Matlab Tutorial: Basics

Topics:

GETTING HELP

Matlab is a program that allows you to manipulate, analyze and visualize data. MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. The drawback to using Matlab is the specific syntax you will have to learn prior to being able to operate the software efficiently. The purpose of this tutorial is to introduce you to the basics of Matlab and give you the skills you will need to complete the homework in this class. It is important to note that Matlab has many additional functions and features that will not be discussed here, but may be helpful in your future work.

Downloading Matlab

Matlab is available for free for Caltech students and staff.

Log into http://software.caltech.edu/ with your IMSS user name and password.

Click on “Start Shopping”.

Matlab can be found under “MathWorks, Inc” heading. Download the latest version of Matlab.

Opening Matlab

On a MAC or PC, just double-click.

In Annenberg computer lab, open **Terminal** application and type:

```
  matlab
```

Entering Data

Data can be also be entered by typing directly into the Command Window, through the editor in an m-file, or through the Workspace Window by entering values in a spreadsheet format. To store a given data set in the *Workspace*, the data must be preceded by a variable name.

The variable name can be any combination of letters and numbers, but it must begin with a letter and they are case sensitive. Variable names are case sensitive. Also, there is a library of reserved words that cannot be used as variable names. These will appear as blue when typing them in .m files.

Variable types

For example,

```
A=3; % simple variable

A= [1 2 3 4]; % row vector

A = [1 2 3; 4 5 6]  % 2x3 matrix
```

After a variable is entered, it will be displayed in the *workspace*. If the variable is double clicked on in the *workspace*, the value of the variable will be displayed in a spreadsheet in a new window. The value of the variable can also be printed in the command window by typing the name of the variable and hitting enter.

Note: A semicolon placed at the end of a statement will suppress output to the window. It's sometimes useful to take a look at your data, but if you have a lot of data printing out to the screen while your program is running, you may slow things down.

The Command Window

Matlab allows you type commands directly into the command window. You can define variables and plot data from the command window.

If the command window is not open, go to Desktop -> Command Window to open it.

Type: (note: >> is the command prompt, do not type ">>")

```
>> x = linspace(0,5);
>> y = exp(x)
>> plot(x,y)
```

m-files

When writing code in Matlab, it is best to write the code in the editor function of the program. You can open the editor by opening a new or existing m-file. The m-file is the file format executed by Matlab. The file is saved in the *work* directory of Matlab by default, but can be saved in an alternate directory. You can move to another directory by clicking the [...] button and browsing to it.
Writing in an m-file allows you to save your code and debug it much more efficiently. In these m-files, it is also helpful to comment your code.

`% A comment line starts with a percent-sign.` For the work you perform today, it may be helpful to write the basic code in an m-file and save it for future reference.

Open a new m-file, go to File->Open…->Script

Type the following in the m-file:

```
% tutorial.m plots an exponential function in red circles
connected by lines 'ro-'

x = linspace(0,5);
y = exp(x);
figure(1)
plot(x,y,'-ro');
title('Exponential function')
legend('e^x')
xlabel('Time')
ylabel('Population')
```

Save the file as "tutorial.m", make note what directory the file was saved in.

Running m-files

You can run Matlab scripts in several ways. We will run the tutorial.m script from the command window and from the script itself.

To run the script from the command window:

Go to Window -> Command

At the command window prompt, type:

>> run tutorial

If you are not currently in the directory where the script is, you can do one of two things, you can add the directory of your script or you can navigate to the directory where the script is.

Add directory to path

Go to File-> Set Path…
Click on "Add with Subfolders", navigate to the directory of your script and click "Open"
Click on "Save" and "Close"

Now you can run your scripts from any directory in the command window.

Getting help with Matlab functions

You can get help with any Matlab functions directly through the command window or online:
http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

For help at the command window, type;

```
>> help linspace
 LINSPACE Linearly spaced vector.
    LINSPACE(X1, X2) generates a row vector of 100 linearly
    equally spaced points between X1 and X2.

    LINSPACE(X1, X2, N) generates N points between X1 and
X2.
    For N < 2, LINSPACE returns X2.

    Class support for inputs X1,X2:
       float: double, single

    See also logspace, :.

    Reference page in Help browser
```

```
        doc linspace
```

<u>Matrices and Vectors</u>

Matlab was designed to deal mainly with numerical data in matrix or vector form. Due to this design criteria, the some of the standard mathematical operations function as matrix operations and not as you might expect.
Before we can examine how to operate on vectors or matrices, we must know how to define them in Matlab. A vector is either a column or a row of numbers, it is defined as follows:

A matrix:
```
M =[1 2 3; 4 5 6; 7 8 9;];
```

The above example creates a three by three matrix. The only key to defining a matrix is to ensure that each column or row has the same number of elements.

A row vector: A column vector:
```
row =[1 2 3 4]; column =[1;2;3;4];
```

Short-cuts to creating vectors and matrices:
```
Z = (1:5); %means Z = [1 2 3 4 5]
Z = (1:3:10); % means Z = [1 4 7 10]
Z = linspace(a,b,n); %means create a vector with evenly-
spaced points between a and b
Z = logspace(a,b,n); %means create a vector with
logarithmically-spaced points between 10^a and 10^b
zeros(a, b); %means create an axb matrix of all zeros
ones(a,b); %same thing but a matrix of all ones
```

One might ask, why would we want to deal with matrices? A simple example is solving systems of equations.

<u>Matrix Manipulation</u>

Being able to easily manipulate the values, columns, and rows of a matrix will make your life much easier. The following are examples of the syntax used with matrices:

Enter the following matrices into your command window:
```
A=[1 2 3]; B=[4 5 6];
```

Now, try the following matrix manipulations:

Exercise 1:

```
C = [A B] % concatenate rows into a single row
D = [A;B]  % concatenate A and B into two rows

E = D(1,2) % get element in row 1, column 2
F = D(2,2:3) % get elements in row 2, columns 2 and 3
G = D(:,3) % get column 3
H = D(1,:) %  get row 1
I = H' % transpose
```

Matrix addition and subtraction

These commands function normally and do not require any additional syntax to have the operation preformed on variables in your workspace. It is important to note, addition or subtraction is performed between elements in the same position in the array or matrix. Therefore, you will need to ensure the matrices you are operating on have the same dimensions.

Exercise 2:
Enter the following variables into your command window:
`A=[1 2 3]; B=[1;2;3];`
Can you subtract A from B? Can you add A to A?

Matrix multiplication and division

'*' and '/' are for matrix multiplication and division, and the dimensions of the matrices must be compatible.
'.*' and './' do element-by-element multiplication and division.

So, [1 2 3].*[4 5 6] gives [4 10 18]
A^2 means the cross-product of A with itself, and A must be a square matrix.
A.^2 means square each element of A.

So, [1 2 3].^3 gives [1 8 27] but [1 2 3]^3 gives an error. Forgetting the dot is one of the most common bugs!

Exercise 3:
Enter the following data into the Matlab command window.
`A=[0 2; 1 4]; B=[1 3; 2 6];`
What is A*B? What is A.*B? What does A.^-1 produce verses A^-1?

Plotting Data

Given a set of data, it can be useful to display it graphically. This is where the plot function is useful. There are many different plot functions in Matlab that are meant for varying types of data. An overview of the different types of plots can be found at:

http://www.mathworks.com/help/techdoc/creating_plots/f9-53405.html

The following is the general syntax for plotting:

```
plot(independent_var, dependent_var, ‘formatting for data’)
```

In general, it is possible to plot multiple sets of data on a given plot by either entering an additional set of independent and dependent variables following the first set in the parentheses as follows:

At the command window, type the following:

```
>> x = 0:.2:20;
>> y1 = sin(x)./sqrt(x+1);
>> y2 = sin(x/2)./sqrt(x+1);
>> plot(x, y1, x, y2)
```

Multiple data sets can also be plotted in the same figure by typing the command *hold on*.

```
>> plot(x,y1);
>> hold on
>> plot(x,y2);
```

If you have several plots active at a given point in time, the figure may need to be stipulated before using the plot function with the following syntax:

```
figure(figure_number)
```

The third potential entry in the plot function is the option to format the way in which the data is plotted. This is normally just a combination of a letter and a symbol. The letter will designate the color of the line or symbol used in plotting the data. The symbol will determine if a line is plotted or if each individual point is plotted with a symbol. The syntax for color and symbol is shown on the attached cheat sheet or can be found in the plot help listing.

Additional syntax relating to plotting is listed below:

Displays a label on the x-axis:

```
xlabel(‘Time in seconds’)
```

Displays a label on the y-axis:

```
ylabel(‘Distance in meters’)
```

Puts a title at the top of the figure:

```
title('Distance vs time')
```

Exercise 4:
Generate a time scale from 0 to 100. Then produce an array of corresponding values for the function y=1-exp(-t./100). Plot the data with labels on the axes and a title.

Function Files
Function files are a specific class of m-files. They are normally written as stand alone functions. This allows them to be called by the user on multiple occasions, but it can have different sets of data applied to it.

A function can have one or many inputs and outputs and are called in the following ways:

```
out = functionname(in1, in2, ..., inN);

[out1, out2, ..., outN] = functionname(in1, in2, ..., inN);
```

Try creating a separate function file and then calling it. For example, the following code could be written in a separate file called "myfunction".

Go to File-> New-> Function

```
function F = myfunction(x,y,z)

% this function adds up 3 numbers and returns the answer

F = x+y+z;
```

Save the file with the same name as your function "myfunction".m

Now, you can call this file from the command window or from an m-file like this:

```
>> a = 1
>> b = 2
>> c = 42
>> sum_abc = myfunction(a,b,c)
```

<u>If/Else, For, While Statements</u>

Three types of loops exist in Matlab. These loops function as a means of selectively operating on variables in your workspace. These loops are IF/ELSE, FOR, and WHILE. An IF/ELSE loop performs a specific operation if a condition is true. If the condition is false, the loop will then proceed to the else condition. If this condition is not met the loop will end by performing the action following else, if one exists. Example syntax is shown below:

```
if (1>2)
    'false'
elseif (1 == 1) || (1<2)
    'true'
end
```

Assignment vs. Equals is important in loops.   An assignment is a = b and means assign value b to variable a. The equals operator is a == b results in 1 or yes if they are equal, 0 or no if they are not.

The FOR loop performs a series of operations a known number of times. The example code below illustrate the syntax:

```
x = 10
for i=1:1:10
    x = x-1
end
```

The final type of loop is a WHILE loop. It requires that a series of operations occur while the condition applied to a specified variable is correct.

The example syntax is:

```
i=10;
while i>5
    i=i-1
end
```

< less than, > greater than, >= greater than or equal to, <= less than or equal to, == equal, ~= not equal, & and, | or, ~ not

Note, == is a test of equality, 1 if yes, 0 if no. = is used in assignment.

Exercise 6:
Given the matrix: M=[1 1; 1 3;]; Use loops to determine how many of the entries are greater than 1 and the location of all qualifying entries (the corresponding row and column).

Solving Ordinary Differential Equations

There are a number of preprogrammed functions that you can use to solve your ODE or system of ODE's in MATLAB: ODE23, ODE45, ODE113, ODE15S, ODE23S.

ODE45 is based on a Runge Kutta (4,5) formula and it is best first try this function first. However many biological systems are "stiff" and ODE45 will take a long time to generate a solution and may give wrong answers. Try ODE15S or ODE23S instead (S is for stiff).

Solving an ODE

You first need to write a function of the form and save the function in a separate .m file, remembering that the file must have the same name as the function name. Then you call the function from passing it a vector representing a time range, initial conditions, and your solution vector.

Let's start with a simple pendulum example:

du1/dt = u2
du2/dt = -wsin(u1)

Open up a .m file and type:

```
function dudt=pendulum(t,u)

% A simple pendulum equation

w=16;
dudt(1,:) = u(2);
dudt(2,:) = -w*sin(u(1));

end
```

Save the file as "pendulum.m"

Now call the differential equation solver from the Command Window or another script file.

```
clear all

% Time span
tspan=[0,2*pi];

% Initial condition
u0=[pi/4;0];


[t,u]=ode45(@pendulum,tspan,u0,[]);

plot(t,u(:,1),'b-', 'LineWidth', 2)
xlabel('time')
ylabel('angle')
```

General notation for solving differential equations

For the function file, (where the differential equation(s) are defined):

Write a function of the form:

```
function dxdt = functionname(t,x,parameter 1,parameter
2,etc.)
```

Write the differential equation
`dxdt` = some function of: x, parameters; (end with a semicolon)

Save this function with the same name as the functionname above.

To call the function from the main Matlab program:

```
[t,X]=ode45(@functionname,t,IC,options,para meter1,
parameter 2, etc.);
```

- IC is the initial condition, you must assign IC, or whatever name you choose for it, a value in your program file before you run the above function call
- For options, normally type in empty brackets: []
- t is your time vector that you need to define previously
- X is the solution vector for your unknown

GETTING HELP

Use the `lookfor` *somephrase* to find a matlab function with descriptions containing the search phrase.

Example: `lookfor` inverse

Full documentation is online at

http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

Or email the TAs. be150-tas@caltech.edu

This tutorial is based in part on the MIT Matlab tutorial for BE course 20-320.