Tutorial Two: Using Simulink for Simulation and Analysis

This tutorial couples Simulink with Matlab. It is assumed that the reader knows how to launch Simulink, access the Library Browser, input simulation diagrams as well as being able to create and execute mfiles.

The example used isthe model of an armature controlled DC motor. In this tutorial we consider the simulation of the response of the DC motor to a trapezoidal waveform, how to get the data from the Simulink Scope block into the Matlab workspace for subsequent processing and how to obtain the transfer function from a Simulink model.

This tutorial is divided into 3 Sections:

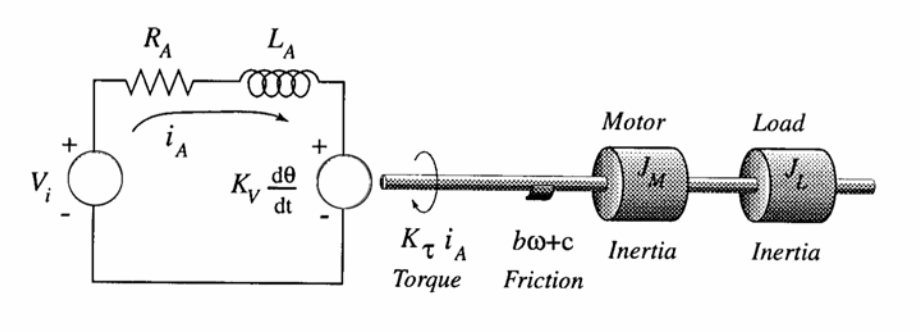
Section I: Simulating the Response of a DC Motor via Simulink. 1
Section II: Getting Data from Simulink into Matlab's Workspace 6
Section III: Numerical transfer function from the Simulink model..... 8

Section I: Simulating the Response of a DC Motor via Simulink.

The objective of this section is to use Simulink to simulate the performance of an armature controlled DC motor when a trapezoidal waveform is applied as the input voltage. Examination of this process will also provide additional insight into the operation of a DC motor.

The objective is to drive the motor with a trapezoidal input voltage. Specifically one that ramps from 0 to 10 volts in the first 100 ms, holds the 10 volts for a second 100 ms and then ramps down from 10 V to 0 V in the third 100 ms interval.

The modeling of the armature controlled DC motor was covered in lecture or may be obtained from a number of texts, the key equations and motor parameters follow



$$v_i = R_A i_A + L_A \frac{di_A}{dt} + K_V \omega$$

$$V(s) = RI(s) + LsI(s) + K_V \Omega(s)$$

Electrical equation

$$T_g = K_T i_A$$

$$T_g(s) = K_T I(s)$$

Coupling equation

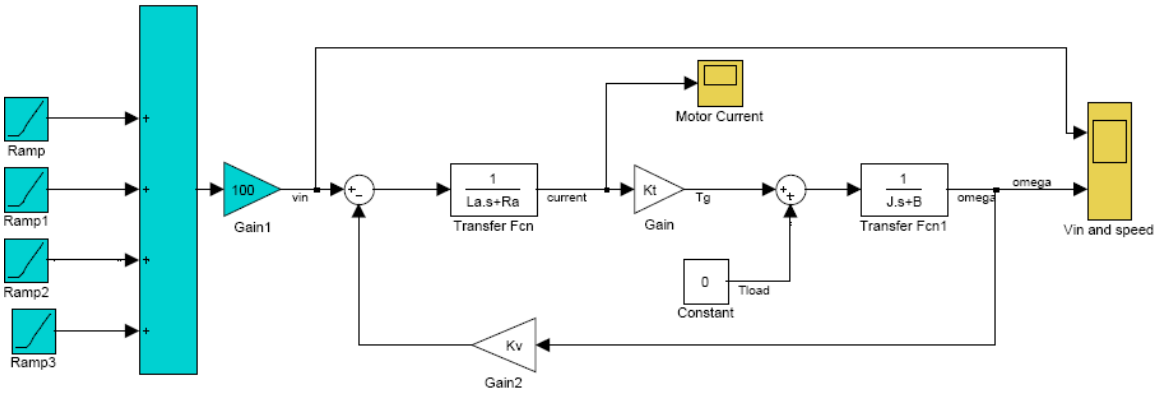
$$T_g = J \frac{d\omega}{dt} + B\omega + T_{load}$$

$$T_g(s) = Js\Omega(s) + B\Omega(s) + T_{load}(s)$$

$$\omega = \frac{d\theta}{dt}$$

Mechanical equation

Use the above equations to create the following Simulink diagram with scopes provided to observer armature current, the input voltage and the motor velocity. Here we set T_{load} to 0 using a constant input block. Values for each block are provided by variables such as J, B etc.

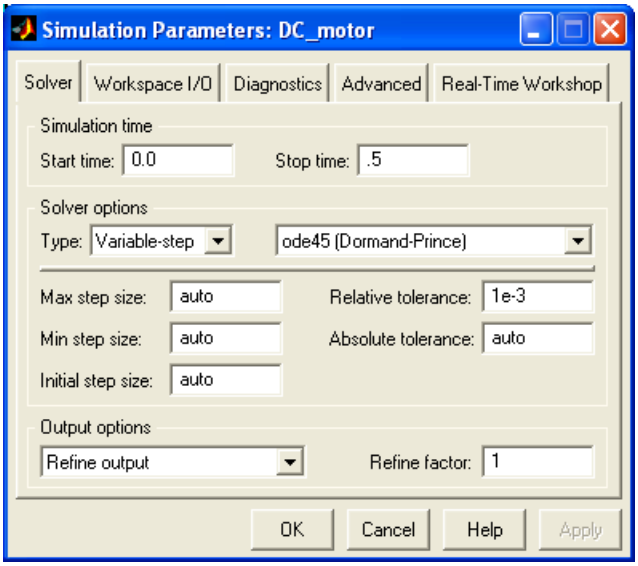


Create an mfile called: **mparms.m** to load the workspace with the variables used in the Simulink model

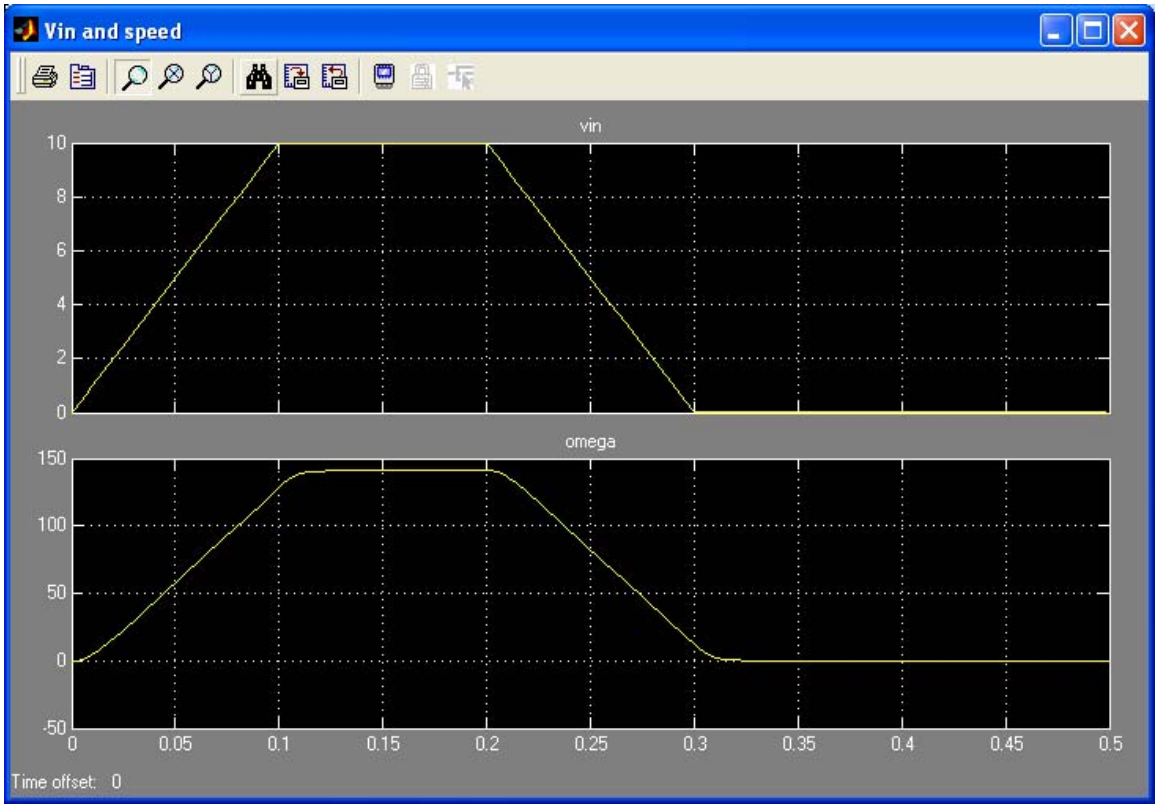
```
% file mparms.m

% mechanical constants
J = 0.0038
```

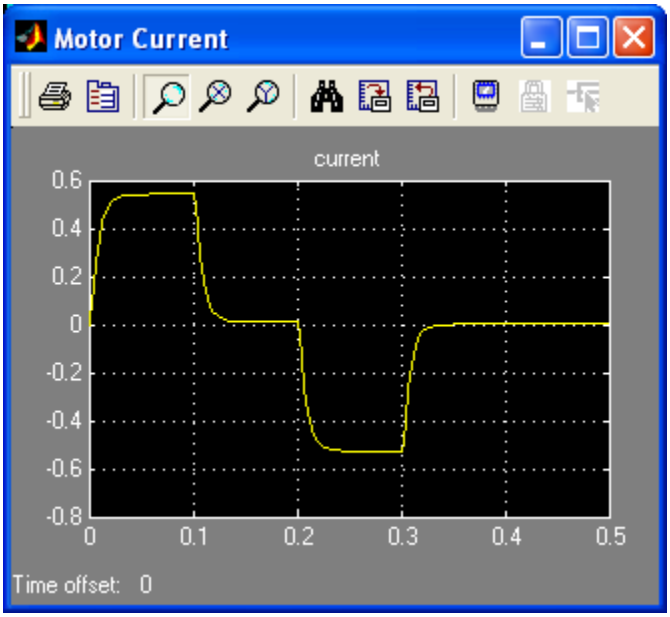
```
B = 9.55e-4
C = 3

%  coupling
Kt = 10.02

%  electrical
Ra = 1.64
La = 3.39e-3
Kv = 0.0708
```

Modify the simulation parameters in Simulink so that the simulation time is reasonable, say 0.5 seconds. Generally you should include more time than the input waveform to see if there is any natural response after the input waveform goes to zero. Also for now use a ode45 solver with variable step.



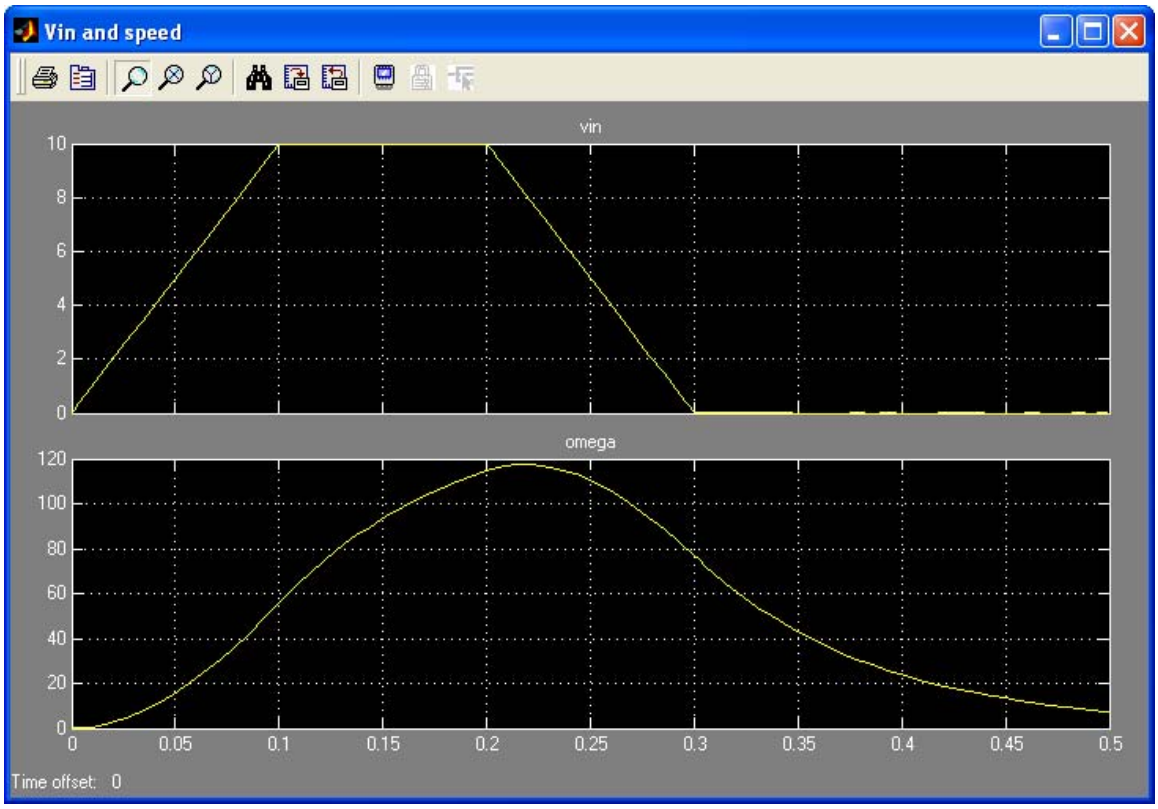
After running the mfile to load variables in the workspace, execute the Simulink simulation and then click on the Scope blocks to open the graphs. You may have to click on the binoculars to see the waveforms.





By placing the model variables in the workspace, if one wanted to see the motor's response to an increase of inertia, all that has to be done is to change the variable J in the workspace and rerun the Simulink model. This response following the next paragraphs.

Two observations should be apparent from the output waveforms (pg 4 and pg 5). **First** that the output velocity, **omega**, looks like a filtered version of the input waveform, **vin**. Recall that the motor model transfer function (from **vin** to **omega**) has two poles, hence the transfer function will filter the spectrum of the input signal based on the location of the poles. If you change values in the model, you change pole locations and essentially increase or decrease the bandwidth or even make the system unstable. The following plot show the input and output waveforms if the inertia J is changed to 10 times the nominal value. Note that the motor shaft continues to move a significant time after the input signal is removed. You should compare this to the nominal values of the motor model on pg 4.

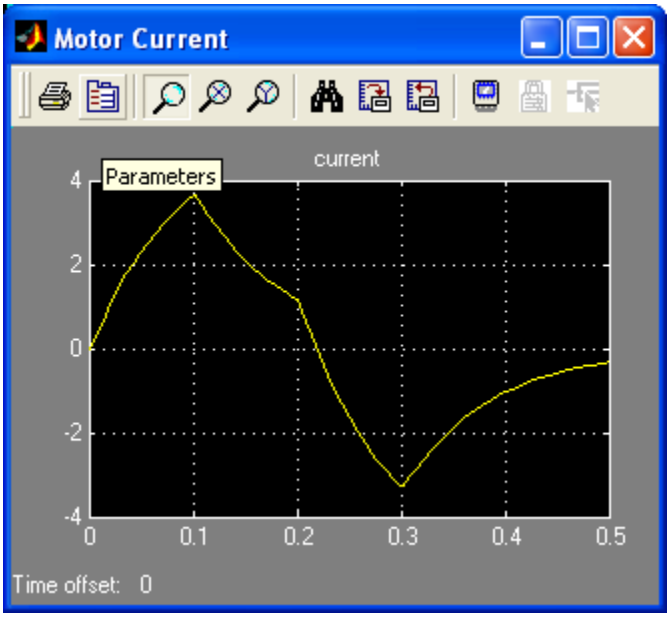


The **second** observation is that the motor current looks like the derivative of **omega (**see pg 4**)**. This is because the torque generated is related to the inertia by the angular acceleration (**omega-dot also called α**) and torque is proportional to current by the parameter **Kt**. Clearly if an external torque load is input or if the viscous damping of the motor, **B**, is increased the current waveform will no longer be as strongly proportional to the inertia.

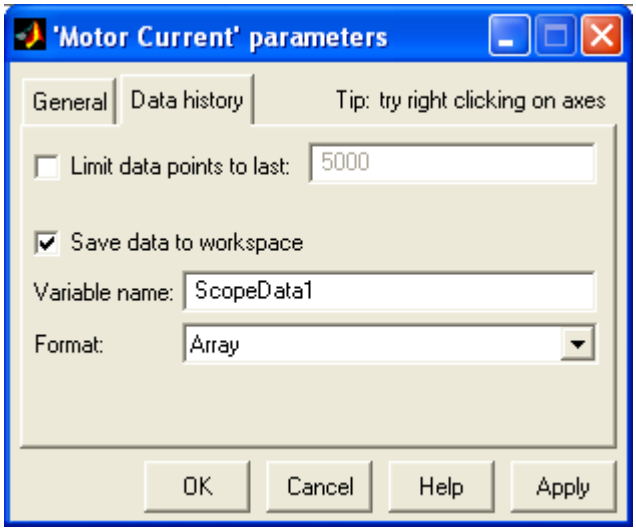
Section II: Getting Data from Simulink into Matlab's Workspace

Sometimes it may be advantageous to obtain the data corresponding to the Scope plots of the Simulink model. For instance we may want to analyze the output data using algorithms implemented in an mfile, or **EXCEL** or may wish to create more detailed/annotated plot for a report.

To do this we will tell the scope block to save its data to the Matlab workspace. For simplicity we will save it as an array. Double click the parameters button on the graph of motor current (this example is for the large inertial load 10*J nominal)



Then go to the Data History tab and set the parameters as shown below.



Execution of the Simulink program will cause an array **ScopeData1** (this could be any name you want) to be created in the workspace. Use the command "**who**" to see a list of variables in the workspace.

```
>> who
Your variables are:
B        J        Kt       La       ScopeData1
C        Jm       Kv       Ra       tout
```

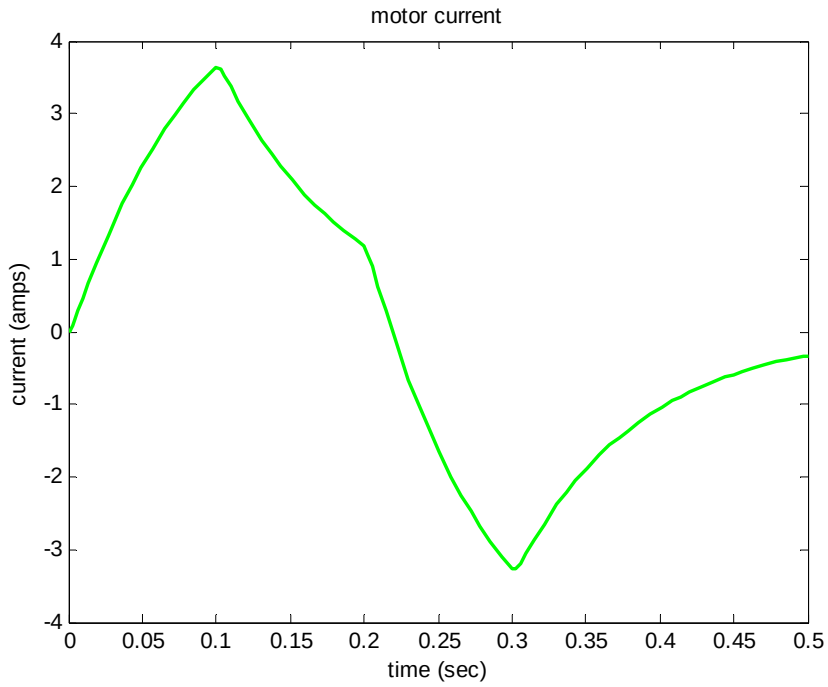
Scopedata1 is a two dimensional array of time and current (90 rows and 2 columns). You can find out the size of a variable by typing:

```
>> size(ScopeData1)
ans =  90   2
```

To plot the current we can execute the following Matlab commands from the command window. Here we want the trace to be green, with a thicker **linewidth** and we want to label the graph and axes. You may need to experiment with linewidth when you import plots into Word. The default produces a trace that is sometimes hard to see.

```
>> plot(ScopeData1(:,1), ScopeData1(:,2), 'g', 'linewidth', 2)
>> title('motor current')
>> xlabel('time (sec)')
>> ylabel('current (amps)')
```

The resulting figure is shown below

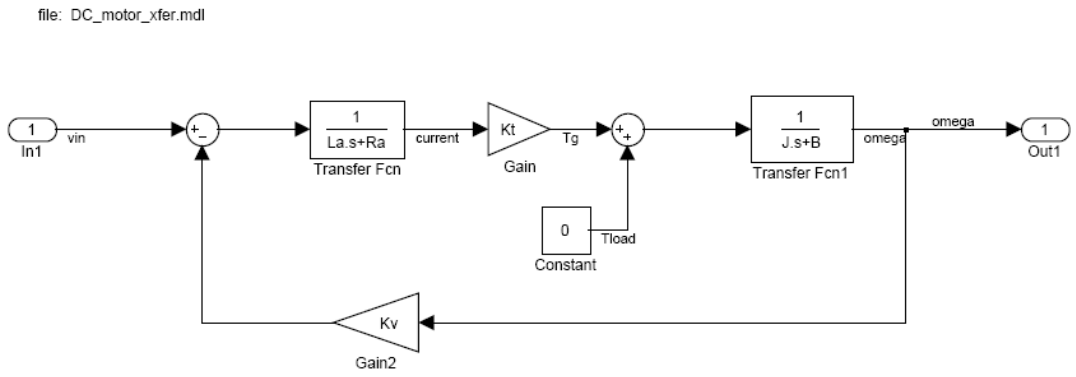


Caution, in the case of multiple traces on a scope, you cannot output the data to an array it must be output as a **structure with time**. So for simplicity one could use multiple single trace scopes to output data if using structures is deemed too difficult or you are not familiar with structures. We will investigate structures in a future tutorial.

Section III: Numerical transfer function from the Simulink model.

The Simulink model of a system can be used to find a numeric transfer function. There are a variety of ways of accomplishing this however we will use a method that works with older versions of Matlab and Simulink and is also compatible with the use of subsystems (which will be discussed in a later tutorial). We will focus on a single input single output transfer function for a linear system (i.e. no non-linear elements or the need to establish operating points) then discuss how to handle multiple inputs and outputs.

Starting with the Simulink mode from **Section I**, replace the input waveform generator with an **in1** block from the **Sources** section of the Simulink Library Browser. Use and **Out1** block from the **Sinks** section to replace the scope input connected to the motor velocity **omega**.



Save the model with a new name. Here we used **DC_motor_xfer.mdl.** Make sure that all the variables needed for the Simulink are loaded into the Matlab workspace to get this solution use the original values. It is generally good practice to click on **Update Diagram** in the **Edit** menu of Simulink to see that all variables are present.

From the Matlab command window type in the following commands (**in bold**):

```
>> [AA, BB, CC, DD] = linmod('DC_motor_xfer');
>> motor_tf = tf(ss(AA, BB, CC, DD))

Transfer function:
      7.778e005
----------------------
s^2 + 484 s + 5.519e004
```

```
>> pole(motor_tf)
ans =
 -300.1398
 -183.8874
```

The first line with suppressed output computes the values of the state and output matrices for the linearized model of the input file. The second line uses these matrices to crease a **ss** object and then the function **tf** to create a transfer function object called **motor_tf**. Finally pole() displays the poles of the model.

If we wanted to include a second input corresponding to Tload, a second input called **in2** would replace the **Constant block** in the diagram. Execution of the code would the generate the following outputs:

```
>> [AA, BB, CC, DD] = linmod('DC_motor_xfer');
>> motor_tf2 = tf(ss(AA, BB, CC, DD))

Transfer function from input 1 to output:
      7.778e005
----------------------
s^2 + 484 s + 5.519e004

Transfer function from input 2 to output:
  263.2 s + 1.273e005
----------------------
s^2 + 484 s + 5.519e004
```

Here if we had examined the elements of AA, BB, CC, and DD we would have found that the BB matrix is 2by2 indicating a second input. The first transfer function is from input **vin** to **omega** while the second is from **Tload** to **omega.** As expected in linear systems they have the same denominator. To access one or the other transfer function you need to use the following syntax

```
>> motor_tf2(1)

Transfer function:
      7.778e005
----------------------
s^2 + 484 s + 5.519e004

>> motor_tf2(2)

Transfer function:
  263.2 s + 1.273e005
----------------------
s^2 + 484 s + 5.519e00
```