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 x_n and y_n we have,

- Input x_n gives output u_n
- Input y_n gives output v_n
- Then, input $a\,x_n + b\,y_n$ gives output $a\,u_n + b\,v_n$

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

[Some of the the material in this section is adapted from notes by Dr Punskaya, Dr Doucet and Dr Macleod]

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:

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

X

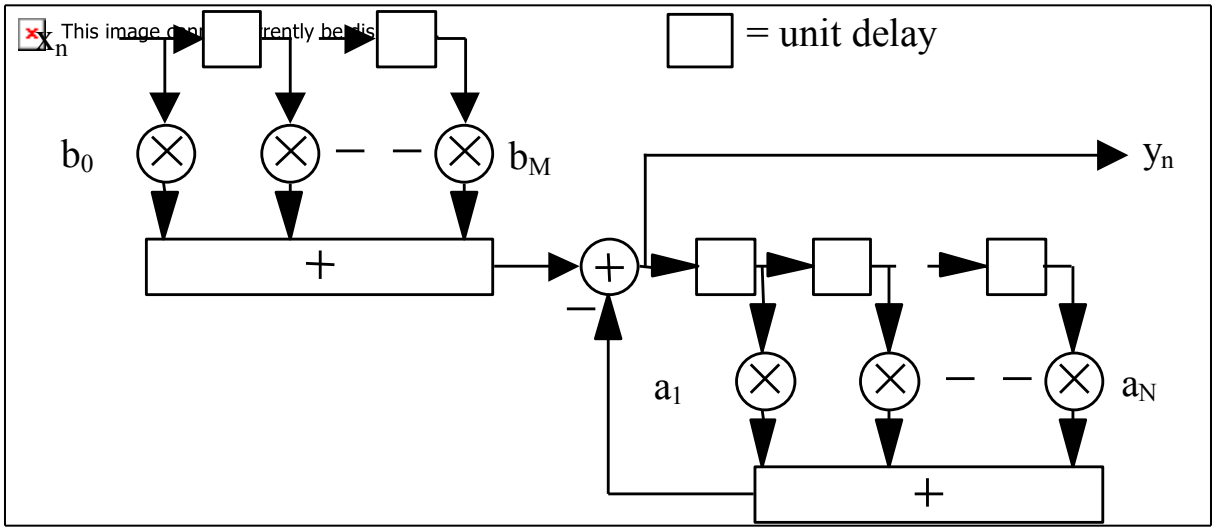
The linear time-invariant digital filter can then be described by the difference equation:

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

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:

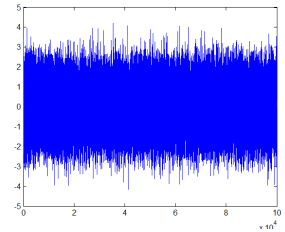
```
freqz(b,a,n);
```

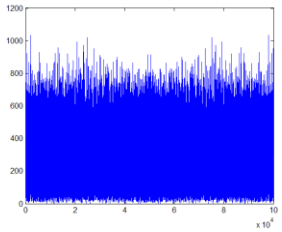
Filtering example:

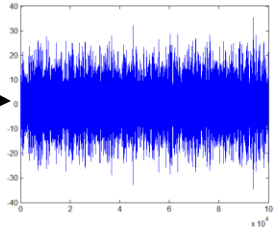
Generate a Gaussian random noise sequence:

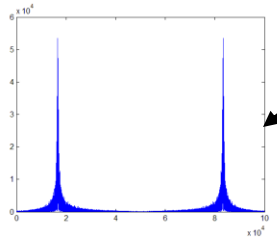
Matlab code:

```
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:

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

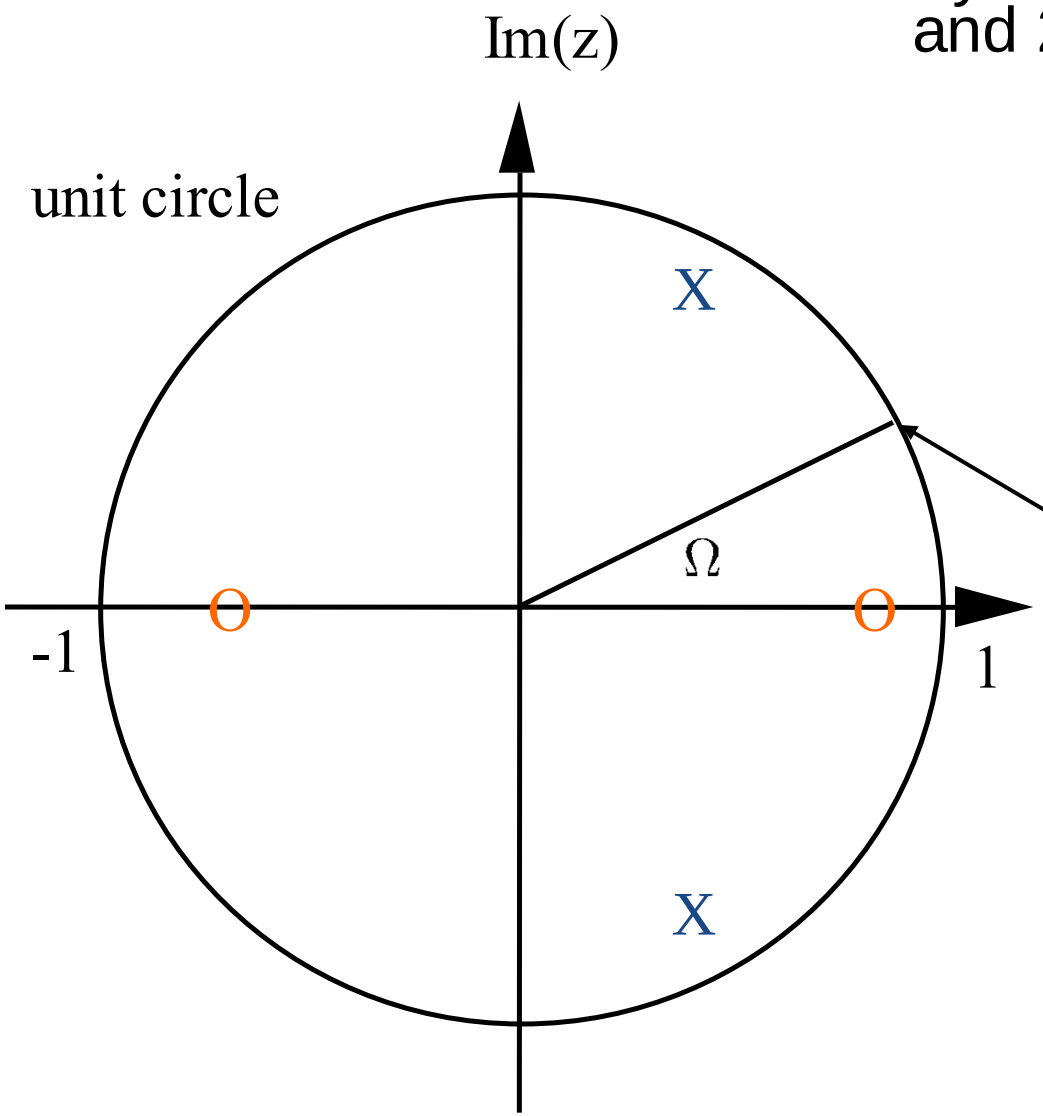
- $H(e^{j\Omega})$ is periodic with period 2π, since $\exp(j(\Omega + 2\pi)) = \exp(j\Omega)$.
- If a_p and b_p in the filter are real coefficients then

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

  (conjugate symmetry).

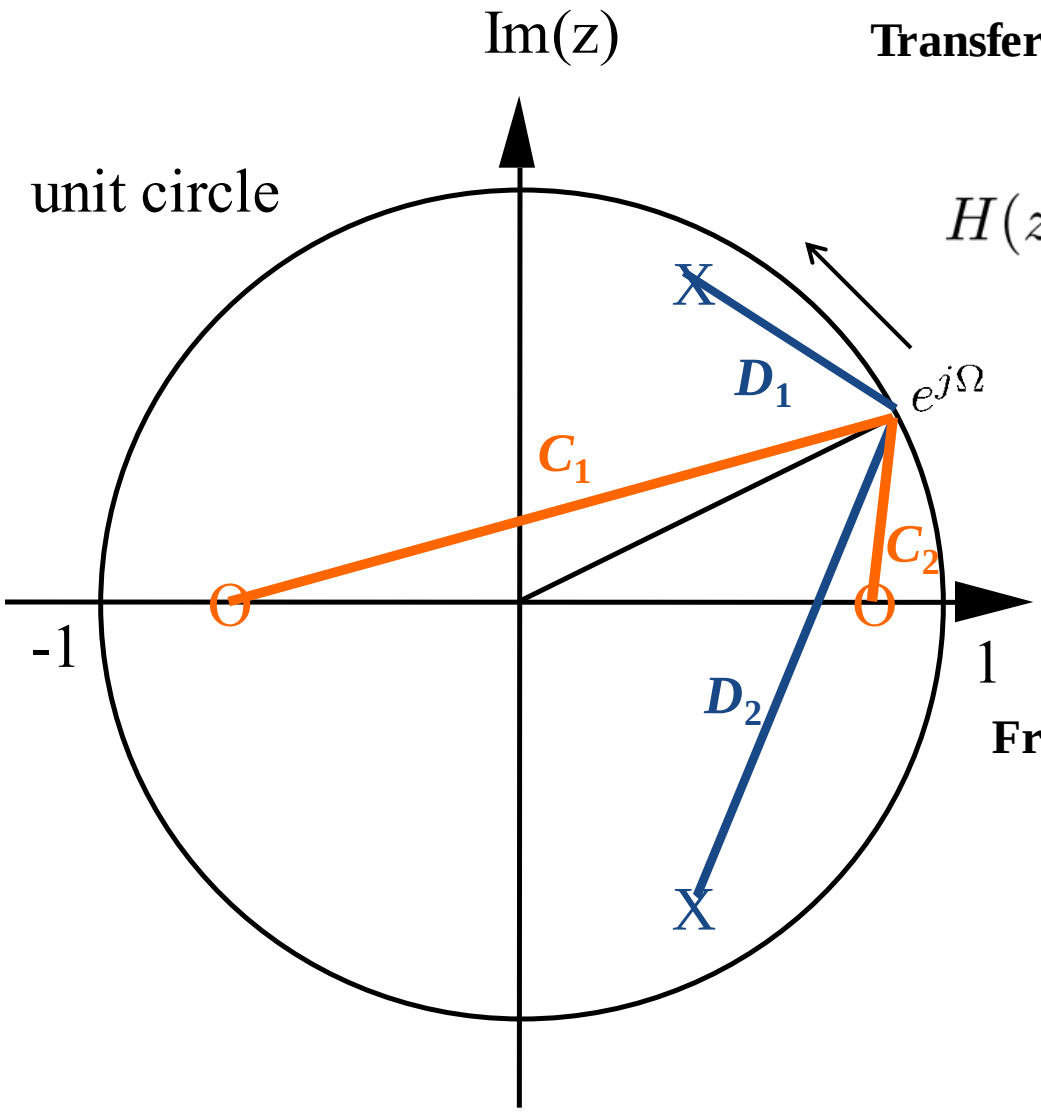
- We will make use of the graphical interpretation of frequency response taught in other courses (see also 3F3 practical). For example, the following illustrates a system with 2 poles and 2 zeros:

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



Proceed around the unit circle with

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

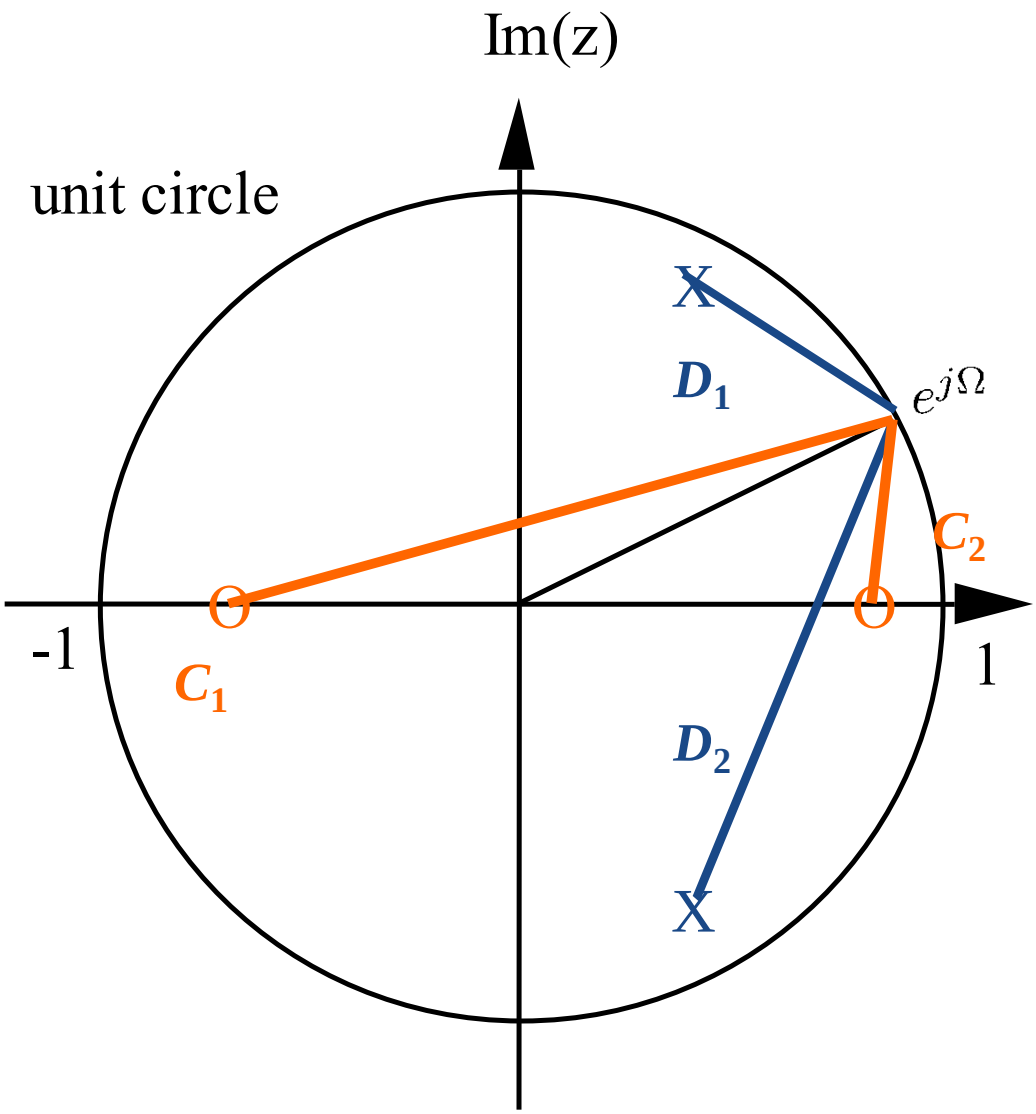


Transfer function:

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

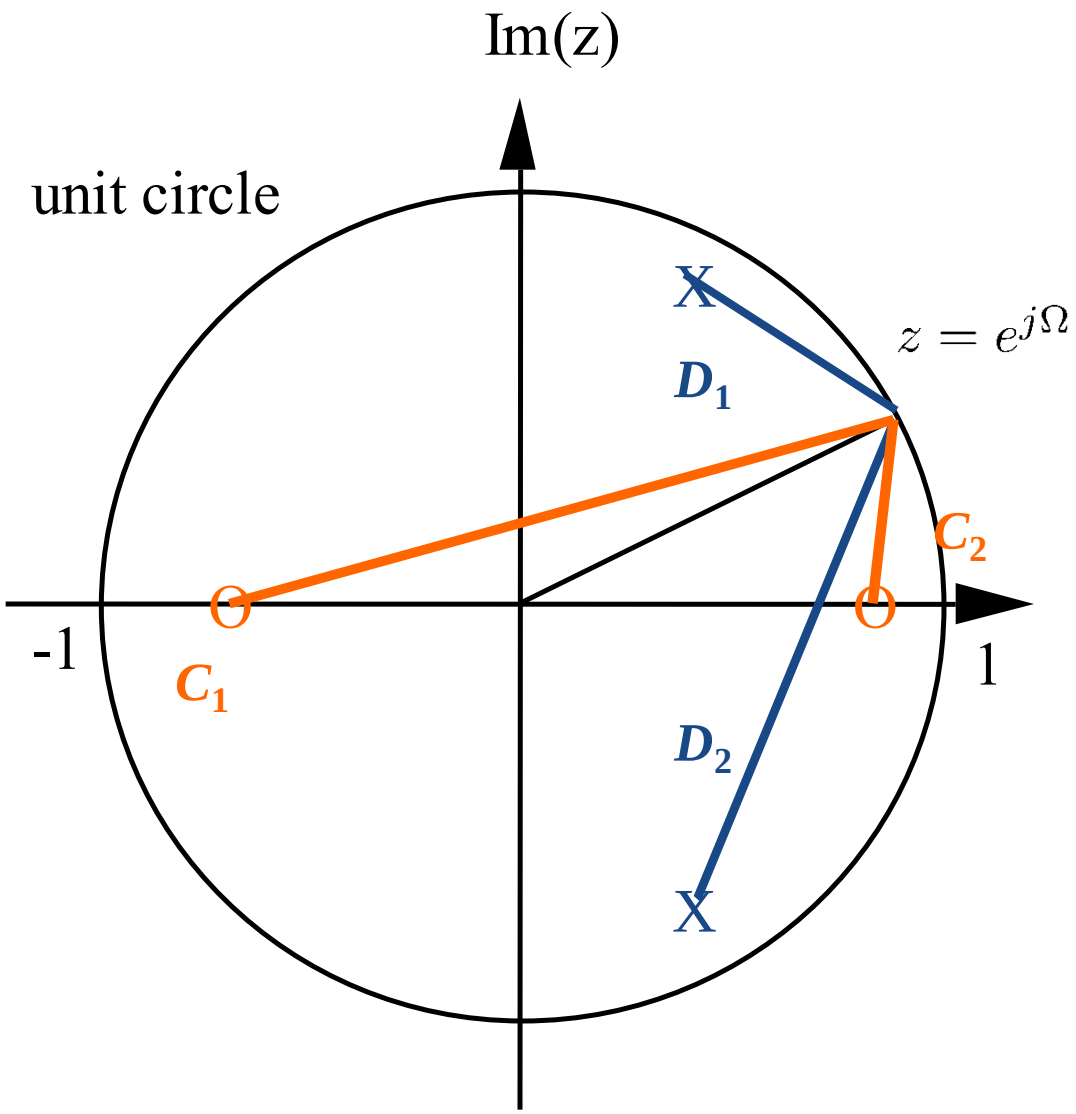
Frequency response:

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



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



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).

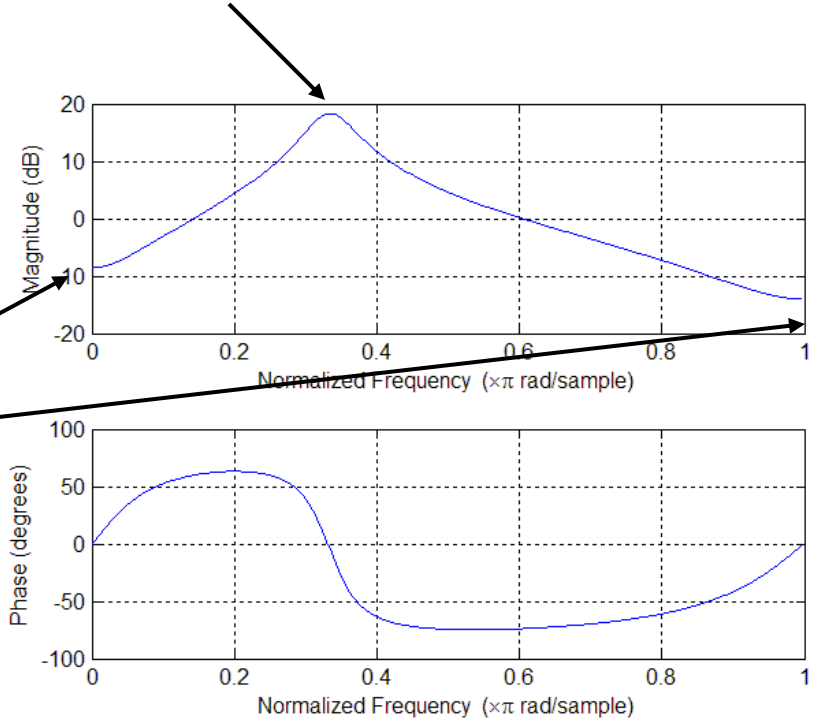
The phase response – more difficult to get "intuition", but similar principle applies

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,

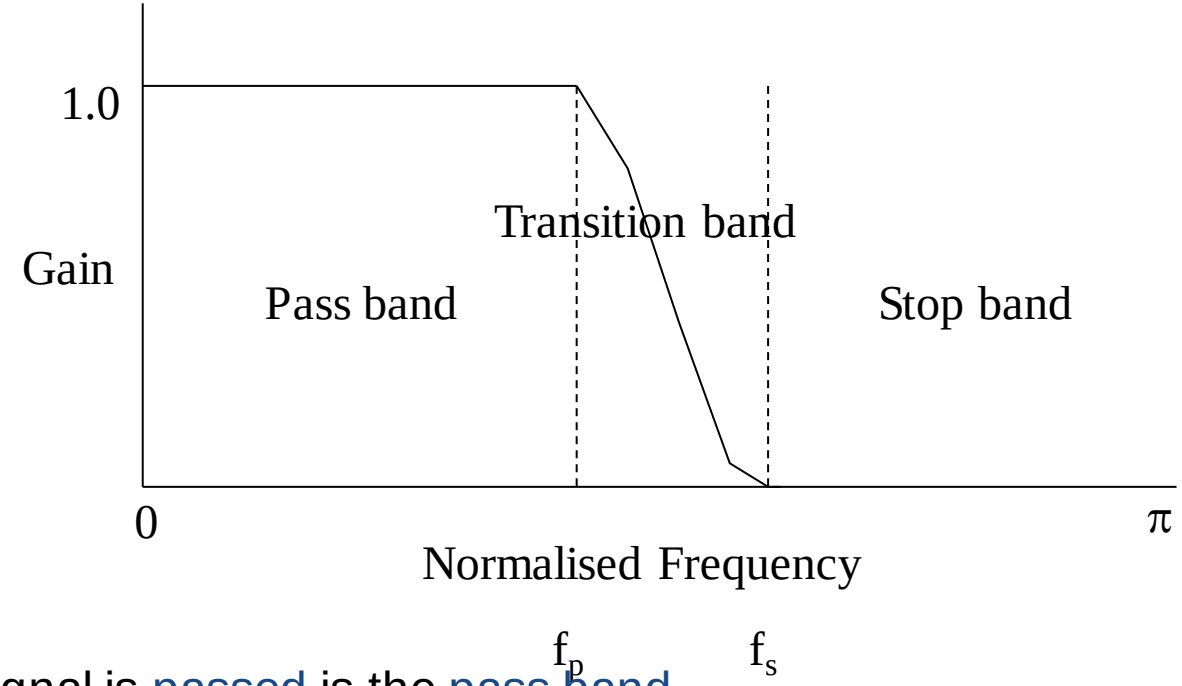
$$\angle(H(e^{j\Omega})) = \Omega(M-N) + \sum_{q=1}^{N} \angle(e^{j\Omega}-c_q) - \sum_{q=1}^{M} \angle(e^{j\Omega}-d_q)$$

Design of Filters

The 4 classical standard frequency magnitude responses are:

Lowpass, Highpass, Bandpass, and Bandstop

Consider e.g. Lowpass:

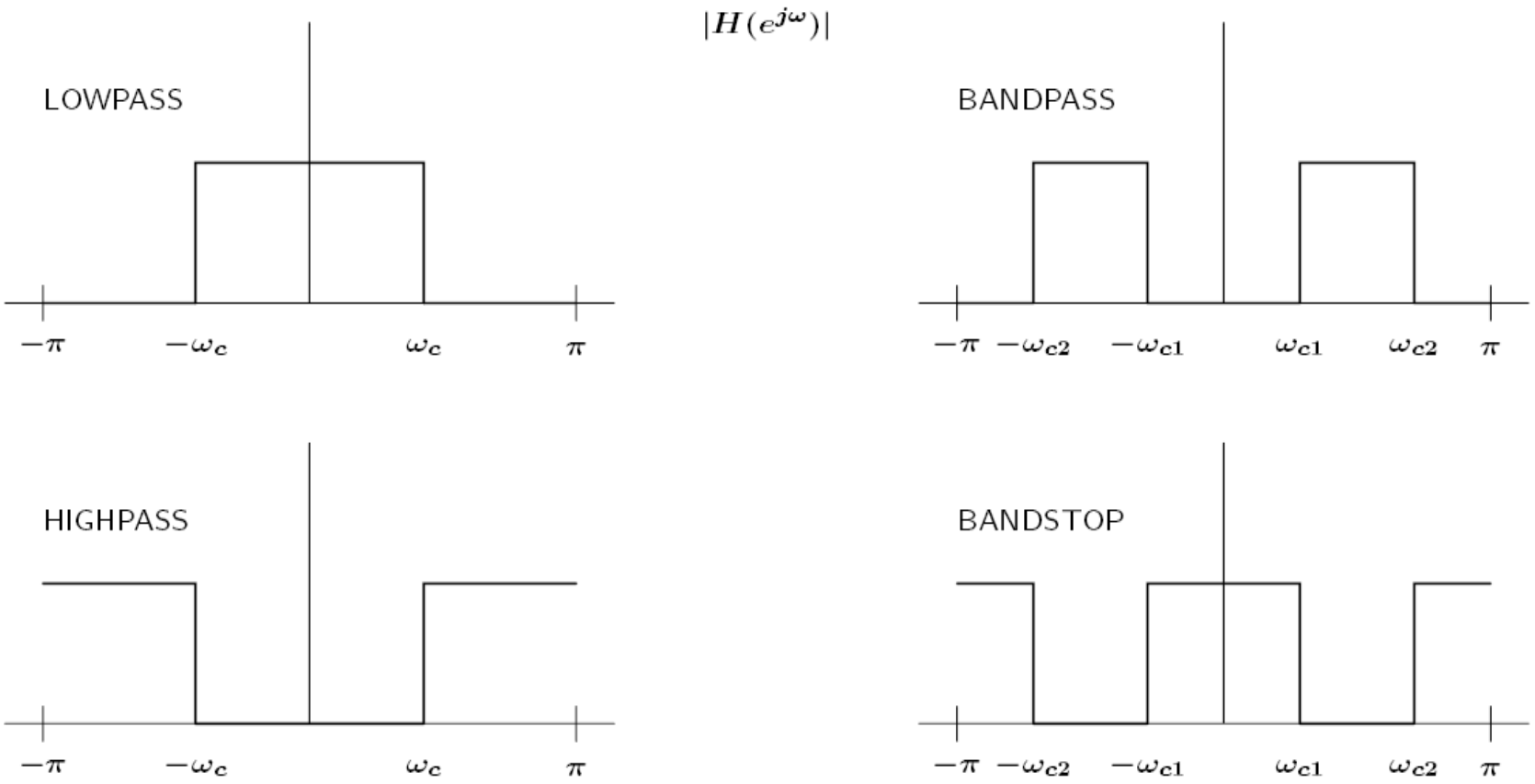


Frequency band where signal is passed is the pass band

Frequency band where signal is removed is stop band

Ideal Filters – Magnitude Response

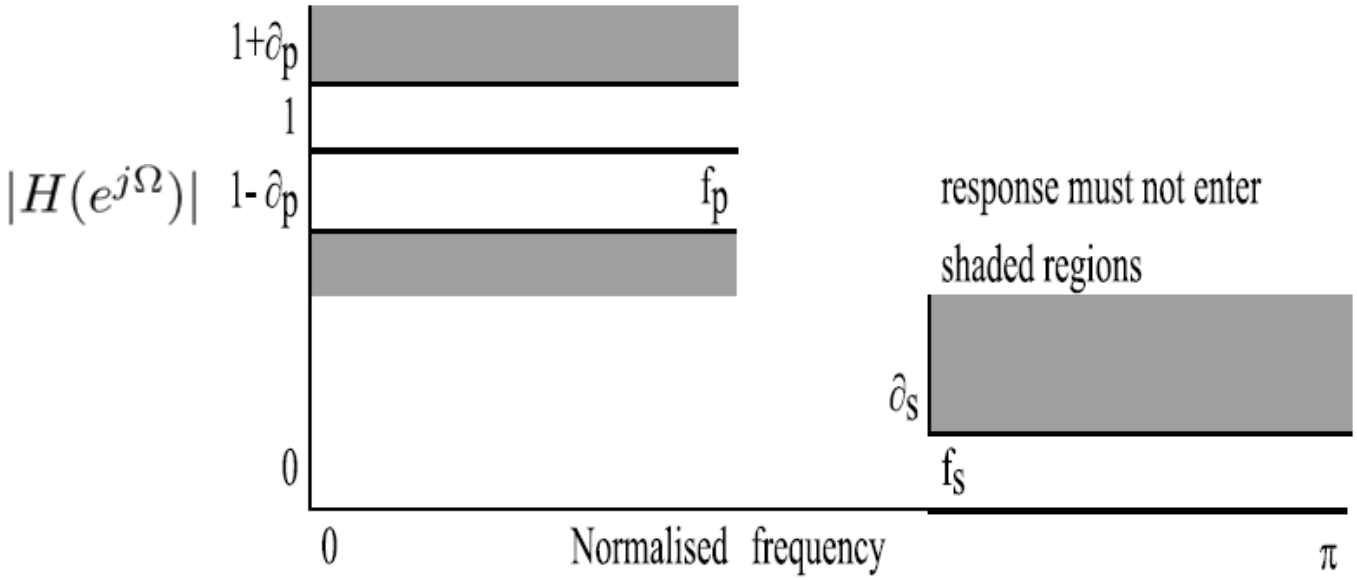
Ideal Filters are usually such that they admit a gain of 1 in a given *passband* (where signal is passed) and 0 in their *stopband* (where signal is removed).



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 $(1+\partial p)$ dB;

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

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

Example: $\partial p = 6\%$:

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

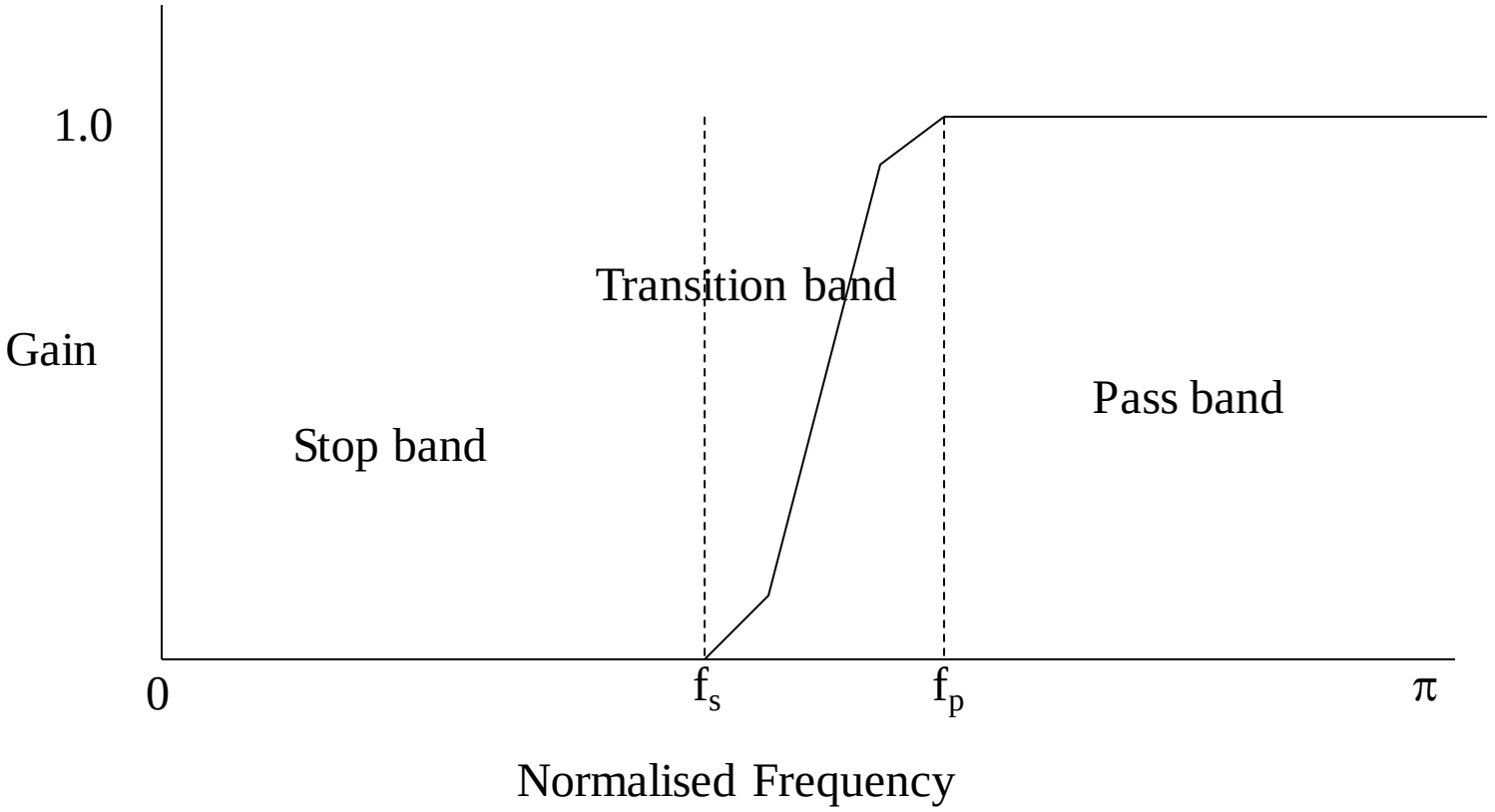
$\partial s = 0.01$:

minimum stopband attenuation = -20 log10 $(\partial s) = 40$dB.

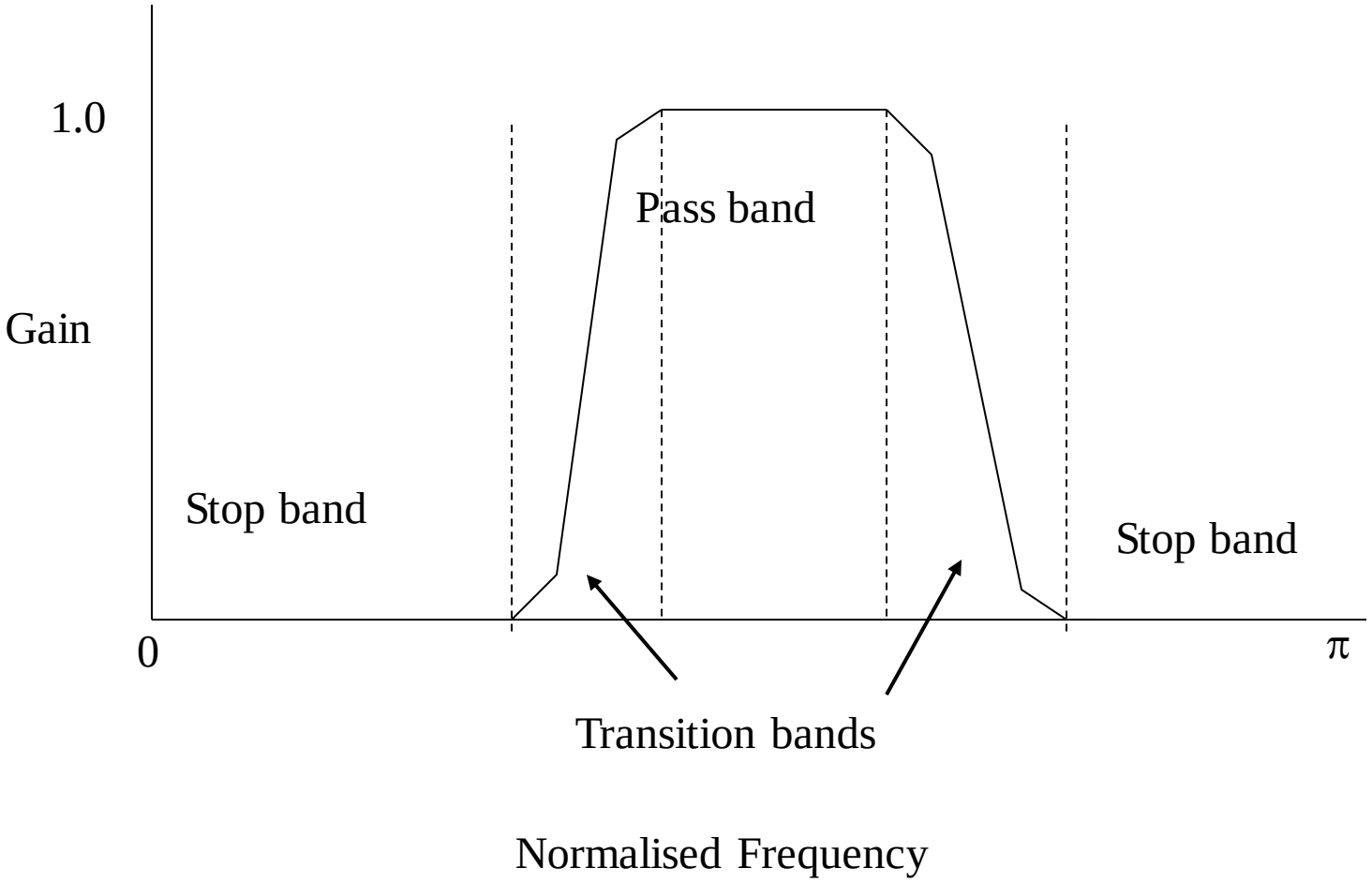
The bandedge frequencies are often called corner frequencies, particularly when associated with specified gain or attenuation (eg gain = -3dB).

Other standard responses:

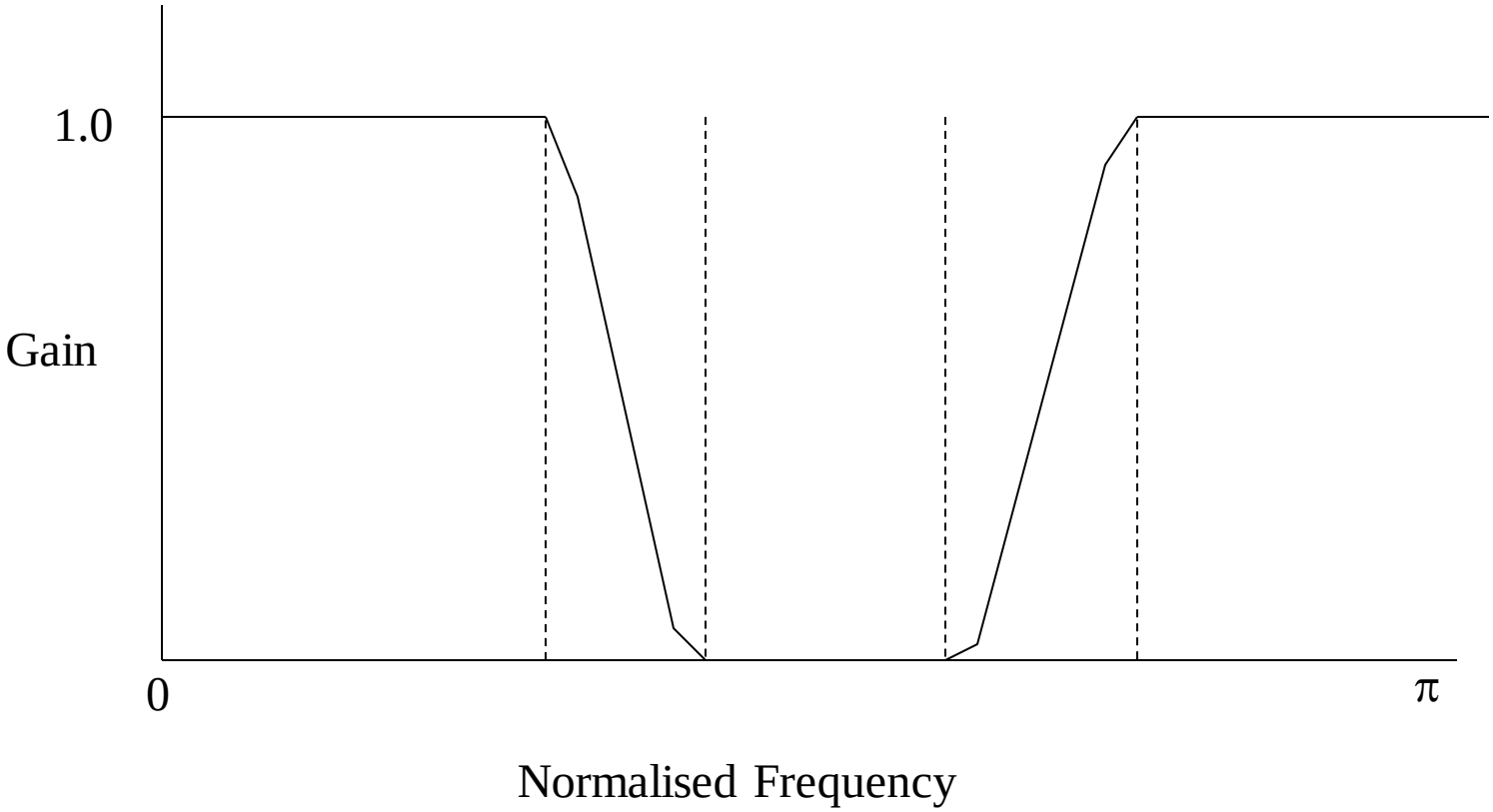
High Pass:



Band Pass:

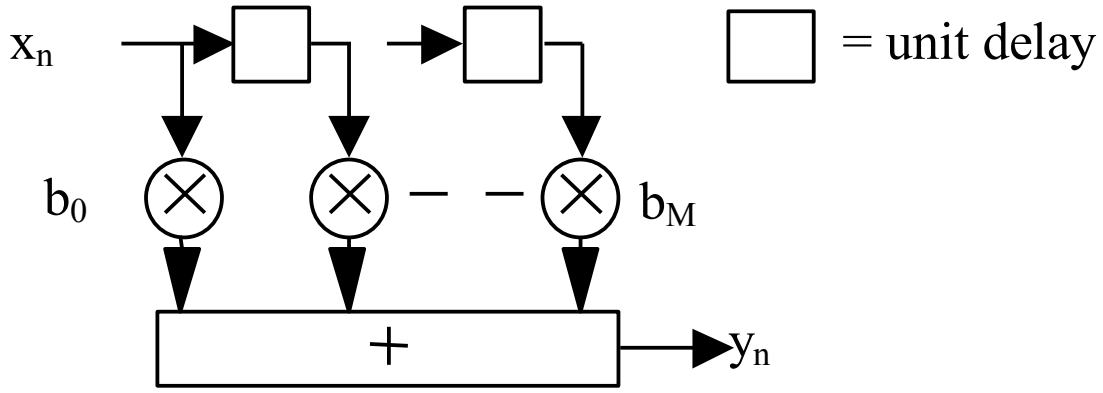

1.0
Gain
Pass band
Stop band
Stop band
0
π
Transition bands
Normalised Frequency

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 response 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):

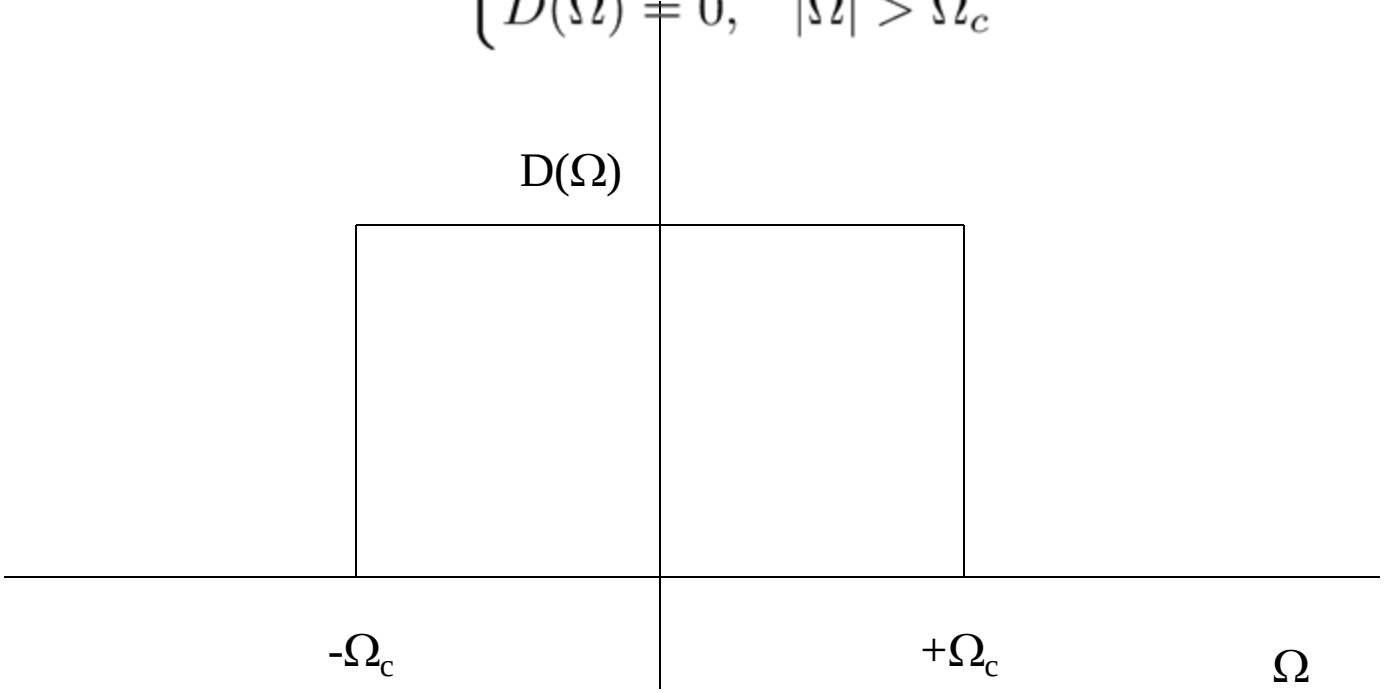
$$d_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(\Omega)\, e^{+jn\Omega}\, d\Omega$$

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,

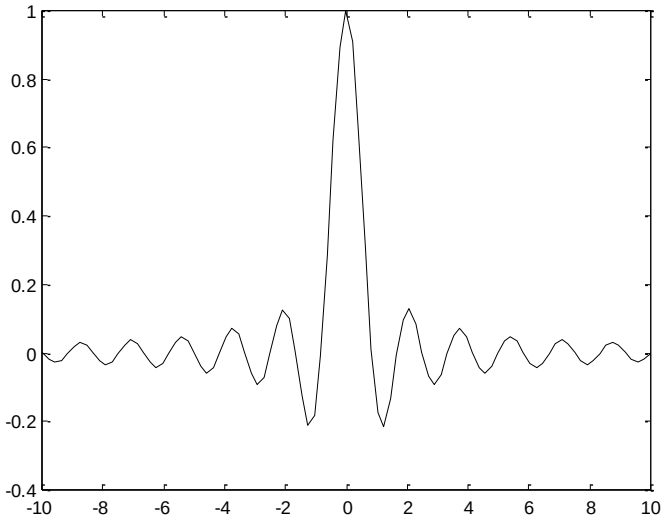
$$\begin{cases} D(\Omega) = 1, & |\Omega| < \Omega_c, \\ D(\Omega) = 0, & |\Omega| > \Omega_c \end{cases}$$



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 :



n ⟶

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

$$h_k = \begin{cases} d_k, & -M/2 \le n \le M/2 \\ 0, & \text{otherwise} \end{cases}, \rightarrow h'_k = h_{k-M/2}(\text{causal delay})$$

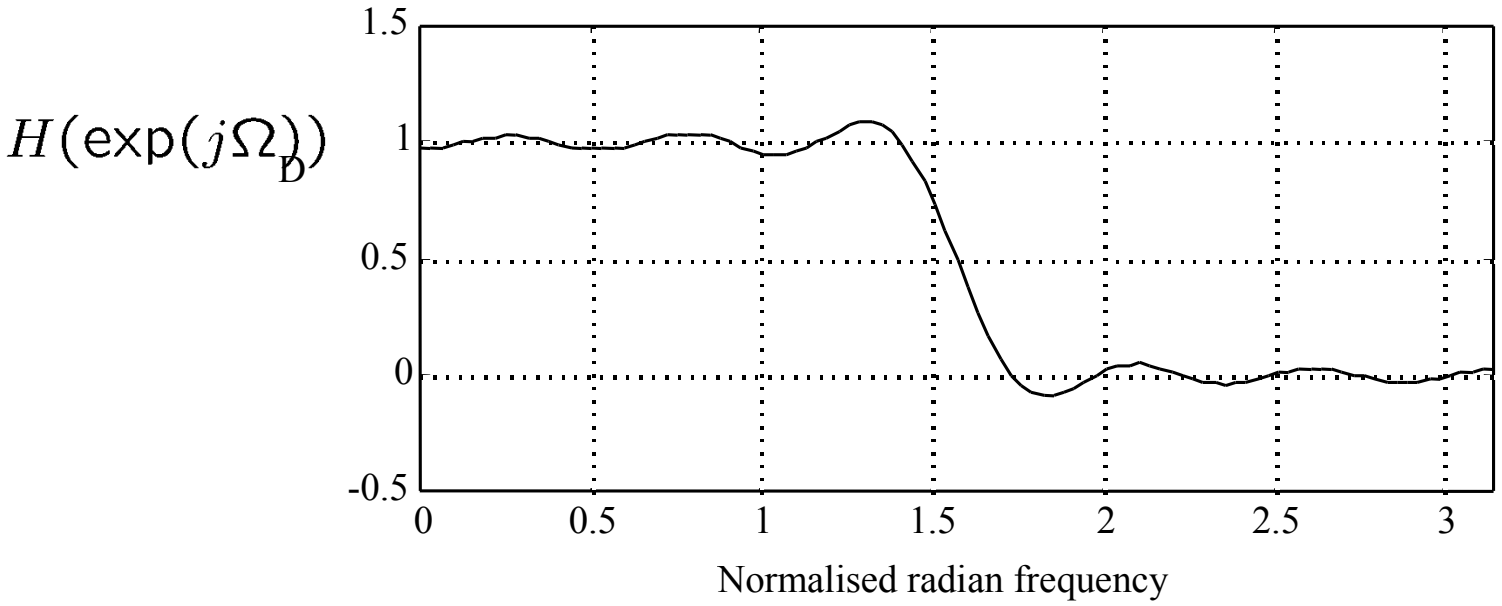
This selection operation is equivalent to multiplying the ideal coefficients by a **rectangular window** extending from -M/2 to M/2 (prior to delaying by M/2).

We can compute the resulting filter **frequency response**, which can now be thought of as a truncated Fourier series approximation of D(Ω), given by the DTFT of

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

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

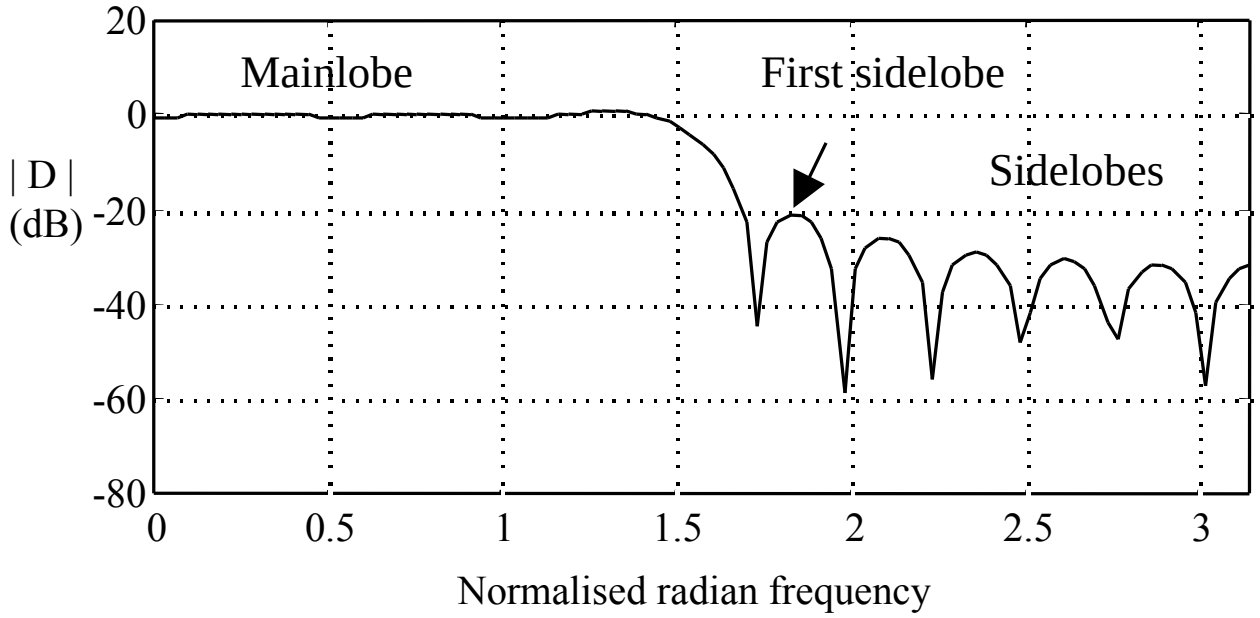
Note the well known **Gibb's phenomenon** (an oscillatory error, increasing in magnitude close to any discontinuities in $D(\Omega)$).



The actual filter would require an added delay of M/2 samples to make it causal. This 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.

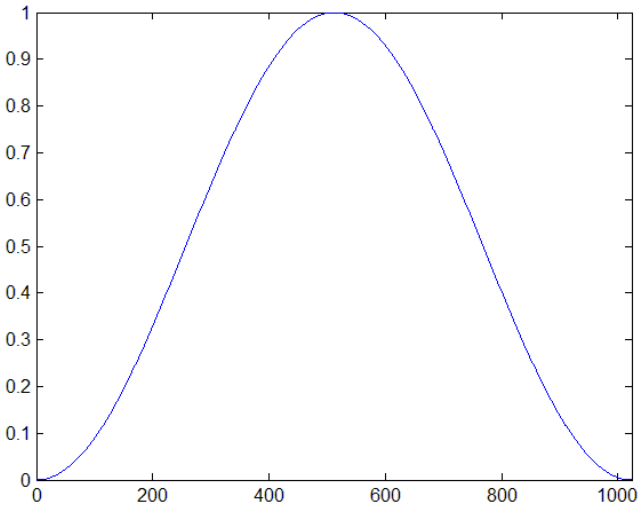
The sidelobes due to the rectangular window can be clearly seen:



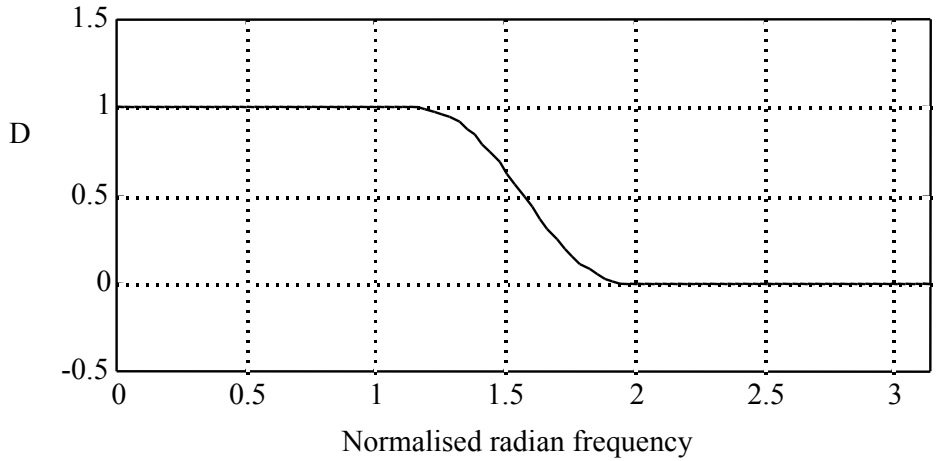
The **high sidelobe level** close to the passband, and the **slow decay** of sidelobe level away from the passband, make this an unsatisfactory response for most purposes.

Use of a window function

A good solution is to create the required finite number of filter coefficients by multiplying the (delayed by M/2) 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,

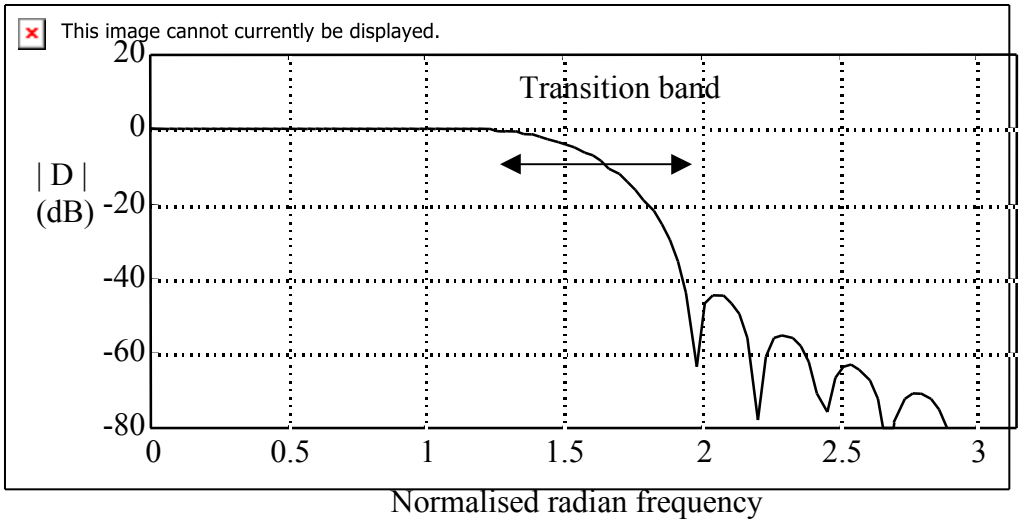


Leading to a much improved frequency response, illustrated below:



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:

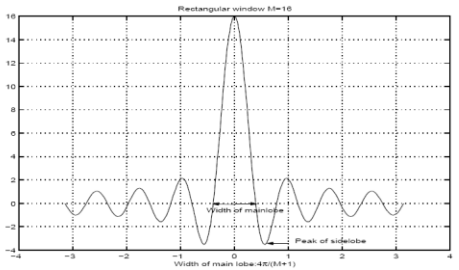
$$D(\Omega) = \begin{cases} 1, & \text{if } |\Omega| \leq \Omega_c \\ 0, & \text{if } \Omega_c < |\Omega| < \pi. \end{cases} \Leftrightarrow d_n = \frac{\Omega_c}{\pi} \frac{\sin \Omega_c n}{\Omega_c n}.$$

- 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:

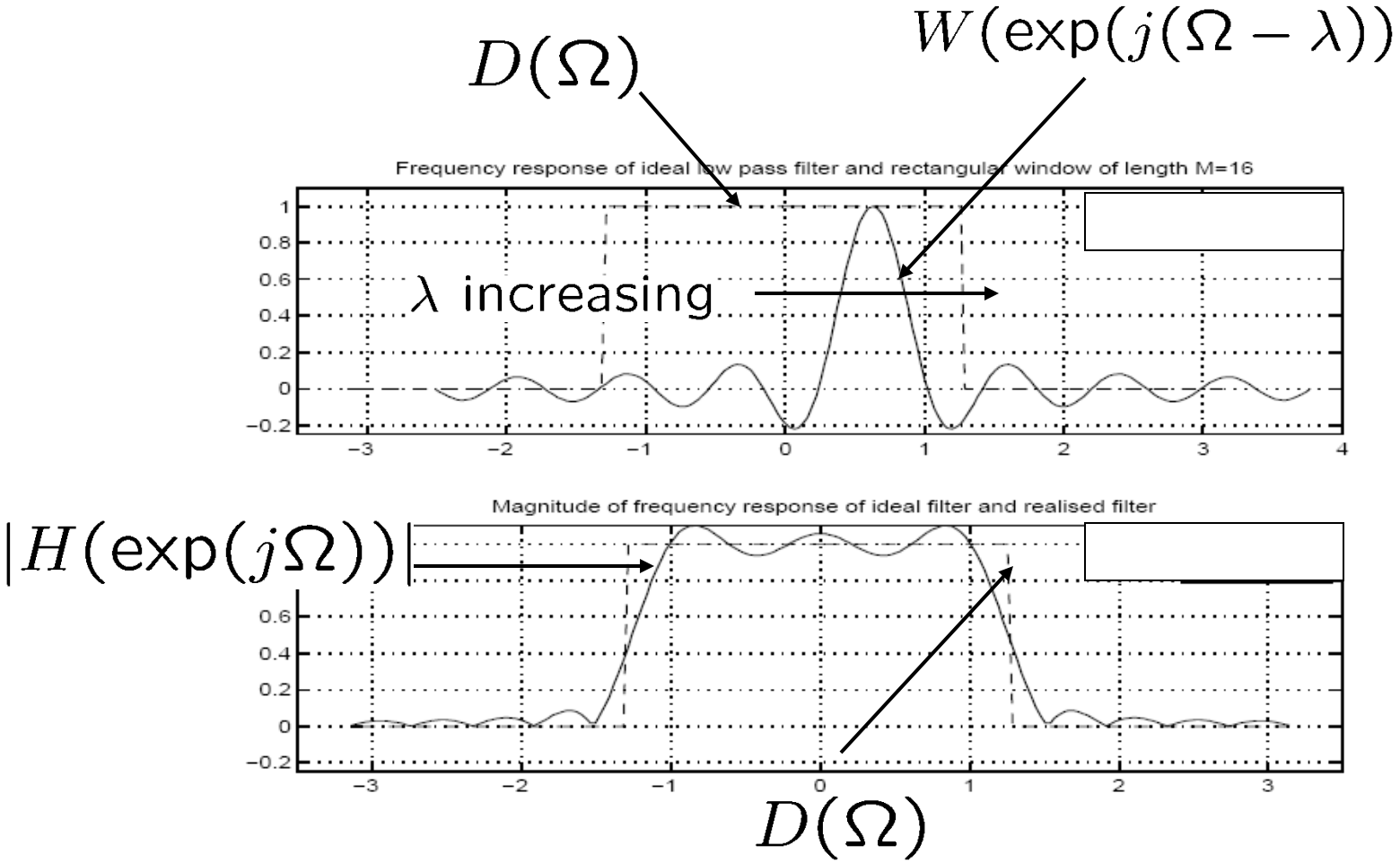
$$H(\exp(j\Omega)) = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(\lambda)\, W(\exp(j(\Omega - \lambda)))\, d\lambda$$

Frequency domain convolution

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

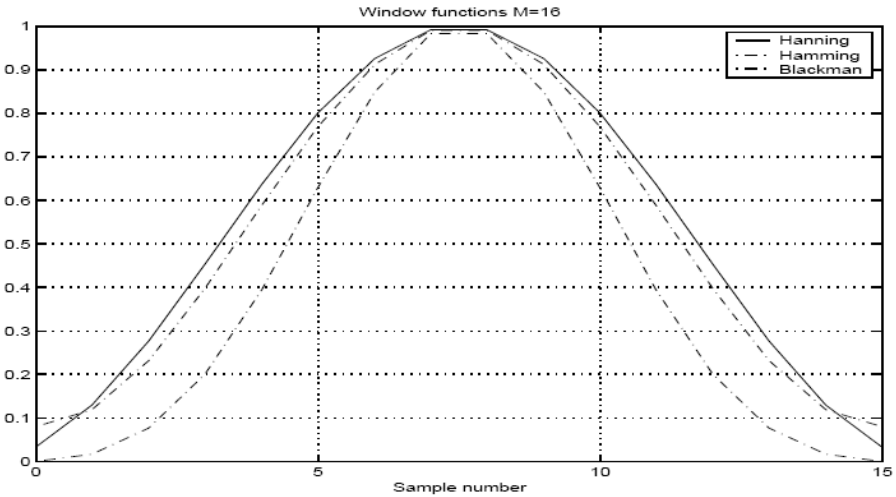


To see the effect of the frequency domain convolution, see the example below, for a rectangle window of length 16:

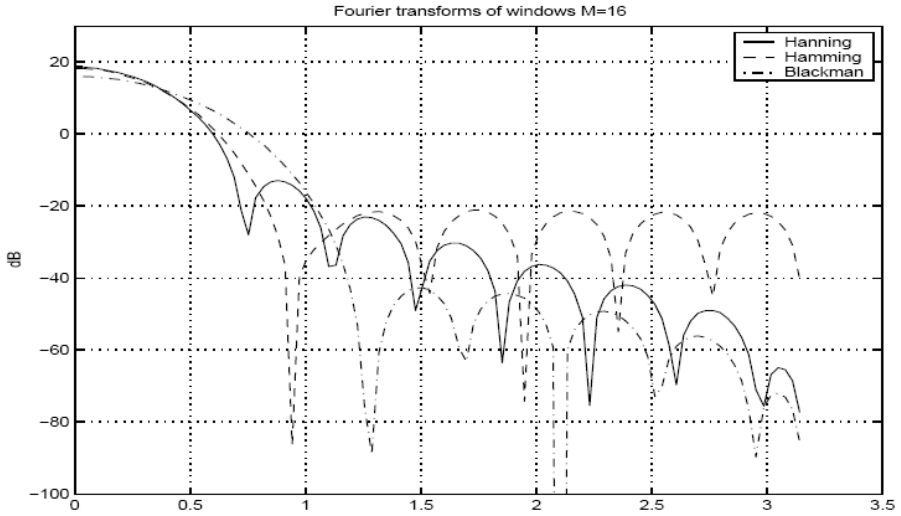


Example window functions:

w_n



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



Window functions

A commonly used family of window function is the Generalized Hamming Window:

$$w_n = \begin{cases} \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right), & n = 0, 1, ...N-1 \\ 0 & \text{Otherwise} \end{cases}$$

Some useful special cases are:

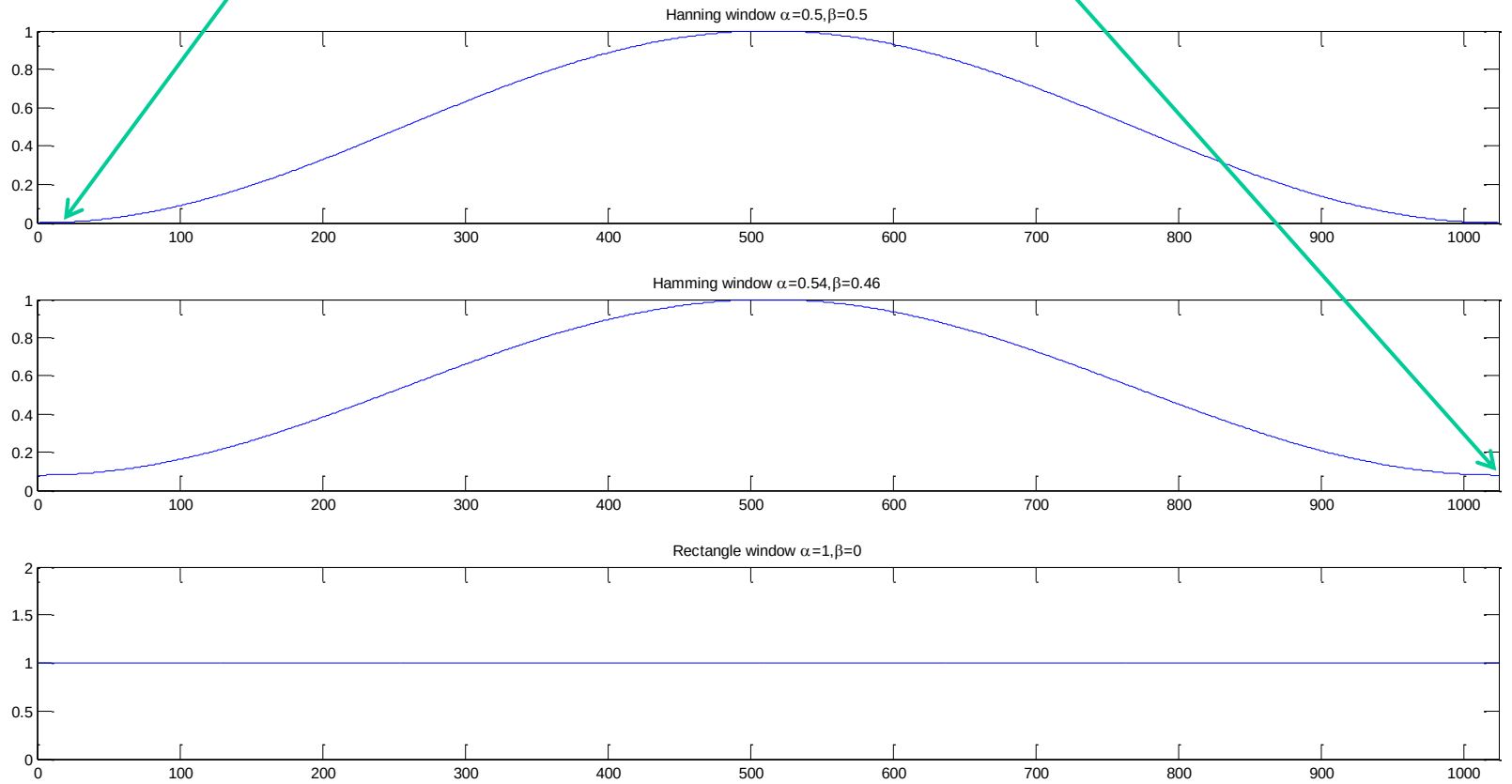
- $\alpha = 1$, $\beta = 0$: the rectangle (or Boxcar or Dirichlet) window has

$$w_n = 1, \;\; n = 0, 1, ...N-1$$

- $\alpha = \beta = 0.5$ corresponds to the Hanning or Hann window
- $\alpha = 0.54$, $\beta = 0.46$ corresponds to the Hamming window

Note Hanning window tapers to zero at edges, hence has faster roll-off of sidelobes at high frequencies, cf. Fourier Series

Hamming window has discontinuity at edges



Spectral Properties of Generalised Hamming Window Functions

This family of windows may be analysed in the frequency domain:

$$w_n = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right)$$

$$= \alpha - \frac{\beta}{2}\left(e^{j2\pi\frac{n}{N-1}} + e^{-j2\pi\frac{n}{N-1}}\right), \quad n = 0, 1, ..., N-1$$

But we have the standard result for DTFT of a modulated function $x'_n = \exp(j\phi n)x_n$:

$$X'(\exp(j\Omega)) = X(\exp(j\Omega - \phi)).$$

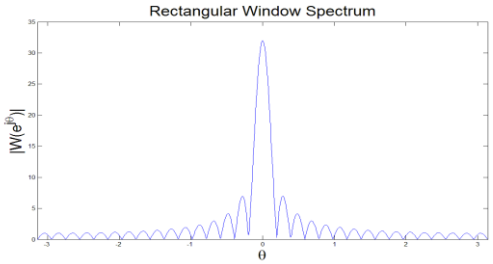
Hence the DTFT is obtained as the sum of three (shifted) rectangle window function spectra:

1 2 3

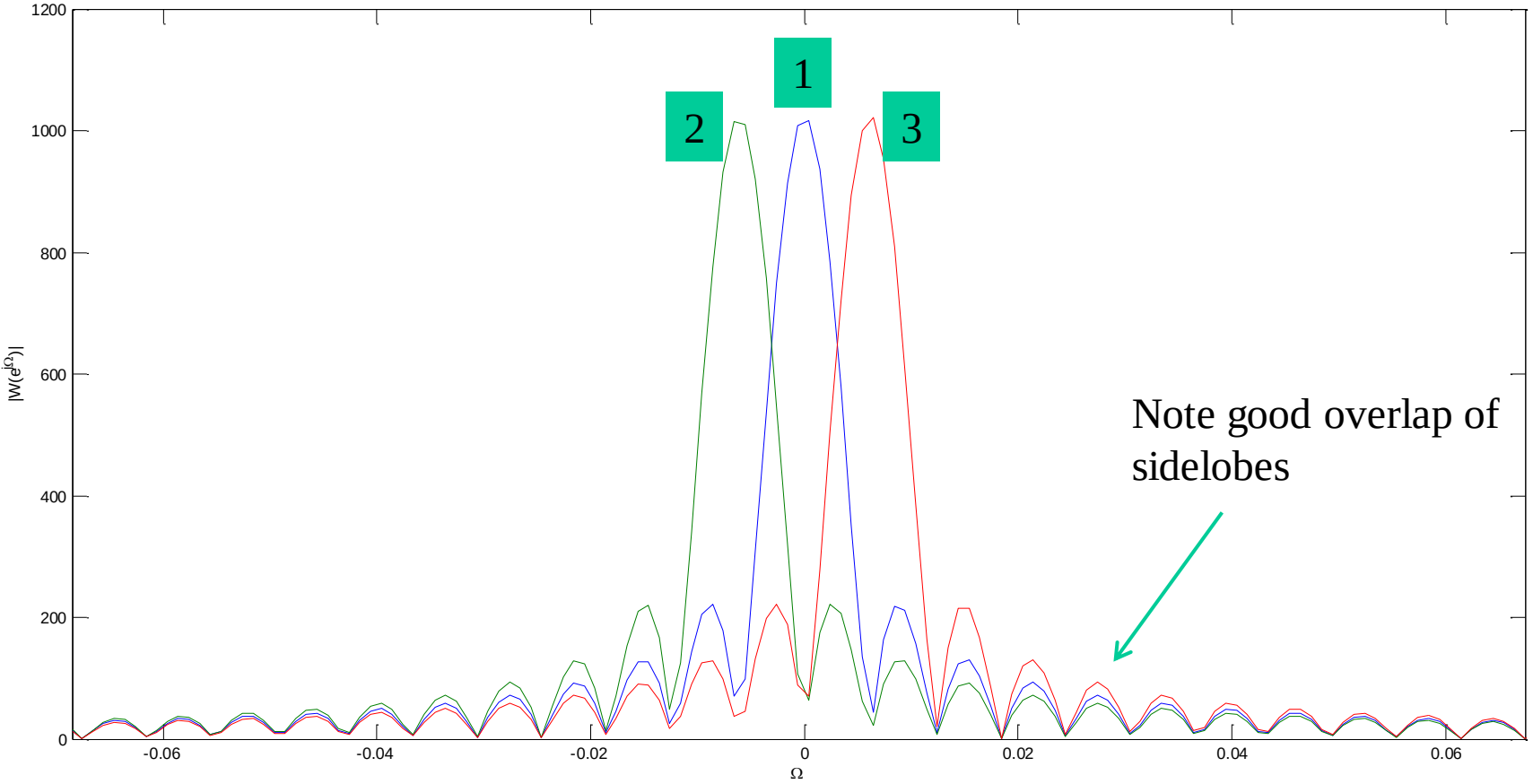
$$W(\exp(j\Omega)) = \alpha W_r(\exp(j\Omega)) - \frac{\beta}{2}\left(W_r\left(\exp\left(j\left(\Omega + \frac{2\pi}{N-1}\right)\right)\right) + W_r\left(\exp\left(j\left(\Omega - \frac{2\pi}{N-1}\right)\right)\right)\right)$$

with $W_r(\exp(j\Omega))$ the spectrum of the rectangular window:

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

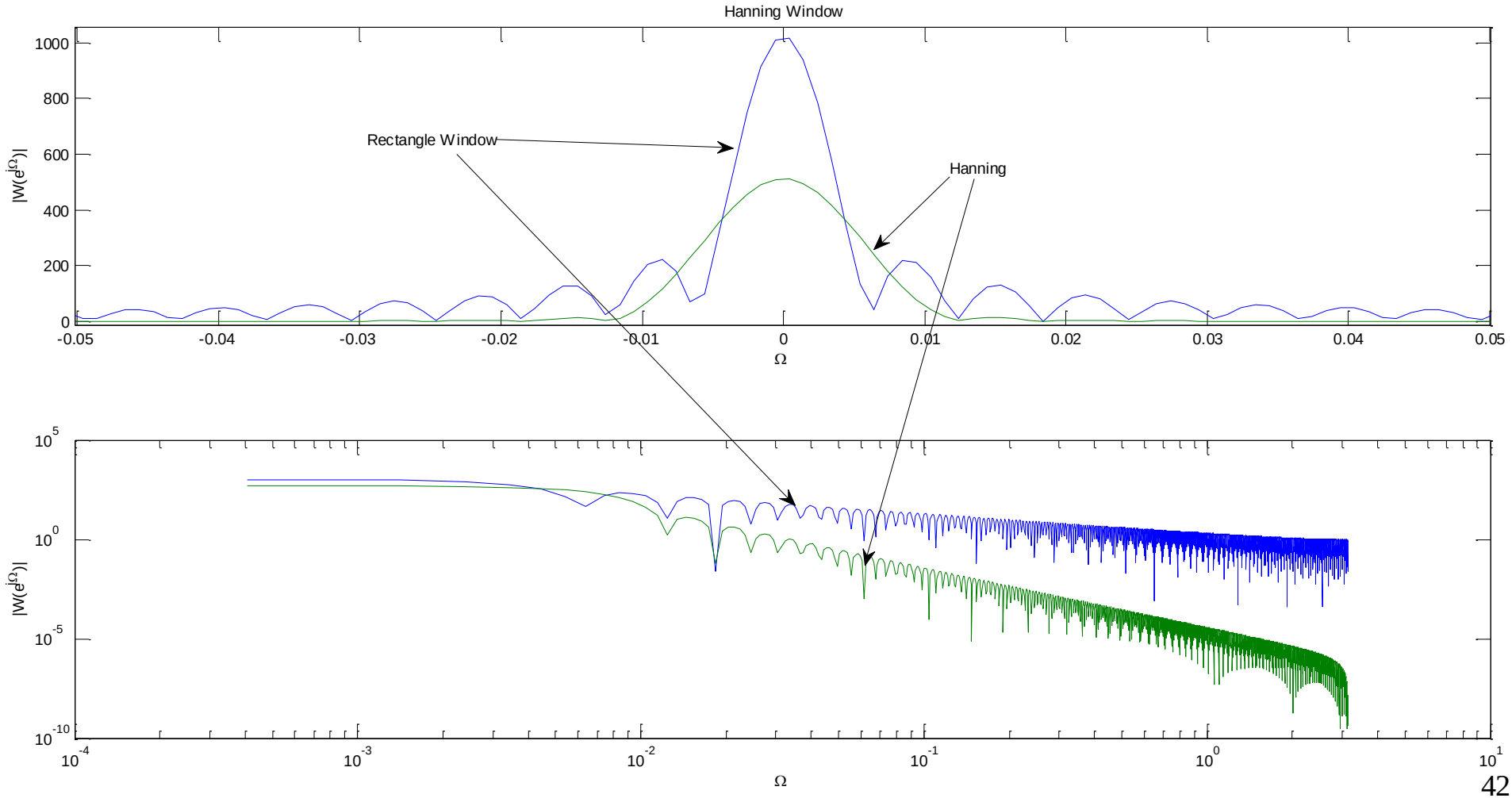


Note that the modulation factor $e^{-j\Omega\frac{N-1}{2}}$ vanishes if the window is time-shifted to be symmetric about $n = 0$.


1200
1000
800
600
400
200
0
|W(e^{jΩ})|
-0.06
-0.04
-0.02
0
0.02
0.04
0.06
Ω
1
2
3
Note good overlap of sidelobes

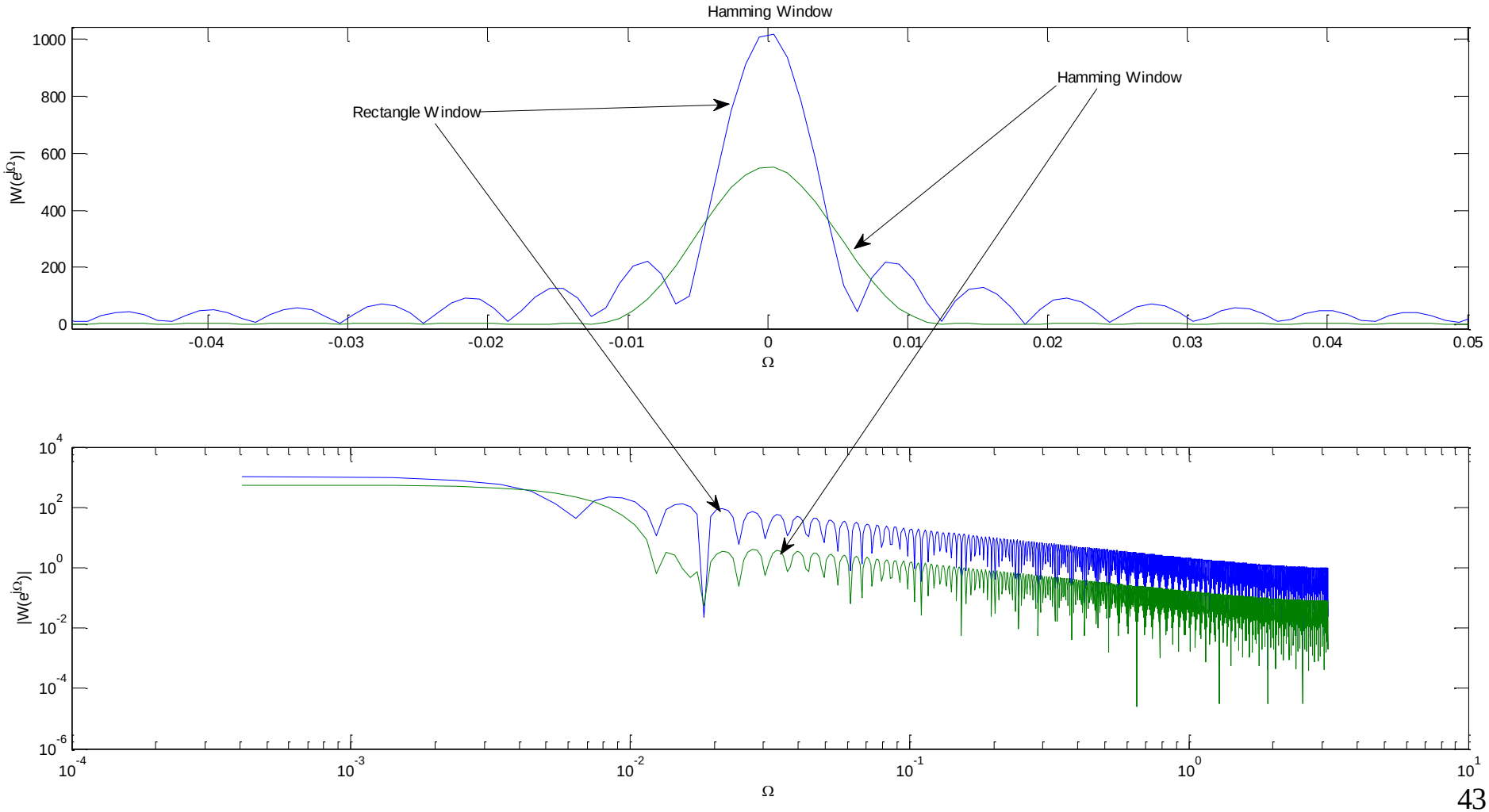
Hanning Window

Hanning Window achieves good sidelobe cancellation, but broader central lobe compared to rectangle window.
Sidelobes decay much more rapidly and linearly on the log-log scale

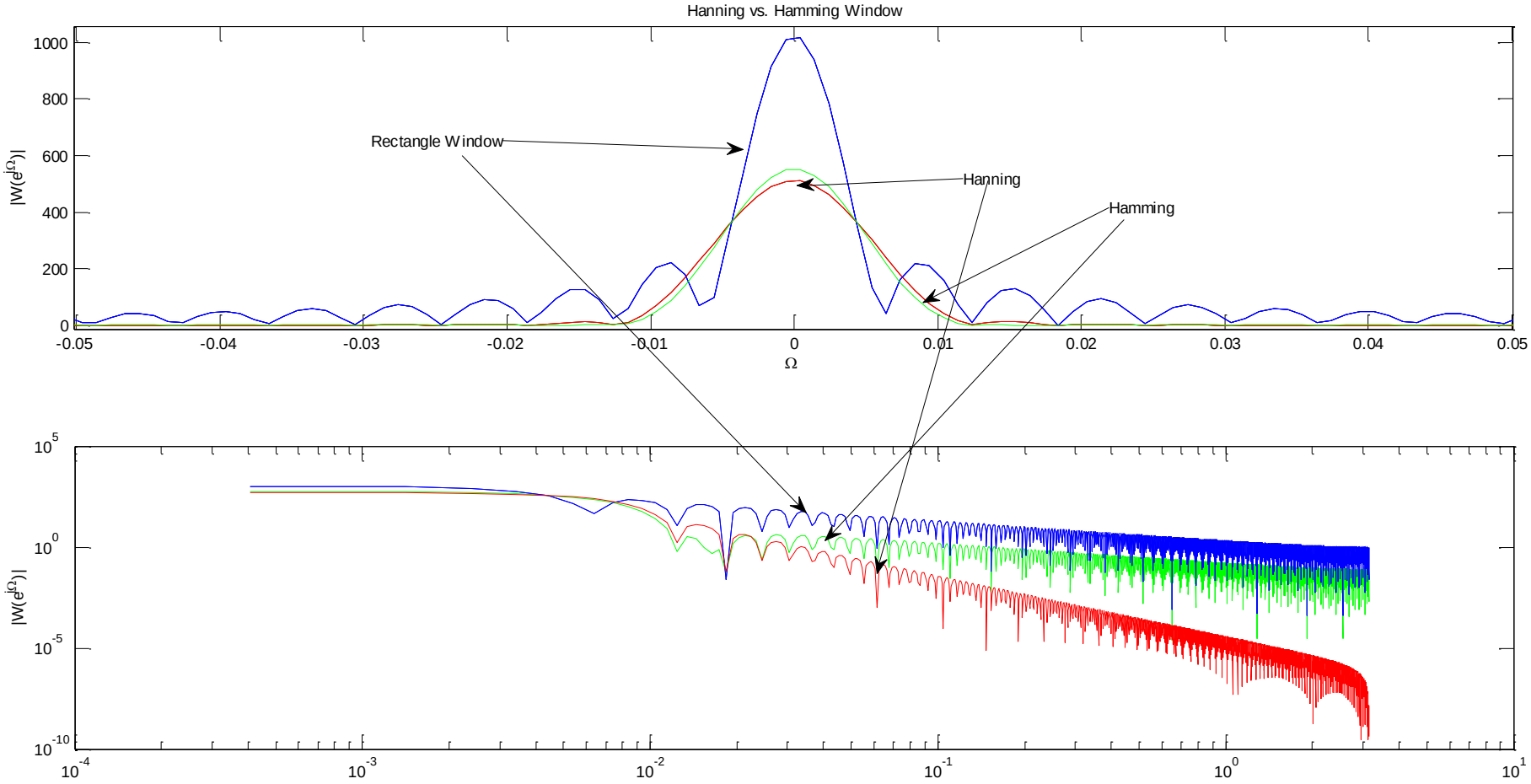


Hamming Window

Hamming Window achieves close to optimal cancellation of the second sidelobe, but again broader central lobe compared to rectangle window.
Sidelobes are nearly constant amplitude



Hanning vs. Hamming Windows:



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.

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.

1.Select a suitable window function.

2.Specify an 'ideal' response $D(\Omega)$.

3.Compute the coefficients of the `ideal' filter.

4.Multiply the ideal coefficients by the window function to give the filter coefficients and delay to make causal

5.Evaluate the frequency response of the resulting filter, and iterate 1-5 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

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 largest 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 $3.1/N = 3.1/25 = 0.124$).

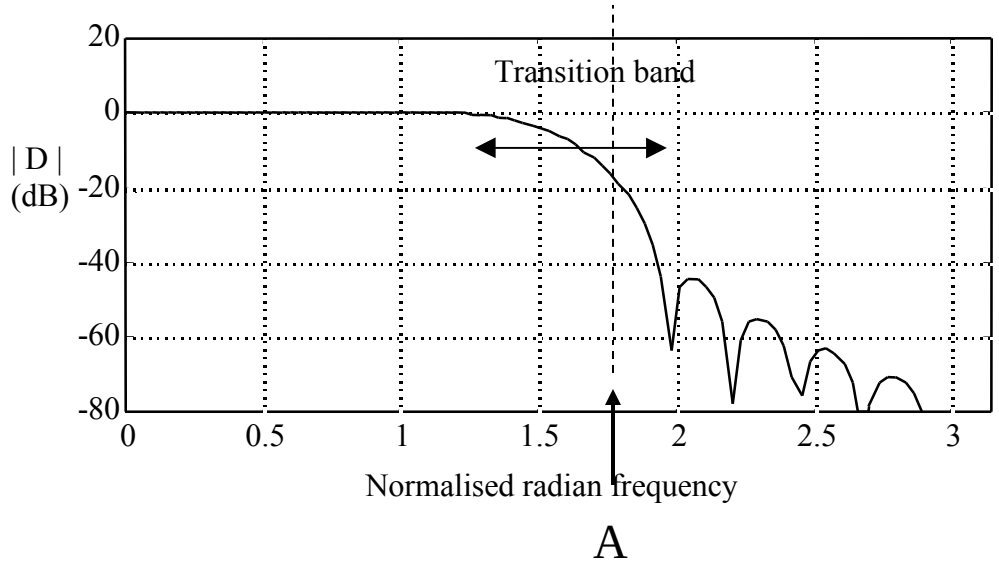
Using the table, the required stopband attenuation (50dB) can probably be obtained by the use of Hamming, Blackman or Kaiser windows.

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:



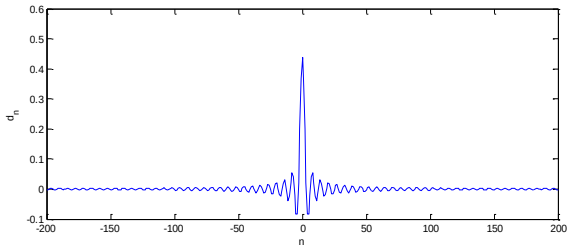
Hence choose an 'ideal' bandedge A which lies in the **middle** of the wanted transition region, i.e. frequency = 1.5+0.5/2 = 1.75 kHz

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 $D(\Omega)$,

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



In practice, this may not be computable in closed form for more complex $D(\Omega)$. Can then approximate by numerical integration, or by discretisation of $D(\Omega)$ on a fine grid and inverse DFT-ing, i.e. take inverse DFT of:

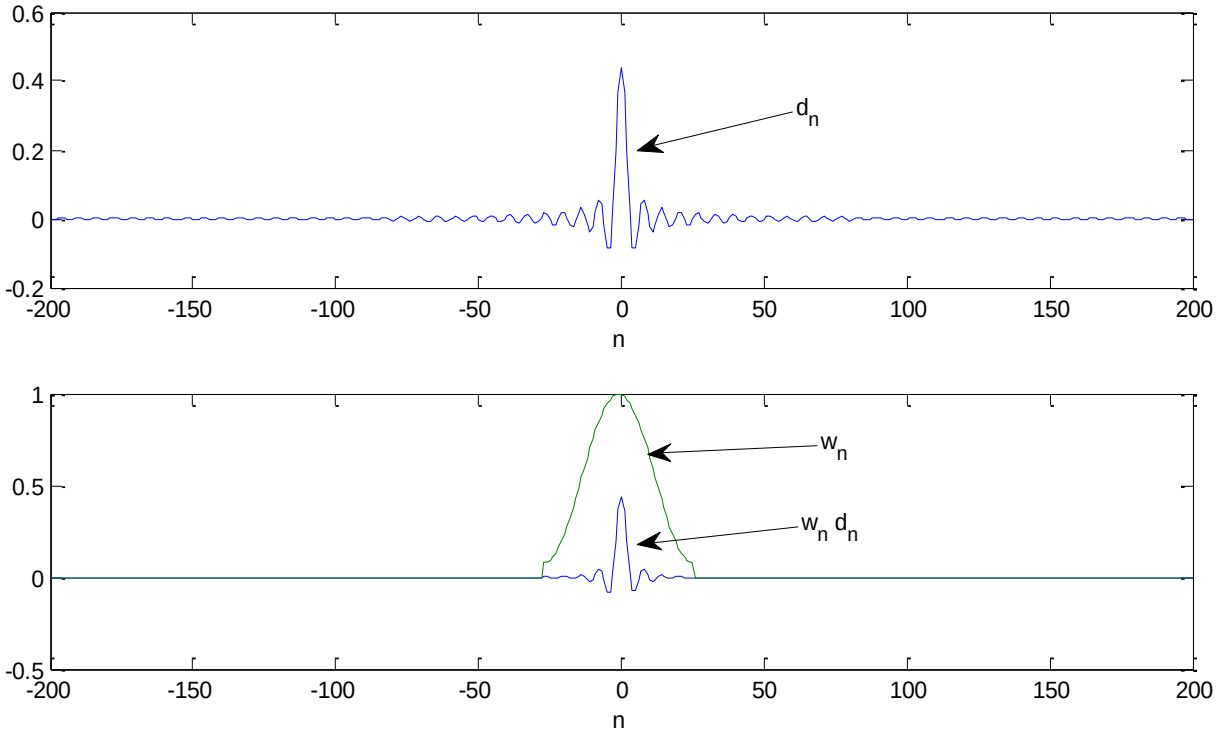
$$D[p] = D(2\pi p/N), \quad p = 0, 1, ...N/2 - 1.$$

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

where the second condition ensures that the coefficients are real-valued.

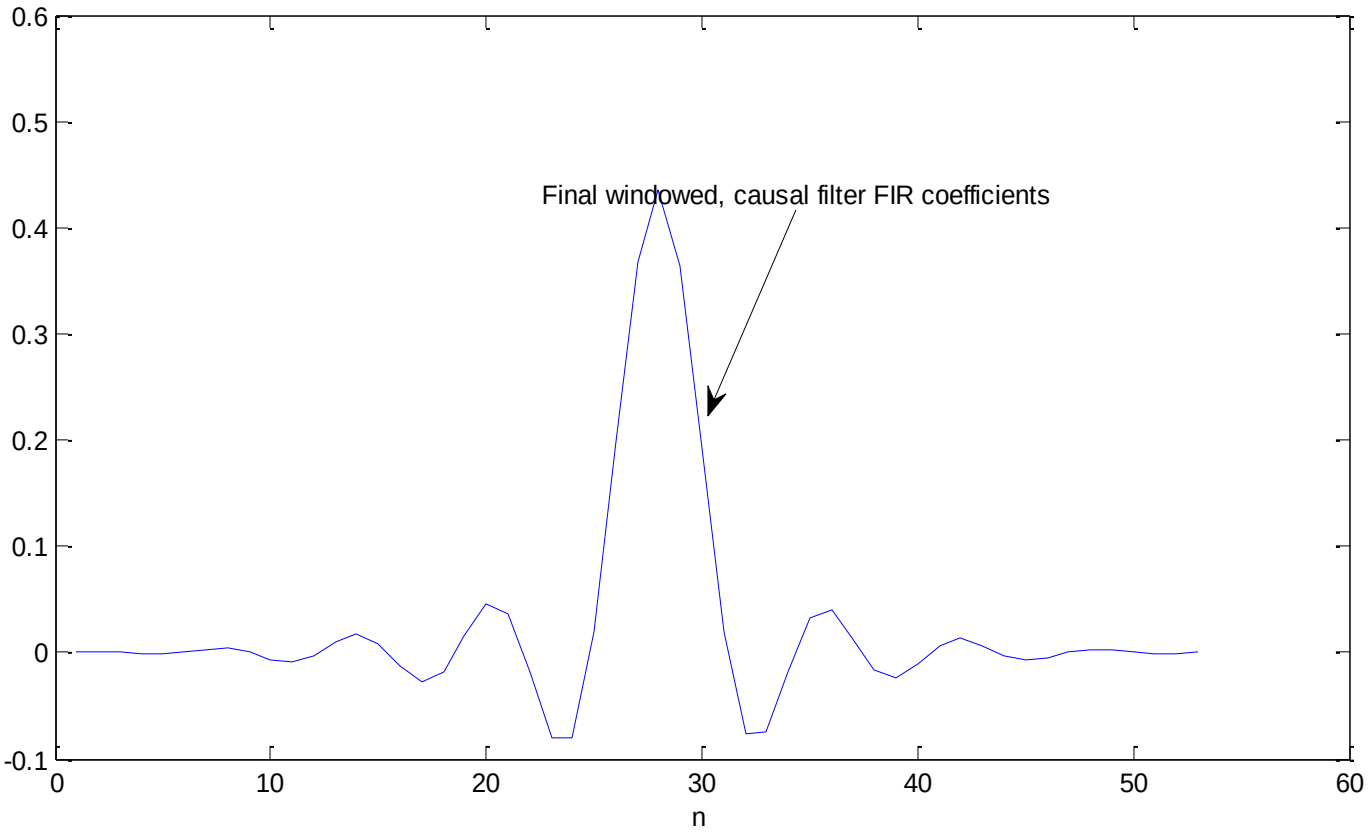
STEP 4 – Multiply to obtain the filter coefficients

Now multiply by the chosen window function, i.e. Hamming with length 53:



Notice though that the filter is non-causal since it has non-zero values for n<0.

This is `fixed' by delaying the impulse response by (in this case) 26 samples. The resulting frequency magnitude response is unchanged, but a constant delay of 26 samples is introduced into the filter output (= a linear phase term in the frequency response):


Final windowed, causal filter FIR coefficients
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0
10
20
30
40
50
60
n

Step 5 – Evaluate the frequency response and iterate

Compute the resulting frequency response of the windowed filter.

If the resulting filter does not meet the specifications, either adjust $D(\Omega)$ (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.

In our example the specifications are almost met. A small reduction in the edge frequency A of the ideal response, and repeat of the design process steps 2-4, 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 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):

$$\begin{aligned} H(\exp(j\Omega)) &= \exp(-jM\Omega/2) \\ &\quad \times \Big\{ b_0 \exp(jM\Omega/2) + b(M) \exp(-jM\Omega/2) \\ &\quad + b_1 \exp(j(M-2)\Omega/2) + b_{M-1} \exp(-j(M-2)\Omega/2) \\ &\quad + \ldots \Big\} \end{aligned}$$

If the filter <u>length</u> 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

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

$$= \exp(-jM\Omega/2)\boldsymbol{\phi}(\Omega)^T \mathbf{b}$$

where

$$\mathbf{b} = [b_0,\, b_1,\, \ldots b_{M/2}]^T$$

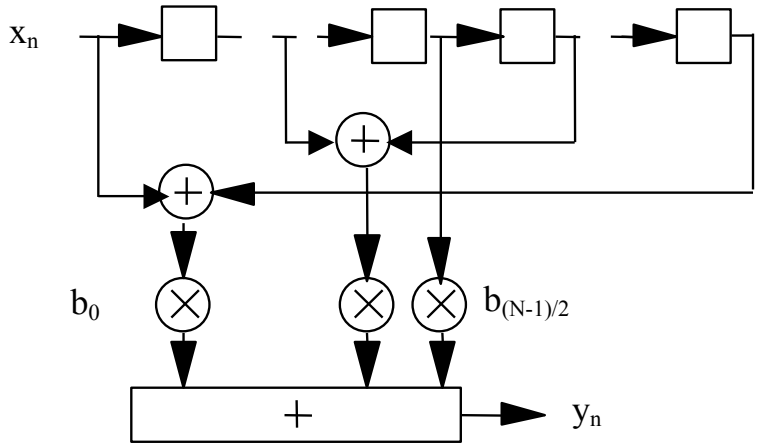
and

$$\boldsymbol{\phi}(\Omega) = 2[\cos(M\Omega/2),\, \cos((M-2)\Omega/2),\, \ldots,\, \cos(\Omega/2)]^T$$

This is a **purely real** function A(Ω) (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 – i.e. there is no relative phase distortion of the frequency components of the signal.

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 $\pi/2-m\Omega/2$)

Note that symmetric FIR filters 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:



Least Squares Linear Phase Filter Design

We can easily specify a least squares problem for design of a Linear Phase filter. Specify the desired response at a pre-chosen set of P frequencies $\{\Omega_i\} < \pi$:

$$A_d(\Omega_i), \;\; i = 1, 2, ..., P$$

Now specify the error between the actual gain and the desired gain for a filter with coeffiicent vector $\mathbf{b}$:

$$\epsilon(\Omega_i) = A(\Omega_i) - A_d(\Omega_i) = \boldsymbol{\phi}(\Omega_i)^T \mathbf{b} - A_d(\Omega_i).$$

Now form all of the errors into a vector:

$$\boldsymbol{\epsilon} = \boldsymbol{\Phi}\mathbf{b} - \mathbf{A}_d$$

where

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\Omega_1)^T \\ \boldsymbol{\phi}(\Omega_2)^T \\ ... \\ \boldsymbol{\phi}(\Omega_P)^T \end{bmatrix}, \;\; \mathbf{A}_d = \begin{bmatrix} A_d(\Omega_1) \\ A_d(\Omega_2) \\ ... \\ \\ A_d(\Omega_P) \end{bmatrix}$$

Now specify the least squared error:

$$\begin{aligned} E &= \sum_{i=1}^{P} \epsilon(\Omega_i)^2 = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\ &= (\boldsymbol{\Phi}\mathbf{b} - \mathbf{A}_d)^T (\boldsymbol{\Phi}\mathbf{b} - \mathbf{A}_d) \\ &= \mathbf{b}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi}\mathbf{b} + \mathbf{A}_d^T \mathbf{A}_d - 2\mathbf{A}_d^T \boldsymbol{\Phi}\mathbf{b} \end{aligned}$$

and differentiate to find the minimum wrt $\mathbf{b}$:

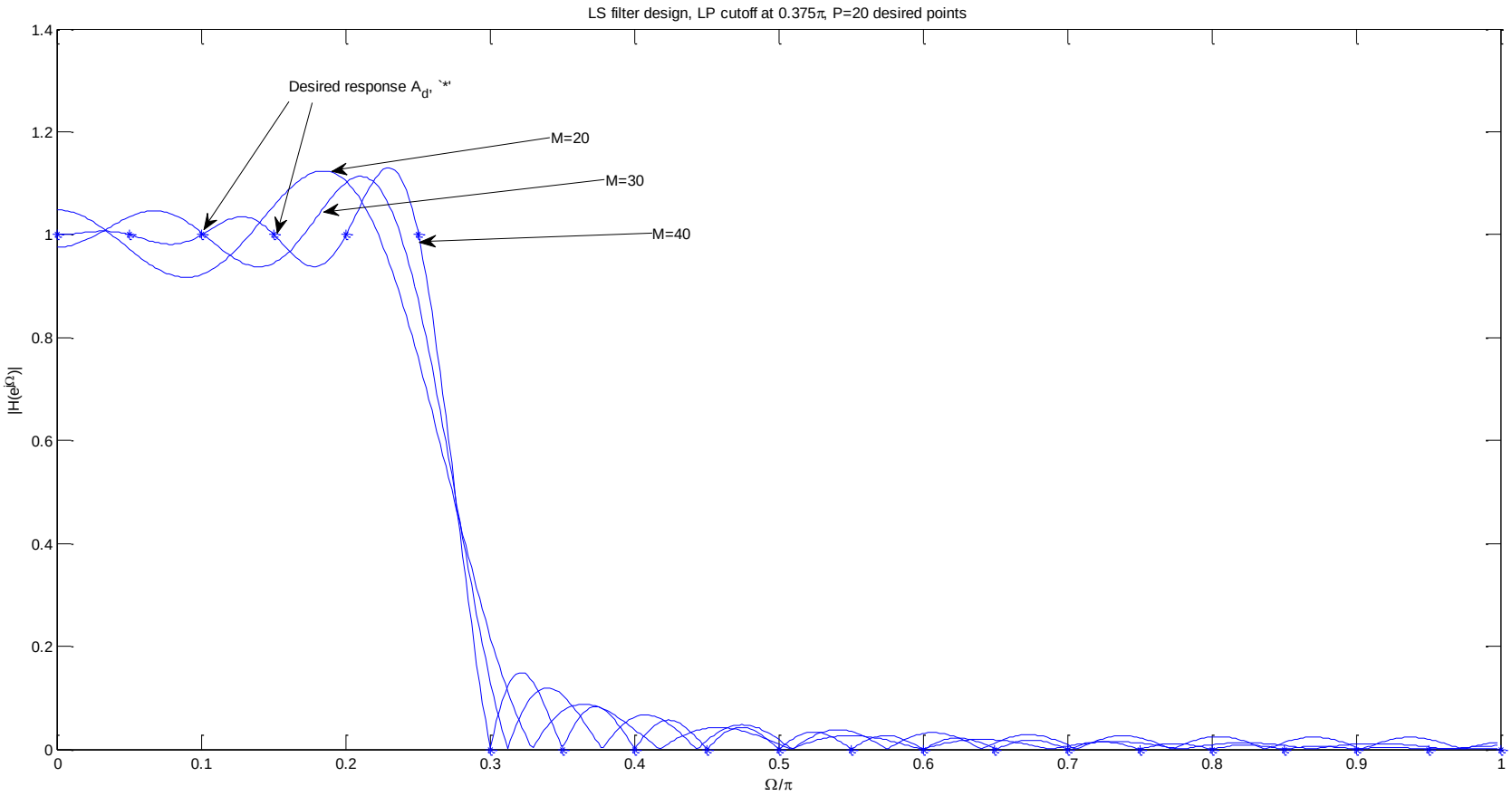
$$\frac{dE}{d\mathbf{b}} = 2\boldsymbol{\Phi}^T \boldsymbol{\Phi}\mathbf{b} - 2\boldsymbol{\Phi}^T \mathbf{A}_d = 0.$$

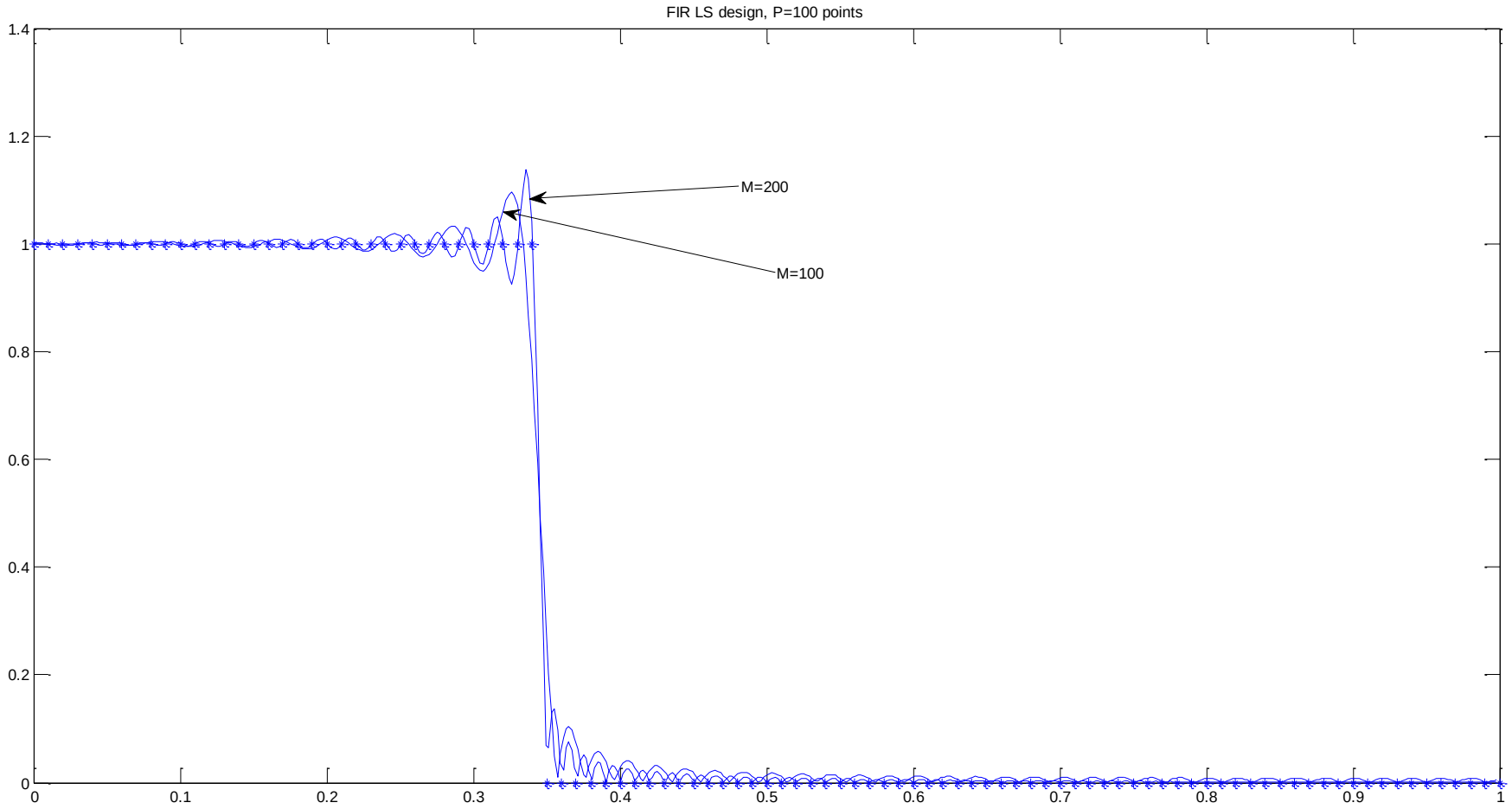
This has a unique solution if $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ is full rank, i.e. $P \geq (M-1)/2$ and all Ω_i are distinct from one another:

$$\mathbf{b}_{LS} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{A}_d.$$

This method, however, does not guarantee the size or nature of the error E, but is simple to implement and the result is obtained without iteration.

Example low-pass LS filter design:




FIR LS design, P=100 points
M=200
M=100
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Minimax Design of Linear Phase FIR filters

A second method of Linear Phase FIR design considered is **non-linear optimisation.**

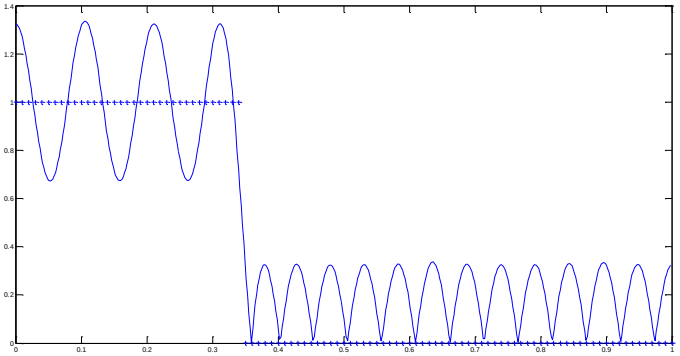
For example the classic algorithm devised by Parks and McClellan, which **designs linear phase (symmetric) filters** or **antisymmetric filters**. The filters designed by the Parks and McClellan algorithm have minimised maximum error ("**minimax error**") with respect to a given target magnitude frequency response, i.e. minimise the following error with respect to the filter coefficients:

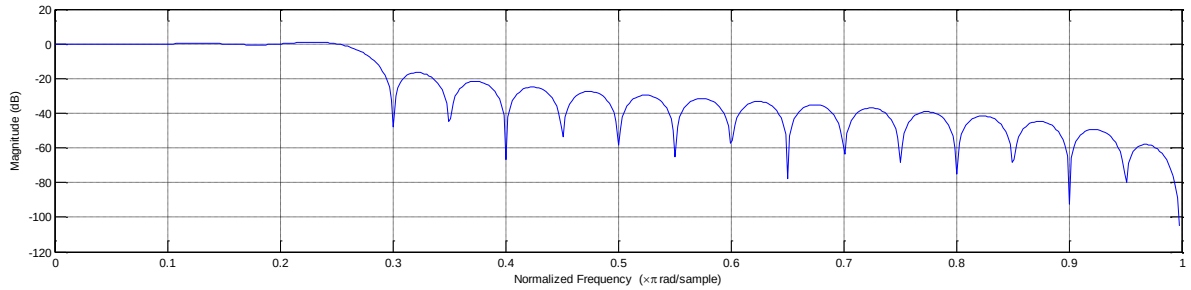
$$\min_{\mathbf{b}} \max_{\Omega} |A_d(\Omega) - A(\Omega)|$$

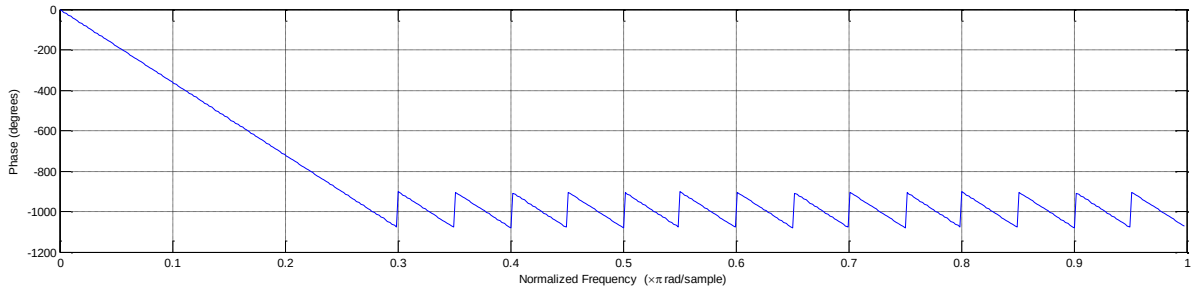
The method uses an efficient algorithm called the Remez exchange algorithm. Better performance with guaranteed error bounds compared to LS filter design, but more complex to implement the design.

Example: Remez Exchange designed FIR filter with order M=40, desired response shown as `+'.

|H|







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.

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

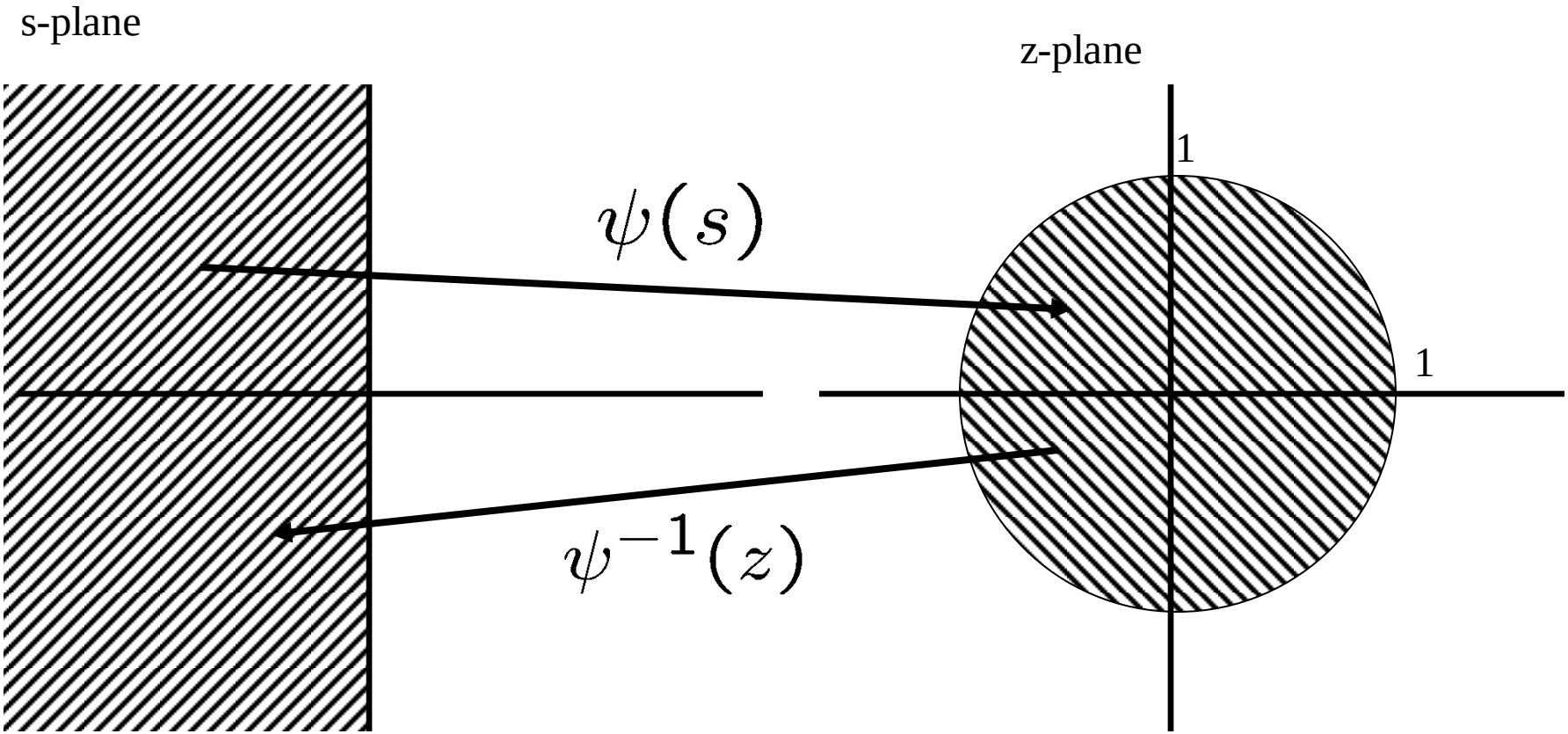
$$|z|^2 = \frac{(1+0)^2+\omega^2}{(1-0)^2+\omega^2} = 1$$

   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.

Hence the BLT preserves the following important features of $H(j\omega)$:

(1) the $\omega \leftrightarrow \Omega$ mapping is monotonic, and

(2) $\omega = 0$ is mapped to $\Omega = 0$, and $\omega = \infty$ is mapped to $\Omega = \pi$ (half the sampling frequency). Thus, for example, a lowpass response that decays to zero at $\omega = \infty$ produces a lowpass digital filter response that decays to zero at $\Omega = \pi$.

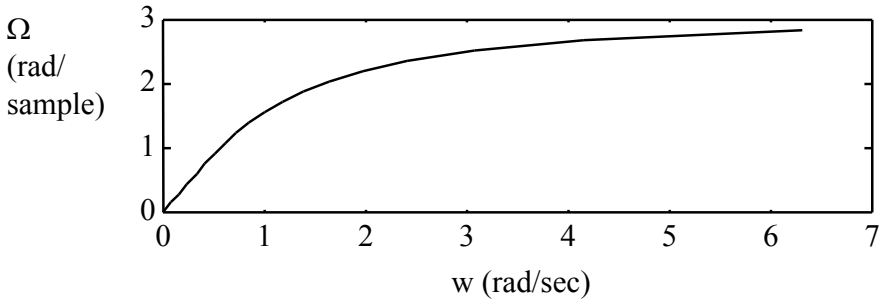


Figure - Frequency warping

If the frequency response of the analogue filter at frequency ω is $H(j\omega)$, then the frequency response of the digital filter at the corresponding frequency $\Omega = 2\arctan(\omega)$ is also $H(j\omega)$. Hence -3dB frequencies become -3dB frequencies, minimax responses remain minimax, etc. – see derivation shortly.

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

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

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:

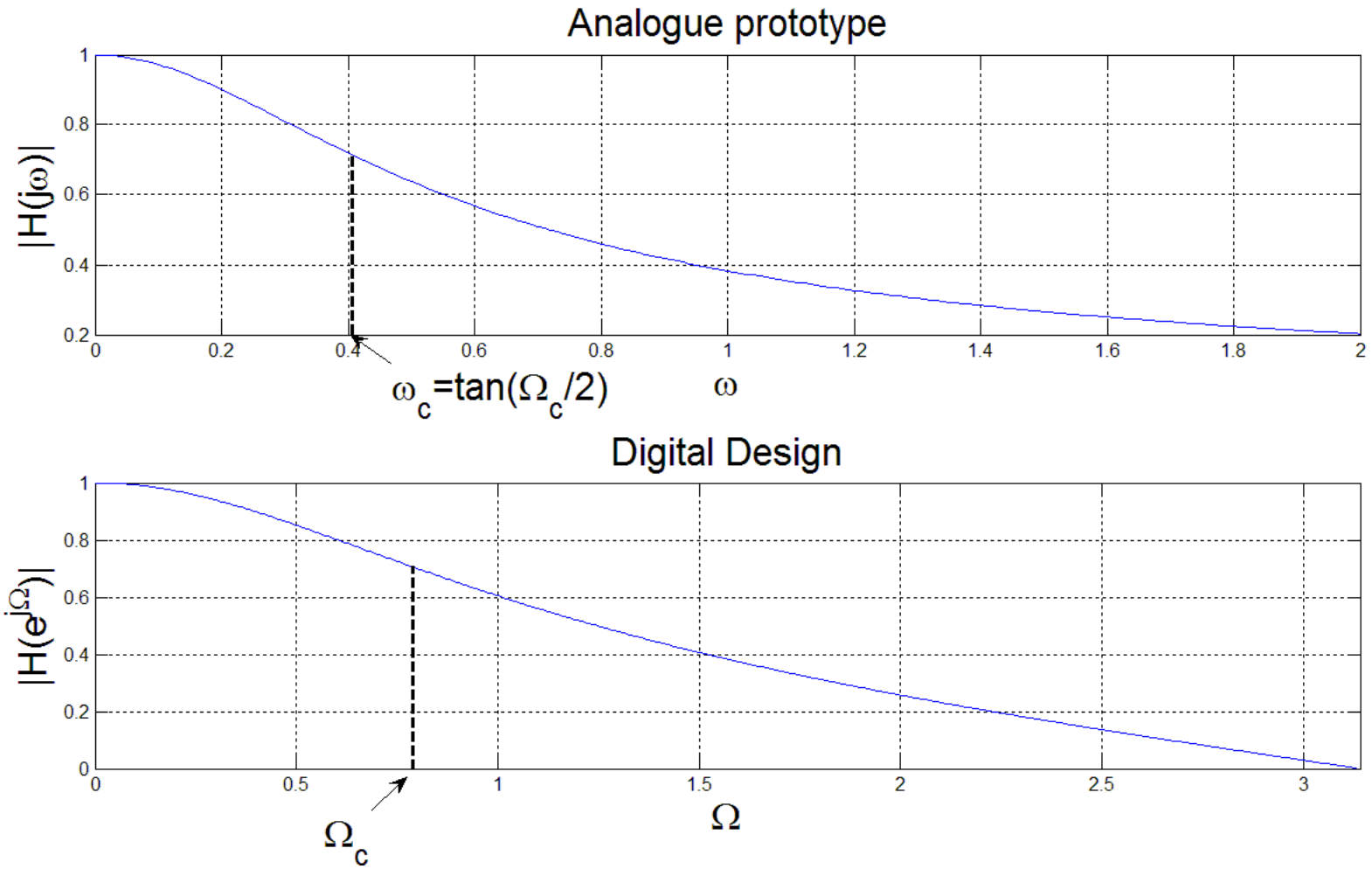
$$\begin{aligned}\hat{H}(z) &= H\left(\psi(z)\right) \\ &= \frac{1}{1+\psi(z)/\omega_c} \\ &= \frac{1}{1+\frac{1-z^{-1}}{1+z^{-1}}/\omega_c} \\ &= \frac{\omega_c(1+z^{-1})}{\omega_c(1+z^{-1})+(1-z^{-1})} \\ &= \frac{\omega_c(1+z^{-1})}{(\omega_c+1)+(\omega_c-1)z^{-1}} \\ &= \frac{0.2929(1+z^{-1})}{1-0.4142z^{-1}}\end{aligned}$$

Normalise to unity for recursive implementation

Keep 0.2929 factorised to save one multiply

i.e. as a direct form implementation:

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



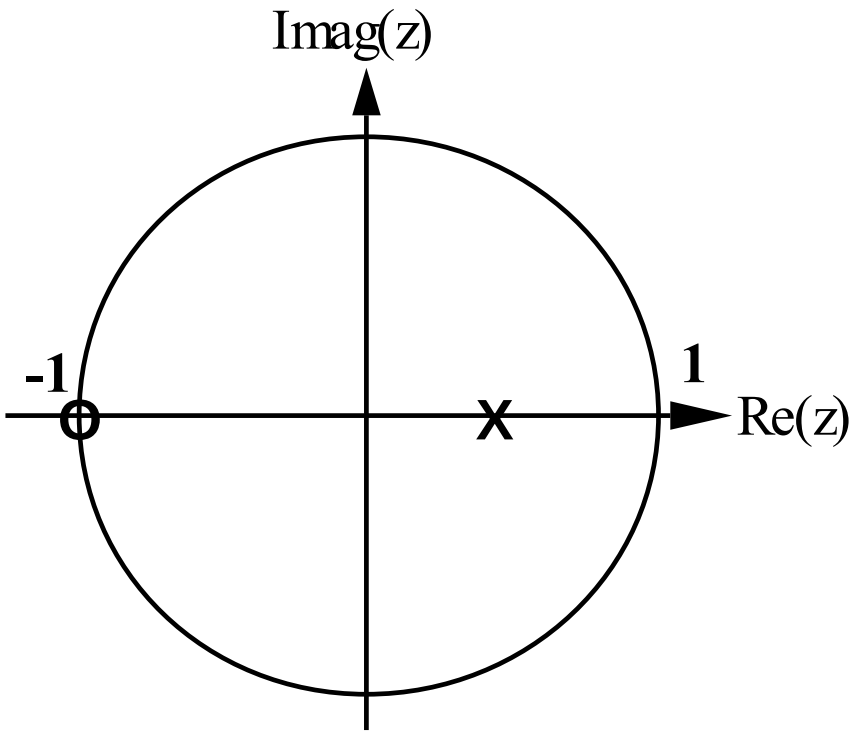
Note that the digital filter response at zero frequency equals 1, as for the analogue filter, and the digital filter response at $\Omega = \pi$ equals 0, as for the analogue filter at $\omega = \infty$. The –3dB frequency is $\Omega = \pi/4$, as intended.

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.

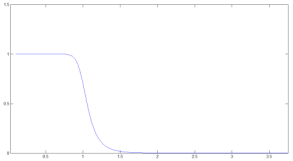
The example above could be computed, for example, by typing

```
[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

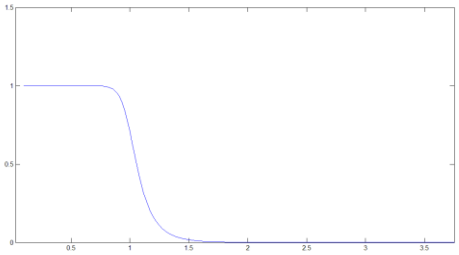


Analogue designs exist for all the standard filter types (lowpass, highpass, bandpass, bandstop). The common approach is to define a standard lowpass filter, and to use standard analogue-analogue transformations from lowpass to the other types, prior to performing the bilinear transform.

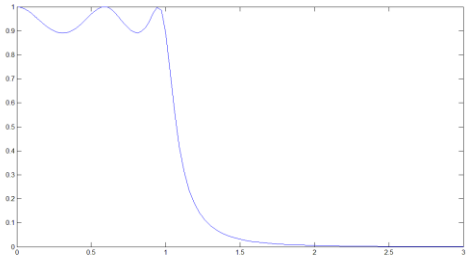
It is also possible to transform from lowpass to other filter types directly in the digital domain, but we do not study these transformations here.

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

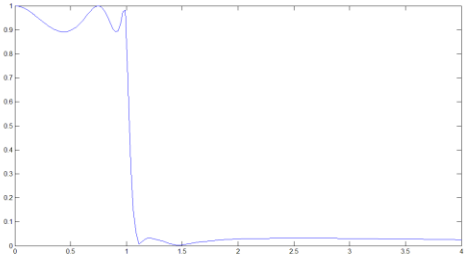
1. Butterworth – maximally flat frequency response near $\omega=0$



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



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



Butterworth (maximally flat)

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

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

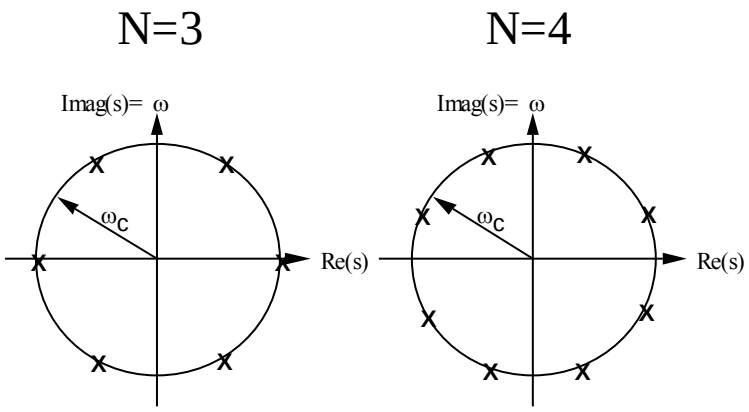
This has unit gain at zero frequency (s = j0), and a gain of -3dB (= $\sqrt{0.5}$) at s = jω_c.

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, say $\{\lambda_i,\ i = 1, ..., P\}$, so that

$$H(s) = \prod_{i=1}^{P} \frac{1}{s - \lambda_i}$$

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 $0.0 < \texttt{Wn} < 1.0$, 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

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

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

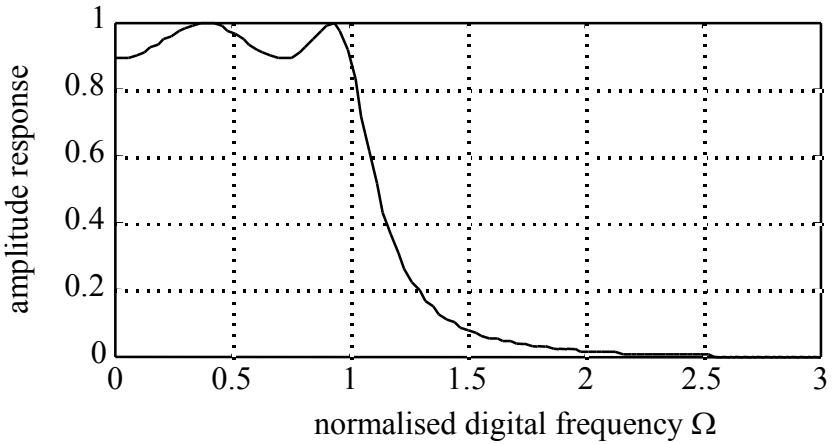


Figure - Type I fourth order Chebyshev LPF frequency response

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

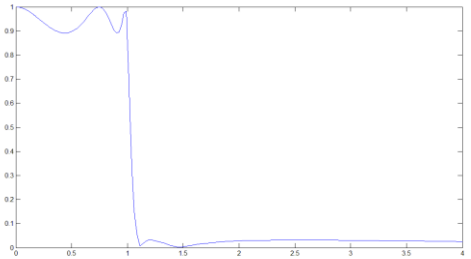
Chebyshev filters are characterised by the frequency response:

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

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

Now analyse e.g. the low-pass to band-pass case, 3.
We substitute for s with:

$$s = \frac{s'^2 + \omega_l \omega_u}{s'(\omega_u - \omega_l)}$$

Now with $s = j\omega$ and $s' = j\omega'$:

$$\omega = \frac{\omega'^2 - \omega_l \omega_u}{\omega'(\omega_u - \omega_l)}$$

and solving for the transformed frequency:

$$\omega' = \frac{\omega(\omega_u - \omega_l) \pm \sqrt{\omega^2(\omega_u - \omega_l)^2 + 4\omega_u \omega_l)}}{2}$$

In particular,
$\omega = 0 \rightarrow \omega' = \sqrt{(}\omega_u \omega_l)$,
$\omega = 1 \rightarrow \omega' = \omega_u$ or ω_l
and $\omega = \infty \rightarrow \omega' = 0$ and ∞, all as required.

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.

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**.

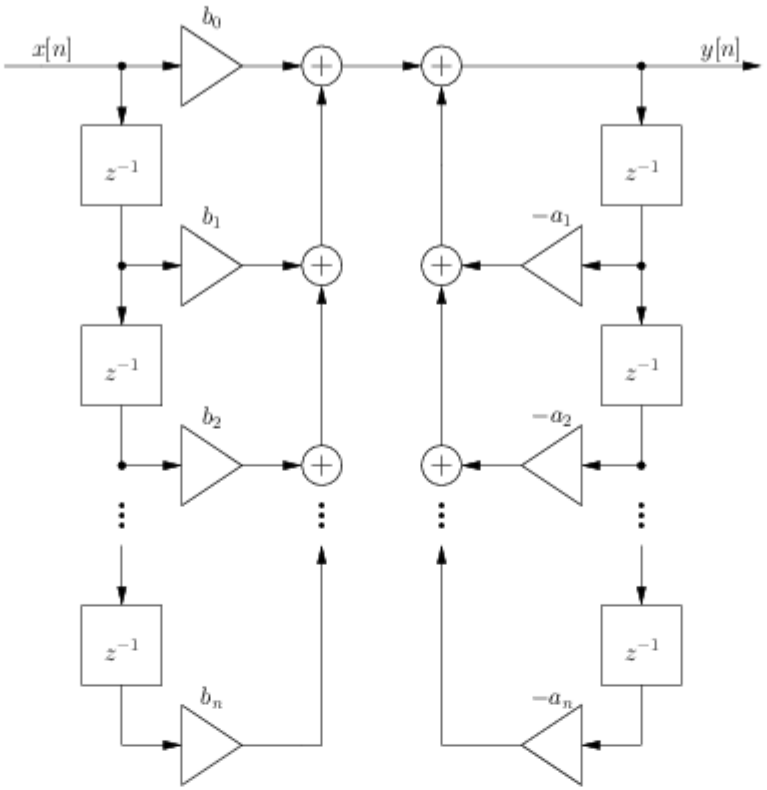
The use of **fixed-point** arithmetic takes much less area than **floating-point** (so is cheaper) or can be made to go faster. The **area** of a fixed-point parallel multiplier is proportional to the product of the coefficient and data wordlengths, making **wordlength reduction** advantageous.

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 (i.e. multiply/divide by powers of 2)

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 – cascade and parallel

Canonic form IIR sections

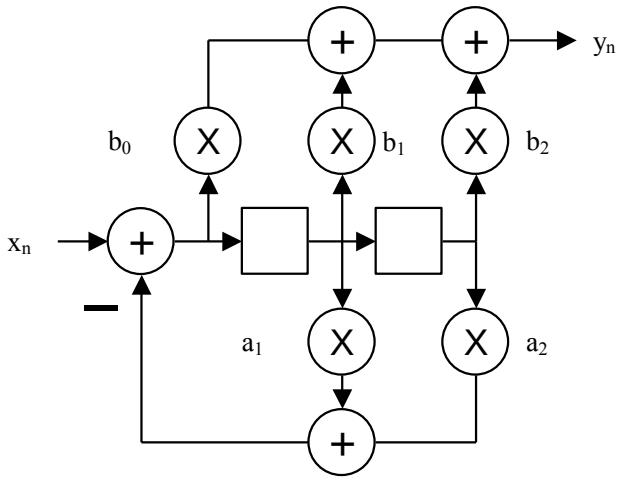
The earlier Figure showed an implementation with separate FIR and IIR stages, called Direct Form I.

We can minimise the number of delay stores by putting the feedback stage first and then using the same delay stores for both parts, since:

$$H(z) = \frac{B(z)}{A(z)} = B(z) \times \frac{1}{A(z)} = \frac{1}{A(z)} \times B(z)$$

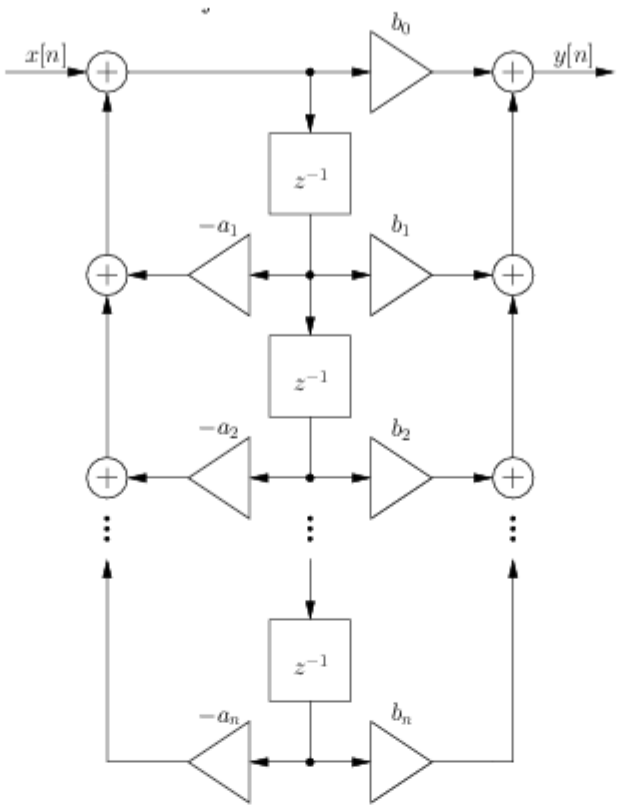
. This is called the *canonic form* ('canonic' means minimum), or Direct Form II.

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]

And for general filter orders:

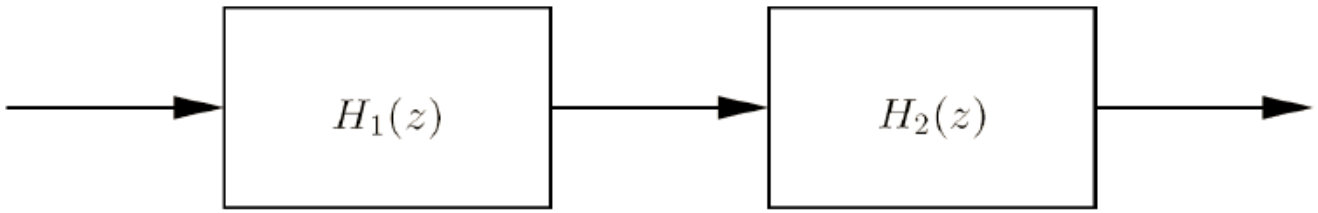


Cascade form IIR filters

- The idea: rewrite the transfer function as a product of filters

$$H(z) = H_1(z) \times \cdots \times H_K(z)$$

- Cascade Structure - the output of one filter is the input to another one, for $K = 2$



- If we ignore finite precision effects

  ⇒ the order of filters in a cascade can be changed without altering the transfer function

Cascades typically use first and second order (biquadratic) 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 $\mathbf{b_0 = 1,\ b_1 = -2r\cos(\Omega),\ b_2 = r^2}$.

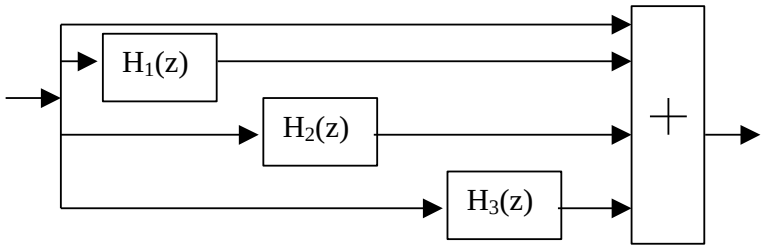
(In principle, b_0, b_1 & b_2 could all be multiplied by a common scale factor, but it is usually advantageous to set $b_0 = 1$ throughout, to avoid unnecessary multiplications, and use a single overall gain factor.)

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

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

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, 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.

When using fixed-point arithmetic many other considerations come into play; some of the most important are:

- Overflow – overflow of fixed point registers can easily occur. Scalings can be applied to the filter coefficients to make this unlikely/ impossible.
- Quantisation noise – this becomes a more significant effect in fixed point implementations.
- Nonlinear effects – in extreme cases with IIR filters limit cycles can be present when there is no change input; similarly dead bands can occur where the filter does not respond to certain low level inputs
- Generally much harder to do the design – software exists to assist, but nowadays most (but not all!) implementations are in floating point arithmetic.

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')

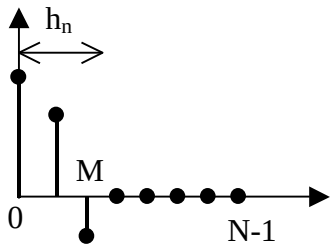
Cyclic convolution

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:

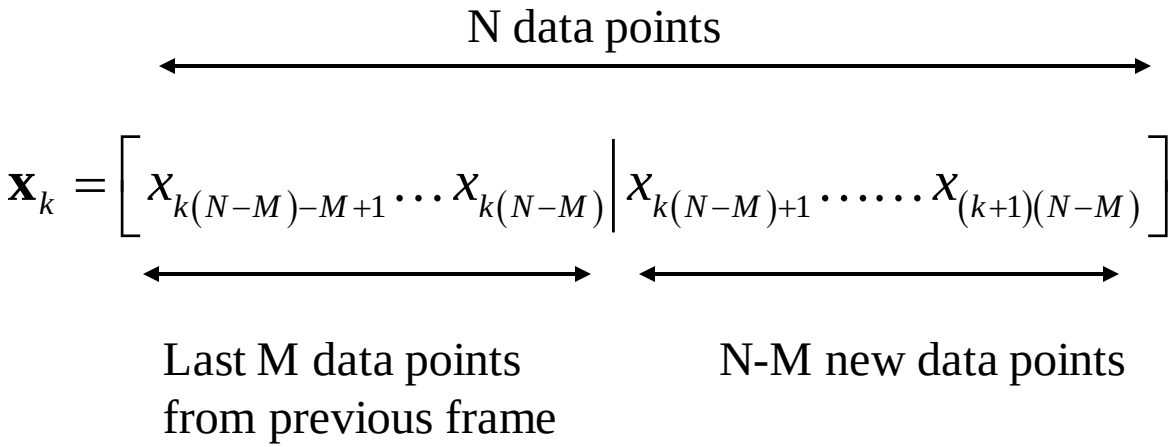
STEP 1

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 kth `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.