ECE-520 Lab 3

System Identification For One Degree of Freedom Systems Using Matlab's System Identification Toolbox

Overview

In this lab you will be utilizing Matlab's system identification toolbox to try and determine a model for both of your one degree of freedom systems (torsional and rectilinear) You will compare these models with a your continuous time model that after it has been discretized assuming a zero order hold. You will be storing these models for later comparison. In Lab 4 we will utilize the system identification toolbox to determine models for the two degree of freedom systems you have already modeled. Feel free to explore the System Identification Toolbox and tell me any cool stuff you found out. The following is just a very basic introduction.

Background

There are two basic issues we must deal with before we get into how to use the system identification toolbox. The first issue is the difference between the form of the state equations we will get out from the system identification toolbox and the form of the equations we want to have. The second issue is the types of inputs we need in order for the system identification toolbox to work correctly.

The ECP systems have an extra delay in the system, which we must account for. Since we want access to the states (for state variable feedback) we want our state model to be of the form

$$\begin{bmatrix} q_1(k+1) \\ q_2(k+1) \\ u(k) \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & h_1 \\ g_{21} & g_{22} & h_2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} q_1(k) \\ q_2(k) \\ u(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_1(k) \\ q_2(k) \\ u(k-1) \end{bmatrix}$$

The system identification program will be told that we have a delay, so it will not care about the last state (it just passes though). Hence we will only give the system identification program two outputs (for the first two states) that we need to determine the g_{ij} *and the* h_i*. Hence if we can find the two state model*

$$\begin{bmatrix} q_1(k+1) \\ q_2(k+1) \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \begin{bmatrix} q_1(k) \\ q_2(k) \end{bmatrix} + \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} u(k)$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_1(k) \\ q_2(k) \end{bmatrix}$$

we will have all that we need.

However the system identification program is not likely to give us directly what we want. The system identification toolbox will give us something in the form

$$\begin{bmatrix} p_1(k+1) \\ p_2(k+1) \end{bmatrix} = \begin{bmatrix} \tilde{g}_{11} & \tilde{g}_{12} \\ \tilde{g}_{21} & \tilde{g}_{22} \end{bmatrix} \begin{bmatrix} p_1(k) \\ p_2(k) \end{bmatrix} + \begin{bmatrix} \tilde{h}_1 \\ \tilde{h}_2 \end{bmatrix} u(k)$$

$$y = \begin{bmatrix} \tilde{c}_{11} & \tilde{c}_{12} \\ \tilde{c}_{21} & \tilde{c}_{22} \end{bmatrix} \begin{bmatrix} p_1(k) \\ p_2(k) \end{bmatrix}$$

The problem here is clearly the $\tilde{C}$ matrix, which we want to be diagonal so we can access the states. The obvious solution to this problem is to rewrite the state equations in another form, using a different set of basis vectors. Specifically, we have

$$p(k+1) = \tilde{G}p(k) + \tilde{H}u(k)$$
$$y(k) = \tilde{C}p(k)$$

Now let $p(k) = Pq(k)$, or $q(k) = P^{-1}p(k)$ so we have

$$q(k+1) = P^{-1}p(k+1) = P^{-1}\tilde{G}Pq(k) + P^{-1}\tilde{H}u(k)$$
$$y(k) = \tilde{C}Pq(k)$$

How do we choose P ? We want C to be a diagonal matrix, so $P = \tilde{C}^{-1}$, and then $G = P^{-1}\tilde{G}P$ and $H = P^{-1}\tilde{H}$.

Next we must deal with the types of inputs the system identification toolbox expects. The best way to think about this is to think about how we used sinusoids of different frequency to construct the Bode plot of the system in the first two labs. We needed to include sinusoids with enough different frequencies that we could accurately construct the Bode plot. If we had just used sinusoids at two different frequencies we would not have produced a very good Bode plot. The basic idea here is then that whatever input we put into the system must have sufficient ``spectral content'' that we could construct an accurate Bode plot if we have output at each of the frequencies that makes up the input. If you have taken ECE-300, this means we want the Fourier transform of the input to have spectral content up to the highest frequency we expect to see in our system. In controls we say we need an input with persistent excitation.

Finally, system identification is something of an art, and it takes practice to get good at it.

Part A: Tutorial Using the One degree of Freedom Rectilinear System

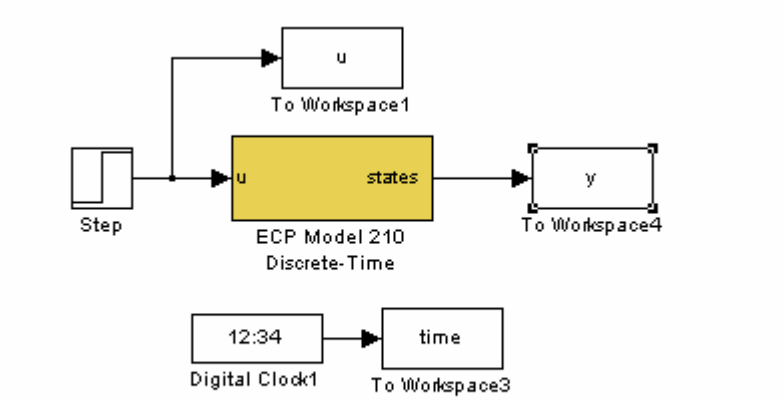
In the following steps, you should try and follow along and do the same things with your systems I did with my system. After this brief tutorial, you will be set free to explore on your own.

Be sure to load the correct controller personality file for the ECP system (and reset the system every time before you run it) !!!

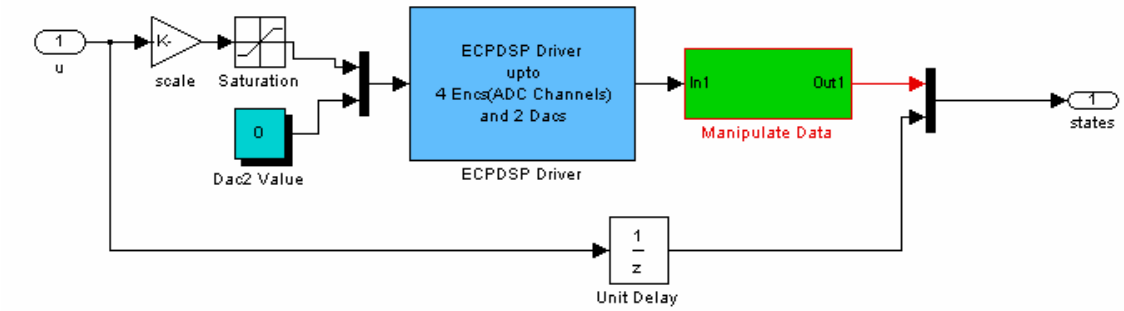
Step 1: Set Up the System. Set up the system the way you did in Lab 1. Be sure the Matlab model file for that system is in your current directory.

Step 2: Get the files. Copy the contents of the Basic Files folder into a folder for Lab 3. Compile **ECPRDPRest** and run it.

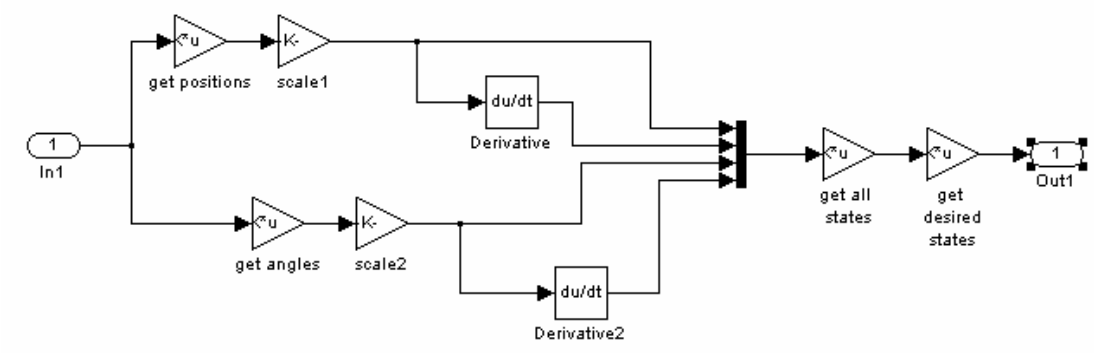
Step 3: Looking at the guts Start Simulink and open the file **Model210_DT_openloop.mdl.**It should look like the following:



If we look inside the *ECP Model 210 Discrete-Time* block, we see the following:

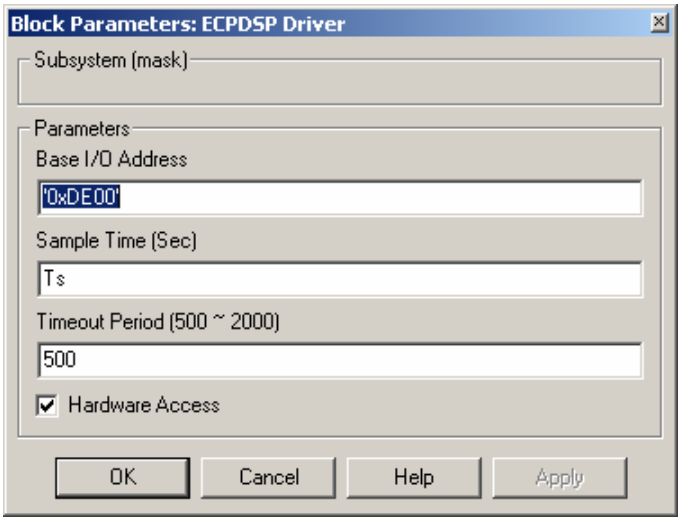


The top two state coming out of the system will be ``real states'', while the bottom state will be a delayed version of the input $u(k-1)$. If we look inside the *Manipulate Data* block, we see the following:

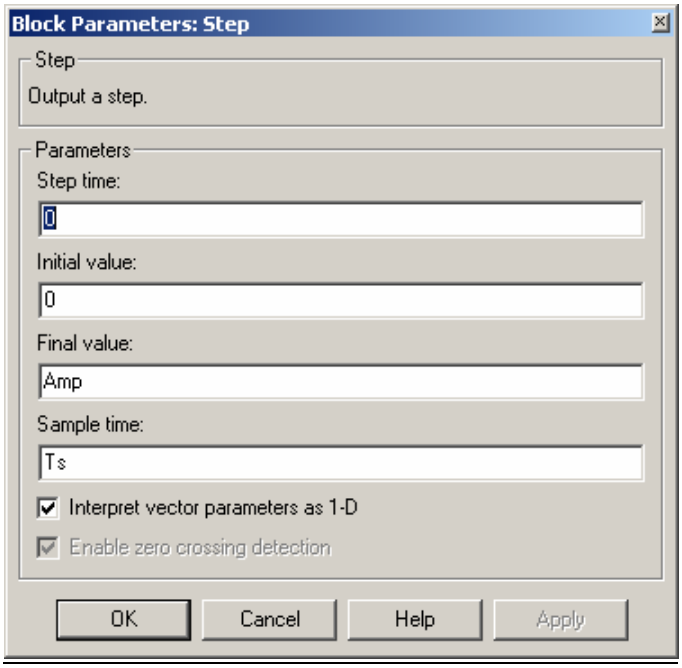


The only part you have to concern yourself with is the final block, which is a mask to pick off the states we want. Just after the *get_all_states* block the state variables are in the vector $\begin{bmatrix} x_1 & \dot{x}_1 & x_2 & \dot{x}_2 & x_3 & \dot{x}_3 & \theta_1 & \dot{\theta}_1 \end{bmatrix}^T$ The vector *get_desired_states* in the program **DT_openloop_driver.m** is currently set so the output of this block will only be the position and velocity of the first cart.

If we look inside the *ECPDSP Driver* block, we get to set a few parameters, as shown below:

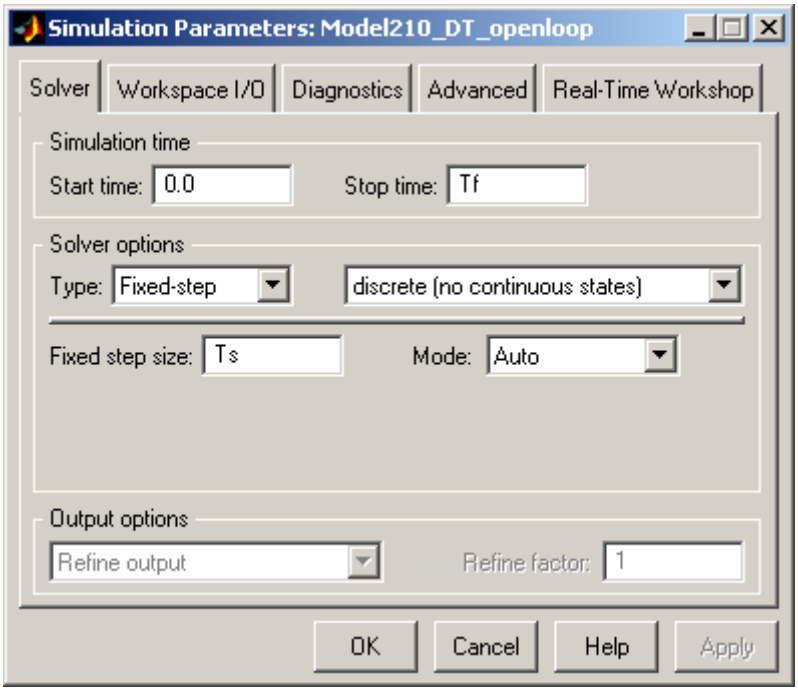


The variable Ts is the sample time, and it will be set in **DT_openloop_driver.m.** If we look inside the step command,



In the *Block Parameters* for the *Step* command we again see the sample time set at Ts and the final value set to Amp, which is also set initially by **DT_openloop_driver.m.** However, you are very likely to want to change the value of Amp as you go on. This can be done in Matlab's workspace.

If we look under simulation parameters, we should see the following



Ts and Tf are set by **DT_openloop_driver.m**, though you may want to change the final time in Matlab's workspace. I would suggest a final time (Tf) of between 3 and 20 seconds or so, it really depends on your input to the system.

Step 4: A Step Response Set the amplitude to something you think will work to move the system but not hit a stop and be sure the final time is sufficiently long. Do this in the script **DT_openloop_driver.m**, then run the script. Then compile and run **Model210_DT_openloop.mdl.**

Step 5: Save your data! We need to save our data so we can use it later. Since the variables time, u, and y are in Matlab's workspace, lets put them into a convenient file. I'll name my file **bobstepR,** so at Matlab's prompt I will type

save bobstepR time u y

When I want to get the time, input, and output vectors that go with this particular step input, I just type

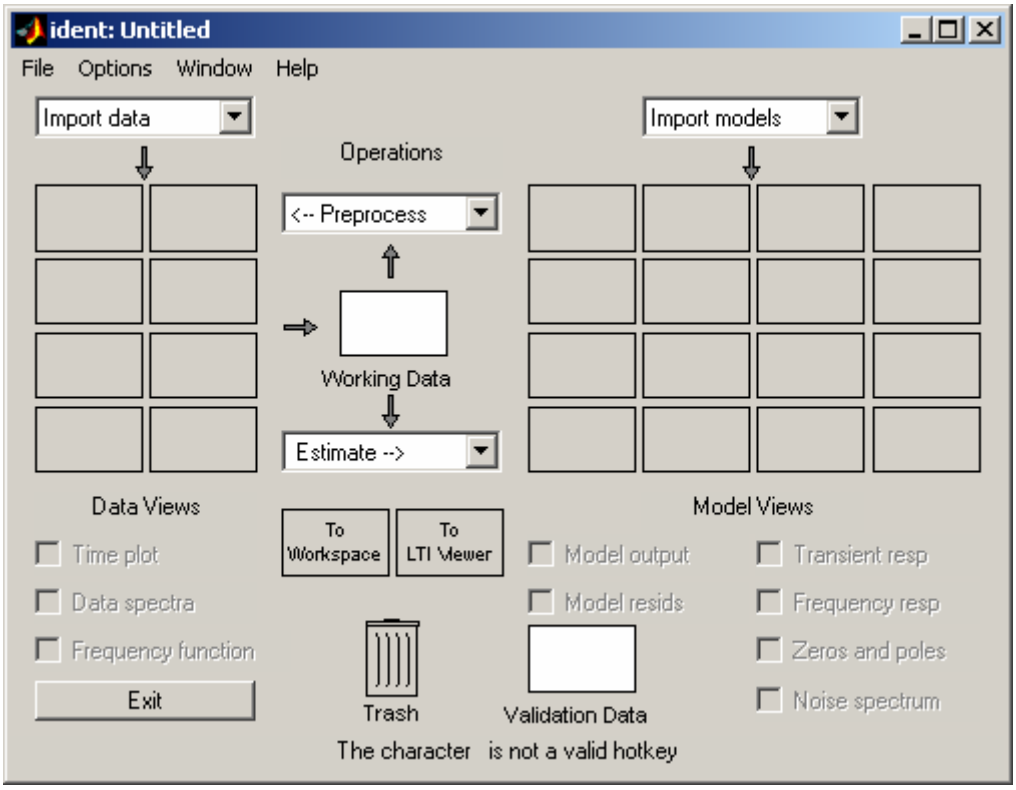
load bobstepR

and the time, u, and y variables will be available in the workspace. They will overwrite any other values of these variables that are already in the workspace, so be careful about this.

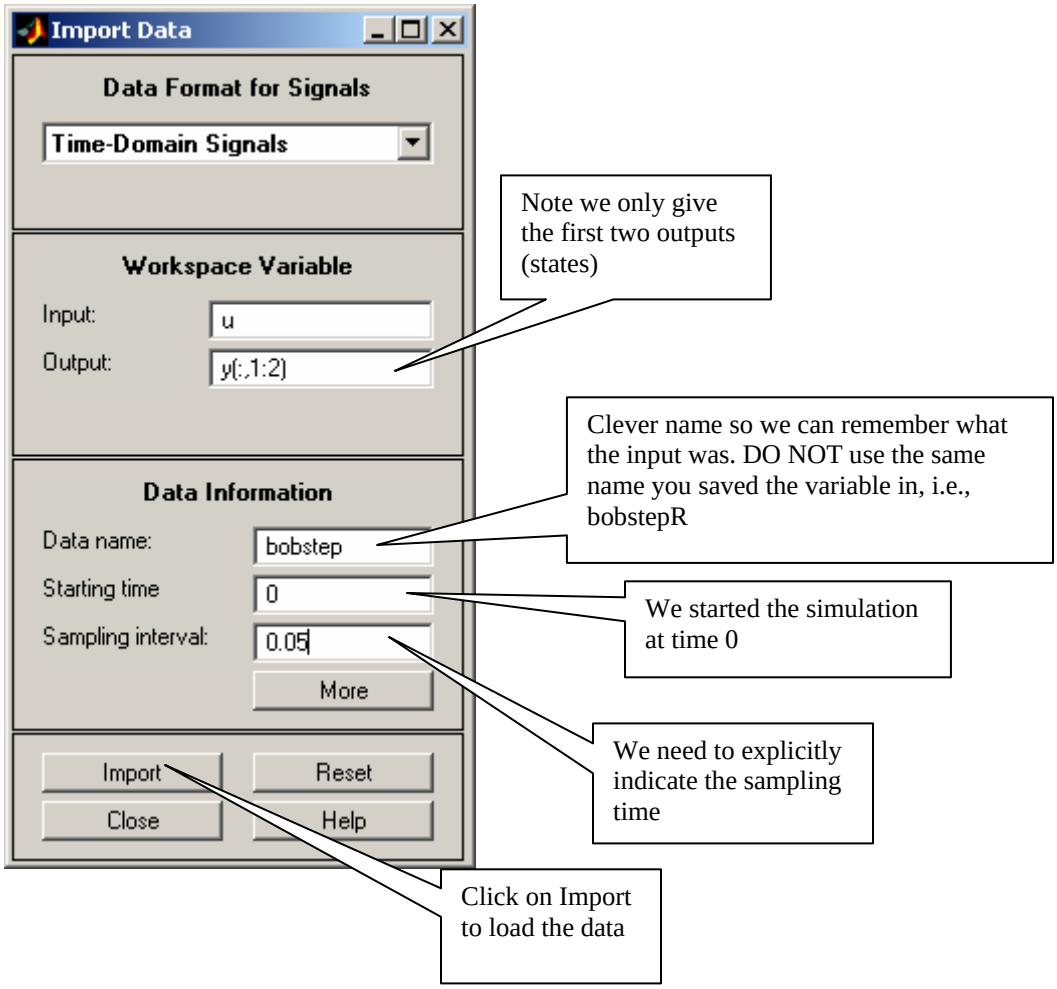
Step 6: Starting up Matlab's System Identification Toolbox At the Matlab prompt, type

ident

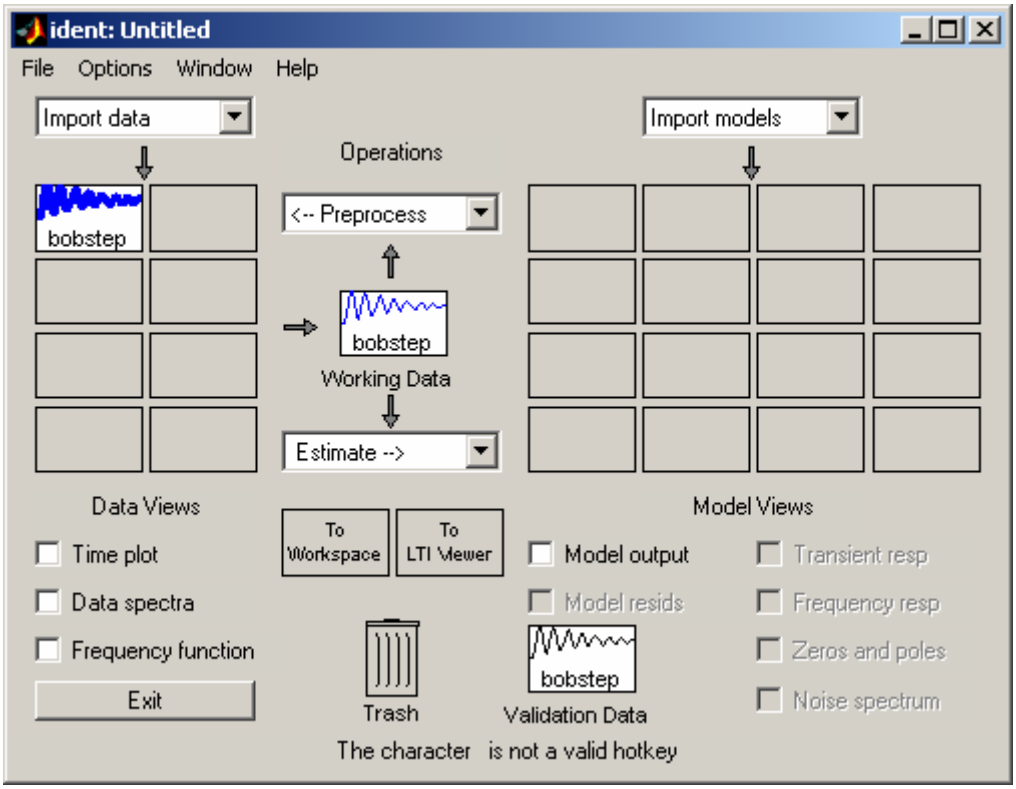
and you should get the following winidow:



Step 7: Importing Data into the Toolbox In order to use the toolbox, we need to give it some data. Under **Import Data**, select the option **Time Domain Data** and your should get the following window. I have filled in the data for my input.

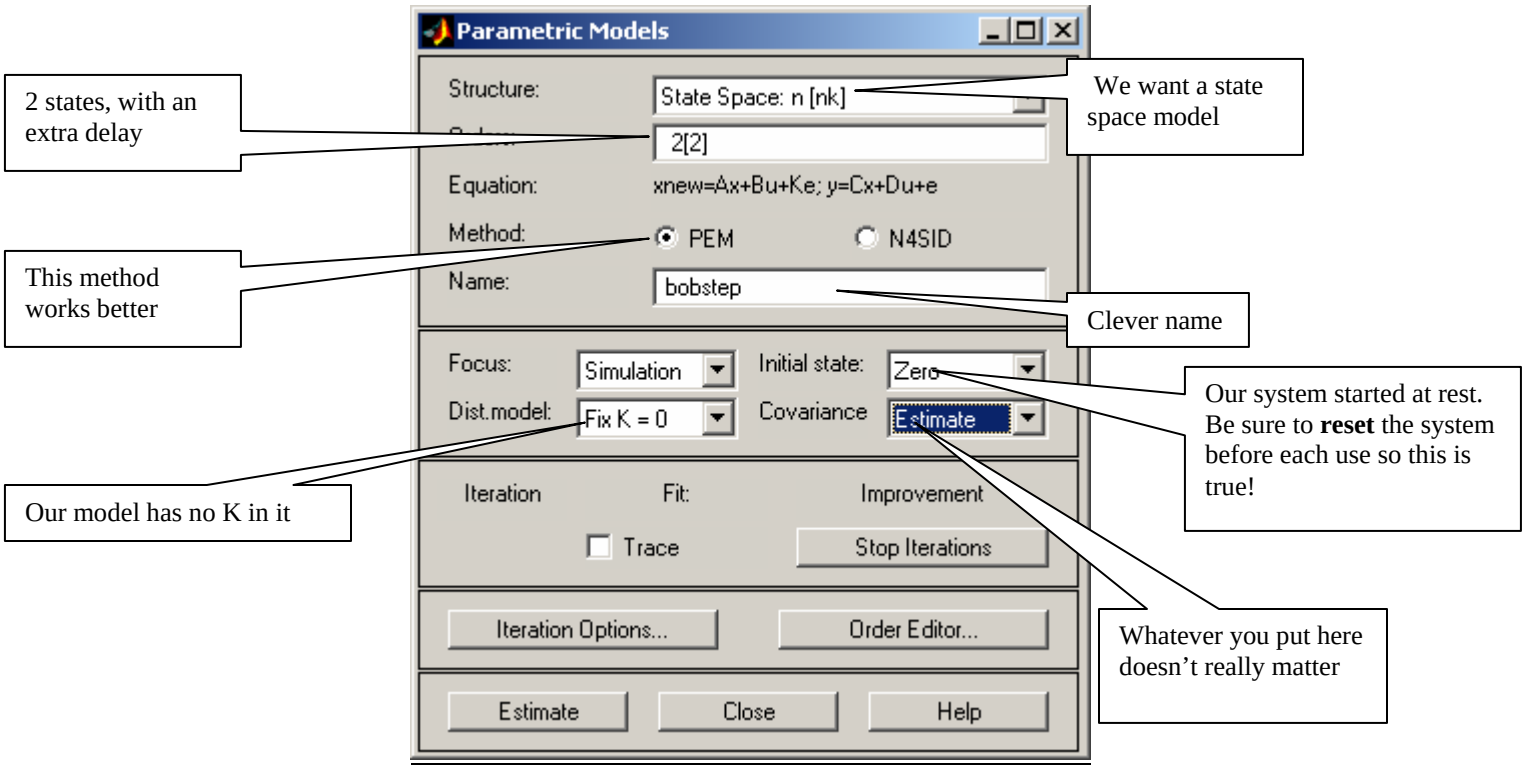


Once you import the data, you should see something like the following:

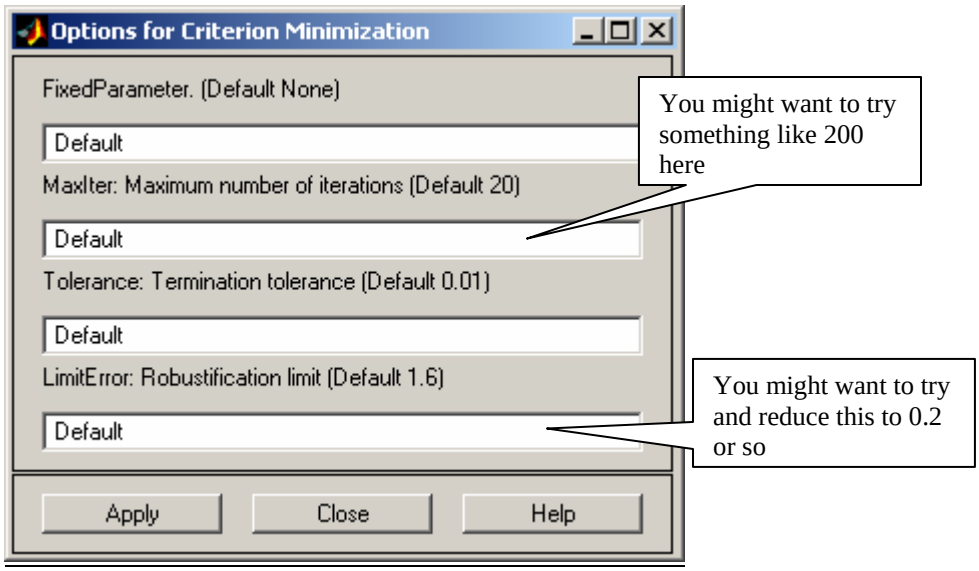


At this point we are ready to use the data to make our first discrete-time model. Since the only data that we have is from **bobstep,** that becomes the *Working Data.* To change the *Working Data*, we just drag a data set over to *Working Data*, as you will see.

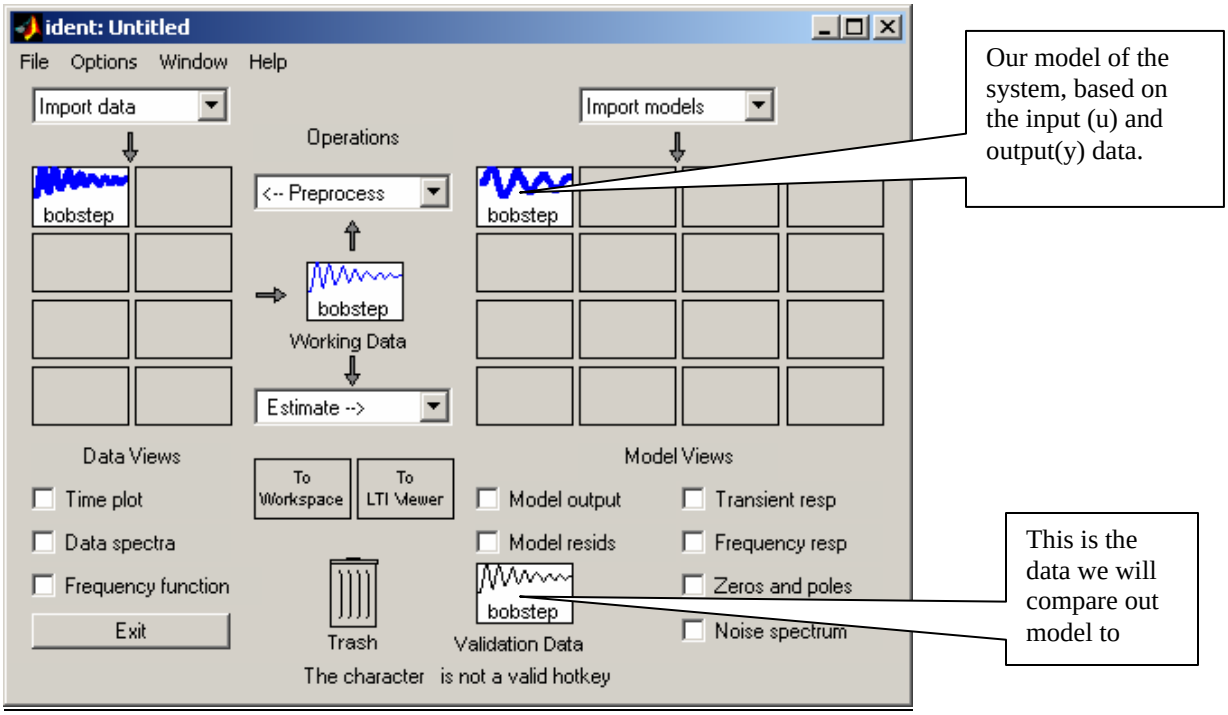
Step 8: Selecting the Model Now we need to select the model type. We want a state variable model, and we know that there will be an extra delay in out system. Under **Estimate**, select **Parametric Models.** You should get the following window, where I have again filled in the required fields.



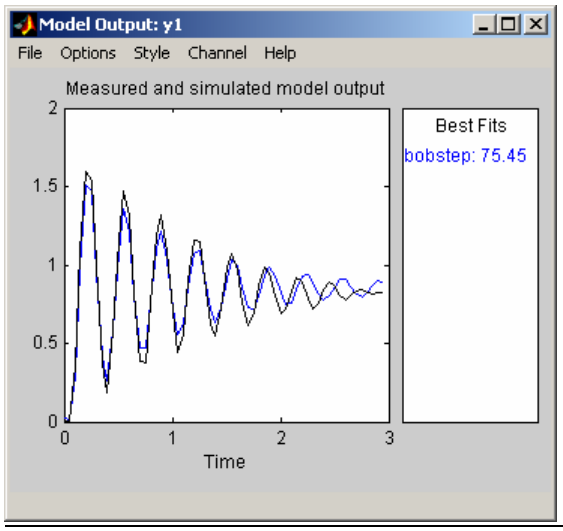
If we click on **Iteration Options**, we see we have some choices. Most of the time the defaults values will work fine, but sometimes you may want to change them.

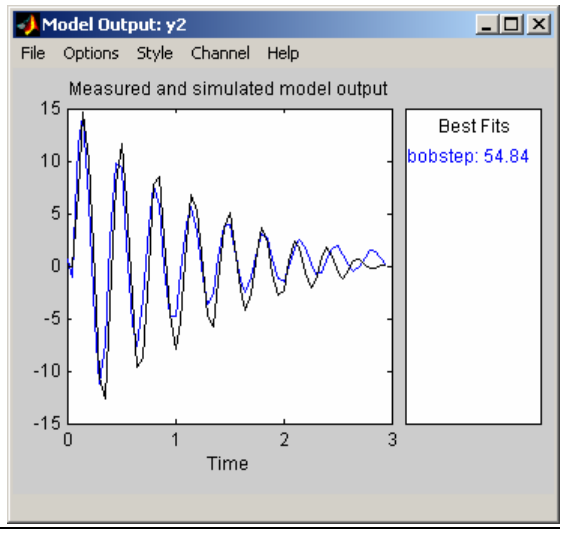


Step 9: Create the Model To make the model, click on **Estimate.** You should then get a model, as shown below:



We should now look at how closely the model matches our input. To do this, be sure **Validation Data** has the same name as the input data set. The click on **Model output.** The following is what I got (for a 3 second sample):

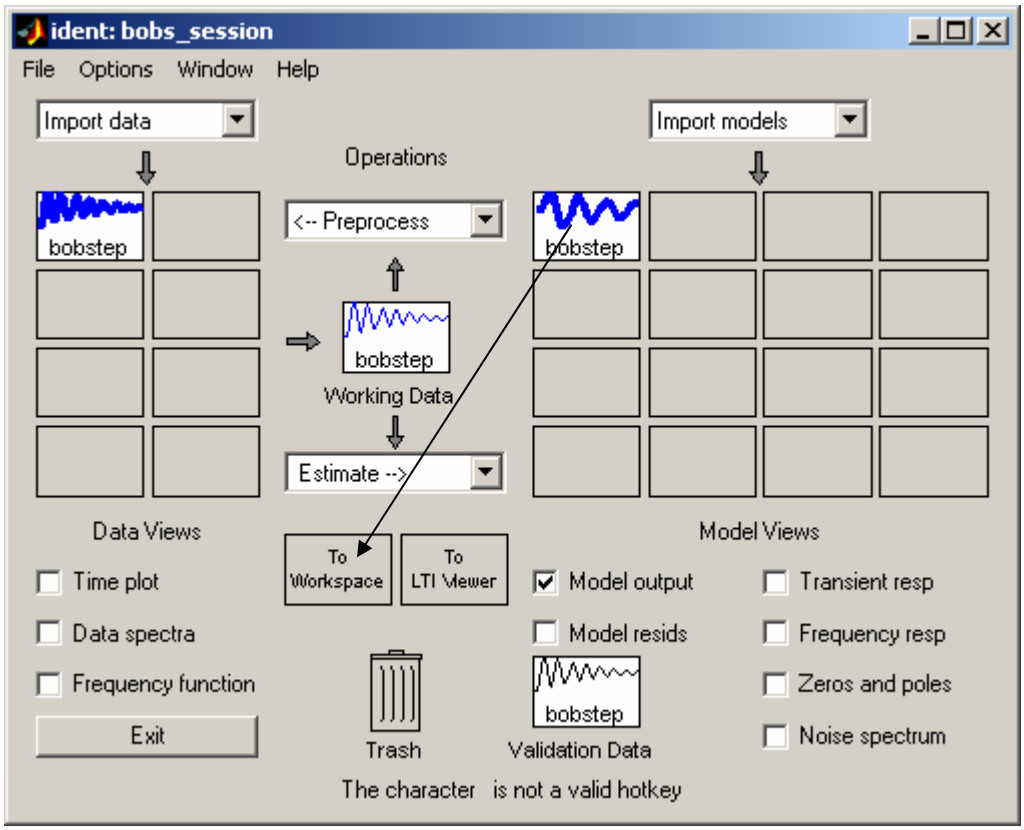




To go from one channel to the other select **Channel** and then choose whichever one you want. The black in the above figures show the real signal (the y values we imported) and the blue represents the output of our model.

Step 10: Save the Session! It may come as a shock to some of you, but the ECP systems have a nasty habit of locking up the computer. In order to save what you have done, you should save the session. Select **File** and the **Save Session As…** **You should save your session often to avoid loosing your work.**

Step 11: Comparing Models. At this point you are undoubtedly wondering how well does the model you just constructed compared to a sampled version of the continuous time system you made in Lab 1. In order to do this we first need to get our discrete-time model into Matlab's workspace. To do this, we just drag it to the **To_Workspace** box.



Now our workspace has access to the discrete-time model. Next we need to load the data we used to create this model. This is important since it gives us the system input and time vector. To do this, type

load bobstepR

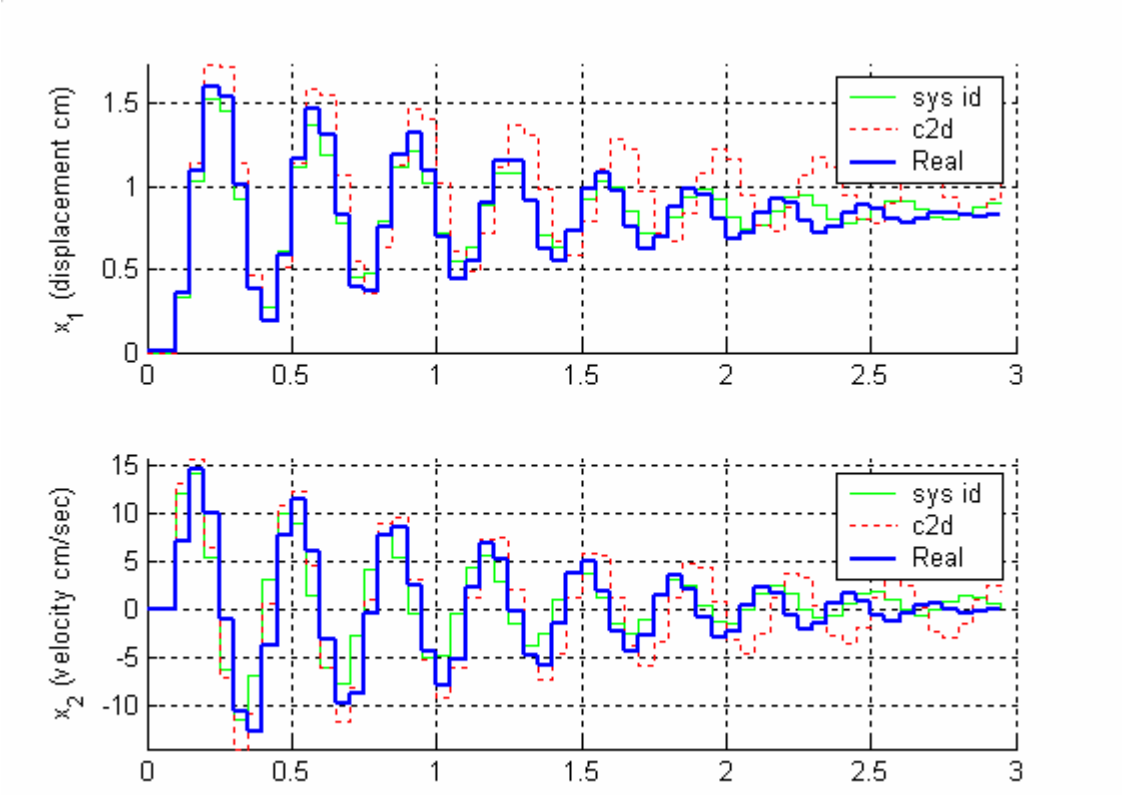
(Note: At this point we don'r really need to do this, but will will later on)

Next we use the program **compare_sys_id.m**. The arguments to this file are the input (u), the output (y), the time (t), the disdrete-time model file (model), and the length of time you want to

plot the simulation for (Tend). In addition, you need to edit the file so it loads the correct continuous time model file from Lab 1. Then, for my system, I would type

compare_sys_id(u,y,time,bobstep,3);

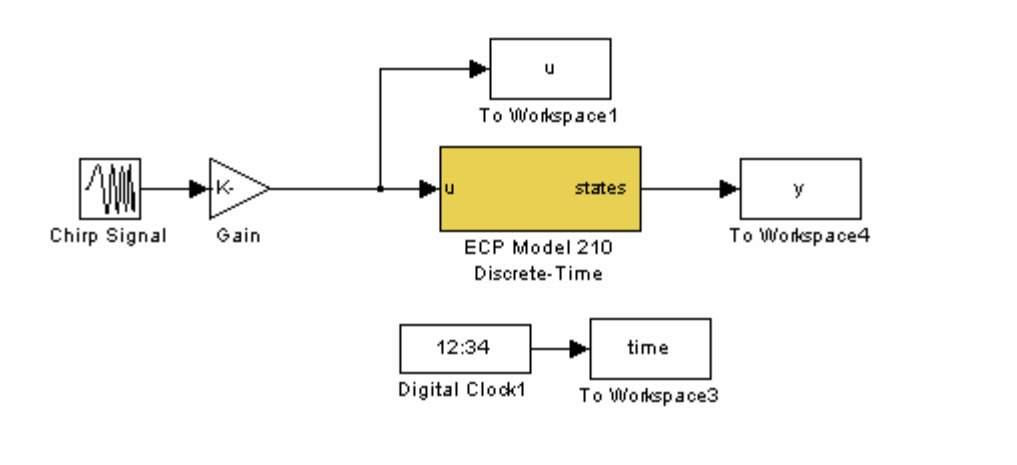
and I got the following output (yours will most likely be different):



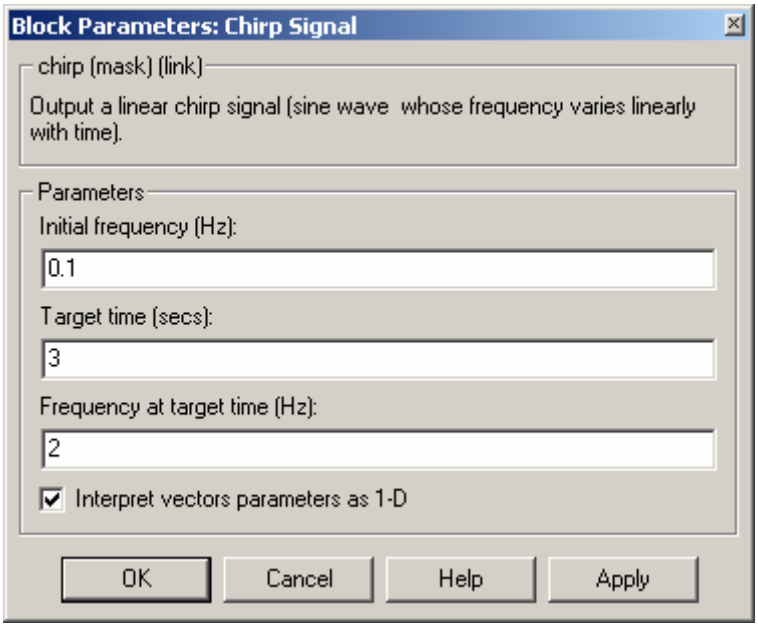
Here we see the response of the model created using the system identification toolbox(sys id), the response of the model made discretizing your continuous time model (c2d) , and the response of the real system (Real). The color schemes looked good to me, but feel fee to modify them. If you do change color schemes or line types, be sure they print well in black and white (unless you print your memos in color!)

Step 12: A New Data Sets Now we want to try and make a model of our system using a different input. Let's assume this time I use an input of a chirp. This is a signal that starts with a low frequency and ends with a high frequency (it sounds like a bird's chirp—hence its name).

First you need to disconnect the step from the input in **Model210_DT_openloop.mdl** and insert a chirp signal. You will notice that there is no way to control the amplitude of the chirp signal, so you will need to insert a gain to be sure to scale the signal to a small value. It is a good idea to start with a very small value in this gain (0.001) and then increase it until your system moves a reasonable amount. Your model file should look like the following:



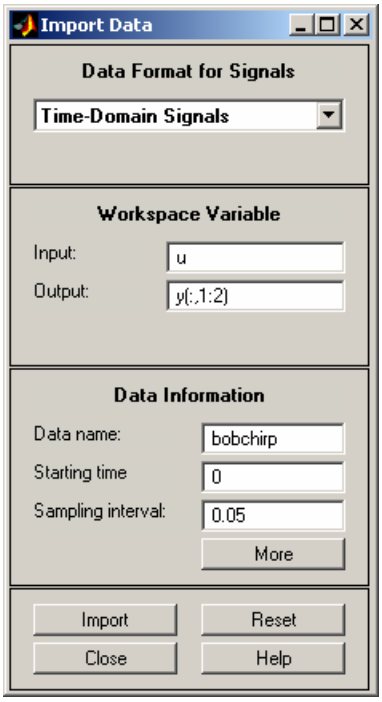
The parameters I used for my system is shown below, with a gain of 0.01. You should note that we are using very low frequencies since our system cannot really respond very quickly.



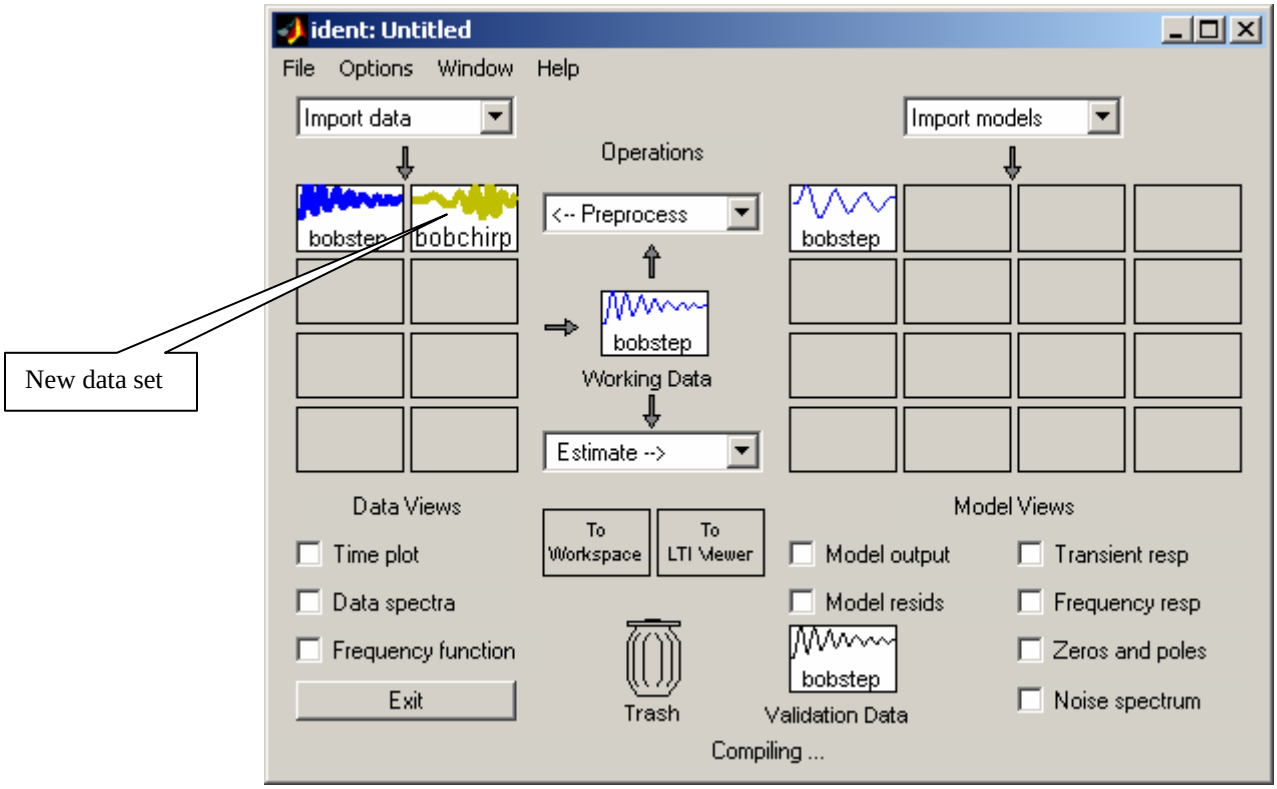
You should set the gain very low to begin with, compile the system and run it. Once you have found a good gain and your system moves a reasonable amount, save your data. I used

save bobchirpR u time y

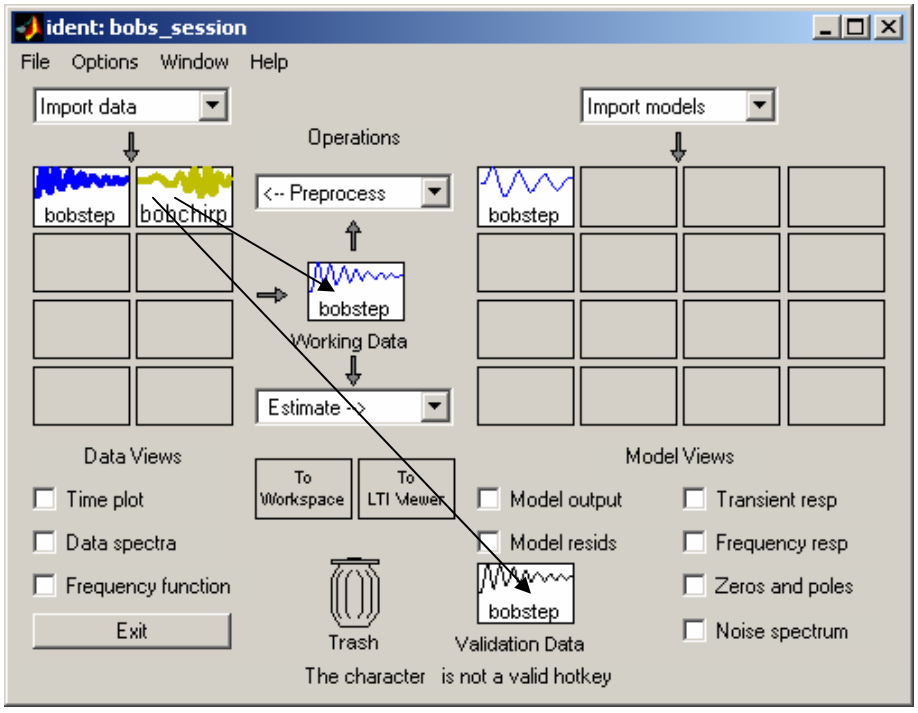
Step 13: Load Newe Data into the System Identification Toolbox. Now we load this data set into the toolbox.



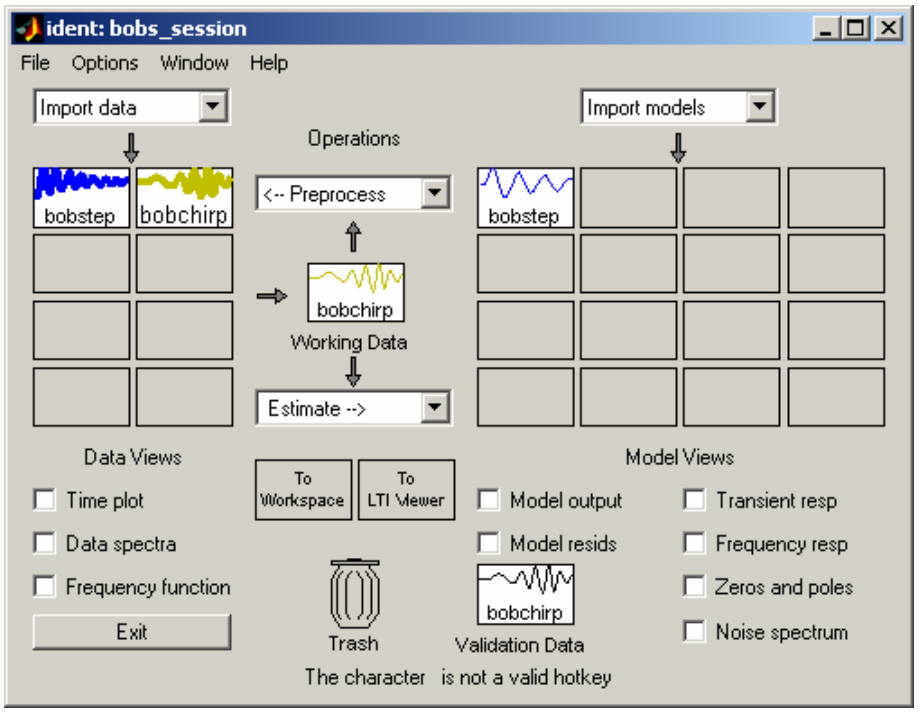
And it shows up in the main window



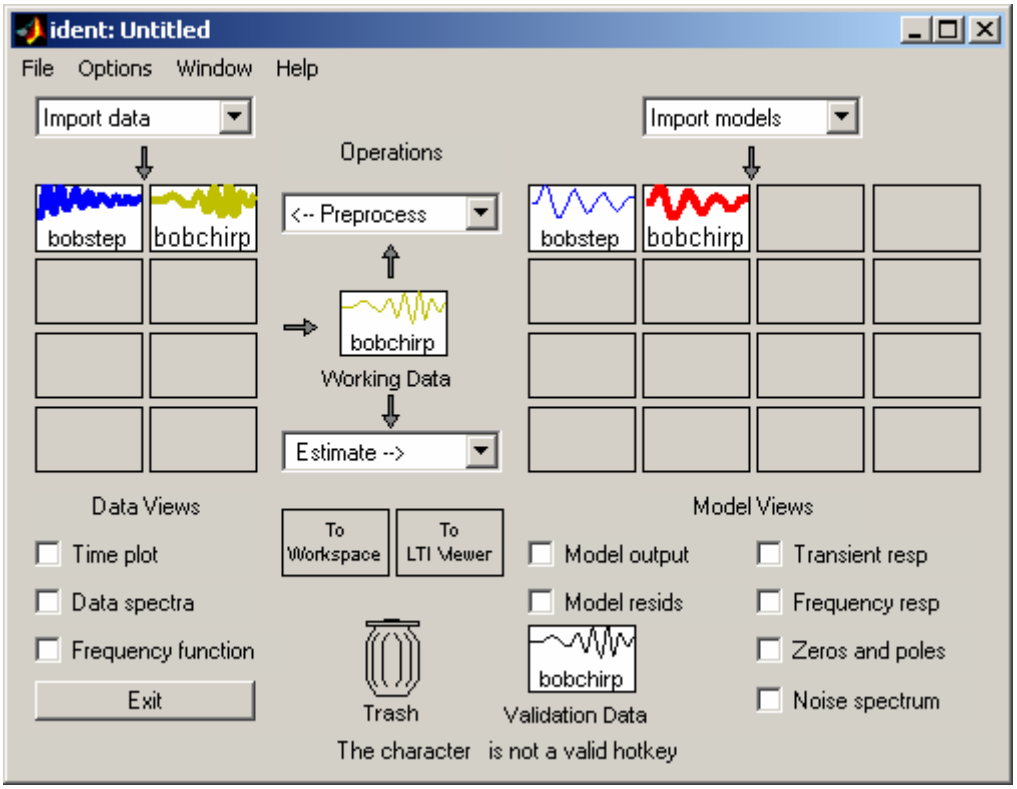
Step 14: Creating a Model from the New Data Set. Now we need to drag the new data to the *Working Data* area and the *Validation Data* area, as shown below:



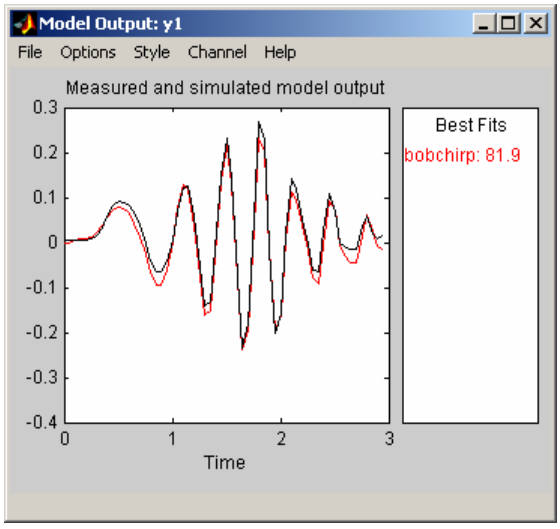
Which results in

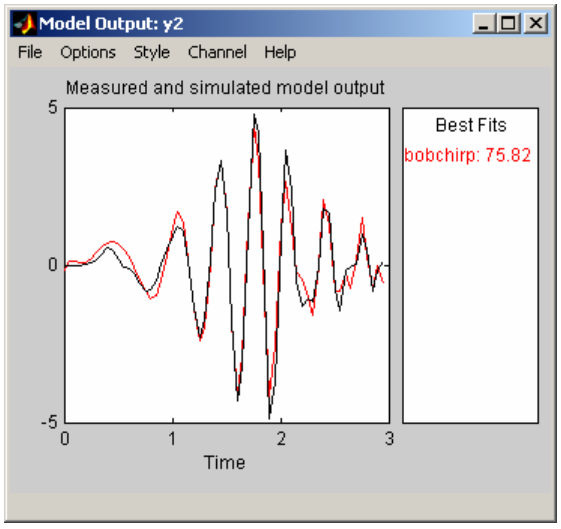


Step 14: A New Model.Next we construct a new state space model with this data set, which will be put into the model part of the window, as shown below:

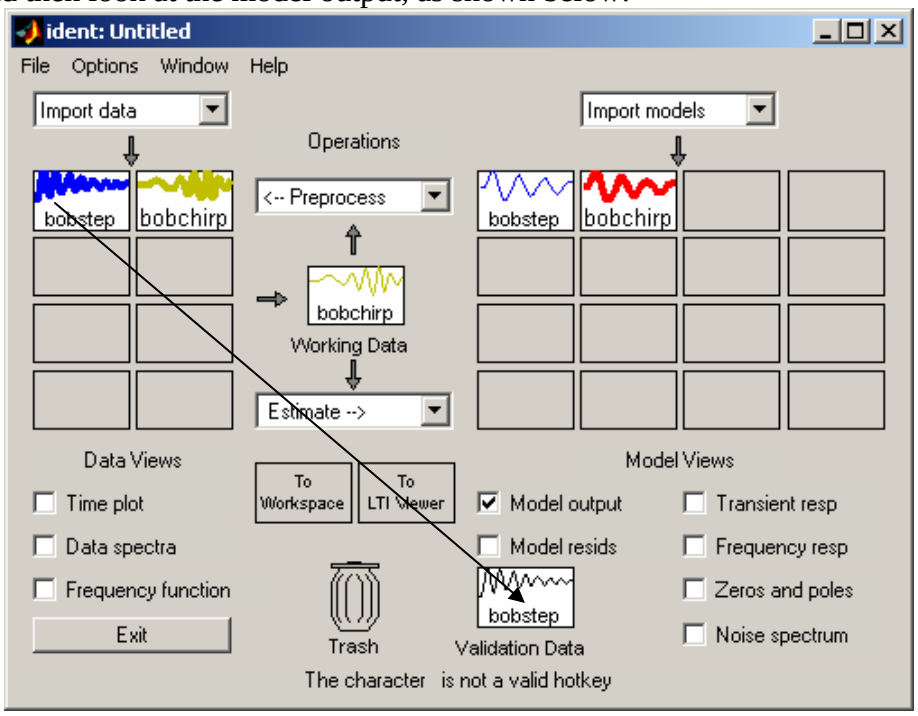


The comparison between the model and the input data (used to construct the model) is then shown below (for my system)

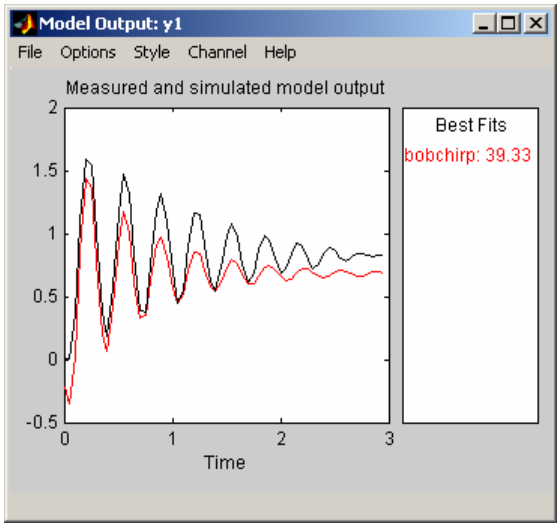


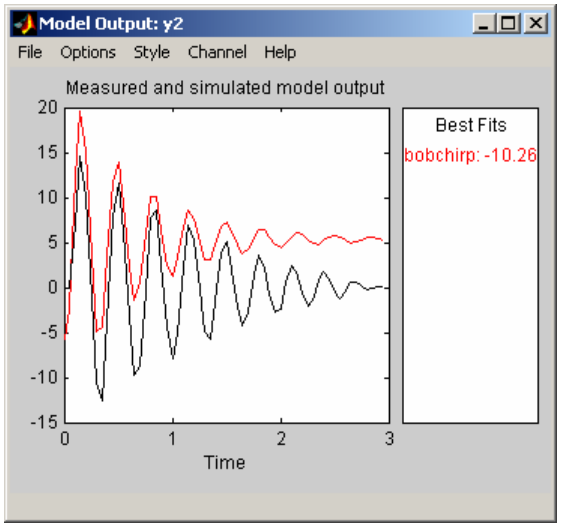


Step 15: We Finally Stop Cheating (Sort of). At this point you should be told that it is generally considered to be quite bad in system identification, and in many other fields, to use one data set to generate the model, and then try an validate the model by comparing it to the data used to generate it. This is certainly a good first step, since if the model doesn't match the data used to generate it, the model is not likely to work well for other inputs. Hence, in this case we will utilize the **bobstep** data set as a validation data set. To do this, we drag **bobstep** to the *Validation Data* box, and then look at the model output, as shown below:

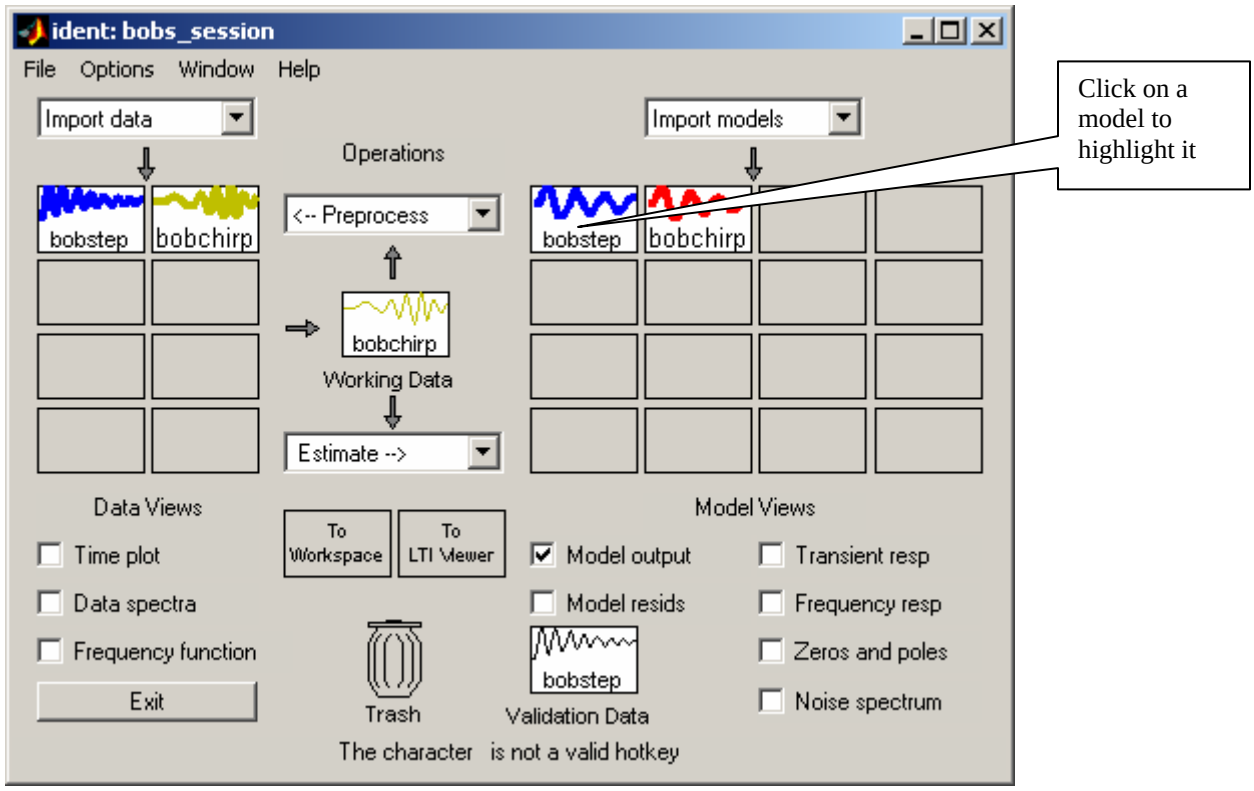


At this point, we can see how our newest model, **bobchirp**, matches the measured output when the input is a step. As these figures show, there is an offset between the model output and the measured output for this input. This is fairly common for these systems, and there are some ways we can try to fix this, which we will mention later in this lab.

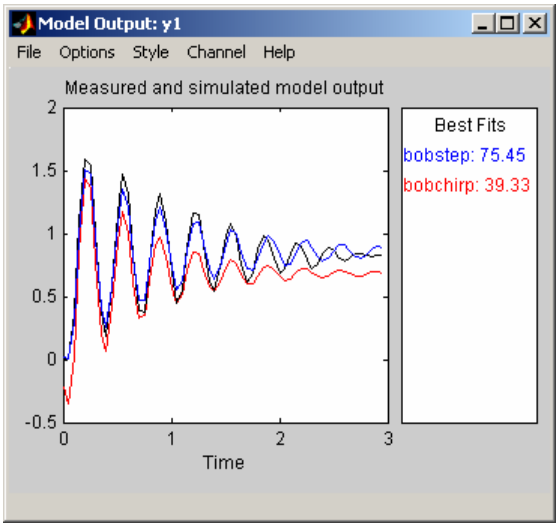


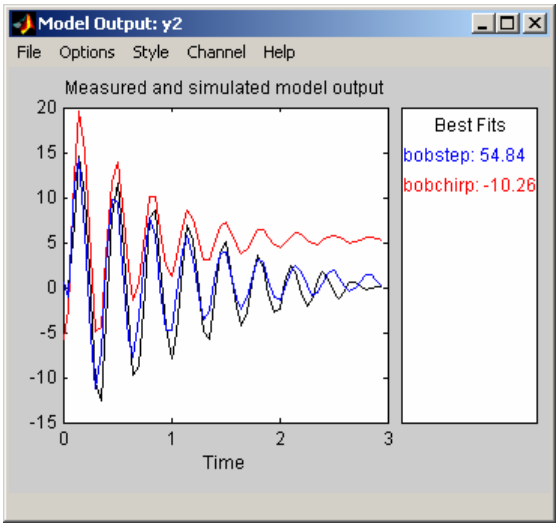


If we want to compare how well the models from both **bobchirp** and **bobstep** compare when the input is a unit step we make sure they are both highlighted, as shown below:

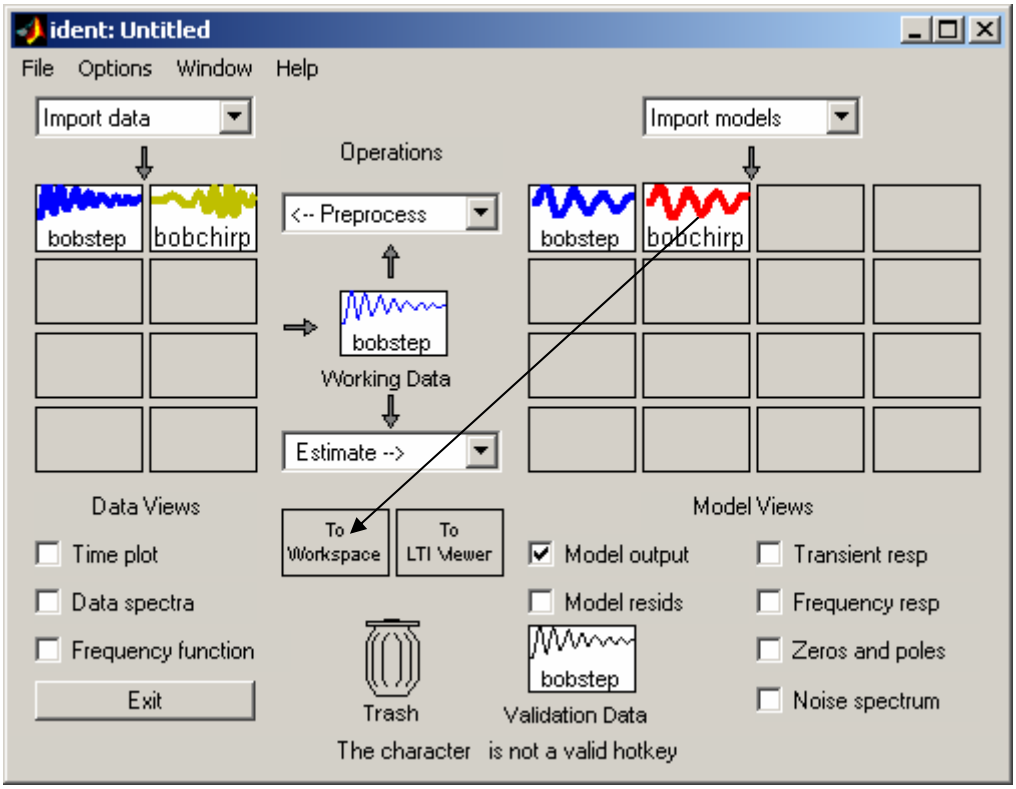


The outputs for the different models and the true output are shown in different colors, as shown below.





Step 16: Comparing with the Discretized Continuous Time Model (Again). In order to do this we again need to get our discrete-time model into Matlab's workspace. To do this, we just drag it to the *To_Workspace* box.



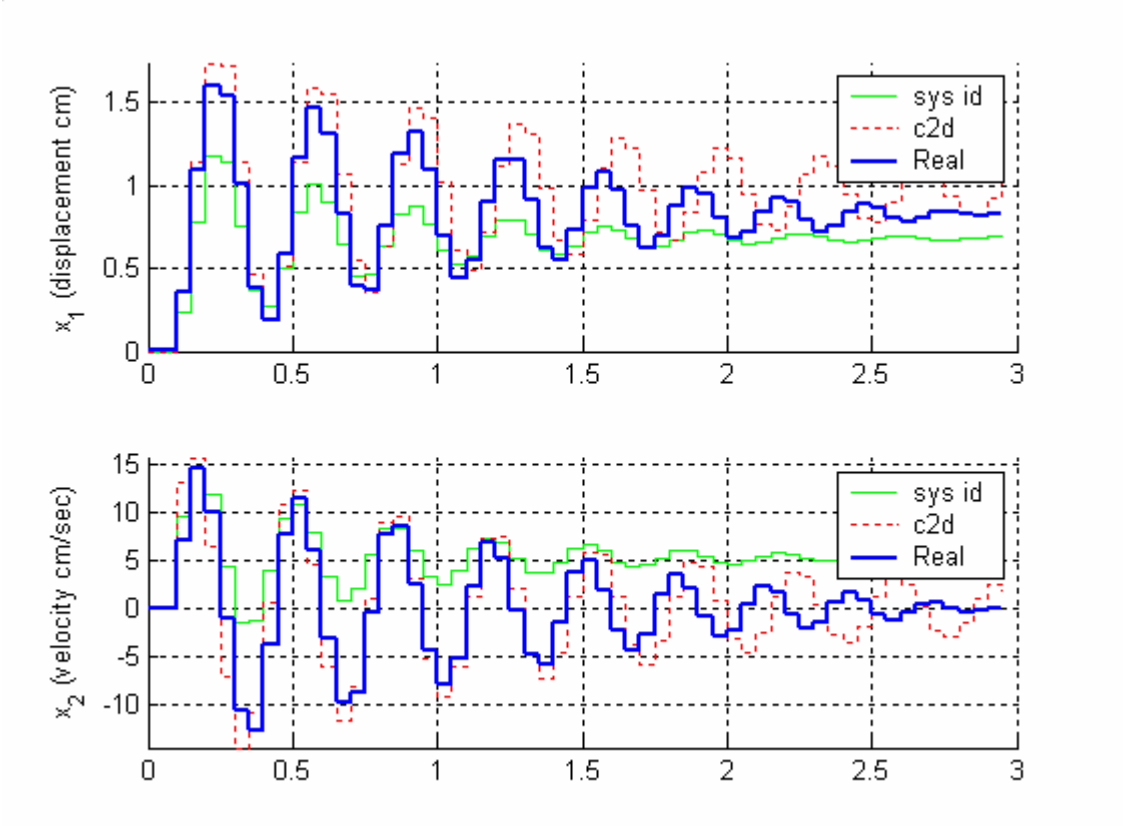
Now our workspace has access to the discrete-time model. Next we need to load the data we want to test our model against. This is important since it gives us the system input and time vector. If we are going to test against a step input, type

load bobstepR

Next we again use the program **compare_sys_id.m**. The arguments to this file are the input (u), the output (y), the time (t), the disdrete-time model file (model), and the length of time you want to plot the simulation for (Tend). In addition, you need to load the correct continuous time model file from Lab 1. For my system, I would type

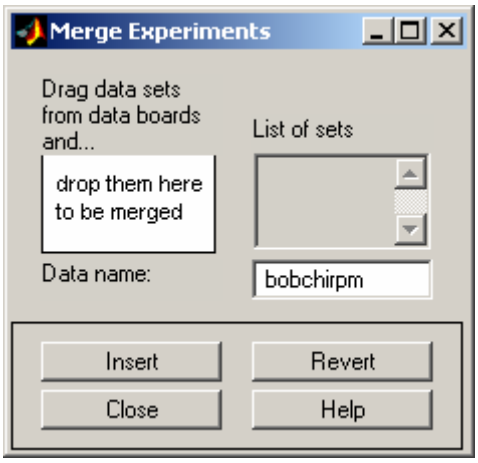
compare_sys_id(u,y,time,bobchirp,3);

and I got the output shown on the following page.

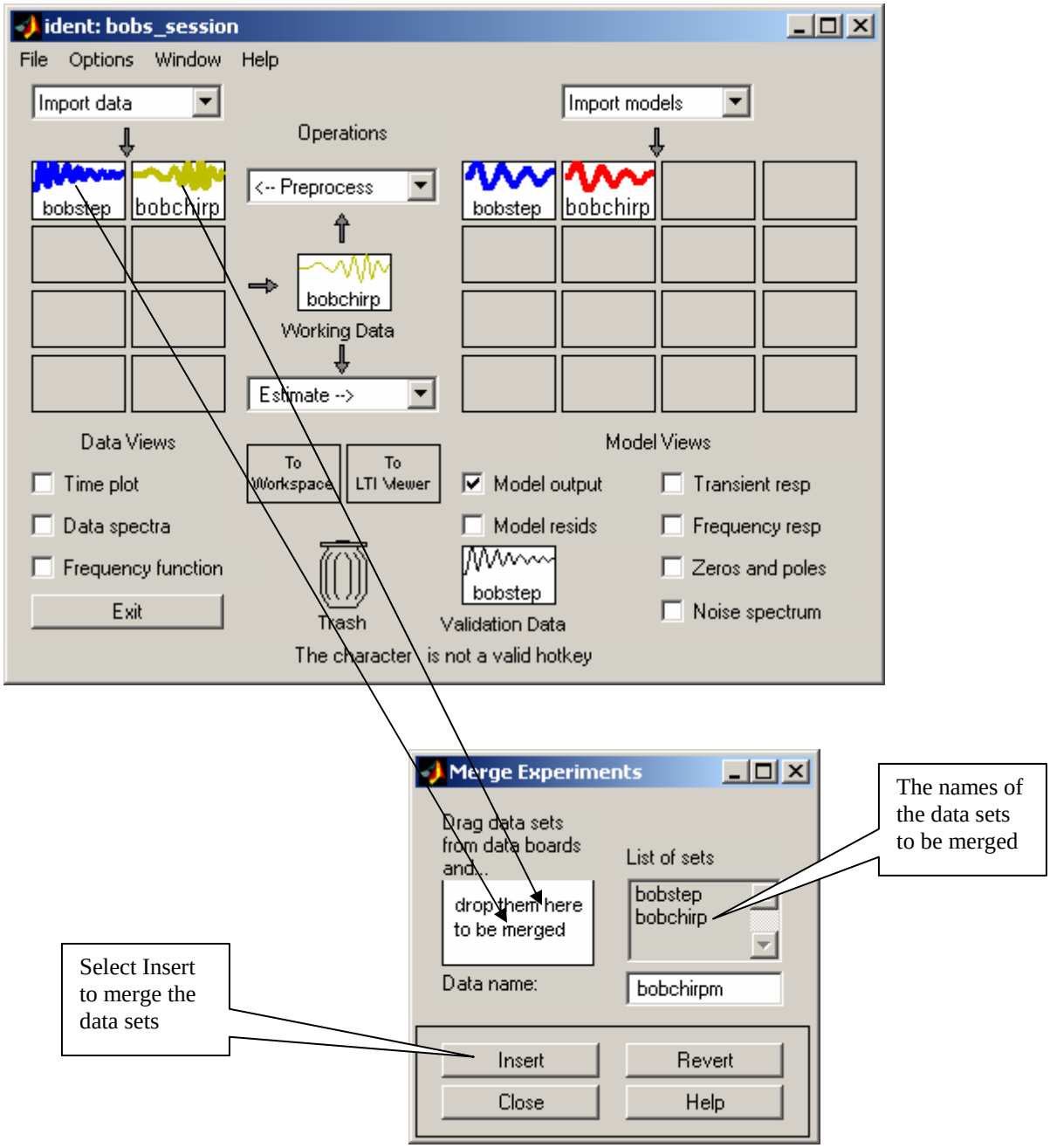


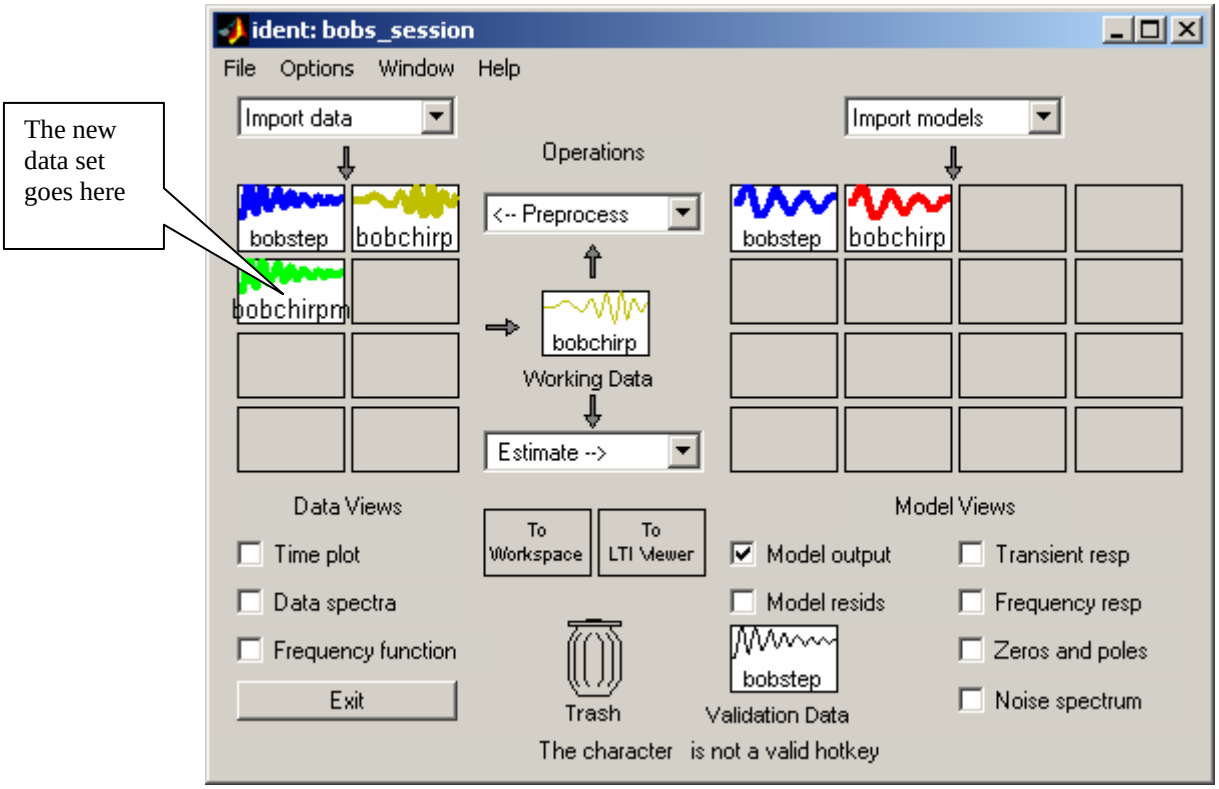
Here we see the response of the model created using the system identification toolbox(sys id), the response of the model made discretizing your continuous time model (c2d) , and the response of the real system (Real).

Step 17: Merging Data Files.Sometimes we want to combine more than one data file to try and make a model. To do this, under **Preprocess** select **Merge Experiments.** A new window will pop up as follows:



In order to merge the experiments, you need to drag them over and drop them where indicated, and then select **Insert**, as shown on the next page


ident: bobs_session
File Options Window Help
Import data
Operations
Import models
bobstep
bobchirp
<-- Preprocess
bobstep
bobchirp
bobchirp
Working Data
Estimate -->
Data Views
Model Views
To Workspace
To LTI Viewer
Time plot
Data spectra
Frequency function
Exit
Model output
Model resids
Transient resp
Frequency resp
Zeros and poles
Noise spectrum
bobstep
Trash
Validation Data
The character is not a valid hotkey
Merge Experiments
Drag data sets from data boards and...
drop them here to be merged
List of sets
bobstep
bobchirp
Data name:
bobchirpm
Insert
Revert
Close
Help
The names of the data sets to be merged
Select Insert to merge the data sets



You can now use the merged data set to construct a new model, and the response of the new model to any of the other data sets. Note, however, that sometimes merging files does not produce a better model than the two individual files did.

Part B: Some Possible Inputs/Guidlines

Below I have listed some of the things that I have tried. You do not need to try these, but they may be a good starting point. I also found for my systems that it was more difficult to get a good model for the rectilinear system than for the torsional system.

Generally Good Inputs are: *Pulses, Chirp, Bandlimited White Noise, Multiple Sine Waves*

Combinations of these are also good (use a summation block to add the inputs). Keep the frequencies at a maximum of 4-5 Hz, usually lower. The power in the white noise sources should be very small (start with 0.00001 and then increase it) I found that using pulses with a period of 5 seconds and a duty cycle of 50% in parallel with another input often worked fairly well. For many of the sources, such as noise or chirp sources, the system identification toolbox tends to produce a better model the more data it has to work with (i.e., the longer the system is allowed to run).

Part C: For You to Do

For each of your systems, you need to try and find *three* treasonably good models, in addition to *one* model using only a step input to create the model. (A total of four models.) For *each* of these models you need to include the following:

1) A description of the input source (or sources if you merged data). This needs to be sufficiently detailed that I, or future students, could exactly duplicate your input.

2) The corresponding name of the state variable model. What I mean is that at the end of the program **compare_sys_id.m** a state variable model is stored in the file *sys_id_1dof_model210.mat*.This file needs to be renamed and you need to tell me the name and relate it to the inputs used to create the model (from requirement 1 above). Be sure to use a naming convention so you can tell the torsional models from the rectilinear models! In lab 5 you will be comparing the different models when we try and use state variable feedback to control the system, and you need to know which file goes with which source.

3) A plot comparing the response of your model created by the system identification toolbox and the response of the discretized version of your continuous time model, for a step input. This is the output of the **compare_sys_id.m** program.

4) The name of the discretized state variable model for both the rectilinear and torisional systems. Again, at the end of the program **compare_sys_id.m** the discretized state variable model is stored in the file *c2d_1dof_model210.mat* .You need to change the name, or at least be sure the model file for the 205 system is named differently.

The body of your memo should contain the information from parts 1,2, and 4 above. You should try and see if you can put the information in a table. You should also indicate which of your input sources, other than the step, seemed to work best for creating a model that matched a step input.