ESE 406/505 & MEAM 513 - SPRING 2011
HOMEWORK #12
DUE 18-Apr-2012 (Monday, 23-Apr-2012 with late pass, Q#3 ONLY)

1. You might find problems 4.34(c) and 4.35(c) in the textbook worthwhile. You don't have to submit anything for this problem.

   Answers: 4.34 $D_3(z) = \dfrac{4.2z - 3.8}{z - 0.6}$ 4.35 $u(k) = 0.6u(k-1) + 4.2e(k) - 3.8e(k-1)$

2. Submit more evidence that you are making progress on the final project. This part must be submitted separately from the rest of the homework and you may not use a late pass for this question. There will be a separate collection box in class for this problem. You must submit at most 2 pages (or 1 double-sided page). Show a block diagram, some response graph(s), and give a short and technically precise description of what you are doing. At the end, put a big box around what you think are the main difficulties.

3. This problem is an adaptation of problem 8.8 in the text. Read that problem first. Notice that you clearly don't need integral feedback at all to meet the design requirements. It turns out that you can almost exactly meet the specs in the original problem with a purely proportional gain of unity. (You should be sure you can verify that this is true). Let's make the problem more interesting. We will replace the given plant, $G_P(s) = \dfrac{1}{s(s+1)}$, with a 3rd-order plant, $G_P(s) = \dfrac{1}{s(s+1)(s+5)}$.

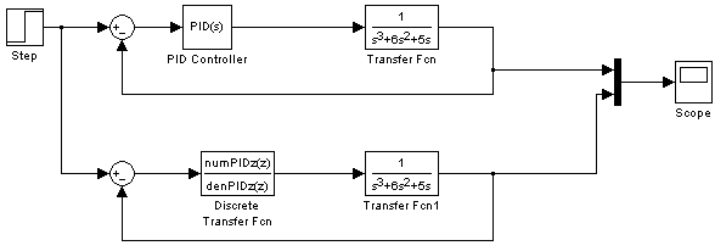
   a. Make a bode plot of $G_P(s)$. Find the frequency at which the phase crosses -180°. Look at the gain of $G_P(s)$ at this frequency to figure out the proportional feedback gain, K_u, required to reach neutral stability, and the expected period, P_u, of the oscillations. Use SIMULINK to confirm that this gain does indeed result in neutral stability. (Answer: K_u=29.85, P_u=2.8sec)

   b. Now use the Ziegler-Nichols "ultimate sensitivity method", summarized in Table 4.3 (remember the typo: should be Kp=0.6Ku), to set the gains of a PID controller. Use SIMULINK to examine the closed-loop response to a step input. (Answer: K_P=17.9, K_D=6.3, K_I=12.77. The closed-loop response is somewhat lightly damped.).

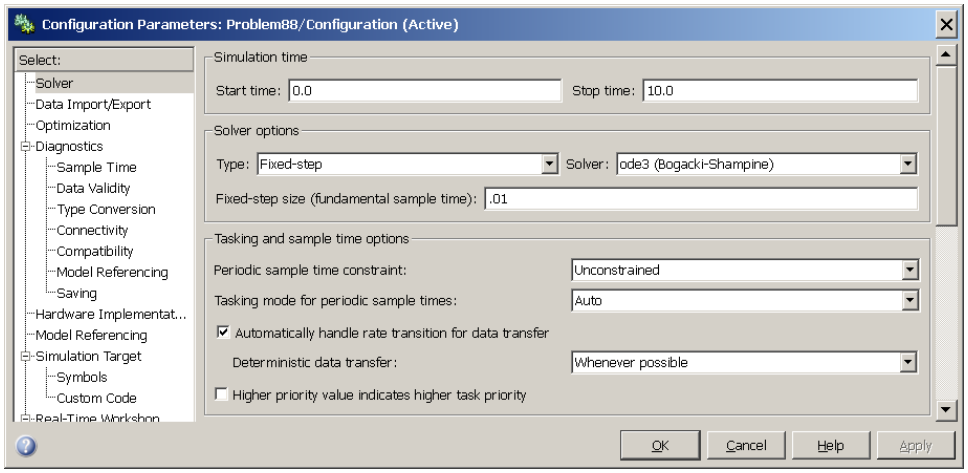
   c. Now make a bode plot of the LOOP transfer function, which is the product of your PID controller and $G_P(s)$. Determine the phase margin. How much pure time delay would have to be added to the loop to make the system unstable? (Answer: 21.9° & 0.22 seconds.) The Ziegler-Nichols compensator is not very good, because the phase is too close to -180° at the crossover frequency. We could do much better if we actually design for good "phase margin", for example by using a lead compensator.

d. Now let's convert to discrete-time. Let's start with a sample time of 0.04 seconds (sampling frequency of 25 Hz, which is much higher frequency than any poles in our system). Use the "Tustin" method to convert the continuous-time PID compensator into a discrete-time compensator (you may use matlab's C2D function to get the discrete-time filter coefficients). What is the discrete-time transfer function of your digital compensator? Use SIMULINK to compare the closed-loop step responses of the total system with both the analog and discrete compensators.

Hints: Use a discrete-time transfer function from the "discrete" library to put your digital PID compensator in your SIMULINK model. You might use a model that
]



In order to get SIMULINK to automatically handle the mix of discrete-time and continuous-time blocks in the same model, you have to choose the "Automatically handle rate transition" option under the Configuration Parameters dialog. You should also be sure to set a fixed time step such that the sample rate is an integral multiple of the time step:



(Answers: $\frac{332.1z^2 - 627.5z + 296.3}{z^2 - 1}$ & discrete-time response very closely matches analog design.)

e. Increase the sample time to 0.2 seconds (very close to the predicted amount of delay that will result in neutral stability) and again compare step responses of the analog

and discrete-time implementations. Remember to compute a new discrete-time version of the compensator, because this conversion depends on sample time. (Answers: $\frac{81.98z^2 - 123z + 46.16}{z^2 - 1}$ & discrete-time response is now unstable.)

f. Now let's try to implement our digital controller (with sample time 0.04 sec) as an S-function of the form we are using in the project. We can re-write the compensator as follows, using "w" for the control and "e" for the error, to avoid notational confusion ("u" is used as the input to the S-function):

$$w[k] = w[k-2] + 332.1e[k] - 627.5e[k-1] + 296.3e[k-2]$$

So, we need to save 2 previous values of the control and 2 previous values of the error. Let's make those our states. In other words, we will have 4 states:

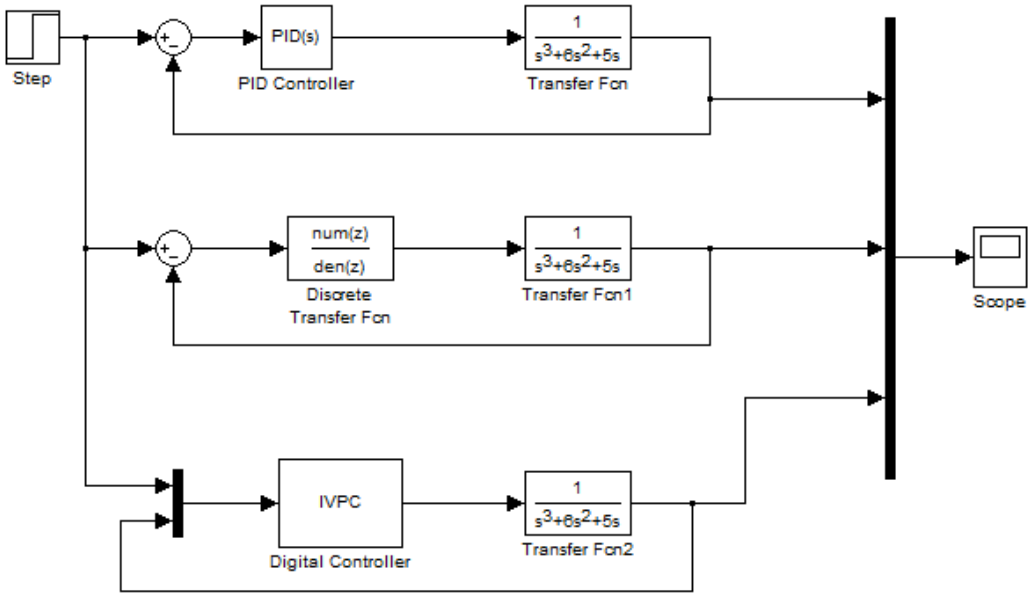
x_1 = old value of e
x_2 = old old value of e
x_3 = old value of w
x_4 = old old value of w

Now, since we pass the desired response into the S function as u_1 and the measured response as u_2, the error at the current iteration is calculated as $e = u_1 - u_2$. The "update equations" for each state can thus be written as follows:

new x_1 = current value of $e = u_1 - u_2$
new x_2 = old value of $e = x_1$
new x_3 = current value of $w = x_4 + 331.2(u_1 - u_2) - 627.5\, x_1 + 296.3 x_2$
new x_4 = old value of $w = x_3$

The output of the S function is just the current value of w. Since SIMULINK calls the digital output routine *BEFORE* it calls the digital input, we have to re-enter the formula that updates x_3 as the output.

All of this has been implemented for you in a modified version of the file "IVPC.m". You should carefully study it to understand what is going on. Copy-and-paste the S-function block from the project SIMULINK model (`P.mdl`) into the model you made above. You should get something that looks like this:



Now run the simulation and verify that you get EXACTLY the same answer with the S-function as you got with the digital transfer function (curves 2 and 3 overlap perfectly).

g. The implementation we used in the previous part was not very efficient. We have only 2 poles in our digital transfer function, yet we had 4 discrete-time states. And we had to think carefully about what we were doing to get the update equations and output equation correct. Our approach had the advantage of being somewhat intuitive or "seat of the pants" (at least once we understand what is going on). But there is an easier and more systematic approach.

We can convert our discrete-time numerator and denominator to a discrete-time state-space implementation using MATLAB's `ss2tf` command. This will give us A, B, C, D matrices

$$\underline{x}[k+1] = A\underline{x}[k] + Be[k]$$
$$w[k] = C\underline{x}[k] + De[k]$$

This can be directly implemented in the S function: we don't have to write-out the update equations individually. We just compute "e" from the 2 inputs when the S function is called and then compute either the updated "x" vector (all at once) or the updated control output, as required.

Update the S function to use this approach and validate that you get the same answer as before answer. Submit the modified IVPC.m code, highlighting the lines that changed from the implementation in the previous question.

h. One final note. You may wonder why we passed both the desired and measured response into the S Function. We could have computed the error using a subtraction node outside the S Function, and then the input would have been "e".

   One reason is that we wanted to more closely mimic what happens in a real digital controller. Generally, there is a sensor on the input and a sensor on the output, and the digital controller gets readings of both of these.

   A second, and more important, reason is that we want to be able to do "feed-forward" control on the user input alone. We might do this with another discrete-time dynamic system, this time using only the user input, rather than the error, as the input to the dynamic system.

   The output of the S function might then be the sum of the output of our dynamic system for feedback and our dynamic system for feed-forward.

   One of the nice things about digital control is that we can relatively easily implement non-linear feed-forward, for example including limits on desired response to prevent excessive errors, which can lead to integrator windup and other problems.

   You don't have to submit anything for this part.