SIGNALS AND SYSTEMS LABORATORY 4:
Polynomials, Laplace Transforms and Analog Filters in MATLAB

INTRODUCTION

Laplace transform pairs are very useful tools for solving ordinary differential equations. Most applications involve signals that are *exponential* in the time domain and *rational* in the frequency domain. MATLAB provides tools for dealing with this class of signals. Our goal in this lab is to get acquainted with these tools and develop some familiarity with the classical Butterworth and Chebychev lowpass and bandpass filters.

POLYNOMIALS

The rational functions we will study in the frequency domain will always be a ratio of polynomials, so it is important to be able understand how MATLAB deals with polynomials. Some of these functions will be reviewed in this Lab, but it will be up to you to learn how to use them. Using the MATLAB 'help' facility, or a MATLAB manual, study the functions 'roots', 'polyval', and 'conv'. Then try these experiments:

Finding the roots of a polynomial

```
»a=[1 10 35 50 24];
»r=roots(a)
r =
 -4.0000
 -3.0000
 -2.0000
 -1.0000
```

The row vector a contains the coefficients of the polynomial

(1) $$A(s) = s^4 + 10s^3 + 35s^2 + 50s + 24 = (s+1)(s+2)(s+3)(s+4) .$$

The roots function returns the zeros of this polynomial. This function is not bulletproof, however. It, along with any root finder, will have some difficulty when roots are repeated several times. (We will demonstrate this later.) The roots function will return complex values when appropriate:

```
»roots([1 0 1])
ans =
 0 + 1.0000i
 0 - 1.0000i
```

In other words $s^2 + 1 = (s + j)(s - j)$, where j is the square root of -1.

The 'roots' function may have trouble when there is a root of very large multiplicity. To see this, try the following.

```
»a=poly(eye(3)); % constructs the polynomial coefficient of (s-1)^3
»roots(a)
ans =
 1.0000
 1.0000 + 0.0000i
 1.0000 - 0.0000i

»a=poly(eye(10)); % constructs the polynomial coefficients of (s-1)^10
»roots(a)
ans =
1.0474
1.0376 + 0.0284i
1.0376 - 0.0284i
1.0130 + 0.0445i
1.0130 - 0.0445i
0.9846 + 0.0428i
0.9846 - 0.0428i
0.9633 + 0.0256i
0.9633 - 0.0256i
0.9555
```

The result looks good for a third order root, but is quite bad for a 10th order root. Use ‘help’ and ‘type’ to find out how poly(eye(3)) constructs the polynomial $(s-1)^3$.

Multiplying Polynomials

The ‘conv’ function in MATLAB is designed to convolve time sequences, the basic operation of a discrete time filter. But it can also be used to multiply polynomials, since the coefficients of *C*(*s*)=*A*(*s*)*B*(*s*) are the convolution of the coefficients of *A* and the coefficients of *B*. For example:

```
»a=[1 2 1];b=[1 4 3];
»c=conv(a,b)
c =
 1 6 12 10 3
```

In other words,

(2) $$(s^2+2s+1)(s^2+4s+3)=s^4+6s^3+12s^2+10s+3\,.$$

In MATLAB, try roots(a), roots(b), and roots(c). What happens?

Adding Polynomials

If a and b are row vectors representing polynomials, and *C*(*s*)=*A*(*s*)+*B*(*s*) is the sum of polynomials, then it is tempting to think that in MATLAB we need only write c=a+b. But this will work only when a and b have the same length. MATLAB will generate an error message when they have different lengths. There needs to be a *left justification* before the addition. The following homebrew function ‘polyadd.m’, found on the web page under ‘Functions for Lab 4’, could be used to add polynomials:

```
function z=polyadd(x,y)
% z=polyadd(x,y)
% for finding the coefficient vector for the sum
% of polynomials: z(s)=x(s)+y(s)
        m=length(x);n=length(y);
        if m>=n
                z=x+[zeros(1,m-n),y];
        else
                z=y+[zeros(1,n-m),x];
        end
```

Using this function and the function ‘conv’ one can now do polynomial algebra numerically. To construct *D*(*s*)=*A*(*s*)*B*(*s*)+3*C*(*s*), for example, we could write

```
»d=polyadd(conv(a,b),3*c);
```

Evaluating Polynomials

When you need to compute the value of a polynomial at some point, you can use the built-in MATLAB function ‘polyval’. The evaluation can be done for single numbers or for whole arrays. For example, to evaluate

$A(s) = s^2 + 2s + 1$ at s = 1,2, and 3, type

```
»a=[1 2 1];
»polyval(a,[1:3])
ans =
 4 9 16
```

to produce the vector of values $A(1)$ = 4, $A(2)$ = 9, and $A(3)$ = 16. The variables need not be real. For example, they can be complex:

```
»a=[1 0 1];
»z=sqrt(-1);
»polyval(a,[z,z+1])
ans =
 0 1.0000 + 2.0000i
```

IMPULSE RESPONSE

The impulse response, $h(t)$, of an LTI system is its response to an impulse function, $\delta(t)$. This is illustrated as follows:

$$\begin{tikzcd}\delta(t) \arrow[r] & \boxed{\boldsymbol{H}} \arrow[r] & h(t)\end{tikzcd}$$

The response of the system $\boldsymbol{H}$ to a signal e^{st} is $H(s)e^{st}$, where the complex impedance, or transfer function, $H(s)$ is the Laplace transform of $h(t)$. The response to $e^{j\omega t}$ is the complex frequency response $H(j\omega)$, which is the Fourier transform of $h(t)$ and also the complex impedance evaluated at $s = j\omega$.

LAPLACE TRANSFORMS

The Laplace transform provides an *s-domain* or *frequency domain* version of a time signal. Consider the common Laplace transform pair

(3) $$h(t) = e^{at}u(t) \quad \xleftrightarrow{Laplace} \quad H(s) = \frac{1}{s-a}$$

where $u(t)$ is the unit step function. The time function $h(t)$ and the frequency function $H(s)$ are alternate ways of describing the same signal. In the time domain, $h(t)$ is exponential. In the frequency domain, $H(s)$ is rational. The choice of the letter h for the above signal is commonly used for filter functions. In the time domain, $h(t)$ is the *impulse response function* of the filter. In the frequency domain, $H(s)$ is the *transfer function* of the filter. If we set $s = j\omega$, then we get the *complex frequency response function* $H(j\omega)$.

A rational function is, by definition, the ratio of two *polynomials*.

(4) $$H(s) = \frac{B(s)}{A(s)}, \text{ where}$$

(5) $$B(s) = b_1 s^{m-1} + b_2 s^{m-2} + \ldots + b_{m-1}s + b_m \text{ and } A(s) = a_1 s^{n-1} + a_2 s^{n-2} + \ldots + a_{n-1}s + a_n\,.$$

This method of numbering the coefficients is not standard in mathematics, but it is standard for MATLAB. If one constructs a row vector

$$b = [b_1 \;\; b_2 \;\; \ldots \;\; b_m]$$

in MATLAB, and uses it for polynomial coefficients, then the polynomial *B*(*s*) is what MATLAB thinks you are talking about. The first element is the coefficient of the highest power of *s*, and this power is the size of the vector minus one. Thus the vectors [0 1 2] and [1 2] both represent the polynomial s+2. Leading zeros have no effect, but trailing zeros change things. The vector [1 2 0] represents the polynomial s(s+2), but the vector [1 2] represents s+2. It is a good idea to trim leading zeros since they can cause trouble when used with some of the MATLAB tools.

MATLAB provides a variety of tools for resolving Laplace transform pairs when *H*(*s*) is rational. We begin with the polynomial tools.

Rational Functions

The rational function *H*(*s*)=*B*(*s*)/*A*(*s*) requires polynomials to describe the numerator and denominator. Therefore we need two polynomial coefficient row vectors for a parameterization. The denominator should be *normalized* in the sense that the leading coefficient should be one. After all, if both *B*(*s*) and *A*(*s*) are multiplied by the same constant, *H*(*s*) will not change. Thus we can force the coefficient of the highest power in the denominator polynomial to be one. For example, the rational function

(6) $$H(s) = \frac{s^2 + 1}{4s^3 + 4s^2 + 2s + 1}$$

has normalized representation b=[.25, 0, .25], and a=[1, 1, .5, .25]. Using the MATLAB 'help' facility, study the MATLAB functions 'residue', 'freqs', 'bode', 'nyquist', and 'rlocus'. We will use only the first two of these, but they all deal with rational *H*(*s*) in some way.

Evaluating rational functions, and the frequency response

To evaluate *H*(*s*), we evaluate the numerator and denominator and then divide. For vectors, the MATLAB element-by-element division operator './' can be used. Suppose we want to evaluate *H*(*s*)=*B*(*s*)/*A*(*s*) on the frequency range zero to fmax. The following will do it:

```
»fmax=1000;
»wmax=2*pi*fmax;
»w=[0:wmax/1000:wmax];
»H=polyval(b,j*w)./polyval(a,j*w);
```

There are some important things to recognize here. First, we must evaluate H at $s=j\omega$. By this, we are converting frequency values to angular frequency values by using $\omega = 2\pi f$. In the above sequence of commands, a, b, and j, have not been defined. The vectors a and b represent *H*(*s*) and have been previously constructed, but unless you have defined them to be something else, the symbols 'i' and 'j' will be assumed to be the square root of -1. The vector H will contain all the evaluations of $H(j\omega)$ (1001 of them), and will be complex. The functions 'abs' and 'angle' can be used to put things into polar form. The functions 'real' and 'imag' extract the real and imaginary parts. Actually, there is an easier way to do the evaluation, once the vector w is constructed. The MATLAB statement

```
»H=freqs(b,a,w);
```

does the same thing as the statement above using 'polyval', except it uses the built-in MATLAB function 'freqs'. This brings up a final comment. In MATLAB, the functions that need numerator and denominator polynomials follow the convention that the *numerator comes first*. If we typed 'freqs(a,b,w)' we would get values of $1/H(s)$. Keep this in mind when things don't work.

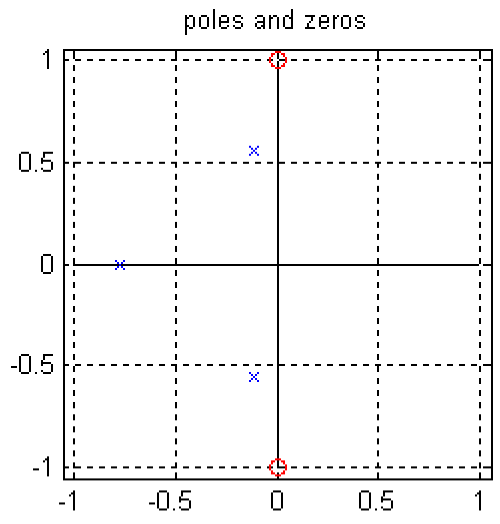
The poles and zeros of $H(s)$ *and pole-zero plots*

The *poles* of $H(s)$ are the roots of the denominator polynomial $A(s)$. At a pole, H becomes infinite. The *zeros* of $H(s)$ are the roots of the numerator polynomial $B(s)$. At a zero, H is zero. A pole-zero plot of $H(s)$ simply places the poles (using the symbol 'x') and the zeros (using the symbol 'o') on the complex plane. This plot reveals a lot about the Laplace transform pair $h(t) \leftrightarrow H(s)$, after you know what to look for. Download the m-file 'pzd.m' from the web page under 'Functions for Lab 4'. Use 'help' and 'type' to see what the file does.

Note that we are using the MATLAB convention that the numerator comes first. For example, try using 'pzd.m' to graph the poles and zeros of the transfer function in equation (6). Type

```
»a=[1 1 .5 .25];b=[.25 0 .25];
»pzd(b,a)
```

The result is shown below. You may think of this as the *complex s-plane*. The horizontal axis is the real axis (σ), and the vertical axis is the imaginary axis ($j\omega$).



Partial Fraction Expansion

If $B(s)$ has degree less than that of $A(s)$, and if $A(s)$ does not have repeated roots, then the Laplace transform pair for $H(s) = B(s)$ /A(s) is

(7) $$h(t) = \sum_{k=1}^{n} \gamma[k] e^{\alpha[k]t} u(t) \quad \xleftrightarrow{Laplace} \quad H(s) = \sum_{k=1}^{n} \frac{\gamma[k]}{s - \alpha[k]}.$$

The right hand side of the above is the *partial fraction expansion* of $H(s)$. The problem is to compute the *poles* (the alpha's) and the *residues* (the gamma's) knowing only the coefficients of the polynomials $A(s)$ and $B(s)$. The poles are the roots of the polynomial $A(s)$. The residues can be computed with pencil and paper via the formula

(8) $$\gamma[k] = B(\alpha[k]) \lim_{s \to \alpha[k]} \frac{(s - \alpha[k])}{A(s)}.$$

But the MATLAB function 'residue' can compute the parameters for you. For example, Let

(9) $$H(s) = \frac{s^2 + 1}{(s+1)(s+2)(s+3)} = \frac{s^2 + 1}{s^3 + 6s^2 + 11s + 6}.$$

To do a partial fraction expansion, type

```
»b=[1 0 1]; % B(s)
»a=[1 6 11 6]; % A(s)
»[gamma,alpha,k]=residue(b,a)
gamma =
 5.0000
 -5.0000
 1.0000
alpha =
 -3.0000
 -2.0000
 -1.0000
k = []
```

This means that $H(s)$ has the partial fraction expansion

$$H(s) = \frac{5}{s+3} + \frac{-5}{s+2} + \frac{1}{s+1}.$$

The parameter k is empty when the degree of $B(s)$ is less than the degree of $A(s)$. Partial fraction expansion in MATLAB will vary, and not be reliable, when there are repeated poles.

A tool for plotting a Laplace Transform Pair

We can put together what we have developed to construct a tool for exhibiting the elements of a Laplace transform pair. The function 'plotLTP.m', found under 'Functions for Lab 4' on the web page, will make a 4-panel plot: a pole-zero diagram, a graph of $h(t)$ from zero to tmax, and graphs of the magnitude and phase of the complex frequency response function $H(j\omega)$ from zero to fmax. Type

```
»help plotLTP

plotLTP(b,a,tmax,fmax)
Plot Laplace transforms
First, plot a pole zero diagram of H(s)=B(s)/A(s).
Then plot the inverse Laplace transform h(t) from 0 to tmax,
and the phase and magnitude of H from 0 to fmax.
Note: any impulsive parts in h(t) will not be plotted.
```

Try 'plotLTP.m' for a pair of polynomials representing b and a.

LOWPASS AND BANDPASS FILTERS

There are ways of choosing the poles and zeros of a filter $H(s)$ so that it acts as a lowpass or bandpass filter. A lowpass filter with cutoff frequency 1 kHz should *pass* all sinusoids whose frequency is less than 1 kHz, and *stop* those with frequencies above 1 kHz. *Stop* means that the output will be zero. There are a few classical filter designs that are used extensively for bandpass filters. These include the Butterworth and Chebychev filters, named for their respective inventors. To get an impression of what these filters do, use the tool 'plotLTP.m' (plot Laplace Transform Pair). Use the MATLAB 'help' command to investigate the functions 'butter', 'cheby1', and 'cheby2'. Then build some filters and display the results using 'plotLTP.m'. To construct a Butterworth lowpass filter with 10 poles, whose cutoff frequency is 1 Hz, type the following commands.

```
»[b,a]=butter(10,2*pi*1,'s');
»plotLTP(b,a,10,4)
```

Note that the function 'butter' asks for radian frequency while 'plotLTP.m' uses actual frequency. This explains the appearance of 2π in the first line. The string 's' will request an analog (rather than digital) filter. You may have to play with the parameter 'tmax' to get a useful display of the impulse response. To construct a bandpass filter whose pass band is 10 Hz to 20 Hz, use the following:

```
»[b,a]=butter(4,2*pi*[10,20],'s');
»plotLTP(b,a,1,40)
```

This example shows the utility of transfer function zeros on the imaginary axis, in this case at $\omega = 0$. Whenever this happens, the frequency response function goes to zero. Thus the zeros of bandpass filters occur in the *stop bands*. Since our second example is a bandpass filter, the zeros at frequency zero force the zero-frequency or DC response to be zero.

Assignment:

1. How does poly(eye(3)) construct the polynomial $(s-1)^3$?

2. For each of the following functions, find a partial fraction expansion, and the inverse Laplace transform. Then use the function 'plotLTP.m' to get graphical information. (Make a hardcopy only after you determine what tmax and fmax ought to be so that $h(t)$ and $H(j\omega)$ use most of the graph, but are not truncated.)

   (1) $H(s) = \dfrac{s(2s+1)}{s^3 + (3/2)s^2 + (13/16)s + (5/16)}$

   (2) $G(s) = \dfrac{(2s+1)}{s^3 + (3/2)s^2 + (13/16)s + (5/16)}$

Verify that $h(t)$ is the time derivative of $g(t)$, from the graphs.

3. Now consider the transfer function $F(s) = G(s/\omega_0)$. Choose an interesting value of ω_0, like $2\pi \times 10^6$, and re-work problem 2. What do you see and why do you see it? Can you see that *normalized* transfer functions can be frequency scaled to suit any application?

4. *Estimating the bandwidth of a lowpass filter from the impulse response*

An ideal lowpass filter with cutoff frequency f_c has a non-causal impulse response of the form

$$h(t) = 2f_c \cdot \mathrm{sinc}(2\pi f_c t)\,.$$

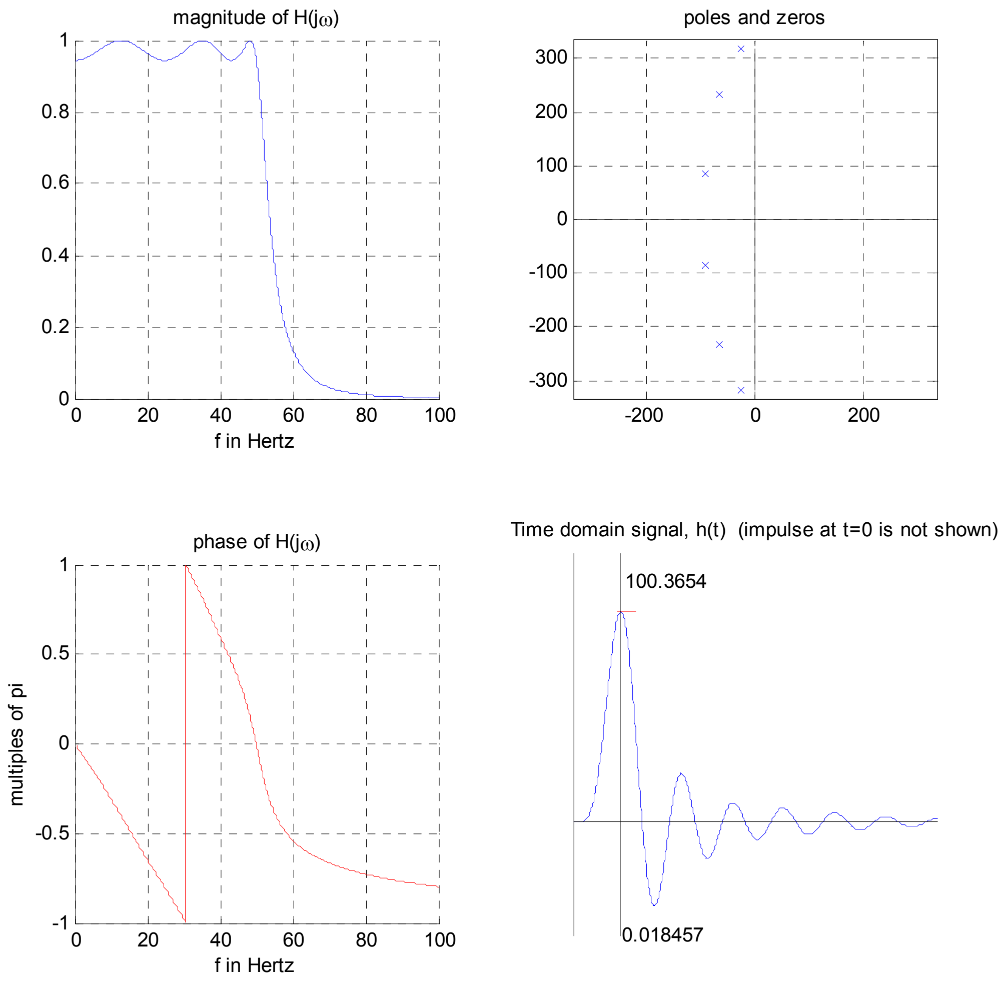
This signal has a central peak whose height is $2f_c$. It will then cross zero $1/(2f_c)$ seconds to the right of the position of the peak. How well do these properties hold for realizable lowpass filters? Using Butterworth lowpass filters, designed with 'butter' and displayed with 'plotLTP.m', make these measurements for filter orders of 2, 4, 6 and cutoff frequencies of 1, 10, and 100 Hz. Compare the actual values with the expected ones. (Using these rules, one can turn the tables and estimate the bandwidth from the impulse response.)

Assignment:

5. The figure below is the output of 'plotLTP.m' for some filter. Find the command that designed the filter. It is either 'butter', 'cheby1', or 'cheby2'. The command will look like

```
[b,a]= <type>(<parameter list>,'s')
```

where <type>. is either 'butter', 'cheby1', or 'cheby2'. Determine which, and find the input parameters. (Use the 'help' facility for these three functions.) Remember, 's' will request the analog filter.



Assignment

6. Consider the transfer function

$$F(s) = -\frac{s - 1/2}{s + 1/2}$$

Use 'plotLTP.m' to compute and plot the *impulse response*, *pole-zero diagram*, and *Bode plots* for $F(s)$. What good is this filter? Now re-do this experiment for $F(-s)$, which incidentally is the *matched filter* for $F(s)$. (You will study these in communications). The function 'plotLTP.m' thinks $F(-s)$ is causal and unstable. How can an unstable system be all-pass? Explain what is going on, and what you would do to recognize $F(-s)$ as a stable, anti-causal transfer function. If you are ambitious you might try to re-write 'plotLTP.m' to account for causal *and* anti-causal transfer functions.