BINGHAMTON UNIVERSITY
State University of New York

EECE 301
Signals & Systems
Prof. Mark Fowler

Note Set #29

- D-T Systems: FIR Filters

FIR Filters (Non-Recursive Filters)

FIR (Non-Recursive) filters are certainly the most widely used DT filters.

There are several reasons for this:

- Simple & effective design methods exist
- Designs are always stable (since only poles at origin)
- Linear phase designs can be easily achieved

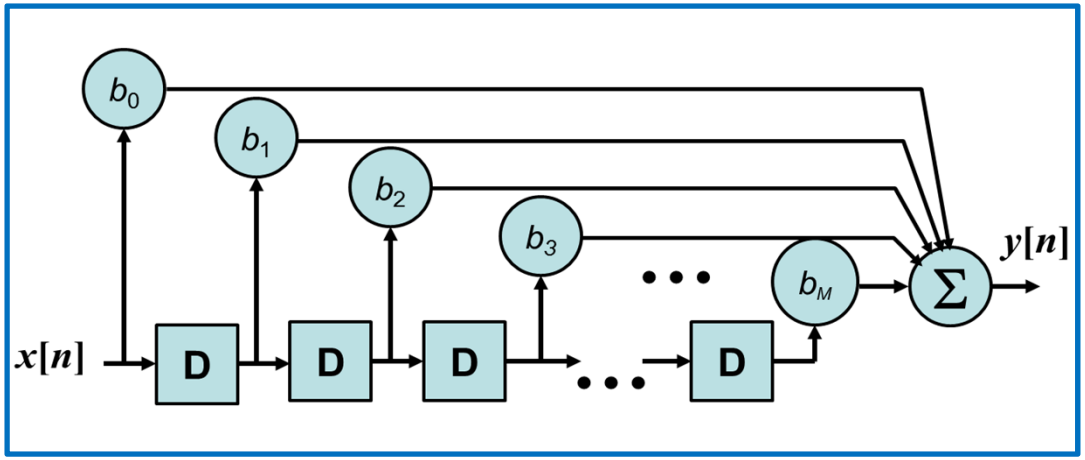
We'll look at 3 methods

So… it's easy to meet part of the requirements for ideal filters!

However, the only real downside of FIR filters is that they can be computationally complex…

High-quality frequency response requires "long" filters ($M > 100$)

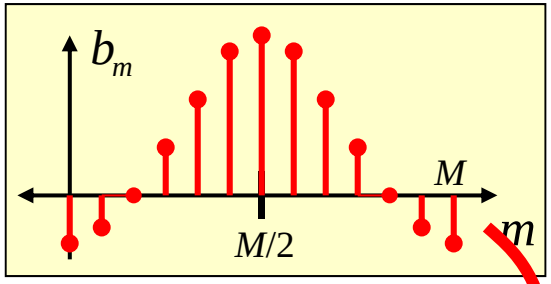
- That means your "computer" will have to a lot of computation
  - M+1 Multiplies & M Additions



So… The goal in FIR design is to get the Freq. Resp. specs you want with the smallest *M*!!!

FIR LPF Design – Windowed sinc Method (fir1)

We looked at a truncated sinc as a way to design the b_m:



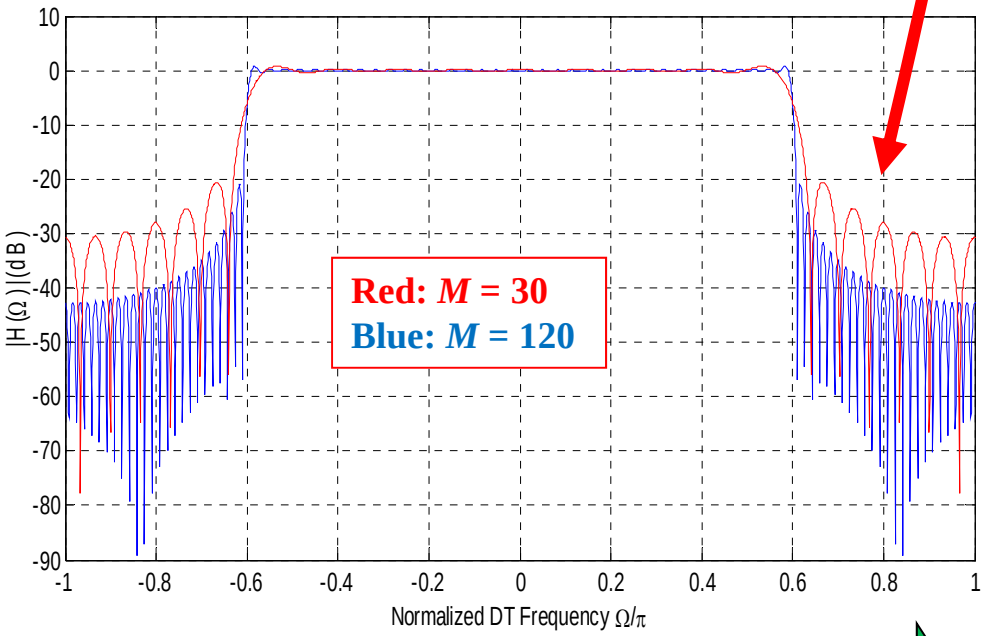
But… we saw that this gave only mediocre designs…
… in particular, it gave poor stop-band performance.

We could use math to explain why but the reason is pretty easy to see by recalling that the frequency response of an FIR filter is the DTFT of the b_m coefficients:

$$H(\Omega) = DTFT\{b_m\} = b_0 + b_1 e^{-j\Omega} + \ldots + b_M e^{-j\Omega M}$$

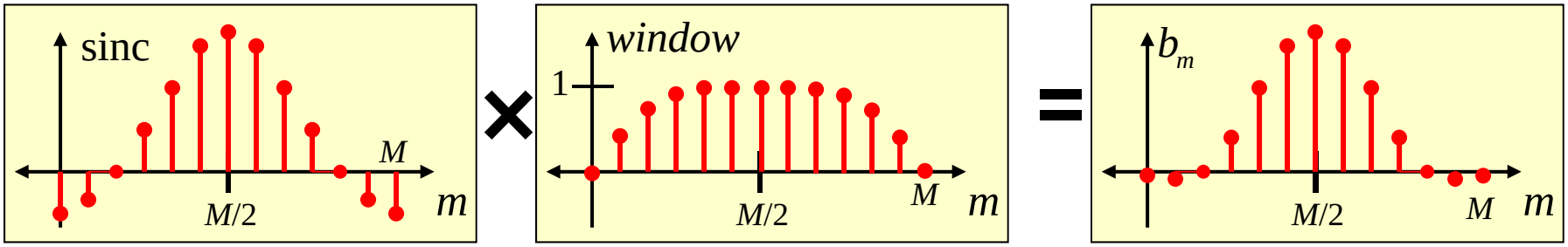
Now… we can reason that:

- The truncation causes discontinuities at the edges of the b_m sequence
- Discontinuities (i.e., rapid changes) require high frequencies
- Thus… the poor stop-band!!!



This reasoning also gives us an idea as to how to fix it:

- Multiply the sinc-generated b_m by a tapering "window"
- Then… we can try different shaped windows to see which is best



MATLAB has a function that lets you easily design filters this way.

That's a "one"

The command is called fir1 and here is how it is used:

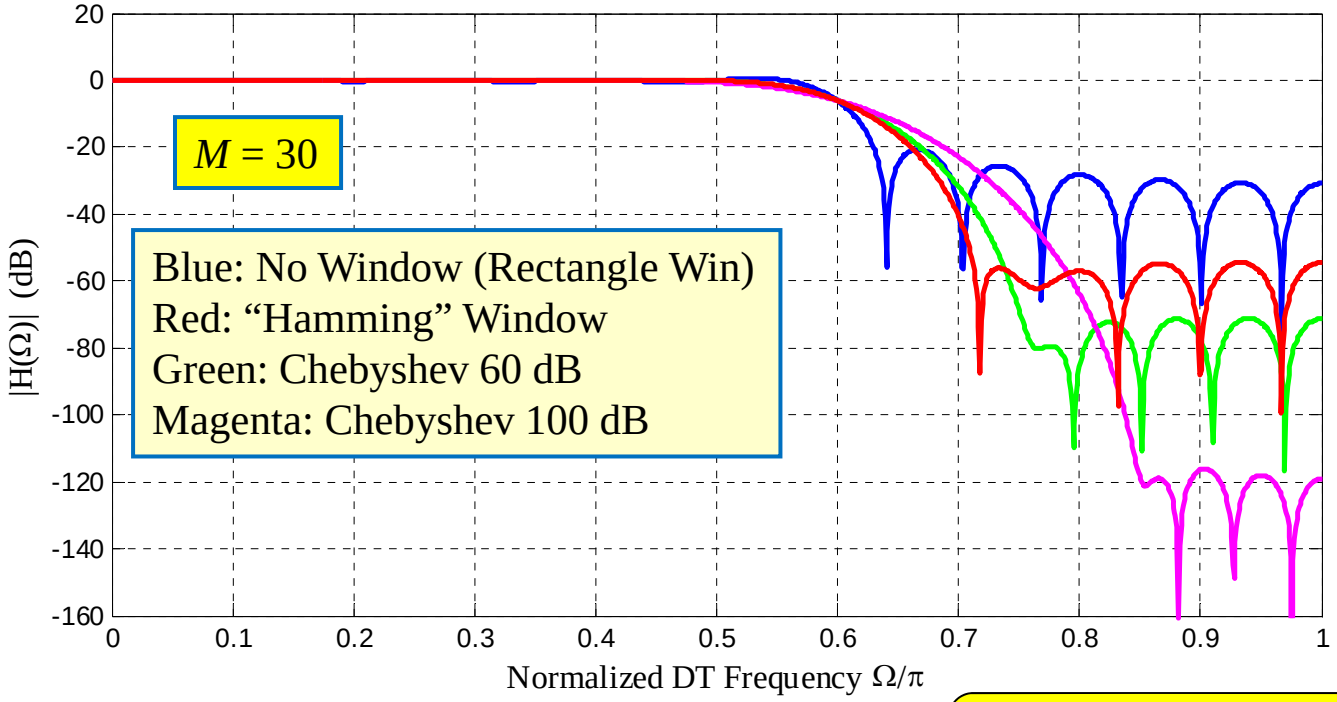
```
>> b = fir1(M,omega_c,window(M+1,opt))
```

Filter "Order"

Normalized Cutoff Frequency between 0 & 1

Specified Window (length = M+1) Some have optional parameters!

fir1 has many other options and can be used to design more than just LPF… see MATLAB help.



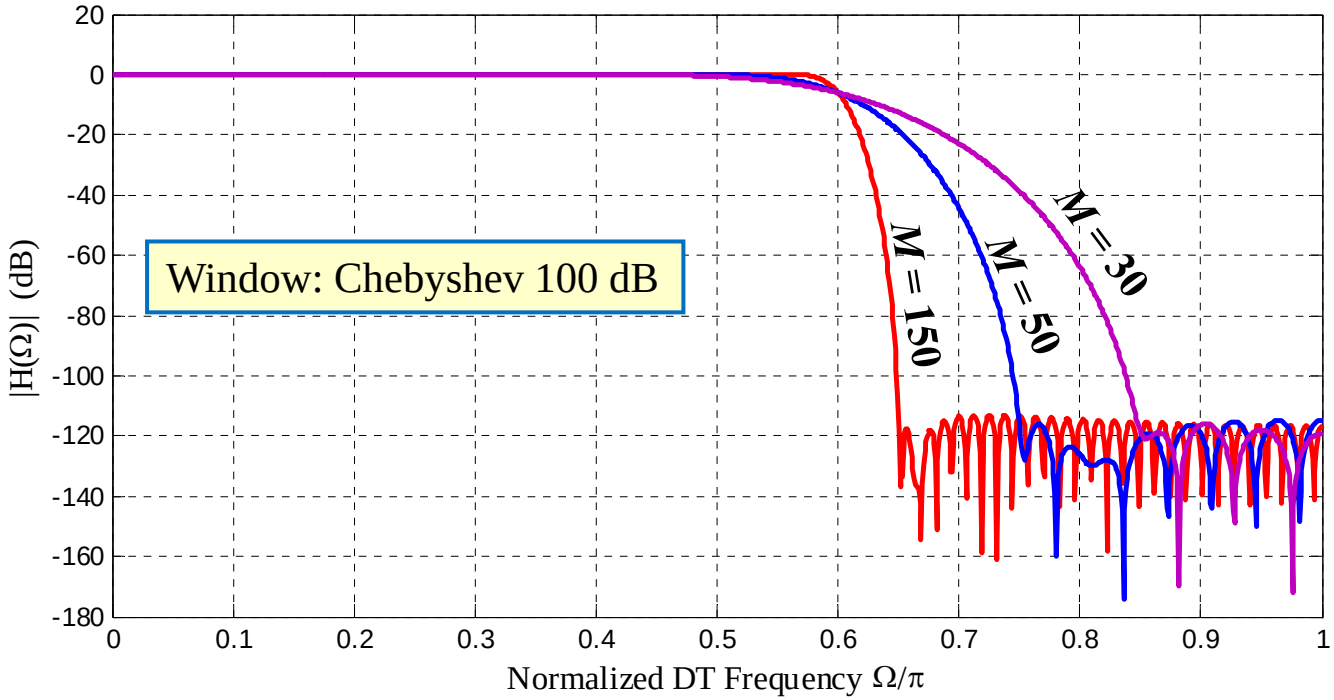

```
>> b_rect=fir1(30,0.6,rectwin(31));
>> b_hamm=fir1(30,0.6,hamming(31));
>> b_chebwin=fir1(30,0.6,chebwin(31,60));
>> b_cheb_100=fir1(30,0.6,chebwin(31,100));
```

Note: Putting 60 (dB) in here gets a stopband of about -70dB

Note: Putting 100 (dB) in here gets a stopband of about -120dB

A General Trend: For a fixed M… choosing the window to get more stopband attenuation (good!) causes the transition band to widen (bad!!).

A General Trend: For a fixed window… choosing Larger M narrows transition band (GOOD!).



So… the general design approach is "guided" trial and error:

- Pick window to get desired stopband level (Chebyshev window helps with this)
- Increase order (*M* = "order") to get desired transition band width
- Note that passband is VERY flat (good!)

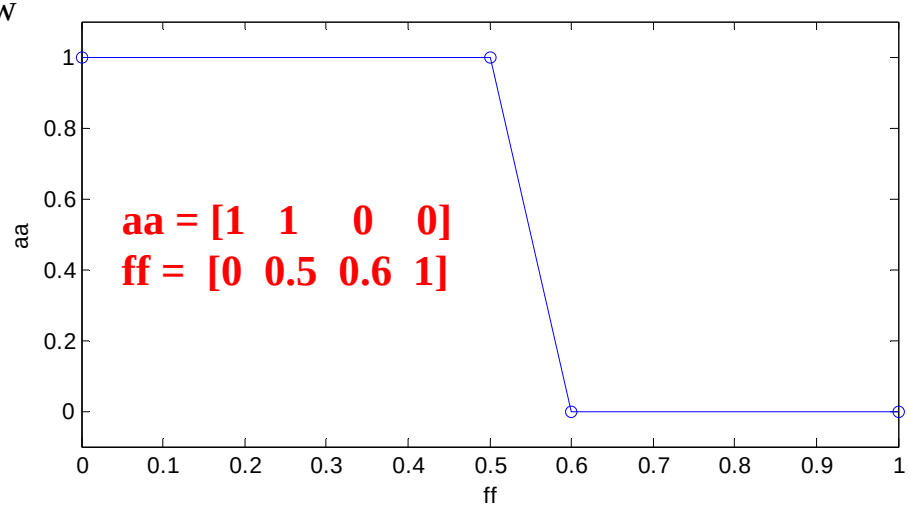
FIR LPF Design – "Frequency Sampling" Method (fir2)

Still uses windowing of a non-causal impulse response…
But it comes from IFFTing a user specified a desired frequency response

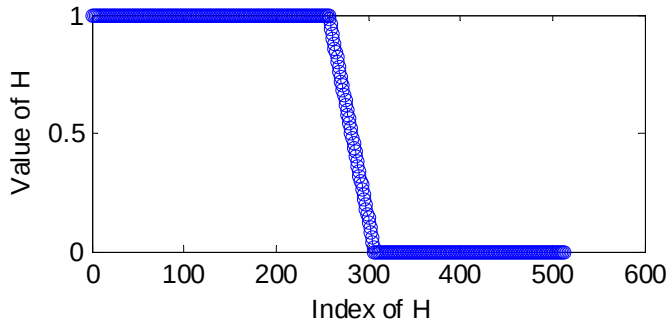
This a simplified version of MATLAB's fir2 command… the real one does not have this constraint and has many options for the design

```
function b = fir2_min(nn, ff, aa)
%   nn = order of the filter... must be an integer...
%        Must be less than 1024!!!
%   ff = row vector of frequency "break points" relative to 1 (which corresponds to Fs/2)
%         First element MUST be 0.   Last element MUST be 1.   Elements must NOT decrease
%   aa = row vector of amplitudes desired   (last element MUST be 0)

% Work with filter length instead of filter order
nn = nn + 1;    %% nn as imput was "order"… now nn is "length"

% Set some parameters needed
   npt = 512;   %%%  npt sets the FFT size used.... This value is good as long as filter order < 1024
  lap = fix(npt/25);   % set # of points to use as transition band if no transition band is spec'd
   wind = hamming(nn);   % use Hamming window

%%% Convert from rows to columns
ff = ff';  aa = aa';
```

MATLAB's fir2 command allows any window

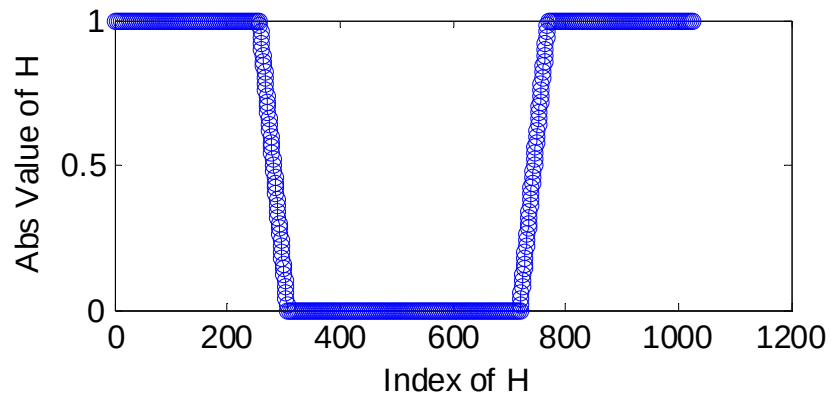


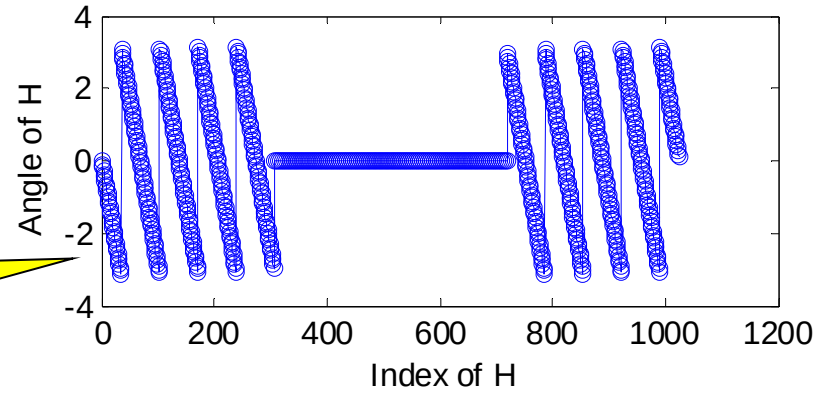
```
% The next few lines and the loop below interpolate breakpoints onto large grid for FFT...
H = zeros(1,npt);
nbrk=length(ff);
nint=nbrk-1;
df = diff(ff');
npt = npt + 1;   % Length of [dc 1 2 ... nyquist] frequencies.
nb = 1;
H(1)=aa(1);
for i=1:nint
   if df(i) == 0
      nb = ceil(nb - lap/2);
      ne = nb + lap;
   else
      ne = fix(ff(i+1)*npt);
   end
   if (nb < 0 || ne > npt)
      error(generatemsgid('SignalErr'),'Too abrupt an amplitude change near end of frequency interval.')
   end
   j=nb:ne;
   if nb == ne
      inc = 0;
   else
      inc = (j-nb)/(ne-nb);
   end
   H(nb:ne) = inc*aa(i+1) + (1 - inc)*aa(i);
   nb = ne + 1;
end
```




```
%% You now have magnitude interpolated onto a fine grid over the positive frequencies
```

```
%%  Now want to apply a linear phase response over these positive frequencies.  The phase slope is related to
%%  amount of delay of filter.... the delay is what is needed to make the filter causal (the delay is half the order)

dt = 0.5 .* (nn - 1); % set delay to half the order (remember that nn is now length and order is length - 1)
rad = -dt .* sqrt(-1) .* pi .* (0:npt-1) ./ (npt-1);  % create j*phi(n) that is a line with desired slope
H = H .* exp(rad);   % multiply magnitude (H) by exp(j*phi(n)) to get mag & phase over positive frequencies

% Now...Append correct values for the negative frequencies.... remember that FT at negative frequencies is just
% conjugate of at positive freqs    Also... since you are putting them "above" the positive freqs things have
% to "run backwards"...  They go "above" because we won't use fftshift when we do the ifft

H = [H    conj(H(npt-1:-1:2))];
```



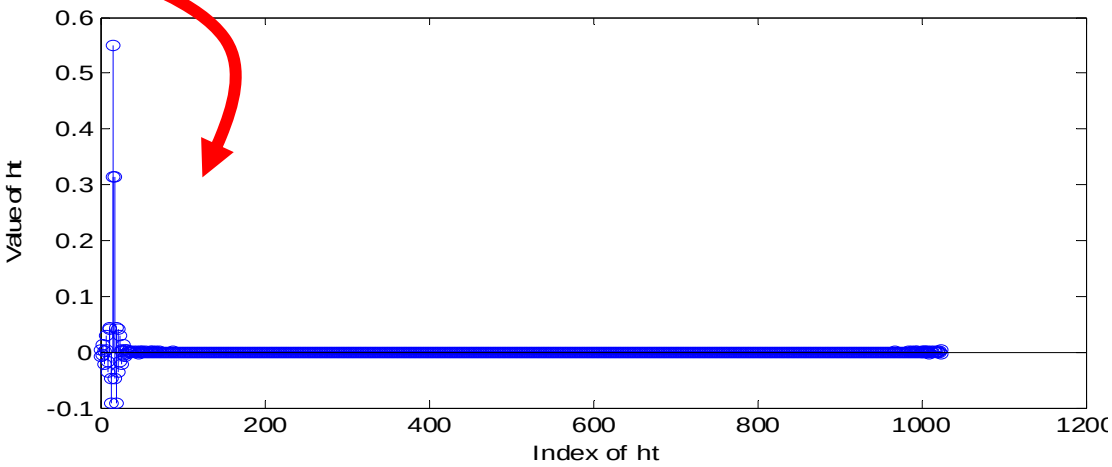


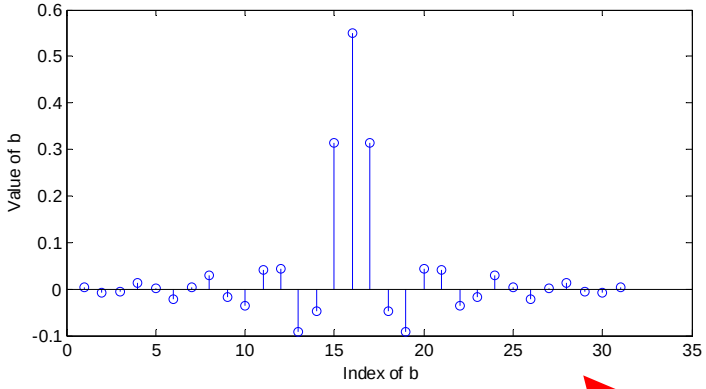
Linear phase… but "wrapped" between -π & π

%%% OK... now have the frequency response spec'd at all freqs on a fine grid and the ordering is positive freqs
%% first then negative freqs.... and we are all set for using ifft without fftshift

ht = real(ifft(H)); %%% technically don't need the real( ) operation but...

%% roundoff causes the imaginary part to be non-zero (but small!) so.... apply real( ) just to be sure





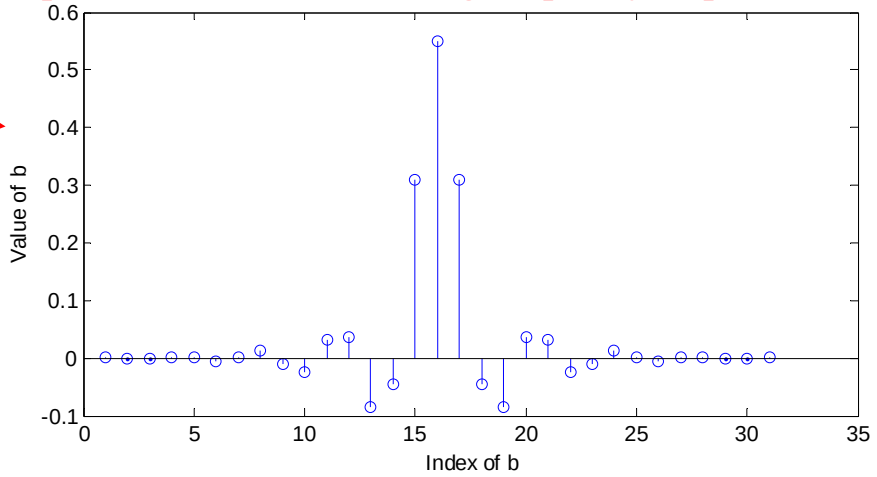
%%% Now you've got an imp. Resp. but it is much longer than desired... so extract the first nn points:

b = ht(1:nn);

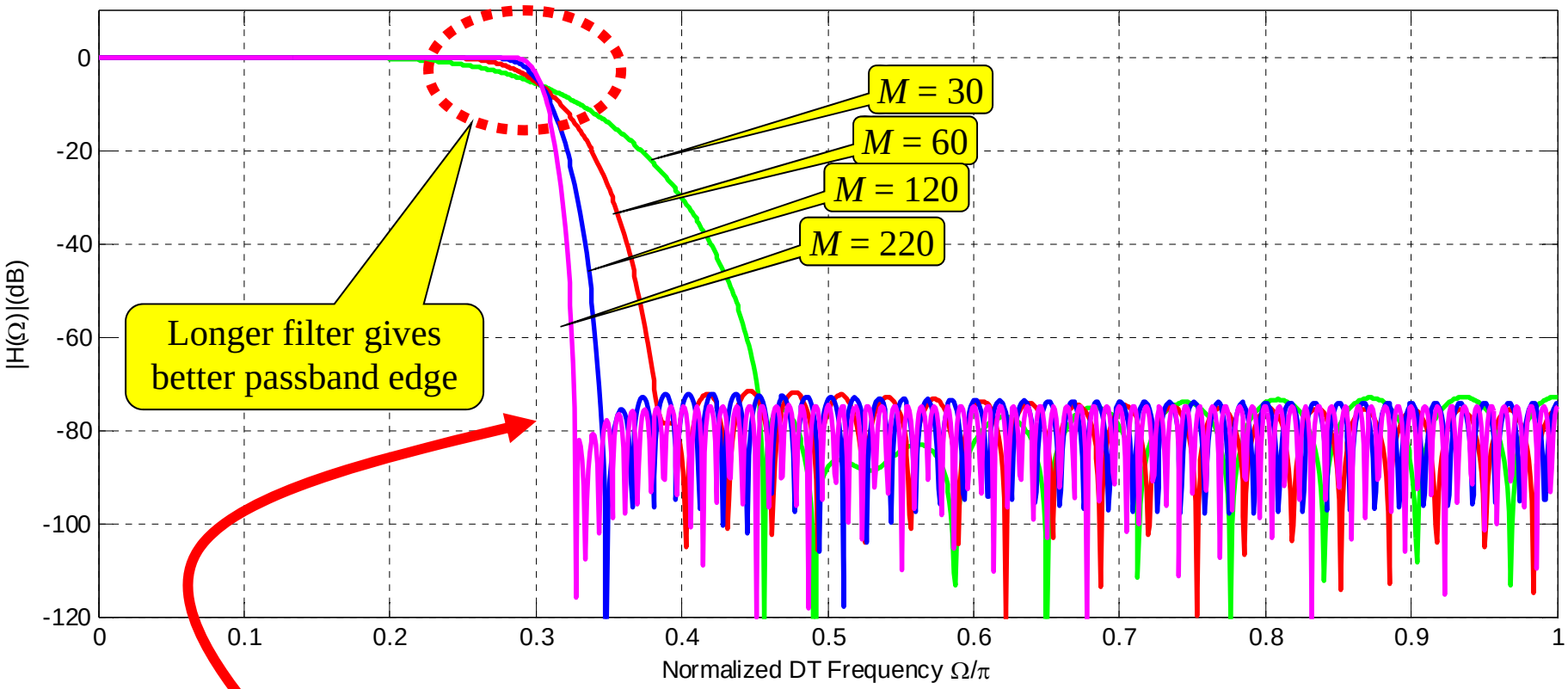
%%% But that abrupt truncation can cause some problems with the resulting frequency response
%% So we apply a window to smooth the
%%% discontinuities at the edges:

b = b .* wind(:).'; % Apply window.

To see the designed filter's frequency response:
>> [H,w] = freqz(b,1,8192);
>> plot(w/pi,20*log10(abs(H)))



B = fir2(M,[0 0.3 0.31 1],[1 1 0 0],chebwin(M+1,60));



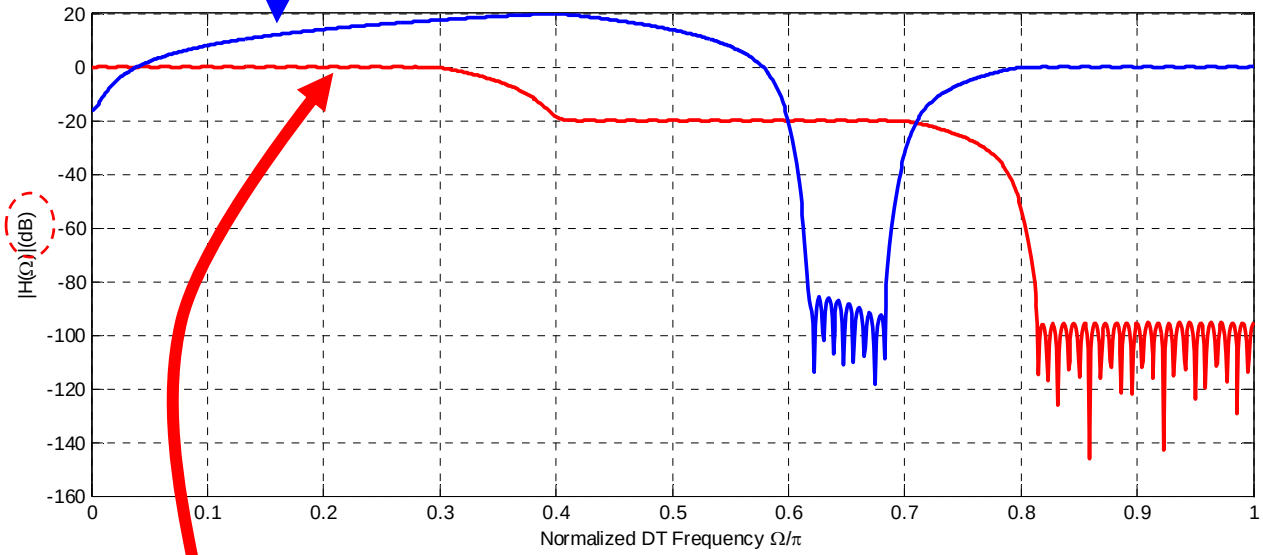
Just because you ASK for a specific transition band does not mean you'll get it!!!

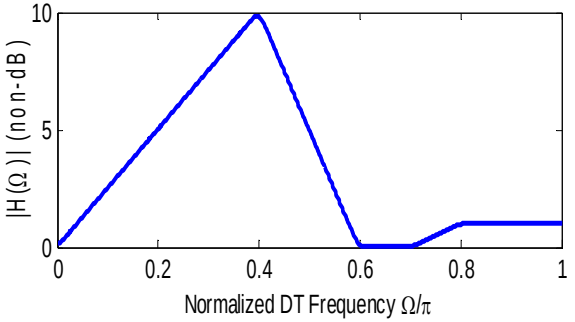
You have to ensure you make your filter long enough to get it!

One advantage of fir2 over fir1: it is easy to get very non-standard filter shapes!!

```
B = fir2(220,[0 0.4 0.6 0.7 0.8 1],[0.01 10 0 0 1 1],chebwin(221,60));
```

Remember… these are non-dB values!






```
B = fir2(220,[0 0.3 0.4 0.7 0.8 1],[1 1 0.1 0.1 0 0],chebwin(221,60));
```

Note: When the last element of aa is non-zero (e.g. for highpass) then the order MUST be specified as being even!!!

FIR LPF Design – Parks-McClellan (firpm)

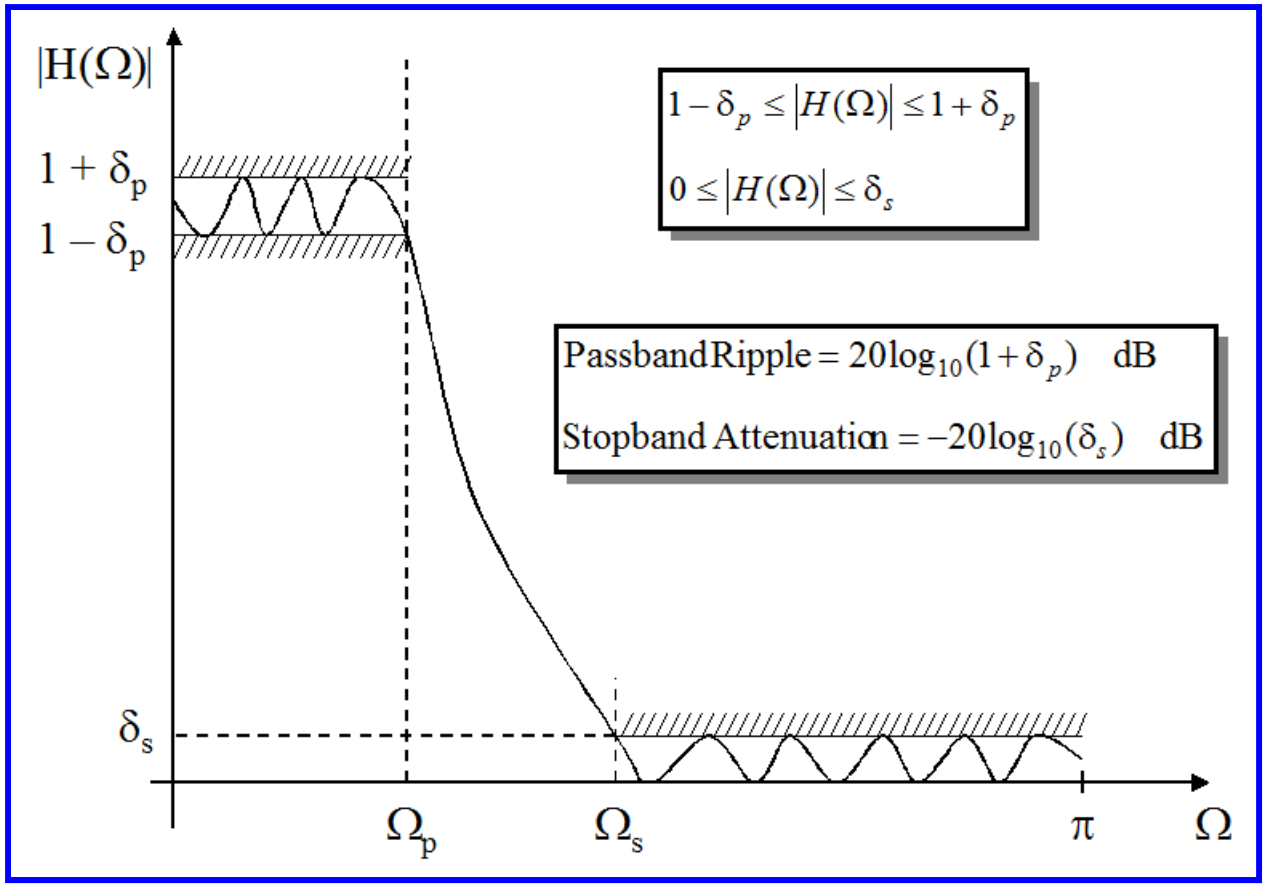
This method is also called “Optimal Equiripple Design”

Unlike the other two methods the math here is quite complex… so we won’t study HOW it does it but only how to apply it

This design method is pretty much the standard for FIR design these days,

Recall our earlier visualization of how we specify a lowpass filter.

firpm and an auxilliary command allow us to specify our desire for these parameters and then design a filter to meet them!



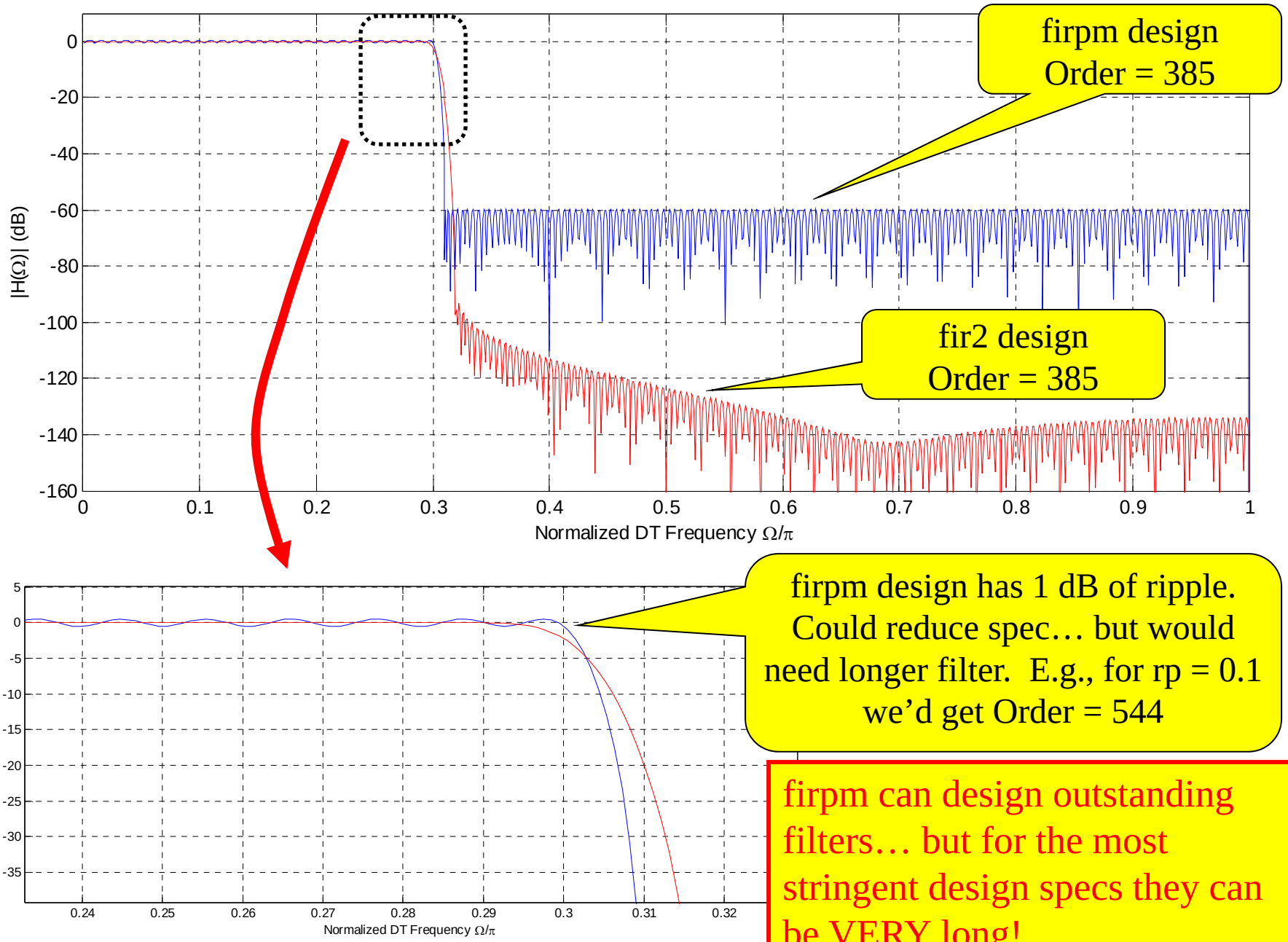
```
% Lowpass Filter Design Specifications:
%         Passband cutoff frequency = 0.3π rad/sample
%         Stopband cutoff frequency = 0.31π rad/sample
% ·       At least 60 dB of stopband attenuation
% ·       No more than 1 dB passband ripple

rp=1; rs=60; % specify passband ripple  & stopband attenuation in dB
f_spec=[0.3 0.31];   % specify passband and stopband edges in normalized DT freq
AA=[1 0];    %%%  specfies that you want a lowpass filter
dev=[(10^(rp/20)-1)/(10^(rp/20)+1) 10^(-rs/20)];   % parm. needed by design routine
Fs=2;   % "Fake" value for Fs so our design is done in terms of normalized DT freq

[N,fo,ao,w]=firpmord(f_spec,AA,dev,Fs);
% estimates filter order and gives other parms needed to run firpm

b=firpm(N,fo,ao,w);     % Computes the designed filter coefficients in vector b
```

Same as the LPF we designed using fir2

About what we got for our fir2 LPF

Our fir2 design gave much lower PBR!!!

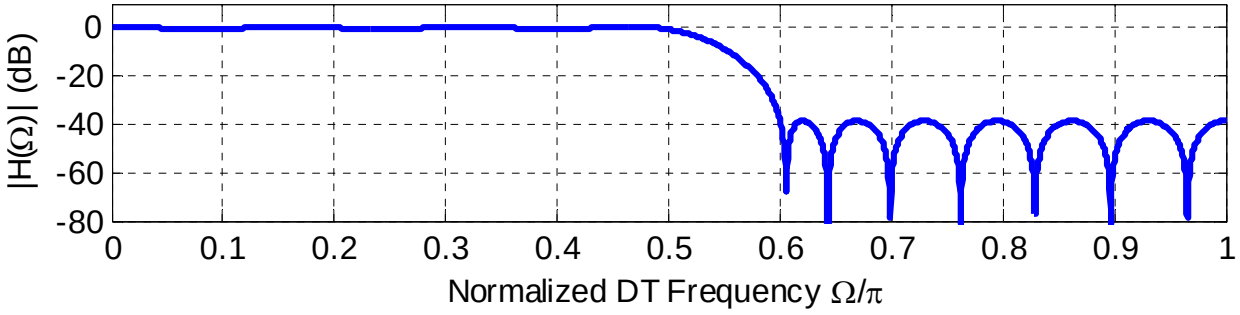
The resulting value for the order for this design is 385!!

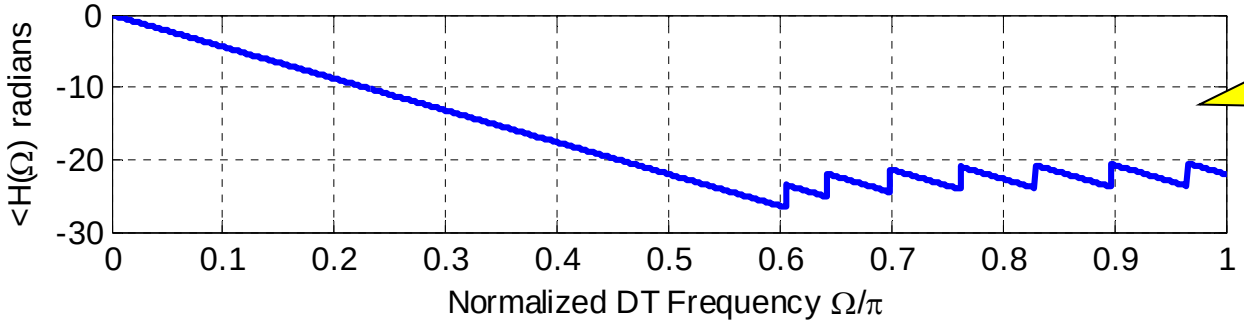


firpm design has 1 dB of ripple. Could reduce spec… but would need longer filter. E.g., for rp = 0.1 we'd get Order = 544

firpm can design outstanding filters… but for the most stringent design specs they can be VERY long!

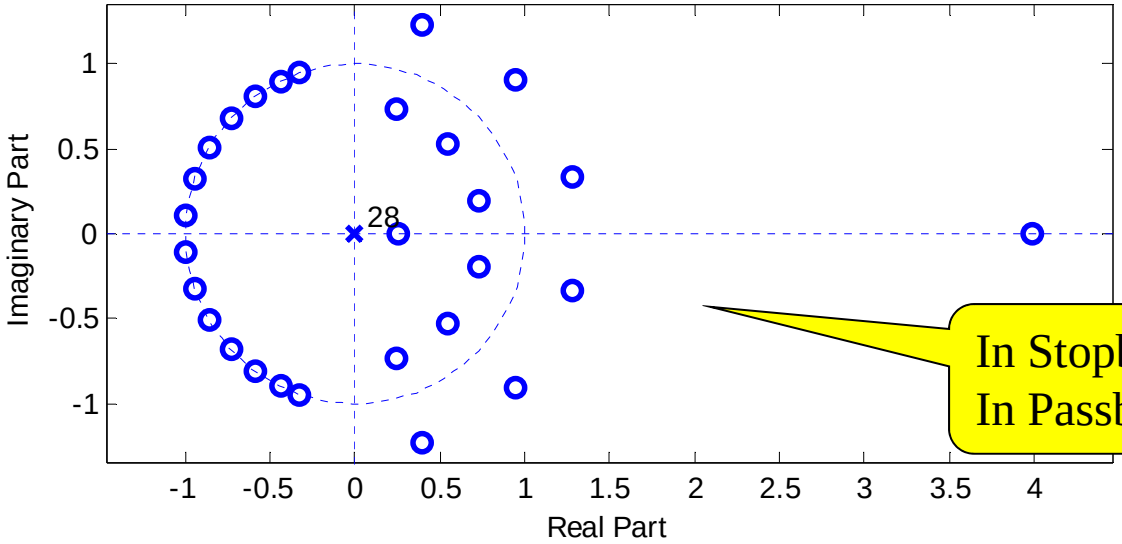
Let’s look at pole-zero plot for a simpler firpm-designed filter…





Linear Phase… all designs by firpm have this very desirable trait!!!

>> zplane(b,1)



In Stopband: zeros placed right on UC
In Passband: zeros “line” the UC