Finding the Spectrum of a Signal

- The Fourier transform of a signal is a function of frequency and is called the spectrum of the signal.

$$W(f) = \int_{-\infty}^{\infty} w(t) e^{-j2\pi ft} dt \quad \text{(Fourier transform)}$$

$$w(t) = \int_{-\infty}^{\infty} W(f) e^{j2\pi ft} df \quad \text{(Inverse Fourier transform)}$$

- Fourier transform of periodic impulse sampled signal

$$F\{w(t)\} = W(f)$$

$$w_s(t) = w(t) \sum_{n=-\infty}^{\infty} \delta(t - kT_s)$$

$$F\{w_s(t)\} = \frac{1}{T_s} \sum_{n=-\infty}^{\infty} W\left(f - (n/T_s)\right)$$

- Quantization adds more harmonics

- We can simulate continuous signals in digital domain if the simulated frequences are less than half the sampling frequency.

- Next we can compute the Fourier transform at discrete values of frequency which is done by means of the Discrete Fourier Transform (DFT).

- The fast algorithm to calculate DFT is called Fast Fourier Transform (FFT). FFT is used in the book to evaluate signal spectrum.

- Normally FFT evaluates the spectrum from 0 to sampling frequency f_S. To have a symmetric figure and more natural figure from $-f_S$ to f_S Matlab is using “fftshift” function.

- Linear filters are implemented through the convolution in time domain. This corresponds to multiplication in frequency domain. Choosing properly linear filters one can adjust the gains at different frequencies and suppress the signal at some frequencies while enhancing it at other frequencies.

- Thus linear filters will serve as spectrum “shaping” mechanism in signal processing applications.

Spectra plotting function

```
% plotspec(x,Ts) plots the spectrum of the signal x
% Ts = time (in seconds) between adjacent samples in x

function plotspec(x,Ts)

N=length(x);                        % length of the signal x
t=Ts*(1:N);                         % define a time vector
ssf=(-N/2:N/2-1)/(Ts*N);            % frequency vector
fx=fft(x(1:N));                     % do DFT/FFT
fxs=fftshift(fx);                   % shift it for plotting

subplot(2,1,1), plot(t,x)          % plot the waveform


xlabel('seconds'); ylabel('amplitude') % label the axes

% plot magnitude spectrum
subplot(2,1,2), plot(ssf,abs(fxs))
xlabel('frequency'); ylabel('magnitude') % label the axes
```

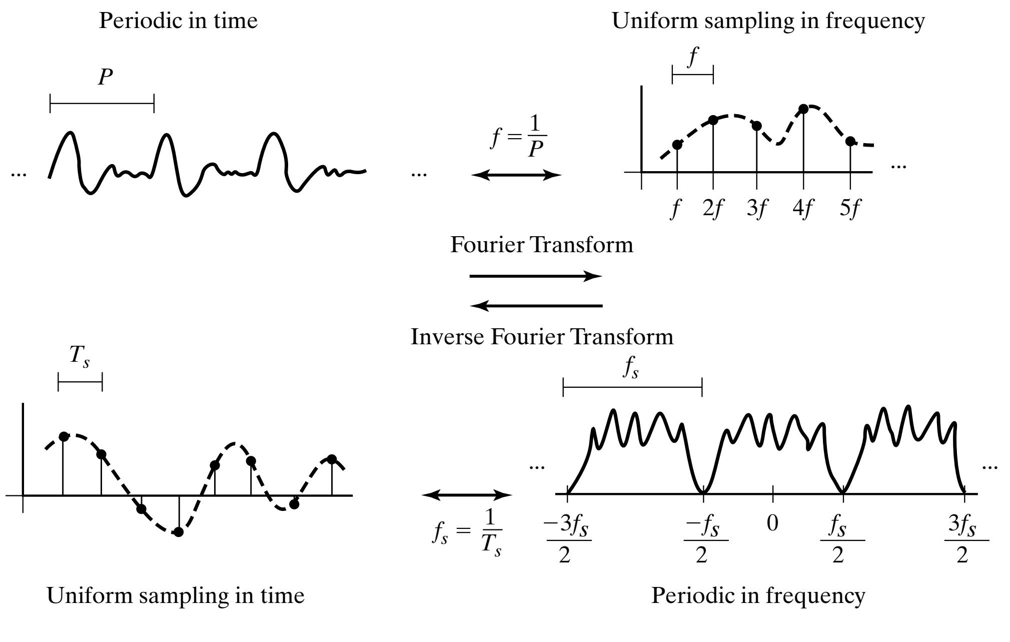
DFT Tidbits (cont'd)

The key factors in a DFT based frequency analysis are:

- The sampling interval T_s is the time resolution, the shortest time over which any event can be observed.
- The sampling rate is $f_s = \frac{1}{T_s}$. As the sample rate increases, the time resolution decreases.
- The total time is $T = NT_s$ where N is the number of samples in the analysis.
- The frequency resolution is $\frac{1}{T} = \frac{1}{NT_s} = \frac{f_s}{N}$. Sinusoids closer together (in frequency) than this value are indistinguishable.
- As the time resolution increases, the frequency resolution decreases, and vice versa.

Samplers (cont'd)

- *Frequency Domain View*:
  - Sampling in time replicates analog spectrum every f_s Hz.
  - When plotting discrete-time signal spectrum, need only plot values over $\frac{-f_s}{2}$ to $\frac{f_s}{2}$ to be able to visualize entire spectrum.



Linear Filters

- *Symbol*:

$$\begin{tikzcd} {} \arrow[r, "u(t)"] & \boxed{h(t)} \arrow[r, "y(t)"] & {} \end{tikzcd} \quad \text{or} \quad \begin{tikzcd} {} \arrow[r, "u"] & \boxed{H(f)} \arrow[r, "y"] & {} \end{tikzcd}$$

- *Time Domain View*: For a linear, time-invariant, continuous-time filter convolution of input u with impulse response h produces output y

$$y(t) = h(t) * u(t) = \int_{\lambda=-\infty}^{\infty} h(t-\lambda)u(t)d\lambda$$

For a causal, finite-impulse-response (FIR), linear, time-invariant, discrete-time system convolution reduces to, e.g.

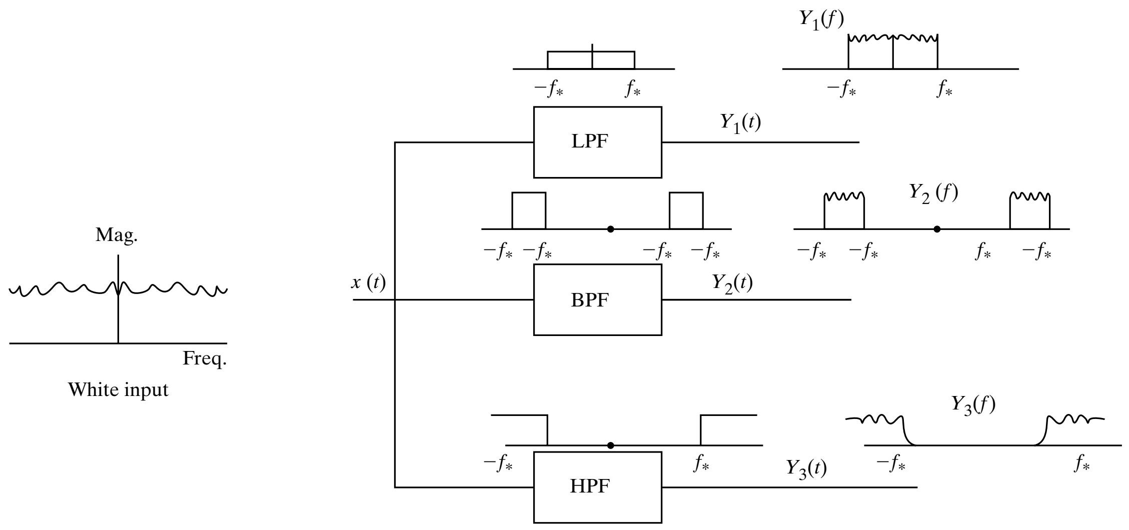
$y[k] = \sum_{i=0}^{2} h[i]u[k-i]$

$$= h[0]u[k] + h[1]u[k-1] + h[2]u[k-2]$$

where $h[k]$ is the (im)pulse response and u and y are the input and output.

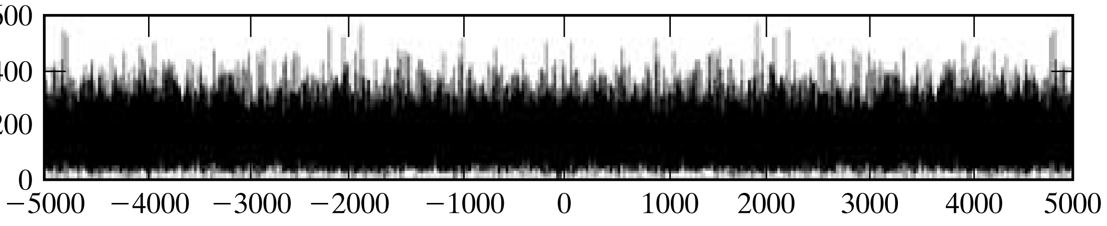
Linear Filters (cont'd)

- *Frequency Domain View*: Convolution in time domain is multiplication in frequency domain (see (E.37)), allowing for frequency content weighting and removal but not creation.
- *Example*: A 'white' signal containing all frequencies is passed through a Low Pass Filter (LPF) leaving only the low frequencies, a Band Pass Filter (BPF) leaving only the middle frequencies and a High Pass Filter (HPF) leaving only the high frequencies.

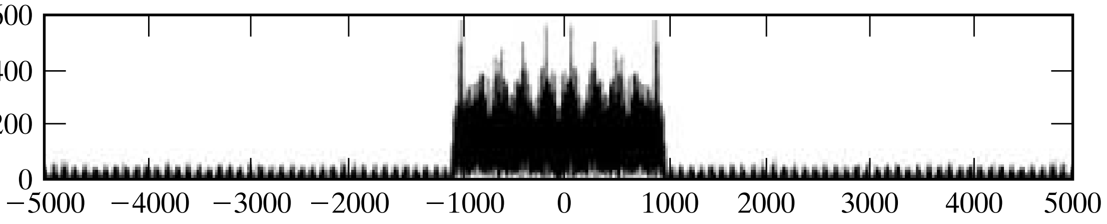


Linear Filters (cont'd)

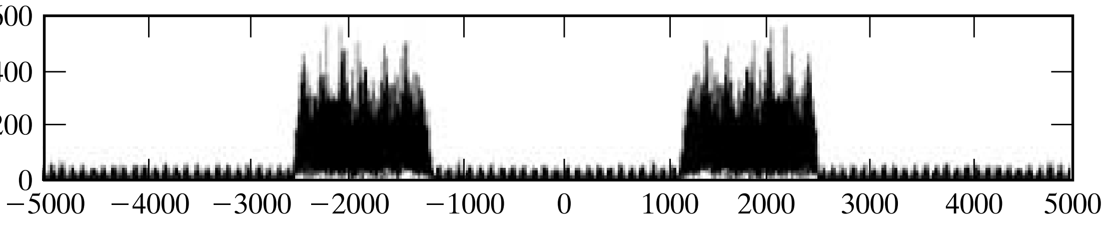
Simulated test from `filternoise` (using `plotspec`)



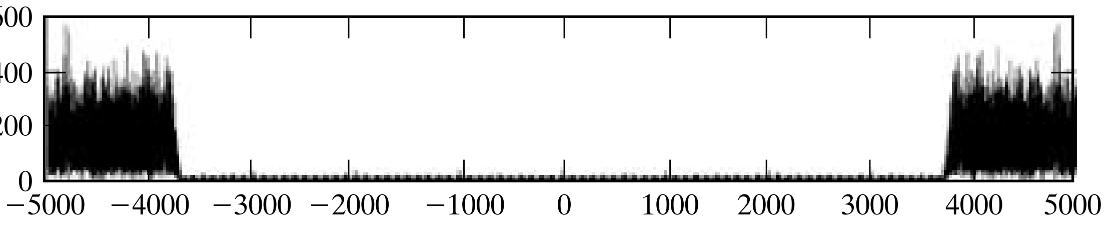
Magnitude spectrum at input



Magnitude spectrum at output of lowpass filter



Magnitude spectrum at output of bandpass filter



Magnitude spectrum at output of highpass filter

Experiment 1. Random noise filtering and spectra observation

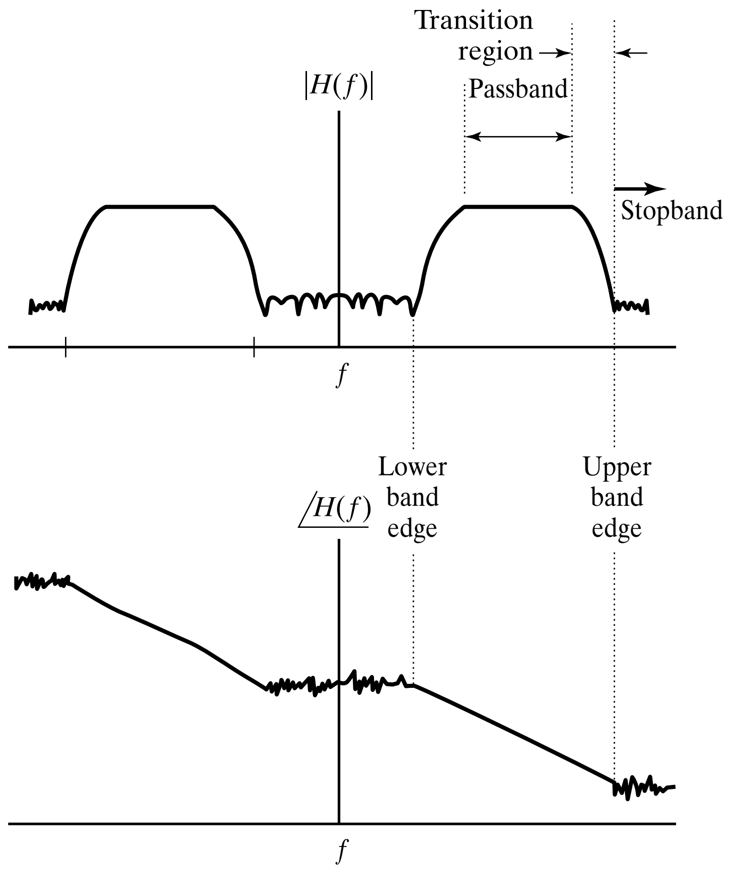
```
% filternoise.m filter a noisy signal three ways
time=3;                         % length of time
Ts=1/10000;                     % time interval between
samples
x=randn(1,time/Ts);             % generate noise signal

figure(1),plotspec(x,Ts)        % draw spectrum of input

% specify the LP filter
b=remez(100,[0 0.2 0.21 1],[1 1 0 0]);
ylp=filter(b,1,x);              % do the filtering
figure(2),plotspec(ylp,Ts)      % plot the output spectrum

% BP filter
b=remez(100,[0 0.24 0.26 0.5 0.51 1],[0 0 1 1 0 0]);
ybp=filter(b,1,x);              % do the filtering
figure(3),plotspec(ybp,Ts)      % plot the output spectrum

% specify the HP filter
b=remez(100,[0 0.74 0.76 1],[0 0 1 1]);
yhp=filter(b,1,x);              % do the filtering
figure(4),plotspec(yhp,Ts)      % plot the output spectrum

%Here's how the figure filternoise.eps was actually drawn

N=length(x);                     % length of the signal x
t=Ts*(1:N);                      % define a time vector
ssf=(-N/2:N/2-1)/(Ts*N);         % frequency vector
fx=fftshift(fft(x(1:N)));
figure(5), subplot(4,1,1), plot(ssf,abs(fx))
xlabel('magnitude spectrum at input')
fyl=fftshift(fft(ylp(1:N)));
subplot(4,1,2), plot(ssf,abs(fyl))
xlabel('magnitude spectrum at output of low pass filter')
fybp=fftshift(fft(ybp(1:N)));
subplot(4,1,3), plot(ssf,abs(fybp))
xlabel('magnitude spectrum at output of band pass filter')
fyhp=fftshift(fft(yhp(1:N)));
subplot(4,1,4), plot(ssf,abs(fyhp))
xlabel('magnitude spectrum at output of high pass filter')
```

Filter Design Tidbits

Types:

- lowpass filter
- high pass filter
- bandpass filter
- bandstop (notch) filter

Bandpass filter spectra specification:



Filter Design Tidbits (cont'd)

From `help remez`: (remez(...) function is changed to firpm(...) in recent Matlab versions)

REMEZ performs Parks-McClellan optimal equiripple FIR filter design. i.e. a linear phase (real, symmetric coefficients) FIR filter which has the best approximation to the desired frequency response in the minimax sense.

```
b = remez(fl,fbe,damps)
```

- `b` is the output vector containing the impulse response of the designed filter.
- `fl` is (one less than) the number of terms in `b`.
  - $\odot$ `fl` $\uparrow \Rightarrow$ fit to design specs improves
  - $\odot$ `fl` $\uparrow \Rightarrow$ computational costs increase
  - $\odot$ `fl` $\uparrow \Rightarrow$ throughput delay increases

Filter Design Tidbits (cont'd)

- `fbe` is a vector of frequency band edge values as a fraction of the prevailing Nyquist frequency. For a basic bandpass filter:
  - bottom of stopband (presumably zero)
  - top edge of lower stopband (which is also the lower edge of the lower transition band)
  - lower edge of passband
  - upper edge of passband
  - lower edge of upper stopband
  - upper edge of upper stopband (generally the last value will be 1).
- `damps` is the vector of desired amplitudes of the frequency response at each band edge.

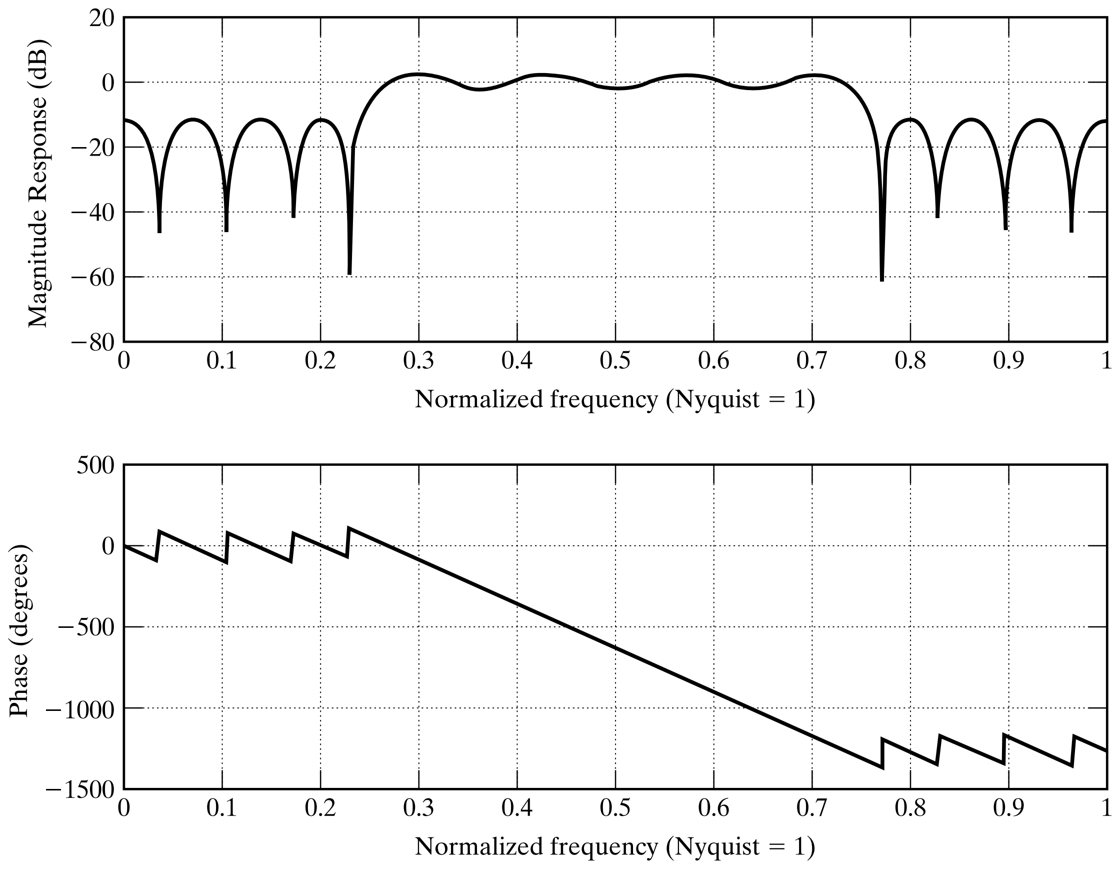
Experiment 2. Filter design with specs (this page only)

Filter Design Tidbits (cont'd)

From `bandex` with

- `fbe=[0 0.24 0.26 0.74 0.76 1]`
- `damps=[0 0 1 1 0 0]`
- `fl=30`

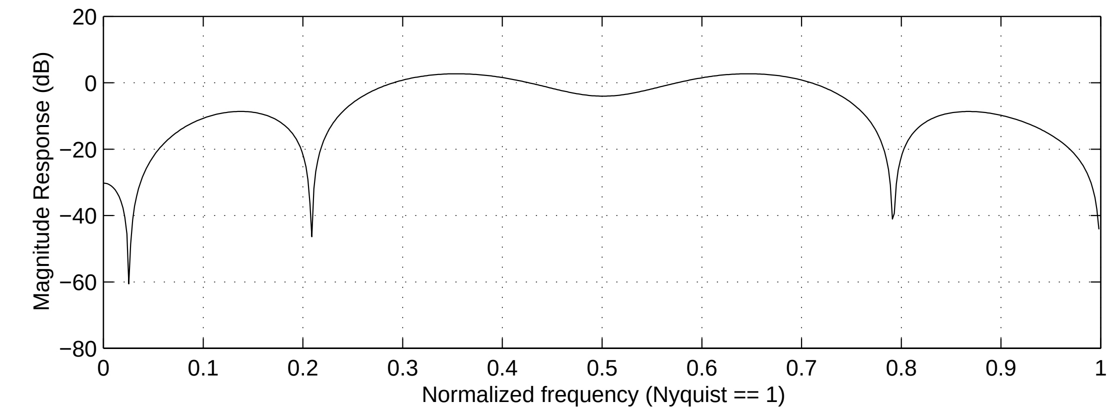
`b=remez(fl,fbe,damps); freqz(b)` produces

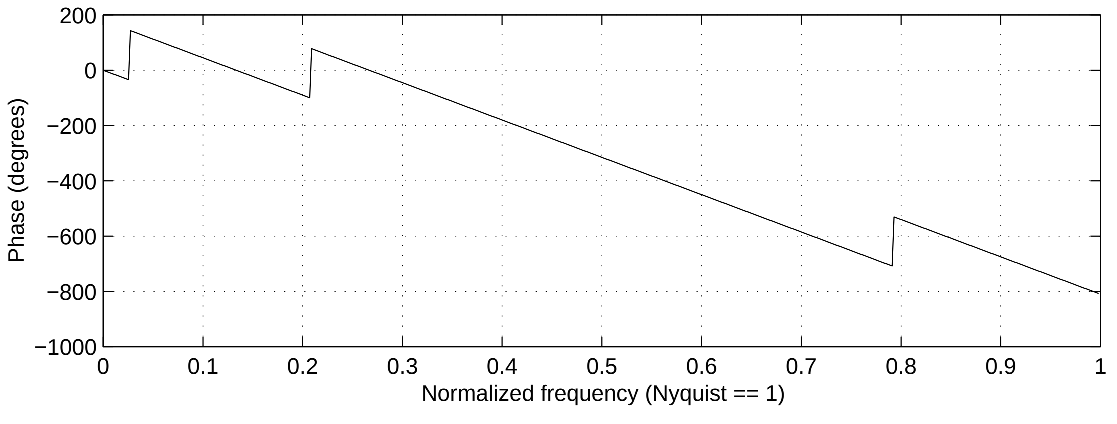


Filter Design Tidbits (cont'd)

To demonstrate criteria fit impact of filter length, repeat preceding example with `fl` halved and doubled.

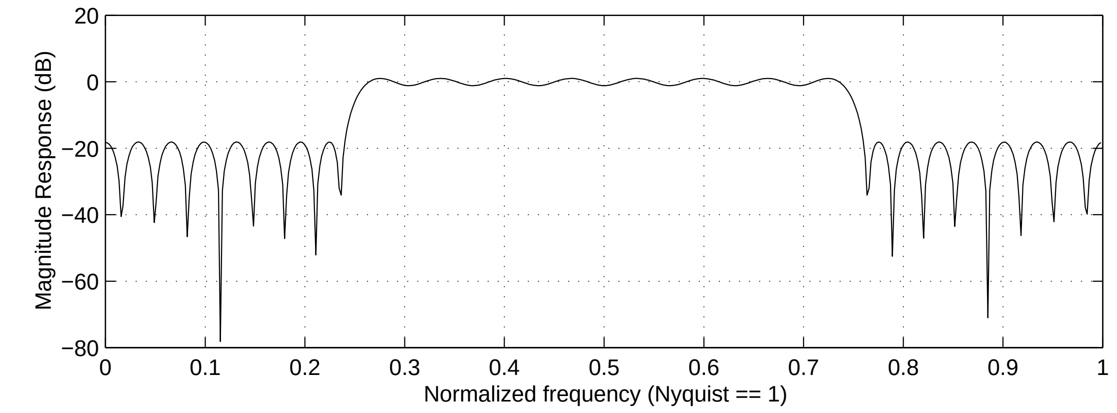
- `fl=15`

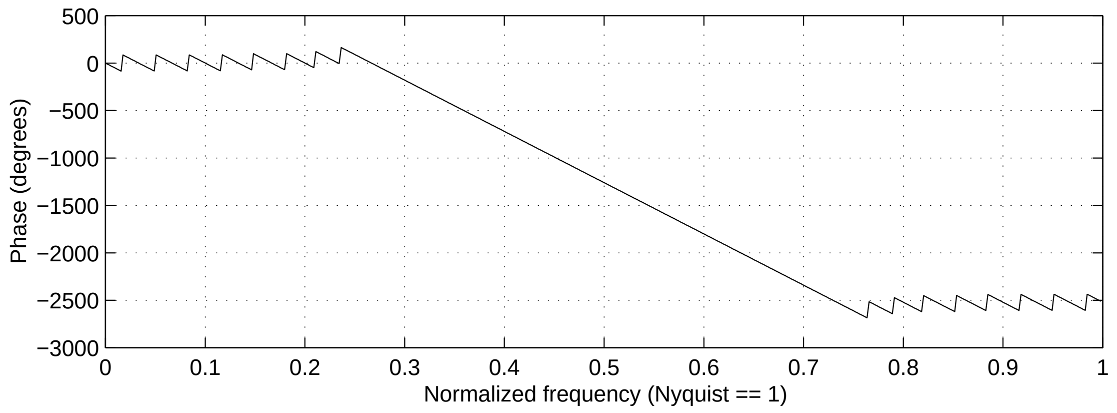




Filter Design Tidbits (cont'd)

- `fl=60`



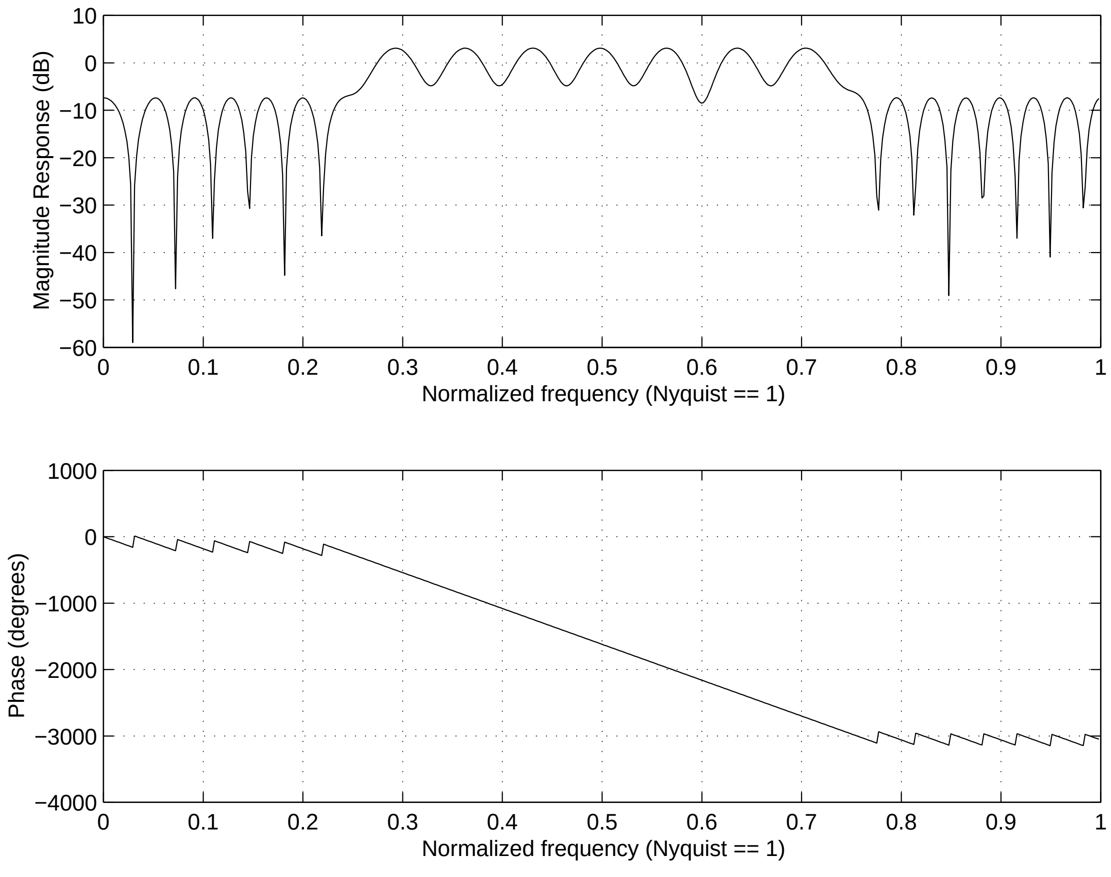


Note improved fit to design specifications with increase in filter length (`fl`).

Filter Design Tidbits (cont'd)

Adding an in-band notch with

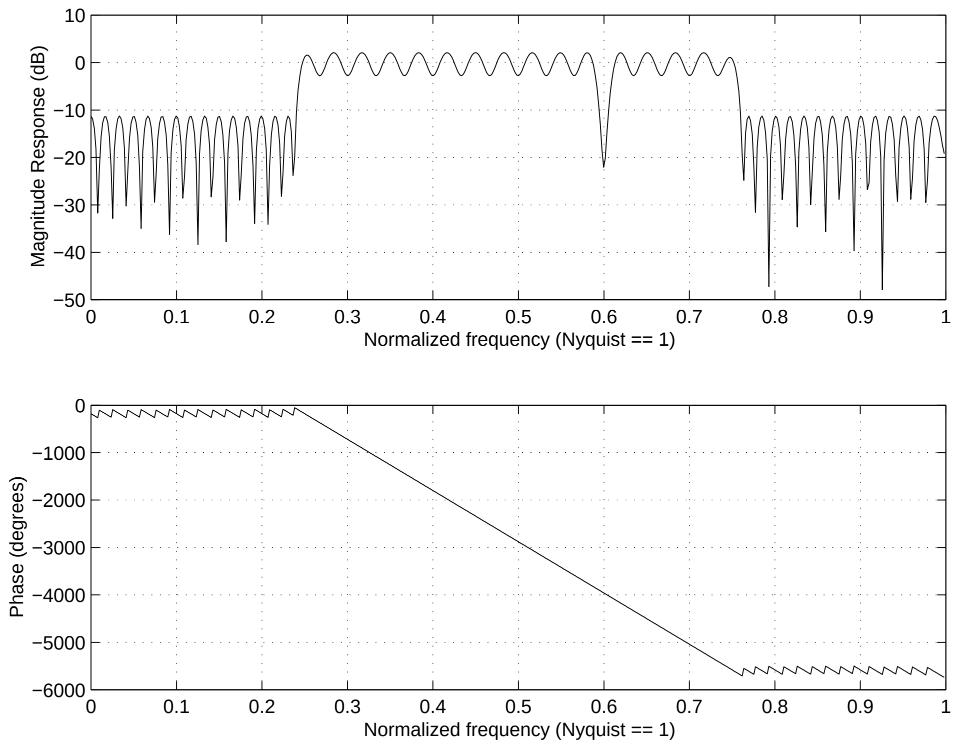
- `fbe=[0 0.24 0.26 0.59 0.595 0.605 0.61 0.74 0.76 1]`
- `damps=[0 0 1 1 0 0 1 1 0 0]`
- `fl=60`



Desired notch (at normalized frequency = 0.6) is barely perceptible.

Filter Design Tidbits (cont'd)

- `fl=120`



Note improved fit to design specifications with increase in filter length (`fl`). Desired notch (at normalized frequency = 0.6) is quite pronounced.

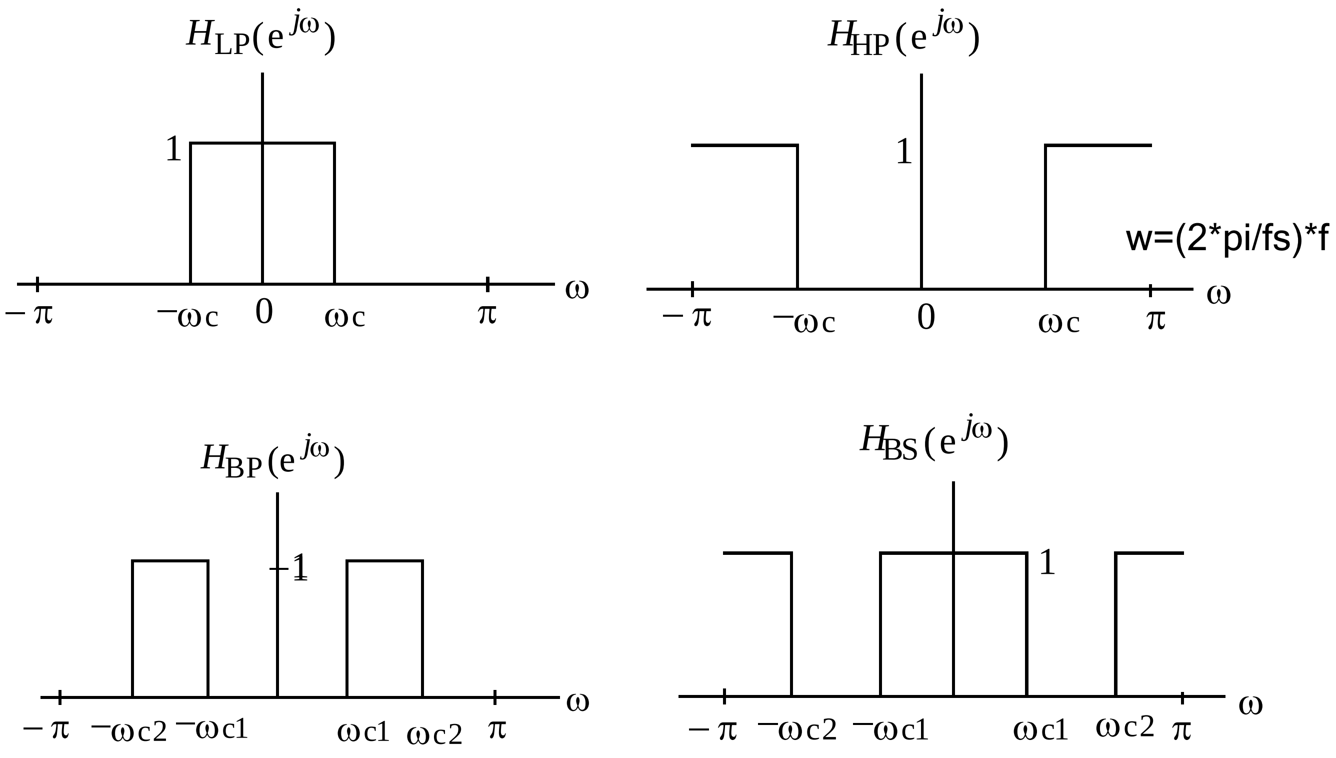
3. Digital Filter Specifications

- Ideal filter is designed to pass signal components of certain frequencies without distortion and should have a frequency response equal to one at these frequencies. It should have a frequency response equal to zero at all other frequencies.

- The range of frequencies where the frequency takes the value of one is called passband

- The range of frequencies where the frequency response takes the value of zero is called the stopband

- There are four basic types of ideal filters (Lowpass, Highpass, Bandpass, Bandstop) with magnitude responses as shown below



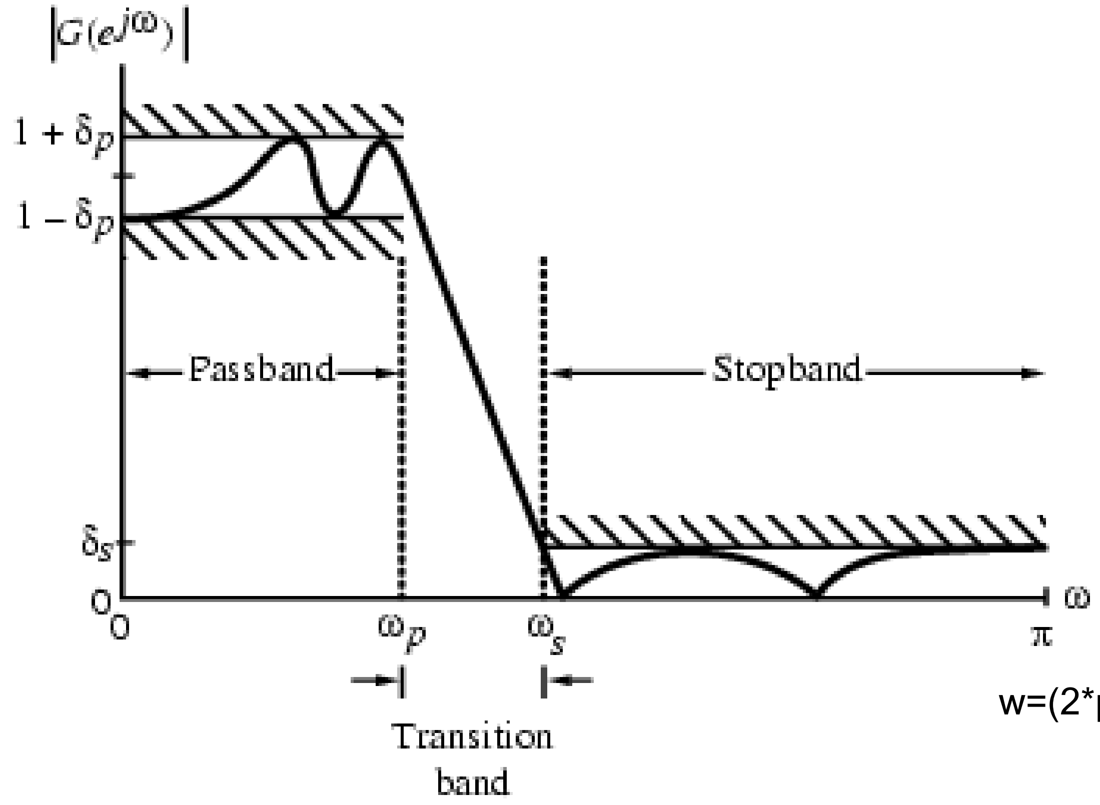
- The frequncies w_c are called cutoff frequencies.

- These filters are unrealizable because their impulse responses infinitely long non-causal

- In practice the magnitude response specifications of a digital filter in the passband and in the stopband are given with some acceptable tolerances

- In addition, a transition band is specified between the passband and stopband

- For example the magnitude response $G(e^{jw})$ of a digital lowpass filter may be given as indicated below



w=(2*pi/fs)*f

- In the **passband** $0 \le w \le w_p$ we require that $H \cong 1$ with a deviation $\pm\delta$

$$1-\delta_p \le |H| \le 1+\delta_p \quad w \le w_p$$

- In the stopband $w_s \le w \le \pi$ we require that $H \cong 0$ with the deviation $\pm\delta_s$

$$|H| \le \delta_s \quad w_s \le w \le \pi$$

Filter specification parameters

- w_p - passband edge frequency
- w_s - stopband edge frequency
- δ_p - peak ripple value in the passband
- δ_s - peak ripple value in the stopband
- Practical specifications are often given in terms of loss function (in dB): $-20\log_{10}|H|$

- Peak passband ripple
  $\alpha_p = -20\log_{10}(1-\delta_p)$

- Peak-to-peak passband ripple
  $\alpha_p = -20\log_{10}(1-2\delta_p)$ or $\alpha_p = -20\log_{10}\left(\frac{1-\delta_p}{1+\delta_p}\right)$

- Minimum stopband attenuation

  $\alpha_s = -20\log_{10}(\delta_s)$

Experiment 3. IIR Filter design using Matlab. Just repeat the experiment and understand parameters

IIR filter design example by Matlab

```
>> help ellip
```

ELLIP Elliptic or Cauer digital and analog filter design. [B,A] = ELLIP(N,Rp,Rs,Wn) designs an Nth order lowpass digital elliptic filter with Rp decibels of peak-to-peak ripple and a minimum stopband attenuation of Rs decibels. ELLIP returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. Use Rp = 0.5 and Rs = 20 as starting points, if you are unsure about choosing them.

```
>> help ellipord
```

ELLIPORD Elliptic filter order selection.
[N, Wn] = ELLIPORD(Wp, Ws, Rp, Rs) returns the order N of the lowest order digital elliptic filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. Wp and Ws are the passband and stopband edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample). For example,

```
    Lowpass:    Wp = .1, Ws = .2
    Highpass:   Wp = .2, Ws = .1
    Bandpass:    Wp = [.2 .7], Ws = [.1 .8]
    Bandstop:   Wp = [.1 .8], Ws = [.2 .7]
```

ELLIPORD also returns Wn, the elliptic natural frequency to use with ELLIP to achieve the specifications.

```
>> [N, Wn]=ellipord(0.1,0.2,0.5,20)

N =3

Wn = 0.1000

>> [B,A]=ellip(3,0.5,20,0.1)

B = 0.0497   -0.0380   -0.0380    0.0497
A = 1.0000   -2.5485    2.2530   -0.6811
```

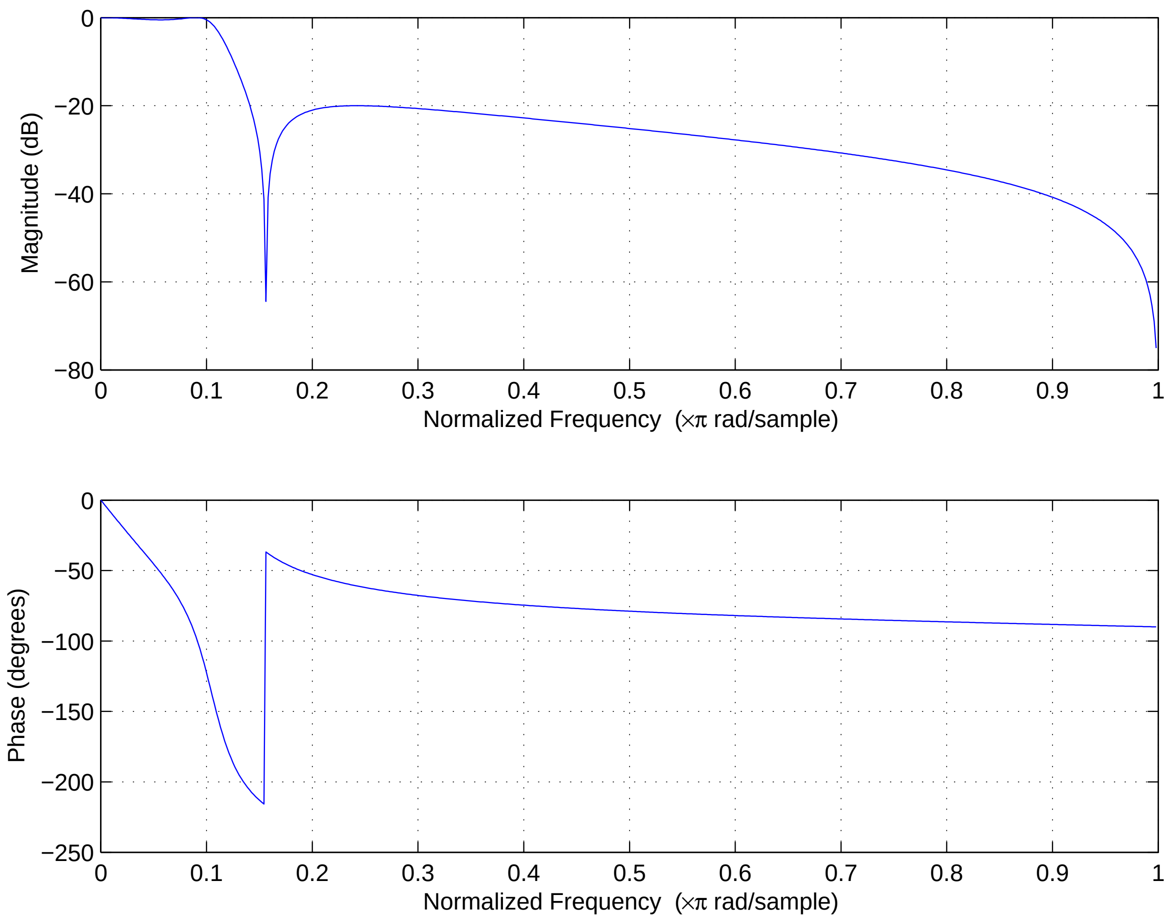


Figure # Frequency response of the designed IIR filter

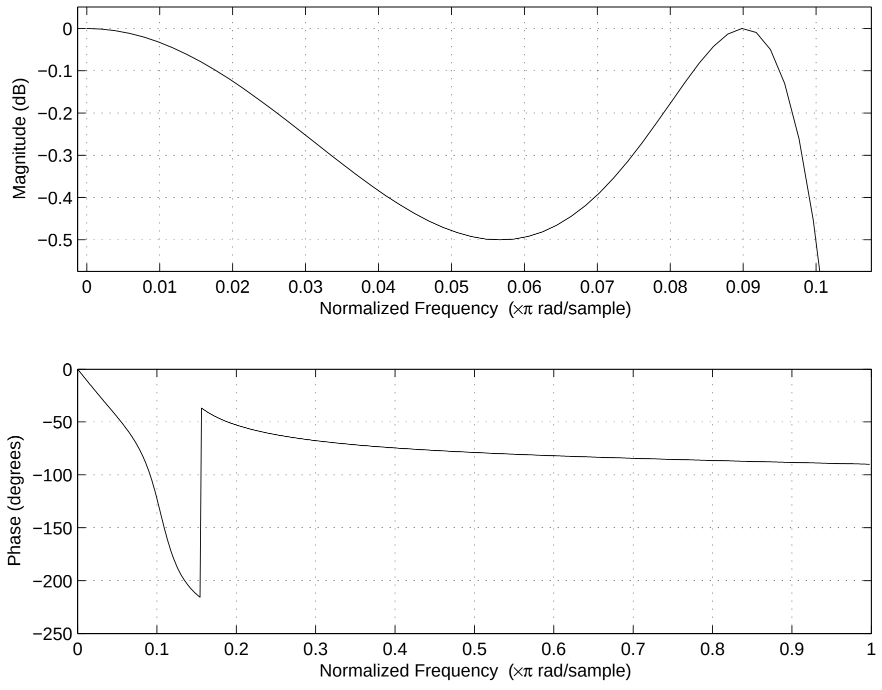


Figure # Frequency response of the designed IIR filter: passband details.

Experiment 4. FIR filter design using Matlab.

FIR Design example by Matlab

```
>> help remez
```

(remez (...) changed to firpm(...))

REMEZ Parks-McClellan optimal equiripple FIR filter design. B=REMEZ(N,F,A) returns a length N+1 linear phase (real, symmetric coefficients) FIR filter which has the best approximation to the desired frequency response described by F and A in the minimax sense. F is a vector of frequency band edges in pairs, in ascending order between 0 and 1. 1 corresponds to the Nyquist frequency or half the sampling frequency. A is a real vector the same size as F which specifies the desired amplitude of the frequency response of the resultant filter B.

```
>> help remezord
```

REMEZORD FIR order estimator (lowpass, highpass, bandpass, multiband) [N,Fo,Ao,W] = REMEZORD(F,A,DEV,Fs) finds the approximate order N, normalized frequency band edges Fo, frequency band magnitudes Ao and weights W to be used by the REMEZ function as follows:

```
        B = REMEZ(N,Fo,Ao,W)
```

The resulting filter will approximately meet the specifications given by the input parameters F, A, and DEV.

$$0.5 = -20\log_{10}(1-2\delta_p)$$

$$\delta_p = \left(1-10^{-0.5/20}\right)/2 = (1-0.944)/2 = 0.028$$

$$20 = -20\log_{10}(\delta_s)$$

$$\delta_s = 0.1$$

```
        [n,fo,mo,w] = remezord( [0.1 0.2], [1 0], [0.028 0.1], 2);
         b = remez(n,fo,mo,w);

>> [n,fo,mo,w] = remezord( [0.1 0.2], [1 0], [0.028 0.1], 2)

n =21
fo = 0  0.1000  0.2000  1.0000
mo = 1 1 0  0
w = 3.57  1.0000
```

```
>> remez(n,fo,mo,w)

ans = 0.0130   -0.0560   -0.0280   -0.0167   -0.0012    0.0224
0.0526    0.0858    0.1172    0.1415    0.1549 0.1549    0.1415
0.1172    0.0858    0.0526    0.0224   -0.0012   -0.0167   -0.0280
-0.0560    0.0130
```

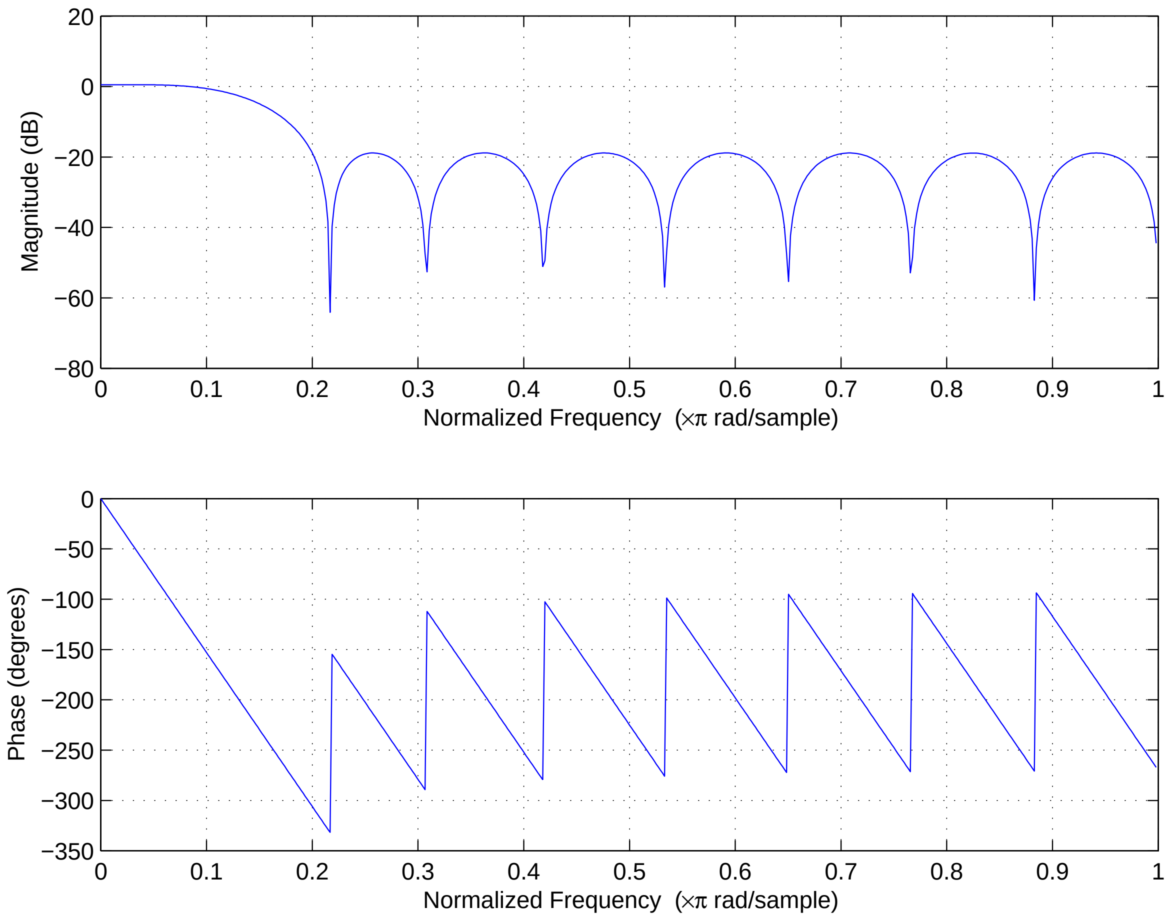


Figure # Frequency response of the designed FIR filter

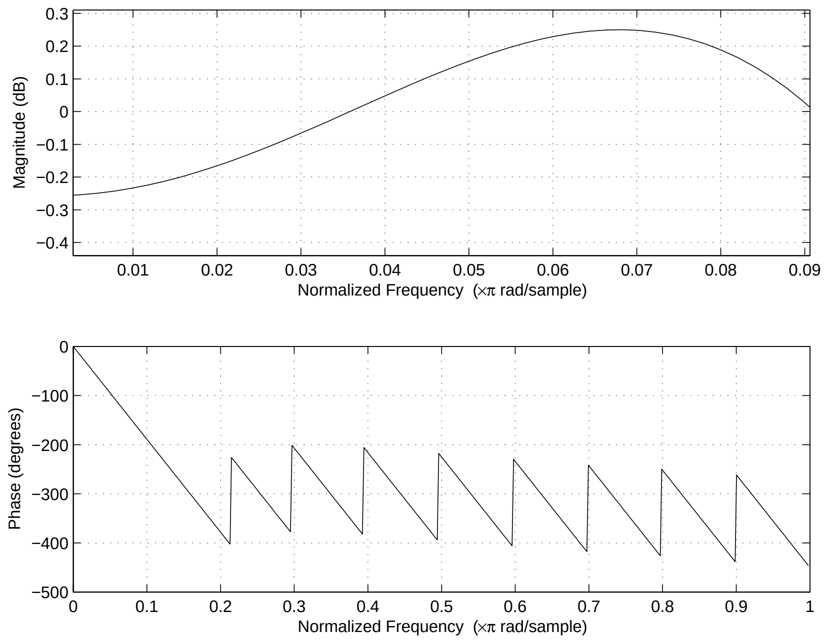


Figure # Frequency response of the designed FIR filter: passband details

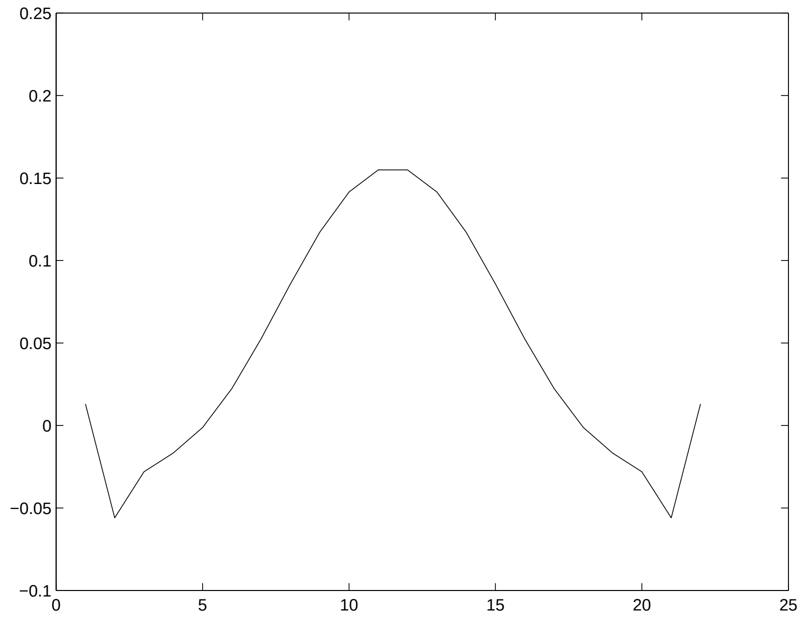


Figure # Filter's shape

Filter implementation

Convolution and Filtering

The mathematical foundation of filtering is convolution. The MATLAB `conv` function performs standard one-dimensional convolution, convolving one vector with another:

```
conv([1 1 1],[1 1 1])
ans =
     1     2     3     2     1

```

A digital filter's output *y*(*k*) is related to its input *x*(*k*) by convolution with its impulse response *h*(*k*).

$$y(k) = h(k) * x(k) = \sum_{l=-\infty}^{\infty} h(k-l)x(l)$$

If a digital filter's impulse response *h*(*k*) is finite length, and the input *x*(*k*) is also finite length, you can implement the filter using `conv`. Store *x*(*k*) in a vector `x`, *h*(*k*) in a vector `h`, and convolve the two:

```
x = randn(5,1);    % A random vector of length 5
h = [1 1 1 1]/4;   % Length 4 averaging filter
y = conv(h,x);
```

Filtering with the “filter” Function and Difference Equation

- It is a difference equation

$$y(k) + a_2 y(k-1) + \cdots + a_{m+1} y(k-m) = b_1 x(k) + b_2 x(k-1) + \cdots + b_{n+1} x(k-m)$$

In terms of current and past inputs, and past outputs, *y*(*n*) is

$$y(k) = b_1 x(k) + b_2 x(k-1) + \cdots + b_{n+1} x(k-n) - a_2 y(k-1) - \cdots - a_{m+1} y(k-n)$$

where the vectors `b` and `a` represent the coefficients of a filter in transfer function form. To apply this filter to your data, use

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