Massachusetts Institute of Technology

16.06/16.07 Matlab/Simulink Tutorial

Version 1.0

September 2004

Theresa Robinson
Nayden Kambouchev

1 Where to Find More Information

There are many webpages which contain bits and pieces about Matlab. Here are a few good ones:

- http://web.mit.edu/6.003/www/
- http://www.mathworks.com/access/helpdesk/help/helpdesk.html

The second link contains the online version of the Matlab Help files.

2 Matlab

2.1 How to Find Help

Matlab contains an exclusive set of documentation in itself. If you do not know exactly the format of a command or do not remember exactly what it does then you can get a short, but complete description by typing

`>> help <command of interest>`

For example,

```
>> help plot
```

shows a complete description of the plot command which we will also describe later. Sometimes the text which `help` displays does not fit in the window. One way to make it pause, so you can read it, is to execute

`>> more on`
before the `help` command. You can undo the pausing effect with
`>> more off`.
The `help` command requires you to know the name of the command which you are interested in. More often though, you will not know the name of the command. When that is the case you can use `lookfor`. Its format is
`>> lookfor <keyword>`
where you supply a keyword describing what you are looking for. The output is a sequence of commands whose description contains the keyword. You can then review the help of each command to see if any of them does what you want. Try running
`>> lookfor integral`

2.2 Arithmetic Operations

Like most programming languages, Matlab uses the standard arithmetic operators for such operations. Examples of those are:
`>> (5+9)*6`
`>> 3.14/6-1`
`>> 4^6`.
The last example computes 4^6. The normal precedence of operations can be changed with the use of parentheses.

2.3 Built-in Functions

In addition to the normal arithmetic operations Matlab has many built-in functions. They can be used directly in expressions. For example,
`>> 3*sin(pi/4)-6`
`pi` is a special built-in constant for $\pi = 3.14159...$ The more common functions which you will probably need are `sin, cos, tan, exp, log, atan, atan2, sqrt`.

2.4 Formatting the Output

By default Matlab displays five significant digits for all answers. You can show more digits by executing
`>> format long`
Matlab also leaves many empty lines in the output. If you want to have a more compact version of the output
`>> format compact`
will remove the unnecessary empty lines.

2.5 Variables

Like all high level languages, Matlab allows you to save the results in variables. There are no limitations in the variable names with the exception that they must be alphanumeric and always begin with a letter. An example of the use of variables is given below:

```
>> radius=4;
>> circumference=2*pi*radius;
>> area=pi*radius^2
```

If you run the example you will notice that the second line does not produce any output while the last one does. This is controlled by the `;` character at the end of the line. When present it suppresses the output of the result.

Sometimes it is possible to lose track of the variables you have created. The commands `who` and `whos` print information about the variables currently defined. `whos` gives detailed information about each variable, while `who` lists only the names of the variables. If you want to undefine a variable you can do that with

`>> clear <variable name>`.

To clear all variable you can execute

`>> clear all`.

Matlab variables have no type; they take whatever data you assign to them.

2.6 Flow Control

Matlab has built-in flow control for conditionals and loops. Assuming that `tmp` is a variable defined before, here is an example of an `if` statement:

```
>> if (tmp>10)
<do something here>
elseif (tmp>5.0)
<do something else here>
else
<do a third thing here>
end
```

The `elseif` and `else` parts are optional and can be used only when required.

Any expression which evaluates to true and false can be used in place of `tmp>10`. The relationship operators are `==`, `>`, `<`, `~=`, `>=` and `<=`.

Loops can be made with `for` and `while`.

2.7 Vectors and Matrices

One of the great advantages of Matlab is the ease of vector and matrix operations. A matrix can be defined in the following manner:

```
>> Amat=[0 1 2 3;
4 5 6 7;
8 9 10 11;
```

`12 13 14 15];`
The elements of each row are separated with spaces and the rows are separated with semicolons. Sometimes the elements can be separated with commas and the semicolons might be missing, but then each row must be on a separate line. Vectors as particular cases of matrices can be defined in the manner described above. There are special shortcuts for defining some commonly used types of vectors. For example, the command
`>> Bvec=2:5;`
creates the same vector as
`>> Bvec=[2 3 4 5];`.
The colon notation works for equally spaced vectors only. If spacing different than 1 is required, the format of the command changes to
`>> Bvec=2:.5:5;`
which is equivalent to
`>> Bvec=[2 2.5 3 3.5 4 4.5 5];`.
Very often when operating with vectors you will need to transpose it (i.e make a vector row from a vector column and the other way around). This is achieved by the prime operator. If `Bvec` is a vector row which has already been defined, then
`>> Cvec=Bvec';`
gives a vector column `Cvec` containing the same elements as `Bvec`. The transpose operation can also be used on matrices.

When operating on matrices rather than scalars the usual rules apply – matrices to be added/substracted must have the same dimensions, matrices to be multiplied must must have dimensions allowing multiplication. The operators for matrix operations are the same as the scalar operators. For most of the predefined Matlab functions, if you give them an input which is a matrix, the output is also a matrix and the function will be applied to each element of the matrix. If you want to multiply two matrices element by element (not matrix multiplication!) you can use `.*`, for example try the following:
`>> Amat*Amat`
versus
`>> Amat.*Amat`
Two similar operators are `./` and `.^` .

The size of a matrix can be found with
`>> size(Amat)`
where `Amat` is a matrix which has been defined earlier. The inverse of a matrix can be found with
`>> inv(Amat)`
and the eigenvalues and eigenvectors of a matrix can be found with
`>> [eigvectors,eigvalues]=eig(Amat)`.

Useful commands for defining matrices are `eye`, `zeros` and `ones`.

Sometimes you may need to get a specific element from a matrix, let's say the element in the third row, second column. This can be achieved with
`>> S=Amat(3,2)`.

Multiple elements can be indexed at the same time. For example, the first two elements from the third row can be obtained with
`>> S=Amat(3,1:2);`.
Note that the second index is a vector. An equivalent way of indexing would be
`>> S=Amat(3,[1 2]);`.
Now it should be obvious that the indices do not have to be consecutive.
```
>> S=Amat(3,[1 3 4]);
```
is perfectly legal even though it may have limited applications. If you want to index from some place to the end use the keyword `end`. For example,
`>> S=Amat(3,1:end);`.
Matlab will automatically substitute `end` with the index of the last column of `Amat`.

2.8 Scripts and Functions

In addition to typing all commands at the Matlab prompt, you can also save long sequences of commands in files called scripts. The files need to have .m as their extension. You can put comments in the files by beginning a line with %. Everything which follows on that lined is ignored and Matlab does not try to interpret it. To execute the commands contained in a file just type its name (without .m) at the command prompt. Note that the file needs to be in the current directory. If it is not, you can change to correct directory with `cd` and go to the directory containing your file.

If you want to create reusable code you can do that by placing your code in functions. The functions are placed in separate files and contain a few additional lines. The function and the file must have the same name. Here is an example of a function, in a file called "ap_rect.m" which computes the area and the perimeter of a rectangle:
```
% This function returns the area and perimeter of a rectangle
% given the height and width of the rectangle.
function [area,perimeter]=ap_rect(first_side,second_side)
area=first_side*second_side;
perimeter=2*(first_side+second_side);
```
To call the function from the Matlab prompt or from a script you can type
`>> [a,p]=ap_rect(2,3);`
The variable `a` contains the area and the variable `b` contains the perimeter. Functions can have as many parameters and outputs as you want.

2.9 Plotting

Simple plots in cartesian coordinates can be made with the `plot` command. Here is an example for sine and cosine:
```
>> x=0:.01:2*pi;
>> y=sin(x);
>> z=cos(x);
```

```
>> plot(x,y,x,z);.
```
`plot` takes a pair of vectors and uses the first one as the x coordinate and the second one as the y coordinate. One line is generated for each pair of inputs. You can specify the color of the lines and their type like
```
>> plot(x,y,'r--',x,z,'g:');
```
Read the help for `plot` for additional information. Logarithmic plots can be created with `semilogx`, `semilogy` and `loglog`.

Titles can be added to the plots with the `title` command. For example,
```
>> title('This is a very nice graph');.
```
X and Y labels are added with
```
>> xlabel('time');
>> ylabel('distance traveled');.
```

A coordinate grid can be displayed with
```
>> grid on.
```
It can be turned off with
```
>> grid off.
```
A second `plot` command erases the previous content of the figure. To keep it execute
```
>> hold on.
```
There is a matching `hold off` command. To clear a figure use `clf` and to open another figure just type `figure`.

2.10 Solving ODEs

Here is an example of how to numerically solve in the interval [0, 20]

$$\dot{y} + y = 0; y(0) = .1$$

The file odetosolve.m contains
```
function [y2]=odetosolve(t,y)
y2=-y;
```
At the command prompt or in a script execute
```
>> options = odeset('RelTol',1e-4,'AbsTol',1e-5,'MaxStep',.05);
>> [t,Y] = ode45(@odetosolve,[0 20],[.1],options);
>> plot(t,Y)
```
and you will have a plot of a decaying exponent.

2.11 Polynomials

A polynomial is stored in Matlab as a vector. The length of the vector is the order of the polynomial, and the elements of the vector are the coefficients of the terms in descending order of the exponent. For instance, to enter the polynomial $p(x) = x^3 - 2x^2 - x + 2$

we enter the vector `[1 -2 -1 2]`. To evaluate a polynomial at a particular point, we can use the `polyval` function. Try the following sequence of Matlab commands, which plots $y = p(x)$ versus x on the interval[-3,3]:

```
>> p=[1 -2 -1 2]
>> x=[-3:0.1:3];
>> y=polyval(p,x);
>> plot(x,y)
```

Matlab can also find the roots of the polynomial. First put a grid on the plot to estimate the roots. Then have Matlab find the roots of the polynomial. Enter:

```
>> grid on
>> roots(p)
```

Verify that the roots Matlab finds for you are correct by examining the plot.

2.12 Laplace transforms

The Laplace transform operator and inverse Laplace transform operator operate on *symbolic* values in Matlab. You can make a symbolic variable by using the `sym` Matlab function. The following steps find the laplace transform of $f(x) = sin(x)$ and the inverse Laplace transform of $G(s) = \frac{0.1}{0.1s+1}$:

```
>> f=sym('sin(t)')
>> F=laplace(f)
>> G=sym('0.1/(0.1*s+1)')
>> g=ilaplace(G)
```

Other commands you may want to try are `diff(f)` and `laplace(diff(f))`.

2.13 Transfer functions

Matlab can also perform operations on transfer functions. Transfer functions are entered by passing the numerator and denominator polynomials as vectors (as described in section 2.12) to the `tf` function. The following commands create two transfer functions, multiply them, and plot the response of both blocks in series to a step input.

```
>> F=tf([1],[1 2 1])
```

Another way to enter transfer functions is by defining `s` as the transfer function s and then entering transfer functions algebraically:

```
>> s=tf('s')
>> G=1/(s+1)
>> H=F*G
>> step(H)
```

Other commands you may want to try are `impulse(H)` and `rlocus(H)` (which may make little sense to you right now but will prove useful as a check on your work later

in the term)

2.14 Saving and loading Matlab data

You can save Matlab data to a Matlab-readable file by using the `save` command. Without any parameters, `save` saves all the available variables to a file called `matlab.mat`. You can specify the filename by passing it as the first parameter, and save only certain variables by passing them after the filename. To load the data, use the `load` command, which works similarly. Try the following:

```
>> x=7
>> y=[2 3]
>> z=[2 1]
>> who
>> save xy.mat x y
>> clear all
>> who
>> load xy.mat
>> who
>> clear all
>> load xy.mat x
>> who
```

3 Simulink

3.1 Introduction

From the Simulink help files:

> Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. [...] For modeling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations. [...] Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors. You can also customize and create your own blocks.

Simulink interfaces well with Matlab, allowing you to define blocks using Matlab functions and plot output using Matlab tools.

3.2 Goals

The goals of this brief Simulink tutorial are to get you comfortable with Simulink and familiarize you with most of the Simulink actions you will be performing in the lab.

By the end of this tutorial you should be able to:

- Start Simulink, start a new model, and save, open and close a model.
- Create a model using transfer functions, sources, sinks, and basic arithmetic.
- Change the parameters of the blocks and the simulation options.
- Use Simulink output in Matlab, including plotting the Simulink output.

3.3 Example: F-8 longitudinal control system

In this example, we will be designing a controller for the longitudinal dynamics of the F-8. Later in this course, you will be deriving the approximate transfer function of the speed of the F-8 to the elevator angle. We will be taking this transfer function as a given. The input of this system is a desired speed, and the output is the speed. The transfer function of the F-8 speed to the elevator angle is approximately:

$$G(s) = \frac{964}{976s^2 + 11.25s + 1} \tag{1}$$

The controller will take the error in the speed as input and output an elevator angle. The F-8 flight dynamics then convert this elevator angle to an airspeed. We will also model a sensor, which will have a small amount of lag.

We are modeling the behavior of the system when the desired speed suddenly changes. This will be modeled with a step function as input.

1. Start Simulink. Type `simulink` at the Matlab prompt. A new window will open with the title "Library:simulink" and a number of icons, representing libraries of blocks for block diagrams.
2. Start a new model. In the File menu of the Simulink library window, click "New" and then "Model". A model window, labeled "untitled", will appear.
3. Add a step function to your model. In the Simulink library window, double click the icon labeled "Sources". The step function is in the third row and the third column. Drag it to the model window.
4. Add three transfer functions to your model. In the library window, double click the icon labeled "Continuous". The transfer function is the second icon in the second row. Drag it to the model window. We'll require three transfer functions, so drag two more copies as well.

5. Add a scope to the model. A scope allows us to examine the output of the system. Double click the icon labeled "sinks". Find the scope icon and add a scope to the model.

6. Add a sum operator to the model. The sum is the first icon in the "Math Operations" library.

7. Change the sum operator so that it performs subtraction rather than addition. Double click on the sum operator. A new window will appear with the title "Block Parameters:Sum" Change the list of signs to "| + −" and click "OK". The block parameters window will close. Note that the sum operator's icon has changed, and now the bottom input is negative.

8. Change the name of the first transfer function to "Controller". Click on the name, "Transfer Fcn", delete the existing name, and type `Controller`

9. Change the name of the second transfer function to "Aircraft" and the third to "Sensor".

10. Connect the step function to the summer. The small triangle on the right side of the step function represents the output. Click on the small triangle and hold the mouse down. A dashed line will appear from the step function's output to wherever you move the mouse. Hold the cursor over the input triangle to the sum icon's positive input and release the mouse button. The dashed line should become a solid black arrow.

11. Connect the output of the sum to the input of the controller, the output of the controller to the input of the aircraft, and the output of the aircraft to the input of the scope.

12. Flip the orientation of the sensor. Select the sensor block by clicking on it. In the "Format" menu, select "Flip Block." The input will then be on the right side of the sensor and the output will be on the left.

13. Create a feedback loop. Hold down the control key and click somewhere on the arrow between the aircraft block and the scope. A dashed line will appear; connect it to the input of the sensor. Connect the output of the sensor to the negative input of the sum operator.

14. You can move any of the icons by holding down the left mouse button and dragging them. The connecting arrows can also be moved in the same way. Move the icons and arrows until the block diagram is clear and attractive.

15. Change the aircraft transfer function to the F-8 elevator angle to speed transfer function in Equation (1). Double click the aircraft icon. The block parameters of

the transfer function will appear. The numerator and denominator have to be set using Matlab matrix notation. In this course we will be using only single-input single-output (SISO) transfer functions, so both the numerator and denominator will be row vectors. The vectors represent the coefficients of a polynomial in order of decreasing exponent of s. So, for our denominator $976s^2 + 11.25s + 1$, the vector is [976 11.25 1]. Enter `[976 11.25 1]` in the "denominator" field. Enter `964` in the numerator field.

16. Make the aircraft icon wide enough to show the whole transfer function. Click on the icon to select it. Then click on one of the corners and drag it until the block shows all of Equation (1). You may have to move some other blocks to keep the diagram neat.

17. We will model the sensor as having a 0.01 second lag in reporting the speed. You will learn later in the course that a simple lag of τ seconds is represented by the transfer function $H(s) = \frac{1}{\tau s+1}$. Change the transfer function of the sensor to $\frac{1}{0.01s+1}$.

    For now, leave the controller transfer function at its default value.

18. Set the simulation to run for 30 seconds. On the "Simulation" menu, click "Configuration Parameters". In the upper right of the window that opens, fill in the "stop time" field with 30. Hit "OK" to close the configuration parameters window.

19. Open the scope window. Double click the scope icon.

20. Run the simulation. In the "Simulation" menu of the model window, click "Start".

21. Examine the output of the simulation. Return to the scope window. To get the scope to show the most appropriate scale, click the binocular icon in the scope toolbar. Notice how the speed varies in response to a step input. Is this an appropriate way for an aircraft to respond?

22. Try a different controller. Change the transfer function of the controller to $\frac{2.5s}{0.15s+1}$. Note that to get $2.5s$ as the numerator, the vector is [2.5 0].

23. Re-run the simulation and observe the output of the scope. Has the new controller improved the output?

24. Now we would like to plot the output in the standard Matlab plot interface.

25. From the "Sinks" library, add a new "To Workspace" icon. Connect it to the output signal.

26. Double click the "To Workspace" icon and change the "save format" to "array"

27. Run the simulation.

28. Return to the Matlab command window. You may have to press enter to get a Matlab prompt.

29. Type `who` to see a list of variables available to you. Note that there are two new variables: `tout` and `simout`. These are the time and output signals from the simulation.

30. Plot the output signal agains the time. Add gridlines, appropriate axis labels, and a title.

31. Save the Simulink model. In the "File" menu of the model window, click on Save. Browse to an appropriate directory and save the model as "F8.mdl".

32. If you have the time and interest, experiment with the transfer function of the controller and observe the output.

33. Close the model by choosing "Close" on the "File" menu. Exit Simulink by closing the library window.

If you have any questions about Matlab, ask your TAs for either 16.06 or 16.07. Simulink questions should be directed to the 16.06 TAs.