongly with another nightingale...
- but weakly with a dove...
- or a heron...
- which is one way sonar can identify enemy ships or illegally fishing vessels

- a copy of a known reference signal is correlated with the unknown signal
- the correlation will be high if the reference is similar to the unknown signal
- the unknown signal is correlated with a number of known reference functions
- a large value for correlation shows the degree of similarity to the reference
- the largest value for correlation is the most likely match

Note: Cross correlation is one way in which sonar can identify different types of vessel. Each vessel has a unique sonar 'signature': the sonar system has a library of pre-recorded sonar signals from different vessels. An unknown sonar echo is correlated with a library of reference echoes: the largest correlation is the most likely match. Aside from identifying enemy ships, this can be used for instance to identify vessels which have been caught fishing illegally.
Convolution

Convolution is a weighted moving average with one signal flipped back to front:

\[ y[n] = \sum_{k=0}^{k=N-1} c[k] * x[n + k] \]

the equation is the same as for correlation, except that the sign of \( k \) is positive rather than negative: this corresponds to taking that signal and flipping it back to front.

The diagram shows how a single point of the convolution function is calculated:

- first, one signal is flipped back to front
- then, one signal is shifted with respect to the other
- the amount of the shift is the position of the convolution function point to be calculated
- each element of one signal is multiplied by the corresponding element of the other
- the area under the resulting curve is integrated

Convolution requires a lot of calculations. If one signal is of length \( M \) and the other is of length \( N \), then we need \((N*M)\) multiplications to calculate the whole convolution function.
**Convolution to smooth a signal**

Since convolution is a weighted moving average, convolving a signal with a smooth weighting function can be used to smooth a signal.

![Convolution of a noisy sine wave...](image)

![with a smoothing function...](image)

![smoothes the signal](image)

The diagram shows how a noisy sine wave can be smoothed by convolving with a rectangular smoothing function - this is just a moving average. This smoothing is one sort of filtering. The smoothing property leads to the use of convolution for digital filtering.

**Note:** Convolution is preferred to correlation for filtering because of the way frequency spectra of the two signals interact. Convolving two signals is equivalent to multiplying the frequency spectra of the two signals together - which is easily understood, and is what we mean by filtering. Correlation is equivalent to multiplying the complex conjugate of the frequency spectrum of one signal by the frequency spectrum of the other which is not at all easily understood.
Similarity between correlation and convolution

Correlation is a weighted moving average, while convolution is a weighted moving average with one signal flipped back to front, so correlation and convolutions are the same except for the flip.

\[
r[n] = \sum x[k] \cdot y[n + k]
\]

\[
r[n] = \sum x[k] \cdot y[n - k]
\]

If one signal is symmetric, then flipping it back to front does not change it: so correlation and convolution become identical.

Note: Because correlation and convolution are identical if the weighting function is symmetric, filters with a symmetric filter function are in fact just weighted moving averages just like correlation and so are easy to understand.
Jean Baptiste Fourier showed that any signal or waveform could be made up just by adding together a series of pure tones (sine waves) with appropriate amplitude and phase.

This is a rather startling theory, if you think about it. It means, for instance, that by simply turning on a number of sine wave generators we could sit back and enjoy a Beethoven symphony.

Of course we would have to use a very large number of sine wave generators, and we would have to turn them on at the time of the Big Bang and leave them on until the heat death of the universe.

The diagram shows how a square wave can be made up by adding together pure sine waves at the harmonics of the fundamental frequency.
Any signal can be made up by adding together the correct sine waves with the appropriate amplitude and phase. Finding these amplitudes and phase against frequency is called a frequency spectrum.

**Note:** Fourier’s theorem assumes we add sine waves of infinite duration - which is of course going to prove impossible in practise, since we do not have infinite time available. Much of the interest (that is, the difficulty) of Digital Signal Processing arises from the need to cope with the real world where infinite signals are not available and the convenient assumptions used to derive elegant solutions and theories begin to break down. The real behavior of limited signals is much more intriguing and much less easy to model with simple mathematics.
The Fourier transform

The Fourier transform is an equation to calculate the frequency, amplitude and phase of each sine wave needed to make up any given signal.

The Fourier transform (FT) is a mathematical formula using integrals over continuous, infinitely long functions.

\[ H(f) = \int_{-\infty}^{+\infty} h(t) e^{-j2\pi ft} \, dt \]

Where \( j \) is the square root of minus one (defined as a number whose sole property is that its square is minus one).

Note: People can get upset at the square root of -1. Any number, when squared, yields a positive result: so at first sight the square of -1 seems like a nonsense. In fact, it is just a device to make the maths work in a way that models the real world. \( j \) has no properties at all except that, when squared, it yields -1. There is no point worrying about what this ‘means’ - it doesn’t mean anything. \( j \) is only worth bothering about when it is squared - when it is -1. If it isn’t squared, it doesn’t mean anything: it is just waiting to be squared some time. \( j \) is called an ‘imaginary’ number - presumably because it is such a weird concept that nobody can imagine what it is.
The sine wave comes from the 'e' term:

\[ e^{-j2\pi ft} = \cos(\theta) - j \sin(\theta) \]

so that the integral is over sine and cosine waves.

Sampled signals are not continuous but consist of samples, and are not infinitely long: so the Discrete Fourier Transform (DFT) is a discrete numerical equivalent using finite sums instead of integrals.

\[ H(f) = \sum_{k=-\infty}^{k=+\infty} h[k] * e^{-j2\pi(k/\Delta f)} \]

The Fast Fourier Transform (FFT) is just a computationally efficient way to calculate the Discrete Fourier Transform - the equation being solved is the same even though the computational method is clever.

Note: Like [correlations](#) and [convolutions](#), the DFT and the FFT involve a lot of multiplying and then accumulating the result - this is typical of DSP operations and is called a 'Multiply/ACcumulate (MAC) operation. It is the reason that DSP chips can do multiplications and additions in parallel. Interestingly, the Fast Fourier Transform is designed to minimise multiplications which are, on conventional processors, slow operations: with DSP chips the multiply is as fast as the addition so the problem may not be the same.
The Fourier Transform can be used to analyse any signal into its frequency components.

The Fourier Transform can be used to analyse any signal into its frequency components.

- a recording of speech...
- can be analysed to show the frequency spectrum

Every signal has a frequency spectrum.
- the signal defines the spectrum
- the spectrum defines the signal

We can move back and forth between the time domain and the frequency domain without losing information.

Note: This statement is true mathematically, but it is quite incorrect in any practical sense - since we will lose information due to errors in the calculation, or due to deliberately missing out some information that we can't measure or can't compute. But the basic idea is a good one when visualising time signals and their frequency spectra.

Understanding the relation between time and frequency domains is useful.
- some signals are easier to visualize in the frequency domain
- some signals are easier to visualize in the time domain
- some signals take less information to define in the time domain
- some signals take less information to define in the frequency domain

For example a sine wave takes a lot of information to define accurately in the time domain: but in the frequency domain we only need three data - the frequency, amplitude and phase.
Fast convolution

Convolution is the same as multiplying the frequency spectra of two signals.

Convolution by multiplying frequency spectra can take advantage of the Fast Fourier Transform - which is a computationally efficient algorithm. So this can be faster than convolution in the time domain, and is called Fast Convolution.

Note: Because convolution is the same as multiplying the frequency spectra, it can be thought of as modifying spectrum of one signal to attenuate, enhance or select certain components. This is what is meant by ‘filtering’ - an operation that is defined by its filtering effect in the frequency, not the time domain.
Short time Fourier transforms

The Fourier transform assumes the signal is analysed over all time - an infinite duration.

This means that there can be no concept of time in the frequency domain, and so no concept of a frequency changing with time. Mathematically, frequency and time are ‘orthogonal’ - you cannot mix one with the other. But we can easily understand that some signals do have frequency components that change with time. A piano tune, for example, consists of different notes played at different times: or speech can be heard as having pitch that rises and falls over time.

The **Short Time Fourier Transform** (STFT) tries to evaluate the way frequency content changes with time. The diagram shows how the Short Time Fourier Transform works:

- The frequency content of speech changes over time.
- By chopping it up into pieces...
- ...and taking the Fourier transform of each piece...
- ...we can see how frequency content changes with time.
We can see that:

- the signal is shopped up into short pieces
- and the Fourier transform is taken of each piece

Each frequency spectrum shows the frequency content during a short time, and so the successive spectra show the evolution of frequency content with time. The spectra can be plotted one behind the other in a 'waterfall' diagram as shown.

**Note:** The Short Time Fourier Transform involves accepting a contradiction in terms because frequency only has a meaning if we use infinitely long sine waves - and so we cannot apply Fourier Transforms to short pieces of a signal. We can work either in frequency, or in time, but not in both: so there can be no concept of frequency 'changing over time'. However, it is a jolly useful concept so we use it anyway.
**Short Signals**

There is a direct relation between a signal's duration in time and the width of its frequency spectrum: short signals have broad spectra; long signals have narrow spectra.

One can see intuitively that to make a short signal requires a broad frequency spectrum. Since the short signal has to be built up from infinite sine waves, the frequency components have to cancel each other outside a narrow range. The shorter the signal, the more components are needed to cancel each other out and so the broader the frequency spectrum.
two pure sine wave components would have a spectrum with two sharp lines - one for each component.

but if they are short signals each line is broadened so the spectrum shows two broad 'blurred' lines.

and the combined spectrum shows a hump with a dip - if the components were too close the dip might disappear.

**Note:** The relation between length of a signal and the breadth of its spectrum is the reason why a sampled signal is limited in frequency resolution by the total time over which we measure it. As a 'rule of thumb' spectral width is $1/T$ where $T$ is the total length of the signal in time. This is the separation at which two adjacent components can just about be distinguished.
The presumption of periodicity

The Fourier Transform works on signals of infinite duration. But if we only measure the signal for a short time, we cannot know what happened to the signal before and after we measured it. The Fourier Transform has to make an assumption about what happened to the signal before and after we measured it.

The Fourier Transform assumes that any signal can be made by adding a series of sine waves of infinite duration. Sine waves are periodic signals. So the Fourier Transform works as if the data, too, were periodic for all time.

This presumption of periodicity is built into the Fourier Transform and, since most of DSP maths is derived from the Fourier Transform theory, it also pervades pretty much all of DSP.
Sometimes the presumption turns out to be correct.

If it happens that the signal is periodic - and that an integral number of cycles fit into the total duration of the measurement - then the Fourier Transform assumes the signal repeats, the end of one signal segment connects smoothly with the beginning of the next, and the assumed signal happens to be exactly the same as the actual signal. This is either pure luck, or possibly careful adjustment of a frequency generator or arrangement of an experiment.
More often than not, the Fourier Transform’s assumption will prove misleading.

If not quite an integral number of cycles fit into the total duration of the measurement then when the Fourier Transform assumes the signal repeats, the end of one signal segment does not connect smoothly with the beginning of the next - the assumed signal is similar to the actual signal, but has little ‘glitches’ at regular intervals. The ‘glitches’ are short signals, so they have a broad spectrum: and this broadening is imposed on the frequency spectrum of the actual signal.
If the period exactly fits the measurement time, the frequency spectrum is correct. If the period does not match the measurement time, the frequency spectrum is incorrect - it is broadened. The size of the glitch depends on when the first measurement occurred in the cycle, so the broadening will change if the measurement is repeated.

**Note:** A sine wave 'should' have a spectrum with a single sharp line. But in practice, if measured by a spectrum analyser, the spectrum will be a broad line - with the sides flapping up and down like Batman's cloak as the glitches move. When we see a perfect single line spectrum - for example in the charts sometimes provided with analogue to digital converter chips - this has been obtained by tuning the signal frequency carefully so that the period exactly fits the measurement time and the frequency spectrum is the 'best' obtainable.
Windowing

The problem with short signals is that they have ends. The ends introduce ‘glitches’ - so an obvious approach is to reduce these glitches. The glitches can be reduced by shaping the signal so that its ends match more smoothly.

Since we can't assume anything about the signal, we need a way to make any signal's ends connect smoothly to each other when repeated. One way to do this is to multiply the signal by a 'window' function.

Windowing suppresses glitches and makes the sampled data more nearly periodic

The easiest way to make sure the ends of a signal match is to force them to be zero at the ends: that way, their value is necessarily the same, regardless of the signal’s shape. Actually, we also want to make sure that the signal is going in the right direction at the ends to match up smoothly. The easiest way to do this is to make sure neither end is going anywhere - that is, the slope of the signal at its ends should also be zero. Put mathematically, a window function has the property that its value and all its derivatives are zero at the ends.
Multiplying by a window function (called windowing) suppresses glitches and so avoids the broadening of the frequency spectrum caused by the glitches.

Note: As with many DSP operations, windowing tampers with the signal. We should always be suspicious of making hidden assumptions and of imposing those assumptions on signals.

Multiplying by a window function suppresses glitches at the ends, and so sharpens the frequency spectrum of a signal. But we should always remember that it in fact applies distortion to the signal. Windowing forces the signal to have a narrower frequency spectrum, whether the original signal had a narrow spectrum or not: so it should only be used with great care.
The diagram shows what can happen if we apply a window function without proper thought.

The transient response really does have a broad frequency spectrum - but windowing forces it to look more as if it had a narrow frequency spectrum instead. Worse than this, the window function has attenuated the signal at the point where it was largest - so has suppressed a large part of the signal power. This means the overall signal to noise ratio has been reduced.

Applying a window function narrows the frequency spectrum at the expense of distortion and signal to noise ratio. Many window functions have been designed to trade off frequency resolution against signal to noise and distortion. Choice among them depends on knowledge of the signal and what you want to do with it.

Note: The choice and specification of window functions is a specialised subject in its own right. Many window functions have been designed, each aiming to achieve the best trade off between conflicting demands. If you are interested in learning more about this subject, I refer you to our ‘advanced’ BORES monograph ‘Introduction to FFT window functions’.
Wavelets and other transforms

Fourier's theorem assumes we add sine waves of infinite duration.

As a consequence, the Fourier Transform is good at representing signals which are long and periodic. But the Fourier Transform has problems when used with signals which are short, and not periodic.

Other transforms are possible - fitting the data with different sets of functions than sine waves. The trick is, to find a transform whose base set of functions look like the signal with which we are dealing.

The diagram shows a signal that is not a long, periodic signal but rather a periodic signal with a decay over a short time. This is not very well matched by the Fourier Transform's infinite sine waves. But it might be better matched by a different set of functions - say, decaying sine waves. Such functions are called 'wavelets' and can be used in the 'wavelet transform'.
Other transforms are possible - for example the Walsh transform matches a signal to a set of square waves. Mathematically, any set of ‘orthogonal’ functions can be used as a basis for a transform. Generally, as with wavelets, we have some difficulty in understanding intuitively what these transforms map - the Fourier Transform happens to fit rather well with what we think we know already about ‘frequency’ so it seems to be easier to interpret.

Note: the wavelet transform cannot really be used to measure frequency, because frequency only has meaning when applied to infinite sine waves. But, as with the Short Time Fourier Transform, we are always willing to stretch a point in order to gain a useful tool.

Note: The Fourier Transform's great popularity derives not really from any special mathematical merit, but from the simple fact that two chaps (Cooley and Tukey) managed to write an efficient algorithm to implement it - called the Fast Fourier Transform (FFT). And now there are lots of FFT programs around for all sorts of processors, so it is likely the FFT will remain the most popular method for many years because of its excellent support.
Filters

Filtering is a process of selecting, or suppressing, certain frequency components of a signal.

Filtering is a frequency selective process
- a filter selects, suppresses, or modifies certain frequency components of the signal
- either to suppress noise, or to shape the spectrum

in a noisy sine wave...

...the noise is spread out in frequency...

...and can be removed by a narrow filter...

...leaving mainly the signal power...

...which is a cleaner sine wave

Filtering is often, though not always, done to suppress noise. It depends on the signal’s frequency spectrum being different from that of the noise. The diagram shows how a noisy sine wave viewed as a time domain signal cannot be clearly distinguished from the noise. But when viewed as a frequency spectrum, the sine wave shows as a single clear peak while the noise power is spread over a broad frequency spectrum. The noisy sine wave may be 'cleaned up' by selecting only a range of frequencies that include signal frequency components but exclude much of the noise.

Filters may also be used to shape the frequency spectrum of a signal: for instance to correct for the characteristics of a transmission circuit such as a telephone line, or to apply more bass to an audio recording.
Note: Digital filters and coffee filters have some similarities beyond just the same name. A coffee filter allows small particles to pass while trapping the larger grains. A digital filter does a similar thing, but with more subtlety. The digital filter allows to pass certain frequency components of the signal: in this it is similar to the coffee filter, with frequency standing in for particle size. But the digital filter can be more subtle than simply trapping or allowing through: it can attenuate, or suppress, each frequency components by a desired amount. This allows a digital filter to shape the frequency spectrum of the signal.
Digital filter specifications

Digital filters can be more subtly defined than analogue filters, and so are defined differently.

Whereas analogue filters are specified in terms of their '3dB point' and their 'roll off', digital filters are specified in terms of desired attenuation, and permitted deviations from the desired value:

- **passband**: the band of frequency components that are allowed to pass
- **stopband**: the band of frequency components that are suppressed
- **passband ripple**: the maximum variation in the passband
- **stopband attenuation**: the minimum attenuation in the stopband

The passband need not necessarily extend to the 3 dB point: for example, if passband ripple is specified as 0.1 dB, then the passband only extends to a point at which attenuation has increased to 0.1 dB. Between the passband and the stopband lies a transition band where the filter's shape may be unspecified. Stopband attenuation is formally specified as the attenuation to the top of the first side lobe of the filter's frequency response. Digital filters can also have an 'arbitrary response': meaning, the attenuation is specified at certain chosen frequencies or bands.
Digital filters are also characterised by their response to an impulse - a signal consisting of a single value followed by zeroes. The impulse response is an indication of how long the filter takes to settle into a steady state: it is also an indication of the filter's stability - an impulse response that continues oscillating in the long term indicates the filter may be prone to instability.

The impulse response is an indication of the filter's stability
Filtering in the frequency domain

Filtering can be done directly by operating on the signal’s frequency spectrum.

The diagram shows how a noisy sine wave can be cleaned up by operating directly upon its frequency spectrum to select only a range of frequencies that include signal frequency components but exclude much of the noise:

- the noisy sine wave contains narrow band signal plus broad band noise
- the frequency spectrum is calculated
- a range outside the signal's frequency components is suppressed
- the time domain signal is calculated from the frequency spectrum
- the resulting signal (shown in the time domain again) looks much cleaner
Filtering in the frequency domain is efficient, because every calculated sample of the filtered signal takes account of all the input samples.

Filtering in the frequency domain is sometimes called 'acausal' filtering because it seems to violate the laws of cause and effect. Because the frequency spectrum contains information about the whole of the signal - for all time values - samples early in the output take account of input values that are late in the signal, and so can be thought of as still to happen. The frequency domain filter 'looks ahead' to see what the signal is going to do, and so seems to look into the future.

In practice this cause and effect argument is nonsense - all it means is we delayed a little until the whole signal had been received before starting the filter calculation - so filtering directly in the frequency domain is perfectly permissible and in fact often the best method. It is often used in image processing.

There are good reasons why we might not be able to filter in the frequency domain- for example we might not be able to afford to wait for future samples - but this should not make us ignore the possibility of frequency domain filtering, which is very often the best method. It is often used in image processing, or certain types of experiment where the data necessarily comes in bursts, such as NMR or infrared spectroscopy.
**Digital filter equation**

Output from a digital filter is made up from previous inputs and previous outputs, using the operation of convolution.

\[
y[n] = \sum_{k=0}^{k=N-1} b[k] \cdot x[n - k] + \sum_{i=1}^{i=M-1} a[i] \cdot y[n - i]
\]

The first sum is over previous inputs: the second is over previous outputs. The terms \(c[k]\) and \(d[j]\) are called the filter ‘coefficients’ and represent the weighting functions in the convolutions.

The filter can be drawn as a block diagram:

The left hand side of the diagram shows the direct path, involving previous inputs: the right hand side shows the feedback path involving previous outputs.

**Note:** The filter block diagram helps to assess hardware requirements for a digital filter. For each coefficient we require: a multiply; and add; and a delay (\(z^{-1}\)). It is always worth drawing block diagrams using realistic elements - for example, adders with only two inputs rather than generalised 'sum units' which may have many inputs - so that the hardware needs become obvious right from the start.
It is worth noting that it is relatively easy to write a simple C programme to implement a general digital filter.

```c
in_sum = 0.0; out_sum = 0.0;
for (k = 0; k < N; k++)
    y[n] = in_sum + b[k] * x[n-k];
for (i = 1; i < M; i++)
    out_sum = out_sum + a[i] * y[n - i];
y[n] = in_sum + out_sum;
```

However, such a simple programme is misleading since it involves feedback - and on any system with finite precision arithmetic the feedback of errors may lead to catastrophic failure.

**Note:** Although it is not all that difficult to design and program digital filters if there are no errors in the computation, it can be very hard indeed to implement satisfactorily in the presence of errors. Since all DSP chips use finite precision - usually fixed point or at best single precision floating point arithmetic - the problem of how to minimise error propagation is central to digital filter design and implementation. This is a specialised subject in its own right. If you are interested to learn more about practical design and implementation of digital filters using finite precision arithmetic, I refer you to our 'advanced' BORES monograph 'IIR Filters' which treats the subject in some depth.
The **impulse response** of a filter defines its response to any sampled signal.

A sampled signal is a series of impulses - one for each sample of the signal - each of height equal to the value of the signal at that time. Each impulse generates an impulse response, and (because the filter is considered to be a linear system) these impulse responses simply add together to create the output signal.
The impulse response is usually written as a function $h(k)$: so the output from a general linear digital filter can also be written as:

$$y[n] = \sum_{k=0}^{k=\infty} h[k] * x[n - k]$$

which looks similar to the original filter equation:

$$y[n] = \sum_{k=0}^{k=N-1} b[k] * x[n - k] + \sum_{i=1}^{i=M-1} a[i] * y[n - i]$$

until we notice that the impulse response form is a sum over an infinite number of terms - whereas the filter is over a finite number of terms. So the filter equation can be realised in practice, while the impulse response convolution may not be practicable: except in the special case where the impulse response is of finite duration (called a Finite Impulse Response filter).

**Note:** If the filter’s impulse response is indeed infinite, then it cannot be realised by the simple convolution - because we cannot compute an infinite number of terms. But the filter may be able to be realised using the filter equation, which requires only a finite number of terms. This gives us a hint that it may be more efficient to use the full filter equation - including the feedback sum - than to use a finite impulse response form.
Frequency response

Since filtering is a frequency selective process, the important thing about a digital filter is its frequency response. The filter’s frequency response can be calculated from its impulse response (which, in turn, can be calculated from its filter equation). The trick is to recognise that the frequency response is a measure of what the filter does to each frequency component of an entire spectrum. Each frequency component can be represented by a sine wave - or, for mathematical convenience, a complex exponential:

\[ x[n] = e^{j2\pi n(f\Delta)} = \cos(2\pi n(f\Delta)) + j\sin(2\pi n(f\Delta)) \]

These terms are just substituted as inputs to the filter expressed in terms of its impulse response:

\[ y[n] = \sum_{k=0}^{k=\infty} h[k] \ast e^{j2\pi(n-k)(f\Delta)} \]

and, since \( n \) is a constant, this reduces to:

\[ y[n] = e^{j2\pi n(f\Delta)} \sum_{k=0}^{k=\infty} h[k] \ast e^{-j2\pi k(f\Delta)} \]

So, the response to a complex exponential input at frequency \( f \) is:

\[ y[n] = H(\omega) e^{j\omega n} \]

where:

\[ H(\omega) = \sum_{k=0}^{k=\infty} h[k] \ast e^{-j\omega n} = \sum_{k=0}^{k=\infty} h[k] \ast e^{-j2\pi k(f\Delta)} \]
The function describes the change in magnitude and phase at the frequency - so it is the frequency response of the filter.

In fact, the frequency response can be calculated back to relate to the filter equation and to the filter coefficients as:

\[
H(f) = \frac{\sum_{k=0}^{k=N-1} b[k] e^{-j2\pi k f \Delta}}{1 - \sum_{i=1}^{i=M-1} a[i] e^{-j2\pi i f \Delta}}
\]

The frequency response \( H(f) \) is a continuous function, although the filter equation is discrete.

**Note:** The filter frequency response is a ‘ratio of polynomials’. A polynomial is a sum of terms raised to various powers - such as: \( a_1 x + a_2 x^2 + a_3 x^3 + \ldots \). The top sum: \( \Sigma b[k] e^{j2\pi f \Delta} \) is a polynomial, as is the bottom: and since one divides the other this is a ratio of polynomials. Luckily for us, we are not too bothered with the mathematics of this equation, because what we want to do is to calculate the required filter coefficients, given the frequency response, rather than to derive or calculate the frequency response. It does not hurt to recognise the equation, though, and especially to note where the filter coefficients occur in it. Sometimes filters may be specified as a ‘ratio of polynomials’ in which case we will just have to get out the text books.

The filter coefficients \( a[k] \) and \( b[i] \) appear as coefficients for the polynomials in the denominator (top) and numerator (bottom) of the ratio of polynomials. So if we could find the polynomial coefficients that would also give us the filter coefficients.
Whilst it is nice to be able to calculate the frequency response given the filter coefficients, when designing a digital filter we want to do the inverse operation: that is, to calculate the filter coefficients having first defined the desired frequency response. So we are faced with an inverse problem.

Sadly, there is no general inverse solution to the frequency response equation.

To make matters worse, we want to impose an additional constraint on acceptable solutions. Usually, we are designing digital filters with the idea that they will be implemented on some piece of hardware. This means we usually want to design a filter that meets the requirement but which requires the least possible amount of computation: that is, using the smallest number of coefficients. So we are faced with an insoluble inverse problem, on which we wish to impose additional constraints.

This is why digital filter design is more an art than a science: the art of finding an acceptable compromise between conflicting constraints.

Note: If we have a powerful computer and time to take a coffee break while the filter calculates, the small number of coefficients may not be important - but this is a pretty sloppy way to work and would be more of an academic exercise than a piece of engineering. It would also, frankly, lack the real interest which arises from having to meet practical considerations and design economical solutions. There is a growing fashion in DSP to throw silicon at problems instead of thinking about them, so efficiency is less admired now than it used to be - no doubt we shall soon see GHz DSP chips which will run hot enough for you to make the coffee to drink during the break while you wait for your inefficient filter to execute.
FIR filters

It is much easier to approach the problem of calculating filter coefficients if we simplify the filter equation so that we only have to deal with previous inputs (that is, we exclude the possibility of feedback). The filter equation is then simplified.

\[ y[n] = \sum_{k=0}^{k=N-1} c[k] \ast x[n - k] \]

If such a filter is subjected to an impulse (a signal consisting of one value followed by zeroes) then its output must necessarily become zero after the impulse has run through the summation. So the impulse response of such a filter must necessarily be finite in duration. Such a filter is called a Finite Impulse Response filter or FIR filter.
And the frequency response is also a simpler equation, because all the bottom half of the ratio (which depends only on previous outputs) goes away.

\[ H(f) = \sum_{k=0}^{N-1} c[k] \cdot e^{-j2\pi f \Lambda} \]

It so happens that this frequency response is just the Fourier transform of the filter coefficients: and the inverse solution to a Fourier transform is well known - it is simply the inverse Fourier transform. So the coefficients for an FIR filter can be calculated simply by taking the inverse Fourier transform of the desired frequency response. This lets us develop a simple recipe for calculating FIR filter coefficients:

- decide upon the desired frequency response
- calculate the inverse Fourier transform
- use the result as the filter coefficients

This is quite an acceptable way to calculate filter coefficients for an FIR filter.

But...

- the inverse Fourier Transform samples the continuous desired frequency response.
- to define a sharp filter needs closely spaced frequency samples - so a lot of them
- so the inverse Fourier transform will give us a lot of filter coefficients
- but we don't want a lot of filter coefficients

So the simple recipe is unlikely to give the best result - best in terms of least number of coefficients which yield a filter that meets the specification.
The window method of FIR filter design

The problem of calculating FIR filter coefficients simply by taking the inverse Fourier Transform of the desired frequency response is that we can do a better job by noting that:

- the filter coefficients for an FIR filter are also the impulse response of the filter
- the impulse response of an FIR filter dies away to zero
- so many of the later filter coefficients for an FIR filter are small
- and perhaps we can throw away these small values as being less important

The idea here is to truncate the filter coefficients to some acceptably small number: hoping that by discarding the later (and so smaller) coefficients we will not ruin the frequency response completely.

Here is a better recipe for calculating FIR filter coefficients based on throwing away the small ones:

- pretend we don't mind lots of filter coefficients
- specify the desired frequency response using lots of samples
- calculate the inverse Fourier transform
- this gives us a lot of filter coefficients
- so truncate the filter coefficients to give us less

We are obliged to take the Fourier transform of the truncated set of coefficients to see if its frequency response still matches our requirement. But...

...truncating the filter coefficients means we have a truncated signal. And a truncated signal has a broad frequency spectrum.
So, truncating the filter coefficients means the filter’s frequency response can only be defined coarsely. At first sight, this seems to leave us no better off than before.
The problem with truncating the filter coefficients to an acceptably small number is that this leaves ‘glitches’ at the ends and so broadens the frequency response. Luckily, we already know a way to sharpen up the frequency spectrum of a truncated signal, by applying a window function. So after truncation, we can apply a window function to sharpen up the filter's frequency response.

So here is an even better recipe for calculating FIR filter coefficients:

- pretend we don't mind lots of filter coefficients
- specify the desired frequency response using lots of samples
- calculate the inverse Fourier transform
- this gives us a lot of filter coefficients
- so truncate the filter coefficients to give us less
- apply a window function to sharpen up the filter's frequency response
This is called the ‘Window’ method of FIR filter design. The precise shape of the filter’s frequency response is now determined by the window function as well as by the specified bands: and most window functions have a fixed attenuation to the top of their first sidelobe, so the choice of window function is central to FIR filter design by this method.

The attenuation to this first sidelobe is 20.96 dB. No matter how many taps you use, this is the fixed limit for a rectangular window.

You cannot improve a fixed window function’s attenuation by throwing coefficients at it.
Some window functions

With the Window method of filter design, the window function determines the shape of the frequency response (for example, the shape of the band edges). Each window has its own fixed shape and maximum attenuation. For example, if you need an attenuation of 20 dB or less, then a rectangle window is acceptable. If you need 43 dB you are forced to choose the Hanning window, and so on.
One might be tempted to ask, why not always use one of the ‘better’ window functions? Sadly, the better window functions need more filter coefficients before their shape can be adequately defined. So if you need only 25 dB of attenuation you should choose a triangle window function which will give you this attenuation: the Hamming window, for example, would give you more attenuation but require more filter coefficients to be adequately defined - and so would be wasteful of computer power.

**Note:** The definition of digital filter attenuation as to the top of the first side lobe is significant here in a practical way. You may notice for example that the rectangle window gives about 21 dB attenuation to the top of its first side lobe but progressively more with further side lobes. If we relaxed the definition, so that our stopband began at the top of, say, the third sidelobe, we could get more attenuation. However, it is generally true that if we do so, then a better choice of window function would do the same job with less coefficients. So the definition is a good practical guide to suitability.

The art of FIR filter design by the window method lies in choosing the window function which meets your requirement with the minimum number of filter coefficients.

**Note:** You may notice that if you want an attenuation of 30 dB you are in trouble: the triangle window is not good enough but the Hanning window is too good (and so uses more coefficients than you need). The Kaiser window function is unique in that its shape is variable. A variable parameter defines the shape, so the Kaiser window is unique in being able to match precisely the attenuation you require without overperforming.
Equiripple (Parks/McLellan) filters

The window method does not correspond to any known form of optimisation. In fact it can be shown that the window method is not optimal - by which we mean, it does not produce the lowest possible number of filter coefficients that just meets the requirement.

The art of FIR filter design by the window method lies in choosing the window function which meets your requirement with the minimum number of filter coefficients. If the window method design is not good enough we have two choices: to use another window function and try again; or to do something clever.

Note: The explanation of Remez Exchange given here is not mathematically correct or complete, but since we are trying to get an idea of what is going on, and not trying to duplicate the thinking of geniuses, it is worth going through anyway. The idea is to come up with some sort of intuitive understanding of what is going on, not to derive and prove the maths.

The Remez Exchange algorithm is something clever. It uses a mathematical optimisation method. Using the window method to design a filter we might proceed manually as follows:

- choose a window function that we think will do
- calculate the filter coefficients
- check the actual filter's frequency response against the design goal
- if it over performs, reduce the number of filter coefficients or relax the design
- try again until we find the filter with the lowest number of filter coefficients possible
In a way, this is what the Remez Exchange algorithm does automatically until it finds the filter that just meets the specification with the lowest possible number of filter coefficients. Actually, the Remez Exchange algorithm never really calculates the frequency response: but it does keep comparing the actual with the design goal.

The Remez/Parks McLellan method produces a filter which just meets the specification without over performing. Many of the window method designs actually perform better as you move further away from the passband, or better than specified within the passband: this is wasted performance, and means they are using more filter coefficients than they need. The Remez/Parks McLellan method performs just as well as the specification but no better, with the minimum number of coefficients. So Remez/Parks McLellan designs have equal ripple - up to specification but no more - in passband and stopband. This is why they are called equiripple designs.

Note: Some books say that Parks/McLellan filters are the ‘best’. This is misleading: Parks/McLellan give you the best approximation in the Least Mean Square sense - and sometimes you may want the shape of a window design. There are no hard and fast rules in digital filter design.
IIR filters

Output from a general digital filter is made up from previous inputs and previous outputs, using the operation of convolution.

\[
y[n] = \sum_{k=0}^{k=N-1} b[k] \ast x[n-k] + \sum_{i=1}^{i=M-1} a[i] \ast y[n-i]
\]

In the general case we have to include the right hand side of the equation - which is the feedback path. The block diagram also shows the feedback path.

Feedback means that, even if the input becomes zero and stays there, the output may go on changing since it feeds on itself. Specifically, If such a filter is subjected to an impulse (a signal consisting of one value followed by zeroes) then its output need not necessarily become zero after the impulse has run through the summation. So the impulse response of such a filter can be infinite in duration. Such a filter is called an Infinite Impulse Response filter or IIR filter.
The impulse response need not necessarily be infinite: if it were, the filter would be unstable. For most practical filters, the impulse response will die away to a negligibly small level. One might argue that mathematically the response can go on for ever, getting smaller and smaller: but in a digital world once a level gets below one bit it might as well be zero. The Infinite Impulse Response refers to the ability of the filter to have an infinite impulse response and does not imply that it necessarily will have one: it serves as a warning that this type of filter is prone to feedback and instability. It is also worth noting that even if the impulse response decays, this does not necessarily mean the filter will be stable in the face of all possible inputs.

Feedback means that an infinite impulse response filter can have an impulse response that is infinite...

...but for most practical applications, the impulse response will die away...

...otherwise the filter would be unstable

Note: The possibility of feedback in IIR filters has a drastic effect when these are implemented using finite precision arithmetic (which means, in practice, whenever they are implemented on a computer). This is because the arithmetic errors, however small they may be, are also fed back and so can grow alarmingly. These errors are non linear (so their effect cannot be calculated using normal linear maths); signal dependent; affected by the processor arithmetic; and dependent on choices the programmer can make. Implementing IIR filters so as to minimise error propagation is a specialised subject in its own right: If you are interested to learn more I refer you to our ‘advanced’ monograph ‘IIR filters’
IIR filter design
As we saw earlier, the frequency response for a general linear filter is a ratio of polynomials.

\[
H(f) = \frac{\sum_{k=0}^{k=N-1} b[k] * e^{-j2\pi k(f\Delta)}}{1 - \sum_{i=1}^{i=M-1} a[i] * e^{-j2\pi i(f\Delta)}}
\]

Again as before, while it is nice to calculate the frequency response given the filter coefficients, the real problem is to calculate the coefficients given the desired frequency response. So we are faced with an inverse problem.

Unfortunately for us, there is no known general inverse solution to the filter frequency response equation: it is an insoluble problem.

If you look at text books on digital filtering, or use a digital filter design package, you will find a rather surprising fact: all digital filters are based on analogue filter prototypes - Butterworth, Elliptic, Bessel, Chebychev. Direct digital IIR filter design is rarely used, for one very simple reason:

- nobody knows how to do it
While it is easy to calculate the filter's frequency response, given the filter coefficients, the inverse problem - calculating the filter coefficients from the desired frequency response - is so far an insoluble problem. Not many text books admit this.

Because we do not know how to design digital IIR filters, we have to fall back on analogue filter designs (for which the mathematics is well understood) and then transform these designs into a sampled data representation.

They key to transforming an analogue filter into a digital one is to note that the analogue filter's impulse response defines it just as completely as does its frequency response: and it is quite easy to calculate the impulse response of an analogue filter using ordinary mathematics.

Here is a recipe for designing an IIR digital filter:

- decide upon the desired frequency response
- design an appropriate analogue filter
- calculate the impulse response of this analogue filter
- sample the analogue filter's impulse response
- use the result as the filter coefficients

This is called the method of impulse invariance.

However, we are now faced with a sampled data system - and because we can't deal with an infinite impulse response we have to truncate the analogue filter’s response. So impulse invariant filters will suffer from aliasing and other problems.
The bilinear transform

IIR filters designed using the method of impulse invariance suffer from aliasing, because they are derived from a sampled version of the analogue filter’s impulse response. This means that if the analogue filter has significant values at and beyond the Nyquist aliasing limit, those parts of the frequency response will be aliased back and will distort the desired frequency response.

The problem with impulse invariant filter design is that it specifies the filter in the time domain - and so it suffers from all the problems of sampled data systems in distorting the frequency response. Specifically, the infinitely long frequency axis is wrapped around again and again due to aliasing.

The bilinear transform method instead maps the frequency response of the analogue filter onto a sampled data frequency response. This effectively avoids aliasing by the simple expedient of squashing the analogue frequency axis at both ends so that it becomes finite - this is called frequency warping - and so that it fits entirely within the Nyquist limit.

The bilinear transform method preserves the values of the frequency response unchanged: but because of the frequency axis warping, it does change the frequencies where those values occur.

To minimise the effects of the bilinear transform’s frequency warping, the analogue filter has to be designed taking into account the frequency warping that will occur. This is called ‘prewarping’.

\[
\begin{align*}
\text{s} &= \frac{(1 - (1/z))}{(1 + (1/z))} \\
\text{f} &= \frac{\arctan(\pi f\Delta)}{\pi\Delta}
\end{align*}
\]

The bilinear transform ‘warps’ the frequency axis so it becomes finite.
Filter programming

The general digital filter equation involves two convolution sums: one operating on the previous inputs, the other operating on the previous outputs.

\[ y[n] = \sum_{k=0}^{k=N-1} b[k] \cdot x[n-k] + \sum_{i=1}^{i=M-1} a[i] \cdot y[n-i] \]

Such an equation is called a difference equation.

The difference equation can be written into a C program quite simply.

```c
in_sum = 0.0; out_sum = 0.0;
for (k = 0; k < N; k++)
    y[n] = in_sum + b[k] * x[n-k];
for (i = 1; i < M; i++)
    out_sum = out_sum + a[i] * y[n - i];
y[n] = in_sum + out_sum;
```

The program has two loops: one implements the first sum, operating on the previous inputs, while the other implements the second sum, operating on the previous outputs.

The arrays \( b[k] \) and \( a[k] \) are the filter coefficients. In the general case, for an IIR filter which has feedback, the filter coefficients are the polynomial coefficients in the equation for the frequency response.
For IIR filters there will be two sets of coefficients. Filter design programs will often (but not always) use the convention that the $b[k]$ coefficients are from the numerator (top), relating to the previous inputs; while the $a[k]$ coefficients are from the denominator (bottom), and are applied to the previous outputs.

For the simpler case of an FIR filter, there is no feedback path so the second sum disappears:

$$H(f) = \sum_{k=0}^{k=N-1} b[k] * e^{-j2\pi k(f\Delta)}$$

$$1 - \sum_{i=1}^{i=M-1} a[i] * e^{-j2\pi i(f\Delta)}$$

and the programme is even simpler:

```c
in_sum = 0.0;
for (k=0; k<N; k++)
    y[n] = in_sum[n] + b[k] * x[n-k];
y[n] = in_sum;
```

In this case there is only one set of filter coefficients.