Tutorial Three: Scripting Simulink Models With m-files

This tutorial expands on the material previously presented by requiring the use of a script m-file in conjunction with a Simulink model to perform more complicated analysis. The m-file script will execute the model, collect data from the Simulink model and perform additional computations in order to obtain the desired result.

As in *Tutorial two*, this tutorial is designed to provide additional insight into the operation of a DC motor and its model. It contains two exercises. The first exercise is to explore the change in bandwidth of the DC motor model as the inertia, J is varied about its nominal value. To do this we will create an ensemble of bode plots and overlay them on the same plot to visually illustrate the change in the 3 dB bandwidth as J varies. The second exercise is to create the speed torque curves of the DC motor model. Here we will utilize the Simulink DC motor model to determine the steady state velocity for a given set of input voltages and load torques. By collecting data in the appropriate manner the speed torque curves will be generated.

As in all tutorials, it is suggested the reader utilize the "help" command to obtain more information about the functions and/or commands used.

This tutorial is divided into 2 Sections:

Section I: The Effects of Inertia on the Bandwidth of a DC Motor

Tutorial Two illustrated the use of the command **[A, B, C, D]=linmod('model_name")** to obtain the state space representation of a Simulink model. Additional commands allowed one to obtain the transfer function. Plotting the bode plot of the transfer function was covered in ***Tutorial One***.

In this exercise we wish to explore how the 3 dB bandwidth of the transfer function changes as the model parameter J, representing the total inertia of the system varies. For simplicity we will use a lumped parameter estimation of J which represents the inertia of the motor itself as well as any load inertia reflected back to the motor's shaft. In the future we will explore a more complicated case. For this exercise we use the file **mparms.m** and **DC_motor_xfer.mdl** for a single input Va as discussed in ***Tutorial Two***.

It should be obvious that one could execute the following set of commands to obtain the desired result:

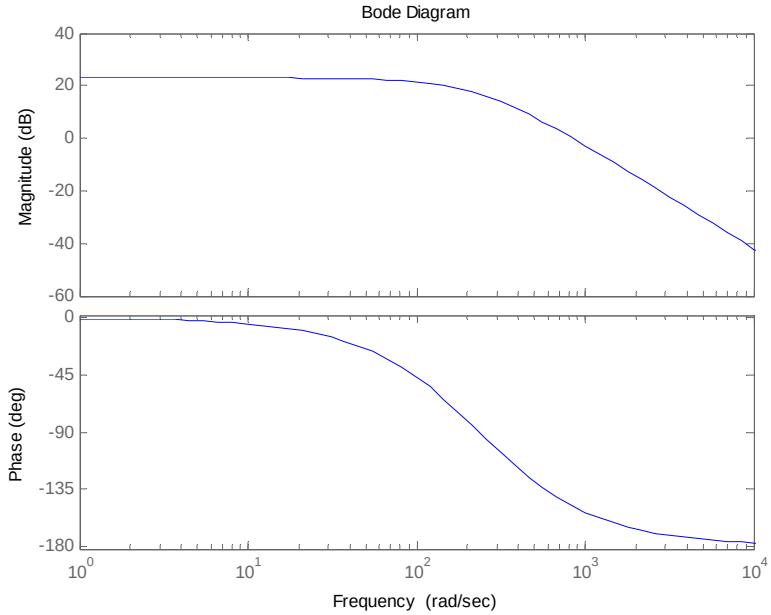
```
>> mparms

    J = 0.0038
    B = 9.5500e-004
    C =3
    Kt =10.0200
    Ra =1.6400
    La =0.0034
    Kv = 0.0708

>> [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)
     -300.1398
     -183.8874

>> bode(motor_tf)
```

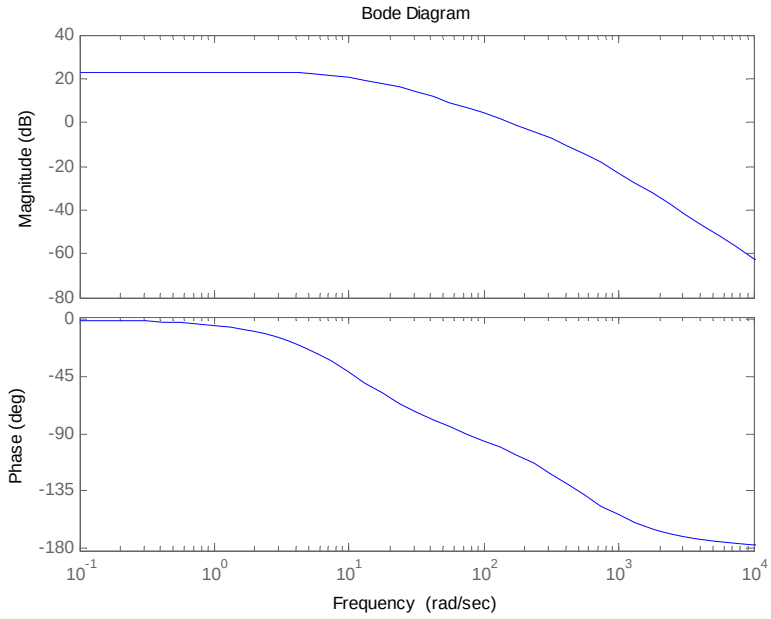
To get the next result change J in the workspace as shown below and execute the same functions as before:

```
>> Jn = J
    Jn = 0.0380
>> J = Jn*10
    J = 0.3800
>> [AA, BB, CC, DD] = linmod('DC_motor_xfer');
>> motor_tf = tf(ss(AA, BB, CC, DD))

    Transfer function:
          7778
    --------------------
    s^2 + 483.8 s + 551.9

>> pole(motor_tf)
     -472.1105
      -11.6905
```

>> bode(motor_tf)

We observe that the poles moved from -300.1398 -183.8874 for the nominal value of J to -482.6348 -1.1436 for J = 10 times the nominal value. This indicates the bandwidth is reduced (not linearly) as the inertia increases. From the two bode plots shown below it is clear that the bandwidth has decreased.

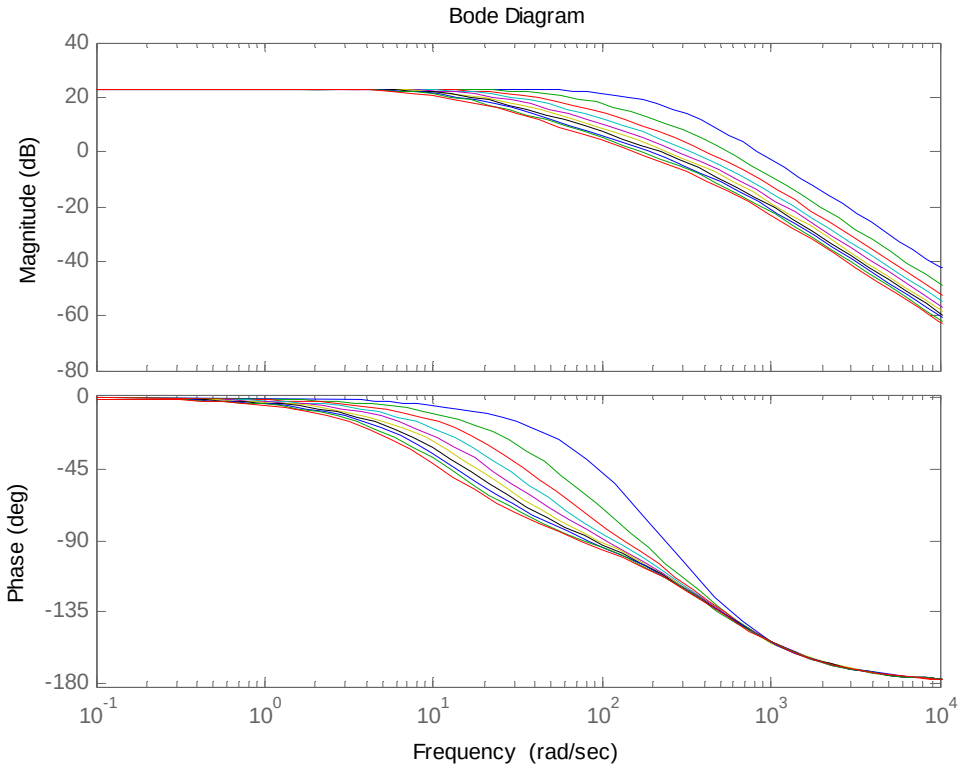




What we would like to do is to automate this process. The following m-file called **vary_J.m** shows one possible way to accomplish this process.

```
%  file:  vary_J.m
%  Uses the Simulink file DC_motor_xfer to generate
%  transfer functions as the parameter J is varied from
%  its nominal value (in mparms) to 10 times its nominal value
%  Bode plots for each value of J are generated and overlaid

mparms; %  load workspace with motor parameters
Jn = J;   % save the original value of J as Jn
for JJ = 1:10
   J= Jn*JJ,  %  compute the value of J to be used
   [aa,bb,cc,dd]= linmod('DC_motor_xfer');
   motor_ss = ss(aa,bb,cc,dd);
   motor_tf= tf(motor_ss);
   bode(motor_tf);  %  plot the bode plot
   hold on %  hold the graph for the next plot
end
```

The resulting plot is shown below with the highest bandwidth corresponding to the blue curve furthest to the right. Matlab automatically cycles though colors using Bode and the hold on command.

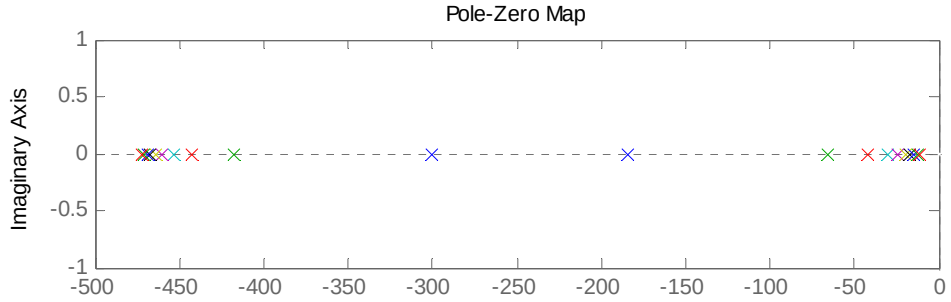


By modifying the m-file we can obtain numeric value of the poles and also look at the pole zero plot. The poles corresponding to each value of J can be obtained by including

a **pole(motor_tf)** without a semicolon in the m-file and also printing out the value of J for each iteration. Tabulated results from the command window are shown below. These were obtained by the painful process of cutting and pasting. In a later tutorial we will explore formatting output to both files and the command window.

Inertia value	Lower Pole	Higher Pole
0.0038	-183.8874	-300.139
0.0076	-66.0411	-417.8604
0.0114	-41.5983	-442.2613
0.0152	-30.431	-453.406
0.0190	-24.0059	-459.8202
0.0228	-19.8250	-463.9927
0.0266	-16.8861	-466.9256
0.0304	-14.7069	-469.1004
0.0342	-13.0262	-470.7776
0.0380	-11.6905	-472.1105

By replacing **bode()** with **pzmap()** in the m-file, one can obtain a pole-zero plot shown below. This is a visual indication (of the above tabulated values) of how the poles of the transfer function move as J varies from Jn to 10*Jn. The blue X's represent the poles {-183.8874, -300.1398 }corresponding to the lowest inertia value which move to the right and left as J increases ending up at the red poles{-11.6905, -472.1105} corresponding to the highest inertia value. Think about this as a root locus plot for varying values of J. It should be clear that the poles do not move linearly with J.

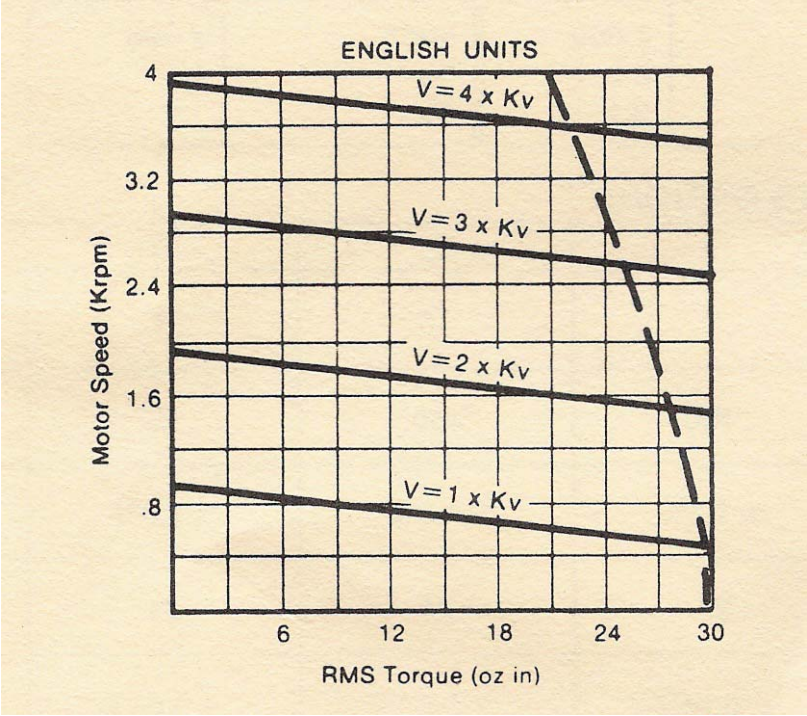


At this point the reader should see the value of using m-files to script the information desired.

Section II: Generating the Speed Torque Curves for a DC motor.

Most motor manufactures provide a set of speed torque (or torque speed) curves as part of the specifications of DC motors. These curves describe the steady state velocity of a DC motor in terms of two inputs: the armature voltage and the constant load torque. Essentially if one knows the applied armature voltage and load torque, the speed can be read off the plot. These curves are useful to determine the speed of a motor given a constant voltage and constant load, typical open loop examples include a motor rotating a fan blade or a motor driving a pump.

The speed vs. torque curves for the family of Torque System 2115 size motors is shown below.



Here the speed is given in 1000 revs/minute, the torque is in oz-in and the applied motor voltage as a factor times the motor parameter Kv. The dotted curve is called the SOAC or safe operating area curve. As long as you are under it the motor will not heat up and cause the insulation of the armature to melt. This curve is normalized to describe different motors in the 2115 line (each having a different Kv value).

For the given speed torque curves, if the applied voltage was 2Kv and the applied torque was 21 oz-in the motor speed would be 1.6 Krpm. It the load increased to 24 oz-in the speed would drop to a little under 1.6 Krpm. This is the point where the 2Kv curve intersects 24 oz-in vertical line. It we were at the original operating point and the voltage dropped to say 1.5v the speed could be found by interpolating between the 2Kv and 1 Kv curves and finding the intersection with the 21 oz-in vertical line.

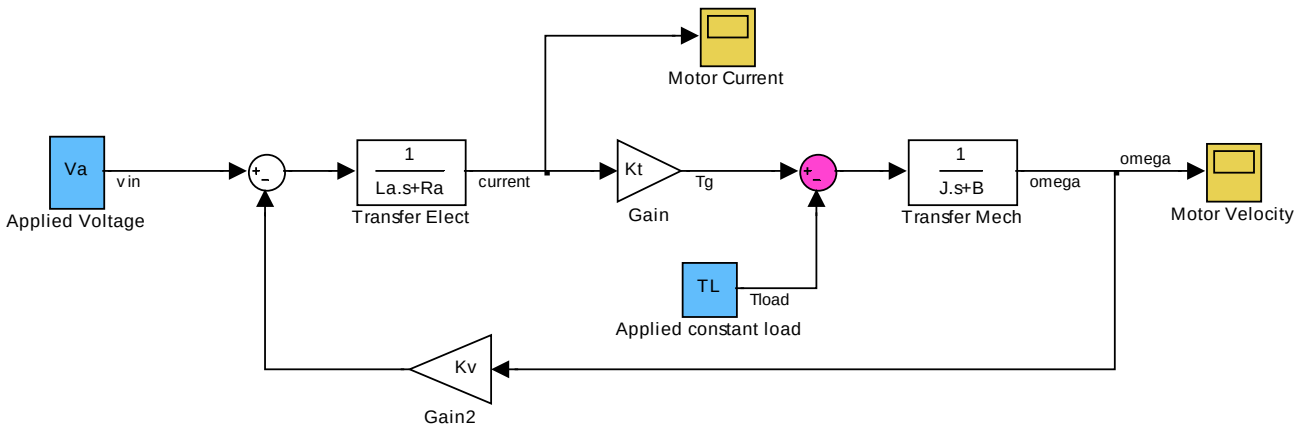
Speed torque curves can be generated analytically if the motor parameters are known or experimentally. In this exercise we will simulate obtaining the speed torque curves by measurement. The Matlab objective of this exercise is to show how a Simulink model generating time data and having explicit inputs can be interfaced to a m-file running a script. This is different than the process used in ***Section I*** in which we did not run the Simulink model but used the **linmod** function to obtain the transfer function.

The process to generate a speed torque curve by measurement is now described. Given the motor to be characterized, a variable DC power supply with a good current capability, and a mechanical loading mechanism that can generate a constant torque (such as a prony brake or dynamometer) perform the following process:

- For each of the voltages in the set $\{V_1, V_2, \ldots V_n\}$ and for each of the load torques in the set $\{T_1, T_2, \ldots T_m)$
  - let the motor reach its steady state velocity (say 60 seconds of operation) and measure its speed
  - record the load torque, motor voltage and steady state speed
- For each value of voltage, plot a curve of measured speed versus load torque. This will result in slanted lines or curves similar to the motor curves shown previously.

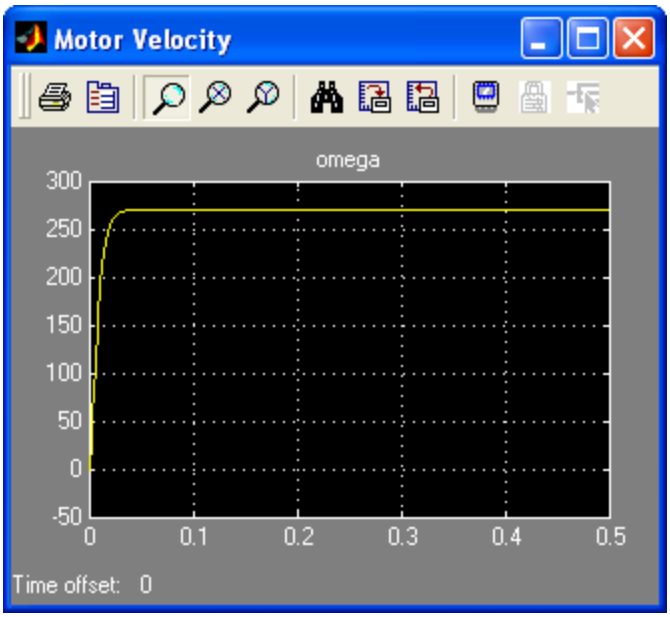
In our simulation we will use our dynamic DC motor model to obtain the steady state value. So essentially we will input two values Va, (the voltage applied to the armature) and TL, (an external load torque) and then wait until the model reaches steady state.

First let us modify our Simulink model so that it looks like the following figure.



Here the inputs, signified by the two blue colored constant blocks are named Va and TL. The magenta summer preceding the mechanical load transfer function block has a negative sign associated with the load torque. This is necessary in that we will define TL as positive values and want then to reduce the torque to the mechanical transfer function. The motor parameters are the same as before and need to be loaded in from mfile **mparms.m**. In addition we have a single input to the Motor Velocity scope. As in ***Tutorial Two***, the scope parameters should be set to save the output to the variable **ScopeData** in an array format and the simulation parameters should be using a variable step and the ode45 solver. The start time should be zero and the end time should be 0.5 sec. Let us call this model **DC_motor_Nov4.mdl**.

Test your system by executing **mparms.m** and setting Va = 20 and TL = 5 in the workspace. Then execute the Simulink model, the velocity plot should look like the following.



To get the steady state velocity, we will choose to use the last element of the array ScopeData. The following command window inputs illustrate this process.

```
>> size(ScopeData)
ans =
   62     2

>> ScopeData(62,2)
270.3303
```

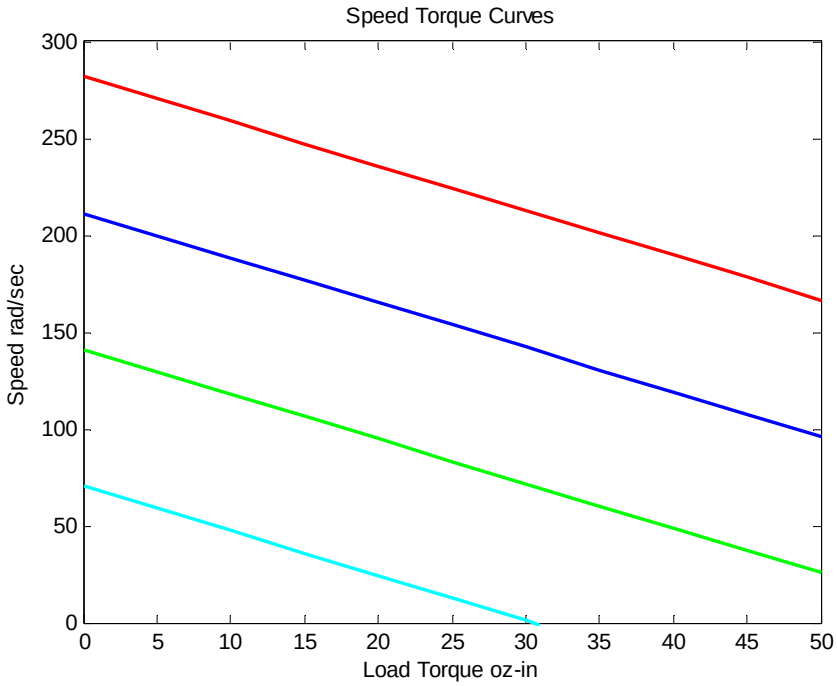
Even though our model includes scopes to view the outputs we will not need them so it is acceptable to save and close the Simulink model once it has been tested.

We now introduce a command that allows use to execute a Simulink model from the command window or an mfile.

```
>> TL = 30
TL = 30

>> sim('DC_motor_Nov4')
>> size(ScopeData)
ans =
   62     2

>> ScopeData(62,2)
ans =
  212.6636
```

Here we tested the process by changing the value of TL and using the **sim()** command to execute the Simulink model. The steady state velocity is now 212.6 rad/sec a decrease from the previous value.

With this information we can now write a mfile script to generate the speed versus torque plots. The resulting plot shows a red curve for 20 volts, blue for 15, green for 10 and cyan for 5 volts.



The listing of the mfile called **speed_torque.m** follows. We leave it to the reader to follow the logic in the code but wish to point out that we have included parameters in the **sim()** command specifically to change the start and stop time to 0 and 1. The arrays **TL_hist** and **Vel_hist** are initialized to empty by **[]**. Data is saved by concatenating new data to the old data using **TL_hist = [TL_hist, TL]**. The remainder of the code is self documented.

```
%file:  speed_torque
%  Generates speed versus torque curve

mparms; %  load workspace with motor parameters
kolor = ['r', 'b', 'g', 'c', 'm'];  %  define a set of colors for the plot
k_indx = 1; %index for color

for Va = 20:-5:5  %  count down due to nature of problem
   % execute code  for each voltage
   TL_hist = [];
   Vel_hist = [];
   for TL = 0:5:50
      sim('DC_motor_Nov4', [0,1]); %run simulation from 0 to 1 second
      [r,c] = size(ScopeData); %  since number of elements may change each time
      final_vel= ScopeData(r,2);  %  get final velocity
      TL_hist = [TL_hist, TL]; % save data
      Vel_hist = [Vel_hist, final_vel];
   end
   plot(TL_hist, Vel_hist, kolor(k_indx), 'linewidth', 2)
   hold on
   k_indx = k_indx +1; %  change color for next value of Va
end

%  fix plot to show only area of interest
axis([0, 50, 0, 300])
%  label the plot
xlabel('Load Torque oz-in')
ylabel('Speed rad/sec')
title('Speed Torque Curves')
```

Note that we have forced the axes only to show positive speed and load torque as is done in industry. If more curves are desired one could change the ***for Va = 20:-5:5*** to say increment in steps of 2 volts by using **20:-2:4**.

The learning point from this section is that one can use the workspace to pass data to a stored Simulink model and then execute the model either form the command line or an mfile. Data from the simulation can be obtained by accessing the variable array associated with the scope block. The Matlab/Simulink environment and commands implement a variety of other ways to accomplish this process.