Lecture 9 / Lab 8

IIR Filters

Review

- In FIR Filtering we considered systems with transfer function Y(z) / X(z) = H(z)
  - Z dom : $b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3}$ …
  - Time : b0x[n] + b1x[n-1] + b2x[n-2] + ….
- Many different transfer functions were realizable. High order filter gave near ideal responses
- Linear Phase

Adding Feedback : Recursive Filters

$Y(z) / X(z) = H(z) = B(z) / A(z)$: where $B(z)$ and $A(z)$ are both polynomials

$$A(z) = 1+a1z-1+a2z-2+\ldots.$$

$$B(z) = b0+b1z-1+b2z-2 + \ldots$$

$$Y(z)\ A(z) = X(z)B(z)$$

$$Y(z) = (b0+b1z-1+b2z-2+\ldots.) / (a1z-1+a2z-2+\ldots.)$$

$$Y[n] = (b0x[n]+b1x[n-1]+\ldots.) - (a1y[n-1] + a2y[n-2] +\ldots.)$$

IIR = Infinite Impulse Response

Familiar filtering operation in time domain : to compute output sample, we sum suitably delayed versions of inputs and outputs multiplied with fixed coefficients

Pole Zero Placement

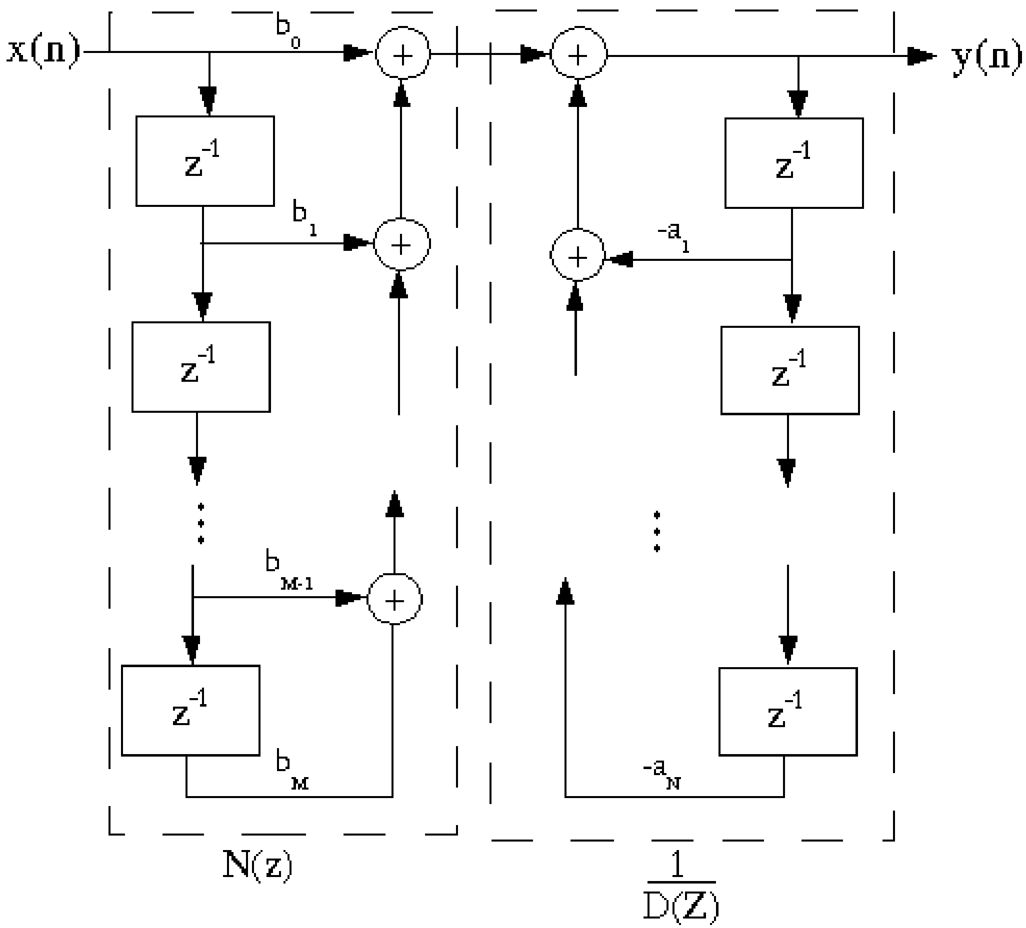
- Factor the polynomials A and B such that H(z) = (z-r)(z-s)… / (z-c)(z-d)…

  zeros poles

- Frequency response is attained by evaluating the expression around the unit circle.
- graphically … the ratio of distance to zeros / poles

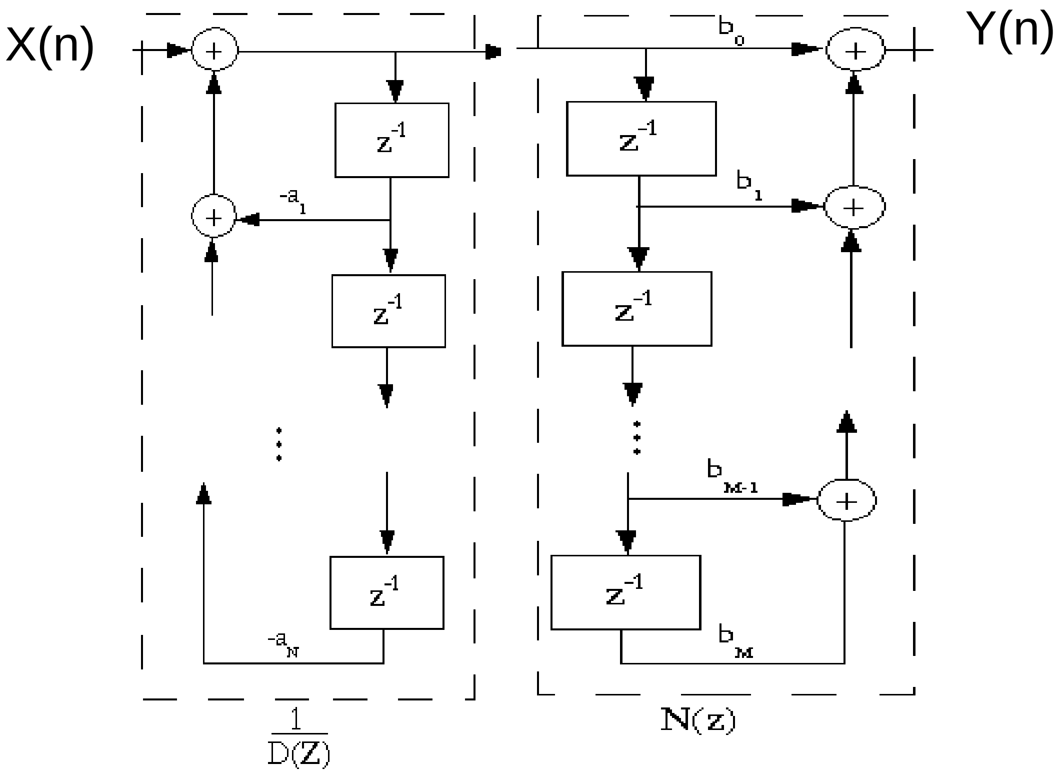
Many IIR filter designs are realized by taking known familiar analog filters and mapping the poles/zeros from the s-plane to z-plane

Direct Form 1 IIR Representation



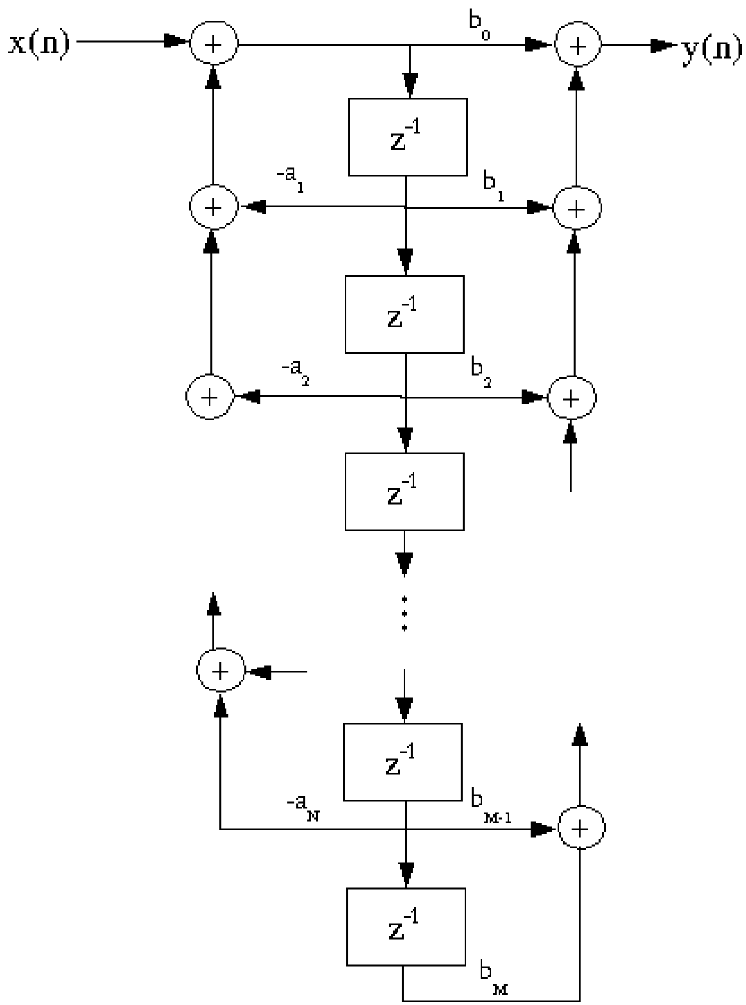
Familiarity with the commonly used forms of IIR filters is important because we will not
Be designing filters ourselves with pole-zero placement, but rather using design sw

Switching the order



Note : response isn't changed by changing the order of the filter sections, but now note that storage elements are redundant

Direct Form II



Common Representation

Examples

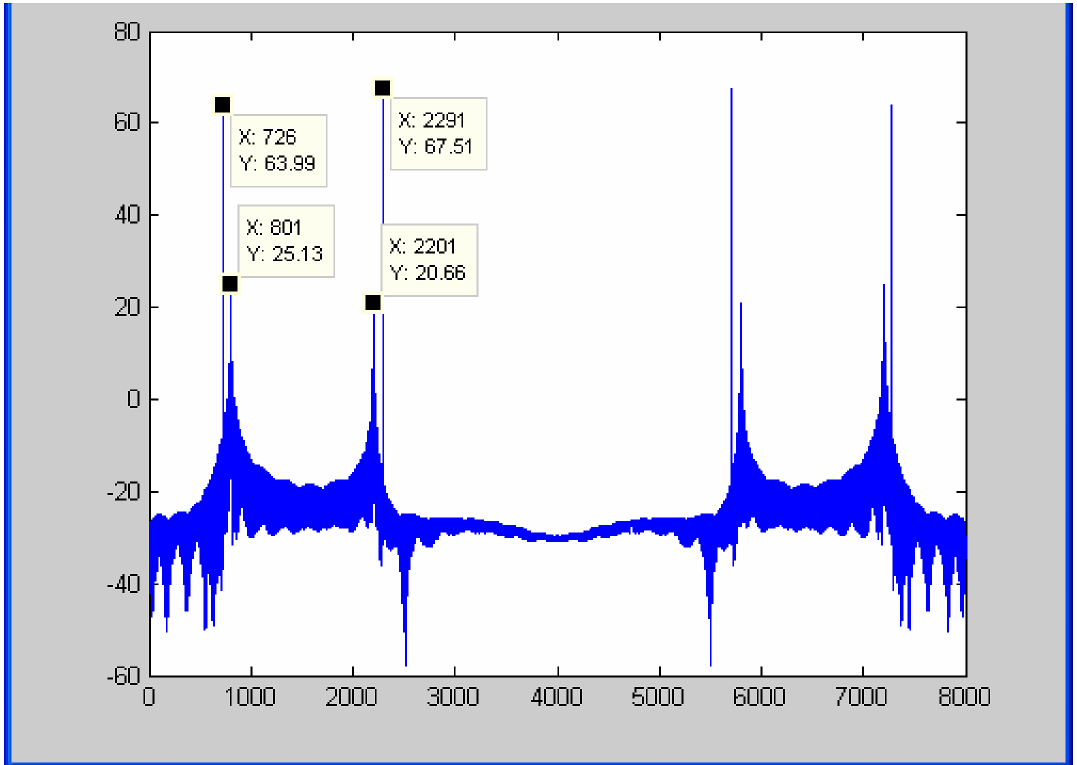
- “Better” (sharper / higher stop attenuation…etc) filter can be realized with lower orders.
  - Less computation required
- Unstable or unstable-like behavior can happen with impulsive input
- High order VERY sensitive to coefficient / data quantization, so filter designed in Matlab using double FP may not work on 16-bit fixed point DSP.
  - C6713 is floating point processor, so not as many worries here

Test Tone from Lab 4

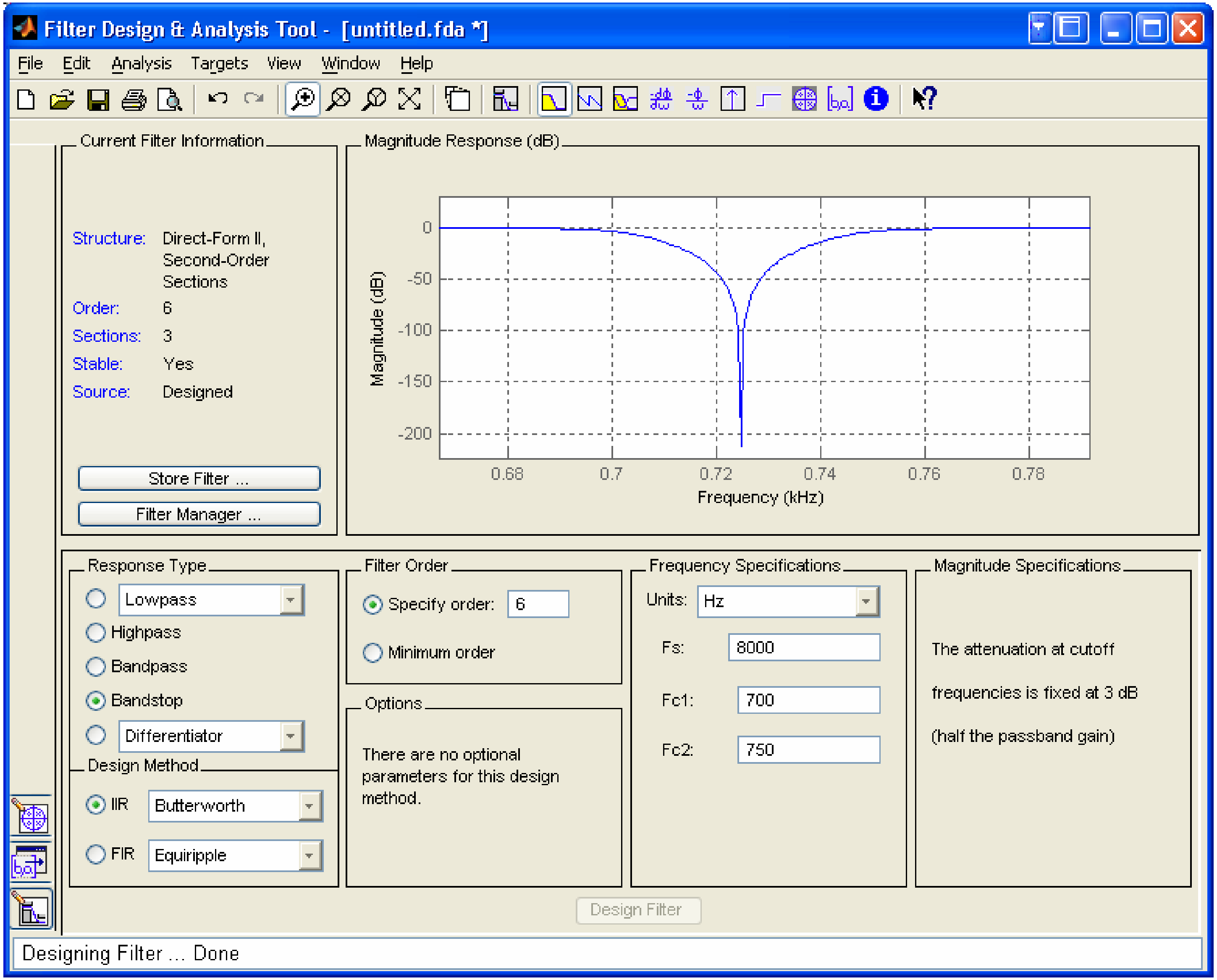
```
test=wavread('test.wav');
test=resample(test,8000,44100);
soundsc(test,8000)
plot(20*log10(abs(fft(test(1:8000)))))
```

To get data in matlab without analog loop

1 second long FFT = 1Hz frequency Resolution.

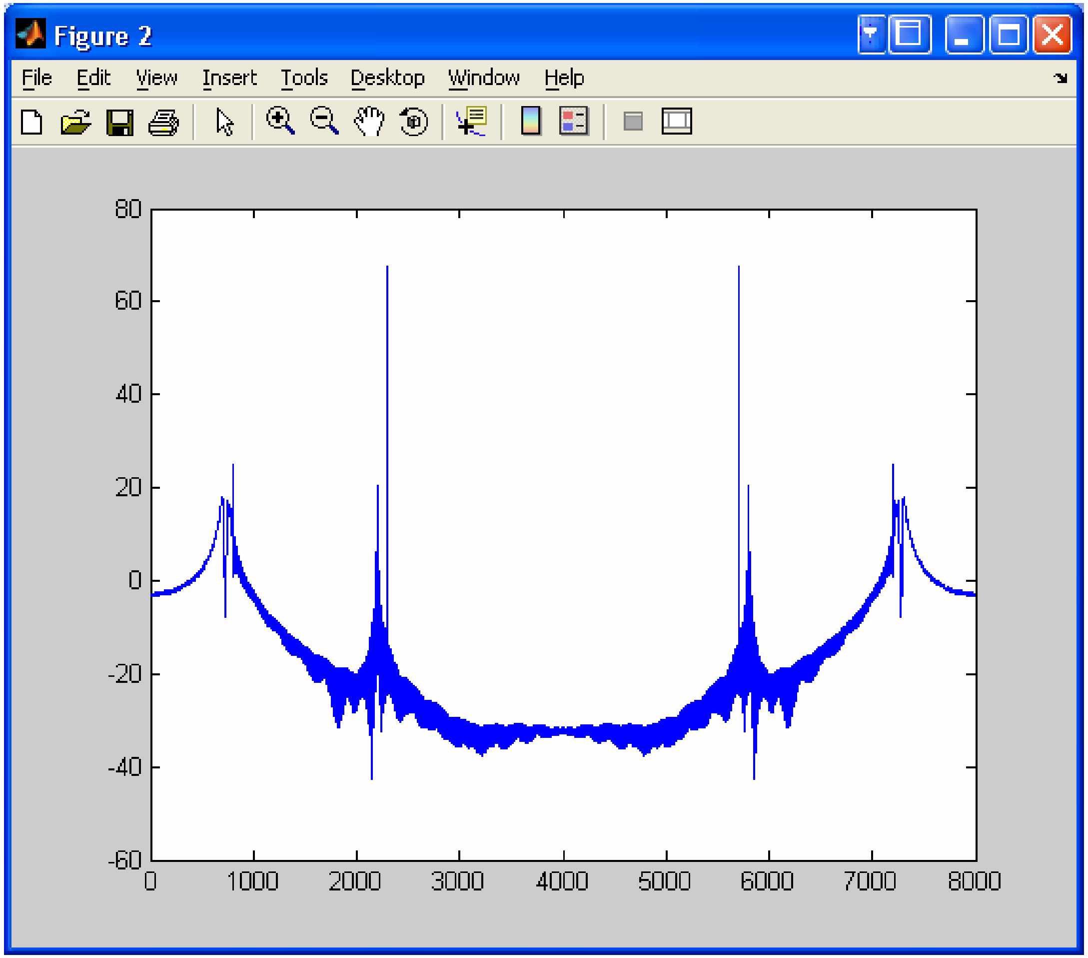


FDA Tool Snapshot



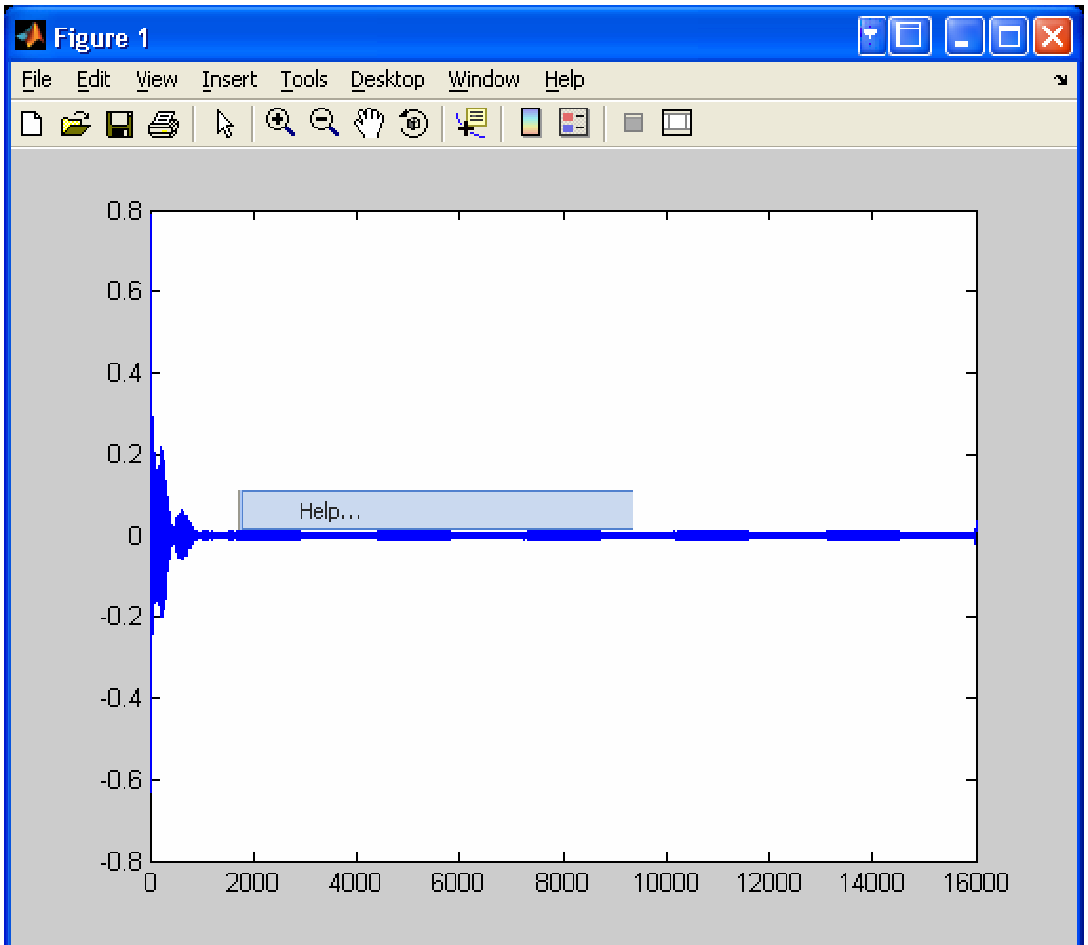
Results of Filter

```
f=filter(Num,Den,test);
soundsc(f,8000);
plot(20*log10(abs(fft(f(1:8000)))))
```



Time domain plot

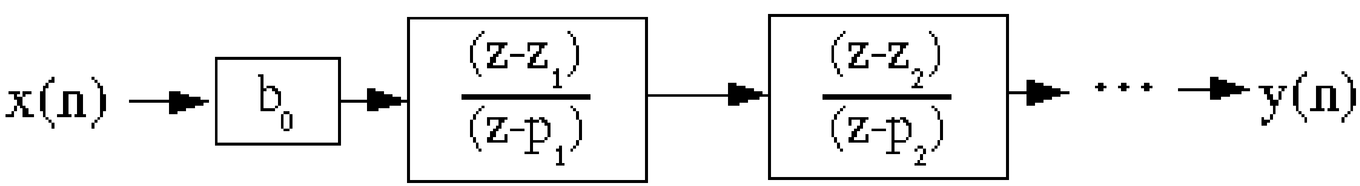
With high tone filtered out as well



Filtered(2000:16000)

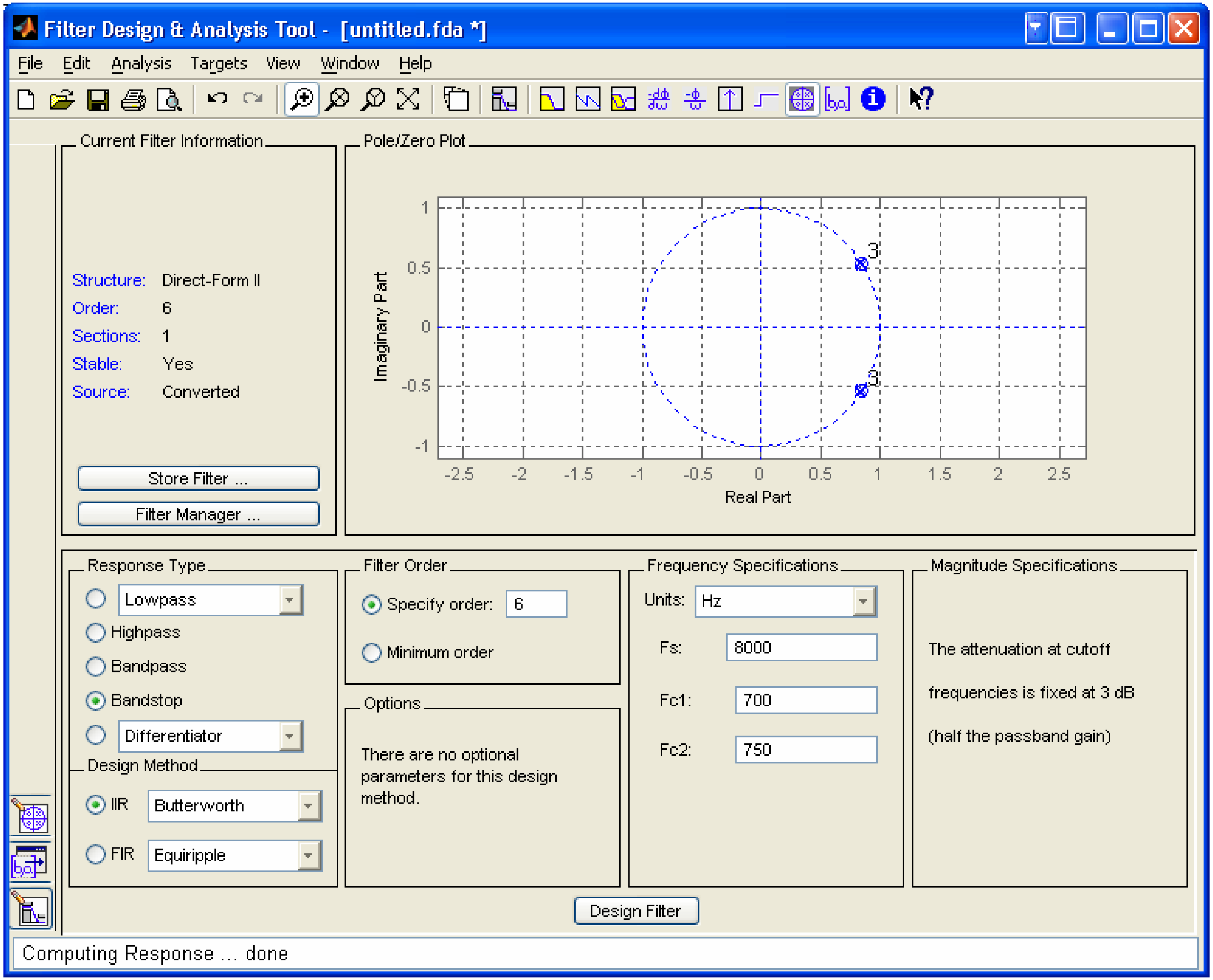
Cascaded Sections

- Second order filters require less dynamic range for intermediate computations if arranged appropriately. Quantization effects are greatly minimized
- Without describing much : transfer function can be factored to show that filter is a cascaded sequence of lower order filters: Ex :

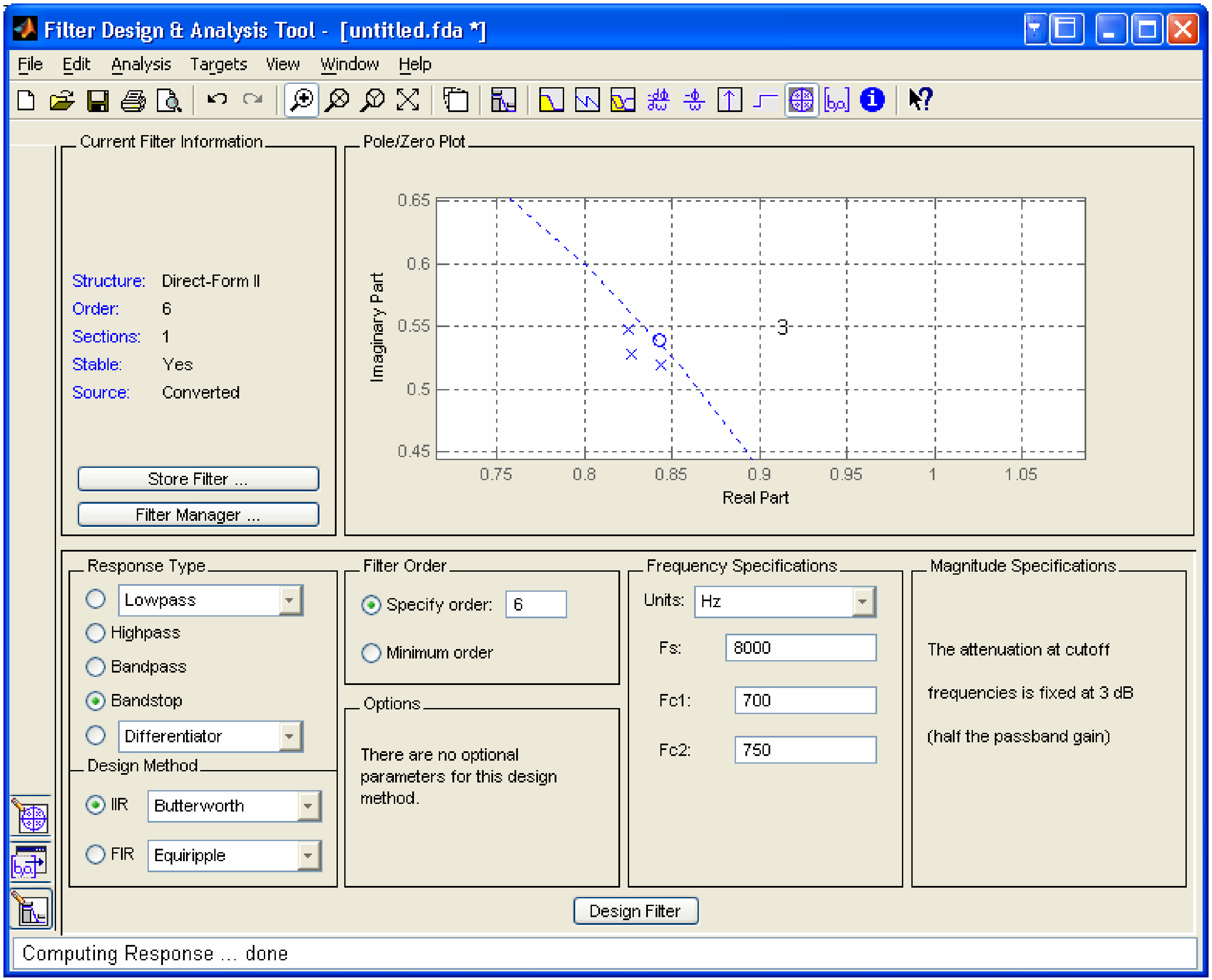


This shows first order sections, but almost exclusively, the accepted form is to implement as cascaded second order sections (SOS) or bi-quads, which combine the complex conjugate pairs of poles and zeros together in one section with real coefficients

Pole-Zero Plot of the Butterworth



Pole-Zero Plot of the Butterworth Bandstop



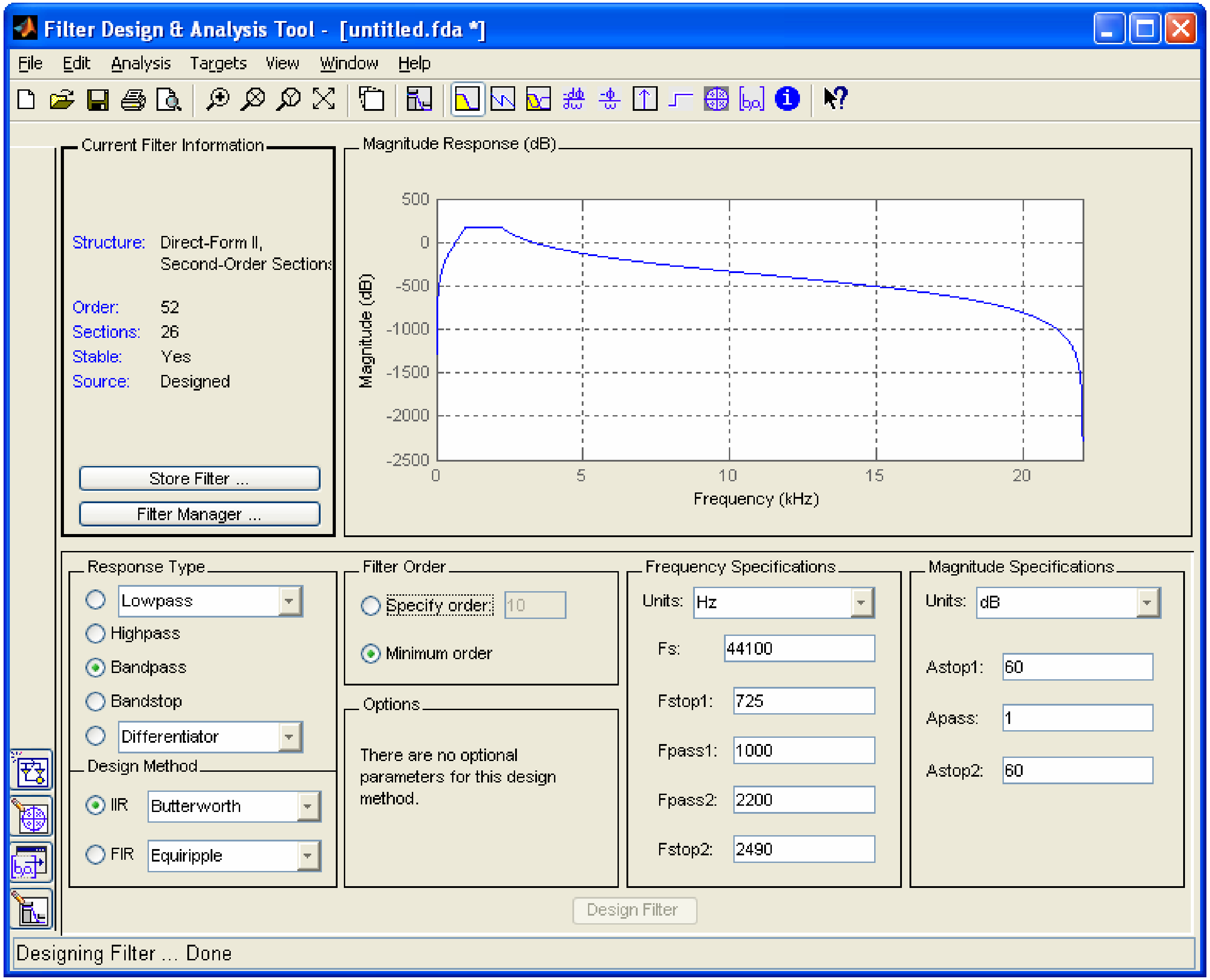
Note that angle Corresponds to The rejection frequency (32 deg)

This is also a little insight into why coefficient quantization could be an issue

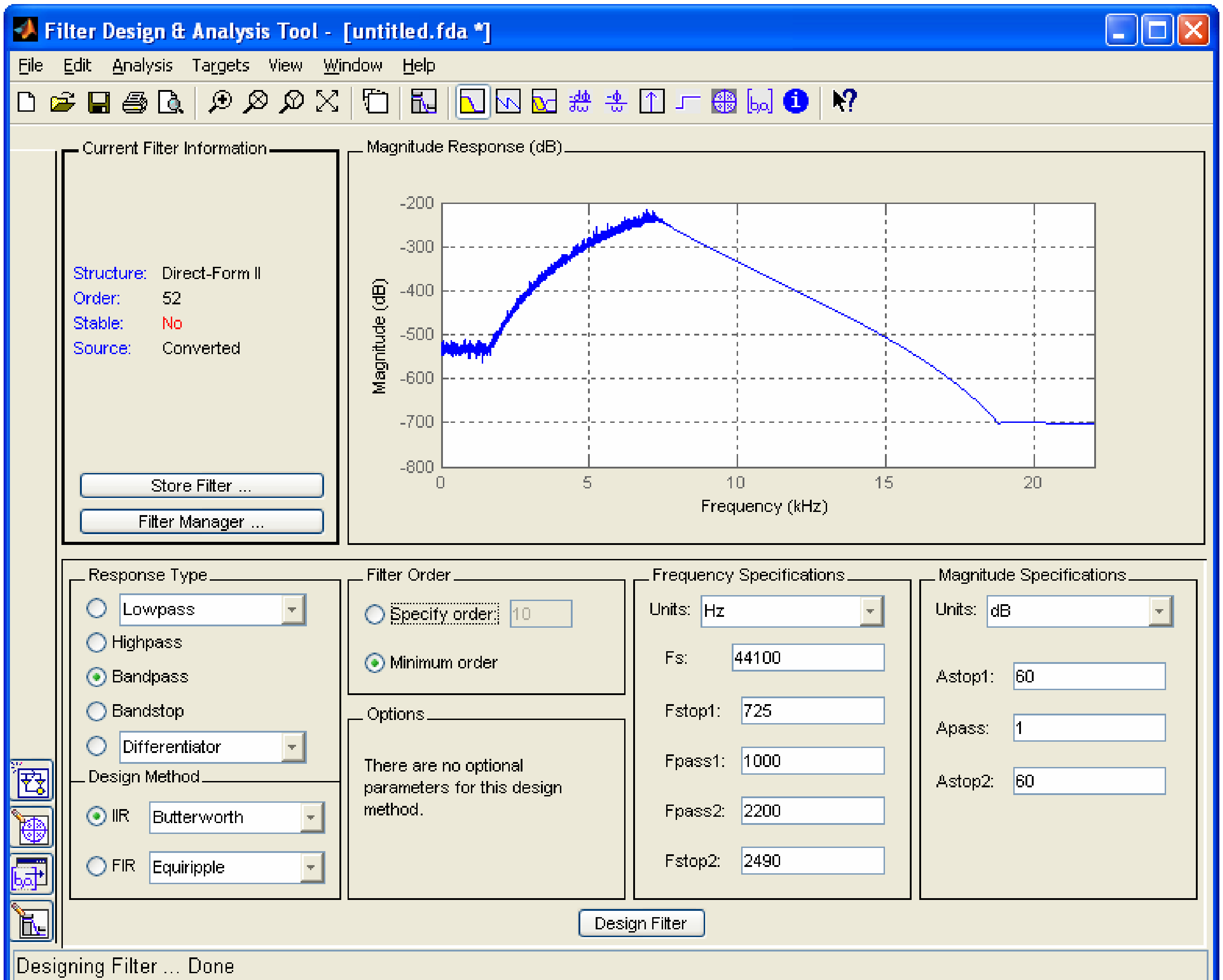
Coefficient/Data Quantization

- Simulate yourself in Matlab by forcibly truncating results at computation stages
  - Forces you to write your own functions
- Matlab Fixed Point Toolbox and Filter Design Toolbox integrate to show effects of finite precision arithmetic on filters
  - Right in FDAtool
  - Lab computers don’t have this!

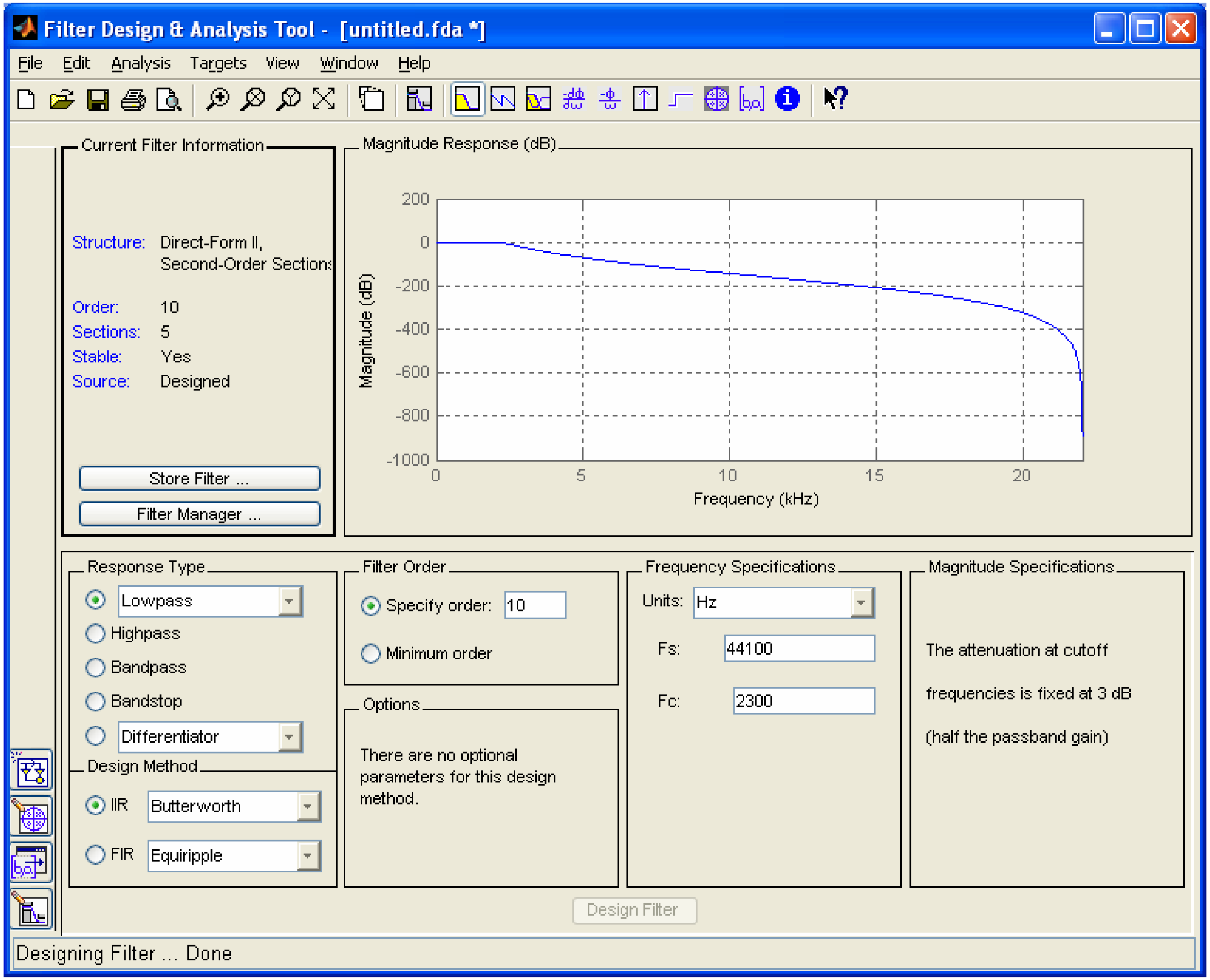
Coefficient Quant



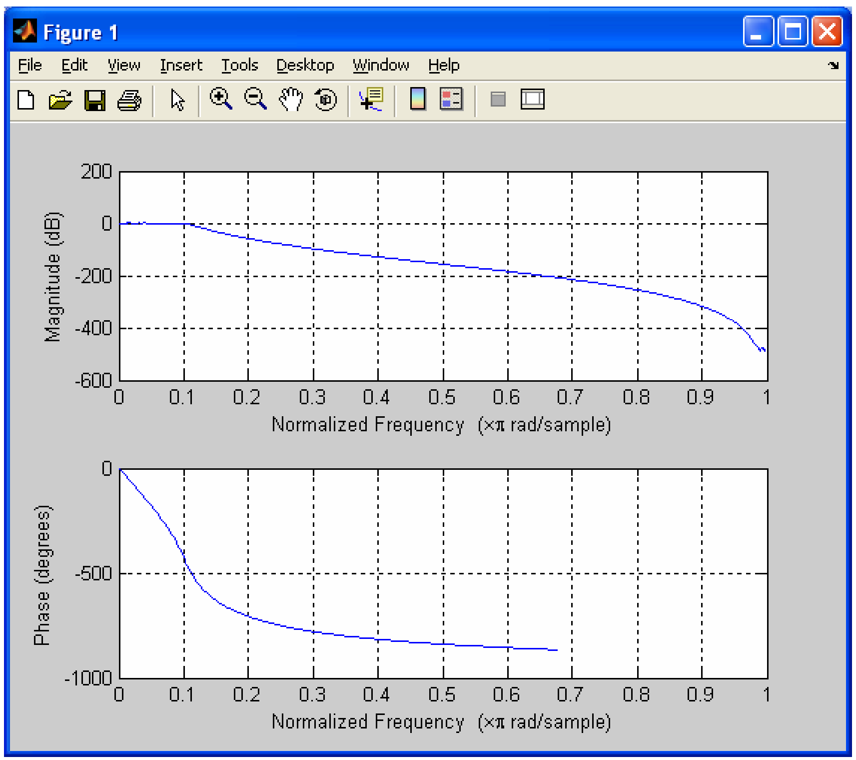
Coefficient Quant



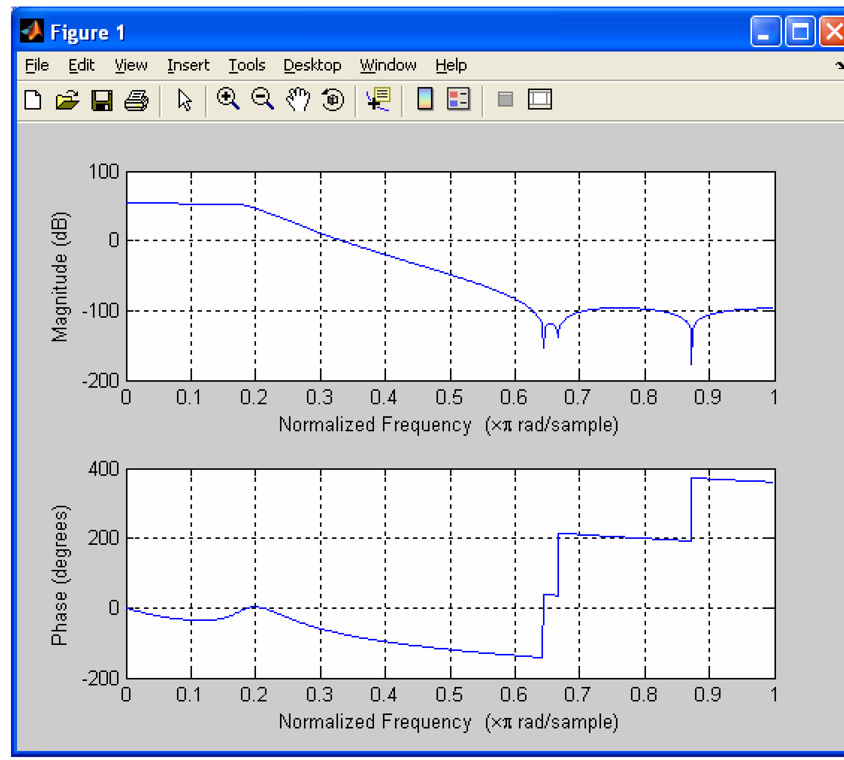
Coefficient Quant



Coefficient Quant



Float



20 bit

TI Library Support

- Functions in DSPLIB
- Most Interesting :
- **Function** void DSPF_sp_biquad (float *x, float *b, float *a, float *delay, float *r, int nx)
- **Arguments**
  - x Pointer to input samples.
  - b Pointer to nr coefs b0, b1, b2.
  - a Pointer to dr coefs a1, a2.
  - delay Pointer to filter delays.
  - r Pointer to output samples.
  - nx Number of input/output samples.

Benchmarks

- Cycles 4 * nx + 76
  - For nx = 60, cycles = 316
  - For nx = 90, cycles = 436

- Process in blocks Circular buffering isn’t really an issue (only 2 delay slots)
- Note that filter delays are external value, and if you don’t mess with them between calls, are preserved.

Lab 5

- Implement IIR Filter with simple real-time analysis of results:
  - Is a tone present?
  - Solutions discussed two weeks from today in class