ECE 3793
Matlab Project 3

Spring 2017 Dr. Havlicek

DUE: 04/25/2017, 11:59 PM

What to Turn In:

Make one file that contains your solution for this assignment. It can be an MS WORD file or a PDF file. Make sure to include your name in your turn-in file and add a title at the top of the first page that says "ECE 3793" and "Matlab Project 3." Number the problems and paste in your Matlab code and the resulting command window output. Paste in the figures and graphs and make sure to include answers for the discussion questions.

Submit your solution electronically on Canvas by uploading your turn-in file to

`ECE-3793-001 > Assignments > Matlab 03`

If you are using the Virtual Labs, make sure to save all your files before you log out!

The Assignment:

In this project, you are going to learn some of the ways that Matlab can help you compute and manipulate Laplace transforms.

1. In Matlab Project 2, you saw how the Matlab `residue` function can help you compute partial fraction expansions for inverting Fourier transforms. The `residue` function can be used in exactly the same way to compute partial fraction expansions for inverting Laplace transforms.

   To see how this works, let's use `reside` to work problem 9.22(e) from the textbook. We are given that

   $$X(s) = \frac{s+1}{s^2+5s+6}, \qquad -3 < \mathrm{Re}\{s\} < -2.$$

   Here is a Matlab session log that shows how to use `residue` to compute a partial fraction expansion for $X(s)$:

```
numer = [1 1];
denom = [1 5 6];
[r p k] = residue(numer,denom)

r =
```

```
    2.0000
   -1.0000


p =

   -3.0000
   -2.0000


k =

     []
```

In the output from `residue`, vector `r` gives the residues of the partial fraction expansion and vector `p` gives the poles. Since $X(s)$ is a proper fraction in this case (the order of the numerator is strictly less than the order of the denominator), `residue` returns a null vector for the direct transmission term `k`. From the `[r p k]` values returned by `residue`, we can write down the partial fraction expansion as follows:

$$X(s) = \frac{2}{s+3} - \frac{1}{s+2}. \tag{1}$$

However, to find $x(t)$ we also need to know the ROC for each term on the right-hand side of (1). Since the given ROC of $X(s)$ is a vertical strip, we know that the signal $x(t)$ must be two-sided. Therefore, one of the terms on the right-hand side of (1) must have a ROC that is a right half-plane and the other one must have a ROC that is a left half-plane. Moreover, these two ROC's must intersect to give the ROC of $X(s)$, which is $-3 < \mathrm{Re}\{s\} < -2$.

The first term on the right-hand side of (1) has a pole at $s = -3$, so it's ROC could be $\mathrm{Re}\{s\} > -3$ or $\mathrm{Re}\{s\} < -3$. The second term has a pole at $s = -2$. So it's ROC could be $\mathrm{Re}\{s\} > -2$ or $\mathrm{Re}\{s\} < -2$. Therefore, the ROC's for the individual terms in (1) must be given by

$$X(s) = \underbrace{\frac{2}{s+3}}_{\mathrm{Re}\{s\}>-3} - \underbrace{\frac{1}{s+2}}_{\mathrm{Re}\{s\}<-2}.$$

Using Table 9.2, we then obtain

$$h(t) = 2e^{-3t}u(t) + e^{-2t}u(-t).$$

(a) Use the Matlab `residue` function to work text problem 9.22(d).

(b) Use the Matlab `residue` function to find $h_3(t)$ for $H_3(s)$ as given in text problem 9.10(c). **Note:** this problem has a direct transmission term and also has a repeated pole of multiplicity two at $s = -1$. To handle this correctly, you should type `help residue` at the Matlab command prompt and carefully read the resulting documentation. From the `[r p k]` values returned by `residue`, you should get the following expression for your partial fraction expansion:

$$H_3(s) = 1 - \frac{2}{s+1} + \frac{1}{(s+1)^2}, \qquad \mathrm{Re}\{s\} > -1.$$

2. In Problem 1, we used `residue` to compute partial fraction expansions for rational Laplace transforms where the numerator and denominator where polynomials in s. To do this, we had to give `residue` one input vector containing the coefficients of the numerator polynomial (we called it `numer`) and another input vector containing the coefficients of the denominator polynomial (we called it `denom`).

   This can be a bit inconvenient in some cases. For example, look at text problem 9.7 for a moment. If we wanted to use `residue` to compute a partial fraction expansion in problem 9.7, then we would have to multiply out the denominator. That would be a pain!

   In fact, given a transfer function $H(s)$, you know that we sometimes want to see it as a ratio of two polynomials in s. That makes it easy to see the relationship between $H(s)$ and the system input-output equation. For example, in Problem 1(b) above you had $H_3(s)$ from text problem 9.10(c):

$$H_3(s) = \frac{s^2}{s^2 + 2s + 1} = \frac{Y_3(s)}{X_3(s)}.$$

   Cross multiplying, we get

$$s^2 Y_3(s) + 2sY_3(s) + Y_3(s) = s^2 X_3(s). \tag{2}$$

   The input-output equation then follows immediately if we apply the inverse Laplace transform to both sides of (2):

$$y_3''(t) + 2y_3'(t) + y_3(t) = x_3''(t).$$

   But we also sometimes want to see the numerator and denominator polynomials of $H(s)$ factored into a product of terms of the form $(s - a)$. That makes it easy to see the poles and zeros, which is important because they tell us a lot about the system. If we have to do a partial fraction expansion by hand, then, as you know, we also need the denominator to be in a factored form.

   Thus, given a rational Laplace transform, we generally need to be able to go back and forth between the factored form and the "multiplied out" form of the numerator

and denominator polynomials. Matlab provides *transfer function* and *zero-pole-gain* system models to make this easier.

Here are brief descriptions of the relevant Matlab functions for doing this:

- To create a transfer function system model from vectors containing the coefficients of the numerator and denominator polynomials, use the Matlab statement `H = tf(numer,denom);`. Here, `numer` is a vector containing the numerator polynomial coefficients and `denom` is a vector containing the denominator polynomial coefficients. A transfer function system model is returned in `H`. If you omit the semicolon at the end of the statement, then the transfer function will be printed to the Matlab command window with the numerator and denominator polynomials *multiplied out*.
- To create a zero-pole-gain system model from vectors containing the roots of the numerator and denominator polynomials, use the Matlab statement `H = zpk(z,p,g);`. Here, `z` is a vector containing the zeros of the transfer function (the roots of the numerator polynomial), `p` is a vector containing the poles of the transfer function (the roots of the denominator polynomial), and `g` is a scalar that specifies the gain of the transfer function (see the notes below). A zero-pole-gain system model is returned in `H`. If you omit the semicolon at the end of the statement, then the transfer function will be printed to the Matlab command window with the numerator and denominator polynomials *in factored form*.

Note: the locations of the poles and zeros specify a transfer function $H(s)$ up to a constant gain factor. For example, if it is given that $H(s)$ has a zero at $s = 1$ and a pole at $s = -1$, then we know that

$$H(s) = \frac{s-1}{s+1} \times (\text{a constant}). \tag{3}$$

The constant in (3) is the scalar `g` in the Matlab function call `H = zpk(z,p,g)`. So, if

$$H(s) = \frac{2(s-1)}{s+1},$$

then we need to call `zpk` with `g = 2`. If

$$H(s) = \frac{s-1}{s+1},$$

then we need to call `zpk` with `g = 1`.

Important Note: in the Matlab documentation for the `residue` function, the variable `k` is used for the vector of direct transmission terms, as in

`[r p k] = residue(numer,denom)`.

But in the Matlab documentation for the `zpk` function, the variable `k` is used in a *different* way for the scalar gain, as in

`H = zpk(z,p,k)`.

These two k's are not the same! To avoid confusion, I am using `g` instead of `k` for the scalar gain in calls to the `zpk` function, as in the calls `H = zpk(z,p,g)` shown on the previous page.

The Matlab functions `tf` and `zpk` can also be used to switch back and forth between the "multiplied out" and "factored" forms of the numerator and denominator polynomials. Here is a brief explanation of how to do this:

- If you have a zero-pole-gain system model `H1` that has the numerator and denominator polynomials in factored form, then you can switch to the "multiplied out" form with the Matlab statement `H2 = tf(H1);` This creates a new transfer function system model `H2` that has the numerator and denominator polynomials multiplied out. As before, if you omit the semicolon at the end of the statement then the new transfer function `H2` will be printed to the Matlab command window.
- If you have a transfer function system model `H2` that has the numerator and denominator polynomials in "multiplied out" form, then you can switch to the factored form with the Matlab statement `H3 = zpk(H2);` This creates a new zero-pole-gain system model `H3` that has the numerator and denominator polynomials in factored form. As before, if you omit the semicolon at the end of the statement then the new zero-pole-gain model `H3` will be printed to the Matlab command window.

So far in Problem 2, we have seen that that the Matlab `tf` function can be used to create a "multiplied out" transfer function system model from vectors of numerator and denominator coefficients like this: `H = tf(numer,denom)`. Then we saw that `tf` can also be used to switch from a "factored form" of the numerator and denominator to a "multiplied out" form like this: `H2 = tf(H1)`.

We saw that the Matlab `zpk` function can be used to create a factored model from vectors of zeros and poles (and a scalar gain) like this: `H = zpk(z,p,g)`. Then we saw that `zpk` can also be used to switch from a "multiplied out" form of the numerator and denominator to a factored form like this: `H3 = zpk(H2)`.

Thus, *one* of the useful things that `tf` and `zpk` can do is turn vectors of coefficients or vectors of poles and zeros into system models.

But sometimes we need to "go the other way." That is, sometimes we need to turn system models into vectors of numerator and denominator coefficients or into vectors of poles and zeros. Matlab provides functions for doing this.

If you have a system model `H` (either a transfer function system model that has the numerator and denominator "multiplied out" or a zero-pole gain system model that has the numerator and denominator in factored form), then

- you can obtain vectors of the numerator and denominator polynomial coefficients like this: `[numer,denom] = tfdata(H,'v');`
- you can obtain a vector of the zeros (the roots of the numerator polynomial) like this: `z = zero(H);`
- you can obtain a vector of the poles (the roots of the denominator polynomial) like this: `p = pole(H);`
- alternatively, you can obtain vectors of the poles and zeros with a single call like this: `[p z] = pzmap(H);`

Finally, here are a few more useful Matlab commands:

- You can call `pzmap` in a different way to convert vectors of numerator and denominator coefficients into vectors of poles and zeros like this:
  `[p z] = pzmap(numer,denom);`
- If you call `pzmap` without any left-hand arguments, then it will make a pole-zero plot in the current figure window. To make a pole-zero plot from vectors of numerator and denominator coefficients, you call it like this: `pzmap(numer,denom)`. To make a pole-zero plot from a system model `H`, you call it like this: `pzmap(H)`.
- If you have vectors of numerator and denominator coefficients, then you can print out the transfer function to the Matlab command window *without* making a system model like this: `printsys(numer,denom);`
- If you have a system model `H`, and if the impulse response is right-sided so that the ROC of $H(s)$ is a right half-plane, then you can determine whether or not the system is stable with the Matlab command `isstable(H);` The function `isstable` returns the logical value "1" (true) if the system is stable, i.e., if all the poles of $H(s)$ are in the left half-plane. Otherwise, `isstable` returns the logical value "0" (false). It is important for you to remember that `isstable` **assumes** that the ROC is the right half-plane to the right of the rightmost pole! You cannot use it if the ROC is a left half-plane or a strip.

We have covered a **lot** of new material so far in Problem 2! But now it's finally time to try these things out and see how they really work. Consider a transfer function

$$H(s) = \frac{2(s+1)(s-2)}{(s+2)(s+3)(s-1)}, \qquad \mathrm{Re}\{s\} > 1.$$

From this expression, several things are immediately clear:

- $H(s)$ has zeros at $s = -1$ and $s = 2$.

- $H(s)$ has poles at $s = -2$, $s = -3$, and $s = 1$.
- The gain is given by `g = 2` (this is useful for calling `zpk`).
- The given ROC is $\mathrm{Re}\{s\} > 1$. So, the ROC does *not* include the $j\omega$-axis. This means that the system is *unstable*.
- Because the ROC is right-sided, we know that $h(t)$ is right-sided. Because there is no term e^{-st_0} in the transfer function, we also know that $h(t)$ starts at zero (see the Laplace transform time shifting property in Table 9.1 of the text or in the formula sheet for Test 2). Together, these two facts mean that $h(t) = 0 \; \forall \; t < 0$. Therefore, the system is causal.

Here is some Matlab code that uses this system to illustrate the Matlab functions `tf`, `zpk`, `zero`, `pole`, `tfdata`, `isstable`, `printsys`, and `pzmap` that we have just learned about:

```
%-------------------------------------------------
% P2a
%
% Use the transfer function
%
%          2(s+1)(s-2)
% H(s) = --------------- , Re{s} > 1
%        (s+2)(s+3)(s-1)
%
% to illustrate the Matlab functions tf, zpk, zero,
% pole, tfdata, isstable, printsys, and pzmap.
%
clear;
z = [-1 2];           % zeros (roots of numerator)
p = [-2 -3 1];        % poles (roots of denominator)
g = 2;                % transfer function gain
H1 = zpk(z,p,g)       % zero-pole-gain system model
H2 = tf(H1)           % convert to transfer function
                      % system model
%
% Get the numerator and denominator polynomial coefficients
%
[numer,denom] = tfdata(H1,'v')
%
% Do it again with the other model
%
[numer,denom] = tfdata(H2,'v')
%
```

```
% Compute a transfer function system model
% from the numer and denom coefficients
%
H3 = tf(numer,denom)  % H3 is the same as H1
%
% Convert to zero-pole-gain system model
%
H4 = zpk(H3)          % H4 is the same as H1
%
% Use the numer and denom coefficients to
% print the transfer function to the
% Matlab command window
%
printsys(numer,denom)
%
% Use the pole function to get the poles
% from system models H1 and H2
%
p = pole(H1)
p = pole(H2)          % same result as the line above
%
% Use the zero function to get the zeros
% from system models H1 and H2
%
z = zero(H1)
z = zero(H2)          % same result as the line above
%
% Use pzmap to get the poles and zeros
%
[p z] = pzmap(H1)
[p z] = pzmap(H2)     % same result as the line above
%
% Generate a pole-zero plot in three different ways
%
figure(1); pzmap(H1);
figure(2); pzmap(H2);
figure(3); pzmap(numer,denom);
%
% Use isstable to show that the system is unstable
%
isstable(H1)
%
```

```
% Use residue to compute a partial fraction
% expansion.  Notice that we don't
% have to multiply out the denominator by hand.
%
[r p k] = residue(numer,denom)
%
% Let's see what variables we have:
%
whos
```

(a) Type in the code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file. Don't forget to include the pole-zero plots in your turn-in file.

(b) Use the `[r p k]` values returned by `residue` to find the impulse response $h(t)$.

(c) The system in problem 9.40 of the textbook has input-output relation

$$y'''(t) + 6y''(t) + 11y'(t) + 6y(t) = x(t).$$

Assume that this system is *causal* and use the Laplace transform to find the transfer function $H(s)$ by hand. Write Matlab code similar to P2a above to do the following:

- use `tf` to compute a transfer function system model `H1`;
- use `zpk` to convert the transfer function system model to a zero-pole-gain system model `H2`;
- use `pzmap` to produce a pole-zero plot;
- use `pole` to find the poles;
- use `isstable` to determine if the system is stable;
- use `residue` to compute a partial fraction expansion and find the impulse response $h(t)$. *Hint:* because the system is causal, the ROC must be the right half-plane to the right of the rightmost pole.

3. There is an alternate way to use the Matlab `tf` function to create transfer function system models. For the alternate way, you use the statement `s = tf('s');` to make `s` a transfer function system model. Then, you can use expressions like

```
H = s / (s^2 + 2*s + 10);
```

to create more transfer function models. Similarly, there is an alternate way to use the Matlab `zpk` function to create zero-pole-gain models. For the alternate way, you use the statement `s = zpk('s');` to make `s` a zero-pole-gain system model. Then, you can use expressions like

```
H = -2*s / ((s-2)*(s^2 - 2*s + 2));
```

to create more zero-pole-gain models.

Note: if the numerator and/or denominator have any complex roots, then `zpk` will combine them into quadratic factors with real coefficients wherever possible. This means that the expression for $H(s)$ that is printed to the Matlab command window will not be in a fully factored form. However, you can still use `pole` and `zero` or `pzmap` to find all of the roots of the numerator and denominator.

Here is some example Matlab code that uses this alternate way to call `zpk` in order to find the poles and zeros of the transfer function $H(s)$ in text problem 9.38 and generate a pole-zero plot:

```
%-------------------------------------------------
% P3a
%
% Use zpk, pole, zero, and pzmap to find the poles
% and zeros and generate a pole-zero plot for the
% Laplace transform H(s) in text problem 9.38.
%
clear;
s = zpk('s');
H = 1 / ((s^2 - s + 1)*(s^2 + 2*s + 1))
H2 = tf(H)
p = pole(H)
z = zero(H)
pzmap(H);
```

And here is example Matlab code that uses the alternative way to call `tf` in order to make a pole-zero plot and find the partial fraction expansion for the Laplace transform $X(s)$ in text problem 9.9:

```
%-------------------------------------------------
% P3b
%
% - Use tf to generate a pole-zero plot for the
%   Laplace transform X(s) in text problem 9.9.
% = Use tfdata to get the numerator and denominator
%   polynomial coefficient vectors.
% - Use residue to compute the partial fraction
%   expansion.
%
clear;
s = tf('s');
```

```
X = 2*(s+2)/(s^2 + 7*s + 12)
X2 = zpk(X)
pzmap(X);
[numer,denom] = tfdata(X,'v');
[r p k] = residue(numer,denom)
```

(a) Type in the example code P3a above and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(b) Type in the example code P3b above and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(c) Write similar Matlab code to make a pole-zero plot and use it to work text problem 9.7.

4. If `H` is a system model, then:

   - the Matlab statement `impulse(H)` will plot the system impulse response assuming that the ROC of $H(s)$ is right-sided.
   - the Matlab statement `step(H)` will plot the system step response, assuming that the ROC of $H(s)$ is right-sided.
   - provided that the system is both causal and stable, the Matlab statement `bode(H)` will generate a log-log plot of the frequency response magnitude in decibels (dB) and a semi-log plot of the frequency response phase in degrees.

   Consider the following Matlab code, which generates a pole-zero plot for the transfer function $H(s)$ in text problem 9.33 and plots the impulse response, step response, and frequency response:

```
%-------------------------------------------------------------
% P4a
%
% For the transfer function H(s) in text problem 9.33,
% - make a pole-zero plot
% - plot the impulse response
% - plot the step response
% - plot the frequency response
%
clear;
s = tf('s');
H = (s+1)/(s^2 + 2*s + 2)
[p z] = pzmap(H)    % print poles and zeros to the cmd window
figure(1); pzmap(H);
figure(2); impulse(H);
```

```
figure(3); step(H);
figure(4); bode(H);
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(b) Modify the code to generate similar plots for the transfer function $H_2(s)$ in text problem 9.10(b).

5. If `F` and `G` are system models (transfer function and/or zero-pole-gain system models), then:

   - the Matlab statement `H = series(F,G)` will create a new system model `H` by connecting systems `F` and `G` in series.
   - the Matlab statement `H = parallel(F,G)` will create a new system model `H` by connecting systems `F` and `G` in parallel.
   - the Matlab statement `H = feedback(F,G)` will create a new system model `H` by connecting the systems `F` and `G` in a negative feedback connection. This is the "usual" connection where the summing node has a "+" on the input signal and a "−" on the feedback signal. In case they both have "+", then the call is `H = feedback(F,G,+1)`.

   In the above, if `F` and `G` are both transfer function system models, then `H` will also be a transfer function system model. If either `F` or `G` is a zero-pole-gain model, then `H` will generally be a zero-pole-gain model. Of course, as we already saw in Problems 2 and 3, you can always use `tf` and `zpk` to convert between model types.

   Consider the following Matlab code which solves text problem 9.17. The system block diagram is given in Fig. P9.17 on page 724 of the text. In the figure, note that there are two feedback systems connected in parallel. However, in each feedback system, the feedback path enters the summing note *without* a "−" sign. This means that we will be calling `feedback` with the third argument "+1" as mentioned above.

```
%-----------------------------------------------------------------
% P5a
%
% Work text problem 9.17.
%   - the system block diagram is given in Fig. P9.17 on p. 724.
%   - The figure involves four LTI system blocks.  They will
%       be called H1 through H4, from top to bottom.
%   - The overall transfer function will be called H.
%   - Note: because the feedback paths enter the summing nodes
%       without a "-" sign, we will have to use the "+1" form
%       of the Matlab feedback command.
```

```
%   - The last statement of this code will print out the
%       final transfer function H(s) to the Matlab command
%       window.
%       - From this, the differential equation relating the input
%         x(t) and output y(t) can simply be written down.
%
clear;
s = tf('s');
H1 = 2/s;
numerH2 = -4;  % vector syntax not needed because it is a scalar
denomH2 =  1;  % but to call tf we do need a denominator
H2 = tf(numerH2,denomH2);
H3 = tf(1/s);
numerH4 = -2;
denomH4 =  1;
H4 = tf(numerH4,denomH4);

H = parallel(feedback(H1,H2,+1),feedback(H3,H4,+1))
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(b) Use the command window output from the last Matlab statement to complete the problem by giving the differential equation that relates the system input $x(t)$ to the output $y(t)$. *Hint:* use the fact that $H(s) = Y(s)/X(s)$ and cross multiply.

6. Matlab provides a function `lsim` to simulate an LTI system. If:

   - `H` is a system model,
   - `t` is a vector of times, and
   - `x` is a vector containing samples of the input signal taken at the times in `t`,

   then the statement `y = lsim(H,x,t);` will simulate the system. The vector `y` will be filled with samples of the output signal taken at the times in `t`.

   The following Matlab code simulates a system with transfer function

$$H(s) = \frac{2s^2 + s + 4}{s^3 + 2s^2 + 6s + 5}, \qquad \text{Re}\{s\} > -\frac{1}{2}$$

   and plots the output for two different input signals.

   - The first input signal $x_1(t)$ is a periodic square wave that starts at time $t = 0$ sec and ends at time $t = 25$ sec. The period is 4 sec and the square wave is sampled 100 times per second. The Matlab `gensig` command is used to make this signal.

- The second input signal is $x_2(t) = e^{-t}u(t)$. This signal also starts at $t = 0$ sec, ends at $t = 25$ sec, and is sampled 100 times per second.

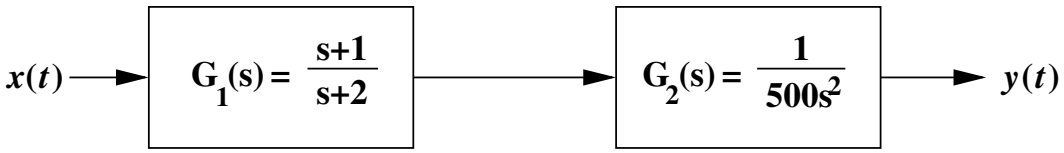
```
%-------------------------------------------------------------
% P6a
%
% Use lsim to simulate an LTI system for two different input
% signals.
%
close all;                    % close any open figure windows
clear;
%
% Make the system model
%
numer = [2 1 4];
denom = [1 2 6 5];
H = tf(numer,denom);
%
% Make input x1 a periodic square wave
%
[x1,t1] = gensig('square',4,25,0.01);
%
% Simulate system and plot the output signal
%
y1 = lsim(H,x1,t1);
figure(1);
plot(t1,y1);
title('Output Signal y_1(t)');
xlabel('Time (seconds)');
ylabel('y_1(t)');
%
% Make input x2 a one-sided decaying exponential
%
t2 = 0:0.01:25;
x2 = exp(-t2);
%
% Simulate system and plot the output signal
%
y2 = lsim(H,x2,t2);
figure(2);
plot(t2,y2);
title('Output Signal y_2(t)');
```

```
xlabel('Time (seconds)');
ylabel('y_2(t)');
```

(a) Type in this code and run it. You can type it in line-by-line at the command prompt or you can create an *m*-file.

(b) Modify the code to simulate the system and plot the output for a periodic pulse input signal $x_3(t)$ that starts at $t = 0$ sec, ends at $t = 25$ sec, has period 4 sec, and is sampled 100 times per second. Use the `gensig` command to make $x_3(t)$.

7. Consider the causal LTI system block diagram shown below:



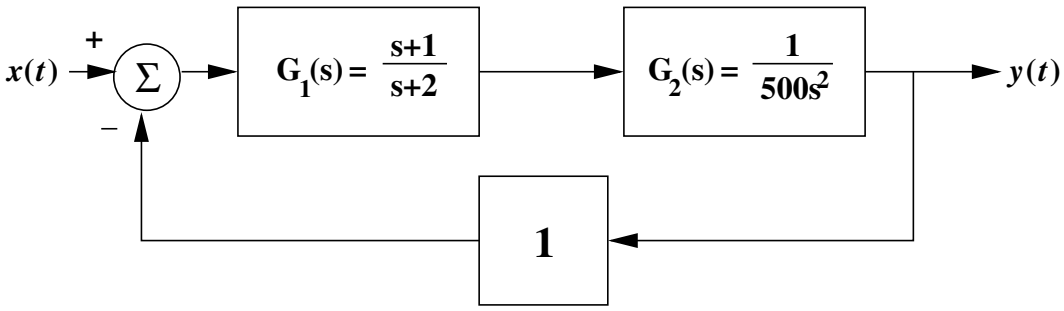
Write Matlab code to do the following:

- use `tf` and `series` to find the overall transfer function $H(s)$.
- use `pole` to find the poles.
- use `pzmap` to generate a pole-zero plot.
- use `isstable` to determine if the system is stable.
- use `impulse` to plot the impulse response.
- use `lsim` to find and plot the output $y(t)$ when the input is given by

$$x(t) = e^{-t} \cos(5\pi t)\, u(t).$$

Run the simulation from $t = 0$ to $t = 5$ sec with a resolution of 1 msec (use `t = 0:0.001:5;` for your vector of sampling times).

8. Consider the causal LTI system block diagram shown below. *Note:* it is the same as the system in Problem 7, but with a negative feedback loop added.



Write Matlab code to do the following:

- use `tf`, `series`, and `feedback` to find the overall transfer function $H(s)$.
- use `pole` to find the poles.
- use `pzmap` to generate a pole-zero plot.
- use `isstable` to determine if the system is stable.
- use `impulse` to plot the impulse response.
- use `bode` to plot the frequency response.
- use `lsim` to find and plot the output $y(t)$ when the input is given by

$$x(t) = e^{-\frac{t}{10000}} \cos\left(\frac{\pi}{500} t\right) u(t).$$

  Run the simulation from $t = 0$ to $t = 20,000$ sec with a resolution of 1 μsec (use `t = 0:0.01:20000;` for your vector of sampling times).

9. The Matlab symbolic math toolbox provides functions `laplace` and `ilaplace` for computing forward and reverse Laplace transforms. It's **important** for you to realize that `laplace` and `ilaplace` implement the **unilateral** transform

$$\mathcal{X}_u(s) = \int_{0^-}^{\infty} x(t)\, e^{-st}\, dt,$$

not the bilateral transform. In particular, `laplace` **assumes** that all signals are zero for $t < 0$. Similarly, for a signal that starts at $t = 0$, the result returned by `ilaplace` will **omit** the unit step function $u(t)$. These important ideas are illustrated in the following Matlab examples:

```
>> syms s t;
>> x1t = exp(-2*t)*heaviside(t);   % x1t and x2t are the SAME as far as
>> x2t = exp(-2*t);                %   laplace is concerned!
>> x3t = exp(-2*(t-3))*heaviside(t-3);   % but x3t is DIFFERENT

>> X1s = laplace(x1t)

X1s =

1/(s + 2)

>> X2s = laplace(x2t)

X2s =

1/(s + 2)                        % the SAME as X1s !
```

```
>> X3s = laplace(x3t)

X3s =

exp(-3*s)/(s + 2)

>> x1t_b = ilaplace(X1s)

x1t_b =
                              % note that the unit step function
exp(-2*t)                     % is OMITTED!

>> x2t_b = ilaplace(X2s)

x2t_b =

exp(-2*t)                     % unit step function OMITTED again

>> x3t_b = ilaplace(X3s)

x3t_b =

heaviside(t - 3)*exp(6 - 2*t)
```

Here are two more examples:

```
>> syms a s t w     % 'a' is a generic symbolic constant.
>> xt = exp(-a*t) * cos(w*t);
>> Xs = laplace(xt)

Xs =

(a + s)/((a + s)^2 + w^2)

>> pretty(Xs)

      a + s
  -------------
         2    2
  (a + s)  + w
```

```
>> x2t = dirac(t);
>> X2s = laplace(x2t,t,s)

X2s =

1
```

Now it's your turn to try `laplace` and `ilaplace`.

(a) Use symbolic math to find the Laplace transform of the signal

$$x(t) = e^{-t}\sin(2t)u(t).$$

(b) Use symbolic math to work text problem 9.21(b).

(c) Use symbolic math to find the impulse response of the causal system $H_1(s)$ given in text problem 9.10(a).

(d) Use symbolic math to find the impulse response of the causal system $H_2(s)$ given in text problem 9.10(b).

(e) Use symbolic math to find the impulse response of the causal system $H_3(s)$ given in text problem 9.10(c).

(f) Use symbolic math to work text problem 9.22(a).