Matlab Tutorial

Basic Introduction

When you open Matlab, you should see three different windows. On the right hand side will be the "Command Window." This is where you will do most of your work such as entering commands and viewing results. The command prompt,

```
>>
```

tells you that Matlab is ready for you to type something.

The left hand side should have two separate windows, one is called the "Launch Pad" and the other is the "Command History." At the Launch Pad, you can find manuals and what not, but for now, click on the tab that says "Workspace." The workspace lists all of the data and variables that we have created. Right now, it should be empty. If not, or if at any time you want to delete all of your data, write

```
>> clear all
```

If you ever need help, you can type **help** to see a list of help topics, or **help** *command* to get help on a specific command. For example, **help sin** will tell you hos to use the "sin" function.

At its simplest, Matlab can be used like a calculator. for example:

```
>> 7 + 3
ans =
  10
```

Matlab automatically created a variable called `ans` (you should see it listed in the workspace). **ans** is temporary and contains the result of whatever the last calculation was. If we want to make the result more permanent, we need to assign it to a name, for example

```
>> a = 7 + 3
a =
   10
```

Note that Matlab displayed the result. If we want to suppress this, use a semicolon after the command.

```
b = 5;
```

Matlab created a new variable named **b** with the number 5, but didn't display the result. You should see the variable **b** in the workspace. To see what is stored in a workspace variable, just type its name at the command prompt, (without the semicolon).

```
>> b
b =
   5
```

One of the biggest advantages of using matlab is its ability to handle vectors and matrices. To create a row vector, write

```
>> a = [1 2 3]
a =
   1    2    3
```

To create a column vector, write

```
>> b = [4; 5; 6]
b =
   4
   5
   6
```

Note that **a** is a [1 x 3] vector, and **b** is a [3 x 1] vector. Thus, they are conformable, and their product will be a [1 x 1] vector. Write

```
>> a * b
ans =
   32
```

In order to flip (or "transpose") a column (row) vector to a row (column) vector, write

```
>> c = b'
c =
   4    5    6
```

Another type of operation you may use frequently is element-by-element multiplication or division. Write

```
>> a .* c
ans =
   4    10    18
```

Note that this is not true vector multiplication but that the 1st element in **a** has been multiplied by the 1st element in **c**, the 2nd element by the 2nd element, etc. Especially important is that we have written .* and not simply * for the element by element operation. To create a matrix write

```
>> A = [1 2 3; 4 5 6]
A =
   1    2    3
   4    5    6
```

Note that names in matlab are case-sensitive, **A** is different from **a**. To get a specific value of the matrix, for example the number in the 1st row and 3rd column, write

```
>> A(1,3)
ans =
   3
```

To get the 2nd row and all columns, i.e. the 2nd row vector, write

```
>> A(2,:)
ans =
   4    5    6
```

Starting with data

Clear the workspace with **clear all**. On the class website, you should find some data in `CA_MSA_pop.csv`. As indicated by the extension `.csv`, this is a comma separated file. If you open it in a text editor, such as Notepad or Word, you will see that each value is separated from the others by commas. The 1st row contains column headers, the 2nd row contains names of all the metropolitan statistical areas (MSAs) and the data begins in the 3rd row. The data is the population in each metropolitan area from 1969-2000. Save this file to somewhere on your computer.

From the Matlab window, click on `File --> Import Data...` An explorer window will pop up, from which you will need to find the location where you saved `CA_MSA_pop.csv` and double-click on it. An import wizard will open up. Matlab tries to split up the data into a matrix with numbers and a matrix with labels. In this case, it isn't smart enough to figure it out on its own, so we will need to help it out. Since the data starts on the 3rd row, and the 1st two are just text, type **2** in the box labelled "Text header lines:" Click on "Next." Matlab now shows you that it will create two variables, one called **data**, **textdata** and one called **colheaders**. The box with the check indicates whether Matlab will actually create these variables or not, make sure that they are all checked. You can also select the variables here and see what they will look like. Note that **textdata** isn't quite right, it wasn't able to correctly figure out where the columns are. Click "Finish."

In the workspace, you should see the three variables **data** and **textdata**, and **colheaders**. Double click on **data** in the workspace. A window will open up showing the matrix. It is always possible to edit numbers in this window. Note that the first column of **data** is the year, and the 2nd through 28th columns represent different metropolitan areas. In the Command Window, create a vector for the year and a matrix of population data by writing:

```
>> year = data(:,1);
>> population = data(:,2:28);
```

Double click on **colheaders** in the Workspace. Note that the names of the MSAs actually begin in column 2. Also, the name **colheaders** doesn't really tell us what is in there. Create a vector of names by writing

```
>> MSA_names = colheaders(2:28);
```

This may be a good time to clear our workspace and save it. If we want to delete a variable, use **clear** *variable*. For example, **clear x** will delete any variable named **x**. To save all of the variables in the workspace, click on **File** –¿ **Save Workspace as**. You can then specify the filename you want to save your warkspace as. Make sure that is has the `.mat` extension. All of the variables in your workspace will be saved to file and can be opened later by clicking on `File --> Open`.

Basic Plotting

The population for Stockton-Lodi MSA is in the 19th column. Plot this data with the year on the x axis and population on the right axis

```
>> plot(year,population(:,19))
```
Give it a title
```
>> title('Population in Stockton-Lodi MSA');
```
Label both axes
```
>> xlabel('Year')
>> ylabel('population')
```

Suppose we want to add the population of the Santa Barbara MSA to the same plot, but want to make it a red dotted line. Santa Barbara is in the 16th column. Write

```
>> hold on
```
`hold on` tells Matlab to keep the previous graph
```
>> plot(year,population(:,16),'r:')
```
For more information on colors and line types, write
```
>> help plot
```

Transformations

One of the easiest ways to make a prediction is to extend a line into the future. This might be correct if population actually grew in a straight line fashion, but often it does not. If it does not follow a straight line, one way to proceed is to "transform" the data so that is does follow a stright line, make prediction from this line, and then "back transform" these predictions to follow the original shape. For example, make a vector of "time periods" from 0 to 200 in increments of 1 and a vector of numbers from 0 to 2 in increments of .01, and plot them. (If you already have a plot open, first close it by closing the figure with the mouse, or by typing **close** in the command window).

```
>> x = 0:1:200;
>> y = 0:.01:2;
>> plot(x,y)
```

y is clearly linear. Now transform **y** to be exponential, and plot it again.

```
>> y = exp(y); plot(x,y);
```

Clearly, **y** is not linear, it grows exponentially. For exponential data, we can make it linear by taking the log of it.

```
>> logy = log(y)
>> plot(x,logy)
```

The logarithm and exponent are "linear transforms" of each other. We will use many such linear transformations.

Lets consider a real example. We want to plot just the population of Stockton-Lodi from 1960 - 1990. Which row is 1990 in? To find out, use the **find** command. Write

```
>> find(year==1990)
>> ans =
   22
```

The **find** function marched through the vector **year**, evalueating whether or not it was equal to "1990". Since the 22nd row of **year** was exactly equal to 1990, **find** returned 22 in this case. We could have easily determined this for ourselves, but the **find** command is very useful one to know in more complex situations. Now plot the data for Stockton from 1969-1990.

```
>> plot(year(1:22),population(1:22,19))
```

This data might grow exponentially, so lets take the log transform and plot it.

```
>> y = log(population(1:22,19));
>> plot(year(1:22),y)
```

Does it appear to be a straight line now? We will use many such transformations in the next assignments.

Common Matlab Mistakes

1. Failing to use **help** to see how a function is used.
2. Giving a variable the same name as a function.
   Example, Do not name a variable **sin**, if you do, you will no longer be able to use the **sin** function.
3. Confusing your rows and columns within vectors and matrices.
4. Improper number of parenthesis.
5. Confusing your `.*, ./ or .^` for `*, /, or ^`