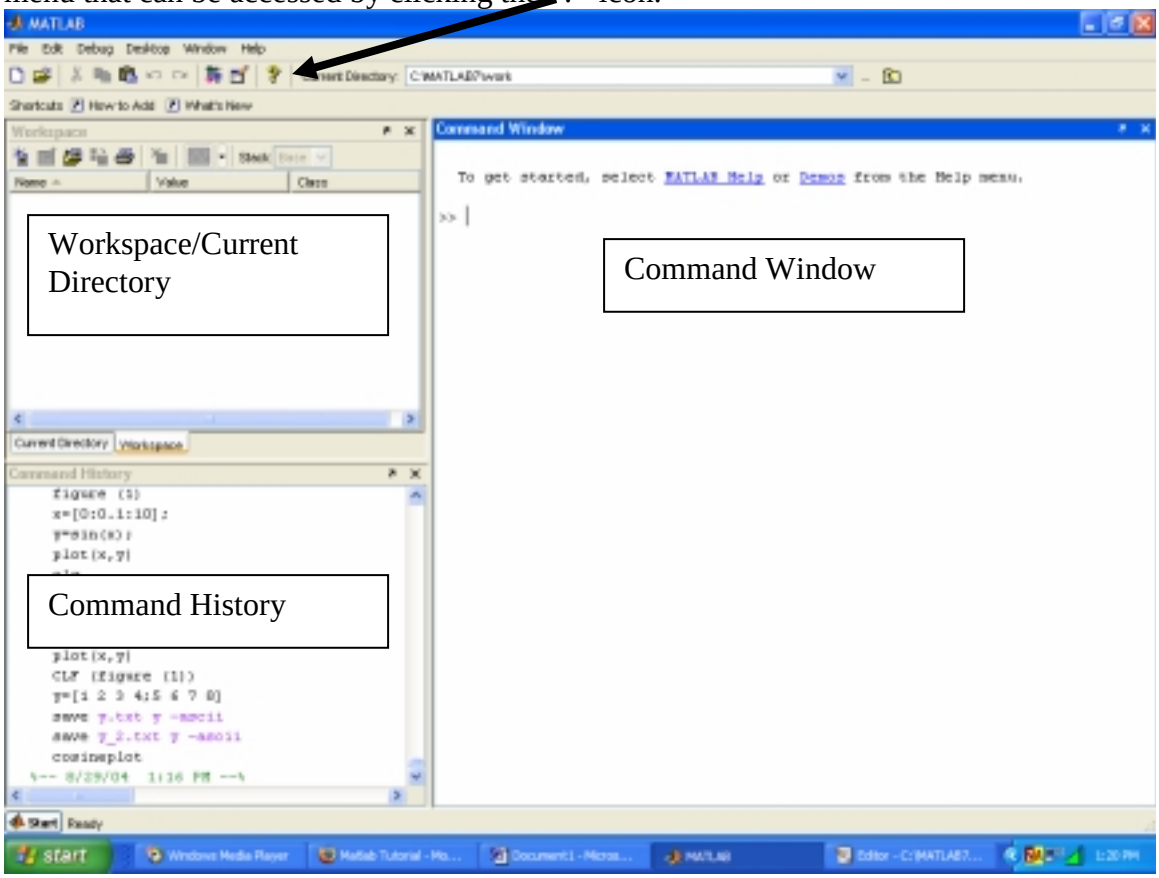
Homework #1: Matlab basics

Adopted from the following webpage for Matlab 5:
http://users.rowan.edu/~shreek/networks1/matlabintro.htm

Introduction:

Matlab (short for MATrix LABoratory) is a language for technical computing, developed by the The Mathworks, Inc. It provides a single platform for computation, visualization, programming and software development. All problems and solutions in Matlab are expressed in notation used in linear algebra and essentially involve operations using matrices and vectors. In this class, you will use Matlab 7 for exploring topics presented in lectures and build a complete working model of an atomic force microscope using Simulink. (Matlab 7 is being used to take advantage of some new features that are not available in earlier versions of Matlab.)This will act as a brief introduction to some basic skills in Matlab 7 that will be useful in this class. In this tutorial, commands to be typed in the command line will be written in **boldface**. Matlab 7 also has a very useful help menu that can be accessed by clicking the "?" icon.



Note: for more information on any command given in this tutorial type **help *name of command*** in the Command Window, and a description of the command will appear in the Command Window. For an example enter the following:
>> **help figure**

The following homework will ask you to produce some output (both numerical and graphical) to be turned in at class. These are highlighted in yellow.

Getting Started:

The ">>" is a prompt, requiring you to type in commands. One of the first commands you should type in is to find out what your working directory is. The working directory is where you will save the results of your calculations. So, type in

>> **pwd**

pwd stands for "print working directory". You probably would get an output like:

```
ans =

C:\MATLAB7\work
```

Matlab always stores the result of its last calculation in a variable called ans (short for answer).

Vectors and Matrices

Variables in Matlab are just like variables in any other programming language (C, C++ etc.); only difference is that you do not have to define them by indicating the type etc. Also, variable names (case sensitive) can be used to refer to a single number (a scalar), a set of numbers (a vector) or an array of numbers (a matrix).
Vectors are nothing but matrices having a single row (a row vector), or a single column (a column vector).
To create a row vector in Matlab, do:

>> **r = [1 2 3 4]**

```
r =
     1     2     3     4
```

A column vector can be created by

>> **c = [1; 2; 3; 4]**

```
c =

    1
    2
    3
    4
```

On the other hand, you can use the ' operator (transpose)

>> **c = r'**

```
c =
```

1
2
3
4

Vectors can also be created by incrementing a starting value with a constant quantity. For example,
>> **r = [0:2:10]**
r =

0 2 4 6 8 10
creates a row vector, with the first element = 0; each element incremented by 2; until the final value of 10.

You can index specific parts of a vector. For example, to get the third element in the vector r, you can do
>> **r (3)**

ans =

4

Matrices are 2 dimensional quantities and are created similar to vectors. We can do
>> **a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]**
a =

1 2 3
4 5 6
7 8 9
10 11 12
which is a 4x3 matrix (4 rows and 3 columns). We can also use the incremenation principle to do
>> **b = [0:2:10; 1:2:11]**
b =

0 2 4 6 8 10

1 3 5 7 9 11
which is a 2x6 matrix. Again, individual elements of the matrix, for instance the element in the 2nd row, 5th column can be accessed using the notation:
>> **b (2, 5)**
ans =

9

The command >>**Test = [1 4 7 10 13; 2 5 8 11 14; 3 6 9 12 15]** will produce the following matrix:

Test=

1 4 7 10 13

2 5 8 11 14

3 6 9 12 15

1. For your homework, give another command that would produce the same 3x5 matrix.

Vector and Matrix Operations:

The basic arithmetic operations +, -, *, / can be used for vectors and matrices. These would generate corresponding output vectors or matrices. For example, to add two vectors:

```
>> a = [1 2 3 4];
>> b = [5 6 7 8];
>> c = a+b
c =
      6 8 10 12
```

The semicolons (;) in the first two commands direct Matlab not to echo the values of the variables a and b on to the screen immediately after you type them. Obviously, only vectors that have the same number of elements can be added or subtracted. Similarly, two matrices with identical number of rows and columns can be subtracted as follows:

```
>> a = [1:3:20; 21:3:40];
>> b = [2:3:20; 22:3:40];
>> c = a-b
c =

   -1   -1   -1   -1   -1   -1   -1
   -1   -1   -1   -1   -1   -1   -1
```

Matrix multiplication using the ***** symbol is possible only if the number of columns in the first matrix equals the number of rows in the second:

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

    1    2    3
    4    5    6

>> b = a'
b =
```

```
    1     4
    2     5
    3     6
```

>> **c=a*b**

```
c =

   14    32
   32    77
```

However, if you want to multiply corresponding elements of two matrices, then you do an array multiply using the **.*** symbol.

>> **a = [1 2 3 4; 5 6 7 8]; b=[2 2 2 2; 3 3 3 3];**
>> **c=a .* b**

```
c =

    2     4     6     8
   15    18    21    24
```

Loops:

The for loop (very similar to C expression) is a simple command for setting up a loop. Example:

>> **for i = 1:10;**
>> **d(i) = i*i;**
>> **end**
>> **d**

```
d =

    1     4     9    16    25    36    49    64    81   100
```

All statements between the for and the end statements will be executed as per the command specifications. Example of a for loop where the increment is not 1 would be >> f**or i = 1:3:20**; ..... etc.
Type in
>> **help for**
for more details.

File Input/Output:

The **save** and **load** commands are used for saving data to disk or loading data from disk, respectively.
To save values in a matrix or vector, called, for instance, y, do:
>> **y = [1 2 3 4; 5 6 7 8];**

>> **save y.txt y -ascii**
The -ascii option ensures that the data is saved in ASCII form, so that it can be read by other programs - Word, Excel Notepad, Wordpad, etc. Examine the file y.txt using one of these programs. The file is generated in the working directory (remember this? If not click here).

Type in the following command

>> **clear y**

This removes the matrix y from your workspace.

An ASCII data file can be loaded using the load command

>> **load y.txt**

This provides a variable called y in the Matlab workspace. All manner of vector/matrix operations can be performed on y, just like any other variable.

>> **y**

ans =

1 2 3 4

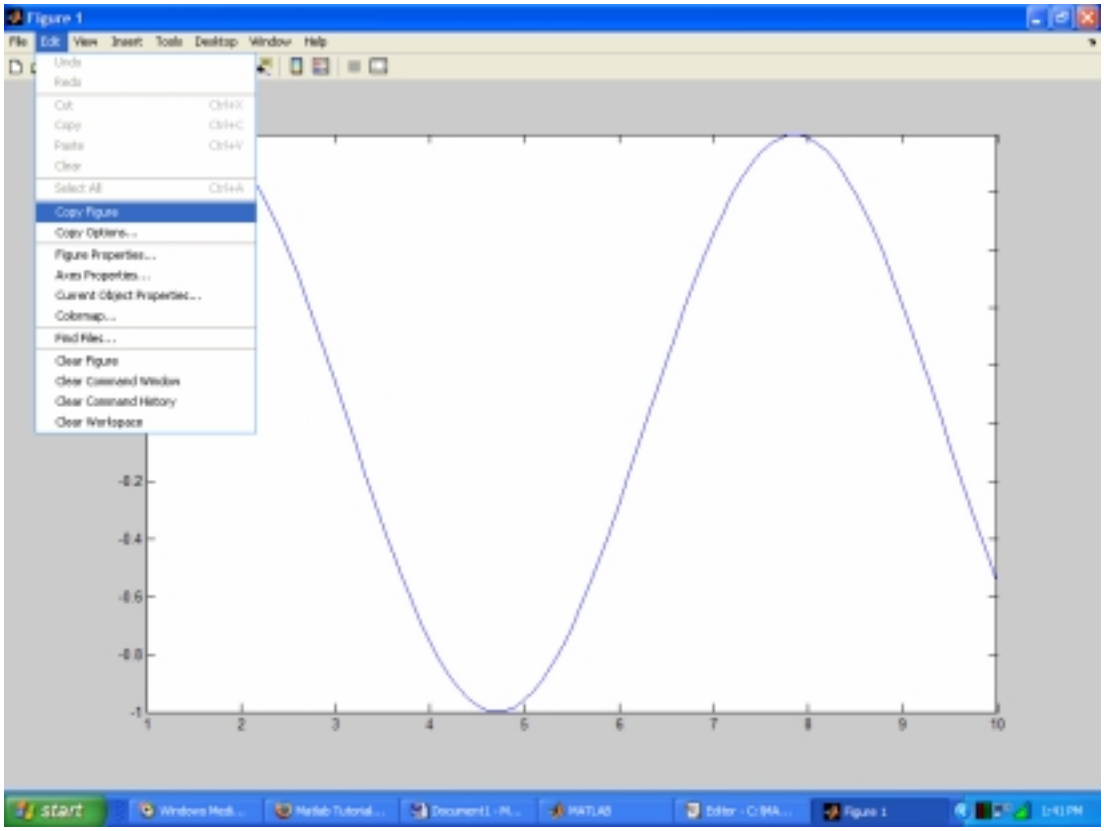
5 6 7 8

Do >> **help save** and >>**help load** for learning all the load/save options.

Plotting:

The **plot** command is used for generating 1-D (functions of one variable) plots. Do >> **help plot** for complete details. Let's make a graph of y = sin(x), for x on the interval x=0, to x = 10.
>> **x = [0:0.1:10];** *(here .1 is the increment)*
>> **y = sin(x);** *(notice how the sin function operates on each element of the entire row vector x, to generate another row vector y)*
>> **figure (1)** (*this should open a blank figure in another window*)
>> **plot (x, y)**

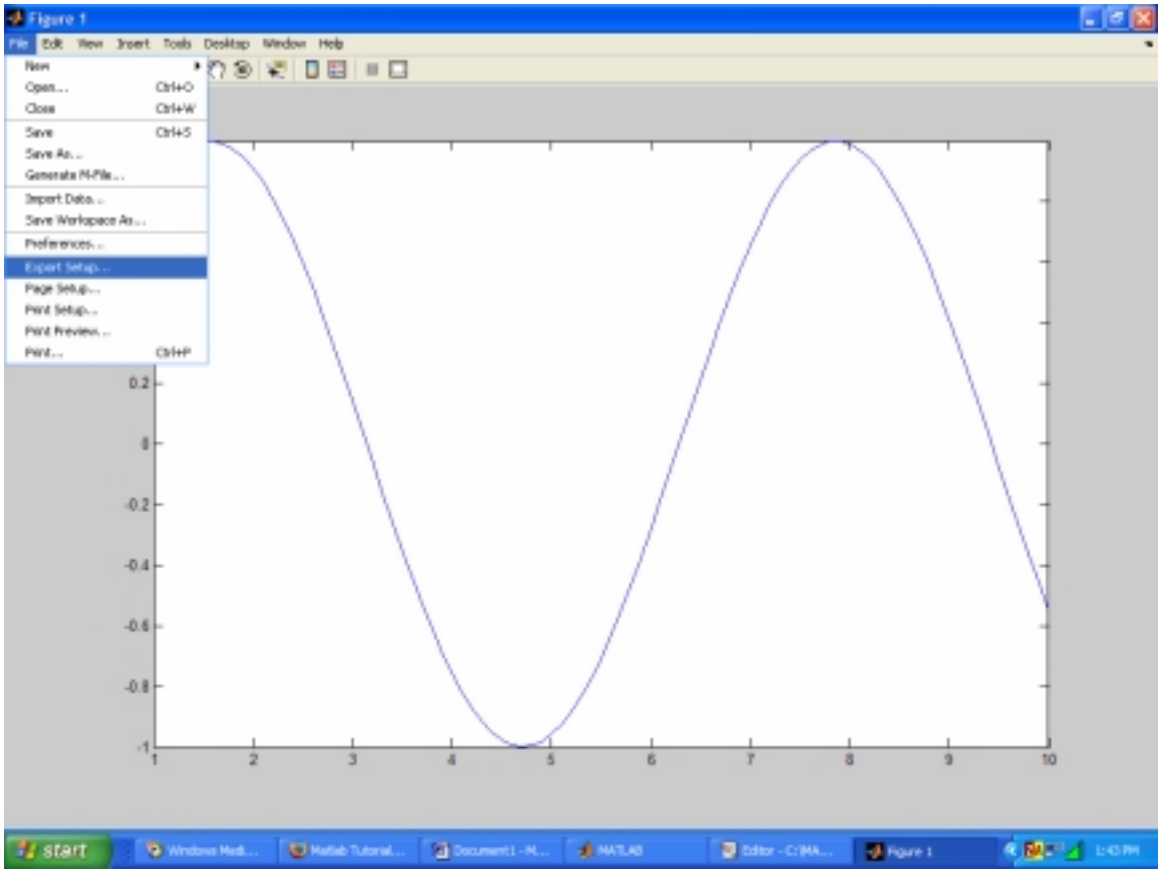
To place a figure on the clipboard, go to the edit menu of the figure and select copy figure.



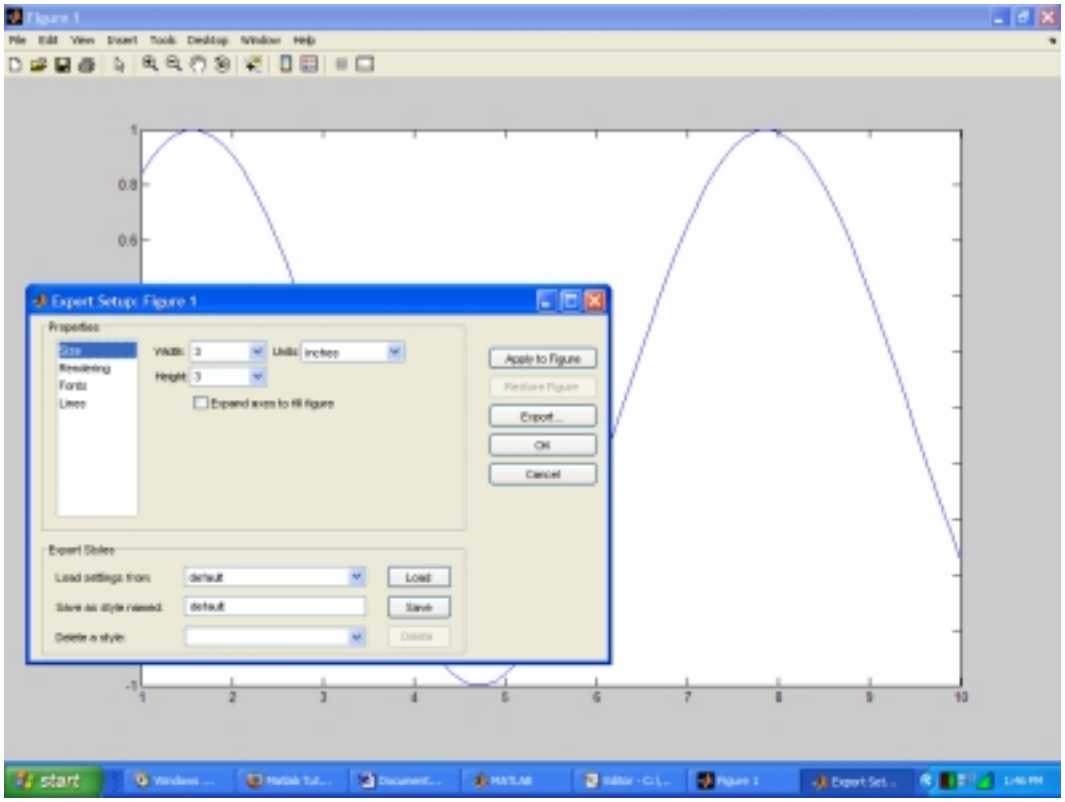
You can now paste this figure into another program such as PowerPoint, Word, Photoshop, etc.

2. Paste this graph into your homework.

To export a figure as a jpg, tiff, etc., go to the file menu and select Export Setup:



This will open the Export Setup window, which allows you to change the figure size, rendering, fonts, etc. before exporting the figure.



Click on Export, and select the type of image you would like to export the figure as.

Now let's make another plot. To generate another plot window, type >> **figure (2)**

>> x2=[0:0.1:10];

>>y2= cos(x2);

plot (x2, y2)

To clear a plot, type in >> **clf (figure(1))**

This should clear figure (1)

Now, make two plots in one figure.

>> **figure (1)**

>> plot (x, y, x2, y2)

Now, change the markers of the plot.

>> **plot (x, y,'+', x2, y2,'*')**

3. Paste this plot into your homework.

Now we will make subplots in the same figure. The subplot (m,n, p) command divides the figure into m by n matrix of small axes, and p is the active subplot

>> **clf (figure (1))**

>> subplot (2, 1, 1)

>> plot (x, y)

>> subplot (2, 1, 2)

>> plot (x2, y2)

4. Paste this plot into you homework.

Matlab also is capable of displaying images in two and three dimensions. The following examples will demonstrate these capabilities.

Load the AFM image file and axis information contained in the txt files called image, Xaxis, and Yaxis that are available on the course webpage. (Make sure that these files are in your current directory). This is an image of a brain lipid extract bilayer suspension being disrupted by the self-assembly of the β-amyloid peptide into fibrils. The aggregation of this peptide plays an important role in the pathogenesis of Alzheimer's disease.

>> **load image.txt**

>> load Xaxis.txt

>> load Yaxis.txt

Note that image is a 512x512 matrix. Open a new figure and use imagesc to plot the image.

>> **figure (4)**

>> imagesc (image)

Note that the image axis are scaled in pixels (1 to 512). The actual dimensions of this image are 1.5x1.5 μm. To add more meaningful axis scales use imagesc again, but this time specify the scaling with the matrices that you loaded earlier (Xaxis and Yaxis). The dimensions of both of these matrices are 1x512.

>> **imagesc (Xaxis, Yaxis, image)**

The values of the scaling matrixes range from 0 for the first value to 1.5 for the last. Notice that the scale on the image now corresponds to these values.

5. For your homework, give a command that would produce a proper scaling matrix for this image.

The image appears rectangular, but it should be square (it is a 5x5 μm image after all). Therefore, the aspect ratio should be adjusted. Use the set command in this form: set(H,'PropertyName',PropertyValue), where H will be gca for "get current axis", and the PropertyValue will be [1 1 1] to make the image square.

>> **set(gca,'DataAspectRatio',[1 1 1]);**

The set command can also be used to change the figure background color from grey to white. This time H will be gcf for "get current figure".

>> **set(gcf,'Color',[1 1 1])**

A color bar can be added.

>> **colorbar**

Change the colormap to gray (the default is called jet).

>> **colormap gray**

There are several colormaps to chose from (autumn, copper, spring, jet, bone, gray, hot etc.), and custom colormaps can also be created.

6. Paste this image in your homework.

To plot this image in three dimension, use the surf command (short for surface plot). Again, the axis scaling should be specified.

>> **figure (5)**

>> **surf (Xaxis, Yaxis, image)**

This image probably appears very black, because Matlab defaults the shading to faceted, which is flat shading with superimposed black mesh lines. We will change the shading to interpolated. Interpolated shading, which is also known as Gouraud shading, is piecewise bilinear; the color in each segment or patch varies linearly and interpolates the end or corner values. (A third option for shading is flat, which is flat shading without the black mesh lines.)

>> **shading interp**

Now turn the grid lines off.

>> **grid off**

Change the background color.

>> **set(gcf,'Color',[1 1 1])**

Change the axis values with the axis command: axis([XMIN XMAX YMIN YMAX ZMIN ZMAX]) sets the scaling for the x-, y- and z-axes on the current 3-D plot.

>> **axis ([0 1.5 0 1.5 -1 250])**

It can also be helpful to change the view of the 3D image with the view command: view (AZ, EL) sets the angle of the view from which an observer sees the current 3-D plot. AZ is the azimuth or horizontal rotation and EL is the vertical elevation (both in degrees).

>> **view (30, 50)**

Now, change the colormap to hot.

>> **colormap hot**

Next, we will label the axis.

>> **xlabel '\mum'** *(\mu tells Matlab to use the greek character μ)*

>> ylabel '\mum'

>> zlabel 'nm'

Now, add a title

>> **title 'The self-assembly of A\beta disrupting a bilayer'**

Finally, we will add a light to the image.

>> light1=lightangle(90, 30);

The command lightangle (AZ, EL) is similar to the view command, but this time we are defining the position of the light source shining on the image. We named our light "light1" so that we could change its properties later using the set command. Also, more than one light can be added to each figure, and this is useful in keeping track of these different lights.

7. Paste this 3D image into your homework.

Now we will change the color of the light using the set command.

>> **set (light1, 'color', [1 0 0])**

In this command, we first identify the object we want to set (light1), then the property we want to change ('color'), and finally we define the changes. For the color, it is defined by three numbers that correspond to red, green, and blue. The intensity of the different colors is determined by a value ranging from 0 (no intensity) to 1 (highest intensity). By changing the relative intensities of the three colors, any color of light can be used on the image. In the above command, we made the light a red light. The next two commands will change the light to green and blue respectively.

>> **set (light1, 'color', [0 1 0])**

>> **set (light1, 'color', [0 0 1])**

Now make the light white again.

>> **set (light1, 'color', [1 1 1])**

Strings:

There are two basic executable file types that can be created in Matlab (strings and functions), which are generated by storing a list of Matlab commands in a file given the extension .m
These files are called M-files. To create an M-file, use the New...M-file Option under the File Menu in the Command Window. Type in the following commands in the M-File Editor Window:

```
x = [0:0.1:10];
y = cos(x);
plot(x,y)
xlabel('x')
ylabel('y')
title('A plot of Cosine(x)')
```

Save the file using the Save Option under the File Menu in the M-File Editor Window. Call it, say, **cosineplot.m**. Now, to run this program in Matlab, move over to the Matlab Command Window and just type in >> **cosineplot**

8. Paste this plot in your homework.

A useful command in creating M-files is the diary command (in the form diary FILENAME). This function causes a copy of all subsequent command window input and

most of the resulting command window output to be appended to thenamed file. If no file is specified, the file 'diary' is used.

>> **diary EasySin.m**

>> **Xvalues = 0:0.1:10;**

>> **easysin = sin (Xvalues);**

>> **figure (6)**

>> **plot (Xvalues, easysin)**

>> **set(gcf,'Color',[1 1 1])**

>> xlabel('x')

>> **ylabel('y')**

>> **title('A plot of Cosine(x)')**

>> diary off

Now open the created M-file, and see that all of these entries are recorded. If you close the plot and clear your workspace (>> **clear all**), you can easily recreate the plot by calling the M-file.

>> **EasySin**

9. Paste this plot in your homework