Lesson 07: Filter Design

It's time we abide by the Hippocratic Oath of Signal Processing[1] and start to design filters. Designing a filter may seem daunting, but luckily for us, MATLAB provides us with easy-to-use digital filter design tools,[2] simplifying the process to nothing more than calling functions with the appropriate arguments.

Design Considerations

As an engineer, it is your job to determine the appropriate trade-offs when designing a filter for an application, as it really does come down to using the "right" tool for the job. For now, let's consider only the following types of IIR filters:

Name	Passband	Stopband
Butterworth	Monotonic	Monotonic
Chebyshev I	Equiripple	Monotonic
Chebyshev II	Monotonic	Equiripple
Elliptic	Equiripple	Equiripple

Table 1: IIR Filter Types

Given the table above, we can select a filter type that will suit our needs. For example, we'd use a Butterworth filter in an audio application as we often desire monotonic behavior. After all, we wouldn't want our filter to vary the attenuation of frequencies in our passband and stopband, as that would sound awful.

Once we've decided on a filter type, we need to specify a few properties:

- Filter order
- Peak-to-peak passband ripple
- Stopband attenuation
- Passband/Stopband edge frequencies
- Size of the transition band

That said, for the rest of the lesson we'll be designing a Chebyshev II bandstop filter as the process is largely the same for the other filter types.

[1] Do no harm to the signal. Anything you do will harm the signal.
[2] `https://www.mathworks.com/help/signal/filter-design.html`

Determining Filter Order

When designing a filter, we often specify all the properties but the order and leave that as a final step for MATLAB.

```
fs = 44.1e3;
fn = fs / 2;

Wp = [1.0e3, 19.0e3] / fn;
Ws = [2.2e3, 16.8e3] / fn;
Rp = 4;
Rs = 40;

[n, Ws] = cheb2ord(Wp, Ws, Rp, Rs)
```

```
n =

     5

Ws =

    0.0998    0.7619
```

Here, we're designing a filter to operate at 44.1 kHz, with passband corner frequencies at 1.0 and 19.0 kHz, along with stopband corner frequencies at 2.2 and 16.8 kHz normalized to the Nyquist frequency. We're also allowing for 5 dB of peak-to-peak ripple in the passband and we're mandating the stopband be attenuated by 40 dB.

An important thing to note about MATLAB order functions for bandpass/bandstop filters is that the returned order of the prototype filter is **half** of the actual order. The reason for this is quite simple: a bandpass/bandstop filter is composed of a low-pass and high-pass filter!

Filter Generation

Once we have the order, we are ready to generate the filter:

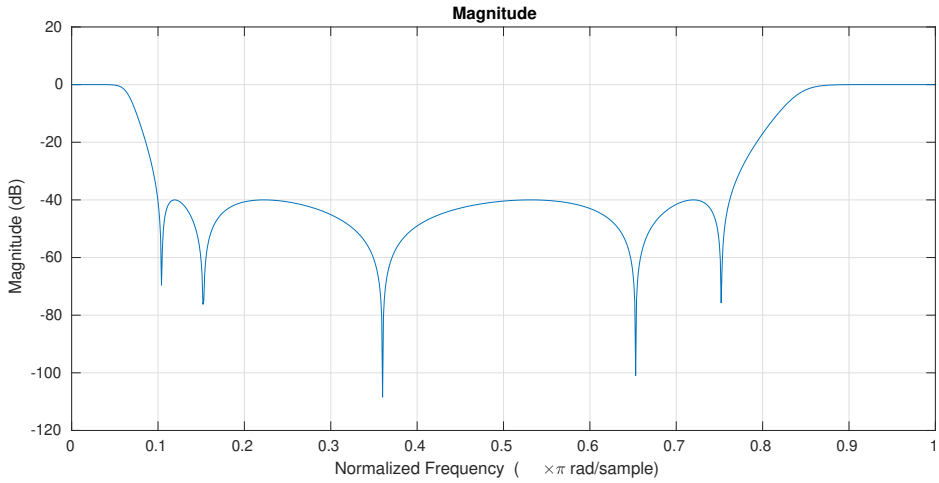
```
[z, p, k] = cheby2(n, Rs, Ws, 'stop')
```

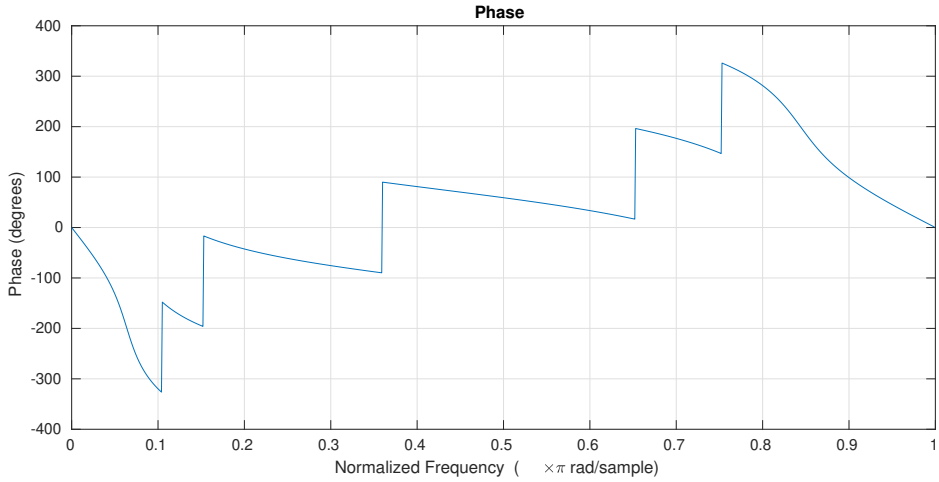
```
z =

   0.4259 + 0.9048i
   0.4259 - 0.9048i
  -0.7120 + 0.7022i
  -0.7120 - 0.7022i
  -0.4622 + 0.8868i
  -0.4622 - 0.8868i
   0.8875 + 0.4609i
   0.8875 - 0.4609i
   0.9469 + 0.3215i
   0.9469 - 0.3215i

p =

  -0.7863 + 0.4216i
  -0.7863 - 0.4216i
  -0.6012 + 0.2788i
  -0.6012 - 0.2788i
  -0.4865 + 0.0000i
   0.9337 + 0.1903i
   0.9337 - 0.1903i
   0.7558 + 0.0000i
   0.8313 + 0.1434i
   0.8313 - 0.1434i

k =

    0.0234
```

Note that this step largely depends on the filter you're trying to generate; hence, you must reference the corresponding MATLAB documentation. Also, note that we're also using the zero-pole-gain variant of the output to avoid numerical instability that comes with a polynomial transfer function.

We can quickly confirm that we got our desired filter with a quick call to `freqz()`:

```
[b, a] = zp2tf(z, p, k);

figure;
freqz(b, a, 1e3);
```





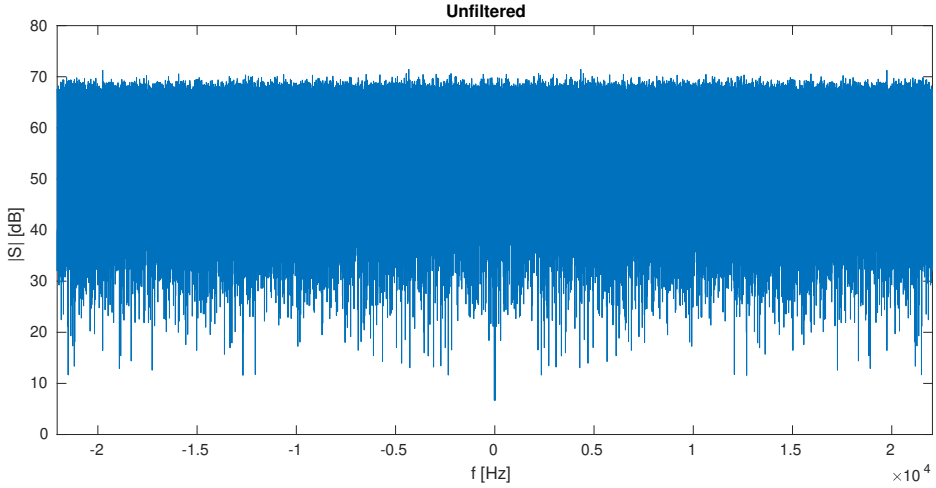
Filtering

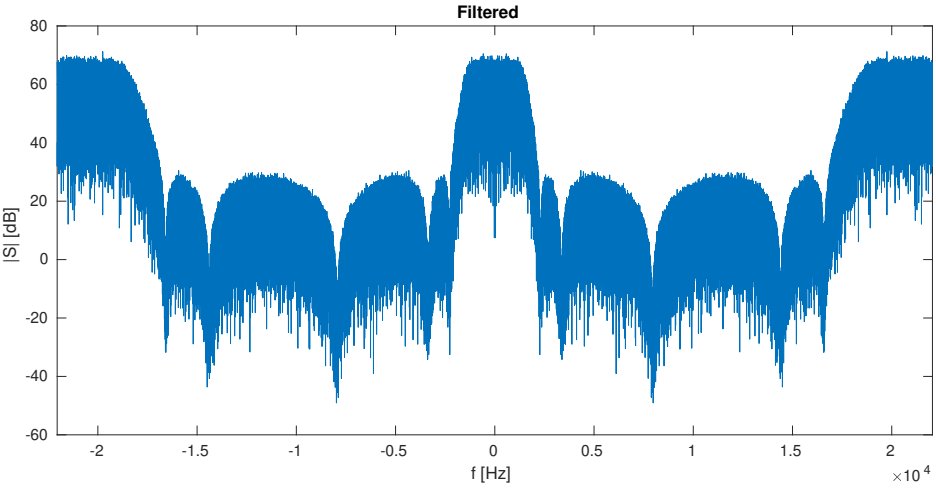
Say you wanted to apply your filter to some signal, as one does, then there are two options in MATLAB:

Polynomial Representation

Let's generate white noise and apply our filter:

```
l = 1e6;
s = randn(1, l);
S = fftshift(fft(s));

s_filt = filter(b, a, s);
S_filt = fftshift(fft(s_filt));

figure;

subplot(2, 1, 1);
plot(fs / l * (-l / 2:l / 2 - 1), 20 * log10(abs(S)));
title('Unfiltered');
ylabel('|S| [dB]');
xlim([-1, 1] * fn);
xlabel('f [Hz]');

subplot(2, 1, 2);
plot(fs / l * (-l / 2:l / 2 - 1), 20 * log10(abs(S_filt)));
title('Filtered');
ylabel('|S| [dB]');
xlim([-1, 1] * fn);
xlabel('f [Hz]');
```



Second-Order-Section Representation

When numerical stability becomes an issue, we can instead represent our digital filter as a cascade of ***Second-Order-Sections (SOS)***:

$$H(z) = gH_L(z) \dots H_1(z) \tag{1}$$

where:

$$H_k(z) = \frac{b_{k0} + b_{k1}z^{-1} + b_{k2}z^{-2}}{1 + a_{k1}z^{-1} + a_{k2}z^{-2}} \tag{2}$$

and g is the overall gain of the system.

We can then realize this as a matrix of the form $\mathrm{M}_{L\times 6}(\mathbb{F})$:

$$H_{\text{sos}} = \begin{bmatrix} b_{10} & b_{11} & b_{12} & 1 & a_{11} & a_{12} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{L0} & b_{L1} & b_{L2} & 1 & a_{L1} & a_{L2} \end{bmatrix} \tag{3}$$

We can apply our filter as a SOS like so in MATLAB:

```
[sos, g] = zp2sos(z, p, k)

s_filt = g * sosfilt(sos, s);
```

```
sos =

    1.0000   -0.8518    1.0000    1.0000   -0.2693   -0.3677
    1.0000    0.9244    1.0000    1.0000    1.2024    0.4392
    1.0000   -1.7749    1.0000    1.0000   -1.6626    0.7116
    1.0000    1.4240    1.0000    1.0000    1.5725    0.7960
    1.0000   -1.8938    1.0000    1.0000   -1.8673    0.9079

g =

    0.0234
```

