Basic Matlab Tutorial

E. Pachepsky[*]

This tutorial is an introduction to Matlab (version 6.5) for people without much programming, mathematical or Unix background. Matlab is an easy software package to use even without much knowledge. That is what makes it so convenient. In this tutorial, some basic and very useful functions are described. These should get you started. Having understood a few basics, it will be pretty easy to expand your knowledge using Help, the internet and manuals.

What is Matlab?

Matlab is a software program that allows you to do data manipulation and visualization, calculations, math and programming. It can be used to do very simple as well as very sophisticated tasks. We will start very simple.

Starting/quitting

To start Matlab, click on the 'Start' button on the left bottom of the screen, and then click on 'All Programs', then 'Math and Stats', then 'Matlab'. A window will pop up that will consist of three smaller windows. On the right there will be a big window entitled 'Command Window'. On the left there will be two windows, one entitled 'Workspace' and another one 'Command History'.

In addition, on the top there is a usual bar with 'File', 'Edit', etc. headings. You can use these as you would in any other software (Word for example). Click on 'File' and look at the available commands there. Do the same for all the other headings. Note that the last heading is 'Help' (very useful!). Therefore, if you are stuck you know where to look.

[*] pachepsk@lifesci.ucsb.edu; Department of Ecology Evolution and Marine Biology, University of California Santa Barbara, Santa Barbara 93106.

To use Matlab, you will mostly be typing in the ‘Command window’. Click on the Command window. Its outline will become dark grey (that’s how you know that you can type into that window). A cursor will start blinking on a line right after ‘>>’ (this is called a prompt).

Let’s start using Matlab by quitting it. In the Command window type `quit` (the letters should appear after the prompt) and hit enter. Matlab will close. Now you have used Matlab!

Help!

Start Matlab again. Let’s explore the Help bar. Look at the third option under the Help heading ‘Using the Desktop’. Click on it. A Help window will come up with a list of topics describing the Desktop. Click on ‘What Desktop Is’. On the bottom of the page that will come up you will find explanations of the buttons, windows and options available on the desktop. Scroll to the bottom. You will see text ‘Drag the separator bar to resize window’. Let’s try that.

Switch to the Matlab window. (To do this, look at the taskbar on the bottom of the screen and find an icon with a little orange and green hill on it that says ‘MATLAB’. Click on it.) Move your mouse to the space between the ‘Command window’ on the right and the windows on the left. The mouse should take a shape of an arrow with two points. Press the left mouse button down and move the mouse left and right. This should move the boundary between the windows.

If you have questions about the desktop in the future, you can go to Help/Using the Desktop for the answers. However, the most useful help option under the Help heading for you now is the second option called ‘MATLAB Help’. Click on it. This will take you to the main page of Matlab help. The categories are on the left, the main text is on the right. You can always go to this page if you have a question. Two very useful features on this page are ‘Index’ and ‘Search’. These are bars on top of the left window that contains the categories. In the ‘Index’ you can search for available functions in Matlab. In fact, the Index is like an index at the back of a Matlab manual book.

For example, click on the Index bar. In the window under ‘Search index for:’ type `logarithm`. The text on the window below will jump to the entry under ‘logarithm’. It has several subheadings. Double click on the subheading ‘natural’ in that window. On the right you will get the description of the `log` function in Matlab, with syntax and example.

The Index bar is very useful to look up the syntax of a function or to see if a particular Matlab function exists. Another useful Help feature is the Search bar (located to the right of the Index bar). That allows you to search Matlab documentation more thoroughly. Therefore, if you cannot find something in the Index, you might want to try using Search.

Close the Help window, and get back to the main Matlab window.

Matlab as a super-calculator

Click on the Command window. Type `2+3` at the prompt and then hit enter. The following will come up below the prompt:

```
ans =

     5
```

As you can see, you can use Matlab as a basic calculator (although that’s not the most efficient use for it).

Notice that there is a new entry in the top left window entitled ‘Workspace’. There is now an entry `ans` there of size 1x1. `ans` is a **variable**. This means that it’s a string of text that has a value (number) assigned to it. To see this, type `ans` at the prompt and then hit enter. As you can see Matlab again returns `ans = 5`, i.e. it remembers that `ans` holds a value of `5`.

`ans` is a special name for a variable in Matlab. It is assigned the value of the answer to the expression that you type at the prompt.

You can create your own variables. For example, type in `x=10`. Now Matlab has another entry in the 'Workspace' window called `x`. Now if you type `x`, Matlab will know that its value is `10`. For example, type `x+5`. Matlab will give you the correct answer `15`.

Matlab has all the math functions that a calculator may have and many more. For example, you can find x^2. Here, we need to learn a bit of notation. To raise x to the power of 2 in Matlab you type `x.^2`. You will get, predictably, `100` since you assigned a value of `10` to `x`.

Among the most familiar functions, Matlab has `sin`, `cos`, `exp`, `log` functions. For example, to find e^2, you type `exp(2)`. You should get `7.3891`. As you can see, to use a function, you put the argument of the function in the parentheses after the function name (without a space between them).

Matlab as a mathematical tool

So far we have used common mathematical functions. However, Matlab allows you to define (and evaluate) your own functions as well. For example, lets define a function $f(x)=x^2+1$. To do this, simply type in `f=x.^2+1`. Since `x` has a value of `10`, the answer is `101`. You can now change the value of x. For example, type `x=5` and enter, then type `f=x.^2+1` again. You should now get `26`.

However, you don't want to type the expression for f(x) every time you want to change the value of x. You might want to define a function f(x) for a range of values of x. To do this in Matlab we need to make x be a range of values. For example, suppose we want to make x to go from 1 to 10. To do this in Matlab, you type `x=1:10`. You will get

```
x =

     1     2     3     4     5     6     7     8     9    10
```

as a result. Now, x is a list of values from 1 to 10. If we now type `f=x.^2+1`, we will get 10 values of `f` for each value of `x`. In other words, we have defined `f` as a function of `x`.

Right now the difference between two consecutive values of x is 1. To change this , we put the step between two consecutive values between the maximum and minimum values, i.e. type `x=1:0.5:10`. You will get

```
x =

  Columns 1 through 8

    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000
4.0000    4.5000

  Columns 9 through 16

    5.0000    5.5000    6.0000    6.5000    7.0000    7.5000
8.0000    8.5000

  Columns 17 through 19

    9.0000    9.5000   10.0000
```

Now x still ranges from 1 to 10, but now it takes on 19 values with a step of 0.5. Since we redefined x, we now need to redefine f(x) as well. This means that we must again type `f=x.^2+1`. f(x) now has 19 values as well.

Plotting basics

One thing we want to do to with functions is plot them. Matlab is a very good tool for that. For example, to plot `f` as a function of `x`, type `plot(x,f)`. A new window will come up with a plot of f(x) as a function of x. Matlab has many features for plotting. We will now learn a few of them. First of all we want to define our axes. This is very simple to do. To define the x-axis, type `xlabel('x')` in the Command window. Now switch to the window with the figure. You will see a label on the x-axis. Let's do the same for the y-axis. Predictably, to do this you need to type `ylabel('f(x)')`. Now, if we want to put a title on our graph the command it `title('Function f')`. As you can see, a lot of the commands are quite intuitive.

Matlab as a tool for data analysis

What about looking at data in Matlab? Usually, data comes in tables. For example, you could have some observation as a function of time, such as the average temperature as a function of a month. This data will have two columns, one for the month and another for the value of a temperature. Or you could have several replicates on an experiment where the average weight of some organisms was observed in environments with different food levels. This data can be recorded as a table with the number of rows equal to the number of replicate experiments and the number of columns equal to the food levels tested.

A mathematical term for a table is a **matrix**. (From now on you can understand the term matrix as table.) Matlab deals with matrices very well. Let's create a matrix of ones in Matlab. To do this, we need to know how many rows and columns we want in a matrix. Suppose we want to create a matrix of ones with 2 rows and 3 columns. To do this, type `ones(2,3)`. The Matlab will return

```
ans =


     1     1     1
     1     1     1
```

Suppose we want to store this matrix in a variable. To do this, type `M=ones(2,3)`. Now, `M` will appear in the Workspace window (notice that its size, 2 by 3, is also stated). The size of a matrix is often referred to as its dimensions. For example, `M` is 'a matrix of dimensions 2 by 3' or simply 'a 2 by 3 matrix'.

Whenever you want to change a variable or remove a variable, you can use command `clear`. If you type `clear M`, this will remove `M` from the Workspace. Try it. Anytime that you have made a mistake defining a variable or a function, or when you reuse the same variable name, it is a very good practice to use the `clear` command. If you want to get rid of all the variables in your Workspace, you can simply type `clear`.

You can also construct your own matrix by typing it in. For example, if you type `M=[2,5;9,7; 4,3]`, you will get a 3 by 2 matrix:

```
     2     5
     9     7
     4     3
```

As you can see, to define a matrix in Matlab, you surround the entries by square brackets. Commas separate column entries and semicolons separate row entries. To look at the whole first row of the matrix `M`, you type `M(1,:)`. Here, the colon means 'show me all the entries in the row(s) indicated'. To look at the whole second column, you type `M(:,2)`. If you want to see only the second and third rows of M, you type `M(2:3,:)`.

Let's work on an example. Suppose you have observations of average temperature as a function of month. Let's generate the data to represent that. As we do this, we will learn several more useful function of Matlab. First, we need to generate a column of months. We already generated a row of numbers from 1 to 10 in the previous section. Now, we need to make a column. Notice that a row is just a matrix of dimensions 1 by n (where n is the length of the row). Similarly, a column is a matrix of dimensions n by 1. These are called **vectors**. Since we know how to make a row, let's start with that. Type `month=1:12`. You will see

```
month =

     1     2     3     4     5     6     7     8     9    10
11    12
```

To make this row into a column, we will **transpose** it. To transpose a matrix means to flip the values in the matrix so that the first row becomes the first column, the second row becomes the second column etc. Therefore, if you transpose a row, you will get a column. To do this type `month=month'`. The apostrophe tells Matlab to transpose month. You will get

```
month =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
```

This is what we wanted. Now, we need to generate values of average temperature for each month. For educational purposes, we will do this naively by picking random numbers between 0 and 80. Matlab provides a convenient function `rand` to generate random numbers between 0 and 1. Try it by typing `rand` at the prompt several times. You will get a different number from 0 to 1 every time. To generate random values between 0 and 80, we can multiply a random number between 0 and 1 by 80. To do this type `80*rand`. Now, we need to generate 12 of these numbers. There are several ways of doing this, we will learn two of them. (In fact, there are several ways of doing almost everything in Matlab. As long as you can do it any one way, it will suffice for most work that you will be doing in the course.)

We need to generate 12 random numbers between 0 and 80. We know how to generate one of them (the command is `80*rand`). Now, we need to string them together. To do this, we will use a new variable called `temp` (for temperature). `temp` should be a column. A column is a matrix of dimensions n by 1. Therefore, the first entry in the variable `temp` will be located in the first row and first (and only) column. To express this in Matlab type `temp(1,1)=80*rand`. The second entry in the column will be located in the 2nd row and 1st column, therefore type `temp(2,1)=80*rand`. You will get the answer giving two random values stored in `temp`. However it is tedious to write all the twelve values by hand. (If you think that twelve values is not that many, imagine having to do it 100 times.) Therefore, we will use a bit of programming, so that we can be lazy.

A bit of programming

`for` loops

Suppose we want to do something several times. You can do this by using a very convenient structure called a **loop**. A loop loops (hence the name) through some commands several times. Getting back to our example, we want to perform a command `temp(i, 1)=80*rand` twelve times, where `i` is a number between 1 and 12. To do this in Matlab you can type:

```
for i=1:12,
temp(i,1)=80*rand;
end
```

Now type `temp`. You will get a row of 12 random numbers between 0 and 80.

Now, what has just happened? You made a `for` loop. A `for` loop performs an operation several times. As you can see, it starts with the word `for` and ends with the word `end`. Let’s translate the line of code to English: `for i=1:12` means let `i` be 1, then 2, then 3, etc. until `i=12` and perform the command `temp(i,1)=80*rand;` for each value of `i`. Note that there is a comma after `12` to signify that what follows is the command you want to perform. End a command with a semicolon. Then you need to let the `for` loop know when to end, and you do this by typing `end`.

Errors

Getting the commas and the parentheses right in Matlab is very important. Matlab (like any other software) is stupid, and if you make even a small syntax error, it will give you an error. Let’s try that.

Retype the previous line but omit the semicolon:

```
for i=1:12, temp(i,1)=80*rand end
```

You will get an error in red:

```
??? for i=1:12, temp(i,1)=80*rand end
                                  |
Error: Missing operator, comma, or semicolon.
```

Matlab is complaining about your syntax. In fact, whenever you get an error, you should first check your syntax.

Getting back to our example, we now have a column of temperatures. We are ready to create our artificial data set by putting the month column and the temperature column together by typing `data=[month,temp]`. The outcome will be a neat table of month and temperature. This is the kind of dataset you might be working with.

Loading files

However, often you will have your data in a file in Excel or text format. The easiest format to use with Matlab is a text format with extension `.txt`. (You can save your Excel data in the text format by using 'Save as…' function under the File heading in the main toolbar and picking the extension on the bottom of the dialog box.) To learn how to work with files in Matlab, let's make a text file with the data that we generated. Open Notepad (to do this click on the Start button on the left bottom of the screen, then click on 'All Programs', then 'Accessories', then 'Notepad'). An empty Notepad window will come up. Now, return to the Matlab window. Select the data that we generated and copy it (as you would copy some text in Word). Now, paste it in the Notepad window and save the file as `data1.txt` in the directory of your choice.

In order to load a data file into Matlab, we have to open the directory where the file is located. To do this, look at the toolbar on top of the main Matlab window. There is a Current directory name there followed by a button with three dots on it. Click that button, it allows you to open a directory. Open the directory where your data file is located. To make sure that you are in the right place, type `ls` a the prompt. This command allows you to list (hence, `ls`) the files in the Matlab's current directory. You should see you file `data1.txt` in the list.

To load the file, type `load data1.txt`. This will create a new variable called `data1` in the Workspace. Notice that Matlab ignores the extension `.txt` when naming the data variable (however, you do need to put it when you are loading a file). Type `data1` to look to make sure that your data is there.

More plotting

Now, let's plot it. We want to plot the first column of data (month) vs. the second column (temperature). To do this, type `plot(data1(:,1), data1(:,2))`. Now, you have a figure that shows you the temperature. Suppose you don't want the line connecting the points, you can type `plot(data1(:,1), data1(:,2), 'o')`. You will now only see circles for each month. (There are other features of the plot function that you can find out about in the Help.) Suppose you want to have the line and the circles both. To do this, you can keep the plot you already have and superimpose the first plot you did on top. To keep the plot that is in the figure now, type `hold on`. Now type `plot(data1(:,1), data1(:,2))` again. On the figure you will now have both circles and the line. After you are done with the plot, type `hold off`. This assures that the next time you make a plot, the current one will be erased.

You can have more than one figure in Matlab. To make a new figure, type `figure`. An empty figure window will appear. Now when you plot in Matlab, your plot will appear in the figure that you looked at most recently. Suppose you want to make a histogram of the temperatures. To do this, type `hist(data1(:,2))`. (As with `plot`, `hist` has several features that allow you to change number of bins, etc. To find out about these, see Help.)

Saving workspace and plots

Now that you've done all this work, you might want to save it. To save all the variables that you have created, you need to save your Workspace. To do this, type `save workspace_name` (you can pick the name). After you close Matlab, you can open that file and it will contain all the variables that you have created. You can also save selected variables. For example to save variable `data1` only, type `save data_name data1`. This will create another `data_name.mat` file that will contain only variable `data1`.

You might also want to save your plots. There are two ways of saving Matlab plots. One of them saves them in the Matlab format `.fig`. This allows you to open the figure later in Matlab and make modifications to it (change axes, add text, etc.). To save a figure like that, simply click on the File heading on the toolbar on top of the figure window and click Save.

You can also save files in other graphical formats. To do this, go to the File heading on the toolbar of the figure window and click Export. This allows you to export your file in several formats (most familiar would probably be `.jpg`). This is useful for saving your completed plots for reports.

Saving commands: writing `.m` files

You just saved your data and your figure. What about all the commands that you typed in? You may want to save the commands if you want to perform this analysis on several data sets. The commands that you have typed in so far are listed in the bottom left window 'Command History'. Moreover, Matlab will remember these if you restart it, but only on this computer. However, if you change computers, or if the computer crashes, etc., you may loose the list of the commands that you have typed and may have to type and remember them over again. Matlab offers an easy way to have a more permanent record of the commands. You can type your commands into a file and save them, and then execute them within Matlab.

Let's try that. In the main Matlab window, on the top toolbar click on File/New/M-file. This will open a new window with Matlab's own text editor. Let's make a file that would load our data file, plot it and calculate the average temperature for the whole year.

To calculate the average temperature for the year, we will use Matlab function `mean()` that (predictably) calculates the mean of the list of numbers that you put in the parentheses. To calculate the average temperature for the year, you would type `mean(data1(:,2))`.

Now, type the commands into the file that you typed at the prompt:

```
load data1.txt
data1
plot(data1(:,1), data1(:,2), 'o')
hold on
plot(data1(:,1), data1(:,2))
hold off
year_avg=mean(data1(:,2))
```

Save the file under name `prog1` into the same directory where the file `data1.txt` is located (since one of the commands in `prog1.m` will load `data1.txt`). The file will be saved with extension `.m`. This is the format that Matlab uses for files that contain commands.

There is another important step. This is to put comments into our file (a comment is a text that is written for the benefit of the person reading the file only, and Matlab simply ignores it). Comments are **very** important! Imagine that you make a file like this for the class, and then you want to use it three months later when you are writing up your last project. It is quite likely that you will not remember when, for what reason and how you wrote this file. It is also likely that you will not remember what `data1` represented. Therefore, you need comments. To make a comment in Matlab, you put `%` in front of the text. Your commented file may look something like this.

```
% created by Leeza Pachepsky, 1/1/04
% loads and plots the data, finds average yearly temperature

load data1.txt %datafile with month vs. average temperature
% view data
data1
% plot data
plot(data1(:,1), data2(:,2), ‘o’)
hold on
plot(data1(:,1), data2(:,2))
hold off
% find average yearly temperature
year_avg=mean(data1(:,2))
```

It might seem an overkill for you right now to put that many comments into such a small file. However, when you start working with more complicated files, it might be very useful to be very diligent about writing comments.

Now, let’s clear the Workspace by typing `clear`. Also, close all the figure windows you have opened. For you to be able to run the commands in the file `prog1.m`, the file has to be in the Matlab’s current directory. If you saved `prog1.m` into another directory, change to that directory. Now, in the main Matlab window type `prog1` (without `.m` at the end) at the prompt. If everything is well, you will either get the data and the value for the `year_avg` printed, and you will get a figure window with the proper plot in it. Another possibility is that Matlab will give you some errors to sort out. Read the errors. If they make sense, fix your file. If they don't make sense, look at the file and make sure everything is correct. Then type `prog1` again. Do this until it works.

Congratulations, you have just written a Matlab program!

SolvLab: a tool for solving differential equations

In this course, you will have to use Matlab to solve difference and differential equations numerically. To do this, you will use a tool that will make doing this easier. It's called SolvLab. It allows you to define and solve systems of differential and difference equations.

We will learn how to use SolvLab by looking at a problem from your textbook (Nisbet, Gurney, *Ecological Dynamics*), Ch. 2 Project 1. (Note that in the textbook, it is suggested that you use another program called Solver to solve the differential equations. Solver is a separate software written in Pascal specifically designed for solving differential equations. In this course we will stick to the more general Matlab. In particular, we will use a Matlab tool SolvLab.)

The problem involves numerically solving a continuous-time logistic model:

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right)$$

where N is the number of individuals in a population, r is the growth rate of a population at low densities and K is the carrying capacity of the environment.

To start, you will need to download three files from the class website: `logpop.m`, `params_logpop.m`, and `SolvLab_logpop.m`. Make sure that you save the files with extension .m (sometimes when files are saved they get extra/other extensions). Once you download these files, change your current Matlab directory to the directory with the files. Now, click on File/Open in the three files. First look at the `logpop.m file`. There, on line 16 you will see the definition of the differential equation:

```
dN    = r * N * (1 - N / K0);
```

(since we are writing math in Matlab, you have to explicitly include all the multiplication signs, parentheses, etc.). This is the most important line of the file. Above that you see that the values of `r` and `K` come from some vector `p`. This vector comes from another file called `params_logpop.m` where all the parameters are defined. Look at this file. There, you will see two headings: parameters and initial values. Under the parameters heading, the parameter values are defined (`r=1.0`, `K=1.0`). Under the initial values, the initial population `N0`, start and finish time and the time increment are defined (with values 2.0, 0.0, 0.01, and 10.0 respectively). The time increment is the Δt (that you heard about in the lectures) that is used in the numerical simulation of the model. It should be small. How small is a very difficult question to answer, but in general it should be so small that if you make it any smaller, the results should not change.

The third file `SolvLab_logpop.m` looks complicated. It collects the information from the two other files and puts them into a Matlab ODE (ordinary differential equation) solver. It also plots the output. You will not, generally, have to play around with this file.

Now let's try to run this example. Go back to the main Matlab window and type `SolvLab_logpop`. You should get the following output:

```
Parameters are:
           r     1.00  Intrinsic growth rate
         K_0     1.00  Carrying Capacity
Initial Values are:
         N_0     2.00  Initial population size
Time Settings are:
         T_0     0.00  Start Time
         Inc     0.01  Time Increment
         T_F     10.00 Finish Time
Edit params_logpop.m to change the above parameters
Plotting N

printing output to logpopOutput1.txt
```

A figure window with a plot should also come up. In this window the state variable `N` is ploted vs. time. You should see that `N` goes to 1 with time. Now you can play around with the parameter values. In file `params_logpop.m`, change the value of `K0` to `4`. Save the file (if you don't, Matlab will not see the changes you made). You will now see that the plot has changed. Now, the population approaches 4 with time. You can play around with various values of all the parameters and initial values. Notice that every time you run the program an output file called `logpopOutput#.txt` is generated where instead of `#` there is a number corresponding to how many times you have run the program. This is done so that the output is not overwrriten every time. The `logpopOutput#.txt` contains two columns: time and the number of individuals at that time.

Let's come back to the problem. You are required to change a constant carrying capacity to one that changes with time:

$$K(t) = K_0 + K_1 \cos(2\pi t / t_p)$$

To do this, we first need to change the `logpop.m` file. There we need to introduce a new variable `Kt`. In Matlab it would be defined as

```
Kt=K0+K1*cos(2*pi/tp*t);
```

Put this line in right after the

```
%ODEs
```

line.
Now replace `K0` in the next line with `Kt`:

```
dN    = r * N * (1 - N / Kt);
```

There are extra parameters in the problem now: `K1` and `tp`. These have to be inserted in the `params_logpop.m` file. Open that file and insert these parameters and their values in the parameter definition. Now under the parameters heading you will have something like

```
%Parameters
Ps={%   Value    Name  Description
           1.0,  'r',   'Intrinsic growth rate';
           1.0,  'K0',  'Carrying Capacity';
           0.1,  'K1',  'Amplitude of oscillations';
           0.5,  'tp',  'Frequency of oscillations';
};
```
You can put in your own values for `K1` and `tp`.

The last change necessary is to load the new parameters in the `logpop.m` file. In that file, under section `%PARAMETERS`, you need to add definitions of `K1` and `tp`:

```
%PARAMETERS
r     = p(1); % Intrinsic growth rate  (1.0)
K0    = p(2); % Carrying Capacity      (0.1)
K1    = p(3); % Amplitude of oscillations
tp    = p(4); % Frequency of oscillations
```

Now you can run your modified problem (make sure that you saved your modifications). Type `SolvLab_logpop` again. You should get a different answer.

You will use SolvLab in the labs when working on the case studies in the class. SolvLab also allows you to solve difference equations that you will also encounter in one of the case studies. The setup for the difference equations is very similar, and you will be introduced to it then.

More

As you do the labs in the course, they will include descriptions of other Matlab commands that you will use. The basics covered in this tutorial are meant to get you started and to give you skills that allow you to learn more as you go.

Couple of tips and tricks

Access to previous commands

By hitting up arrow on the keyboard you can scroll through the previous commands that you have entered.

Hiding the answer

If you want to perform a command but don't want to see the outcome (suppose you are generating a very long array), you can type a semicolon after the command and Matlab will not display the answer but will perform the command. For example, if you type `x=1:100` and hit enter you will get the whole list of numbers 1 through 100. But if you type `y=1:100;` the variable y will be created but will not be displayed

Further resources

Besides the Matlab Help, there is a lot of information about Matlab on the web. There are also numerous books written about Matlab.

Web resources

Here are some useful websites to explore:

The Mathworks (company that release Matlab) documentation website:
http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

Matlab Help Desk online:
http://www-ccs.ucsd.edu/matlab/

A summary of Matlab functions:
http://www-math.cc.utexas.edu/math/Matlab/Manual/ReferenceTOC.html

Library resources

There are a couple of book in the library that you may want to check out such as *Introduction to Matlab 6* by D. Etter, D. Kuncicky and D. Hull and *MATLAB Primer* by K.Sigmon. Also, *Primer of Ecological Theory* by J. Roughgarden is an ecology book all done in Matlab and that has a Matlab intro section in the back (which is more advanced than this tutorial and readable).

EEMB resources

In Leeza's office (Noble Hall 1103), there are a couple of useful Matlab manuals. These can be borrowed for short periods of time (since these are for use by several people in the lab) if you ask her nicely.

Summary of commands and functions

`%` - put before comments in the `.m` files
`'` – transposes a matrix
`clear` – clear the variables in the Workspace
`dims` – gives the dimensions of a matrix
`figure` – makes a new figure window
`for … end` – `for` loop, used to perform commands several times in a row
`help (function_name)` – shows Matlab help in the Command window as overview or about the `function_name` if it is included
`hold on/off` – keeps/does not keep the current plot when the next one is plotted
`load` – load a file
`ls` – show the list of the files in the Matlab's current directory
`mean(data)` – calculate the mean of `data`
`plot(x_vector, y_vector)` – make a plot (has more options, see help for details)
`quit` – quit Matlab
`rand` – produces a random number between 0 and 1

`randn` – produces a random number from a distribution with a mean of 0 and a standard deviation and variance of 1

`save (variable_name)` – save Workspace or a variable if the `variable_name` is included