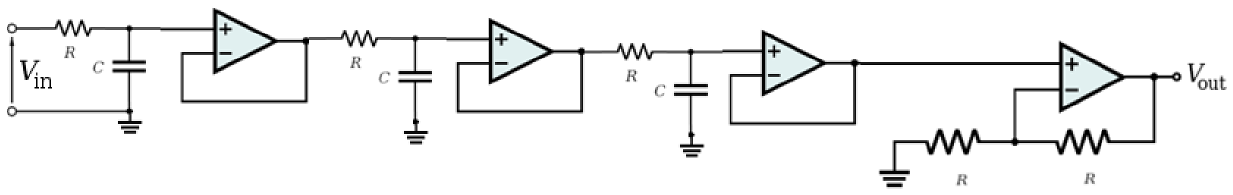
ESE 505 - SPRING 2015 - HOMEWORK #2
DUE 23-Feb-2015

Here are the key concepts we have been working with:

- Representation of dynamic systems using Laplace Transforms, Transfer Functions, and Block Diagrams
- High-level understanding of system response, in terms of properties of transfer function, G(s):
  - Steady-state response, as indicated by G(0).
  - Nature of dynamic response, as indicated by location of poles of G(s) in the complex plane.
- Closed-loop system response and the influence of feedback
  - Closed-loop steady-state response, again found by setting s=0.
  - Variation of closed-loop pole locations with gain, called a "root locus"

In addition to our analytical tools, we need to develop skills using Matlab and Simulink to design control systems.

Please complete the following problems with these ideas in mind. These problems deal with the Black Box system we have been working on in the lab. If you have already done something very similar to what is called for in these problems as part of your lab documentation, you don't have to do it again. You may simply copy your lab work into this assignment and explain briefly that you think your prior work substantially satisfies the intent of the exercise.



1. As we discussed in class, the "Black Box" contains a circuit that is functionally equivalent to the circuit shown above, and we concluded that the corresponding transfer function is:

$$G_P(s) = \frac{V_{out}}{V_{in}} = \frac{2a^3}{(s+a)^3} = \frac{2a^3}{s^3 + 3as^2 + 3a^2s + a^3}$$

   By now, you should feel quite comfortable computing *a* in terms of *R* and *C*. You should also be able to explain why we have the buffers in the circuit and how the non-inverting amplifier works.

   Now we want to use MATLAB and SIMULINK to simulate the response of the Black Box to step inputs.

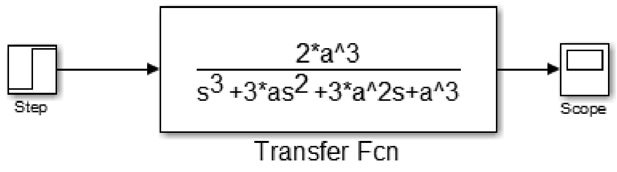
   a. In MATLAB, we can simulate the system with the following commands:

```
a = 21;
num = [2*a^3];
den = [1 3*a 3*a^2 a^3];
Gp = tf(num,den);
[y,t] = step(Gp);
plot(t,y,'-r','LineWidth',2);
grid on; set(gcf,'Color','w');
title('Black Box Step Response');
xlabel('Time [sec]');
ylabel('Response [volts]');
```

The last several lines make the graph look nice. Please be sure you understand how we are specifying the transfer function to MATLAB. ***Submit a graph of the response.***

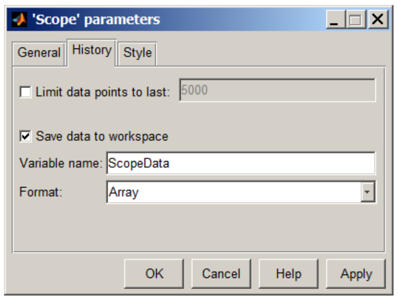
b. We can also build a model of the system in SIMULINK. If you have never used SIMULINK before, you should watch some online tutorial videos, like these (or find your own favorites and post on Piazza):

```
https://www.youtube.com/watch?v=naR2lAnxnZE
https://www.youtube.com/watch?v=PHH4Nk535Jw
```



Submit a screen-shot of the step response generated in Simulink.

c. We often want to use SIMULINK but don't want to use screen capture to show our results. The Scope can export the data to the MATLAB workspace for us. In the Scope, go to Parameters and make the History tab look like this:



☐ Limit data points to last: 5000

☑ Save data to workspace

Variable name: ScopeData

Format: Array

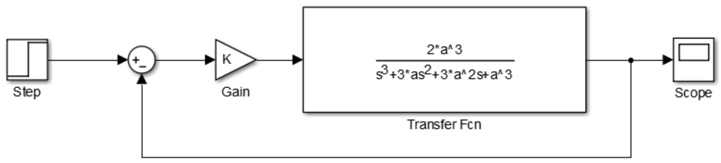
If our SIMULINK model is saved with the filename `BlackBoxTF`, then the following commands can be put in a MATLAB script to generate the plot as in part (a) above:

```
sim('BlackBoxTF');
plot(ScopeData(:,1),ScopeData(:,2),'-r','LineWidth',2);
grid on; set(gcf,'Color','w');
title('Black Box Step Response');
xlabel('Time [sec]');
ylabel('Response [volts]');
```

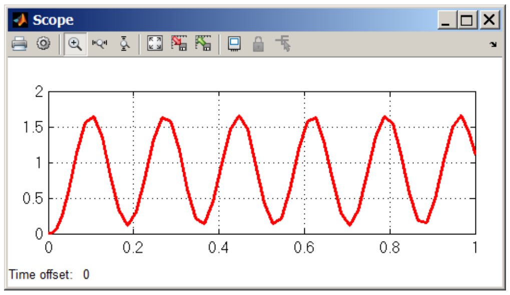
Sometimes, we also want to use measured data or some other input signal as the input to the model, instead of a pure step input or other idealized input from the SIMULINK library. We can use the "From Workspace" block to accomplish this. Google it for more details.

<u>You don't have to submit anything for part (c)</u>.

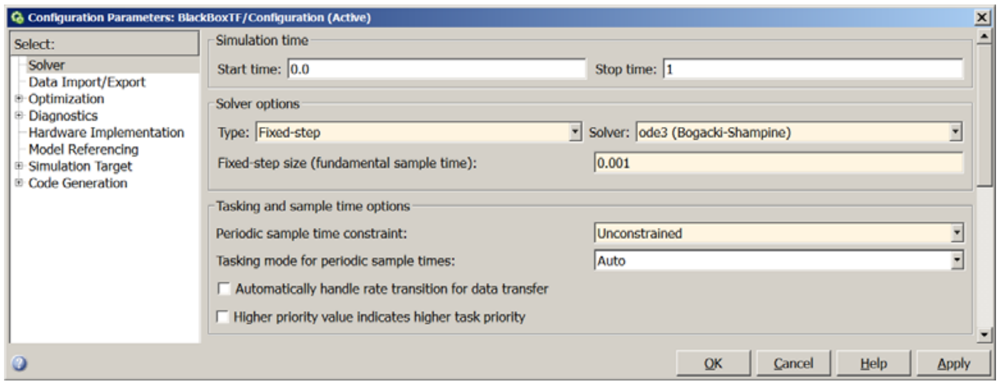
2. Now we want to analyze the closed-loop Black Box system. Start by adding proportional feedback to your SIMULINK model:



With the default numerical integration parameters, you might get something that looks like this when K=4:



The issue is that SIMULINK is taking a very large time step, and the graph looks jagged. We could have fixed this before, but it is important to fix it now. It is very often necessary to use smaller time steps than are used by default. In SIMULINK, go to the "Simulation" menu and choose "Model Configuration Parameters" and change to a fixed-step Solver:



Configuration Parameters: BlackBoxTF/Configuration (Active)

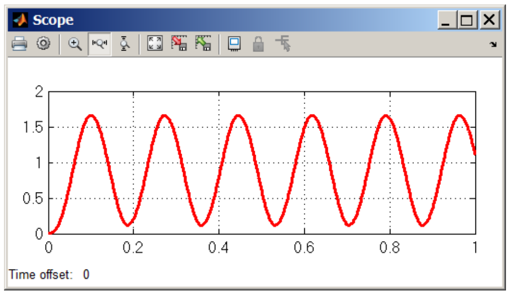
Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation

Simulation time
Start time: 0.0　　Stop time: 1

Solver options
Type: Fixed-step　　Solver: ode3 (Bogacki-Shampine)
Fixed-step size (fundamental sample time): 0.001

Tasking and sample time options
Periodic sample time constraint: Unconstrained
Tasking mode for periodic sample times: Auto
☐ Automatically handle rate transition for data transfer
☐ Higher priority value indicates higher task priority

The numerical simulation results will look much nicer.



Now run the model for K=1, 2, 3, and 4. From the time response, estimate the location in the complex plane of the pole locations for each value of K. (You might want to google "log decrement".). You might want to set up a spreadsheet in which you enter something like the period of the oscillation and the ratio of successive peak amplitudes in the response. And then enter formulas to compute the pole locations. It will probably be difficult to precisely estimate the pole locations when K=1.

Submit a table of values, with the following column headings:

Gain (K)	σ [rps]	ω_d [rps]	ω_n [rps]	ζ [-]

Next, let's use MATLAB to calculate the information in the table from theory. With proportional feedback, we found in class that the closed-loop transfer function is

$$\frac{Y(s)}{Y_d(s)} = \frac{KG_P(s)}{1+KG_P(s)} = \frac{KN_P(s)}{D_P(s)+KN_P(s)}$$

where $N_P(s)$ and $D_P(s)$ are the numerator polynomial and denominator polynomial, respectively, of $G_P(s)$. So, the closed-loop poles come from the roots of the characteristic equation:

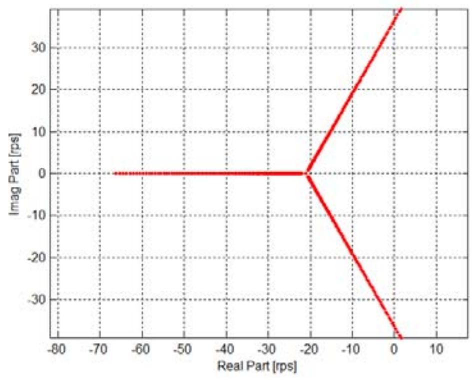
$$D_P(s)+KN_P(s)=0$$

MATLAB can plot the variation of the roots of this equation using the following commands:

```
nump = [2*a^3];
denp = [1 3*a 3*a^2 a^3];
Kvalues = [0.0:0.1:5.0];
rlocus(nump,denp,Kvalues);
axis('equal');  % equal scaling for x and y
```

Note that the "Kvalues" input argument is optional. MATLAB will pick a range of K automatically if you don't specify it. I don't find the default root locus plot to be very attractive, so I often use output arguments on the root locus to get something I like better. You don't have to do this if you are happy with the default.

```
nump = [2*a^3];
denp = [1 3*a 3*a^2 a^3];
Kvalues = [0 5*logspace(-5,0,200)];
[Rmatrix] = rlocus(nump,denp,Kvalues);
figure(1); hold off;
for i=1:length(Kvalues),
    plot(real(Rmatrix(i,:)),imag(Rmatrix(i,:)),'.r','MarkerSize',10);
    hold on;
end;
xlabel('Real Part [rps]');
ylabel('Imag Part [rps]');
grid on; axis('equal');
hold off;
```

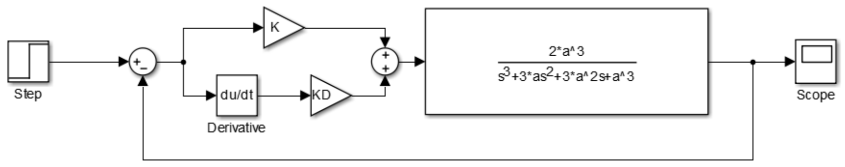


<u>Submit a MATLAB root locus plot. Add points, □, at the pole locations you identified in the previous question for K=1, 2, 3, and 4.</u>

You should be sure you see the connection between the MATLAB root locus and the points you extracted from the simulation. Of course, if you want to add data you identified in the lab, that would be outstanding!

3. Now let's change from proportional feedback to PD, proportional-plus-derivative, and then repeat the process of problem 2, but now consider the changes in the closed-loop pole locations due to variations in the derivative gain, with the proportional gain fixed at the value that yielded neutral closed-loop stability, K=4.

   The SIMULINK part is pretty simple--just modify the feedback to include KD:



   The MATLAB root locus part is a little more tricky. This time, the closed-loop transfer function is given by the following:

$$\frac{Y(s)}{Y_d(s)} = \frac{(K + K_D s)G_P(s)}{1 + KG_P(s)} = \frac{(K + K_D s)N_P(s)}{D_P(s) + (K + K_D s)N_P(s)}$$

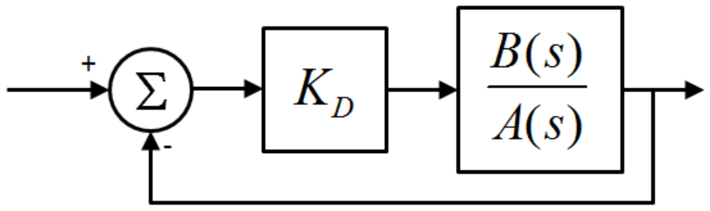
So, the closed-loop poles are the roots of the characteristic equation:

$$D_P(s) + \left(K + K_D s\right) N_P(s) = 0$$

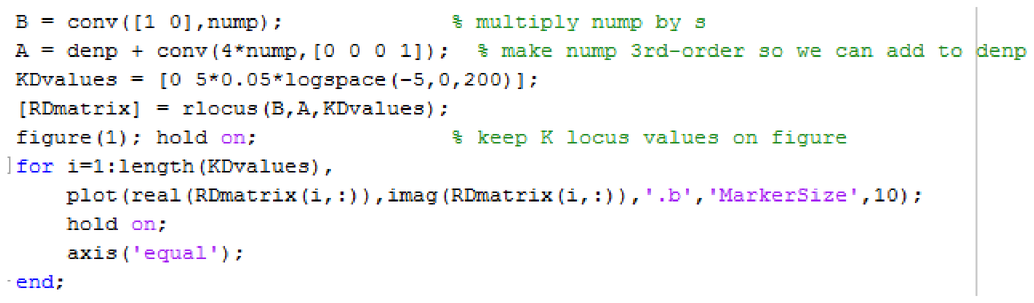
For this problem, we are fixing the proportional gain at K=4, so the characteristic equation is:

$$D_P(s) + 4N_P(s) + K_D s N_P(s) = A(s) + K_D B(s) = 0$$

where $A(s) = D_P(s) + 4N_P(s)$ and $B(s) = sN_P(s)$. This is the same characteristic equation we would get if we had the following block diagram:



The MATLAB "rlocus" command assumes that we have a system of this form, so we need to modify the "num" and "den" arguments to "rlocus" so that it correctly computes the closed-loop pole locations:


```
B = conv([1 0],nump);                   % multiply nump by s
A = denp + conv(4*nump,[0 0 0 1]);   % make nump 3rd-order so we can add to denp
KDvalues = [0 5*0.05*logspace(-5,0,200)];
[RDmatrix] = rlocus(B,A,KDvalues);
figure(1); hold on;                     % keep K locus values on figure
for i=1:length(KDvalues),
    plot(real(RDmatrix(i,:)),imag(RDmatrix(i,:)),'.b','MarkerSize',10);
    hold on;
    axis('equal');
end;
```

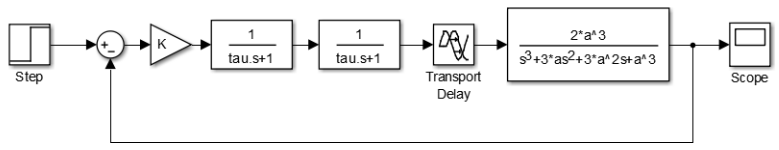
<u>Submit the value of K_D that maximizes the closed-loop damping ratio of the oscillatory poles when K=4.</u>

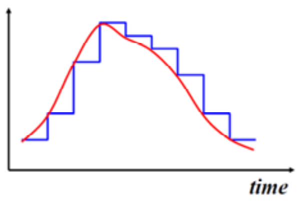
4. Now let's look at the steady tracking. For zero steady-state error, we need $\lim_{s\to 0} \frac{Y(s)}{Y_d(s)} = 1$. To get that result, we need to add integral feedback, $\frac{K_I}{s}$. Do your best to repeat the process of the last 2 questions, this time keeping K=4 and K_D=0.03K, but varying K_I to see the effects on response and on the closed-loop poles.

   <u>Submit the value of K_I that reduces the damping ratio by a factor of 2, compared to K_I=0 (both with K=4 and K_D=0.03K). Where are the other two poles for this value of K_I?</u>

5. Finally, let's go back to purely proportional control do some analysis of what think about what happens when we replace our analog implementation with the Arduino-based digital controller.

   In order to use the PWM output from the Arduino, we introduced two low-pass filters into the feedback loop. We also forced the digital controller to run at a constant "frame rate", updating the control only every 20 ms. We need to include these elements in our SIMULINK model, which now looks something like this:





Our filter used C=1μF and R=3300Ω, so τ=3.3ms. The transport delay, e^{-Ts}, represents the digital processing time. As shown in the figure at right, a digital update time of 0.020 sec creates a transport delay of half that, T=0.010 sec.

With the new elements, the closed-loop poles come from the following characteristic equation[1]:

$$(\tau s+1)^2 D_P(s) + Ke^{-Ts} N_P(s) = 0$$

But we need to have only polynomials in the characteristic equation in order to use MATLAB's rlocus command. So, we can use a Pade Approximation[2] for the time delay, such as the following:

$$e^{-Ts} \approx \frac{-\frac{T}{2}s+1}{\frac{T}{2}s+1}$$

Now the approximate characteristic equation is:

$$(\tau s+1)^2 \left(\frac{T}{2}s+1\right) D_P(s) + K\left(-\frac{T}{2}s+1\right) N_P(s) \approx 0$$

This time, instead of making a pretty plot, let's use the rlocus command, with a second command. With "rlocfind", we can find the gain for any closed-loop pole location simply by clicking on the root locus plot:

```
tau = 0.0033;
T = 0.01;
B = conv([-0.5*T 1],nump);
A = conv(conv([tau 1],[tau 1]),conv([0.5*T 1],denp));
Kvalues = [0 5*logspace(-5,0,200)];
rlocus(B,A,Kvalues); axis('equal')
[Kcrit, Rcrit] = rlocfind(B,A)
```

Submit the values of K that results in neutral stability according to the modified SIMULINK model and according to the root locus.

We have covered a lot of ground in this assignment. Remember that the goal of completing the homework is not to get the "right" answers, but rather to help us solidify our understanding of the key ideas and how they can be applied to our observations in the lab. If you are confused or unsure about something, it is okay to write something like, "I'm not sure about this, because..." in your homework. Of course it is excellent to do additional exploration of the ideas that are not called for here. Better yet, post your questions or new insights on Piazza to share with the whole class!

[1] Please be sure you know where this equation comes from.
[2] `http://en.wikipedia.org/wiki/Pad%C3%A9_approximant`