# Simulink Workshop Quick Reference

This workshop presenter’s guide will help you remember all the steps to perform in this workshop. It is intended as a set of crib notes.

## Notes:

## Standard instructions will be done by the instructor and then by the class, a step at a time. Advanced skills are demo only by instructor.

## Make sure to push class content out to all training room computers prior to the class.

## Typically we only get through only about the first 7-8 sections

## Section 01: Construction Basics

This section will show the Library Browser, how to begin to construct simple models, and how to run and save your new model.

Motivate the problem by getting the audience to think about what it would take to model a falling object. We’re looking for things like gravity, mass, and eventually velocity and position.

[Open a model ready to proceed with this step.](matlab:Step_01)

**Standard Instructions:**

1. Open the Simulink Library browser
2. Create a new model by clicking on the ‘new model’ button on the Simulink Library browser menu bar
3. Add a Constant block from 'Sources'
4. Add a Second Order Integrator block from 'Continuous'
5. Add a Scope from 'Sinks’
6. Add a Terminator block from ‘Sinks”
7. Change the value of the Constant block to -9.8
8. Connect the blocks.  Once all the block connections have been made the model should appear as shown below in figure 1.0.

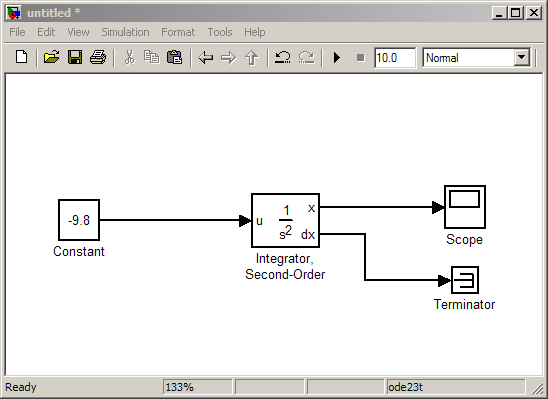


Figure 1.0: new Simulink model with blocks connected by signal lines

1. Press the Play button and double-click the Scope to view the results
2. Save your model
3. Now that the model is built ask how it is different than if you had derived and solved the equations of motion yourself by hand. The difference is you don’t have to solve the equations, Simulink does it for you. All you have to do is know the underlying physics.

**Comments:**

1. Block connections explained

The lines that connect these blocks are called signals and are time varying quantities.  The signal connections create a flow of a time varying signal from a source to a sink.  Connecting the output of one block to the input of another block forms a mathematical relationship between the connected blocks.  Time is not explicitly shown in a Simulink model, but the Simulink solver integrates time in the model.  The solver exercises each block in the proper order during a simulation.  Each block has a set of equations that can be solved as a function of time.  This distinguishes Simulink from C in that Simulink is a high level language providing a canvas to simulate the dynamics of a time-varying system w/o having to write mathematical solvers, as you would in C.

[Open the completed model](matlab:Step_02)

**Advanced Skills:**

* Different ways of opening a model
  + MATLAB start menu
  + Typing ‘Simulink’ at the MATLAB command prompt
  + Clicking the Simulink library browser button on the MATLAB desktop toolbar.

Use this as an example of the many ways to do the same thing in ML/SL.

* Automatically [align/distribute/resize](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/brchq3m.html'%5d,'-helpbrowser')) the blocks. (Be sure to save first.). Be sure to show the alignment guides.
* Hide the names on the integrator blocks
* Play with the [keyboard and mouse shortcuts](matlab:web([docroot%20'/toolbox/simulink/ug/f2-82531.html'%5d)) for zooming and panning. (zoom, unzoom, fit) Arrows keys to move blocks.
* Autoconnect signals from block to block.

## Section 02: Controlling Simulation

This section focuses on changing how Simulink numerically simulates your model.

Either continue with the model from the previous section, or [Open "Step\_02.mdl"](matlab:Step_02)

**Standard Instructions:**

1. Rename the Constant block to ‘gravity’
2. Label acceleration, velocity and position signals by double clicking on the signals
3. Set the initial condition of the position Integrator = 500
4. Set the initial condition of the velocity integrator = 5
5. Change simulation stop time to 100 seconds
6. Change the solver to ode23 and make the max step size 1 second.
7. Find a way to stop the simulation automatically when the position reaches 0.

* *Hint #1:* Look in the Simulink Library Browser 'Sinks' folder for a way to stop the simulation and in the 'Logic and Bit Operations' folder for a way to construct the condition. (use “Compare to Constant” block)
* *Hint #2:* If you're getting errors, look inside the logic block to specify the output data type.

1. Run your model to make sure it stops when the position reaches 0.

[Open the completed model](matlab:Step_03)

**Advanced Skills:**

* Does the simulation stop exactly when the position is 0?  If not, check [Enable Zero Crossing Detection](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f7-8243.html#f7-9506'],'-helpbrowser')) in the “Compare to constant” logic block and re-run the simulation.  Note: zero crossing detection works only with variable step solvers.
* Turn off the limit on the number of data points plotted on the scope
* Plot the position and velocity as separate axes in the same scope
* Open Model Explorer as an alternative way to explore a model. Look at integrator and/or constant blocks. Change their values, or reference ML workspace variables. Just another view into the model as an option.
* [skip] Using the [Model Explorer](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f3-85123.html'%5d,'-helpbrowser')), set up your own [default Configuration Parameters.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f11-35796.html#f11-69562'],'-helpbrowser'))
* Launch Model Explorer
* Select ‘Show Configuration Preferences’ from the Model Explorer ‘View’ menu
* Edit some preferences
* Explore common configuration parameter settings:
  1. Solver selection
  2. Step size
  3. Driving inports with ML data
  4. Logging data
  5. Starting from a previously saved state
  6. RTW settings all in one place
* Run the simulation programmatically using the ‘sim’ command, for example:

>>simOut = sim(gcs, 'StopTime', '10') – or –

>>simOut = sim(‘step\_03’, 'StopTime', '10')

## Section 03: Vectorizing Signals and Model Hierarchy

Simulink can pass vectors and matrices as signals, saving you time and extra lines.

Either continue with the model from the previous section, or [Open "Step\_03.mdl"](matlab:Step_03)

**Standard Instructions:**

1. Make gravity a 3 element vector: [0 0 -9.8]
2. Turn on the labels for signal dimensions. From the 'Format' menu, go to Port/Signal Displays and select Signal dimensions. Press Ctrl-D to [update your diagram.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/bqjpi9r.html'%5d,'-helpbrowser'))
3. Modify the initial conditions to be vectors:

Run the sim. Notice that the simulation stops right away. Query the audience for an explanation.

1. Add a ‘Selector’ block from the “Signal Routing” library and connect it to feed the stop logic as shown in figure 5.5.
2. Configure the Selector block to pass through only the third (altitude) component of the position vector.
3. Run a simulation and verify the simulation stops when the altitude reaches 0.
4. Create a subsystem around the stop logic blocks. Select the stop logic blocks. Right-click on them and select 'Create Subsystem.' Rename the subsystem to 'Stop Logic.' Double-click on the subsystem to look inside. Rename the input port to 'altitude.'

[Open the completed model](matlab:Step_04)

**Advanced Skills:**

* Replace your scopes with Simulink signal viewers, or use the Signal and scope manager. Right-click on the signal, go to 'Create & Connect Viewer' and select a Simulink viewer. [Why might you want to do this?](matlab:web%28%5B%27jar:file:///%27,matlabroot,%27/help/toolbox/simulink/help.jar%21/ug/brg_yqr-1.html%27%5d,%27-helpbrowser%27%29) To avoid having to clutter up your model with scope blocks.
* Discuss the three types of subsystems: virtual, atomic and model reference, but not in great detail. Atomic subsystems and model reference will be covered in more detail in sections 7 and 8.

## Section 04: Library Basics

This section shows the basics of libraries and configurable subsystems.

Either continue with the model from the previous section, or [Open "Step\_04.mdl"](matlab:Step_04)

**Standard Instructions:**

1. Open “drag\_library.mdl.”
2. Drag the Variable Density subsystem to your falling object model above the integrators. Invert it by typing Ctrl+I.
3. Open the subsystem model and explain that it implements a drag equation. Be sure to point out the computation of unit vector so we get the correct direction for application of the resulting force.
4. Connect your altitude and velocity signals to the drag subsystem inports.
5. Use a Gain block to scale the output signal of the drag model by 1/m. Why? Because we are summing accelerations, not force and the output of the drag subsystem is force.
6. Use a Sum block to add your drag and gravitational accelerations together and feed this cumulative result into the velocity integrator.
7. Define the parameters in MATLAB: rho = 1.2; Cd = 0.333; A = 1.5; m = 150. These parameters are used by the drag models in “drag\_library.mdl”
8. Run the model. The falling time should be slower due to wind resistance.

[Open the completed model](matlab:Step_05)

**Advanced Skills:**

* Create a data dictionary script to define rho, Cd, A, and m. Call this script from a Simulink callback. (From the Simulink menu, go to 'File', 'Model Properties.' Click on the [callbacks](matlab:web%28[%27jar:file:///%27,matlabroot,%27/help/toolbox/simulink/help.jar%21/ug/f4-122589.html%27%5d,%27-helpbrowser%27%29) tab and select 'PostLoadFcn.' Type the name of the data dictionary function).
* Show how to modify library linked elements two ways:

1. from the library – rename some blocks in the library subsystem and update diagram to see changes reflected in model
2. from the model – rename or delete blocks in the model, then right click on the subsystem, select Link options\resolve links and push changes to the library.

* [Add drag\_library to the Library Browser.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/brkl0a6-1.html'%5d,'-helpbrowser'))

>> web([docroot '/toolbox/simulink/ug/brkl0a6-1.html'])

* Create directory on ML path containing top-level library file (already done in demo)
* Add customized “slblocks.m” file in this directory (already done in demo)
* Close SL Library browser
* Refresh SL Library browser using >>sl\_refresh\_customizations command or (F5) with SL library browser open
* Re-open SL library browser and look for the new “My library” entry.

## Section 05: Masking Subsystems

This section shows how to mask subsystems and add model callbacks.

Either continue with the model from the previous section, or [Open "Step\_05.mdl"](matlab:Step_05)

**Standard Instructions:**

1. [Open drag\_library.mdl](matlab:open('drag_library.mdl'))
2. Double click the Constant density model in the Drag\_library.mdl. We want to mask the variable density model so it has a similar GUI interface. Masking the subsystem is how we achieve this.
3. Right-click on Variable Density Model and choose 'Mask Subsystem.'
4. Create a mask on Variable Density similar to the one on Constant Density. [More help on masking.](matlab:web([docroot%20'/toolbox/simulink/ug/f8-15210.html'%5d)) By adding the following prompts and associated mask variables:

Prompt                                Mask Variable

a)      Drag Coefficient [N/A]      Cd

b)      Reference Area [m^2]        A

5)    Update diagram (<ctrl-d>) to update the model with the new, masked version of the drag block.

6)    Double click the Drag Model subsystem to open the mask GUI, prompting the user to enter values for the variables associated with the mask.

[Open the library with masked subsystems: "drag\_library\_masked.mdl"](matlab:drag_library_masked)

[Open the completed model: "Step\_06.mdl"](matlab:Step_06)

**Advanced Skills:**

* Define the mass in the [model workspace.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f4-140122.html'%5d,'-helpbrowser'))
  1. Open Model Explorer and select ‘Model Workspace’ in the ‘Contents’ pane.
  2. Change the ‘Data source:’ to ‘M-code’ and enter the command ‘m = 2000;’ in the M-Code edit window as shown below in figure 5.1.
  3. Apply the change
  4. Click the “Reinitialize from Source” button
  5. Explain workspace scoping rules
  6. Run a simulation to show that the mass has a value of 2000, not 150.
* Add a “Conti” graphic to the mask for the variable density subsystem.
* Add a configurable subsystem template to the library.
* Double-Click on the template and check the ‘Member’ boxes for the ‘Constant Density’ and ‘Variable Density’ subsystems
* Drag the configurable subsystem template from the “Drag\_library” to the falling object model
* Set the 'Block Choice' to 'Variable Density'
* Connect the Drag Model subsystem to the falling object model. The result should appear as shown in figure 5.2, below.
* Use [get\_param](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/slref/get_param.html'%5d,'-helpbrowser'))(gcb, 'BlockChoice') and [set\_param](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/slref/set_param.html'%5d,'-helpbrowser'))(gcb, 'BlockChoice', ‘choice’) to change the configurable subsystem from the command line. Note: in this case ‘choice’ will be one of ‘*Variable Density Drag*’ or ‘*Constant Density Drag*’.

## Section 06: Buses

This section introduces buses as a means to organize your signals.

Either continue with the model from the previous section, or [Open "Step\_06.mdl"](matlab:Step_06)

**Standard Instructions:**

1. Convert all the blocks in the model into a subsystem.
2. Rename the subsystem to 'Plant'. The result should appear as shown below in figure 6.1.

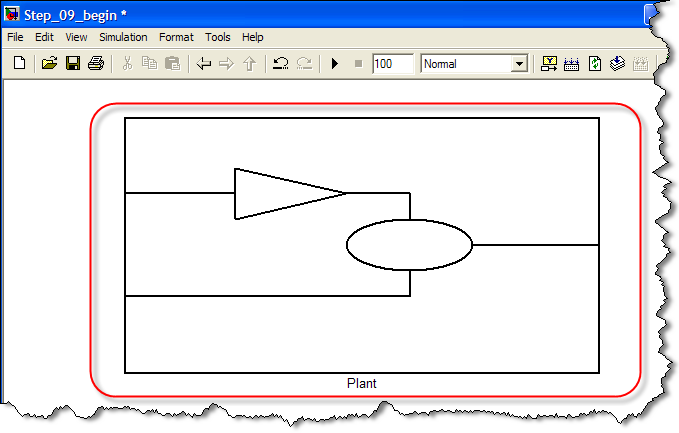


Figure 6.1: creating a plant subsystem out of the existing blocks in the falling object model

1. Put a new input port in “Plant” and name it “thrust”.  The result should appear as shown below in figure 6.2.

1. Add the thrust input signal to the gravity and drag accelerations (remember to also scale ‘thrust’ by 1/m).  The result should appear as shown in figure 6.3, below.

1. Add a new constant block called ‘Thrust’ to the top level of the model and give it a constant value of [0 0 1000].  Connect the new ‘Thrust’ constant block to the ‘Thrust’ input port of the plant subsystem.  The result should appear as shown in figure 6.4, below.
2. From the Signal Routing library, add a ‘Bus creator’ block to the ‘Plant’ subsystem.  Connect the position and velocity signals to the Bus Creator inputs.
3. Add an ‘Output’ port called “State” to the ‘Plant’ subsystem.  Connect the output of the [bus](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/bq4icqm-1.html'%5d,'-helpbrowser')) to the new ‘State’ output port.

1. Delete the scope blocks inside the Plant subsystem.  The ‘Plant’ subsystem should now appear as in figure 6.5, below.

1. Add a Bus Selector block at the root level of the falling object model to get the position and velocity signals back out of the bus and plot them with scopes.  Connect the output of the ‘Plant’ subsystem to the input of the ‘Bus Selector’ block. Select the position and velocity bus signals by double clicking on the “Bus Selector” block and selecting these signals from the “Signals in the bus” pane.

1. Add a pair of ‘Scope’ blocks to the top level model.  Connect the position and velocity outputs, respectively from the bus selector to the two new ‘Scope’ blocks.  The result should appear as shown in figure 6.6, below.

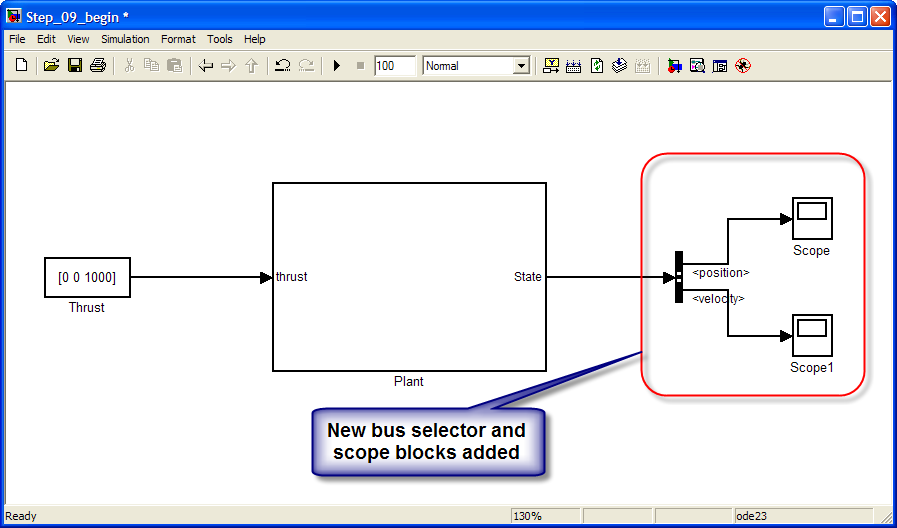


Figure 6.6: connecting scope blocks to the outputs of the bus creator block.

1. Run your model and observe the signal lines.

[Open the completed model: "Step\_07.mdl"](matlab:Step_07)

**Advanced Skills:**

* Name the bus signal and [log](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/brh7636.html'%5d,'-helpbrowser')) it [to the workspace.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f11-46877.html'%5d,'-helpbrowser')) How is the data stored? Explore the logged data in the ML workspace. Show that it is a structure that reflects the bus hierarchy.
* [optional] Use From and Goto blocks to connect position and velocity to the Bus Creator

## Section 07: More on Subsystems

This section talks more about types of subsystems and how they affect how Simulink simulates your model. Algebraic loops are also covered in the ‘Advanced’ section.

Either continue with the model from the previous section, or [Open "Step\_07.mdl"](matlab:Step_07)

**Standard Instructions:**

1. Change the name of the constant 'Thrust' block to 'Velocity' and set it’s value to [0 0 -0.5]
2. Add a sum block and set the second sign to negative.
3. Difference the desired velocity with velocity feed back from the plant subsystem and label the sum block output as ‘error.’
4. Add a PID controller block from the Continuous library and make ‘error’ its input. Connect the PID block output to the plant model. The result should appear as shown in figure 7.1, below.

1. Double-click on the Controller block and set the controller to PI. Set the gains to 50 and 5.
2. Create a ‘Controller’ subsystem that includes the reference velocity, sum block and PID block. The result should appear as shown below in figure 7.2.

1. Change the Controller subsystem input port name to “Velocity” and the output port to “Thrust.”  The result should appear as shown in figure 7.3, below.  Note: to use the name “Velocity” for the inport name, first change the name of the constant block to “Velocity\_Reference” to avoid a name collision/duplication.

1. Connect Scopes to 'thrust' and 'error.'
2. Show the block execution order: “Format/Block Displays/[Sorted Order.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f13-91940.html'%5d,'-helpbrowser'))” Update the diagram with Ctrl-D.  Double click the ‘Controller’ and ‘Plant’ subsystems and inspect the sorted order of each block.  Note that each block is executed in the same block context.

**Basic sorted order notation:** *s:b*

Where: s=system execution context

b=block position in execution context

**Atomic S/S sorted order notation:** p:b{c}

Where: p = Execution context of parent model

b = block execution order in parent model execution context (0-based indexing)

c = subsystem index. Not used for ordering blocks.

1. Make the ‘Controller” subsystem Atomic by right-clicking on the Controller subsystem and go to Subsystem Parameters. Check the box for 'Treat as atomic unit.’ Update the diagram with a <Ctrl-D>.  Again, double click the ‘Controller’ and ‘Plant’ subsystems and inspect the sorted order of each block.  Notice that the blocks in the atomic subsystem are all executed in their own block context. This is when we explain how an Atomic subsystem is different from a virtual subsystem. Also be sure to point out the thicker subsystem border denotes an Atomic subsystem.

[Open the completed model: "Step\_08.mdl"](matlab:Step_08)

**Advanced Skills:**

* Typically, it’s worth taking a few minutes to talk about algebraic loops in front of the audience. I wouldn’t necessarily make them do it with me but instead have them watch here. I explain a really simple example like x = 3\*x – 5 before returning to the lander model.
* Make the Plant subsystem an atomic subsystem too. Update the diagram with a <Ctrl-D>.   [What error do you get?](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/f7-8243.html#f7-19688'],'-helpbrowser')) Explain what algebraic loops are by looking at the top level model. If the plant is an Atomic subsystem too, then its contents must all be executed in a single execution context, and so it couldn’t be executed unless we know the value of the Thrust input from the controller subsystem. But the Controller subsystem can’t be executed without knowing the value of the velocity input, which means the Plant subsystem would need to have been run to compute this. This is a chicken and egg scenario that Simulink can’t solve unless it makes a guess at one of the input initial conditions. The way to solve this problem is to break the feedback loop with a delay block or integrator.
* To correct the algebraic loop problem, add a Unit Delay block to the feedback signal from the Plant to the Controller. Alternatively, in the Plant Subsystem Parameters, check the box for 'Minimize algebraic loop occurrences.' Update/play model.

* In the Plant Subsystem Parameters, check the box for 'Minimize algebraic loop occurrences.'  Update/play the model. Uncheck the 'Minimize algebraic loop occurrences' box in the Plant subsystem and try checking it in the Controller subsystem. Do you get an error?

* Uncheck the algebraic loop box in the Controller subsystem and change the Plant back to a virtual subsystem.

[Open a model ready for control design.](matlab:Step_08_controls)

**Control Design Skills:**

* Split the velocity signal into X, Y, and Z components and use 3 PID control blocks (or [open existing model](matlab:Step_08_controls)).
* Double-click on controllers and push the Tune button.
* Adjust response time, phase margin, and bandwidth.
* Try other types of controllers (PID, P, PD, etc).
* Discretize the controller and discover what conditions lead to instability in the linearized model.
* Observe cross-coupling by changing one controller and trying to Tune another.
* Change the operating point to a snapshot at 10 seconds into simulation.

## Section 08: Referencing Models

This section introduces the concept of referenced models, how to create them and how to simulate with them.

Either continue with the model from the previous section, or [Open "Step\_08.mdl"](matlab:Step_08)

**Standard Instructions:**

1. Convert the controller to a [Model Reference](matlab:web([docroot%20'/toolbox/simulink/ug/bq_5wvg-1.html'%5d)) subsystem by right-clicking on the controller subsystem and selecting “Convert to Model Block.” Take note of any errors or warnings you get, but continue the conversion. A new, untitled model containing a Model Block will appear as shown below in figure 8.1.

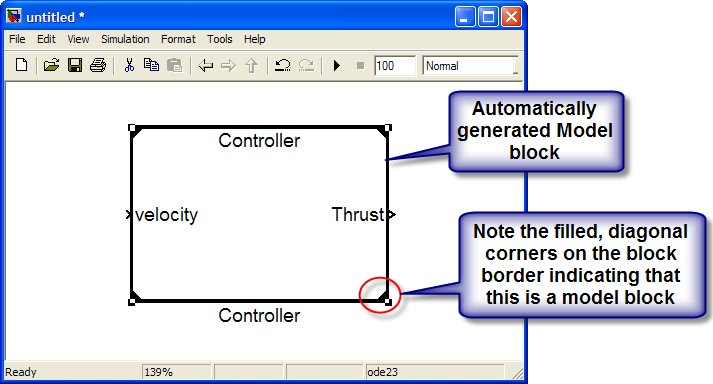


Figure 8.1: new, untitled model containing automatically generated Model Block

Selecting ‘Convert to Model Block’ caused Simulink to copy the contents of the original ‘Controller’ subsystem into a new model, which is automatically named after the original subsystem. Next, Simulink automatically created a new, untitled model containing a ‘Model’ block linked to the new model containing the contents of the original subsystem.

**Note:** When building the new model to contain the contents of the subsystem being converted to a Model Reference, Simulink scans the MATLAB path for any instances of a model with the same name as the original subsystem. If there is, Simulink will add a numeric suffix to the automatically generated model name to avoid any potential name conflicts.

1. Check the MATLAB command window for any errors or warnings and make any changes you were notified of in your configuration settings.
2. Delete the Controller subsystem and drag in the automatically created Model block from the newly created “untitled.mdl” to the falling object model. The result should appear as shown in figure 8.2, below.

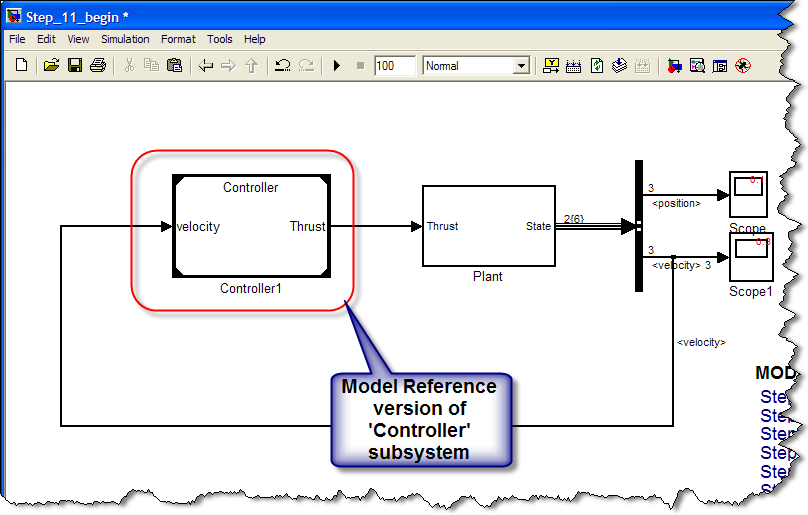


Figure 8.2: falling object model with model reference version of ‘Controller’

1. Double-click on the new Controller block. Double click the ‘Plant’ subsystem to navigate inside. What do you notice about the title bars of each? As discussed previously, there is now a separate model called ‘controller.mdl’ in addition to the ‘Step\_08.mdl’ model
2. Place a scope on the error signal inside the ‘Controller’ model and run your top-level model. The result should appear as shown below in figure 8.3. When you run the model, what do you see on the scope? What do you notice about the time it takes the model to simulate?

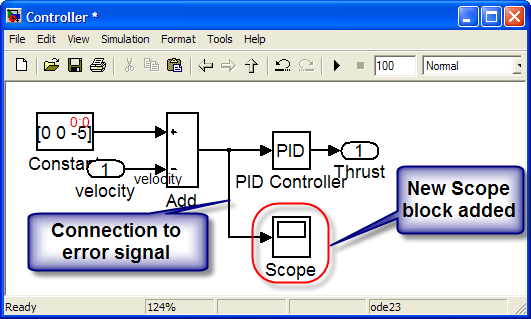


Figure 8.3: adding a scope block inside the ‘Controller’ model

**Simulation time**

You should discover that the first time you attempt to run the model with the Model Reference version of the ‘Controller’ subsystem the simulation takes much longer than when you ran the model previously. This is because by default the referenced model is set up to run in ‘Accelerator’ mode to increase simulation speed, so Simulink is converting the model into executable form to increase simulation speed.

**Scope result**

When the referenced model is in ‘Accelerator’ mode, scopes are disabled and no longer function.

1. Change the Simulation Mode from Accelerator to Normal.
2. Run the model again and note the error. Check Inline Parameters in the Controller's Configuration parameters.
3. Run the model one more time and look at the scope on the controller error.

[Open the completed model: "Step\_09.mdl"](matlab:Step_09)

**Advanced Skills:**

* What files are needed to run your model now? Check out Tools -> Model Dependencies -> [Model Dependency Viewer](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/bq2ifjj-1.html#brbi30n-1'],'-helpbrowser')) and [Generate Manifest.](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/bq2ifjj-1.html#bq2ik30'],'-helpbrowser'))
* What's the [command line way](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/slref/simulink.subsystem.converttomodelreference.html'%5d,'-helpbrowser')) to convert an atomic subsystem to a model reference? Can you go back to a subsystem from a model? – the answer is yes. To see how, [open convertMRToSubsystem.m](matlab:edit('convertMRToSubsystem.m'))
* [optional] Show demo of how a Simulink model can be created programmatically. To run this demo open the script “*create\_Model.m*”, set a breakpoint and single step through the script to build a Simulink model from MATLAB commands

## Section 09: Controlling Sample Rates

This sections talks about how to view and control sample rates within a model.

[Open a model ready to proceed with this step.](matlab:Step_09)

**Standard Instructions:**

1. Open up the Controller model. Double-click the PID block and choose “Discrete-time” under the’Time-domain:’ settings. Select an integration method and set the sample time to 0.5s. (Do you need to change the gain too?)
2. Open up the Configuration Parameters for Controller. These are different now from your original model configuration set. Change the solver to a fixed-step discrete solver with fixed-step size of 0.5s.
3. In the main model, go to Format, Sample Time Display and select colors. Note the error message.
4. Double-click the Controller's thrust outport and set the Sample Time to inherited via the Signal Attributes tab. Update the diagram.
5. Double-click the Controller's velocity inport and set the Sample Time to inherited via the Signal Attributes tab. Update the diagram.
6. Add a Rate Transition blocks from the Signal Attributes library before and after the referenced model. Update diagram

[Open the completed model.](matlab:Step_10)

**Advanced Skills:**

* [optional] What are [some ways](matlab:web('jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/ug/bq2wh78-1.html'%5d,'-helpbrowser')) to organize your configuration sets? Specifically, try using cset=Simulink.ConfigSet to create the configuration sets to be used in each of your models and then load that configuration set in a callback and point your model to it.

## Section 10: Using MATLAB Functions in Simulink

This section shows how to bring MATLAB code into Simulink and how to create bus objects.

[Open a model ready to proceed with this step.](matlab:Step_10)

**Standard Instructions:**

1. Change the Controller such that desired velocity is a function of altitude. To do this, right-click on the reference model, select ModelReference parameters, and change the Model Name to ''Controller\_Step\_11.' This will link the parent model to a pre-built controller model already configured with an eML block to make velocity a function of altitude.
2. In the top-level model change the input to the Controller; supply State instead of velocity.
3. Go to Tools-> Bus Editor and create a new bus named stateBus. It should have two signals in it, one called position with has dimension of 3 and one called velocity which has dimension of 3. Hit Apply and look in your workspace for the new bus object.
4. Save stateBus to a mat file. (use >> save <mat file name> stateBus) Add a ''load <mat file name>'' statement to the main model''s PostLoadFcn.
5. Run your model and see how it looks!

[Open the completed model.](matlab:Step_11)

**Advanced Skills:**

* When would you want to use eML and when would you want to use an [S-function?](matlab:web(['jar:file:///',matlabroot,'/help/toolbox/simulink/help.jar!/sfg/f6-12629.html'%5d,'-helpbrowser')) See below for some considerations and pros/cons of each approach
* Click the link in the controller model (“*Controller\_Step\_11*”) to generate C code from your Controller.
* Click the link in the controller model (“*Controller\_Step\_11*”) to generate PLC code from your Controller.
* Click the link in the model (“*Controller\_Step\_11*”) to generate HDL code from your Controller.

**Comparison of MATLAB Function vs. S-Function**

Generally speaking when deciding between eML or an s-function: use eML unless you need the full Simulink API, you have some c-code already, need something specific to C, or you need something specialized in code generation.

Also consider mentioning the NASA Orion GNC Model Style Guide. It is on our external webpage and contains specific recommendations on when to use Embedded MATLAB, Stateflow, and S-Functions.

 MATLAB Pros

1)      Built-In debugger

2)      Error checking for array bounds and ctrl-c breaks

3)      In 2009a eML uses the Basic Linear Algebra Subroutines (BLAS) for simulation instead of C code as in previous releases.  Using BLAS (which is now multi-core aware) means for many matix operations eML will be much faster than simulating with C, as fast as MATLAB, and numerically equivalent.  It can be difficult to interface to these optimized libraries with C code, so eML has that advantage over C S-functions.

4)      The rich function library and language

5)      Simulates with compiled speed as opposed to the m-file S-function

6)      Easy to modify

7)      Reference M-files on the MATLAB path

8)      Integrate with Simulink and Stateflow

9)      RTW-EC support (TFL, variable naming, etc.)

10)   Integration with custom c-code (eml.ceval)

11)   No TLC required (although LCT generates these automatically)

12)   Generated code is numerically equivalent to ML algorithm

 MATLAB Cons

1)      Does not support states

2)      Only a subset of ML language supported

3)      Doesn’t take full advantage of the Simulink API (it’s only a merged output/update function)

 S-Function Pros

1)      Supports states

2)      Supports reusable functions

3)      Supports full Simulink API

4)      Most versatile, general purpose approach

 S-Function Cons

1)      Required knowledge of C

2)      Requires knowledge of Simulink API

3)      Generally more complex

4)      Harder to write – although this has been mitigated greatly by LCT

5)      Harder to debug – use of run-time debugger not built in 