


Lab 6

Time and frequency domain analysis of LTI systems

I. GENERAL DISCUSSION


In this lab and the next we will further investigate the connection between time and frequency domain responses. In this lab we will establish the theoretical background, begin using the Fourier Transform and investigate these relations using Matlab simulation. Here we will also introduce Matlab programming using m-files and some more useful Matlab functions. In the next lab these same systems will be explored experimentally using real hardware and the Spectrum analyzer. As preparation for this lab you need to read over the following and familiarize yourself with the basics of program creation before coming to lab.

Start with the Matlab users guide (under the Help section of the program) at:

 ▶ MATLAB ▶ User Guide ▶ Programming Fundamentals ▶ Functions and Scripts ▶ Program Development

And read the subsections entitled “Overview”, “Creating a Program” and “Getting the Bugs Out”

Next starting at:

 ▶ MATLAB ▶ User Guide ▶ Programming Fundamentals ▶ Functions and Scripts ▶ Working with M-Files

in the Matlab users guide read the next 4 sections entitled “Overview”, “Types of M-Files”, “Basic Parts of an M-File” and “Creating a Simple M-File”.

Frequency domain techniques are universally used for the analysis and design of wave filters whose purpose is to modify a sinusoid only in amplitude and phase. As such, these filters are necessarily linear and time-invariant. This particularly means their input-output relationship can be characterized using the signals and systems concept of the time domain *impulse response* $h(t)$. Remember, this unique signal is what happens when a fictitious delta function signal, $\delta(t)$ is applied to the input. Although the delta function is counter-intuitive: it has infinite amplitude, zero duration in time but unit area, it possesses the desirable property of containing *all frequencies*. Hitting an LTI filter with a delta is the mathematically the same as having all frequencies simultaneously applied to the filter, so conceptually at least, it should be evident how the resulting impulse response shapes these different frequencies in amplitude and phase (time-delay), and therefore completely generalizes any LTI system. This enables us to then find the response to any arbitrary input signal using time domain convolution.

Since we are usually interested in the steady-state *frequency response* of wave filters, we seek a way to characterize them directly in the frequency domain. That is, we want to know how applied sinusoids (tones) appear at the output. Such a technique exists and requires that we find a frequency domain function as general as the impulse response. This is known as the *system response* or *system function* $H(\omega)$ or $H(f)$, and is easily found by simply taking the Fourier Transform (FT) of the impulse response.

The utility of working directly in the frequency domain avoids the otherwise messy time domain math typically involving ordinary differential equations and often arduous convolution integrations. Generally, this greatly simplifies things, since ordinary differential equations are transformed into algebraic equations, and finding the response to an arbitrary input otherwise

requiring time domain convolution, becomes an exercise in relatively simple algebraic multiplication. The key is having or knowing $h(t)$ and making skillful use of appropriate Fourier Transform “pairs”. This lab uses the frequency domain system function to simulate and experimentally observe the frequency and phase responses for two different low-pass filters (LPF), a first order RC LPF and second order Butterworth LPF. **Observe this only works with applied sinusoids.**

The Fourier Transform (FT) defines a unique relation between a signal in the time domain and its representation in the frequency domain. Being a transform, no information is created or lost in the process, so the original signal can be recovered from knowing the FT and using the Inverse FT. This symmetrical relationship is an example of what is meant by “FT pairs”. Observe the FT results in two pieces of information necessary to invert back to the time domain: amplitude versus frequency, and phase versus frequency. In electrical engineering, signals having both amplitude and phase information are expressed mathematically using single complex numbers.

The FT mappings for real or complex valued continuous time voltage signals are given below:

$$[1a] \quad G(f) = FT[v(t)] = \int v(t)e^{-j2\pi ft} dt$$

or

$$[1b] \quad G(\omega) = FT[v(t)] = \int_{-\infty}^{+\infty} v(t)e^{-j\omega t} dt$$

The Inverse Fourier Transform (IFT or FT^{-1}) is defined by:

$$[2a] \quad v(t) = FT^{-1}[G(f)] = \int_{-\infty}^{\infty} G(f)e^{+j2\pi ft} df$$

or

$$[2b] \quad v(t) = FT^{-1}[G(\omega)] = \int_{-\infty}^{\infty} G(\omega)e^{+j\omega t} d\omega$$

Using the first or second relation of each transform equation depends on whether we are primarily working with Hertzian frequency “ f ” (cycles per second) or radian (angular) frequency “ ω ” (radians per second). Theoretical treatments generally use radian frequency for brevity, while experimental work or work having experimental application use Hertzian frequency because that is what our instruments use. When converting back and forth, don’t forget the factor 2π ! (about 6 if you are doing rough calculations in your head).

Formally, solving the time domain differential equations associated with a physical LTI system, using a Dirac delta function as input and zero initial conditions (no energy in the system), gives the impulse response $h(t)$. From this we can find the system response or system function,

$$[3] \quad FT[h(t)] = H(f)$$

$$[4] \quad FT^{-1}[H(f)] = h(t)$$

This symmetrical relationship defines the usual FT pair:

$$[5] \quad h(t) \Leftrightarrow H(f) \text{ or } h(t) \Leftrightarrow H(\omega) \text{ or sometimes } h(t) \Leftrightarrow H(j\omega)$$

Note the last form is generally associated with “s-domain” theory, where $s = j\omega$. You will learn more about this later in the course when the Laplace transform is discussed.

Suppose we know a filter is linear and time-invariant and want to find its system function. We can do this directly in the frequency domain by specifying the FT of the input and output signals. Consider how this is done. The time convolution property and its dual, the frequency convolution property state that if

$$[6] \quad v_1(t) \Leftrightarrow V_1(\omega) \text{ and } v_2(t) \Leftrightarrow V_2(\omega)$$

convolution in time is equivalent to multiplication in frequency:

$$[7] \quad v_1(t) * v_2(t) \Leftrightarrow V_1(\omega)V_2(\omega), \quad \textbf{(time convolution)}$$

and multiplication in time is equivalent to convolution in frequency:

$$[8] \quad v_1(t)v_2(t) \Leftrightarrow V_1(\omega)*V_2(\omega), \quad \textbf{(frequency convolution)}.$$

Inspection of [9] reveals that if we now let $v_2(t) = h(t)$, and we know that $h(t) \Leftrightarrow H(\omega)$, then

$$[9] \quad v_1(t) * h(t) \Leftrightarrow V_1(\omega)H(\omega).$$

Hence, finding $V_1(\omega)$ and $V_2(\omega)$ we can find $H(\omega)$ since we know

$$[10] \quad V_1(\omega)H(\omega) = V_2(\omega), \text{ or } H(\omega) = \frac{V_2(\omega)}{V_1(\omega)}$$

Eqn. (10) is equivalent to doing this the hard way - by time convolution:

$$[11] \quad v_1(t) * h(t) = v_2(t) = \int_{-\infty}^{+\infty} v_1(\tau)v_2(t-\tau)d\tau$$

but in general is much easier to calculate than the integral in [11].

II. Low-Pass Filters

Let us turn our attention now to the input/output characteristics of two linear time-invariant systems that are different implementations of the basic electrical low pass filter¹. Conceptually, these circuits pass frequencies below some reference frequency, called the *cutoff frequency* f_c (or ω_c) without appreciable attenuation; this range is aptly called the *passband*. The filter otherwise attenuates frequencies above cutoff; this region is called the *stopband*. Ideal low-pass filters have a *brick-wall response* separating the passband and stopband regions that isn't possible to realize experimentally, where anything above f_c is completely blocked and anything below is passed. Therefore, increasing the input frequency with practical filters will result in the output gradually or rapidly decreasing. By convention, the cutoff frequency is defined where the amplitude has decreased by -3dB with respect to the input (also the point where power output has decreased by two). Filter input-output amplitude relationships are usually drawn as two-dimensional x-y log-log plots of magnitude response vs. frequency (Hertzian or radian) called *Bode Plots*, where $y = 20\log_{10}|H(f)|$, and $x = \log_{10} f$. Remember that for the FT to be invertible, both amplitude *and* phase (time-delay) must be retained. The Bode Plot expresses only the amplitude part, while a second plot expresses phase vs. frequency, where $y = \theta(f)$ and $x = \log_{10} f$ resulting in a linear-log relationship. Note that phase is found from the input-output time delay for each frequency.

The simplest possible low-pass filter can be made from a series resistor and capacitor with the output taken across the capacitor as shown.

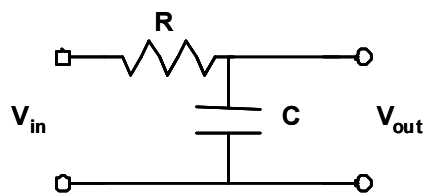


Fig. 1. Single pole low-pass filter.

¹ A very informative discussion of low-pass filters can be found at Wikipedia: http://en.wikipedia.org/wiki/Low-pass_filter. This article has many useful explanatory links.

This circuit, despite its simplicity, is widely used in electrical and computer engineering. It has a cutoff frequency given by $f_c = \frac{1}{2\pi RC}$ [Hz], where the RC product in the denominator is also known as the *time-constant*, $\tau = RC$, since τ has physical dimensions of time. A normalized Bode plot is shown below, where the y-axis shows the gain never exceeds $1 \left[\frac{V}{V} \right]$ (or 0 dB). One advantage of the log-log relationship is the rate at which a filter attenuates frequencies beyond cutoff always appears linear. The RC filter shown has a *roll-off* rate of -20 dB /decade or -6dB / octave. This means that increasing the frequency by a factor of 10 (a decade) will cause the output to decrease by -20 dB or a scalar input-output factor of 1/10 (since $20 \log \frac{1}{10} = -20 \text{ dB}$). An octave is a doubling in frequency, so every increase in frequency by a factor of two will cause the output to decrease by -6 dB or a scalar input-output factor of two (since $20 \log \frac{1}{2} = -6 \text{ dB}$). These relations are shown in Fig. 2.

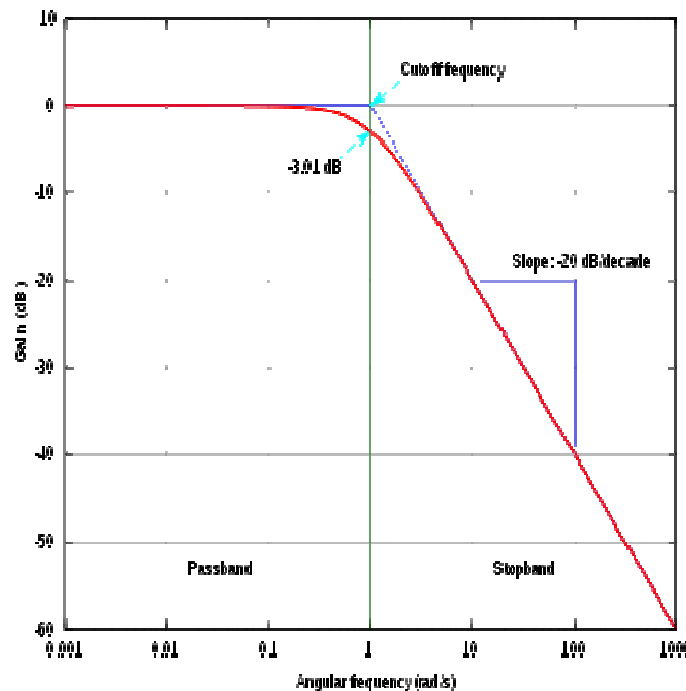
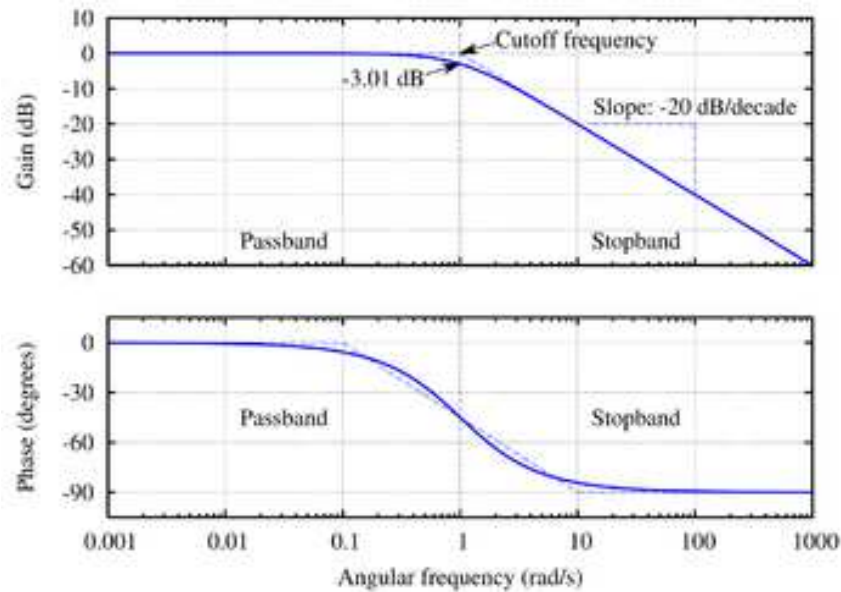


Fig. 2. Bode plot for a normalized single time-constant or 1st order RC LPF². The intercept point of the projected straight line segments occurs at the corner frequency.

This circuit has impulse and system responses: $h(t) = \frac{1}{RC} e^{-t/RC} u(t)$, and $H(f) = \frac{1}{1 + j2\pi fRC}$.

² Graphs ibid. Wikipedia.



The two left plots² show the gain and phase plots for the same RC LPF. Notice particularly that the phase is -45 degrees at the corner frequency. Observe that cascading two of these filters would increase the rolloff rate to -40 dB / decade (-12 dB / octave) and doubling the time delay or -90 degrees phase shift.

[I'm here now! ... scp]

The famous Butterworth LPF has a magnitude gain response given by $|H(f)| = \sqrt{\frac{1}{1 + (2\pi f)^{2n}}}$,

where n is the number of reactive elements (poles) in the LPF. In our case, this would be the number of capacitors. Butterworth only dealt with filters with an even number of poles in his paper. His plot of the frequency response of 2, 4, 6, 8, and 10 pole filters is shown as A, B, C, D, and E in this original graph.

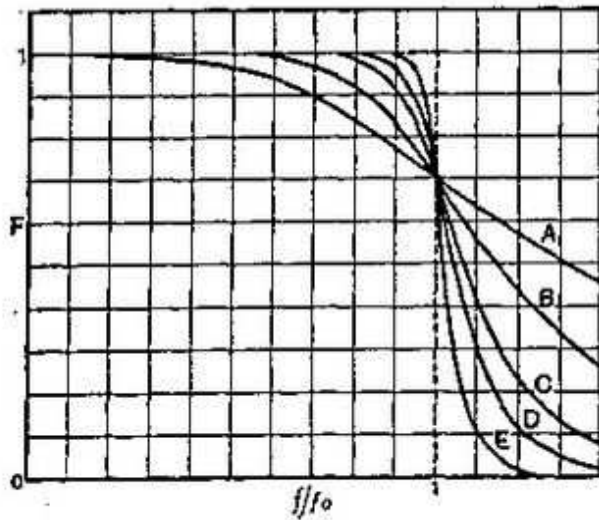


Fig. 3.

Frequency response plot from the original paper.

The RC low pass filter above is a first order Butterworth filter. To look at a more complicated system, we now consider a 3rd order Butterworth filter. First we repeat an RC filter result.

Overview

The frequency response of the Butterworth filter is maximally flat, that is it has no ripples in the passband, and rolls off towards zero in the stopband. These filters are often used in audio work because of their maximally flat amplitude response.

The system response for a second order Butterworth LPF is given by

$$H(f) = \frac{1}{1 + \sqrt{2} + (j2\pi f)^2}$$

III. Lab Assignment

Be sure that you have done the prelab reading indicated above. First we will list some of the functions that you will need to do the following exercises. Often the names used in Matlab differ from those that you might expect from your math courses. Use the Help system to search on these names to better understand how to use them. The goal is to familiarize you with various functions and for you to get proficient at figuring out how to use the help files, manual and other resources to make stuff work in MATLAB.

Useful Functions:

`abs(z)` - returns the magnitude of a complex number or vector

`angle(z)` - returns the phase angle in radians for each element in `z`.

`heaviside(x)` - has the value 0 for $x < 0$, 1 for $x > 0$, and 0.5 for $x = 0$. – This is the unit Step function as you might recall from previous labs.

`ezplot(fun,[min,max])` - a very useful function for quickly plotting both numerical vectors and symbolic functions, if it doesn't look exactly how you want you can specify the plot range in the function call and then by choosing Edit Plot in the tools menu change the axes and many of the plot properties.

`fourier(f,v)` takes the symbolic Fourier transform of function `f` with the output in term of variable `v`.

`ifourier(F,u)` returns the symbolic inverse Fourier Transform with the output in terms of the variable `u`.

`logspace(a,b,n)` – useful for creating vectors spanning a large range with a constant number of points per decade.

`Loglog` – Creates a Log-log scale plot

`semilogx`, `semilogy` – Creates semilog plots

`subplot(m,n,p)` – allows for the arrangement of multiple plots

`subs(S, new)` – this substitutes the values or value of `new` into the symbolic expression `S`, in particular this provides a way to produce an output vector of the values of the symbolic expression `S` evaluated at the input vector `new`'s elements.

The FT of a signal is a complex function in several situations. , hence, when we plot the FT a signal, we plot the absolute value of the FT (Amplitude or Magnitude Response) on a graph, and the phase or angle (Phase Response) on a totally different graph. The phase response represents any time delay which might reside in a signal.

Part I:

First we will get a bit of practice using the fourier transform using MATLAB to do symbolic calculations. Record your results on the results form below.

Declare the symbolic variables w and t .

1) Define a function x as the constant value 1 and find the Fourier transform of this function and record your result.

2) Take the Fourier transform of a cosine with a radian frequency of 1 and record your result.

3) Now you will create a program to take the Fourier transform of a single pulse. First open a blank M-File the easiest way is to right click in the current folder window. Give the function a name and save it. In the M-File define t and w as symbolic variables and using the Heaviside functions create an expression defining a pulse of unit height and width centered on $t=0$. Use `ezplot` to plot this input function over the interval -1 to 1. When in the M-file edit window you can check your program by choosing Save File and run from the debug menu. Be sure that you have properly defined the pulse before proceeding. Next in the M-file take the Fourier transform of this function and have it printed to the command window and use `ezplot` to plot this function over the interval -30 to 30. Use the subplot command before each use of `ezplot` so that your final M-file will plot first the pulse and then its Fourier transform. Turn in both a printout of your completed M-File and the plot output showing the two graphs. Later go back and manually rewrite the symbolic Fourier Transform result that was output in terms of sin function and verify that you get the expected and more familiar expression for this result.

4) Use Matlab to compute the Fourier transform of the impulse response of the single pole RC filter. You will here create an M-file that will compute the transfer function from the impulse response and then produce a two part Bode plot of the magnitude and phase of the filters frequency response. Here roughly are the steps to follow, use the help file and the error statement that will inevitably show up in the command window to debug your program.

- a. Declare the variables t and w as symbolic
 - b. Enter an expression for the impulse response h with a time constant that corresponds to 66.3 kHz
 - c. Compute H , take the symbolic Fourier Transform of the impulse response (this is the frequency response)
 - d. Create a logarithmically spaced vector x of 100 points between 10^3 rad/sec and 10^7 rad/s
 - e. Create a vector that consists of the Magnitude values of H evaluated at the x points
 - f. Create another vector consisting of the phase angle of H also evaluated over x and in units of degrees.
 - g. Make a log-log subplot of the magnitude vs frequency
 - h. Make a semilog subplot of the phase vs. log frequency.
- This is the standard Bode plot of magnitude and phase.

Turn in both a printout of the M-file that you wrote to accomplish the above and your Bode plots.

5) Now repeat the above exercise to create the Bode plots of the frequency response for a 2nd order Butterworth filter with a -3 db corner frequency also of 66.3 KHz, this will be the same frequency used in the hardware lab to follow.

Part II:

Here we will see how to use a powerful set of MATLAB commands to design a filter like that you will build and measure in the next week's lab and to easily plot its step response, impulse response and the Bode plot of its transfer function. We will learn how a linear system can be defined in MATLAB by defining the numerator and denominator of its transfer function.

Here we will be using the following new commands:

Buttap
lp2lp
step
impulse
bode

as well as:

subplot
figure

to produce an M-file that will design a 2nd order Butterworth filter with a 1 rad/sec bandwidth, plot its various response functions, then scale the response to the frequency of the filter you will measure next week and to plot its response functions as well. Be sure to use the MATLAB help functions to look up the correct syntax and use of the various commands. As you type them in the M-File editing window highlights and underlines will indicate syntax problems to make debugging easier.

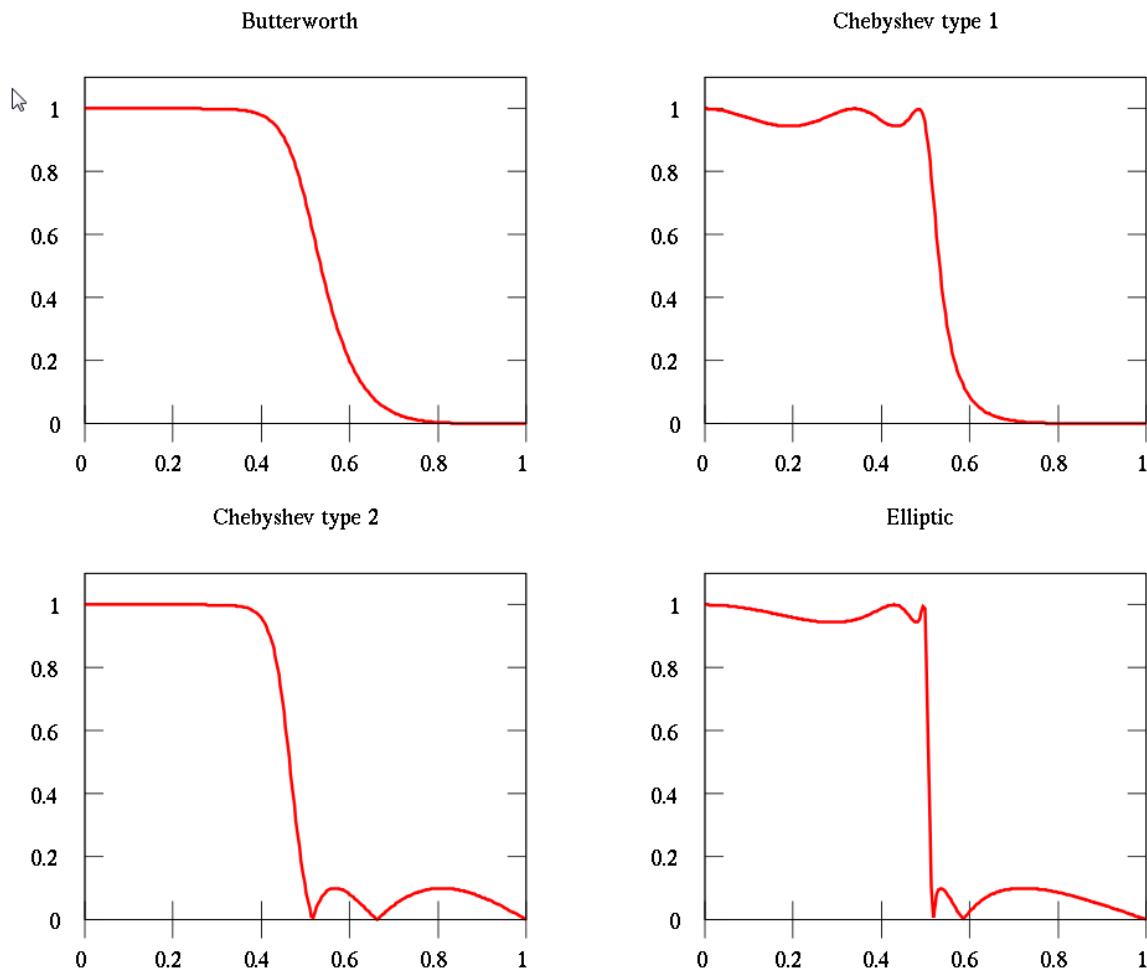
MATLAB also has other specialized functions to produce designs for other common filter types:

- besself Bessel Filters have a maximally flat (smooth) phase response which is good for keeping pulses looking like pulses (attention to those in the digital realm). These filters are proof that "phase matters".
- butter Butterworth Filters have the "flattest" (smoothest) possible amplitude response within their pass and stop bands.
- cheby1 Chebyshev filters of type 1 have a steeper rolloff than a Butterworth filter but with the additional cost of more pass band ripple.

cheby2 Chebyshev filters of type 2 have a steeper rolloff than a Butterworth filter but with the additional cost of more stop band ripple.

Ellip Elliptic filters, also known as Cauer filters, have equalized amounts of ripple in both the pass band and the stop band. These have the fastest cutoff for a given filter order.

Here is a picture of the response of some of the 5th order versions of these filters that makes clearer their different response characteristics and their tradeoffs.



Now to the actual lab procedure:

- 1) You will be creating an M-file that will perform all of the following steps and plots. Create an M-file and give it a name.
- 2) Using the command `butter` generate and print the zeros, poles and gain of a second order Butterworth filter.

3) Given these values define the transfer function that defines this response. The numerator of the transfer function will be defined by the zeros (z) and the denominator will be defined by the poles (p). Write two expressions for the vectors `num[...]` and `den [...]` that define these values.

Here is a bit more explanation on this method of defining a linear system:

Transfer functions are defined in MATLAB by storing the coefficients of the numerator and the denominator in vectors. Given a continuous-time transfer function

$$H(s) = \frac{B(s)}{A(s)}$$

where $B(s) = b_M s^M + b_{M-1} s^{M-1} + \dots + b_0$ and $A(s) = s^N + a_{N-1} s^{N-1} + \dots + a_0$. Store the coefficients of $B(s)$ and $A(s)$ in the vectors `num = [b_M b_{M-1} ... b_0]` and `den = [1 a_{N-1} ... a_0]`. In this text, the names of the vectors are generally chosen to be `num` and `den`, but any other name could be used. For example,

$$H(s) = \frac{2s + 3}{s^3 + 4s^2 + 5}$$

is defined by

```
num = [2 3];
den = [1 4 0 5];
```

Note that all coefficients must be included in the vector, even zero coefficients.

A transfer function may also be defined in terms of its zeros, poles and gain:

$$H(s) = \frac{k(s - z_1)(s - z_2) \dots (s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_n)}$$

To find the zeros, poles and gain of a transfer function from the vectors `num` and `den` which contain the coefficients of the numerator and denominator polynomials, type

```
[z,p,k] = tf2zp(num,den)
```

The zeros are stored in `z`, the poles are stored in `p`, and the gain is stored in `k`. To find the numerator and denominator polynomials from `z`, `p`, and `k`, type

```
[num,den] = zp2tf(z,p,k)
```

4) Using program command statements of the form

Subplot (X,Y,Z), step (num,den)

create a figure with the step, impulse and bode plots for the response for the 2nd order Butterworth filter with the plots arranged in one column and three rows. Stretch them out to fill the page vertically.

5) Use the command “figure” to create another figure for your next set of plots.

6) In the bad old days when computing power was expensive and pencils were cheap, filters would be all be first designed from formulas to be low pass filters with a 1 rad/sec cutoff frequency. You would file these for when you needed a similar filter. Another set of formulas were then used to transform these filters into high pass or band pass filters at different frequencies. This practice has persisted into MATLAB which has specific functions to scale the computed filters to the desired frequency and type.

Use the lp2lp (low pass to low pass) function to scale your filter to a pass band frequency of 100KHz.

7) Finally in the new figure create a 3 row by one column matrix of the following graphs”

Step response, impulse response, and Bode plot.

Turn in the Printed values of your poles and zeros for the original design and the transfer function of the scaled filter and the two plots of the three responses.

Check to make sure that your work is correct. Make sure that qualitatively the impulse response looks like the derivative of the step response at each point. Make sure that your two Bode plots have the correct -3dB frequencies.

Turn in printouts of your poles and zeros the normalized (1 rad/sec) and scaled filters, their respective plots and the single m-file that generates all of this, which when you think about it is remarkably compact give the complexity of the underlying calculations being performed.

1.1 Lab Results:

Answers Lab6 #=__ **Name:**_____**Section:**_____

Compare the results of Parts I, II, and III. Unnormalize the Matlab and Simulink results for the cutoff frequency to compare with your hardware results. Find the effect of using a sine wave input of amplitude $3 + (0.1 \text{ times } \#)$ amplitude, not 1.

Unnormalized RC filter plots

Matlab plots $|H(w)|$

angle $H(w)$

Simulink plots $|H(w)|$

angle $H(w)$

Hardware plots $|H(w)|$

angle $H(w)$

Unnormalized 2nd order Butterworth plots

Matlab plots $|H(w)|$

angle $H(w)$

Simulink plots $|H(w)|$

angle $H(w)$

Hardware plots $|H(w)|$

angle $H(w)$