3F3 – Digital Signal Processing (DSP)

Simon Godsill

www-sigproc.eng.cam.ac.uk/~sjg/teaching

Course Overview

- 12 Lectures
- Topics:
  - Digital Signal Processing
  - DFT, FFT
  - Digital Filters
  - Filter Design
  - Filter Implementation
  - Random signals
  - Optimal Filtering
  - Signal Modelling
- Books:
  - J.G. Proakis and D.G. Manolakis, Digital Signal Processing 3rd edition, Prentice-Hall.
  - Statistical digital signal processing and modeling -Monson H. Hayes –Wiley
- Some material adapted from courses by Dr. Malcolm Macleod, Prof. Peter Rayner and Dr. Arnaud Doucet

Digital Signal Processing - Introduction

- Digital signal processing (DSP) is the generic term for techniques such as filtering or spectrum analysis applied to digitally sampled signals.
- Recall from 1B Signal and Data Analysis that the procedure is as shown below:

- T is the sampling period
- $f_0 = 1/T$ is the sampling frequency
- Recall also that low-pass anti-aliasing filters must be applied before A/D and D/A conversion in order to remove distortion from frequency components higher than $f_0/2$ Hz (see later for revision of this).

• Digital signals are signals which are sampled in time (“discrete time”) and quantised.

• Mathematical analysis of inherently digital signals (e.g. sunspot data, tide data) was developed by Gauss (1800), Schuster (1896) and many others since.

• In 1948 A H Reeves proposed Pulse Code Modulation for digital **transmission** of signals.

• Digital **storage** of sampled analogue signals was used from the 50s, and is now common - DAT, CD etc.

• Electronic digital signal processing (DSP) was first extensively applied in geophysics (for oil-exploration) then military applications, and is now fundamental to communications, broadcasting, and most applications of signal and image processing.

There are many advantages in carrying out digital rather than analogue processing; among these are flexibility and repeatability.

The flexibility stems from the fact that system parameters are simply numbers stored in the processor. Thus for example, it is a trivial matter to change the cut-off frequency of a digital filter whereas a lumped element analogue filter would require a different set of passive components. Indeed the ease with which system parameters can be changed has led to many adaptive techniques whereby the system parameters are modified in real time according to some algorithm. Examples of this are adaptive equalisation of transmission systems, adaptive antenna arrays which automatically steer the nulls in the polar diagram onto interfering signals. Digital signal processing enables very Digital signal processing enables very complex linear and non-linear processes to be implemented which would not be feasible with analogue processing. For example it is difficult to envisage an analogue system which could be used to perform spatial filtering of an image to improve the signal to noise ratio.

DSP has been an active research area since the late 1960s but applications tended to be only in large and expensive systems or in non real-time where a general purpose computer could be used. However, the advent of d.s.p chips enable real-time processing to be performed at very low cost and already this technology is commonplace in domestic products.

Sampling Theorem (revision from 1B)

For a continuous time signal $g(t)$ with Fourier transform $G(\omega)$, the Fourier transform of the corresponding sampled signal $g_s(t)$, sampled at a rate $\omega_0 = 2\pi/T$ rad.s^{-1} is given by:

$$G_s(\omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} G(\omega - n\omega_0) \tag{1}$$

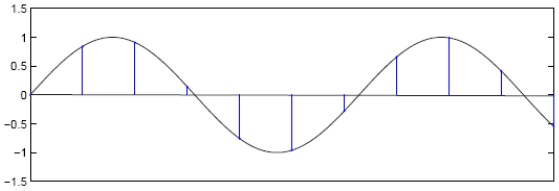
i.e. **The Fourier transform of the sampled signal is simply a summation of shifted versions of the original Fourier transform of the continuous signal** $G(\omega)$

Implications:

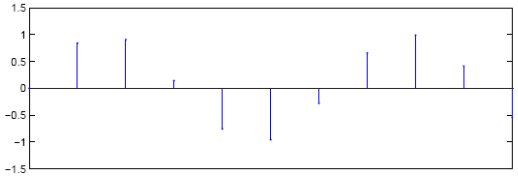
- Spectrum of sampled signal is always *periodic*.
- Can reconstruct an analogue signal *perfectly* from its digital samples provided its bandwidth is $< \omega_0/2$.
- For signals with bandwidth $\geq \omega_0/2$ we must pre-filter with an ideal lowpass filter having cut-off frequency $\omega_0/2$ to prevent *aliasing*
- Practical considerations - design of anti-aliasing filters with finite transition bandwidth, ...

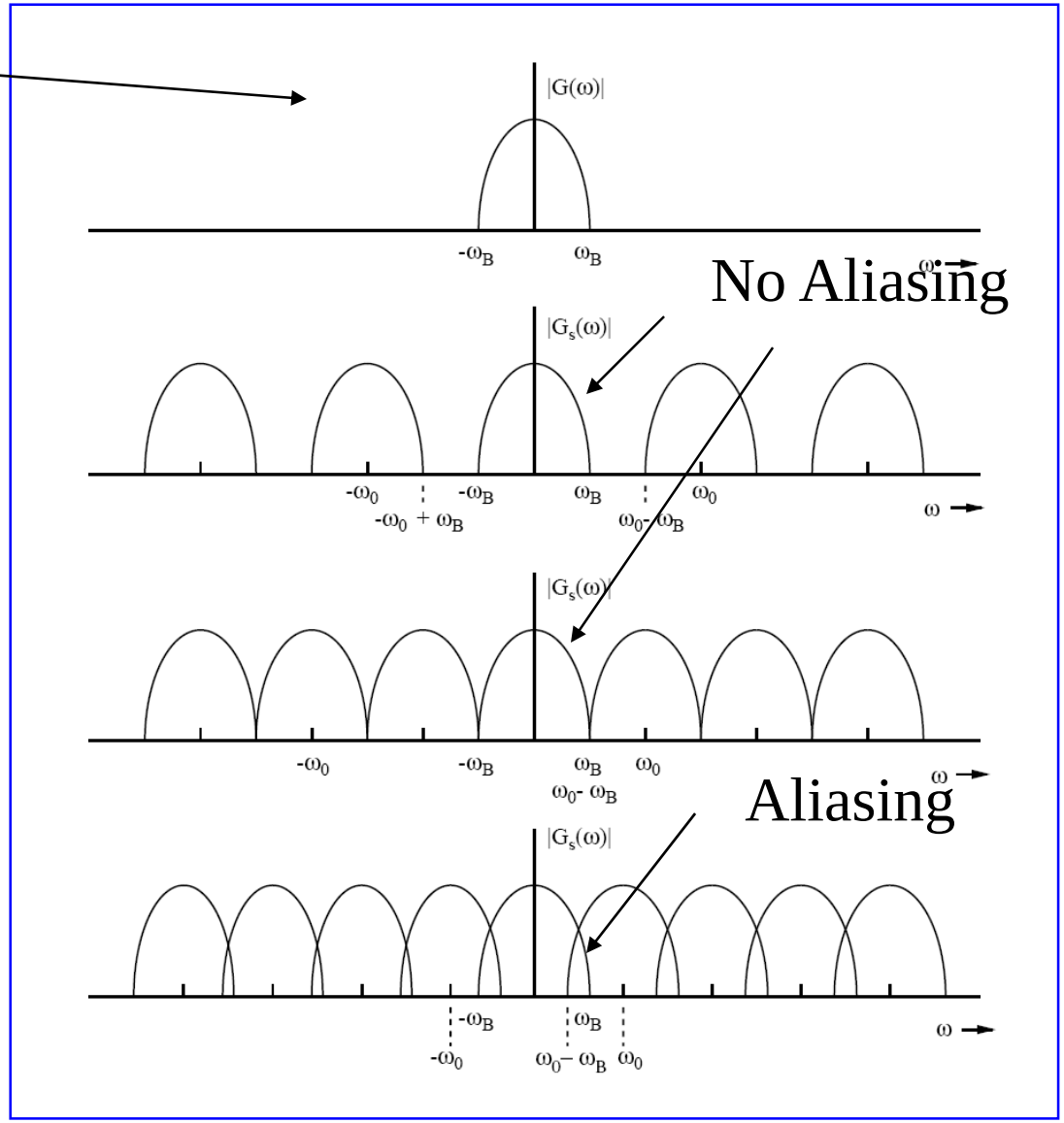
Sampled Signal Spectra:

Continuous signal g(t)



Sampled signal $g_s(t)$
(various values of ω_0)





Sketch of proof: [revision, see IB]

Define a mathematical representation of the sampled signal $g_s(t)$ using a train of δ-functions:

$$g_s(t) = g(t) \sum_{n=-\infty}^{\infty} \delta(t - nT)$$
$$= g(t)\delta_p(t)$$

Fourier series representation of periodic function $\delta_p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$:

$$\delta_p(t) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \mathrm{e}^{jn\omega_0 t}$$

where $\omega_0 = 2\pi/T$.

Hence:

$$g_s(t) = g(t)\frac{1}{T} \sum_{n=-\infty}^{\infty} \mathrm{e}^{jn\omega_0 t}$$
$$= \frac{1}{T} \sum_{n=-\infty}^{\infty} g(t)\mathrm{e}^{jn\omega_0 t}$$

Take Fourier Transform of $g_s(t)$:

$$G_s(\omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} G(\omega - n\omega_0)$$

Sampling Theorem: Summary

- Theorem shows us that we may represent a signal perfectly in the digital domain, provided the sampling rate is at least twice the maximum frequency component (`bandwidth') of the signal

- Denote the sampled values of a signal/function using the shorthand:

$$x_n = x(nT)$$

T is the sampling period
$f_0 = \frac{1}{T}$Hz is the sampling frequency, or sampling 'rate'
$\omega_0 = 2\pi f_0$rad/s is the sampling frequency in radians

The DFT and the FFT

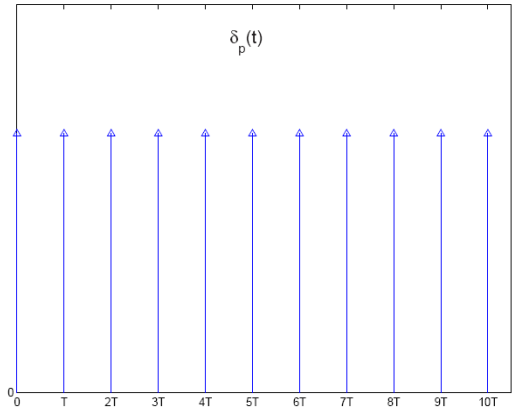
- The Discrete Fourier Transform is the standard way to transform a block of sampled data into the frequency domain (see IB)
- The Fast Fourier Transform (FFT) is a fast algorithm for implementation of the DFT
- The FFT revolutionised Digital Signal Processing. It is an elegant and highly effective algorithm that is still the building block used in many state-of-the-art algorithms in speech processing, communications, frequency estimation, …

The Discrete Time Fourier Transform (DTFT)

The DTFT is defined as the Fourier transform of the sampled signal. Define the sampled signal in the usual way:

$$g_s(t) = g(t) \sum_{p=-\infty}^{\infty} \delta(t - pT)$$

$\delta_p(t)$



Take Fourier transform directly

$$\begin{aligned} G_s(\omega) &= \int_{-\infty}^{\infty} g_s(t)\, e^{-j\omega t}\, dt \\ &= \int_{-\infty}^{\infty} g(t) \sum_{p=-\infty}^{\infty} \delta(t - pT)\, e^{-j\omega t}\, dt \\ &= \sum_{p=-\infty}^{\infty} g(pT) e^{-j\omega pT} = \sum_{p=-\infty}^{\infty} g_p e^{-j\omega pT} \end{aligned}$$

using the 'sifting' property of the δ-function to reach the last line.

Note that this expression, known as the DTFT, is a periodic function of frequency, usually written as:

$$G(e^{j\omega T}) = \sum_{p=-\infty}^{\infty} g_p \, e^{-j\omega pT} \qquad (*)$$

The signal sample values may be expressed in terms of the DTFT by noting that the equation above has the form of a Fourier series *(as a function of ω)* and hence the sampled signal can be obtained directly as:

$$g_n = \frac{1}{2\pi} \int_0^{2\pi} G(e^{j\omega T}) \, e^{+jn\omega T} \, d\omega T$$

[You can show this for yourself by first noting that (*) is a complex Fourier series with coefficients g_p]

The Discrete Fourier Transform (DFT)

The DFT is the fundamental building block of many modern signal processing systems. We will later derive a fast algorithm for DFT computation, known as the *Fast Fourier Transform* (FFT). The FFT is possibly the most widely used digital signal processing algorithm ever invented.

The Discrete Time Fourier Transform (DTFT):

$$G(e^{j\omega T}) = \sum_{p=-\infty}^{\infty} g_p \, e^{-j\omega pT}$$

expresses the spectrum of a sampled signal in terms of the signal samples but is not computable on a digital computer for two reasons:

1. The frequency variable ω is continuous.
2. The summation involves an infinite number of samples

These problems can be overcome by:

1. evaluating the DTFT at a finite collection of discrete frequencies
2. Performing the summation over a finite number of data points.

Step 1. has no undesirable consequences since we can always include any frequency of interest in the collection .

Step 2. does have consequences, since signals are generally not of finite duration. These consequences can be rigorously analysed by windowing analysis.

The discrete set of frequencies chosen is arbitrary. However, since the DTFT is *periodic* we generally choose a uniformly spaced grid of N frequencies covering the range $\omega T = 0 \rightarrow 2\pi$. If the summation is then truncated to just N data points, we get the DFT:

$$G_p = G(e^{j\frac{2\pi}{N}p}) = \sum_{n=0}^{N-1} g_n \, e^{-j\frac{2\pi}{N}np} \tag{1}$$

The inverse DFT can be used to obtain the sampled signal values from the DFT:

Multiply each side of equation (1) by $e^{j\frac{2\pi}{N}pq}$ and sum over $p = 0$ to $N-1$:

$$\sum_{p=0}^{N-1} G_p \, e^{j\frac{2\pi}{N}pq} = \sum_{p=0}^{N-1} \sum_{n=0}^{N-1} g_n \, e^{-j\frac{2\pi}{N}np} \, e^{j\frac{2\pi}{N}pq}$$

$$= \sum_{n=0}^{N-1} g_n \sum_{p=0}^{N-1} e^{j\frac{2\pi}{N}(q-n)p}$$

Note the orthogonality property of the complex exponentials:

$$\sum_{p=0}^{N-1} e^{j\frac{2\pi}{N}(q-n)p} = \begin{cases} N & n = q \\ 0 & n \neq q \end{cases}$$

Hence:

$$g_q = \frac{1}{N} \sum_{p=0}^{N-1} G_p \, e^{j\frac{2\pi}{N}pq}$$

The above equations for G_p and g_n are the Discrete Fourier Transform pair, summarised as:

$$G_p = \sum_{n=0}^{N-1} g_n \, e^{-j\frac{2\pi}{N}np}$$

$$g_n = \frac{1}{N} \sum_{p=0}^{N-1} G_p \, e^{j\frac{2\pi}{N}pn}$$

Properties of the DFT

- G_p is periodic, i.e.

$$G_{p+N} = G_p, \quad \text{for each } p$$

- g_n is periodic, i.e.

$$g_{n+N} = g_n, \quad \text{for each } n$$

[This may seem strange since we defined the DFT by truncating the signal. However, the inverse DFT formula implies that the data are periodic if we calculate values of x_{p+N} where $p + N$ lies outside the usual range $\{0, 1, ..., N - 1\}$].

- For *real* data g_n we have conjugate symmetry, i.e.

$$G_p = G^*_{-p} = G^*_{N-p}$$

[You should check that you can show these results from first principles]

Rewriting in terms of a sequence of sampled values x_n and transformed values X_p, the Discrete Fourier Transform (DFT) of a sequence of data $\{x_n\}$ is given by:

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}, \quad p \in \{0, 1, ..., N-1\} \tag{1}$$

$$x_n = \frac{1}{N} \sum_{p=0}^{N-1} X_p e^{j\frac{2\pi}{N}np}, \quad n \in \{0, 1, 2, ..., N-1\} \tag{2}$$

Can think of this as a vector operation:

- Take a vector of samples as input:

$$\mathbf{x} = [x_0, \, x_1, \, ..., x_{N-1}]^T$$

- Get a vector of frequency values as output:

$$\mathbf{X} = [X_0, \, X_1, \, ..., X_{N-1}]^T$$

Can write this as:

$$\mathbf{X} = \mathbf{M}\mathbf{x}$$

where $\mathbf{M}$ is the appropriate (NxN) matrix

The Fast Fourier Transform (FFT)

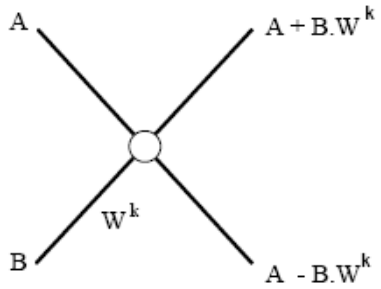
- The discovery of the FFT was 'first' announced by Cooley and Tukey in 1965 - their paper is the most cited mathematical paper ever written
- They were actually over 150 years late - the principle of the FFT was later discovered in an obscure section of Gauss' (as in Gaussian) own notebooks in 1806
- Computation of X_p for $p = 0, 1, \ldots, N-1$ requires on the order of N^2 complex multiplications and additions [assuming that the complex exponentials have been pre-computed and stored].
- The Fast Fourier Transform reduces the required number of arithmetic operations to the order of $\frac{N}{2}\log_2(N)$ when N is a power of 2.
- This speed-up is dramatic for large N - for $N = 1024$ the speed-up is a factor of 205, for $N = 8192$ the speed-up is 1260, ...
- There are many different types of FFT algorithm, and many different derivations possible, including some which operate for N not a power of 2.
- Here we consider the most basic 'radix-2' algorithm which requires N to be a power of 2.

Derivation

- The FFT derivation relies on redundancy in the calculation of the basic DFT
- A recursive algorithm is derived that repeatedly rearranges the problem into two simpler problems of half the size
- Hence the basic algorithm operates on signals of length a power of 2, i.e.

$$N = 2^M \qquad \text{(for some integer M)}$$

- At the bottom of the tree, we have the classic FFT `butterfly' structure (details later):



First, take the basic DFT equation:

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}$$

Now, split the summation into two parts: one for even n and one for odd n:

$$\begin{aligned} X_p &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \, e^{-j\frac{2\pi}{N}(2n)p} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{N}(2n+1)p} \quad (*) \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np} + e^{-j\frac{2\pi}{N}p} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np} \\ &= A_p + W^p B_p \end{aligned}$$

where

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}$$

$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

$$W = e^{-j\frac{2\pi}{N}}$$

- Notice now that A_p and B_p are themselves DFTs each of length $N/2$:
  - A_p is the DFT of a sequence $\{x_{2n}\} = \{x_0, x_2, ...x_{N-4}, x_{N-2}\}$
  - B_p is the DFT of a sequence $\{x_{2n+1}\} = \{x_1, x_3, ...x_{N-3}, x_{N-1}\}$

- We know, however that the DFT is periodic in the frequency domain (in this case with period N/2). This leads to further simplifications, as follows.

To see how this simplifies, look at the original DFT in (*) above, but evaluated at frequencies $p + N/2$:

$$X_{p+N/2} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n}\, e^{-j\frac{2\pi}{(N/2)}n(p+N/2)} + e^{-j\frac{2\pi}{N}(p+N/2)} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\, e^{-j\frac{2\pi}{(N/2)}n(p+\frac{N}{2})}$$

Now, simplify terms as follows:

$$e^{-j\frac{2\pi}{(N/2)}n(p+N/2)} = e^{-j\frac{2\pi}{(N/2)}np}, \quad e^{-j\frac{2\pi}{N}(p+N/2)} = -e^{-j\frac{2\pi}{N}p}$$

Hence,

$$X_{p+N/2} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n}\, e^{-j\frac{2\pi}{(N/2)}np} - e^{-j\frac{2\pi}{N}p} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\, e^{-j\frac{2\pi}{(N/2)}np}$$

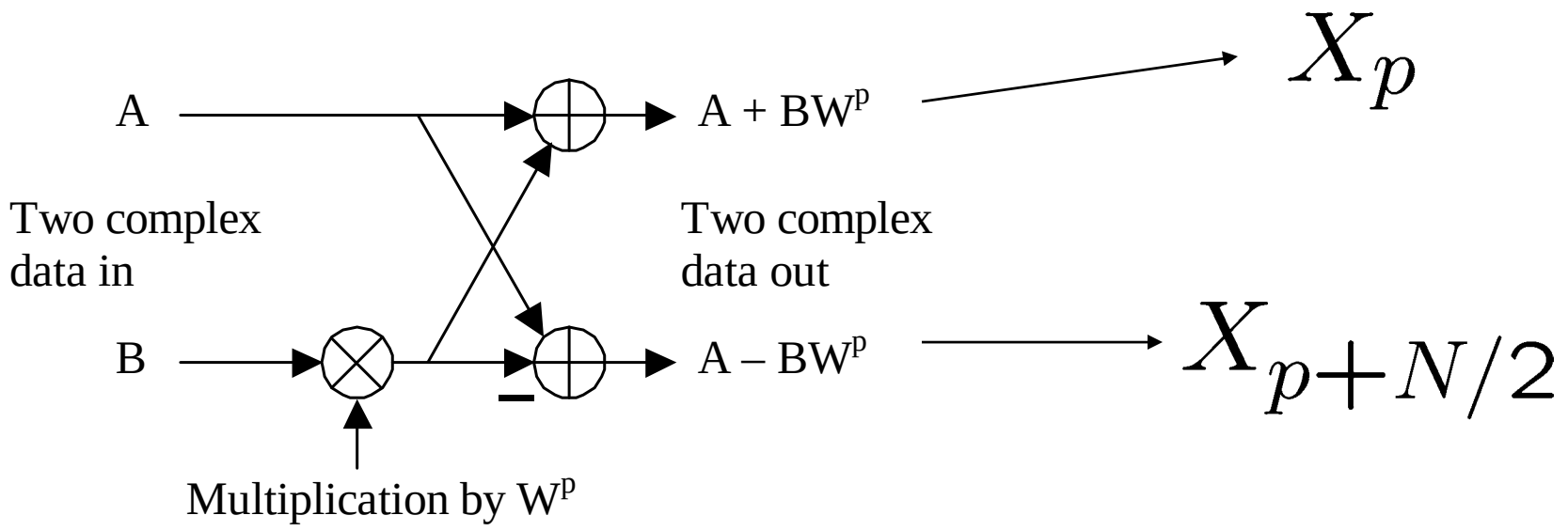
$$= A_p - W^p B_p$$

with A_p W^p and B_p defined as before

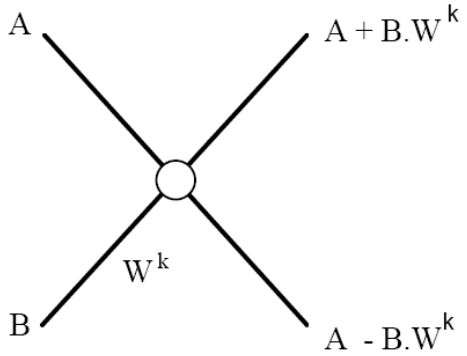
Now, compare the equations for $X_{p+N/2}$ with that for X_p:

$$X_p = A_p + W^p B_p, \quad X_{p+N/2} = A_p - W^p B_p$$

This defines the FFT butterfly structure:



Or, in more compact form:



('Butterfly')

Computational load:

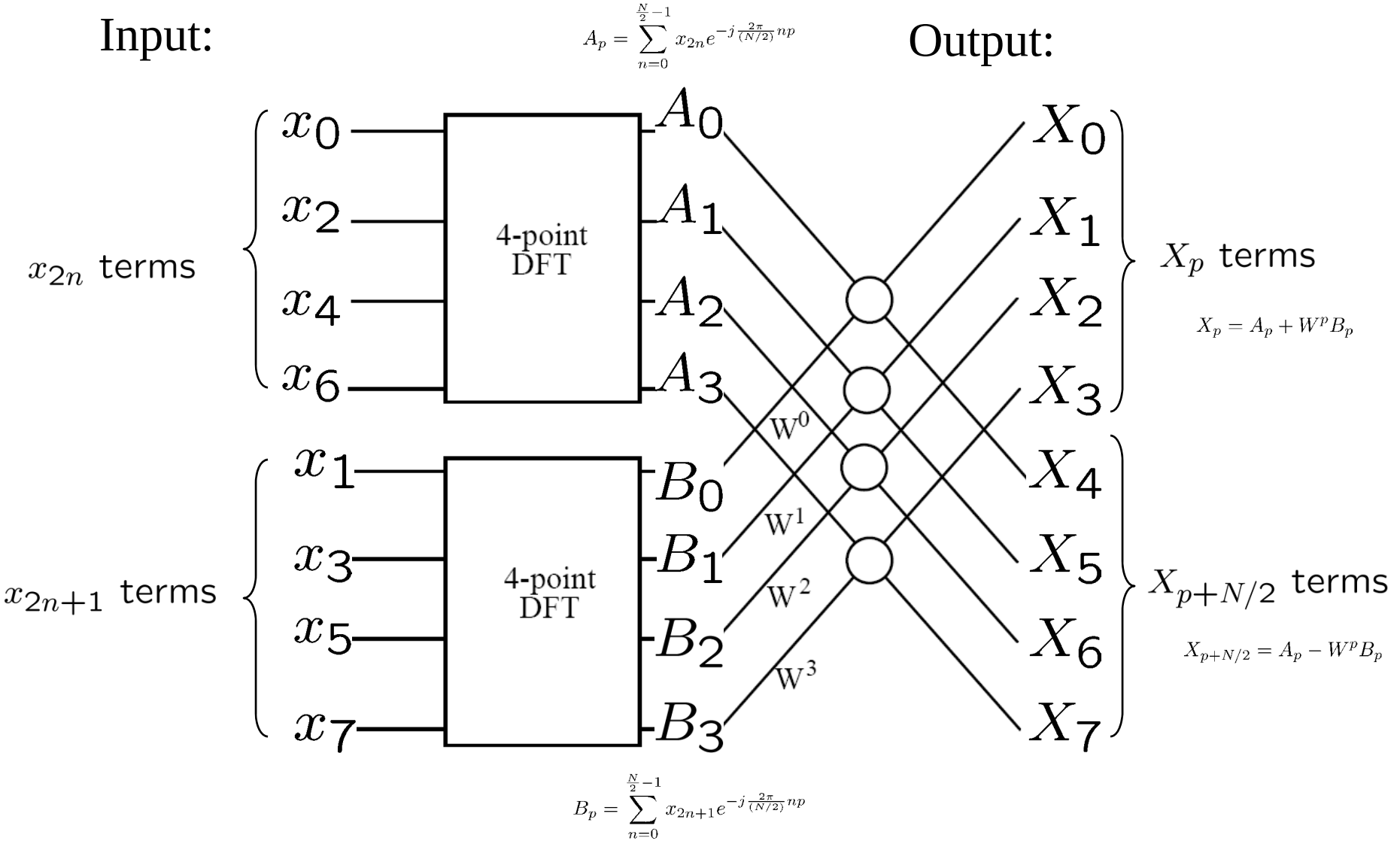
Look at the two required terms (W^p assumed precomputed and stored):

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}, \quad B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np},$$

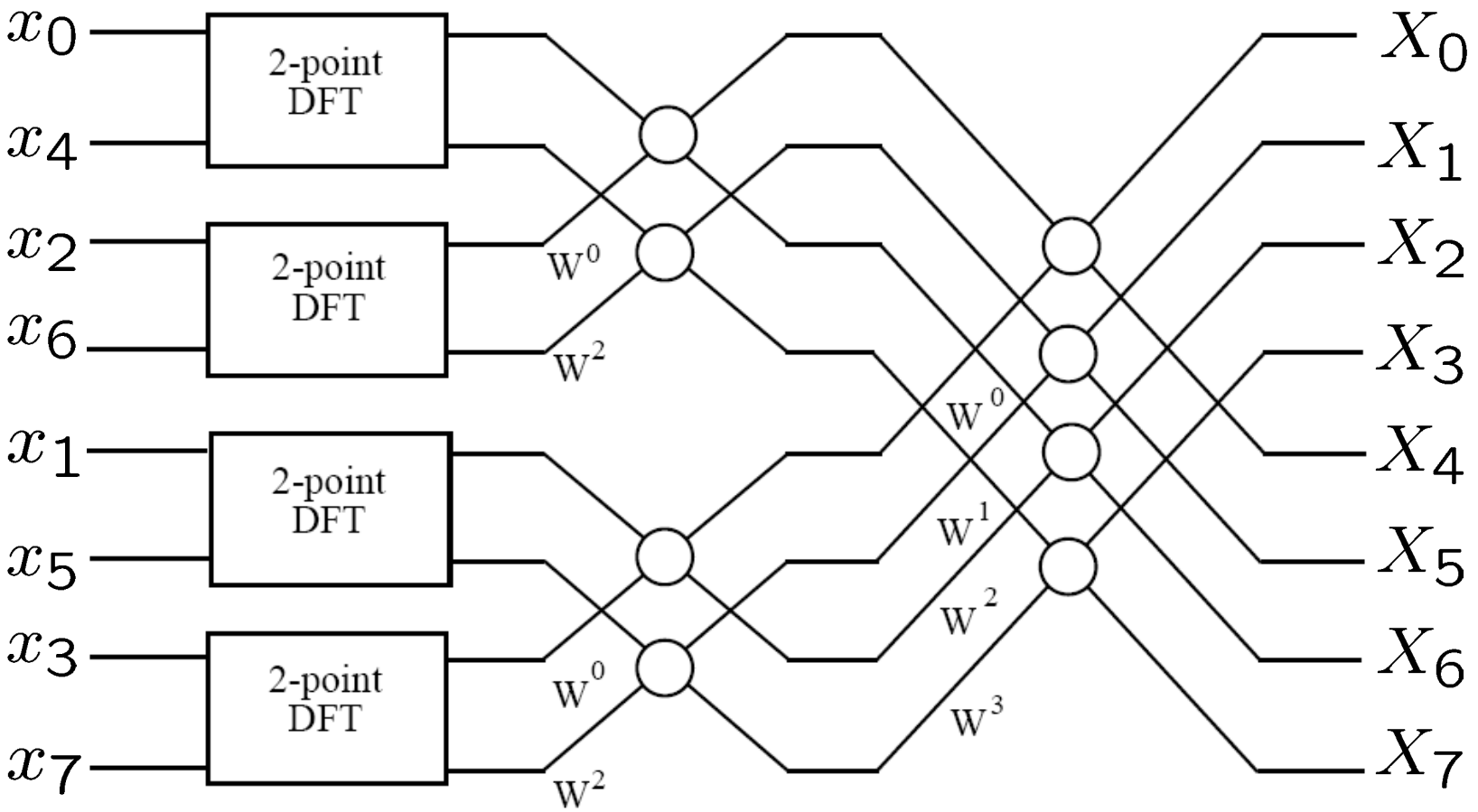
- The terms A_p and B_p need only be computed for $p = 0, 2, ..., N/2 - 1$, since $X_{p+N/2}$ has been expressed in terms of A_p and B_p - hence we have uncovered redundancy in the DFT computation.
- Thus calculate the A_p and B_p for $p = 0, 1, ..., N/2 - 1$ and use them for calculation of both X_p and $X_{p+N/2}$

- The number of complex multiplies and additions is:
  - A_p requires $N/2$ complex multiplies and additions; so does B_p. The total for all $p = 0, 1, ..., N/2 - 1$ is then $2(N/2)^2$ multiplies and additions for the calculation of all the A_p and B_p terms.
  - $N/2$ multiplies for the calculation of $W^p B_p$ for all $p = 0, 1, 2, ..., N/2 - 1$
  - $N = N/2 + N/2$ additions for calculation of $A_p + W^p B_p$ and $A_p - W^p B_p$
- Thus total number of complex multiplies and additions is approximately $N^2/2$ for large N
- The computation is approximately halved compared to the direct DFT evaluation

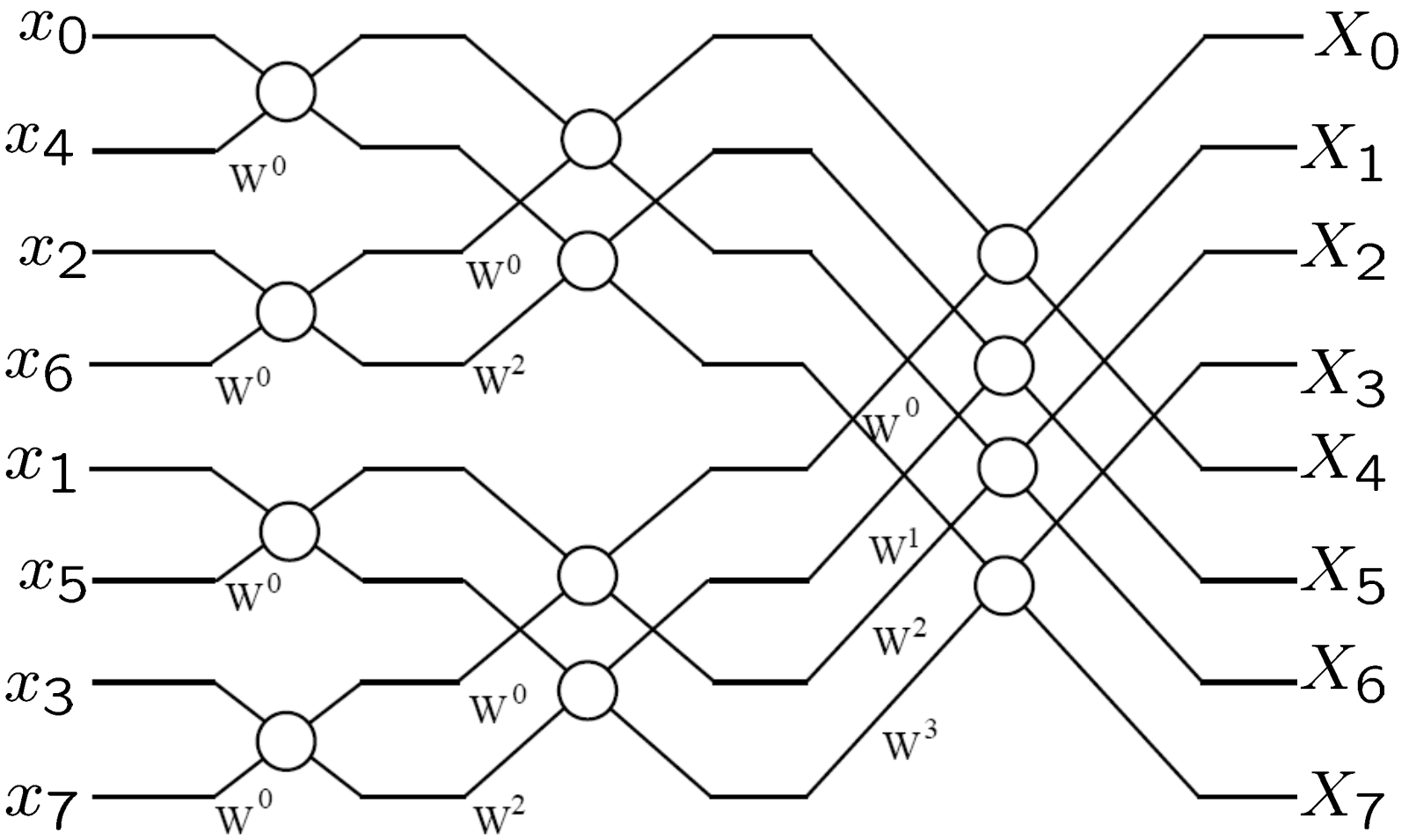
A flow diagram for a N=8 DFT is shown below:



Assuming that $(\frac{N}{2})$ is even, the same process can be carried out on each of the $(\frac{N}{2})$ point DFTs to further reduce the computation. The flow diagram for incorporating this extra stage of decomposition into the computation of the $N = 8$ point DFT is shown below.



It can be seen that if $N = 2^M$ then the process can be repeated M times to reduce the computation to that of evaluating N single point DFTs. Thus the flow chart for computing the $N = 8$ point DFT is as shown below.



Examination of the final chart shows that it is necessary to shuffle the order of the input data. This data shuffle is usually termed *bit-reversal* for reasons that are clear if the indices of the shuffled data are written in binary.

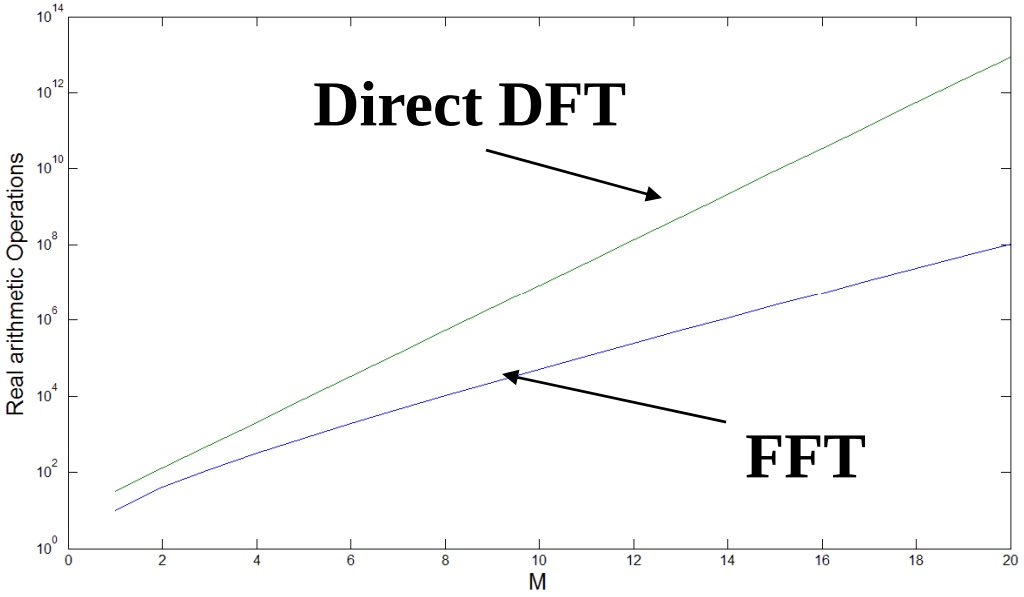
Binary	Bit Reverse	Decimal
000	000	0
001	100	4
010	010	2
011	110	6
100	001	1
101	101	5
110	011	3
111	111	7

Computational Load of full FFT algorithm:

The type of FFT we have considered, where $N = 2^M$, is called a radix-2 FFT. It has $M = \log_2 N$ stages, each using $N / 2$ butterflies

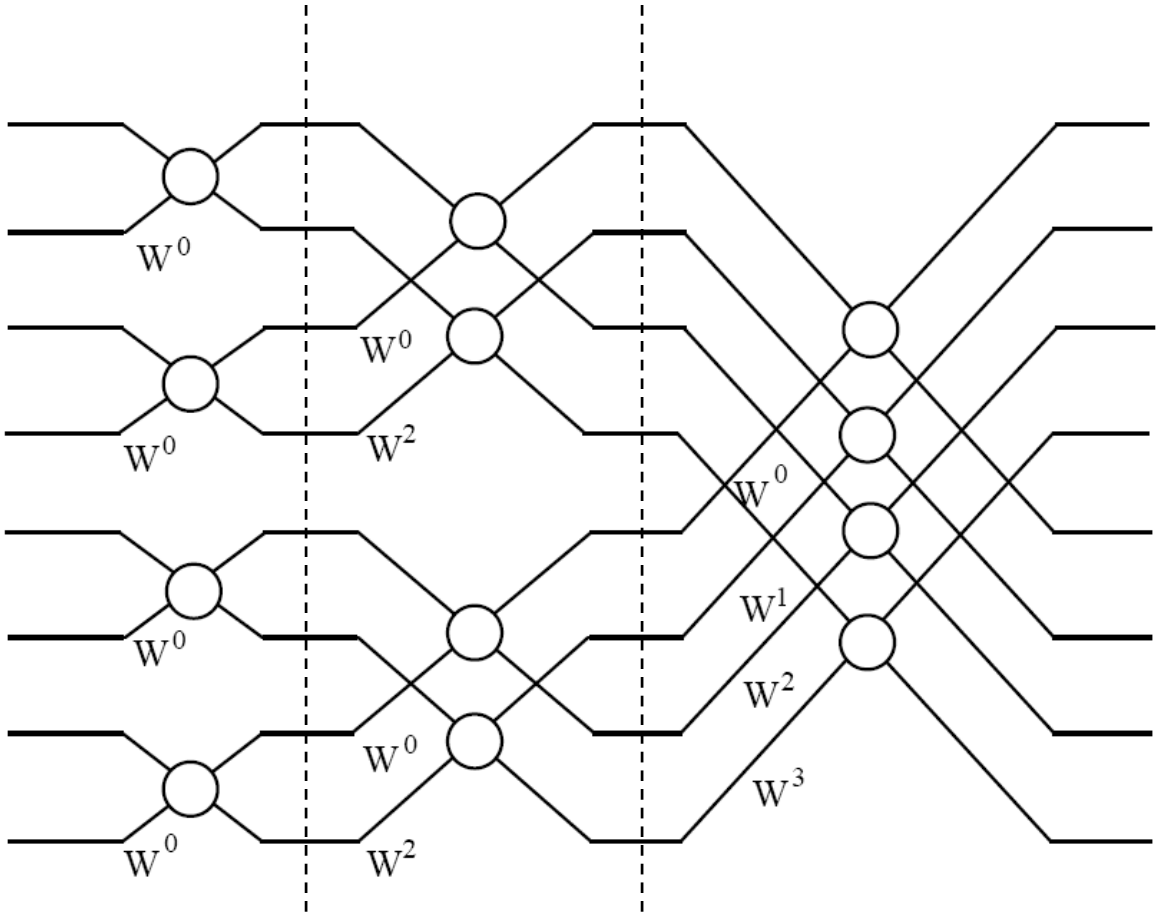
Since a complex multiplication requires 4 real multiplications and 2 real additions, and a complex addition/subtraction requires 2 real additions, a butterfly requires 10 real operations. Hence the radix-2 N-point FFT requires $10(N / 2)\log_2 N$ real operations compared to about $8N^2$ real operations for the DFT.

This is a huge speed-up in typical applications, where N is 128 – 4096:



The FFT algorithm has a further significant advantage over direct evaluation of the DFT expression in that computation can be performed *in-place*. This is best illustrated in the final flow chart where it can be seen that after two data values have been processed by the *butterfly* structure, those data are not required again in the computation and they may be replaced, in the computer or in the chip memory, with the values at the output of the *butterfly* structure.

Input



Output

The Inverse FFT (IFFT)

Apart from the scale factor 1 / N, the Inverse DFT has the same form as the DFT, except that the conjugate W^* replaces W. Hence the computation algorithm is the same, with a final scaling by 1 / N.

Other types of FFT

There are many FFT variants. The form of FFT we have described is called “decimation in time”; there is a form called “decimation in frequency” (but it has no advantages).

The "radix 2" FFT must have length N a power of 2. Slightly more efficient is the "radix 4" FFT, in which 2-input 2-output butterflies are replaced by 4-input 4-output units. The transform length must then be a power of 4 (more restrictive).

A completely different type of algorithm, the Winograd Fourier Transform Algorithm (WFTA), can be used for FFT lengths equal to the product of a number of mutually prime factors (e.g. 9*7*5 = 315 or 5*16 = 80). The WFTA uses fewer multipliers, but more adders, than a similar-length FFT.

Efficient algorithms exist for FFTing **real (not complex) data at about 60% the effort of the same-sized complex-data FFT.**

The Discrete Cosine and Sine Transforms (**DCT and DST) are similar real-signal algorithms used in image coding.**

Applications of the FFT

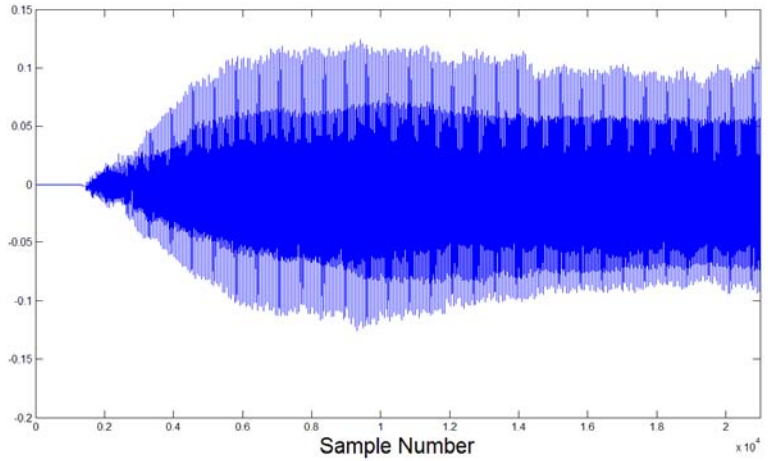
There FFT is surely the most widely used signal processing algorithm of all

It is the basic building block for a large percentage of algorithms in current usage

Specific examples include:

• Spectrum analysis – used for analysing and detecting signals
• Coding – audio and speech signals are often coded in the frequency domain using FFT variants (MP3, …)
• Another recent application is in a modulation scheme called OFDM, which is used for digital TV broadcasting (DVB) and digital radio (audio) broadcasting (DAB).
• Background noise reduction for mobile telephony, speech and audio signals is often implemented in the frequency domain using FFTs
….

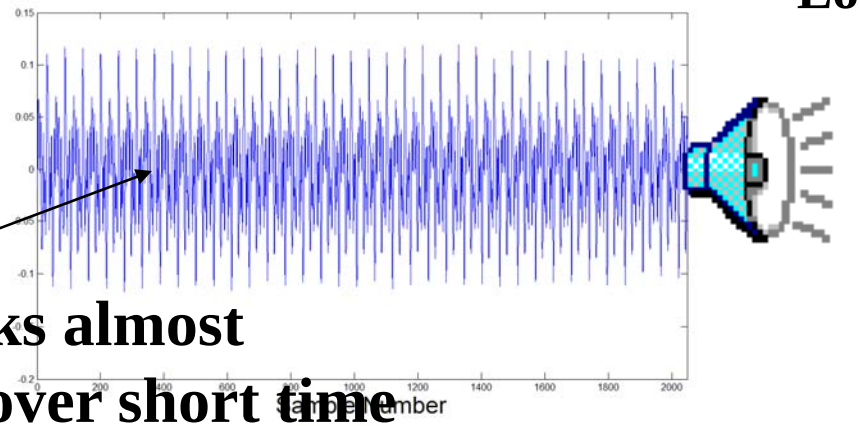
Case Study: Spectral analysis of a Musical Signal



Sample rate is 10.025 kHz (T=1/10,025 s)

Extract a short segment:

Load this into Matlab as a vector x

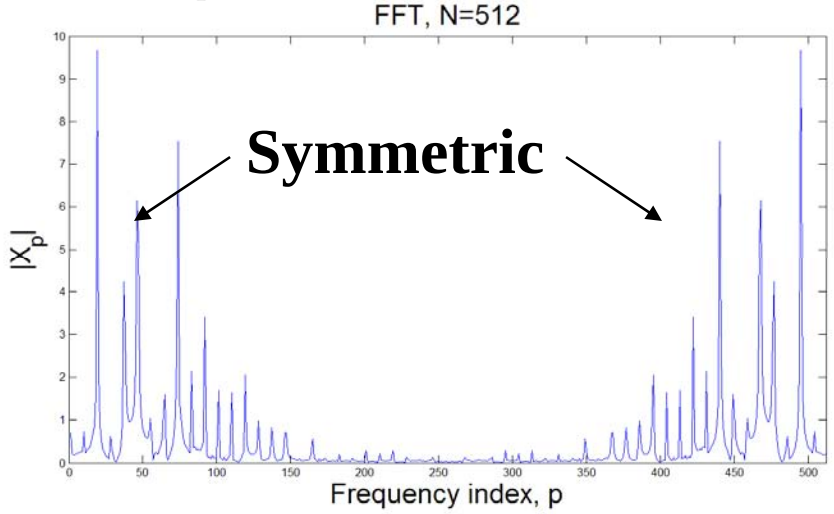


Take an FFT, N=512:

X=fft(x(1:512));

Note: looks almost Periodic over short time interval

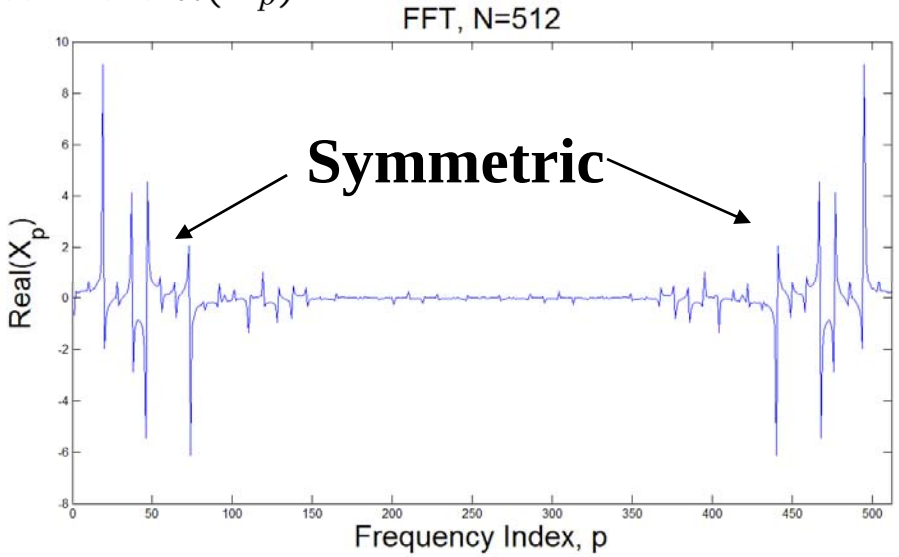
Modulus $|X_p|$



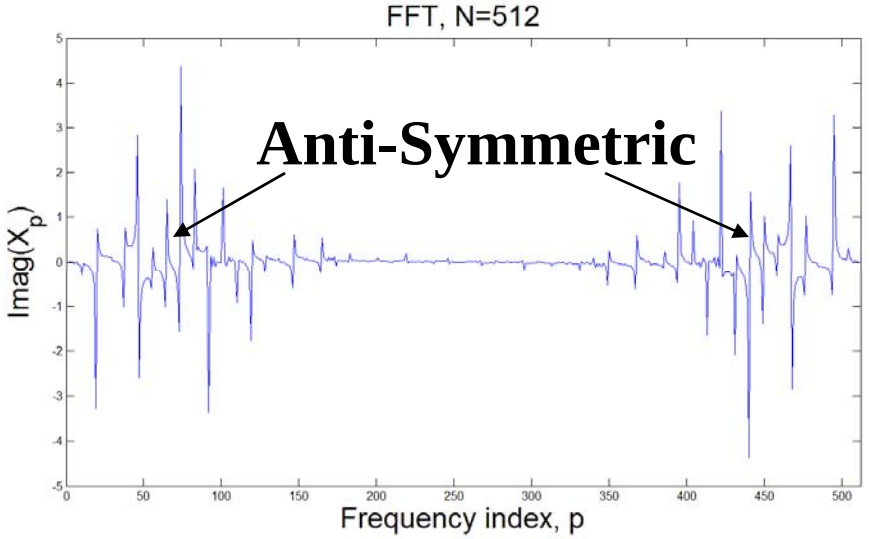
Note Conjugate symmetry as data are real:

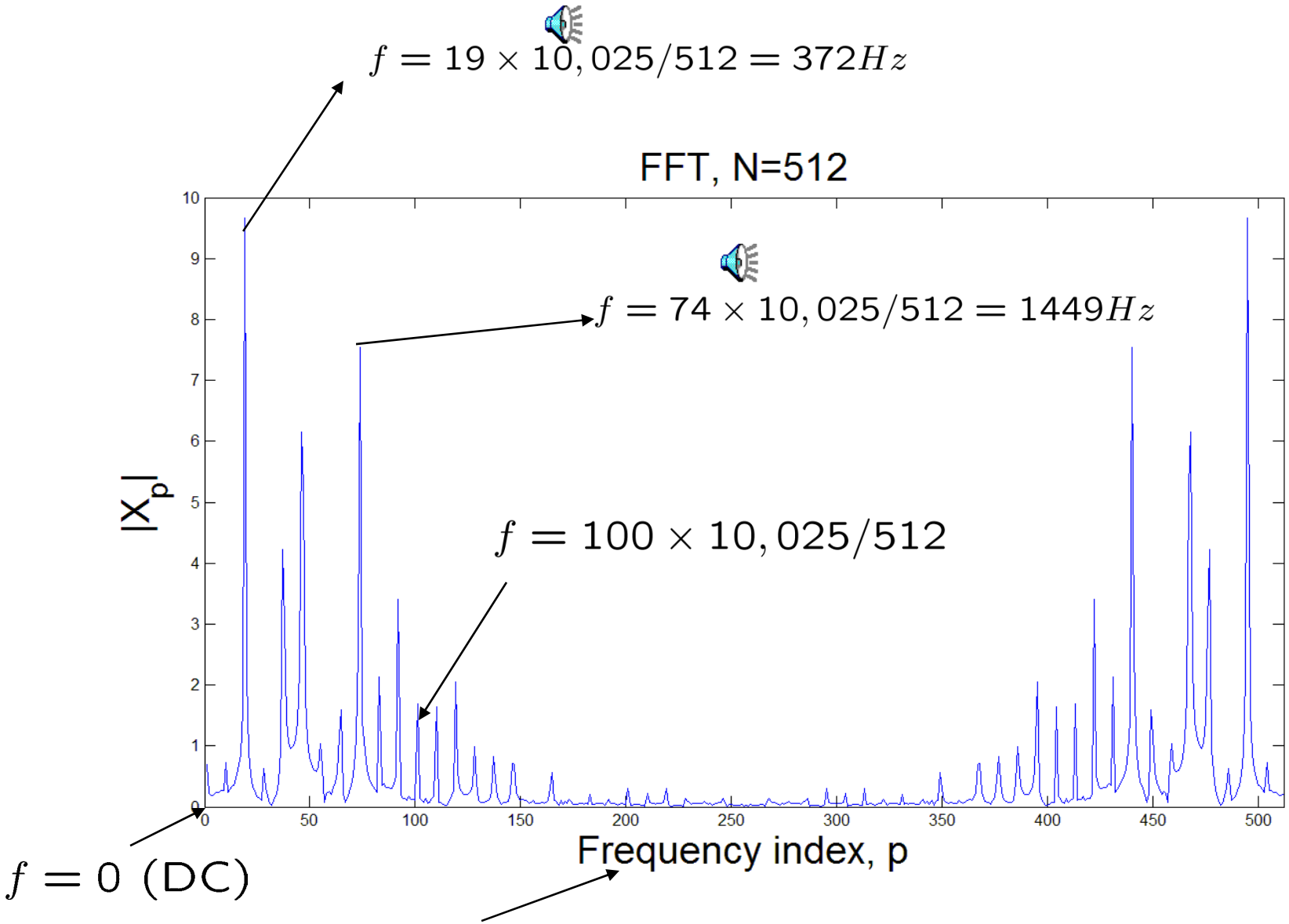
$$X_p = X^*_{N-p}$$

Real Part $\Re(X_p)$



Imaginary Part $\Im(X_p)$





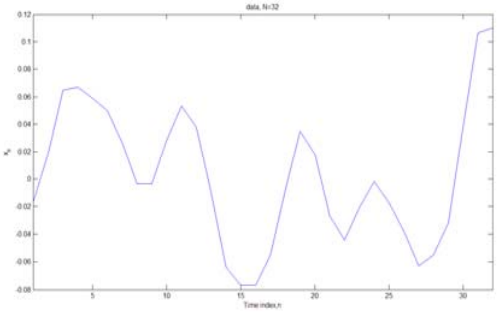
Frequency index corresponds to true frequency

$f = \frac{p f_0}{N} = \frac{p}{NT}$

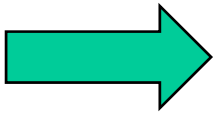
3F3 – Digital Signal Processing

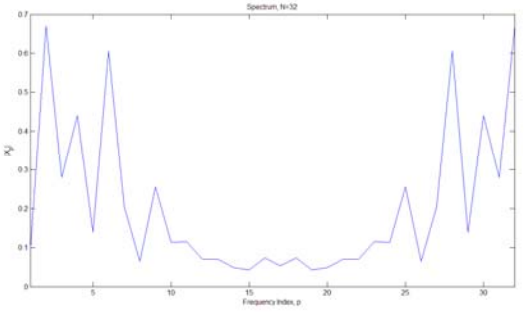
The Effect of data length, N

N=32



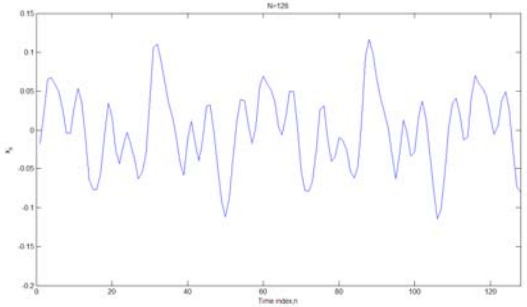
FFT



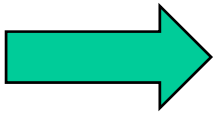


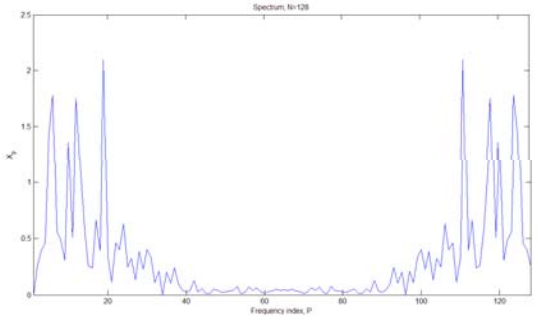
Low resolution

N=128

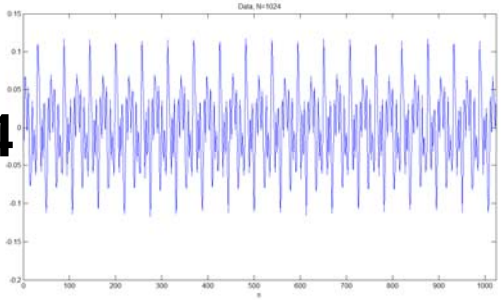


FFT

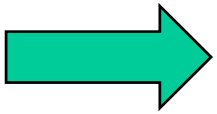


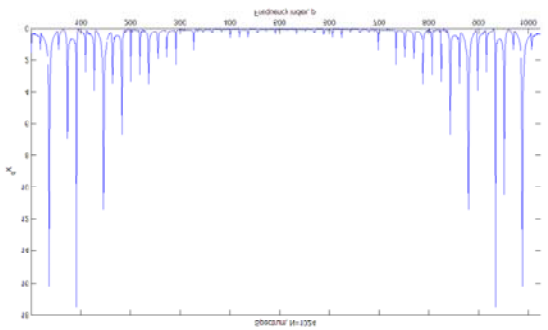


N=1024



FFT





High resolution

The DFT approximation to the DTFT

DTFT at frequency $\omega = \frac{2\pi p}{NT}$:

$$X(e^{j\frac{2\pi p}{NT}}) = \sum_{n=-\infty}^{\infty} x_n e^{-j\frac{2\pi p}{NT}n}$$

DFT:

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}$$

- Ideally the DFT should be a `good' approximation to the DTFT
- Intuitively the approximation gets better as the number of data points N increases
- This is illustrated in the previous slide – resolution gets better as N increases (more, narrower, peaks in spectrum).
- How to evaluate this analytically?
  - View the truncation in the summation as a multiplication by a rectangle window function
  - Then, in frequency domain, multiplication becomes *convolution*

Analysis:

Consider DTFT:

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x_n \, e^{-jn\omega T}$$

Truncate the summation, as for the DFT:

$$X_w(e^{j\omega T}) = \sum_{n=0}^{N-1} x_n \, e^{-jn\omega T}$$

Now, note that this is equivalent to an infinite summation, but with x_n pre-multiplied by a rectangle window function:

$$X_w(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} w_n x_n \, e^{-jn\omega T}$$

with

$$w_n = \begin{cases} 1, & n = 0,\ 1,\ 2,\ ...,\ N-1 \\ 0, & \text{otherwise} \end{cases}$$

$w[n]$

$x[n]$

$x[n]w[n]$

DTFT of w_n is $W(e^{j\theta})$

Now, take the DTFT of the windowed signal $x_w = w_n x_n$ directly:

$$
\begin{aligned}
X_w(e^{j\omega T}) &= \sum_{n=-\infty}^{\infty} \{x_n\, w_n\}\, e^{-jn\omega T} \\
&= \sum_{n=-\infty}^{\infty} x_n \left\{ \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta}) e^{jn\theta}\, d\theta \right\} e^{-jn\omega T} \\
&= \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta}) \sum_{n=-\infty}^{\infty} x_n\, e^{-jn(\omega T-\theta)}\, d\theta \\
X_w(e^{j\omega T}) &= \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta})\, X(e^{j(\omega T-\theta)})\, d\theta
\end{aligned}
$$

Inverse DTFT of $W(e^{j\theta})$

We see that the spectrum of the windowed signal is the convolution of the infinite duration signal spectrum and the window spectrum.

What is the DTFT of the window w_n?

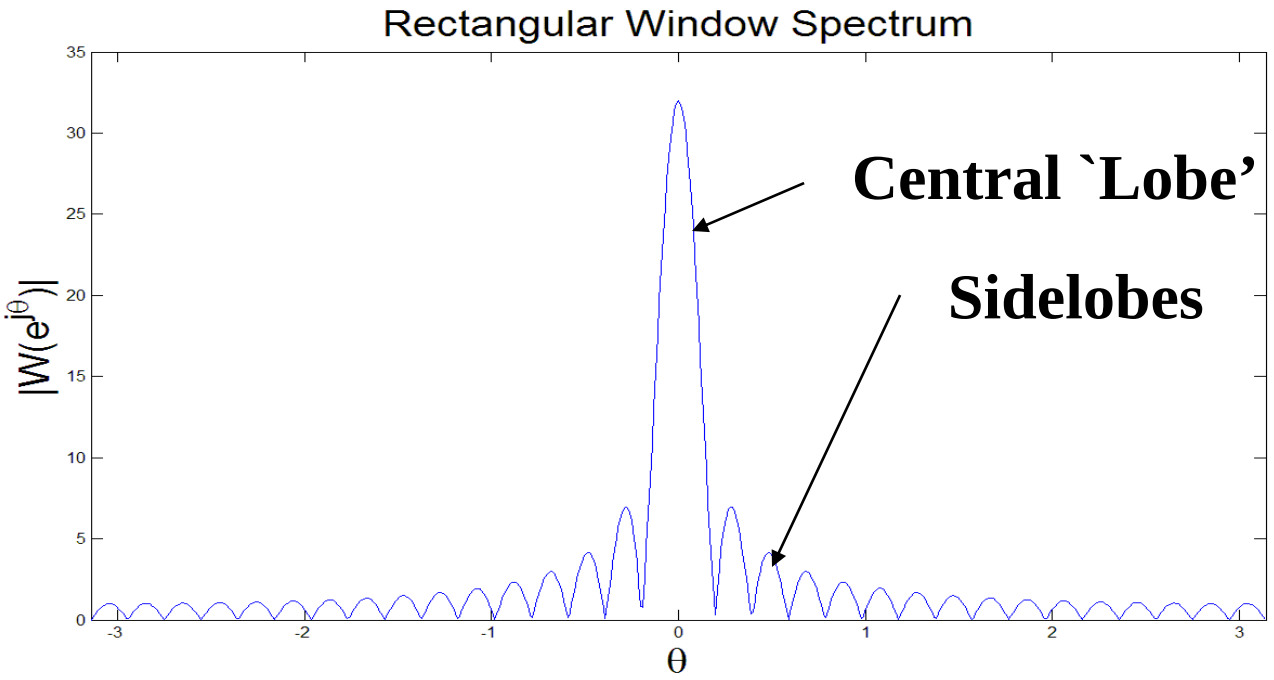
[Subst. $\theta = \omega T$ - makes no difference to form of results]:

$$W(e^{j\theta}) = \sum_{n=0}^{N-1} 1.e^{-jn\theta}$$

$$= e^{-j(N-1)\theta/2} \frac{\sin(N\theta/2)}{\sin(\theta/2)}$$

[Check you can get this result yourself as an extra examples question]

N=32

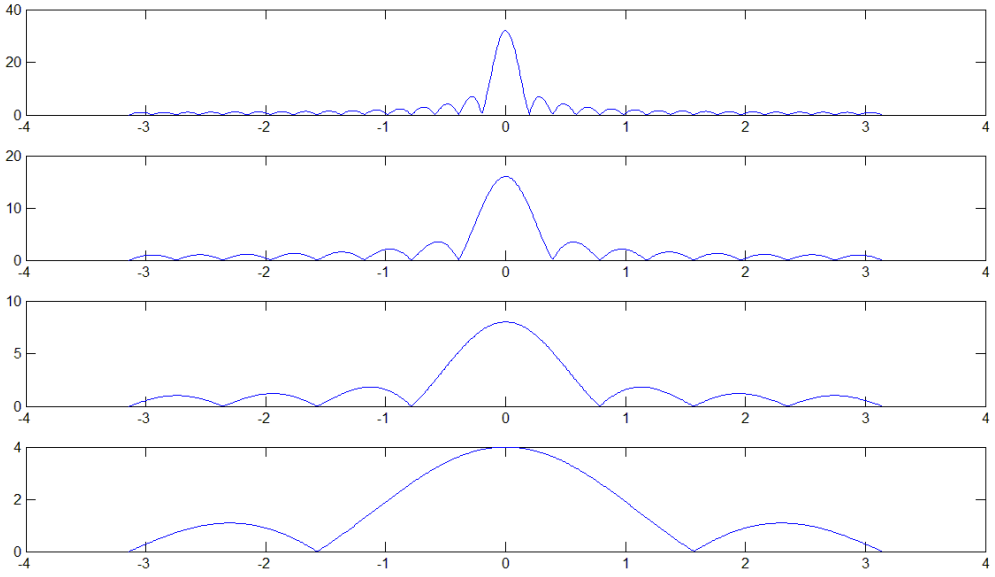


N=32

N=16

N=8

N=4



Lobe width inversely proportional to N

Now, imagine what happens when the sum of two frequency components is DFT-ed:

$$x_n = \exp(j\omega_1 nT) + \exp(j\omega_2 nT)$$

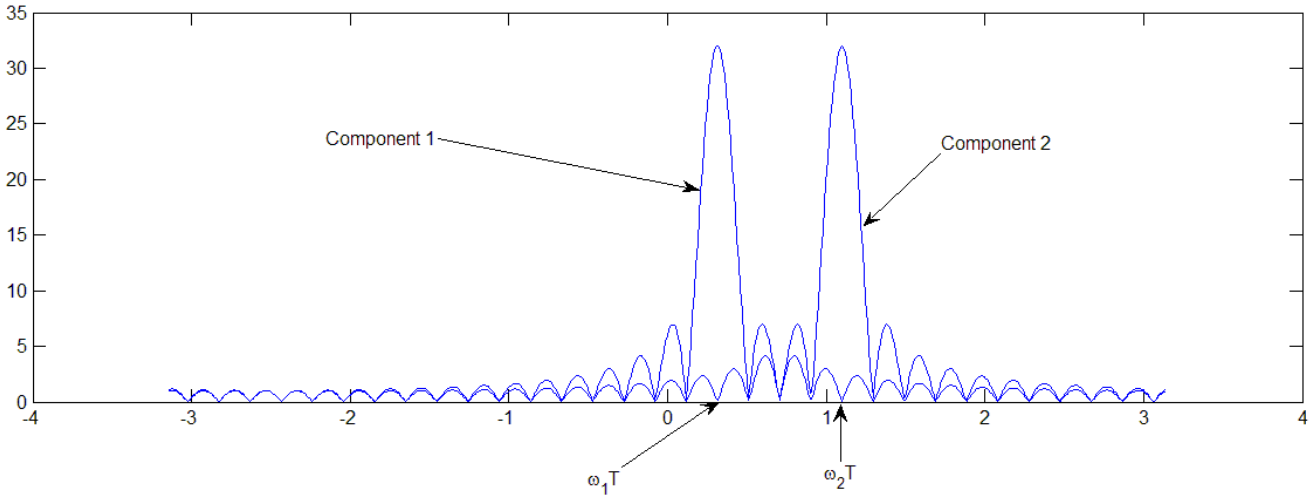
The DTFT is given by a train of delta functions:

$$X(e^{j\omega T}) = 2\pi \sum_{n=-\infty}^{+\infty} \delta(\omega T + 2n\pi - \omega_1 T) + \delta(\omega T + 2n\pi - \omega_2 T)$$

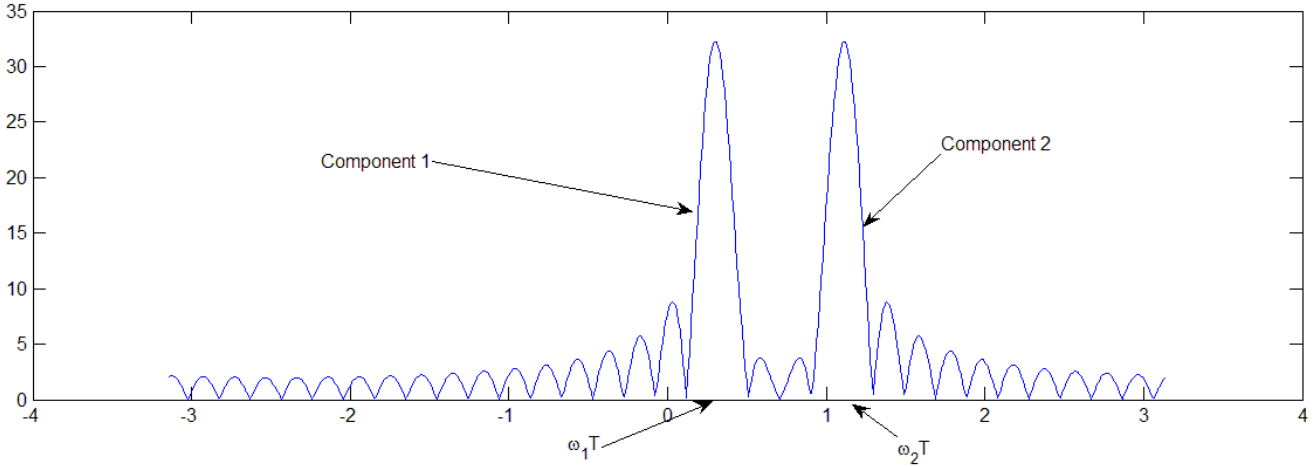
Hence the windowed spectrum is just the convolution of the window spectrum with the delta functions:

Now consider the DFT for the data:

Both components separately



Both components Together



ωT ⟶

Summary

- The rectangular window introduces broadening of any frequency components (`smearing')  and sidelobes that may overlap with other frequency components (`leakage').
- The effect improves as N increases
- However, the rectangle window has poor properties and better choices of w_n can lead to better spectral properties (less leakage, in particular) – i.e. instead of just truncating the summation, we can pre-multiply by a suitable window function w_n that has better frequency domain properties.
- More on window design in the filter design section of the course – see later

Section 2: Digital Filters

- A filter is a device which passes some signals 'more' than others (`selectivity'), e.g. a sinewave of one frequency more than one at another frequency.
- We will deal with linear time-invariant (LTI) digital filters.
- Recall that a **linear** system is defined by the principle of linear superposition:

If, for any signals $x1_n$ and $x2_n$ we have,

- Input $x1_n$ gives output $y1_n$
- Input $x2_n$ gives output $y2_n$
- Then, input $a\,x1_n + b\,x2_n$ gives output $a\,y1_n + b\,y2_n$

- If the linear system's parameters (coefficients) are constant, then it is Linear Time Invariant (LTI).

[Much of this material is based on material by Dr Malcolm Macleod]

Frequency response of a LTI digital system

Rather than write ωT, where ω is in rads/sec and T is the sample interval in seconds, we will use the normalised radian frequency Ω, where $\Omega = \omega T$ is in units of rads/sample. Hence $\Omega = 2\pi$ is the sampling frequency, and $\Omega = \pi$ is half the sampling frequency.

If a single frequency cisoid $x_n = \exp(jn \Omega)$ is input to a linear digital system (for all time; $-\infty < n < \infty$), all signals inside the system, including the output signal, will also have time variation of the form $\exp(jn \Omega)$.

Thus if

$$x_n = \exp(jn \Omega)$$

then

$$y_n = \beta(\Omega) \exp(jn \Omega),$$

where $\beta(\Omega)$ is a complex function of frequency, called the frequency response of the system. The 'magnitude' response is simply $| \beta(\Omega) |$.

Write the input data sequence as:

$$x_n, \quad n = -\infty, \ldots + \infty$$

[Recall that x_n is shorthand for $x(nT)$, the sampled signal at time nT]

And the corresponding output sequence as:

$$y_n, \quad n = -\infty, \ldots + \infty$$

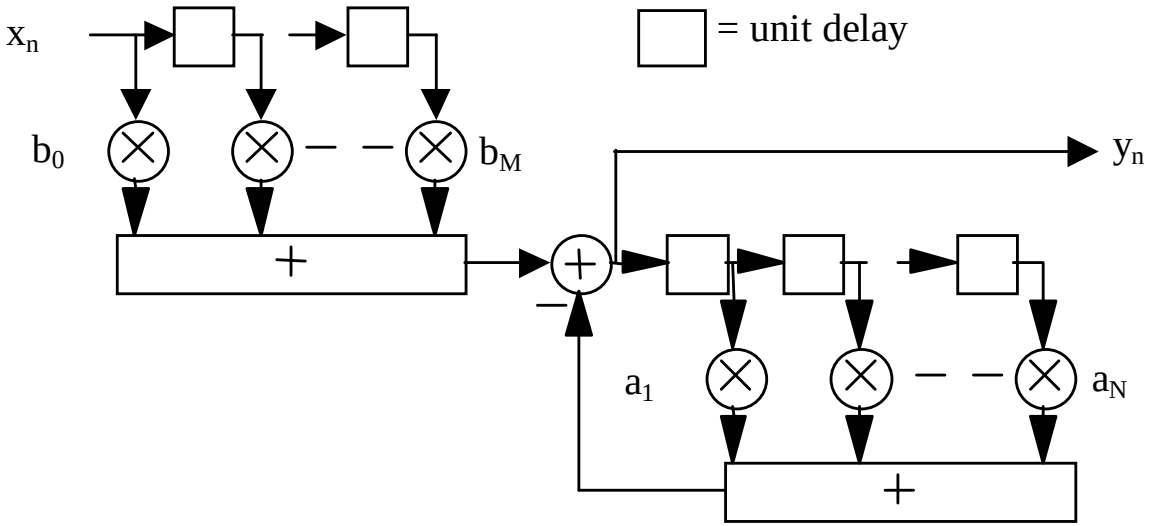
The linear time-invariant digital filter can then be described by the difference equation:

$$\begin{aligned} y_n &= -a_1 y_{n-1} - a_2 y_{n-2} - \ldots - a_N y_{n-N} + b_0 x_n + \ldots + b_M x_{n-M} \\ &= -\sum_{k=1}^{N} a_k y_{n-k} + \sum_{k=0}^{M} b_k x_{n-k} \end{aligned}$$

where the coefficients $\{a_k\}$ and $\{b_k\}$ are real.

The larger of M or N is known as the *order* of the filter.

A direct form implementation of (3.1) is:



The operations shown in the Figure above are the full set of possible linear operations:

- constant delays (by any number of samples),
- addition or subtraction of signal paths,
- multiplication (scaling) of signal paths by constants - (incl. -1),

Any other operations make the system non-linear.

Matlab filter functions

Matlab has a filter command for implementation of linear digital filters.

The format is

```
y = filter( b, a, x);
```

where

b = [b_0 b_1 b_2 ... b_M]; a = [1 a_1 a_2 a_3 ... a_N];

So to compute the first P samples of the filter's impulse response,

```
y = filter( b, a, [1 zeros(1,P)]);
```

Or step response,

```
y = filter( b, a, [ones(1,P)]);
```

To evaluate the frequency response at n points equally spaced in the normalised frequency range θ=0 to θ= π, Matlab's function `freqz` is used:

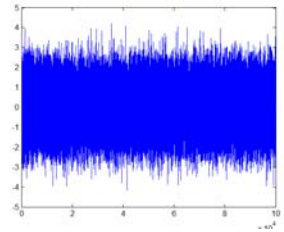
```
freqz(b,a,n);
```

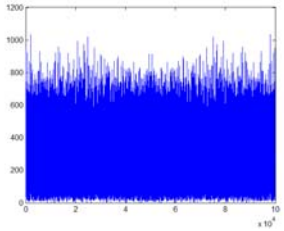
Filtering example:

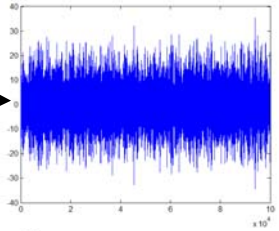
Generate a Gaussian random noise sequence:

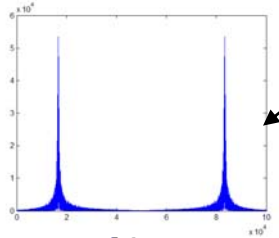
Matlab code:

```
x=randn(100000,1);
plot(x)
plot(abs(dft(x)))
soundsc(x,44100)

a=[1 -0.99 0.9801];
b=[1 –0.1 –0.56];

y=filter(b,a,x);
plot(y)
plot(abs(dft(y)))
soundsc(y,44100)
```









Selective amplification
Of one frequency

Impulse Response

- The "digital impulse" is the sequence

$$\delta_n = 1,\ 0,\ 0,\ ...$$

  and its z-transform is

$$\Delta(z) = 1.$$

- The output sequence y_n in response to δ_n is the *impulse response*, h_n.

- We know that

$$Y(z) = H(z)X(z),$$

  so when $x_n = \delta_n$,

$$Y(z) = H(z)\Delta(z) = H(z)$$

  .

- Hence $H(z)$ is the z-transform of impulse response h_n.

Transfer Function, Poles and Zeros

$$Y(z) = -\sum_{k=1}^{N} a_k Y(z) z^{-k} + \sum_{k=0}^{M} b_k X(z) z^{-k}$$

Hence, rearranging, get the transfer function of the filter:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

The roots of the numerator polynomial in *H(z)* are known as the zeros, and the roots of the denominator polynomial as poles. In particular, factorize *H(z)* top and bottom:

$$H(z) = b_0 \frac{\Pi_{q=1}^{N}(1 - c_q z^{-1})}{\Pi_{q=1}^{M}(1 - d_q z^{-1})}$$

Then, $\{c_q\}$ are the zeros and $\{d_q\}$ are the poles of the system.

Clearly all the poles $\{d_q\}$ must lie within the unit circle for filter stability, as for any discrete time system.

[Recall that $\Omega = \omega T$ is the normalised frequency]

Frequency Response

- The frequency response is defined as:

$$\beta(\Omega) = H(z)|_{z=\exp(j\Omega)} = H(e^{j\Omega})$$

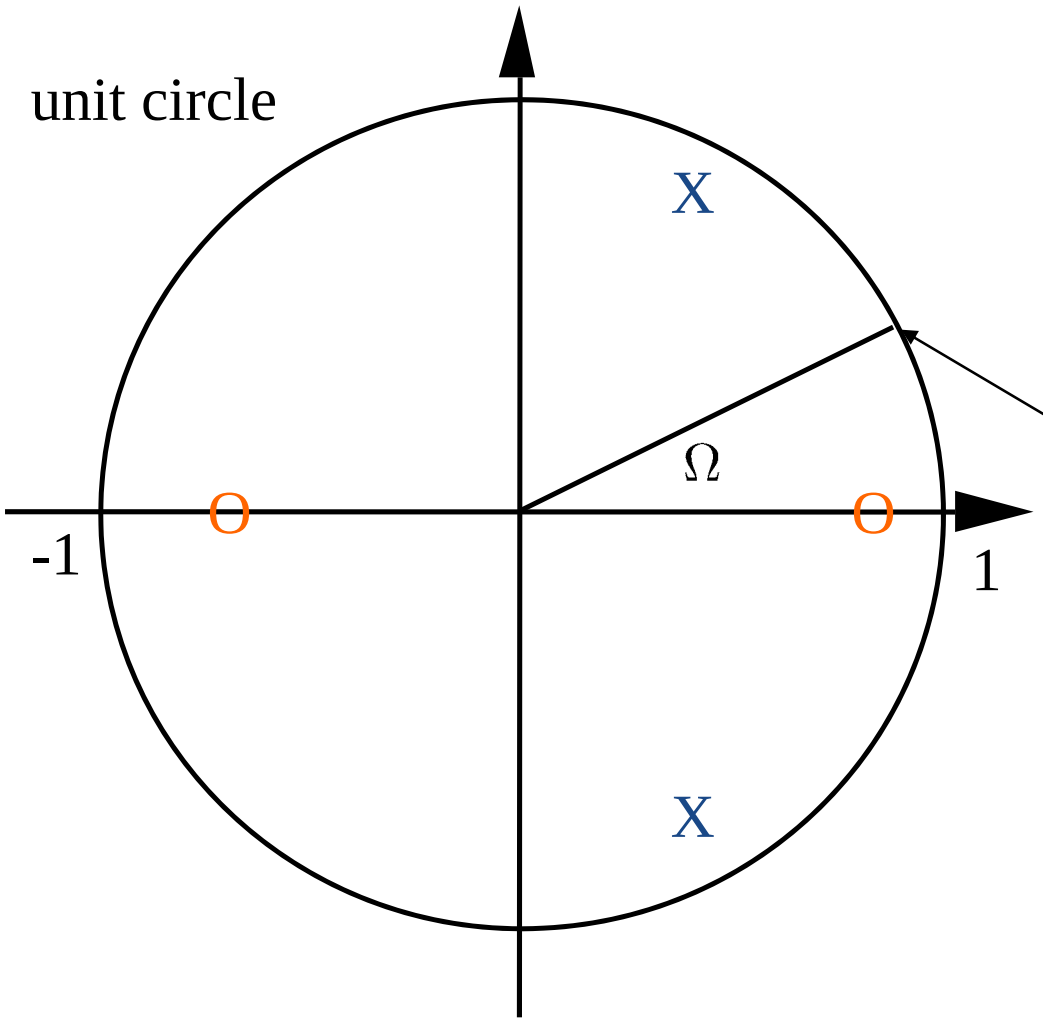
- $H(e^{j\Omega})$ is periodic with period 2π, since $\exp(j(\Omega + 2\pi)) = \exp(j\Omega)$.
- If a_p and b_p in the filter are real coefficients then

$$\beta(-\Omega) = \beta^*(\Omega)$$

  (conjugate symmetry).

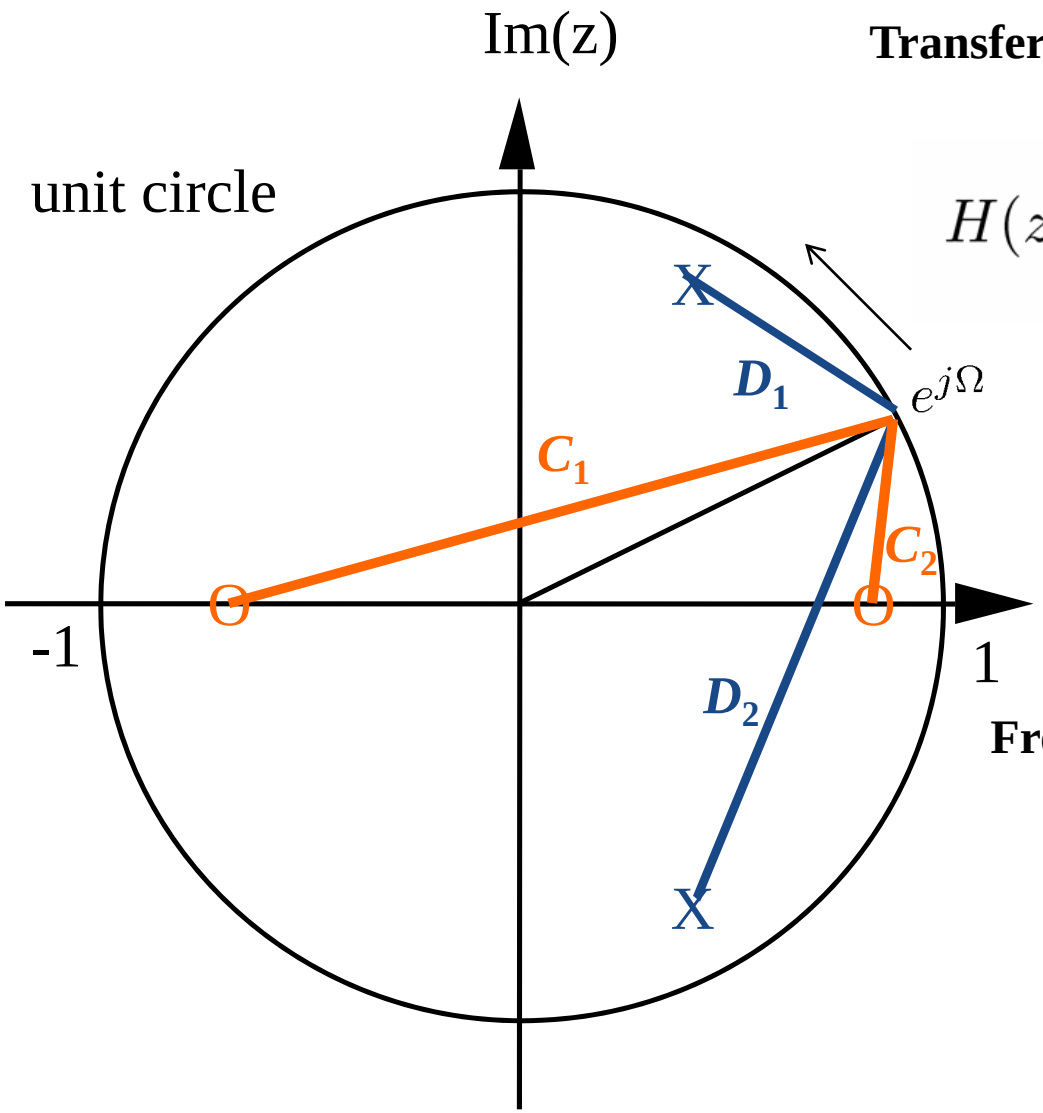
- We will make use of the graphical interpretation of frequency response taught in other courses (see also 3F3 practical). For example, the following illustrates a system with 2 poles and 2 zeros:

System has 2 poles (x) and 2 zeros (o)



Proceed around the unit circle with

$$z = e^{j\Omega}$$

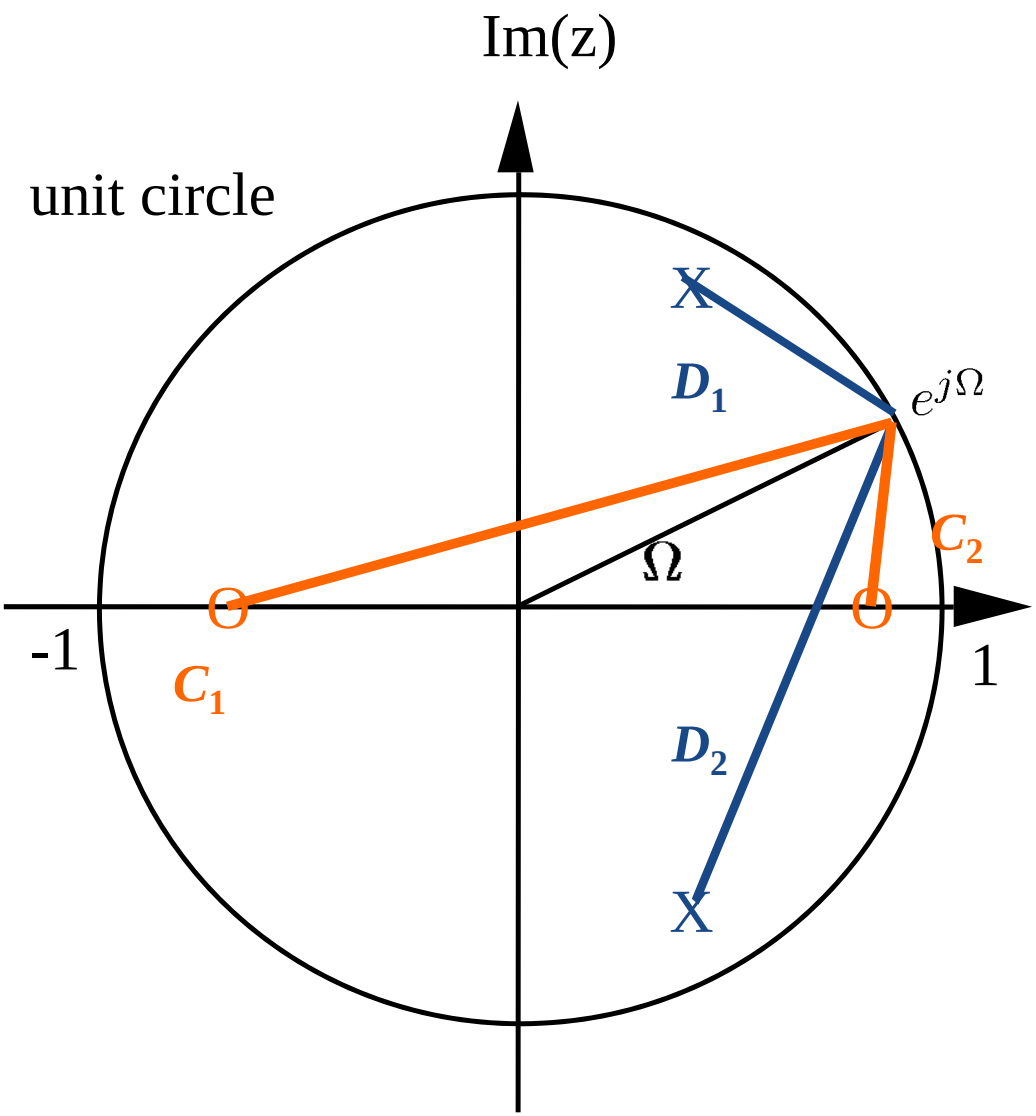


Transfer function:

$$H(z) = b_0 \frac{\Pi_{q=1}^{M}(1 - c_q z^{-1})}{\Pi_{q=1}^{N}(1 - d_q z^{-1})}$$

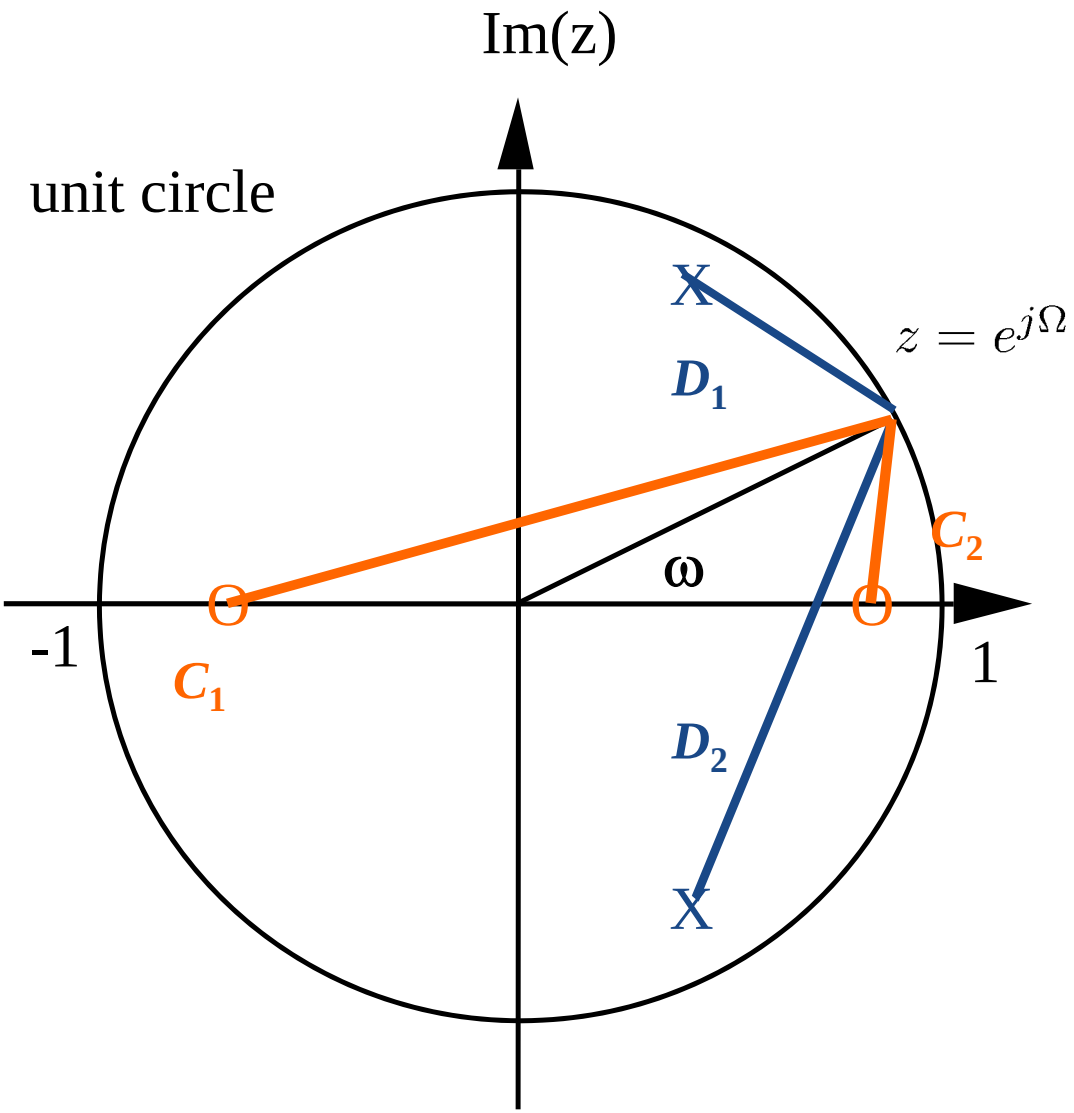
Frequency response:

$$|H(e^{j\Omega})| = b_0 \frac{C_1 C_2}{D_1 D_2}$$



The **magnitude** of the frequency response is given by b_0 times the **product of the distances from the zeros** to $e^{j\Omega}$ **divided by the product of the distances from the poles to** $e^{j\Omega}$

The **phase** response is given by the **sum of the angles from the zeros** to **minus the sum of the angles from the poles** to $e^{j\Omega}$ plus a linear phase term $(M\text{-}N)\Omega$



Thus when $z = e^{j\Omega}$ 'is close to' a pole, the magnitude of the response rises (resonance).

When $z = e^{j\Omega}$ 'is close to' a zero, the magnitude falls (a null).

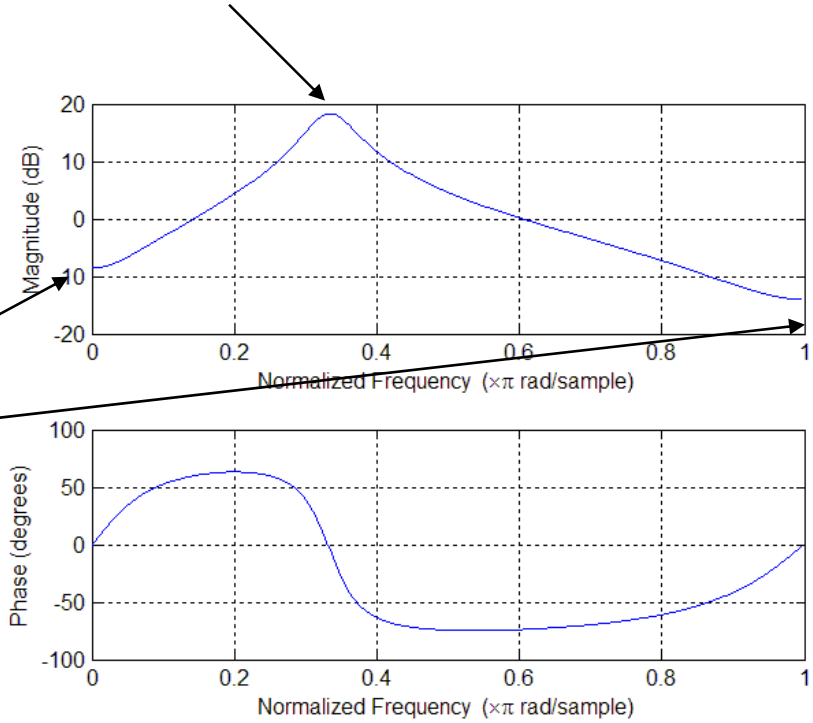
The phase response – more difficult to get “intuition”, but similar principle applies

Calculate frequency response of filter in Matlab:

```
b=[1 -0.1 -0.56];
a=[1 -0.9 0.81];
freqz(b,a)
```

Peak close to pole frequency

Troughs at zero frequencies



More generally we have the frequency response of the filter as:

$$H\left(e^{j\Omega}\right) = \frac{\sum_{k=0}^{M} b_k e^{-jk\Omega}}{1+\sum_{k=1}^{N} a_k e^{-jk\Omega}}$$

and in factorised form:

$$H(e^{j\Omega}) = b_0 \frac{\Pi_{q=1}^{N}(1-c_q e^{-j\Omega})}{\Pi_{q=1}^{M}(1-d_q e^{-j\Omega})} = b_0 \frac{e^{-jN\Omega}\,\Pi_{q=1}^{N}(e^{j\Omega}-c_q)}{e^{-jM\Omega}\Pi_{q=1}^{M}(e^{j\Omega}-d_q)}$$

Taking the complex modulus,

Distance from unit circle to zero

Distance from unit circle to pole

$$|H(e^{j\Omega})| = b_0 \frac{\Pi_{q=1}^{N}|e^{j\Omega}-c_q|}{\Pi_{q=1}^{M}|e^{j\Omega}-d_q|}$$

$$= b_0 \frac{\prod_{q=1}^{N} \text{Distance from } e^{j\Omega} \text{ to } c_q}{\prod_{q=1}^{M} \text{Distance from } e^{j\Omega} \text{ to } d_q}$$

and argument,

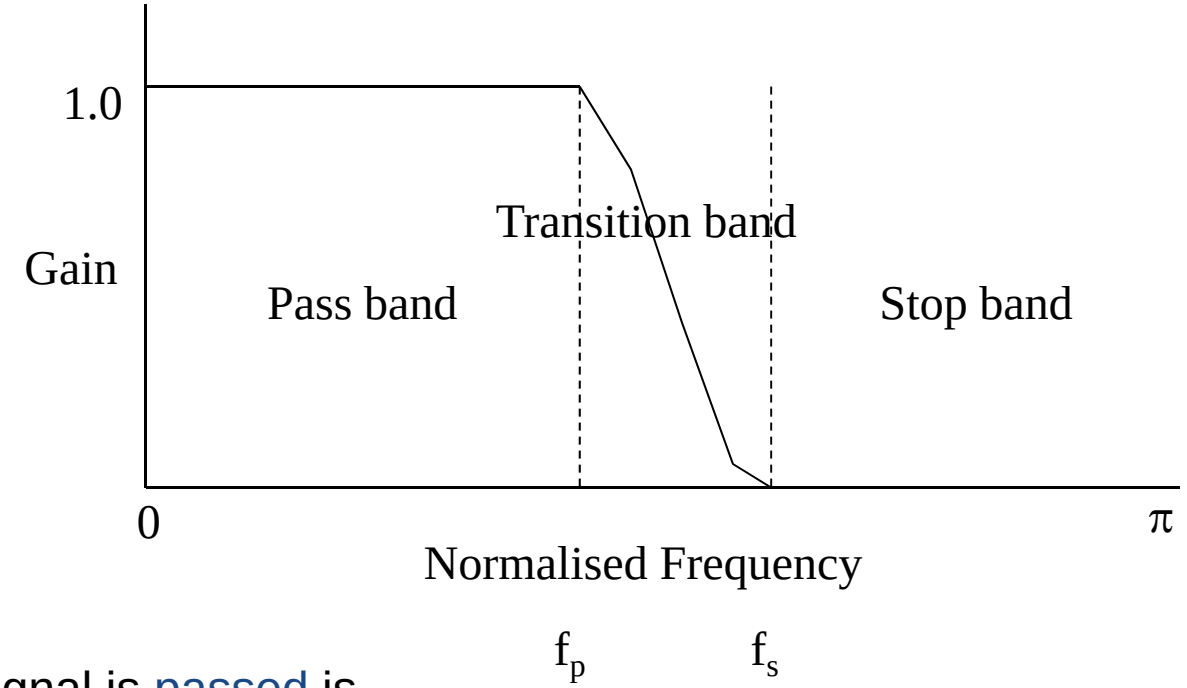
$$\angle(H(e^{j\Omega})) = \Omega(M-N) + \sum_{q=1}^{N} \angle(e^{j\Omega}-c_q) - \sum_{q=1}^{M} \angle(e^{j\Omega}-d_q)$$

Design of Filters

The 4 classical standard frequency magnitude responses are:

Lowpass, Highpass, Bandpass, and Bandstop

Consider e.g. Lowpass:



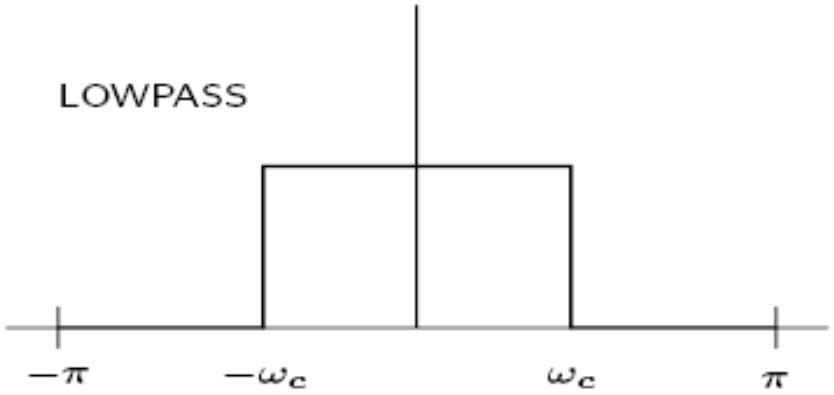
Frequency band where signal is passed is passband

Frequency band where signal is removed is stopband

Ideal Low-pass Filter

- **Low-pass:** designed to pass low frequencies from zero to a certain cut-off frequency and to block high frequencies

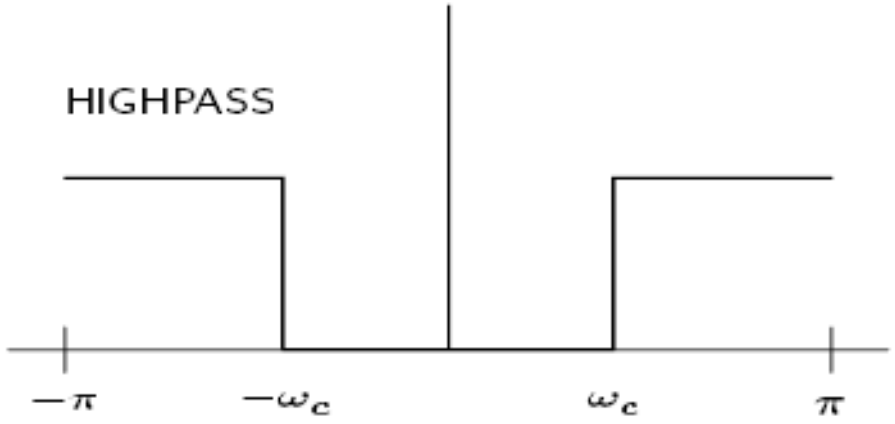
Ideal Frequency Response



Ideal High-pass Filter

- **High-pass:** designed to pass high frequencies from a certain cut-off frequency to π and to block low frequencies

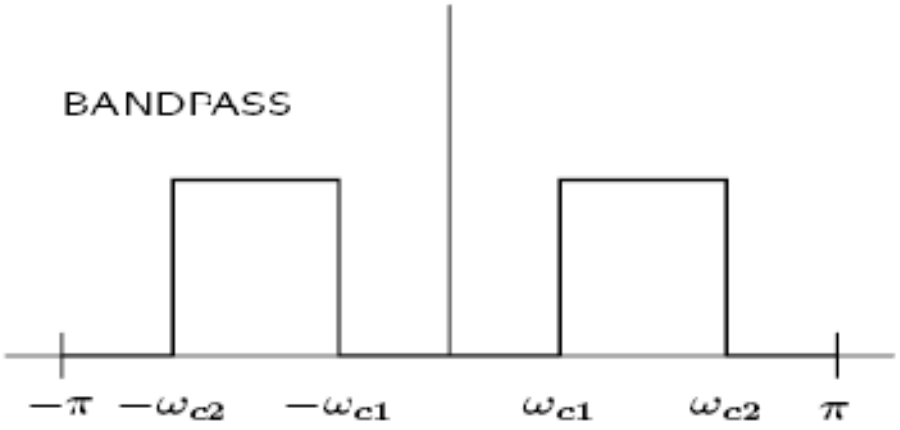
Ideal Frequency Response



Ideal Band-pass Filter

- **Band-pass:** designed to pass a certain frequency range which does not include zero and to block other frequencies

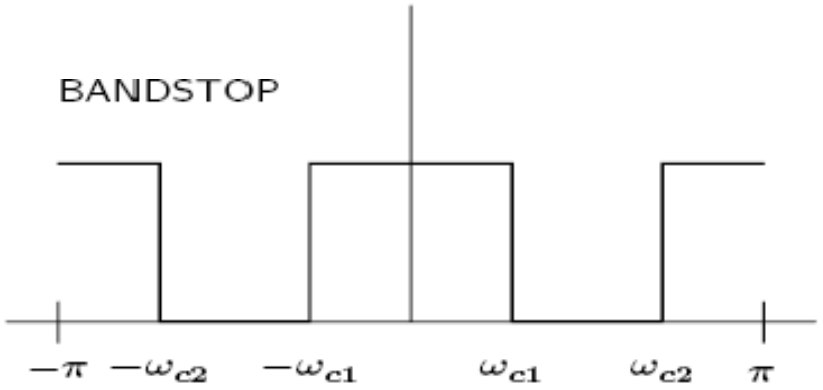
Ideal Frequency Response



Ideal Band-stop Filter

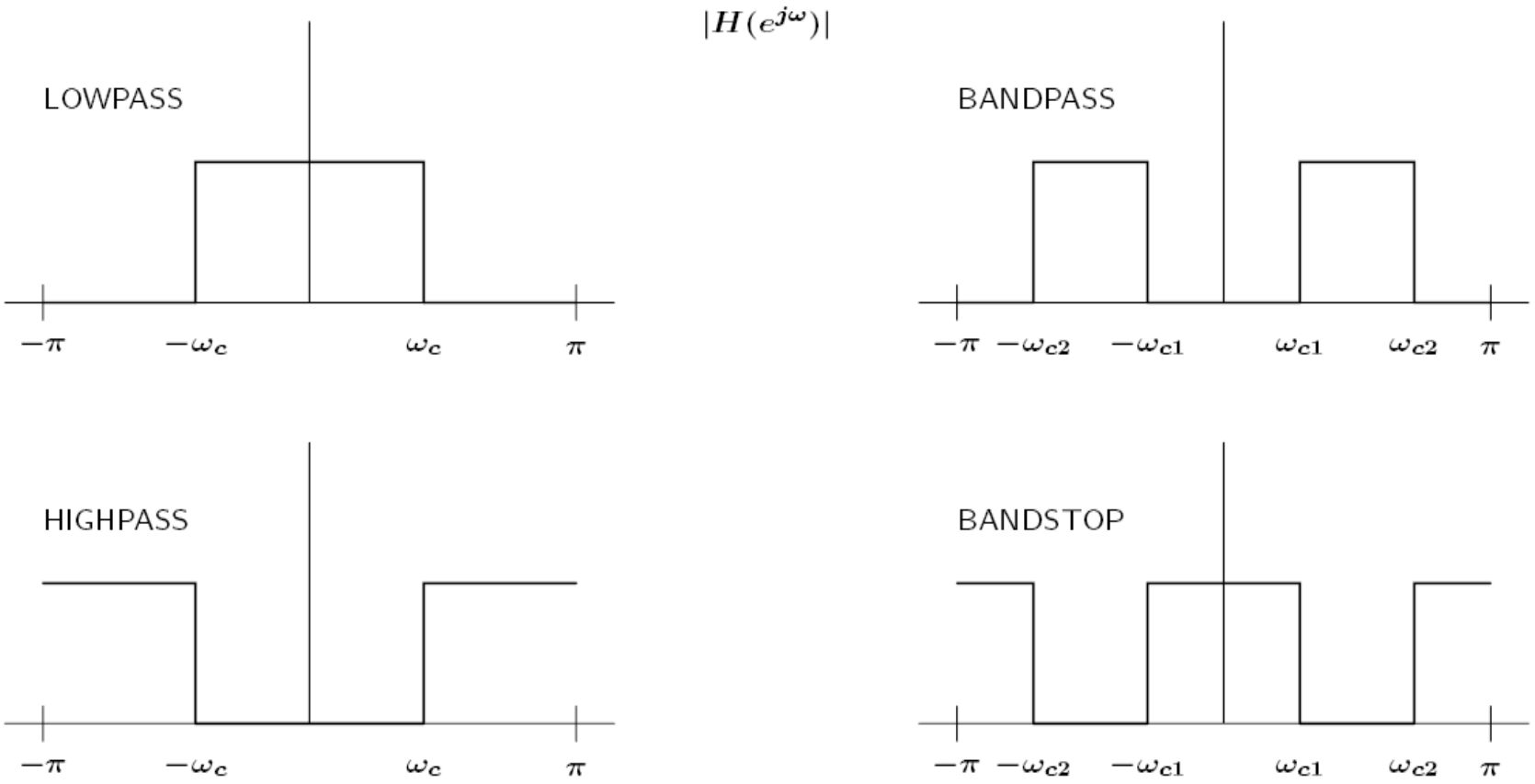
- **Band-stop:** designed to block a certain frequency range which does not include zero and to pass other frequencies

Ideal Frequency Response



Ideal Filters – Magnitude Response

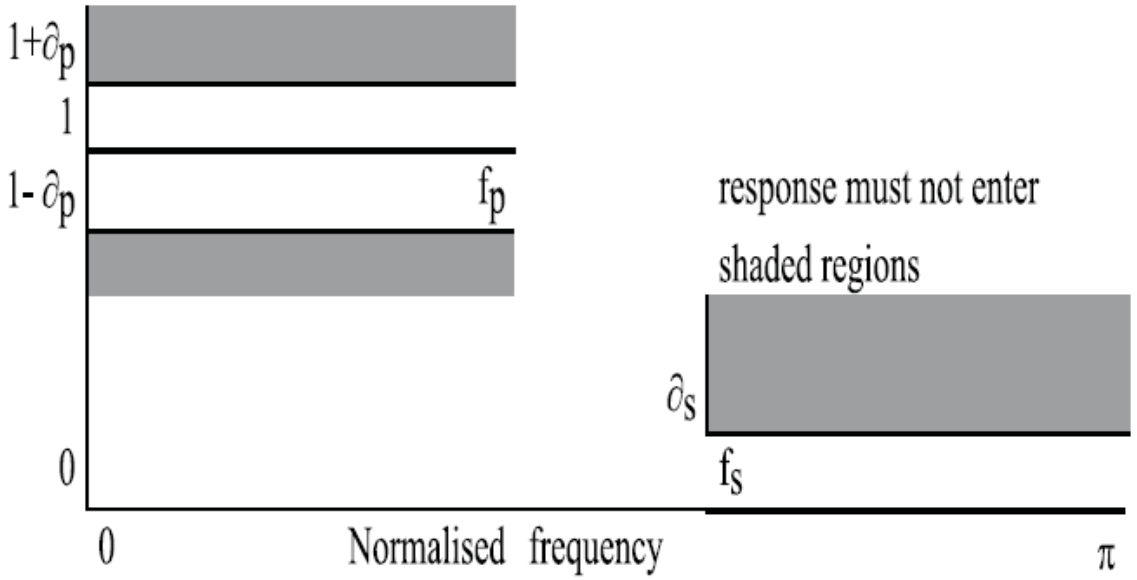
Ideal Filters are usually such that they admit a gain of 1 in a given *passband* (where signal is passed) and 0 in their *stopband* (where signal is removed).



It is impossible to implement the above responses (or any response with finite width constant magnitude sections). Any realisable filter can only approximate it.

[Another requirement for realisability is that the filter must be causal (i.e. $h_n=0$, $n<0$).]

Hence a typical filter specification must specify maximum permissible deviations from the ideal
- a maximum passband ripple ∂p and a maximum stopband amplitude ∂s
(or minimum stopband attenuation) :



These are often expressed in dB:

passband ripple = 20 log10 (1+∂p) dB,

or peak-to-peak passband ripple $\cong$ 20 log10 (1+2∂p) dB;

minimum stopband attenuation = -20 log10 (∂s) dB.

Example: ∂p = 6%:

peak-to-peak passband ripple $\cong$ 20 log10 (1+2∂p) = 1dB;

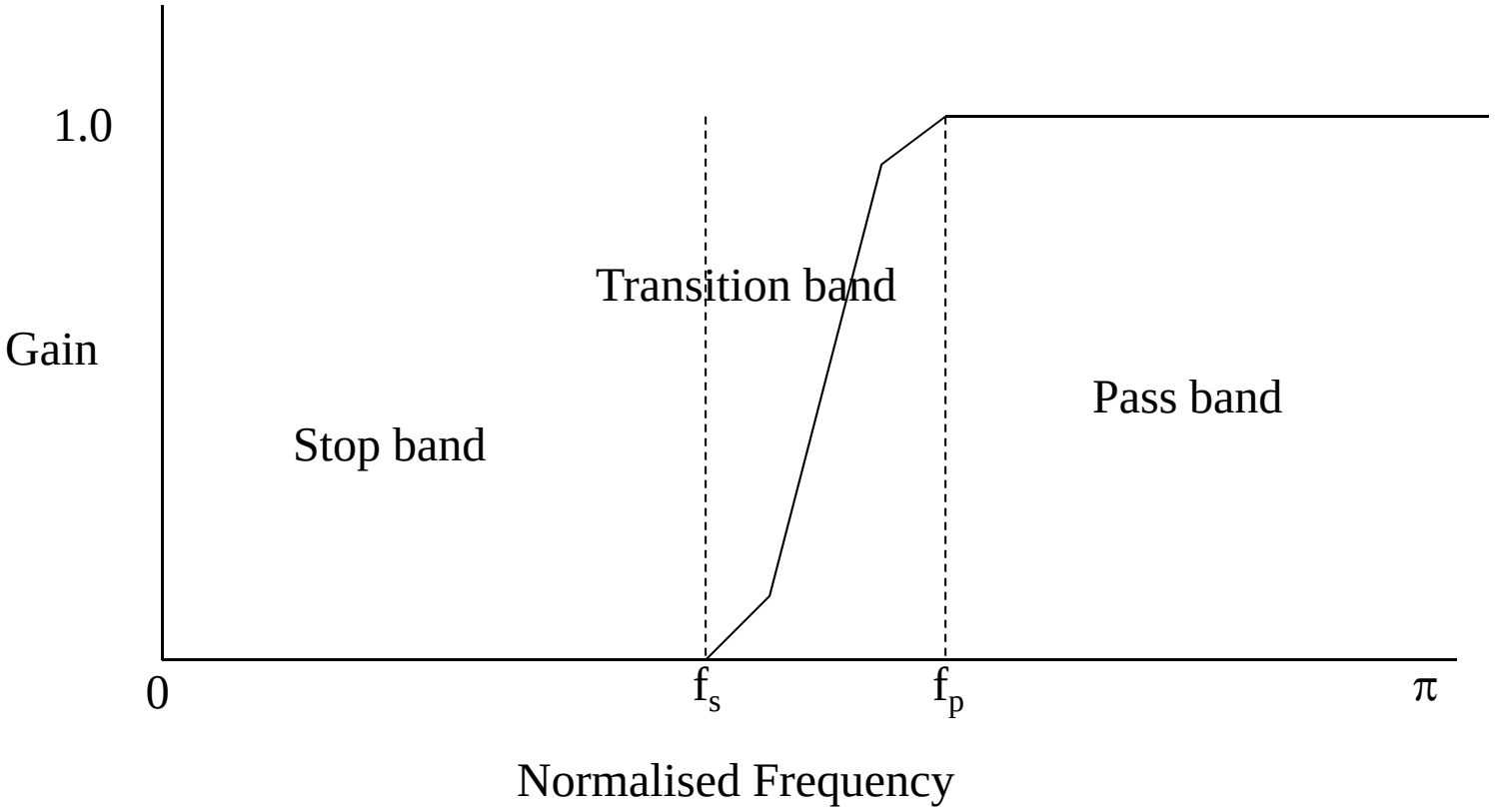
∂s = 0.01:

minimum stopband attenuation = -20 log10 (∂s) = 40dB.

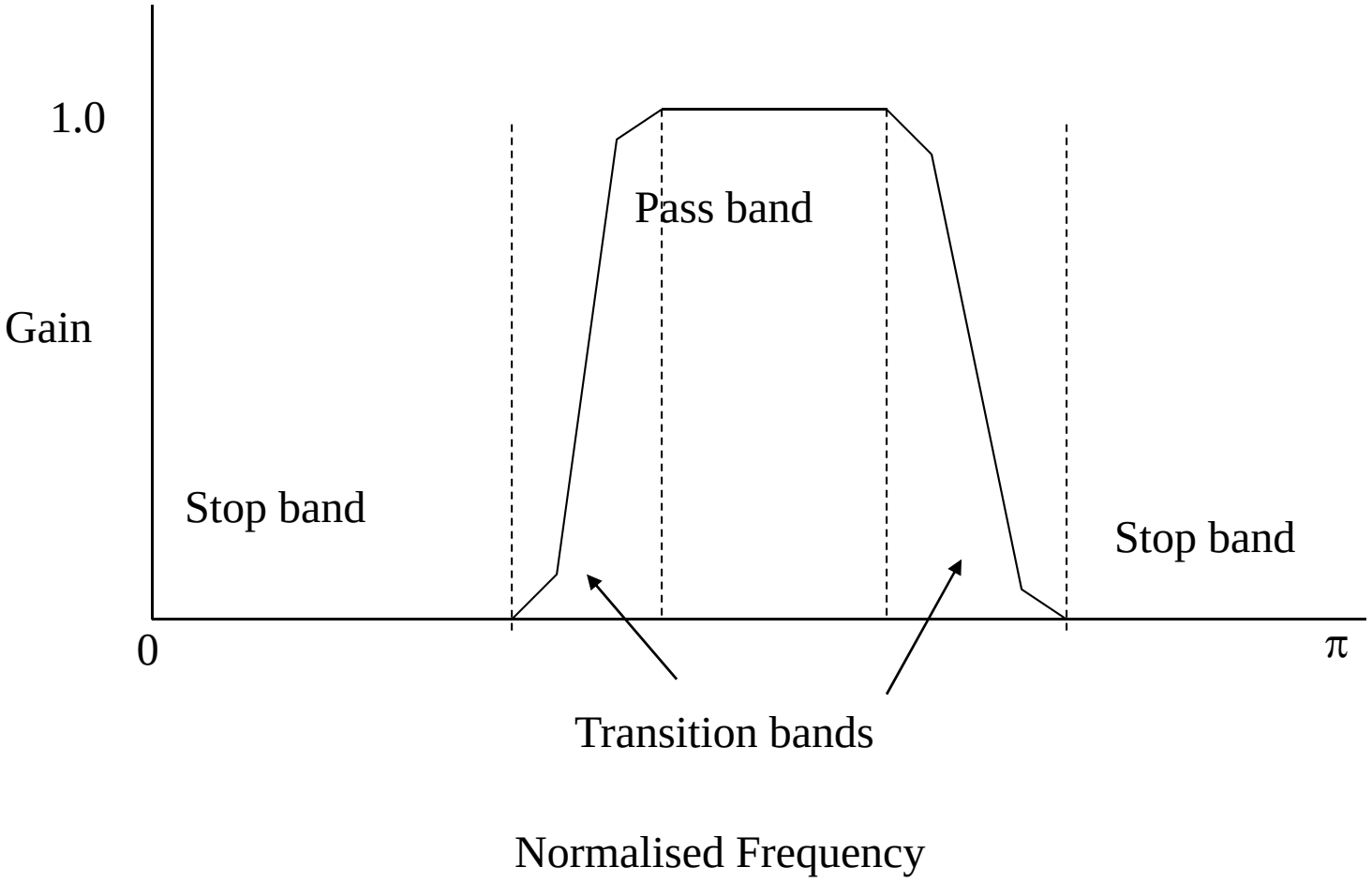
The bandedge frequencies are often called corner frequencies, particularly when associated with specified gain or attenuation (eg gain = -3dB).

Other standard responses:

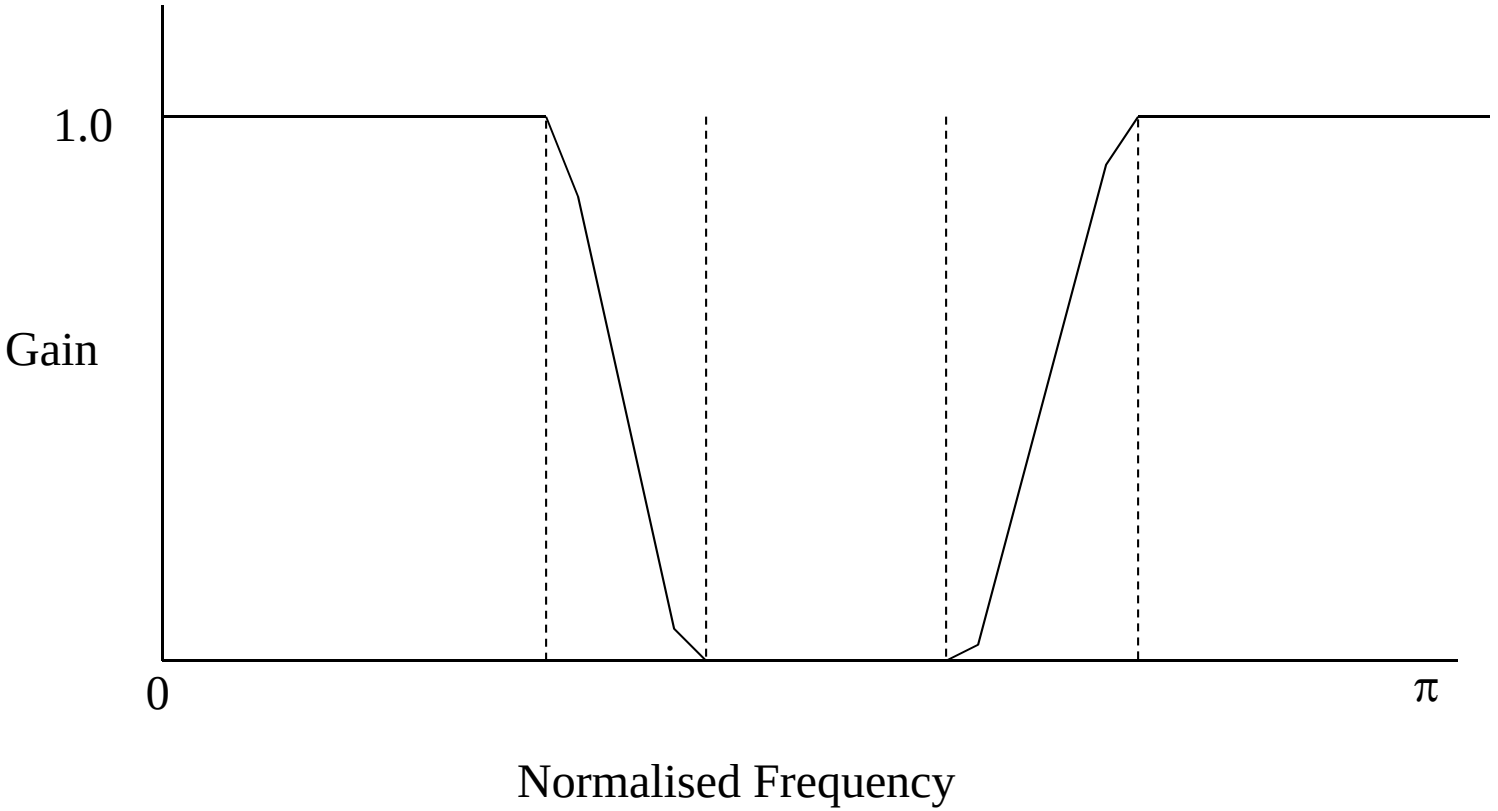
High Pass:



Band Pass:

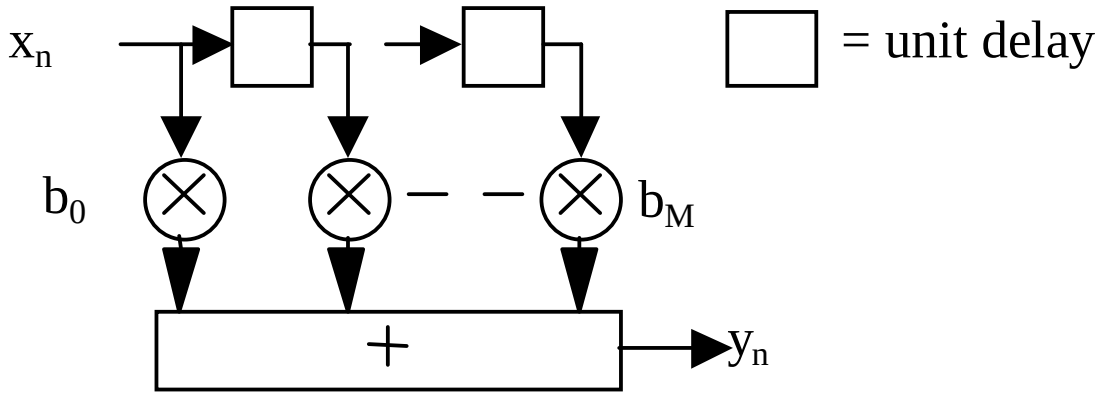

1.0
Gain
Pass band
Stop band
Stop band
0
π
Transition bands
Normalised Frequency

Band Stop:



FIR Filters

The simplest class of digital filters are the Finite Impulse Response (FIR) filters, which have the following structure:



and difference equation:

$$y_n = \sum_{k=0}^{M} b_k x_{n-k}$$

Can immediately obtain the impulse response, with $x_n = \delta_n$

$$h_n = y_n = \sum_{k=0}^{M} b_k \delta_{n-k} = b_n$$

$$[\text{since } \delta_{n-k} \text{ is non-zero only for for } n = k]$$

Hence the impulse response is of finite length M+1, as required

FIR filters also known as feedforward or non-recursive, or transversal

Design of FIR filters

Given the **desired frequency response** D(Ω) of a filter, can compute an appropriate inverse DTFT to obtain its ideal impulse response. Since the coefficients of an FIR filter equate to its impulse response, this would produce an "ideal" FIR filter.

However, this "ideal" impulse is not actually constrained to be of finite length, and it may be non-causal (i.e. have non-zero response at negative time). Somehow we must generate an impulse response which is of limited duration, and causal

In order to obtain the coefficients, simply inverse DTFT the desired response (since impulse response is inverse DTFT of frequency response):

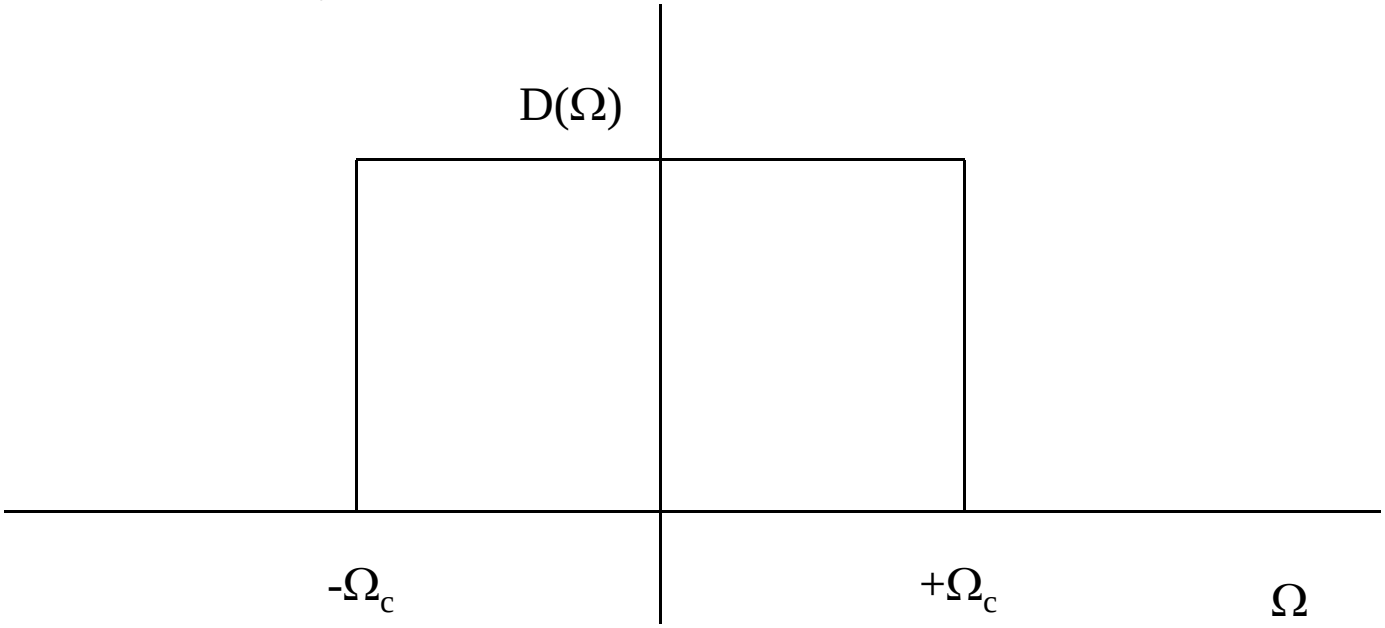
$$d_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(\Omega)\, e^{+jn\Omega}\, d\Omega$$

If the ideal filter coefficients d_n are to be real-valued, then $D(\Omega)$ must be conjugate symmetric, i.e.

$$D(-\Omega) = D^*(\Omega)$$

. We will consider the simplest case, a frequency response which is purely real, and therefore symmetric about zero frequency. For example, consider an ideal lowpass response,

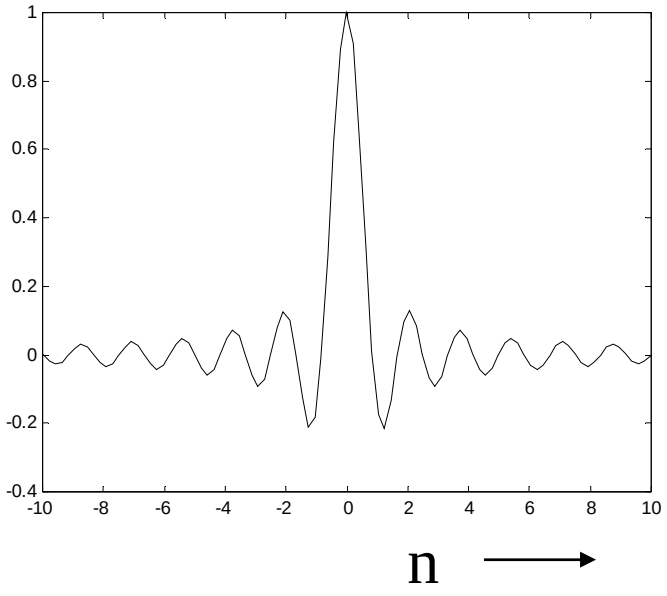
$$\begin{cases} D(\Omega) = 1, & |\Omega| < \Omega_c, \\ D(\Omega) = 0, \Omega_c < |\Omega| < \Omega_c \end{cases}$$



The ideal filter coefficients can in this case be calculated exactly:

$$d_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} D(\Omega)\, e^{+jn\Omega}\, d\Omega = \frac{\Omega_c}{\pi}\frac{\sin(n\Omega_c)}{n\Omega_c}$$

This 'sinc' response is symmetric about sample n=0, and infinite in extent :



To implement an order-M FIR filter, assume we **select only a finite length section** of d_n.

For the sinc response shown above, the best section to select (that is, the one which gives minimum total squared error) is **symmetric about 0**, i.e.

$$h_n = \begin{cases} d_n, & -M/2 \le n \le M/2 \\ 0, & \text{otherwise} \end{cases}$$

[The resulting filter is non-causal, but it can be made causal simply by adding delay.]

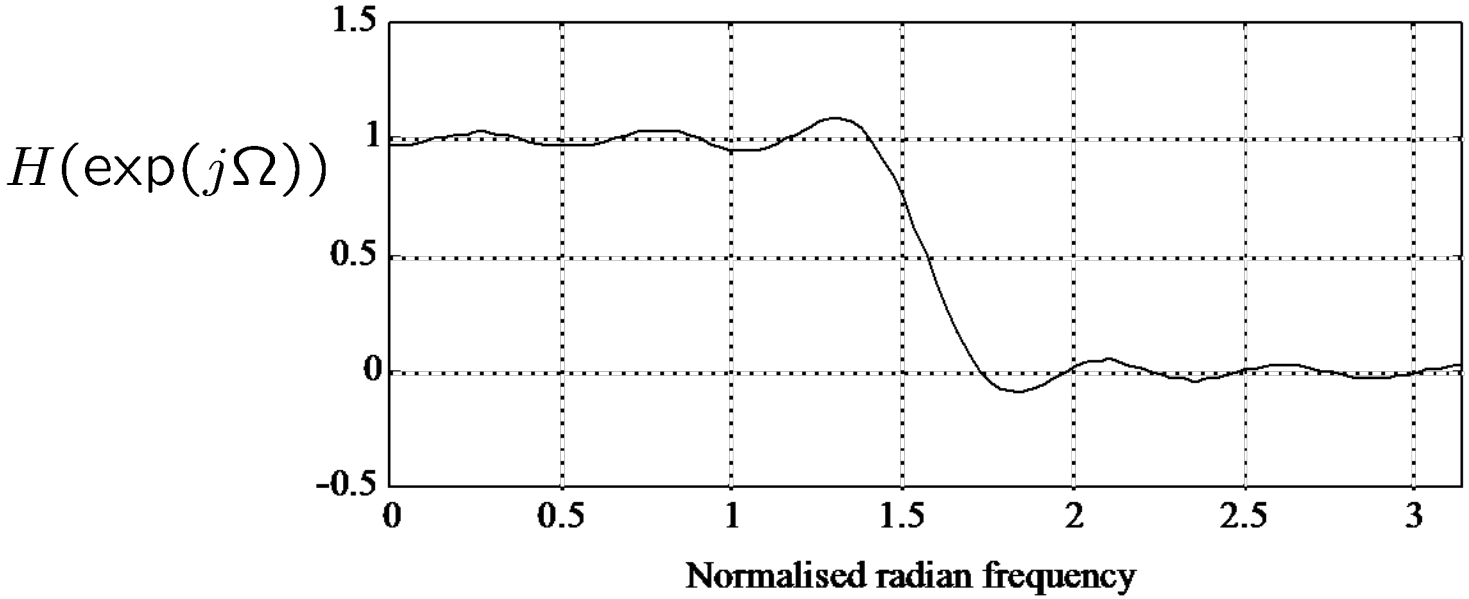
This selection operation is equivalent to multiplying the ideal coefficients by a **rectangular window** extending from -M/2 to M/2.

We can compute the resulting filter **frequency response**, which is a "truncated Fourier series approximation" of D(Ω), given by

$$H(e^{j\Omega}) = \frac{\Omega_c}{\pi} \sum_{k=-M/2}^{+M/2} \frac{\sin(k\Omega_c)}{k\Omega_c} \exp(-jk\Omega)$$

This is illustrated below for the case M=24 (length 25)
and $\Omega_c = \pi/2$ (cut-off frequency = 0.25 x sample frequency).

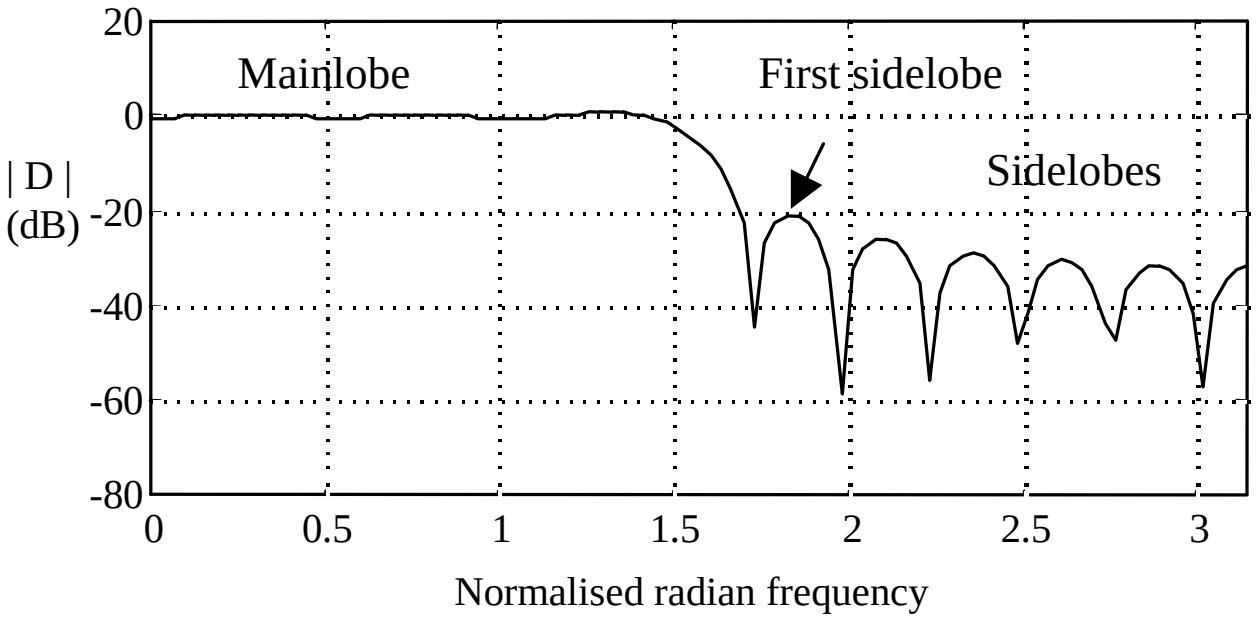
Note the well known **Gibb's phenomenon** (an oscillatory error, increasing in magnitude close to any discontinuities in $D(\Omega)$).



The actual filter would require an added delay of M/2 samples, which does not affect the amplitude response, but introduces a linear phase term to the frequency response.

Now replot the frequency response on a dB amplitude scale.

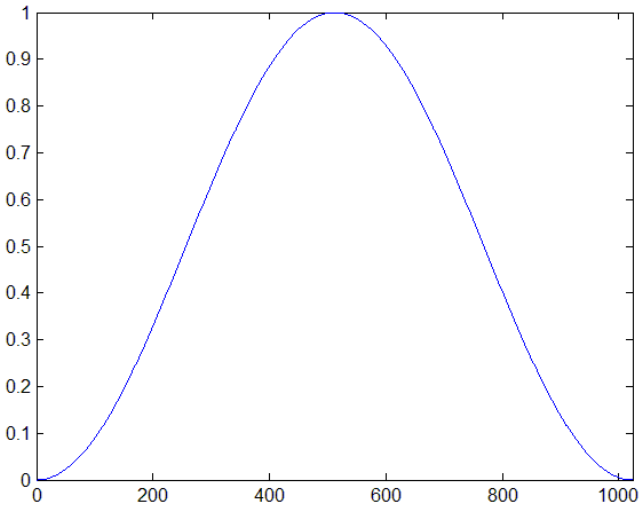
The sidelobes due to the rectangular window can be clearly seen:



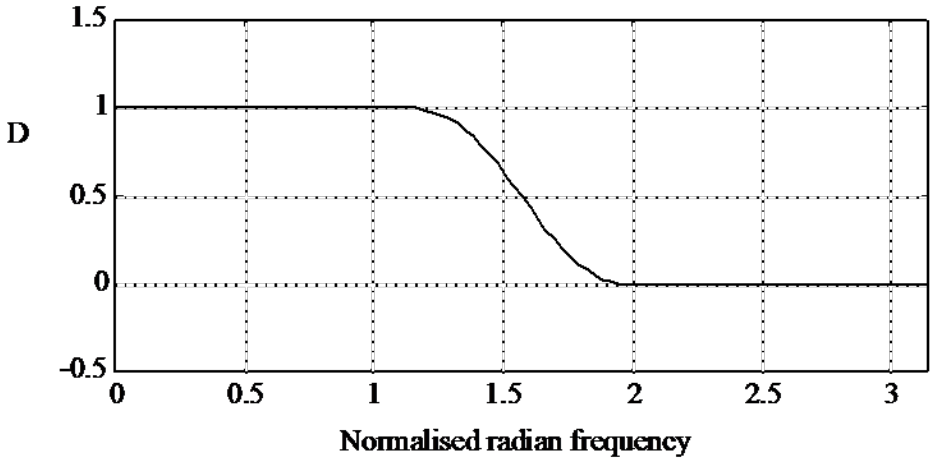
The **high sidelobe level** close to the passband, and the **slow decay** of sidelobe level away from the passband, make this an unsatisfactory response for most purposes.

Use of a window function

A good solution is to create the required finite number of filter coefficients by multiplying the infinite-length coefficient vector d_n by a finite-length **window** w_n with non-rectangular shape, e.g. the **raised cosine (Hann or Hanning) window** function,

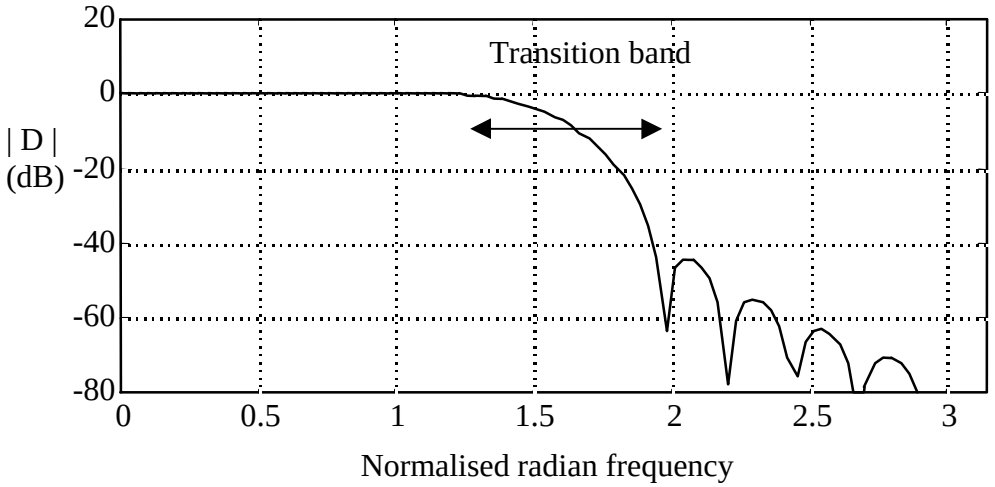


Leading to a much improved frequency response, illustrated below:



The **sidelobes have been greatly reduced**, but the **transition** from passband to stopband has been **widened**. The -3dB frequency has moved from 1.55 rad/sample down to 1.45 rad/sample, illustrating the general point that the choice of window affects the frequencies at which specified gains are achieved.

Again plotting the response on a dB amplitude scale, we have:



The greatly reduced first sidelobe level, more rapid decay of sidelobes, and the broader transition band, are clearly seen.

Analysis

- The desired frequency response and its ideal filter impulse response are:

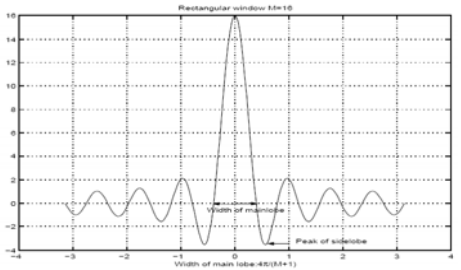
$$D(\Omega) = \begin{cases} 1, & \text{if } |\Omega| \leq \Omega_c \\ 0, & \text{if } \Omega_c < |\Omega| < \pi. \end{cases} \Leftrightarrow d_n = \frac{\Omega_c}{\pi} \frac{\sin \Omega_c n}{\Omega_c n}.$$

- Note that the desired impulse response is of infinite duration and must be truncated.
- We carry this out by multiplication with a suitable window function, w_n.
- It follows that the Fourier transform $H(\exp(j\Omega))$ of the truncated filter $h_n = d_n\, w_n$ is:

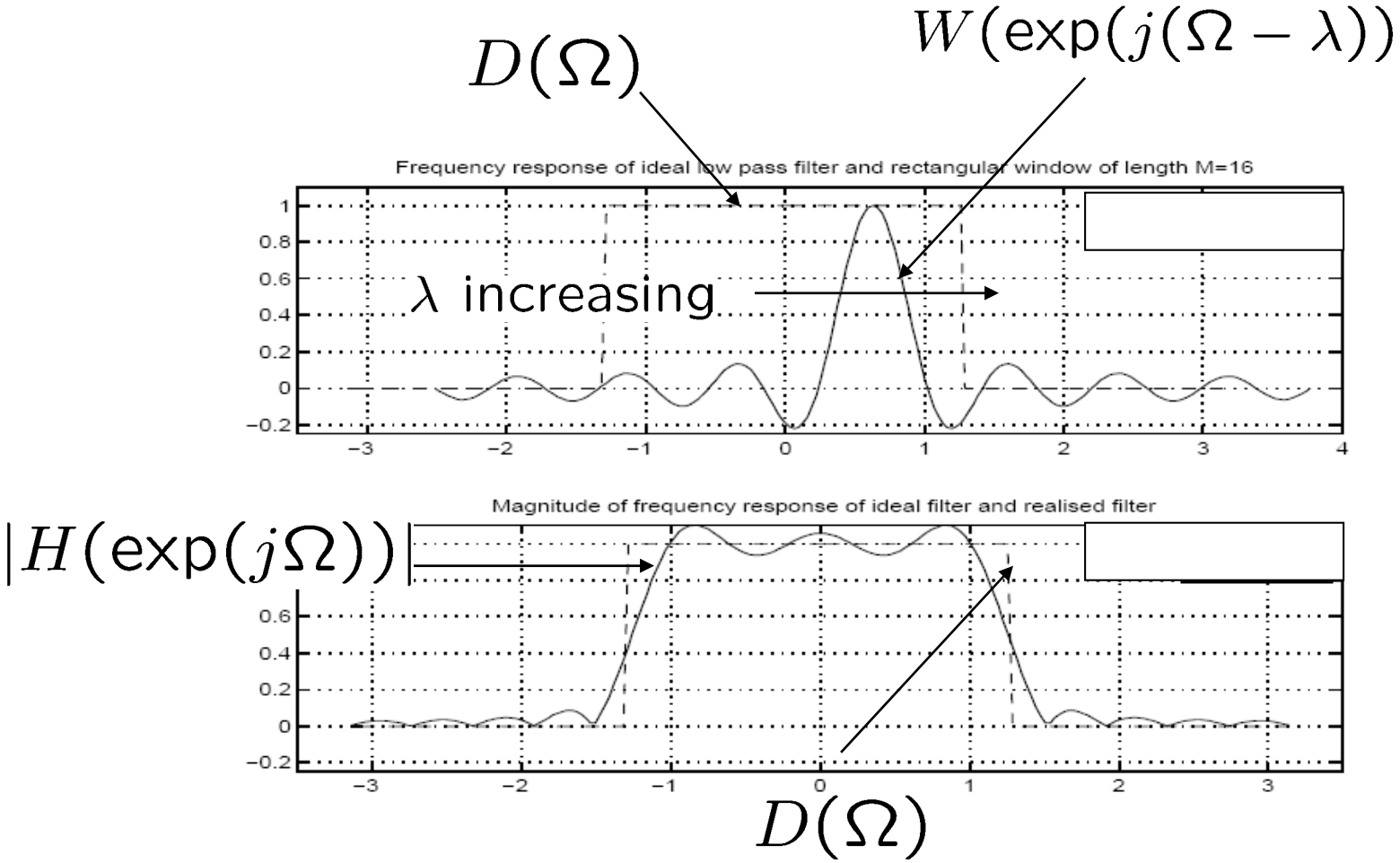
$$H(\exp(j\Omega)) = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(\lambda)\, W(\exp(j(\Omega - \lambda)))\, d\lambda$$

Frequency domain convolution

where $W(\exp(j\Omega))$ is the DTFT of the window function.

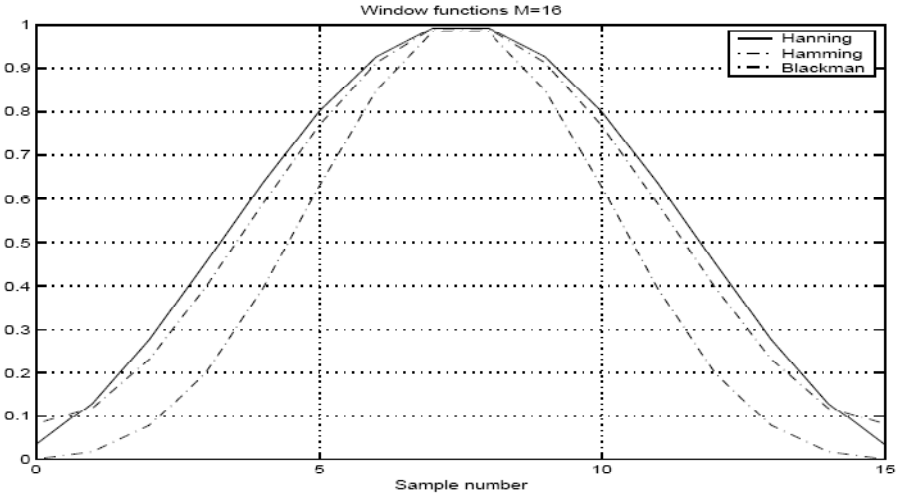


To see the effect of the frequency domain convolution, see the example below, for a rectangle window of length 16:

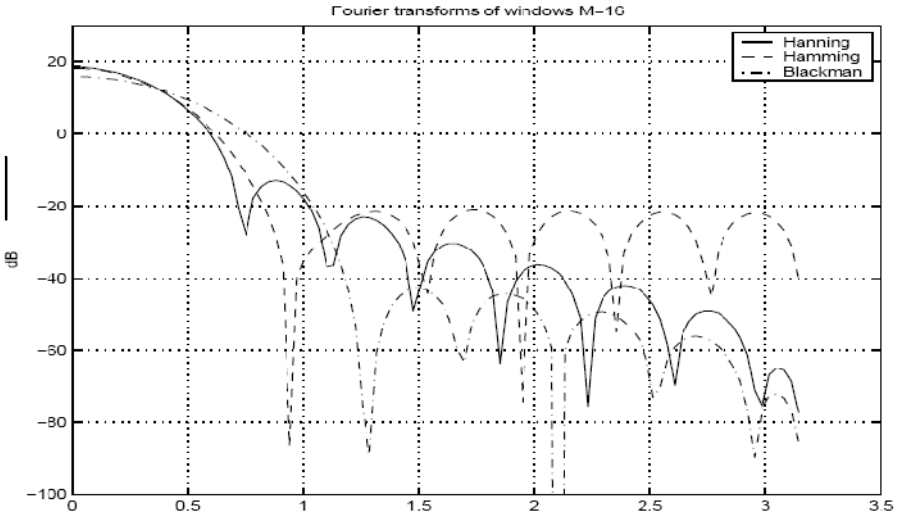


Example window functions:

w_n



$|W(\exp j\Omega)|$



Using the window method for FIR filter design

The window method is conceptually simple and can quickly design filters to approximate a given target response. However, it does not explicitly impose amplitude response constraints, such as passband ripple, stopband attenuation, or 3dB points, so it has to be **used iteratively** to produce designs which meet such specifications.

There are **5 steps** in the window design method for FIR filters.

- Select a suitable window function.
- Specify an 'ideal' response $D(\Omega)$.
- Compute the coefficients of the “ideal” filter.
- Multiply the ideal coefficients by the window function to give the filter coefficients.
- Evaluate the frequency response of the resulting filter, and iterate if necessary

Example:

Obtain the coefficients of an FIR lowpass digital filter to meet these specifications:

passband edge frequency (1dB attenuation)	1.5 kHz
transition width	0.5 kHz
stopband attenuation	>50 dB
sampling frequency	8 kHz

Step 1 – Select a suitable window function

Numerous window functions available – see Matlab command `Window'

Each offer different tradeoffs of transition width, sidelobe level, …

Examples include:

Rectangle

Hann or Hanning

Hamming,

Blackman,

Kaiser - includes a 'ripple control' parameter ß which allows the designer to tradeoff passband ripple against transition width.

Choosing a suitable window function can be done with the aid of published data such as this [taken from "Digital Signal Processing" by Ifeachor and Jervis, Addison-Wesley]:

Name of window function	Transition width/ sample frequency	Passband ripple (dB)	Main lobe relative to side lobe (dB)	Maximum stopband attenuation (dB)
Rectangular	0.9 / N	0.75	13	21
Hann(ing)	3.1/N	0.055	31	44
Hamming	3.3/N	0.019	41	53
Blackman	5.5/N	0.0017	57	74
Kaiser (β=4.54)	2.93/N	0.0274		50
Kaiser (β=8.96)	5.71/N	0.000275		90

However, the above table is worst-case.

For example, in earlier example the use of a Hanning window achieved a main lobe level of –42dB (cf –31 dB) and a normalised transition width of $0.7/2\pi = 0.11$ (cf $3.1/N = 3.1/25 = 0.124$).

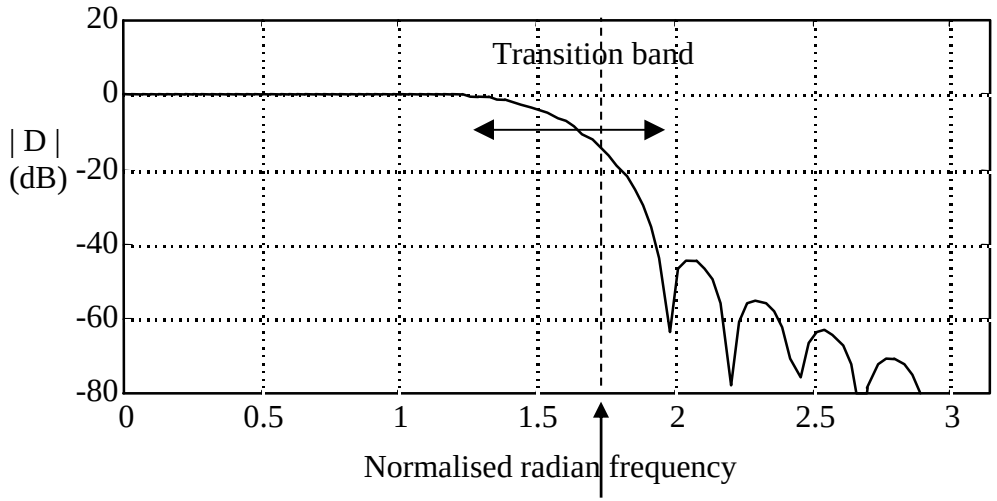
Using the table, the required stopband attenuation (50dB) can probably be obtained by the use of Hamming, Blackman or Kaiser windows.

Try a Hamming window. The table indicates that the transition width (in normalised freq.) is 3.3/N.

Require a normalised transition width of $0.5/8 = 0.0625$, so the required N is 52.8 (ie. N=53).

Step 2 – Specify an 'ideal' response D(Ω)

The smearing effect of the window causes the transition region to spread about the chosen ideal bandedge:



Hence choose an 'ideal' bandedge A which lies in the **middle** of the wanted transition region, i.e. frequency = 1.5+0.5/2 = 1.75 kHz

So, A = 1.75/8 x 2π rad/sample.

Step 3 – Compute the coefficients of the ideal filter

The ideal filter coefficients d_n are given by the inverse **Discrete time** Fourier transform of $D(\Omega)$,

$$d_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(\Omega)\, e^{+jn\Omega}\, d\Omega$$

For our example this can be done analytically, but in general (for more complex *D(Ω)* functions) it will be computed approximately using an N-point **Inverse Fast Fourier Transform** (IFFT).

Given a value of N (choice discussed later), create a sampled version of *D(Ω)*:

$$D[p] = D(2\pi p/N), \;\; p = 0, 1, ...N-1.$$

[Note frequency spacing 2π/N rad/sample]

If the Inverse FFT, and hence the filter **coefficients**, are to be purely **real-valued**, the frequency response must be **conjugate symmetric**:

$$D(-2\pi p/N) = D^*(2\pi p/N) \tag{1}$$

Since the Discrete Fourier Spectrum is also periodic, we see that

$$D(-2\pi p/N) = D(2\pi - 2\pi p/N) = D((N-p)2\pi/N) \tag{2}$$

Equating (1) & (2) we must set

$$D[N-p] = D^*[p] \quad \text{for} \quad p = 1, ..., (N/2-1).$$

- The Inverse DFT of $D[p]$ is an N-sample time domain function, d'_k.
- For d'_k to be an accurate approximation of d_k, N must be made large enough to avoid time-domain aliasing, as illustrated below.
- For our example we start by trying a 64-point DFT. Hence the frequency grid spacing is $2\pi/64$.
- Hence, with cut-off frequency $A = 0.5kHz$ we fill samples 0 to 14 of the discrete spectrum with ones, and also samples 50 to 63; the central part is zeroed.

Matlab code:

```
N=64;              ic = N*1.75/8 + 1;
D=zeros(1,N);
D(1:ic)=ones(1,ic);
D((N-ic+2):N)=ones(1,ic-1);
da=real(ifft(D));
```

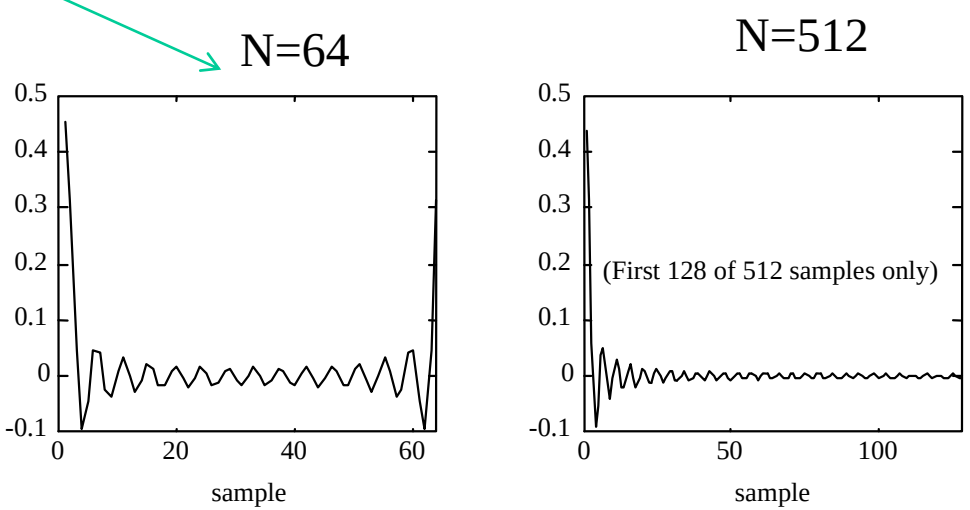


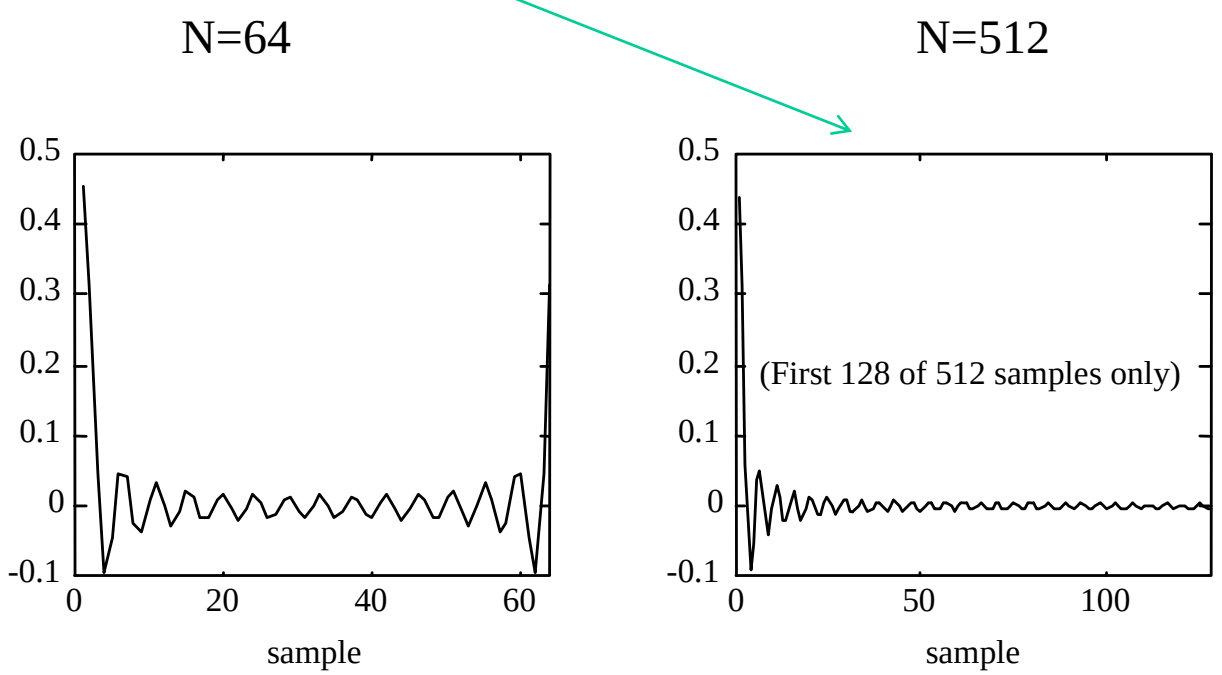
Figure. Approximate ideal responses, N=64 and N=512.

The IFFT gives the LH plot in the above Fig. (repeated below)

Observe the time domain aliasing caused by too short a transform, so try N=512.

Now s = 2π/512, so A/s = 56, so fill elements 0 to 56 and 456 to 511 of the discrete spectrum with ones, and the rest with zeros.

The first 128 of the 512 samples of the new approximate ideal response are shown in the RH plot of the Fig. below



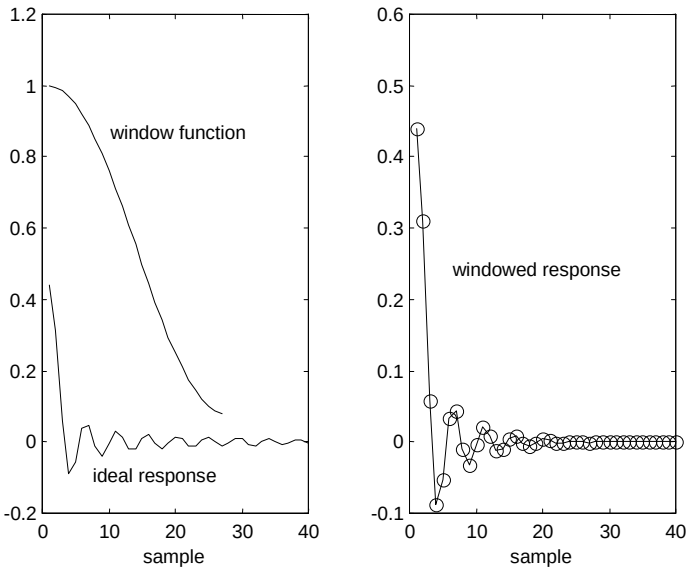
STEP 4 – Multiply to obtain the filter coefficients

The choice of a **zero phase spectrum** resulted in an ideal impulse response **centred on sample 0** of the output, and symmetric

The centre of the window function is therefore to be aligned with sample 0, and the negative-indexed samples of the window are moved up to the top end of the block, by adding N to their indexes. (Remember, the DFS is periodic with period N.)

The Figure below shows, on the left, the first 40 samples of the ideal coefficient array, that is, the central and RH samples of the ideal impulse response.

It also shows the central and RH samples of the window function. The RH plot is their product, the central and RH samples of the resulting filter impulse response.

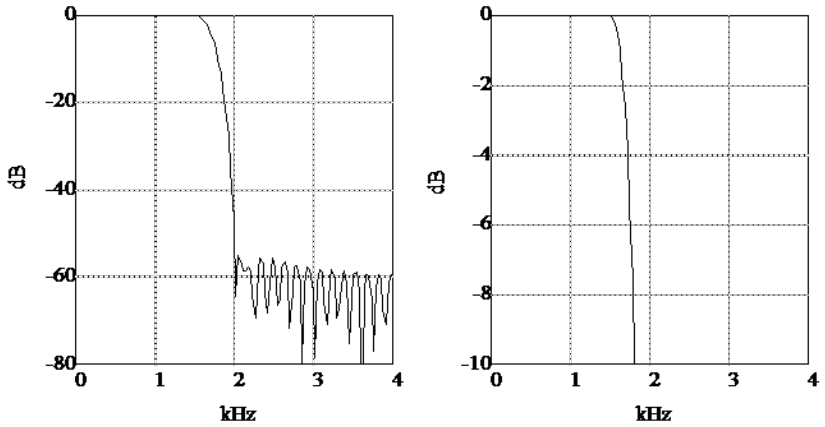


Step 5 – Evaluate the frequency response and iterate

If the resulting filter does not meet the specifications, either adjust D(Ω) (for example, move the band edge) and repeat from step 2, or adjust the filter length and repeat from step 4, or change the window (and filter length) and repeat from step 4.

The frequency response is computed as the DFT of the filter coefficient vector.

In our example this gives the (Discrete Fourier) spectrum shown below.



The specifications are almost met; the LH plot shows the response is not quite -50dB at 2 kHz. However, the RH plot shows that the -1dB frequency is at 1.625 kHz , well above the limit of 1.5kHz. Hence simply reducing the edge frequency A of the ideal response, and repeating the design process, is all that is required in this case to meet the specification.

Performance of the window method of FIR filter design

The window method is **conceptually simple** and easy to use iteratively. It can be used for non-linear-phase as well as linear-phase responses.

However, it is **inflexible**; for example, if a bandpass filter has different upper and lower transition bandwidths, the narrower of them dictates the filter length. There is no independent control over passband ripple and stopband attenuation. The bandedge frequencies are not explicitly controlled by the method.

It has **no guaranteed optimality** - a shorter filter meeting the specifications can almost always be designed.

Matlab implementation of the window method

Matlab has two routines for FIR filter design by the window method, `FIR1` and `FIR2`.

`B = FIR2(N,F,M)` designs an Nth order FIR digital filter and returns the filter coefficients in length N+1 vector `B`.

Vectors `F` and `M` specify the frequency and magnitude breakpoints for the filter such that `PLOT(F,M)` would show a plot of the desired frequency response.

The frequencies in `F` must be between 0.0 < `F` < 1.0, with 1.0 corresponding to half the sample rate. They must be in increasing order and start with 0.0 and end with 1.0.

Note the frequency normalisation used by Matlab, where 1.0 equals **half** the sample rate.

By default `FIR2` uses a Hamming window. Other available windows can be specified as an optional trailing argument. For example, `B = FIR2(N,F,M,bartlett(N+1))` uses a Bartlett window, or `B = FIR2(N,F,M,chebwin(N+1,R))` uses a Chebyshev window. Other windows are computed using routines `Boxcar, Hanning, Bartlett, Blackman, Kaiser` and `Chebwin`.

Design of FIR filters by optimisation

The second method of FIR design considered is **non-linear optimisation.**

First consider a classic algorithm devised by Parks and McClellan, which **designs linear phase (symmetric) filters** or **antisymmetric filters** of any of the standard types.

Digression: Linear Phase Filters

The **frequency response** of the direct form FIR filter may be **rearranged** by **grouping** the terms involving the **first and last** coefficients, the **second and next to last**, etc.:

$$\begin{aligned} H(\exp(j\Omega)) &= \sum_{k=0}^{M} b_k \exp(-jk\Omega) \\ &= b_0 \exp(-j0) + b_M \exp(-jM\Omega) \\ &\quad + b_1 \exp(-j\Omega) + b_{M-1} \exp(-j(M-1)\Omega) \\ &\quad + \ldots \end{aligned}$$

and then taking out a common factor exp(-*jMΩ*/2):

$$H(\exp(j\Omega)) = \exp(-jM\Omega/2) \times \left\{ b_0 \exp(jM\Omega/2) + b(M) \exp(-jM\Omega/2) + b_1 \exp(j(M-2)\Omega/2) + b_{M-1} \exp(-j(M-2)\Omega/2) + \ldots \right\}$$

If the filter length M+1 is odd, then the final term in curly brackets above is the single term $b_{M/2}$, that is the centre coefficient ('tap') of the filter.

Symmetric impulse response: if we put $b_M = b_0$, $b_{M-1} = b_1$, etc., and note that $\exp(j\theta)+\exp(-j\theta) = 2\cos(\theta)$, the frequency response becomes

$$H(\exp(j\Omega)) = 2\exp(-jM\Omega/2) \times \left\{ b_0 \cos(M\Omega/2) + b_1 \cos((M-2)\Omega/2) + \ldots \right\}$$

This is a **purely real** function (sum of cosines) **multiplied by** a **linear phase** term, hence the **response has linear phase**, corresponding to a pure delay of M/2 samples, ie half the filter length.

A similar argument can be used to simplify antisymmetric impulse responses in terms of a sum of sine functions (such filters do not give a pure delay, although the phase still has a linear form $\pi/2$-$m\Omega/2$)

Minimax design of linear phase FIR filters

The filters designed by the Parks and McClellan algorithm have minimised maximum error ("**minimax error**") with respect to a given target magnitude frequency response, i.e. minimise the following error with respect to the filter H:

$$\max_{\Omega} |H_d(\Omega) - H\left(e^{j\Omega}\right)|$$

The method uses an efficient algorithm called the Remez exchange algorithm.

In this algorithm (which copes with an arbitrary number of pass- and stop-bands) the error (i.e. difference between actual and desired frequency response magnitude) is multiplied by a weighting factor which can be different for each band.

The program then minimises the maximum weighted error.

The optimum solution has many frequencies (approximately equal in number to half the filter length) at which the weighted error equals the minimax value:

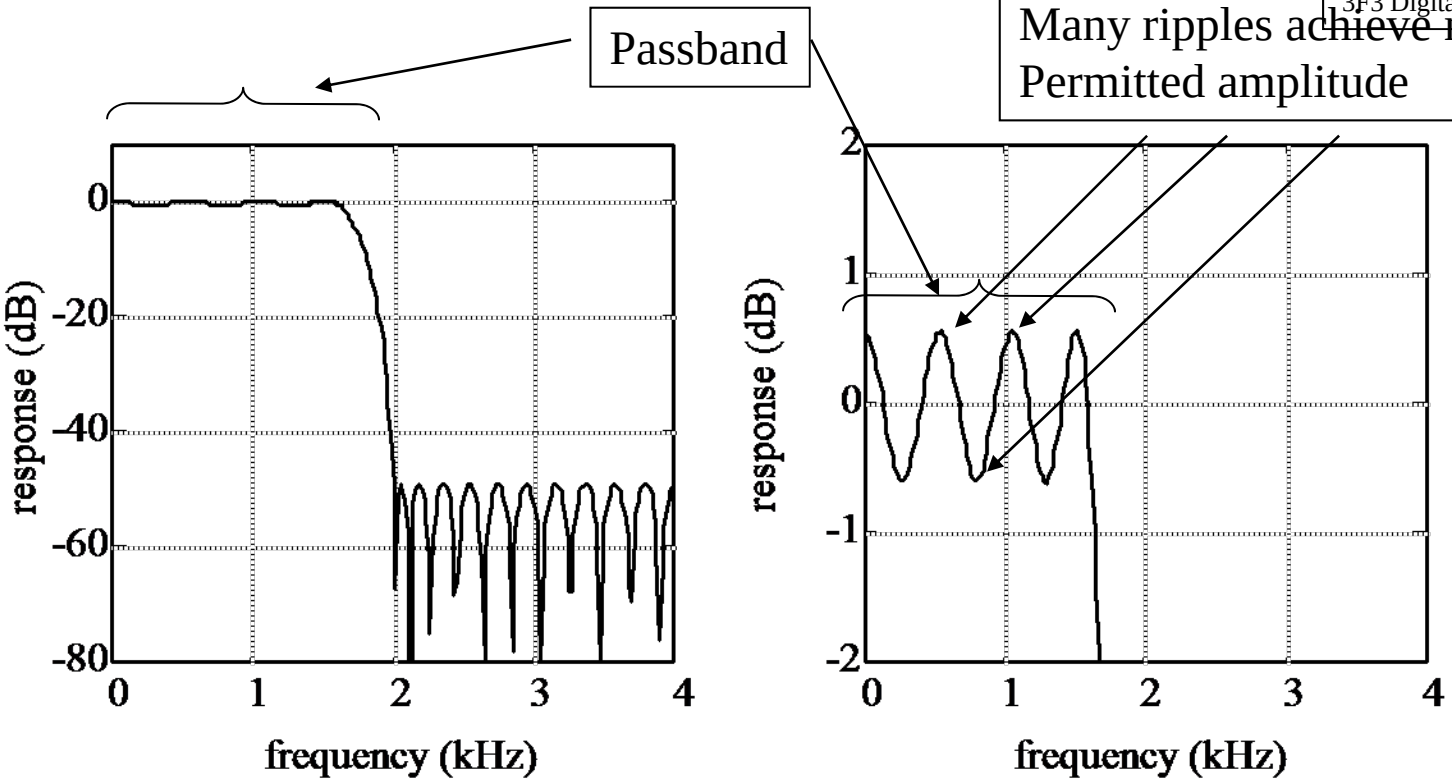


Figure - Overall and passband-only frequency response of length 37 minimax filter

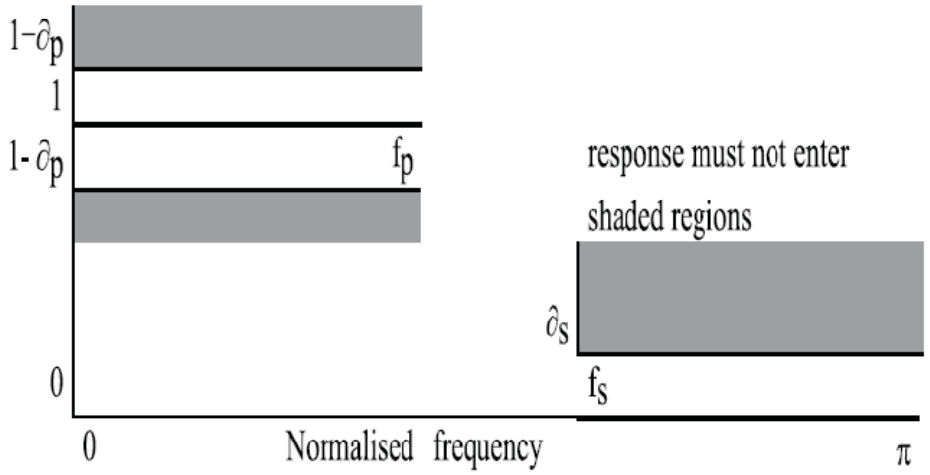
The weights can be determined in advance from a minimax specification.

For example, if a simple lowpass filter has a requirement for the passband gain to be in the range 1-∂p to 1+∂p, and the stopband gain to be less than ∂s, the weightings given to the passband and stopband errors would be ∂s and ∂p respectively.

Formulae are available for estimating the required filter length (eg Ifeachor and Jervis, sec. 6.6.3); these have been devised for specific filter types (lowpass, bandpass), and for narrow transition bandwidths. Unfortunately, they are not reliable for all specifications (as shown in the following example).

The method is used iteratively, adjusting the filter length until the specifications are met.

The detailed algorithm is beyond the (time!) constraints of this module.



Example

Obtain the coefficients of an FIR lowpass digital filter to meet these specifications:

passband edge frequency	1.625 kHz
passband pk-to-pk ripple	<1 dB
transition width	0.5 kHz
stopband attenuation	>50 dB
sampling frequency	8 kHz

The passband ripple corresponds to ±6%, while the stopband attenuation is 0.32%, hence the weighting factors are set to 0.32 and 6.

Using the relevant length estimation formula gives order N=25.8 hence N=26 was chosen, ie length =27. This proved to be substantially too short, and it was necessary to increase the order to 36 (length 37) to meet the specifications.

The Matlab routine is called as follows:

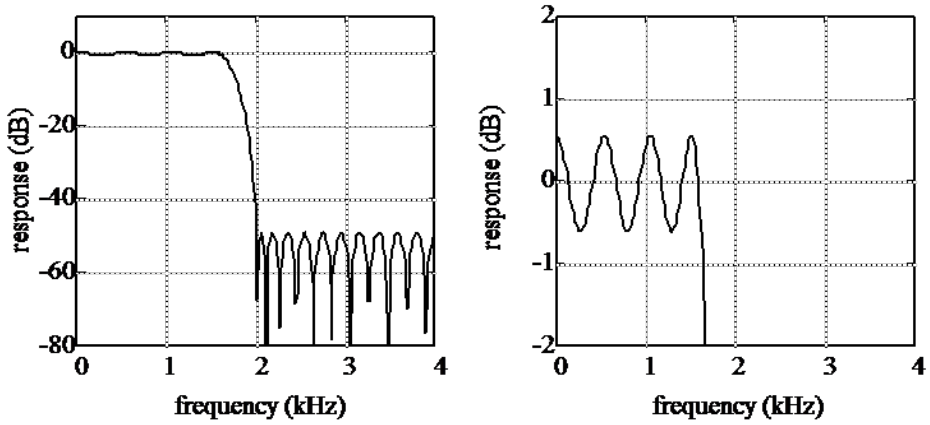
`b = remez(n,f,m)` designs an `n`th order FIR digital filter and returns the filter coefficients in length `n`+1 vector `b`. Vectors `f` and `m` specify the frequency and magnitude breakpoints [as for FIR2]. `b = remez(n,f,m,w)` uses vector `w` to specify weighting in each of the pass or stop bands in vectors `f` and `m`.

Note again the frequency normalisation, where 1.0 equals **half** the sample rate.

The call which finally met this filter specification was:

```
h = remez(36,[0 1.625 2 4]/4, [1 1 0 0], [0.32 6]);
```

The resulting frequency response is as shown previously:

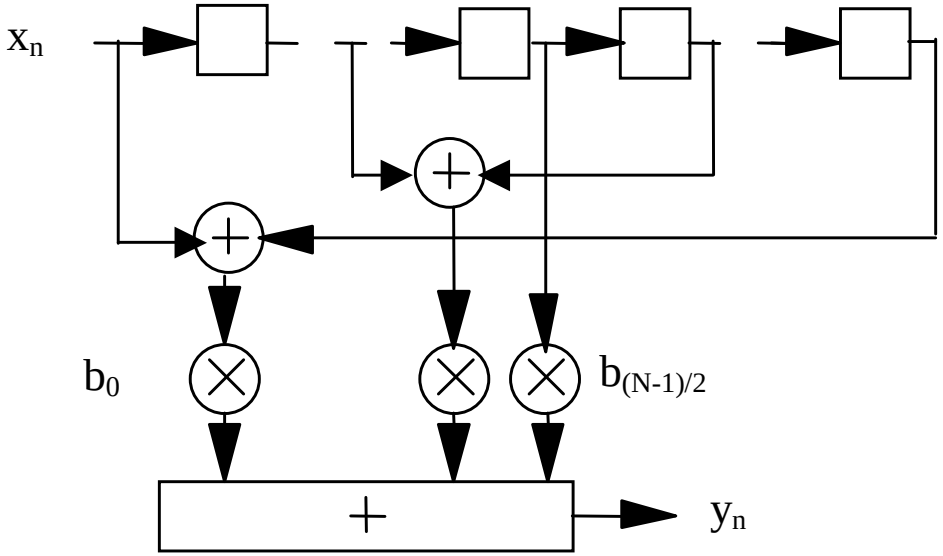


The Parks-McClellan Remez exchange algorithm is widely available and versatile.

Its main apparent limitation is that linear phase in the stopbands is never a real requirement, and in some applications strictly linear phase in the passband is not needed either.

The linear phase filters designed by this method are therefore longer than optimum non-linear phase filters.

However, symmetric FIR filters of length N can be implemented using the folded delay line structure shown below, which uses N/2 (or (N+1)/2) multipliers rather than N,so the longer symmetric filter may be no more computationally intensive than a shorter non-linear phase one.



Further options for FIR filter design

More general non-linear optimisation (least squared error or minimax) can of course be used to design linear or non-linear phase FIR filters to meet more general frequency and/or time domain requirements.

Matlab has suitable optimisation routines.

IIR filter design

To give an Infinite Impulse Response (IIR), a filter must be recursive, that is, incorporate feedback. (But recursive filters are not *necessarily* IIR). The terms "Recursive" or "IIR" filter are used to describe filters with **both** feedback and feedforward terms.

There are two classes of method for designing IIR filters:

(i) generation of a digital filter from an analogue prototype,

(ii) direct non-linear optimisation of the transfer function.

The most useful method in practice is the bilinear transform.

Design of an IIR transfer function from an analogue prototype

Analogue filter designs are represented as Laplace-domain (s-domain) transfer functions. The following methods of generating a digital filter from the analogue prototype are **not** much used:

- Impulse invariant design - The digital filter impulse response equals the sampled impulse response of the analogue filter. **But** the resulting frequency response may be significantly different (due to aliasing).
- Step invariant design – As above but step responses are equal. Used in control system analysis.
- Ramp invariant design – As above but ramp responses are equal.
- Forward difference (Euler) – resulting digital filter may be unstable.
- Backward difference.

The most useful method in practice is the *bilinear transform.*

Properties of the bilinear transform

The bilinear transform produces a digital filter whose **frequency response** has the same characteristics as the frequency response of the analogue filter (but its impulse response may then be quite different).

There are excellent design procedures for analogue prototype filters, so it is sensible to utilise the analogue technology for digital design.

We define the bilinear transform (also known as Tustin's transformation) as the substitution:

$$s = \psi(z) = \frac{1 - z^{-1}}{1 + z^{-1}}$$

•Note 1. Although the ratio could have been written (z-1)/(z+1), that causes unnecessary algebra later, when converting the resulting transfer function into a digital filter;

•Note 2. In some sources you will see the factor (2/T) multiplying the RHS of the bilinear transform; this is an optional scaling, but it cancels and does not affect the final result.

To derive the properties of the bilinear transform, solve for z, and put s = a+jω:

$$z = \frac{1+s}{1-s} = \frac{1+a+j\omega}{1-a-j\omega};\ \text{hence}\ |z|^2 = \frac{(1+a)^2+\omega^2}{(1-a)^2+\omega^2}$$

Look at two important cases:

1. The imaginary axis, i.e. a=0. This corresponds to the boundary of stability for the analogue filter's poles.

   With a=0, we have

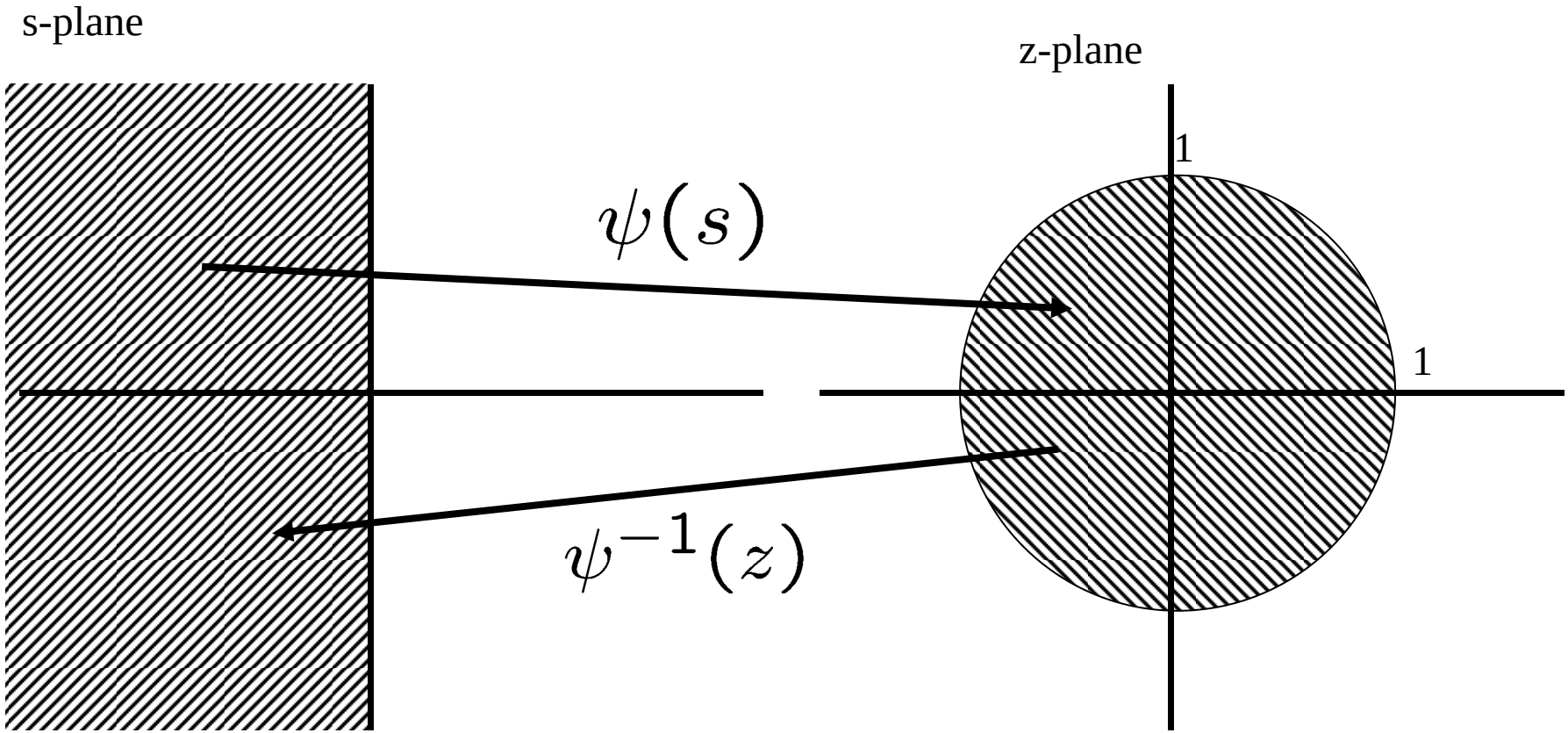
$$|z|^2 = \frac{(1+0)^2+\omega^2}{(1-0)^2+\omega^2} = 1$$

   Hence, the imaginary (frequency) axis in the s-plane maps to the unit circle in the z-plane

2. With a<0, i.e. the left half-plane in the s-plane we have

$$|z|^2 = \frac{(1+a)^2+\omega^2}{(1-a)^2+\omega^2} =< 1, \quad (a<0)$$

Thus we conclude that the bilinear transform maps the Left half s-plane onto the interior of the unit circle in the z-plane:



This property will allow us to obtain a suitable frequency response for the digital filter, and also to ensure the stability of the digital filter.

$$s = \psi(z) = \frac{1 - z^{-1}}{1 + z^{-1}}; \quad z = \psi^{-1}(s) = \frac{1 + s}{1 - s}$$

Stability of the filter

Suppose the analogue prototype H(s) has a stable pole at a+jω, i.e.

$$H(a + j\omega) \to \infty, \;\; a < 0$$

Then the digital filter $\hat{H}(z)$ is obtained by substituting $s = \psi(z)$,

$$\hat{H}(z) = H\left(\psi(z)\right)$$

Since H(s) has a pole at a+jω, $H\left(\psi(z)\right)$ has a pole at $\psi^{-1}(a + j\omega)$ because

$$\hat{H}(\psi^{-1}(a + j\omega)) = H(\,\psi(\,\psi^{-1}(a + j\omega)\,)) = H(a + j\omega) \to \infty$$

However, we know that $\psi^{-1}(a + j\omega)$lies within the unit circle. Hence the filter is *guaranteed stable* provided H(s) is stable.

$$s = \psi(z) = \frac{1 - z^{-1}}{1 + z^{-1}}; \quad z = \psi^{-1}(s) = \frac{1 + s}{1 - s}$$

Frequency Response of the Filter

The frequency response of the analogue filter is

$$H(j\omega)$$

The frequency response of the digital filter is

$$\begin{aligned}\hat{H}\left(\exp(j\Omega)\right) &= H\left(\psi(\exp(j\Omega))\right) \\ &= H\left(j\tan(\Omega/2)\right)\end{aligned}$$

$$\begin{aligned}\psi(\exp(j\Omega)) &= \frac{1 - \exp(-j\Omega)}{1 + \exp(-j\Omega)} \\ &= \frac{\exp(-j\Omega/2)(\exp(j\Omega/2) - \exp(-j\Omega/2))}{\exp(-j\Omega/2)(\exp(+j\Omega/2) + \exp(-j\Omega/2))} \\ &= \frac{j\sin(\Omega/2)}{\cos(\Omega/2)} \\ &= j\tan(\Omega/2)\end{aligned}$$

Hence we can see that the frequency response is *warped* by a function

$$\omega = \tan(\Omega/2)$$

Analogue Frequency

Digital Frequency

Hence the BLT preserves the following important features of H(jω):

(1) the ω↔Ω mapping is monotonic, and

(2) ω = 0 is mapped to Ω = 0, and ω = ∞ is mapped to Ω = π (half the sampling frequency). Thus, for example, a lowpass response that decays to zero at ω = ∞ produces a lowpass digital filter response that decays to zero at Ω = π.

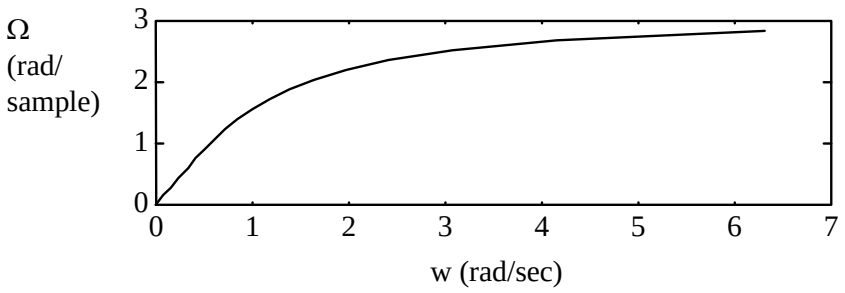


Figure - Frequency warping

If the frequency response of the analogue filter at frequency ω is H(jω), then the frequency response of the digital filter at the corresponding frequency Ω = 2 arctan(ω) is also H(jω). Hence -3dB frequencies become -3dB frequencies, minimax responses remain minimax, etc.

Design using the bilinear transform

The steps of the bilinear transform method are as follows:

1. "Warp" the digital critical (e.g. bandedge or "corner") frequencies Ω_i , in other words compute the corresponding analogue critical frequencies $\omega_i = \tan(\Omega_i/2)$.

2. Design an analogue filter which satisfies the resulting filter response specification.

3. Apply the bilinear transform to the s-domain transfer function of the analogue filter to generate the required z-domain transfer function.

Example – Bilinear Transform

Design a first order lowpass digital filter with -3dB frequency of 1kHz and a sampling frequency of 8kHz

Consider the first order analogue lowpass filter

$$H(s) = \frac{1}{1 + (s/\omega_C)}$$

which has a gain of 1 (0dB) at zero frequency, and a gain of -3dB (= $\sqrt{0.5}$) at ω_c rad/sec (the "cutoff frequency ").

First calculate the normalised digital cutoff frequency:

$$\Omega_c = \frac{1kHz}{8kHz} 2\pi = \pi/4$$

Calculate the equivalent pre-warped analogue filter cutoff frequency:

$$\omega_c = \tan(\Omega_c/2) = \tan(\pi/8) = 0.4142 rad.s^{-1}$$

Apply Bilinear Transform:

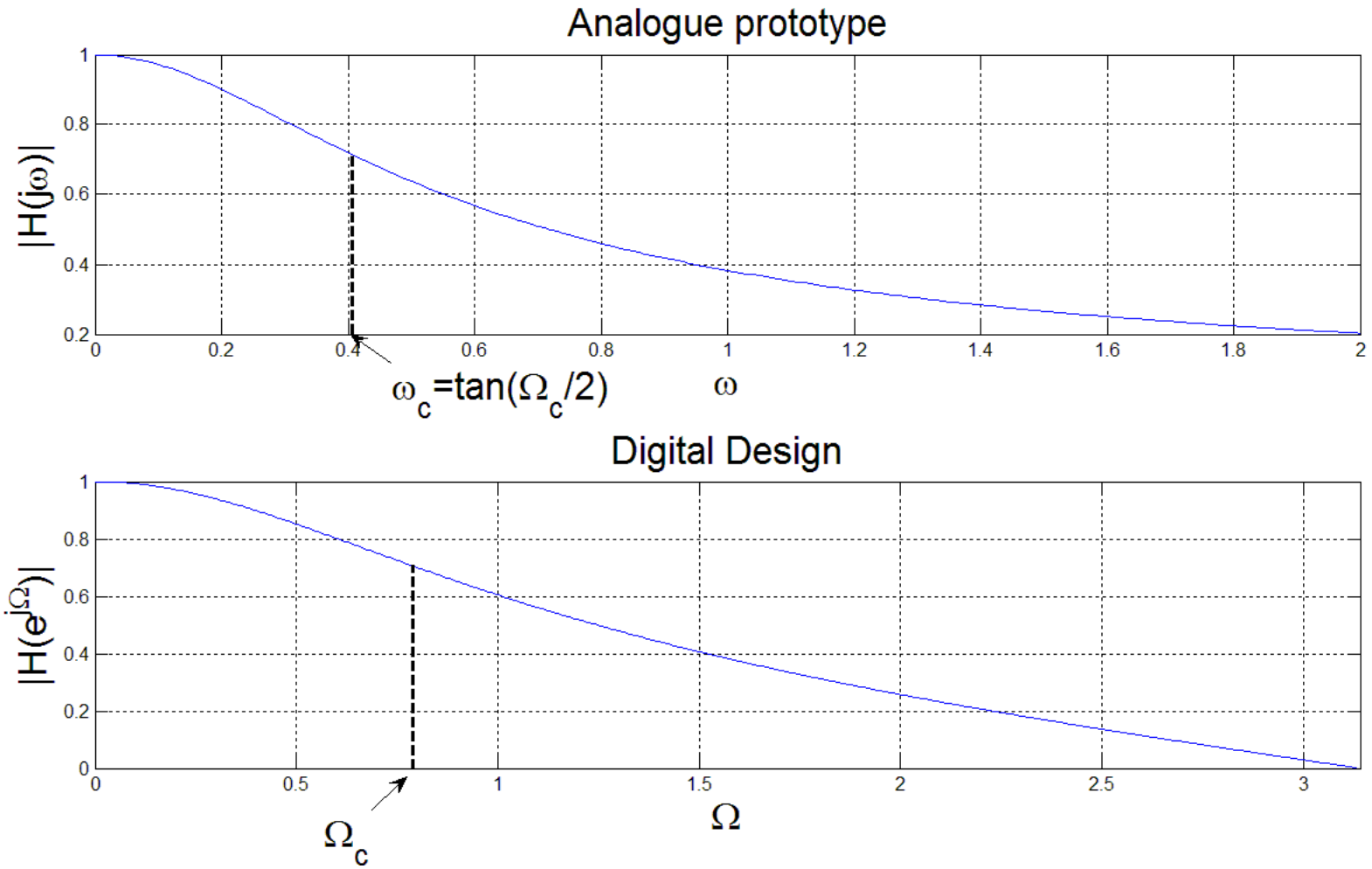
$$\begin{aligned}\hat{H}(z) &= H\left(\psi(z)\right) \\ &= \frac{1}{1+\psi(z)/\omega_c} \\ &= \frac{1}{1+\frac{1-z^{-1}}{1+z^{-1}}/\omega_c} \\ &= \frac{\omega_c(1+z^{-1})}{\omega_c(1+z^{-1})+(1-z^{-1})} \\ &= \frac{\omega_c(1+z^{-1})}{(\omega_c+1)+(\omega_c-1)z^{-1}} \\ &= \frac{0.2929(1+z^{-1})}{1-0.4142z^{-1}}\end{aligned}$$

Normalise to unity for recursive implementation

Keep 0.2929 factorised to save one multiply

i.e. as a direct form implementation:

$$y_n = 0.4142y_{n-1} + 0.2929\left(x_n + x_{n-1}\right)$$



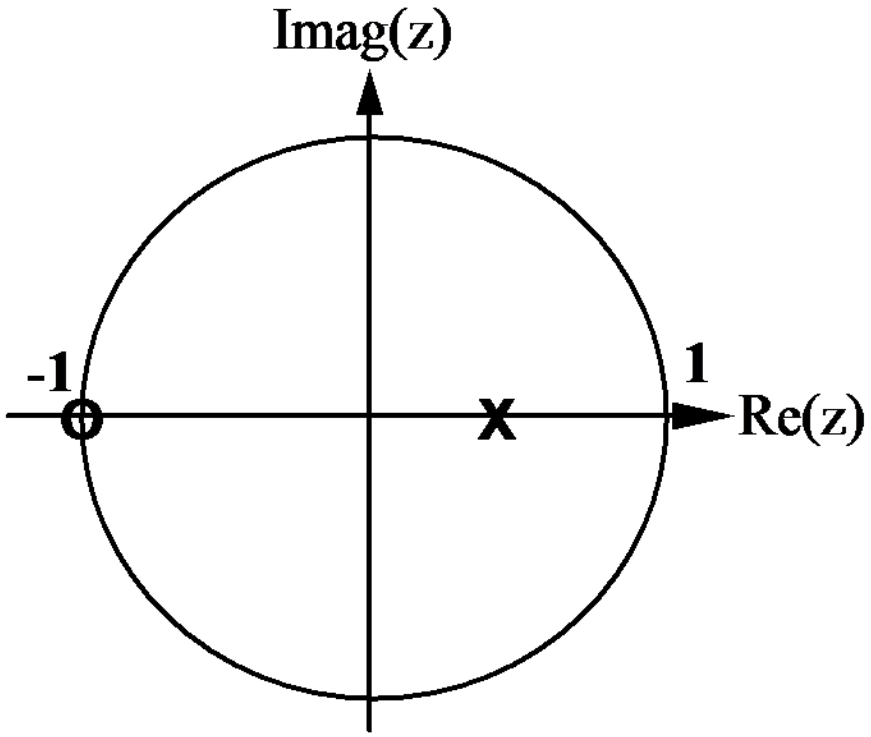
Note that the digital filter response at zero frequency equals 1, as for the analogue filter, and the digital filter response at $\Omega = \pi$ equals 0, as for the analogue filter at $\omega = \infty$. The –3dB frequency is $\Omega = \pi/4$, as intended.

Pole-zero diagram for digital design.

Note that:

a) The filter is stable, as expected

b) The design process has added an extra zero compared to the prototype
- this is typical of filters designed by the bilinear transform.



There is a Matlab routine `BILINEAR` which computes the bilinear transformation.

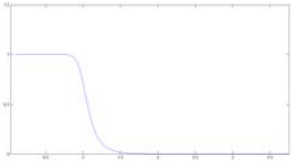
The example above could be computed, for example, by typing

```
[NUMd,DENd] = BILINEAR([0.4142],[1 0.4142],0.5)
```

which returns

```
NUMd =

   0.2929    0.2929

DENd =

   1.0000   -0.4142
```

Analogue filter prototypes

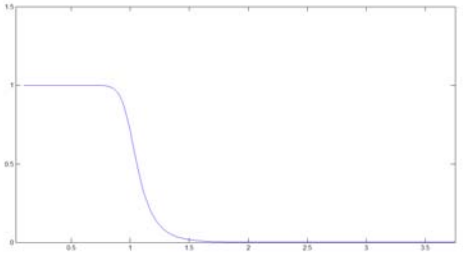


Analogue designs exist for all the standard filter types (lowpass, highpass, bandpass, bandstop). The common approach is to define a standard lowpass filter, and to use standard analogue-analogue transformations from lowpass to the other types, prior to performing the bilinear transform.

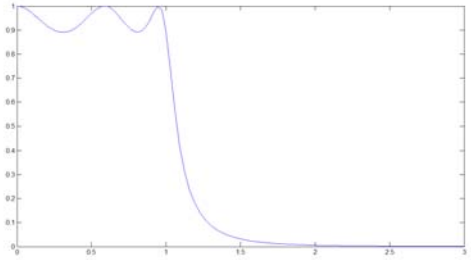
It is also possible to transform from lowpass to other filter types directly in the digital domain, but we do not study these transformations here.

Important families of analogue filter (lowpass) responses are described in this section, including:

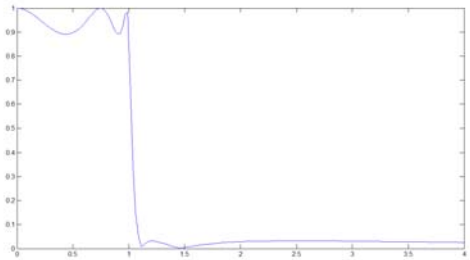
1. Butterworth – maximally flat frequency response near $\omega=0$



2. Chebyshev – equiripple response up to ω_c, monotonically decreasing > ω_c



3. Elliptic – equiripple in passband, equiripple in stopband.



Butterworth (maximally flat)

An Nth-order lowpass Butterworth filter has transfer function H(s) satisfying

$$H(s)H(-s) = \frac{1}{1 + (s/(j\omega_c))^{2N}}$$

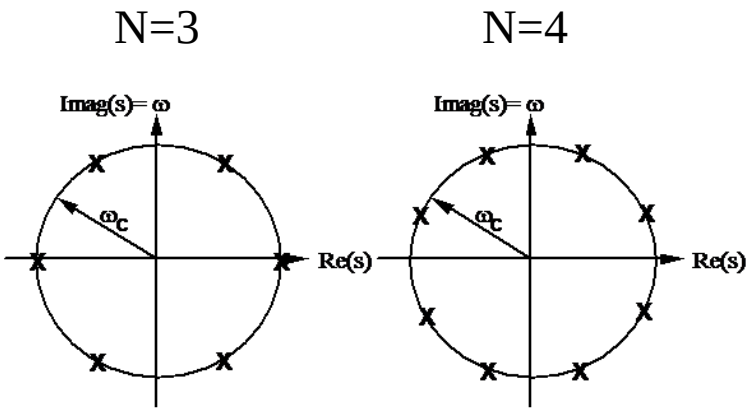
This has unit gain at zero frequency (s = j0), and a gain of -3dB (= √0.5) at s = jωc.

The poles of H(s)H(-s) are solutions of

$$\left(\frac{s}{j\omega_c}\right)^{2N} = -1$$

i.e. at

$$s = j\omega_c e^{j(2k+1)\pi/(2N)}$$



as illustrated on the right for N = 3 and N = 4:

Clearly, if λ_i is a root of H(s), then $-\lambda_i$ is a root of H(-s).

Thus we can immediately identify the poles of H(s) as those roots lying in the left half-plane, for a stable filter.

The frequency magnitude response is obtained as:

$$H(j\omega)H(-j\omega) = |H(j\omega)|^2 = \frac{1}{1+(\omega/\omega_C)^{2N}} \qquad (*)$$

Butterworth filters are known as "maximally flat" because the first 2N-1 derivatives of (*) w.r.t. ω are 0 at $\omega = 0$.

Matlab routine `BUTTER` designs digital Butterworth filters (using the bilinear transform):

`[B,A] = BUTTER(N,Wn)` designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors `B` and `A`. The cut-off frequency `Wn` must be $0.0 <$ `Wn` < 1.0, with 1.0 corresponding to half the sample rate.

Butterworth order estimation

Equation (*) can be used for estimating the order of Butterworth filter required to meet a given specification.

For example, assume that a digital filter is required with a -3dB point at $\Omega_c = \pi/4$, and it must provide at least 40dB of attenuation above $\Omega_s = \pi/2$.

Warping the critical frequencies gives $\omega_c = \tan(\pi/8) = 0.4142$ and $\omega_s = \tan(\pi/4) = 1$.

40dB corresponds to $|H(e^{j\Omega})|^2 = 10^{-4}$, so find N by solving

$$\frac{1}{1+\left(\omega_S/\omega_C\right)^{2N}} < 10^{-4} \qquad \Rightarrow \qquad 2N > 10.45$$

Hence, since **N must be integer**, choose N = 6.

Matlab provides a function buttord for calculation of the required Butterworth order

Other Types of Analogue Filter

There is a wide range of closed form analogue filters. Some are all-pole; others have zeros. Some have monotonic responses; some equiripple. Each involve different degrees of flexibility and trade-offs in specifying transition bandwidth, ripple amplitude in passband/stopband and phase linearity.

The meaning of "equiripple" is illustrated in Figure 10.2, which shows a type I Chebyshev response which is equiripple in the passband $0<\omega<\omega c=1$, and monotonic in the stopband.

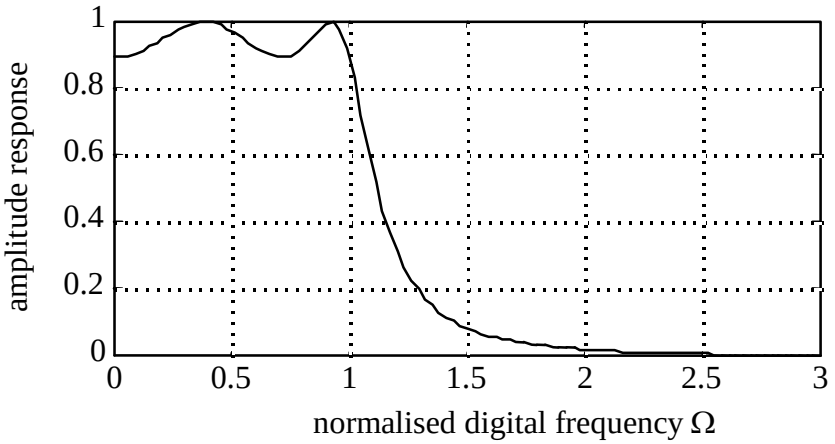
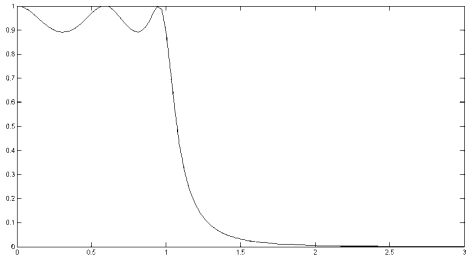


Figure - Type I fourth order Chebyshev LPF frequency response

For a given bandedge frequency, ripple specification, and filter order, narrower transition bandwidth can be traded off against worse phase linearity

Chebyshev filters are characterised by the frequency response:

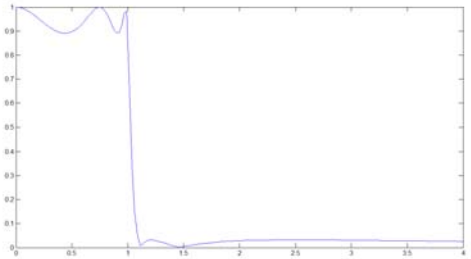
$$|H(j\omega)|^2 = \frac{1}{1 + \frac{\epsilon}{1-\epsilon} T_n(\omega)^2}$$



Where $T_n(\omega)$ are so-called Chebyshev polynomials.

Elliptic filters allow for equiripple in both pass and stopbands. They are governed by a similar form:

$$|H(j\omega)|^2 = \frac{1}{1 + \mu^2 E^2(\omega)}$$



Where $E(\omega)$ is a particular ratio of polynomials.

Other filter types include Bessel filters, which are almost linear phase.

Transformation between different filter types (lowpass to highpass, etc.)

Analogue prototypes are typically lowpass. In order to convert to other types of filter one can first convert the analogue prototype in the analogue domain, then use the bilinear transform to move to digital as before.

The following procedures may be used, assuming a lowpass prototype with cutoff frequency equal to 1:

1. Lowpass to Lowpass

   Set $s' = s/\omega_c$ to give lowpass with cutoff at ω_c

2. Lowpass to Highpass

   Set $s' = \omega_c/s$ to give highpass with cutoff at ω_c

3. Lowpass to Bandpass

   $s' = \frac{s^2+\omega_l\omega_u}{s(\omega_u-\omega_l)}$ to give bandpass with lower cutoff at ω_l, upper cutoff at w_u

4. Lowpass to Bandstop

   $s' = \frac{s(\omega_u-\omega_l)}{s^2+\omega_l\omega_u}$ to give bandstop with lower cutoff at ω_l, upper cutoff at w_u

Example: The transfer function of a second order Butterworth lowpass filter with cutoff frequency 1 is

$$\frac{1}{s^2+\sqrt{2}s+1}$$

From this, a second order highpass filter with cutoff frequency ω_c can be designed:

$$\frac{1}{(\omega_c/s)^2+\sqrt{2}(\omega_c/s)+1}=\frac{s^2}{\omega_c{}^2+\sqrt{2}s\omega_c+s^2}$$

From here, a digital highpass filter can be designed, using the bilinear transform and setting

$$\omega_c=\tan(\Omega_c/2)$$

Comparison of IIR and FIR filters

If the desired filter is **highly selective** (that is, its frequency response has small transition bandwidths or "steep sides"), then the **impulse response will be long** in the time domain. Examples include narrowband filters and lowpass /highpass /bandpass filters with steep cutoffs.

For an FIR filter, a long impulse response means the filter is long (high order), so it requires many multiplications, additions and delays per sample.

An IIR filter has active *poles* as well as *zeros*. Poles, acting as high-Q resonators, can provide highly selective frequency responses (hence long impulse responses) using much lower filter order than the equivalent FIR filter, hence much lower computational cost.

Although it is still true that a **more selective response** requires a **higher order filter**.

On the other hand, the closer to linear the phase is required to be, the higher the order of IIR filter that is needed. Also the internal wordlengths in IIR filters need generally to be higher than those in FIR filters; this may increase the implementation cost (e.g in VLSI).

An FIR filter is inherently stable, unlike an IIR filter. Hence an FIR implementation involving inaccurate (finite precision, or 'quantised') coefficients will be stable, whereas an IIR one might not. (However, it is desirable in either case to compute the **actual** frequency response of the filter, using the actual quantised values of the coefficients, to check the design.)

Implementation of digital filters

So far we have designed a digital filter to meet prescribed specifications, with the result expressed as a rational transfer function H(z). We now consider implementation. Typical options are:

Implementation type	**Multiplication speed / cost**
1. Pre-1980 high speed hardware implementation	Fixed-point. Dedicated multiplier ICs. Power-hungry, expensive.
2. Pre-1980 microprocessor	Fixed-point. Microcoded. Slow.
3. Fixed-point DSP IC (cheaper; goes faster)	Take same time as additions
4. Custom VLSI - fixed-point arithmetic IC (faster or less area than floating point)	Either take same time, but more IC area, or same area but more time
5. Floating-point μprocessor	May take more time than additions
6. Floating-point DSP IC	Take same time as additions
7. Custom VLSI - floating-point arithmetic	Probably take same time as additions

If **speed** is the main concern, then if multiplications take longer than additions, we aim to reduce the **number of multiplications**; otherwise to reduce the **total operation count**.

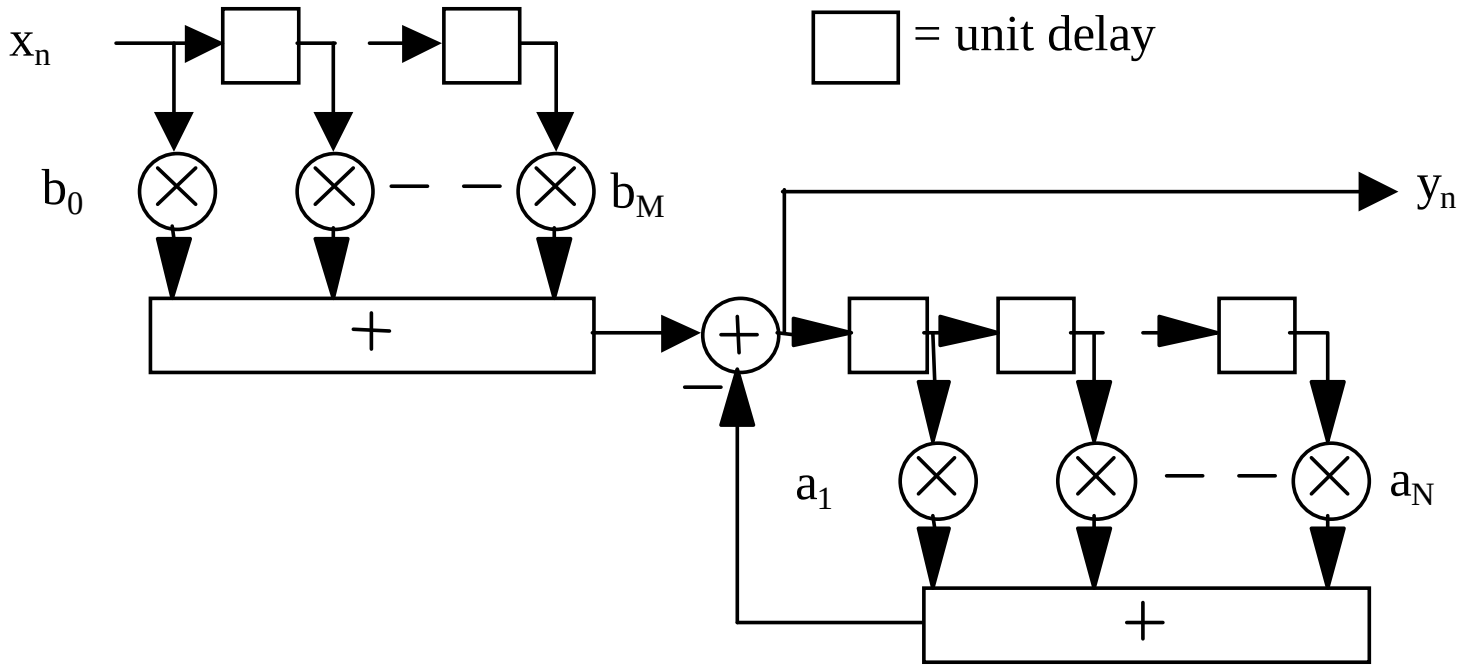
The use of **fixed-point** arithmetic takes much less area than **floating-point** (so is cheaper) or can be made to go faster. The **area** of a fixed-point parallel multiplier is proportional to the product of the coefficient and data wordlengths, making **wordlength reduction** advantageous.

Hence much work has gone into structures which allow reductions in

- the number of multipliers; or
- the total operation count (multipliers, adders and perhaps delays); or
- data or coefficient wordlengths

If **power consumption** is the concern, then **reducing** total **operation count** and **wordlength** are desirable. Also fixed point is much better than floating point. Since general multiplication takes much more power than addition, we try to **reduce the number of multiplications**, or to replace general multiplications by, for example, binary shifts.

Recall the Direct Form I implementation considered so far:



Structures for IIR filters - Cascade and Parallel

Implementing a digital filter in direct form is satisfactory in (for example) Matlab's `filter` routine, where double precision floating-point is used.

However in fixed point or VLSI implementations direct form is not usually a good idea.

1. alternative structures may decrease multiplications or overall computation load;

2. when fixed-point coefficients are used, the response of alternative structures is much less sensitive to coefficient imprecision (coefficient quantisation); and

3. when fixed-point data are used, alternative structures may add less quantisation noise into the output signal.

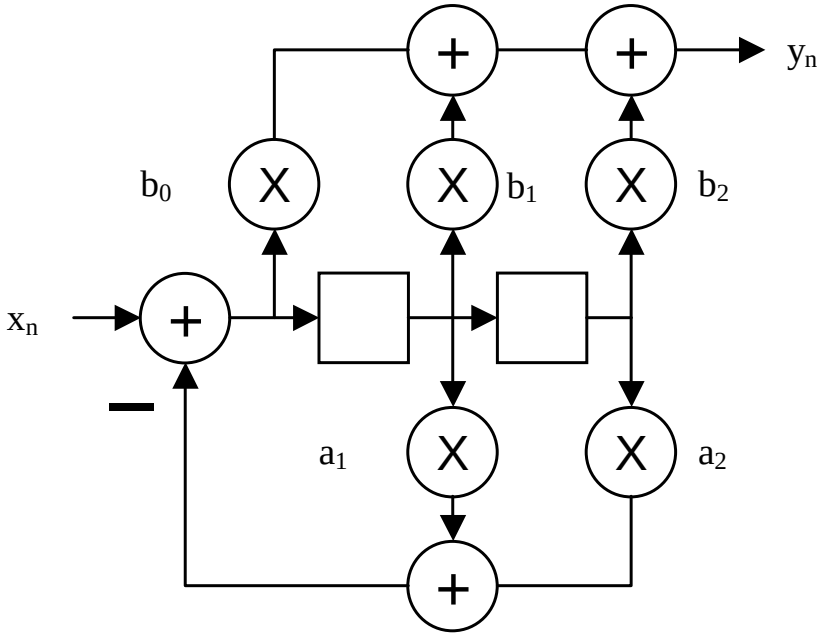
We therefore consider alternative forms of IIR filter, their operation count and sensitivity to finite precision effects.

Canonic form IIR sections

The earlier Figure showed an implementation with separate FIR and IIR stages, called Direct Form I.

We can minimise the number of delay stores by putting the feedback stage first and then using the same delay stores for both parts. This is called the *canonic form* ('canonic' means minimum), or Direct Form II.

A canonic form filter can be of arbitrary order, but the following example has 2 poles and 2 zeros; this is called a *biquadratic* section:



[Check for yourself that this gives the same output as the Direct Form I structure]

Sensitivity to coefficient quantisation

If the filter coefficients are quantised, the resulting **errors in coefficient value** cause errors in pole and zero positions, and hence filter response. Consider a filter with four poles at z = -0.9. If implemented as a direct form filter it would have the following denominator polynomial in its transfer function:

$$(1 + 0.9z^{-1})^4 = 1 + 3.6z^{-1} + 4.86z^{-2} + 2.916z^{-3} + 0.6561z^{-4}$$

Now let us add an "error" of -0.06 to the third coefficient, changing it from 4.86 to 4.8. The roots of the resulting polynomial are

-1.5077, -0.7775+0.4533i, -0.7775-0.4533i, and -0.5372

They have been hugely modified. The filter is unstable (first pole radius > 1).

If, by contrast, the filter were implemented as a cascade of 4 first-order sections, each implementing a denominator term $(1 + 0.9z^{-1})$, an error of the same size would have much less effect. For example a change of one coefficient from 0.9 to 0.84 clearly just moves one root from 0.9 to 0.84: (a) a smaller change, and (b) affecting only one root.

This illustrates the fact that a **cascade realisation displays much lower sensitivity to coefficient quantisation** than a direct realisation.

Cascades typically use first and second order sections

To obtain complex (resonant) roots with real filter coefficients requires at least a second-order section. Each complex root, with its inevitable conjugate, can be implemented by a single second-order section. For example, a root at $r\exp(j\Omega)$ and its conjugate $r\exp(-j\Omega)$ generate the real-coefficient second-order polynomial

$$(1 - r\exp(j\Omega)z^{-1})(1 - r\exp(-j\Omega)z^{-1}) = 1 - 2r\cos(\Omega)z^{-1} + r^2z^{-2}$$

so, to place zeros at $r\exp(\pm j\Omega)$, set $\mathbf{b_0 = 1,\ b_1 = -2r\cos(\Omega),\ b_2 = r^2}$.

(In principle, b_0, b_1 & b_2 could all be multiplied by a common scale factor, but it is usually advantageous to set $b_0 = 1$ throughout, to avoid unnecessary multiplications, and use a single overall gain factor.)

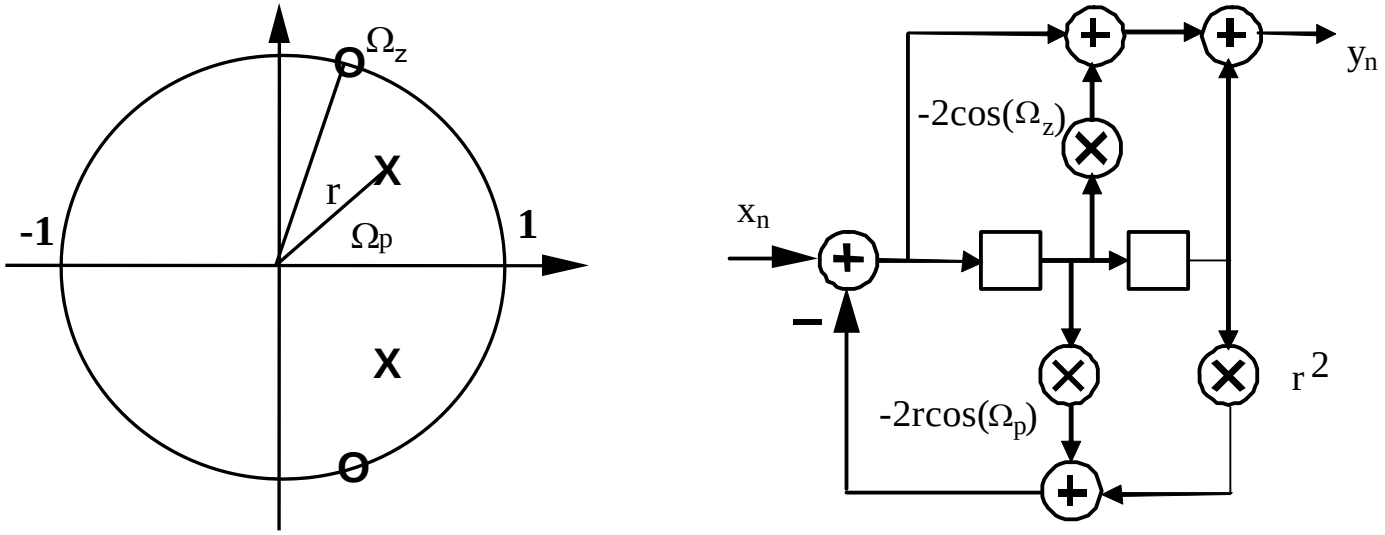
Or to place poles at $r\exp(\pm j\Omega)$, set $\mathbf{a_1 = -2r\cos(\Omega),\ a_2 = r^2}$.

Real poles may be implemented by first **or** second order sections.

Zeros on the unit circle

Many filters (IIR and FIR) have zeros on the unit circle. Hence $r = 1$ above, so that $b_2 = 1$. This does not require a multiplier.

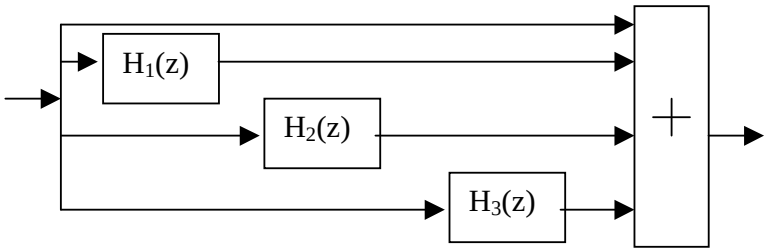
A biquadratic section with two resonant poles at radius r, frequency Ω_p, and two zeros on the unit circle at frequency Ω_z, is illustrated below.



Implementing a high-order filter with many zeros on the unit circle as a cascade of biquadratic sections requires fewer total multiplications than a direct form implemention.

Parallel form IIR filters

An IIR filter can be implemented as a *parallel* summation of low order sections:



Partial Fraction Expansion is used to compute the *numerator* coefficients of the parallel form.

$$\frac{b_0 + b_1 z^{-1} + b_2 z^{-2} \ldots}{\left(1 + a_1 z^{-1} + a_2 z^{-2}\right)\left(1 + c_1 z^{-1} + c_2 z^{-2}\right)\ldots} = B + \frac{A_1 + A_2 z^{-1}}{\left(1 + a_1 z^{-1} + a_2 z^{-2}\right)} + \frac{C_1 + C_2 z^{-1}}{\left(1 + c_1 z^{-1} + c_2 z^{-2}\right)} \ldots$$

The parallel form is little used, because:

- It sometimes has an advantage over the cascade realisation in terms of internally generated quantisation noise (see later), but not much.
- Longer coefficient wordlengths are usually required.
- Zeros on the unit circle in the overall transfer function are not preserved, therefore no saving of multipliers can be obtained for filters having such zeros.

Finite wordlength effects in digital filters

Many digital filters are implemented using fixed point binary 2's-complement arithmetic. For a B bit representation, with A bits before the binary point and B-A bits after it, all values in the filter are quantised to integer multiples of the LSB $q \equiv 2^{-(B-A)}$ and the number range is

$$-2^{(A-1)} \le \left(x \equiv kq \equiv k2^{-(B-A)} \right) < 2^{(A-1)}$$

for example, a B=12 bit number with A=2 bits before the binary point is in the range -2048/1024 to +2047/1024 inclusive. We will represent such values as (B,A).

Overflow, saturation arithmetic, and scaling

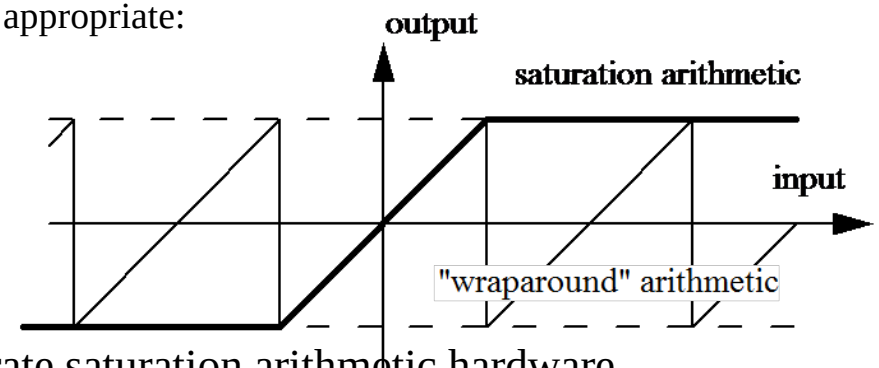
If the result of any calculation in the filter exceeds its number range, then **overflow** occurs. By default, a value slightly greater than the maximum representable positive number becomes a large negative number, and vice versa. This is called **wraparound**; the **resulting error is huge**. In IIR filters it can result in very large amplitude "overflow oscillations".

There are two strategies which can be used to avoid problems of overflow.

Scaling can be used to ensure that values can never (or hardly ever) overflow, and/or **saturation arithmetic** can be used to ensure that if overflow occurs its effects are greatly reduced.

In saturation arithmetic, the results of all calculations are first computed to full precision. For example, the addition of 2 (B,A) values results in a (B+1,A+1) value; the multiplication of a
(B,A) value by a (C,D) value results in a (B+C-1, A+D-1) value.

Then instead of merely *masking* the true result to a (B,A) field, which causes overflow, the higher order bits of the true result are processed to detect overflow. If overflow occurs, the maximum possible positive value or minimum possible negative value is returned as appropriate:



Some DSP ICs incorporate saturation arithmetic hardware.

Scaling

l_1 scaling

Assume that the input to a filter (or section of a filter) is bounded by $|x(n)| < C$, and that its impulse response is h(k), k=0, 1, ... Then its output is bounded by $C \Sigma | h(k) |$.

$\Sigma | h(k) |$ is known as the *l_1 norm* of the filter impulse response. It often does not have a convenient analytical form, but computing it numerically is easy.

Thus if the maximum permissible output magnitude is D, overflow cannot occur provided we scale the output by the factor

$$D/(C \Sigma | h(k) |);$$

this is known as **l_1 scaling**.

However if we reduce the magnitude of signals, the ratio of signal power to quantisation noise power becomes smaller, so **scaling worsens the noise performance of the filter**.

frequency-response scaling

The input signal which gives rise to the largest possible output is unlikely to occur in practice, so a less conservative scaling approach is often used.

If the frequency response of the filter is G(exp(jΩ)), then a sinewave of frequency Ω and peak amplitude C at the input will give a sinewave of peak amplitude C | G(exp(jΩ)) | at the output. To scale so that a single sinewave cannot overflow, use scale factor

$$\frac{D}{(C \max\{ | G(\exp(j\Omega)) | \})}$$

this is known as **frequency-response scaling**.

l_2 scaling

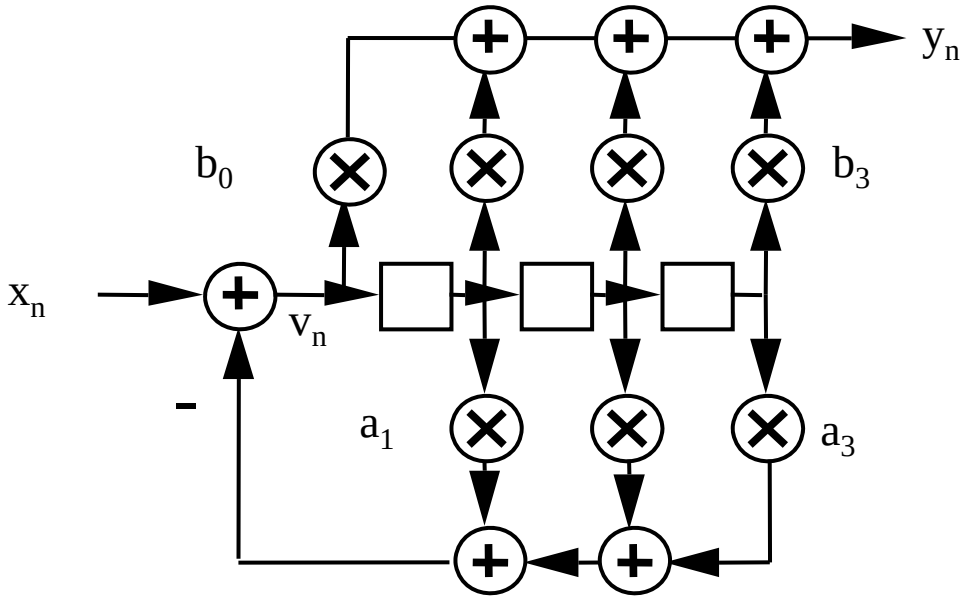
Less conservative scaling still is obtained using the scale factor $\frac{D}{\left(C\sqrt{\sum h(k)^2}\right)}$, which is based on the rms impulse response; this is known as **l_2 scaling**.

saturation arithmetic is still needed if frequency-response or l_2 scaling is used

If frequency response scaling or l_2 scaling is used, overflow is still possible, so in IIR filters saturation arithmetic must then be used as well.

Application of scaling to a single section

Consider the direct form II filter illustrated below:



First, the internal signal v_n must be scaled so that it does not overflow the number range.

This is achieved by computing the impulse response **from x_n to v_n** (for l_1 or l_2 scaling) or the frequency response from X to V (for frequency response scaling).

To prevent overflow of v_n, the signal into the filter must be scaled *before the input*. This may be implemented as a simple **binary shift,** by using the next smaller power of 2.

Overflow of the filter *output* is then prevented by computing the impulse or frequency response **from the input to y(n)** (taking into account any scaling just introduced between the input and x_n). Any further scaling required is implemented by scaling all the *coefficients* of the FIR part (b_0 ... b_M) by the necessary scale factor.

Application of scaling to cascade and parallel realisations

The application of scaling to a cascade realisation is based on the process described above; however, at each step, you must compute the impulse response or frequency response **from the input of the overall filter** to the point of interest, taking account of all scaling already included up to that point.

Again, the scaling at section inputs may be implemented using simple binary shifts, or by incorporating it into the FIR coefficient scaling of the preceding section.

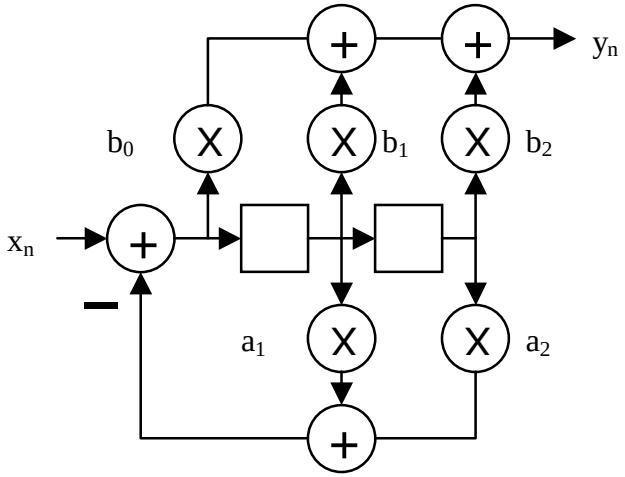
For a parallel realisation, scaling is *computed* independently for each section, but all section outputs must be scaled by the same amount, so the overall scaling of each section must be made the same. Finally scaling is applied to the final adder(s) which add together the outputs of the parallel sections.

Roundoff (quantisation) noise generation

The output of a multiplier has more bits than its inputs (for example, a 16 by 16 two's complement multiplier outputs a 31-bit two's complement value). Therefore to store the output it **has to be (re)quantised** (that is, low order bits have to be thrown away). Hence an error called quantisation noise or roundoff noise is added at that point.

The **noise variance** at the multiplier output, **assuming rounding** is used, is $q^2 / 12$, where q is the LSB size after quantisation. (The same as for quantisation of analogue signals.)

Consider the Direct Form II filter below, and assume that the output of each multiplier is individually quantised.



It is often assumed that the quantisation noise at each multiplier output is white (independent from sample to sample). And also that it is independent between multipliers, so that the noise variances ("powers") add.

(The assumption of whiteness is actually a *very poor* model if the signal is narrowband, but it is reasonable for large amplitude wideband signals. The assumption of independence can also be poor.)

The quantisation noise from the multipliers of an FIR filter (b_0 ... b_3 in our example) therefore adds white noise directly to the output signal.

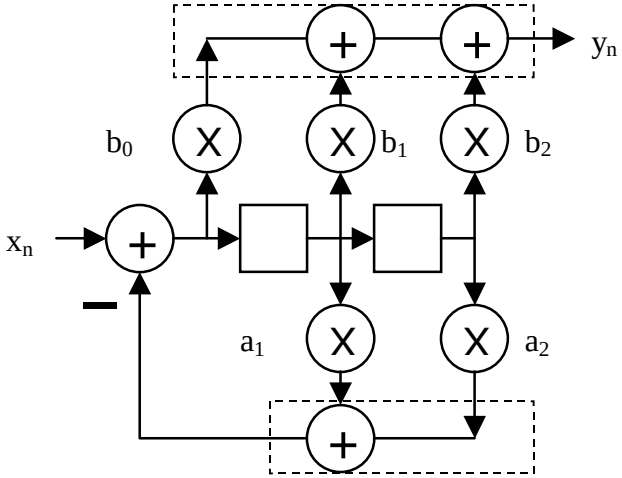
In IIR filters, the white quantisation noise from the feedback multipliers filter (a_1, a_2) is fed to the **input** of the filter, so the resulting noise spectrum at the filter output is **coloured**; its spectrum is proportional to the square of the filter's frequency response magnitude.

Hence roundoff noise level is affected by **data** wordlengths, **filter response**, **filter structure** and (to a certain extent) by **section ordering** in cascade structures. Further details are in specialist texts.

Hardware support for reducing quantisation noise

DSP ICs, and some VLSI filters, provide an **accumulator** store of **longer wordlength** than the data wordlength (e.g. a 32-bit accumulator for a 16-bit DSP). The multiplier outputs are accumulated at the longer wordlength, and then the accumulator output is **only quantised once**.

For example, in the following figure, the three FIR additions would be into the accumulator, which then would be quantised to generate y_n. Similarly the two feedback (IIR) additions, and possibly the addition of input x_n, would be into the accumulator, which would be quantised to generate v_n. This approach significantly reduces roundoff noise.



Limit cycles

Zero-input limit cycles are self-sustaining oscillations, caused by the **rounding of the results** of computations.

For example, consider the second-order filter $y_n = x_n - 0.9\, y_{n-2}$

This is a stable second order IIR filter with complex poles at $\pm j\sqrt{0.9}$. If rounding to the nearest LSB is used at the output of the multiplier, then when $y_{n-2} = \pm1,\pm2,\pm3$, or ±4 LSB, the computation $0.9y_{n-2}$ will give the result $\pm1,\pm2,\pm3$, or ±4 LSB respectively.

Hence a limit cycle of the form

y(n) = 4,4,0,0,4,4,0,0,... (or the same pattern with 3,2, or 1) may occur.

Effectively, the reason is that the rounding non-linearity has increased the feedback gain to 1, turning the system into an oscillator. Limit cycles may occur for real or complex poles.

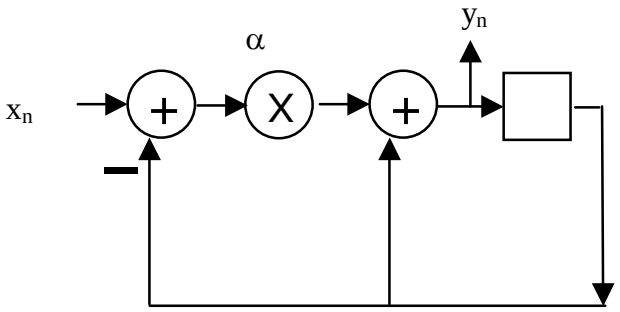
Limit cycles are troublesome in some applications, especially with short data wordlengths, where the limit cycle may be relatively large. With the longer wordlengths of DSP ICs, it is often possible to ignore limit cycles.

One solution is to quantise toward 0 (**truncation**) instead of rounding the multiplier output. But the extra roundoff noise due to truncation may require the data wordlength to be increased by 1 or 2 bits. Another solution is to use certain forms of digital filter (such as Wave filters) which do not support limit cycles. However these are computationally more expensive.

Deadbands

Consider a simple digital lowpass filter such as is commonly used for smoothing:

$y_n = y_{n-1} + \alpha(x_n - y_{n-1})$ (equivalent to $y_n = (1 - \alpha)y_{n-1} + \alpha x_n$)



The transfer function is $H(z) = \alpha / (1 - (1 - \alpha)z^{-1})$. This has unit gain at zero frequency (z=1), and a pole at $(1 - \alpha)$. The time constant is approximately $(1/\alpha)$ samples, for $\alpha << 1$.

If $| \alpha(x_n - y_{n-1}) | < 0.5$ LSB, (which implies $| (x_n - y_{n-1}) | < (0.5/\alpha)$), then the multiplier output will round to zero, and the filter output will therefore remain constant. Hence a constant output error, $x_n - y_{n-1}$, known as the *deadband*, arises. It can be up to $(0.5/\alpha)$ LSB.

If, for example, $(1/\alpha)=10000$ to give a time constant of 10000 samples, then the size of the LSB of the internal arithmetic must be 5000 times smaller than the permissible size of the deadband. This implies 13 extra bits (since $2^{12} = 4096$).

Coefficient quantisation

We showed earlier that the cascade form is much less sensitive to coefficient quantisation than a high order direct form filter. (This is also true of the parallel form.)

If the filter has zeros on the unit circle, as in this case, the cascade realisation has the advantage that these zeros stay on the unit circle (because a coefficient $b_2 = 1.0$ is unaffected by quantisation), although their **frequencies** may be altered.

A traditional way to study the relative merits of different filter *structures* was to **analyse the sensitivity** of the frequency response magnitude to random (often Gaussian) perturbations of the coefficients, and to use this as a measure of the likely sensitivity of a given structure to coefficient quantisation. Various **structures**, including Lattice and Wave filters, give even **lower sensitivity** to coefficient quantisation than the cascade realisation. However, they generally require a substantially increased number of multipliers.

For a *specific filter design*, you should compute the **actual** filter response with quantised coefficients, and then modify it if necessary.

In dedicated hardware, such as custom ICs, where there are significant benefits from reducing coefficient wordlengths, **discrete optimisation** can be used to search for the finite-wordlength filter with the closest response to a given specification. Some discrete optimisation algorithms, including Genetic Algorithms, are available in software libraries.

FIR filter implementation by fast convolution

A length-N FIR filter requires in general N multiplications and N-1 additions per output sample. If the filter is symmetric, the number of multiplications may be halved, as explained before. But for a highly selective response (narrow transition band) the filter order may be high. An alternative method of FIR filtering, called *fast convolution*, uses the FFT to reduce the computation load.

The key result is that if

signal vector $\mathbf{x} = [x_0\ x_1\ ...\ x_{N-1}]$ has DFT $\mathbf{X} = [X_0\ X_1\ ...\ X_{N-1}]$,

and vector $\mathbf{h} = [h_0\ h_1\ ...\ h_{N-1}]$ has DFT $\mathbf{H} = [H_0\ H_1\ ...\ H_{N-1}]$,

then the *inverse DFT* **y** of $\mathbf{H} \bullet \mathbf{X} = [X_0H_0,\ X_1H_1,\ ...\ X_{N-1}H_{N-1}]$ is the

circular convolution of **x** and **h**, defined as:

$$y_m = \sum_{n=0}^{N-1} h_n x_{\mathrm{mod}(m-n,N)}$$

where '$\mathrm{mod}(P, N)$' denotes P represented in modulo N arithmetic.

To see why this is so, take

$$Y_m = H_m X_m, \quad \text{where} \quad H_m = \sum_{n=0}^{N-1} h_n e^{\frac{-j2\pi nm}{N}}, \quad X_m = \sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi nm}{N}}$$

Thus,

$$Y_m = \sum_{n_1=0}^{N-1} h_{n_1} e^{\frac{-j2\pi n_1 m}{N}} \sum_{n_2=0}^{N-1} x_{n_2} e^{\frac{-j2\pi n_2 m}{N}}$$

[Use separate variables n_1 and n_2 to distinguish terms in the two summations]

Taking inverse DFTs:

$$y_p = \frac{1}{N} \sum_{m=0}^{N-1} \left\{ \sum_{n_1=0}^{N-1} h_{n_1} e^{\frac{-j2\pi n_1 m}{N}} \sum_{n_2=0}^{N-1} x_{n_2} e^{\frac{-j2\pi n_2 m}{N}} \right\} e^{+\frac{jmp2\pi}{N}}$$

[Reorder summations]

$$= \frac{1}{N} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} h_{n_1} x_{n_2} \sum_{m=0}^{N-1} e^{\frac{-j2\pi(n_1+n_2-p)m}{N}}$$

[Summation is a Geometric Progression – check you can get this result yourself]

$$= \frac{1}{N} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} h_{n_1} x_{n_2} \times \begin{cases} N, & \text{mod}(n_1 + n_2 - p, N) = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$= \frac{N}{N} \sum_{n_1=0}^{N-1} h_{n_1} x_{\text{mod}(p-n_1,N)} = \sum_{n=0}^{N-1} h_n x_{\text{mod}(p-n,N)}$$

[Required result]

Now, we show how to use this result to give a fast FIR filtering method. Consider filtering a sequence x with a filter h having order M. The required convolution is:

$$y_n = \sum_{m=0}^{M} h_m \, x_{n-m}$$

Now, choose a frame length N>>M. We notice that for M-1<n<N,

mod(n-m,N)=n-m

In other words, the result of cyclic convolution is the same as that of standard convolution:

$$y_n = \sum_{m=0}^{M} h_m \, x_{n-m} = \sum_{m=0}^{M} h_m \, x_{\mathrm{mod}(n-m,N)}, \quad M-1 < n < N$$

Standard convolution (`filtering')

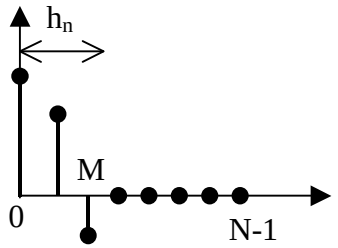
Cyclic convolution

This means that we can use fast cyclic convolution methods (based on DFT/FFT) to calculate the filtered output for n=M,…,N-1

The **overlap-save** method filters a long sequence of data x in chunks of length N-M, as follows:

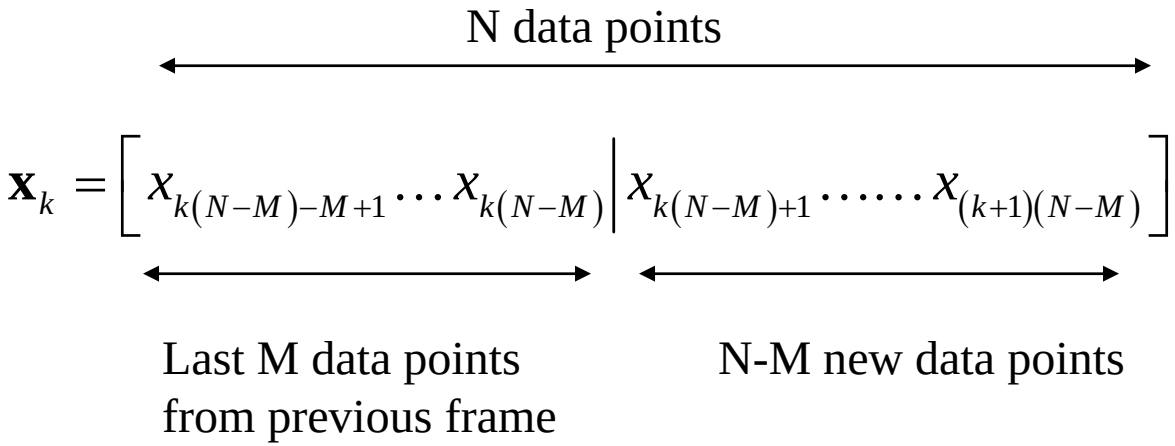
STEP 1

h_n is the impulse response of the FIR filter, and is of length M+1. Choose a much longer blocklength N, append N-(M+1) zeros to make the vector **h** and compute its DFT **H** via the FFT. Note that **H** only needs to be calculated once.



STEP 2 – For k=0, 1, 2, …

Construct the k^{th} `frame' of data $\mathbf{x}_k$ as follows:



[When k=0, set previous frame values to zero]

Then compute the DFT $\mathbf{X}_k$ of the vector $\mathbf{x}_k$, multiply $\mathbf{X}_k$ by $\mathbf{H}$ sample-by-sample, and IDFT the result to give $\mathbf{y}_k$. The **last** N-M samples of $\mathbf{y}_k$ are the **next** N-M filter outputs:

```
for k=1: ...  % [Note Matlab convention to start at k=1]
   if k==1
      X=fft([ zeros(1,M) x((1:N-M))]);
   else
      X=fft([ x((k-1)*(N-M)+(1-M:N-M))]);
   end
   y=real(ifft(H.*X));
   output((k-1)*(N-M)+(1:N-M)) = y(M+1:N); %last N-M samples of y
end
```

FIR filter implementation by fast convolution is an example of a **block based** signal processing method.

The saving can be significant - for example if M=100 and N=1024, the FFT-based method (assuming efficient FFTs are used for real data) requires about 33% the number of operations of the direct method.