ELEC 342 Lab 1 : Introduction to MATLAB

Getting started:

1. MATLAB is available from both Windows and Linux. Reboot the PC and select your preferred operating system.

2. Login using your ENCS login name and password.

3(a). To start MATLAB from Windows select the MATLAB icon (double click) from the desktop.

3(b). To start MATLAB from Linux, type `matlab &` from the command prompt. It is recommended that you first create a subdirectory to hold your MATLAB related file. For example, the following Linux commands will create a subdirectory called MATLAB in your home directory (the home directory is the directory which you are automatically placed in upon logging in:

```
ted@deadflowers ~ 2:34pm >  mkdir MATLAB
ted@deadflowers ~ 2:34pm > cd MATLAB
ted@deadflowers ~/MATLAB 2:43pm >  matlab &
[1] 11168
ted@deadflowers ~/MATLAB 2:43pm >
```

In the above commands, the "`ted@deadflowers ~ 2:34pm >`" is known as the command prompt. Your prompt may appear different depending on how your account has been configured. Linux commands are entered on the command line following the prompt followed by pressing the enter key. The command to start MATLAB is simply: `matlab &`. By including the & symbol on the command line after the word `matlab`, control will return to the prompt and you will be able to continue entering commands from that window. If you do not specify the & symbol, control will not return to prompt until you quit MATLAB.

After a few seconds, the main MATLAB window as shown in Figure 1 will appear:

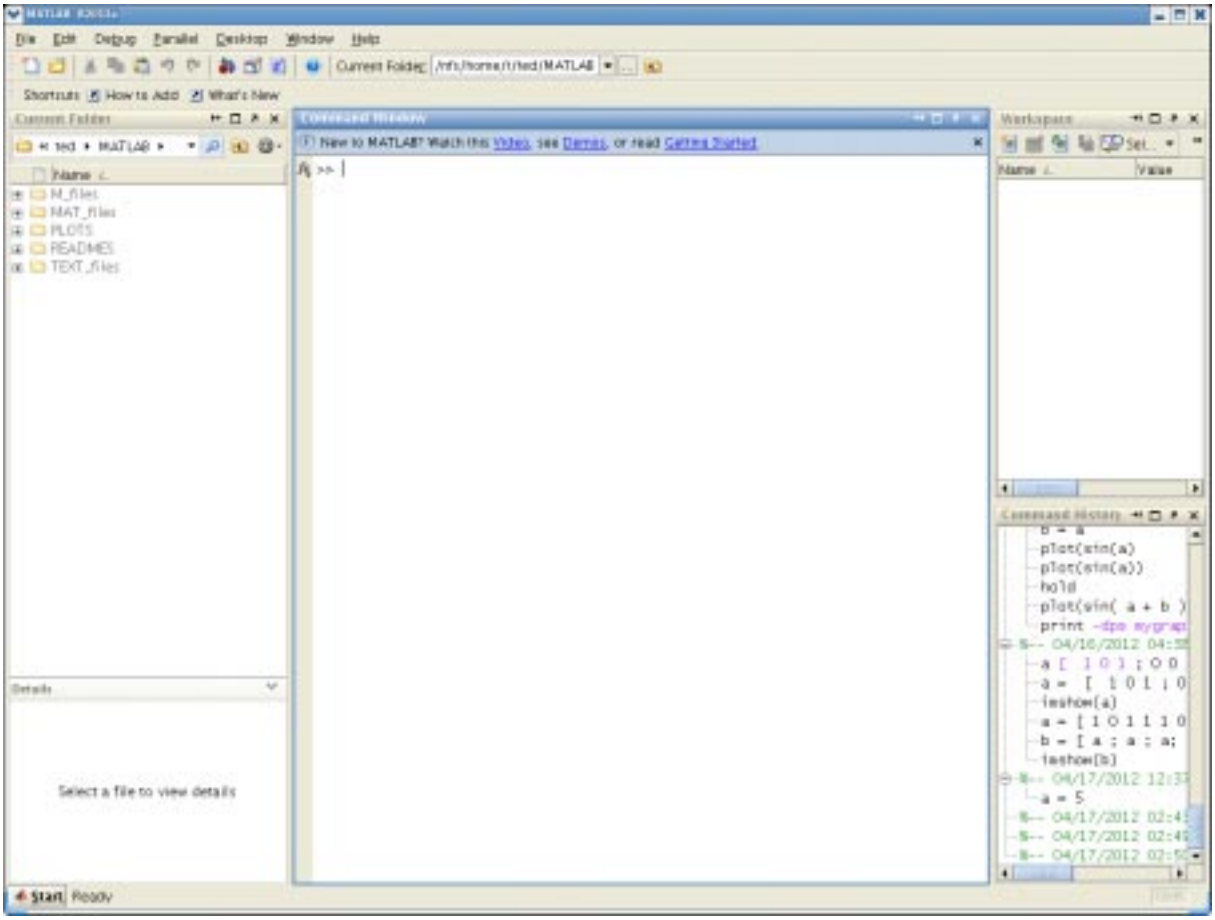


Figure 1: Main MATLAB window.

The main MATLAB window is subdivided into four subwindows:

Command window: This is the central window which contains the >> prompt . MATLAB commands are entered from this prompt in this command window and results will be displayed (echoed) in the window.

Workspace window: This are shows any variable which may exist and their values. It should be initially empty.

Command History: This window is displayed below the Workspace window and it displays a history of previous commands which may have been entered (during a previous MATLAB session).

Current Folder: Located on the top left portion of the main window, it displays any files and directories which may exist in the working directory from where you started MATLAB from.

Assignment operator:

The assignment operator in MATLAB is the = symbol. It is used to assign a value to a variable. A value can either be a literal, or a MATLAB expression which evaluates to some value.
Let us create a variable called A and assign it an initial value of 5. From the >> prompt in the command window type the following (followed by the Enter key)

```
>> A = 5
```

MATLAB will display the following in the Command window:

```
A =

     5

>>
```

Observe that the Workspace now contains a variable called A with value 5 size 1x1 and that this-variable consists of 8 bytes of memory storage. Refer to Figure 2 for details.

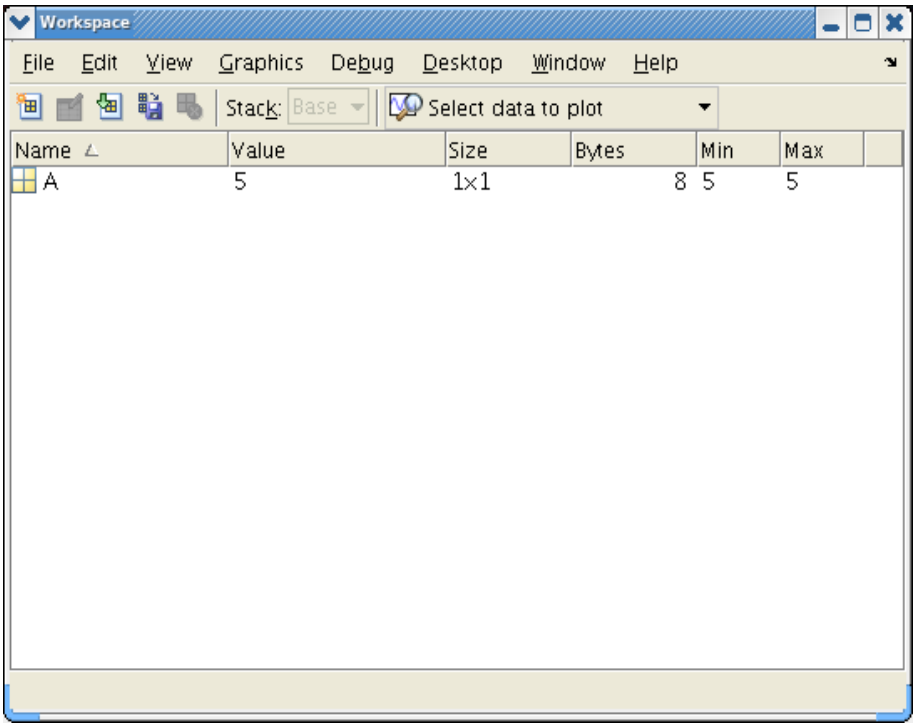


Figure 2: Workspace window.

Assign a new value of 6.9 to variable A:

```
>> A = 6.9

A =

    6.9000
```

Suppressing the display:

If you do not wish to have MATLAB echo the results of the command you type in, simply terminate the command with a ; (followed by the Enter key). MATLAB will perform the command but the results will not be displayed:

```
>> B = 10 ;
>>
```

Basic Arithmetic Operators for Scalars

You may have wondered what the 1x1 entry for the Size of variable A in the Workspace window refers to. MATLAB essentially treats all variables as arrays, a simple scalar variable such as our A and B in the examples thus shown are treated as a 1x1 array of doubles. MATLAB supports the following arithmetic operators when working with scalar values: `+, -, *, /, ^`. The `*` operator is used to denote multiplication, `/` is the division operator, and `^` is the power operator used to perform x^y. Some examples:

```
>> lenght = 5

lenght =

     5

>> width = 4

width =

     4

>> area = lenght * width

area =

    20

>> powers_of_two = 2 ^ 0

powers_of_two =
```

```
     1

>> powers_of_two = 2 ^ 1

powers_of_two =

     2

>> powers_of_two = 2 ^ 2

powers_of_two =

     4

>> powers_of_two = 2 ^ 3

powers_of_two =

     8

>> powers_of_two = 2 ^ 4

powers_of_two =

    16

>> divide_by_zero = 1 / 0

divide_by_zero =

   Inf
```

Interestingly enough, MATLAB returns the answer "`Inf`" as a result of dividing by 0. Some of may have come across this symbol from your programming courses. `Inf` is one of the reserved values in the IEEE 754 floating point standard and is used to represent infinity. In general, dividing by 0 is usually not a good thing to do...

The `ans` variable:

Type the following in the Command window:

```
>> 43
```

MATLAB will display:

```
ans =

    43
```

MATLAB will create a variable called ‘ans’ if you do not explicitly assign one yourself.

Operator Precedence:

What will the result of the following expression be?

```
>> 3 + 4 / 3
```

Depending on the order in which we perform the two arithmetic operators contained in the expression the result can be either 2.3333 or 4.333. Which one is correct ???

```
ans =

    4.3333
```

In MATLAB (just as in any other high-level programming language), the * and / operators are said to be of higher precedence than the + and - operators. There is a very common mnemonic (memory aid” which goes along the lines of:

My **D**ear **A**unt **S**tephanie

which helps one to remember that **M**ultiplication and **D**ivision are to be performed first before **A**ddition and **S**ubtraction.

What will be the answer to:

```
3 + 4 ^ 2 / 8 ?
```

The ^ operator has higher precedence than either multiplication or division and is thus performed first giving the answer of:

```
ans =

     5
```

MATLAB allows the use of parenthesis to explicit force precedence (and to aid in readability of complex arithmetic expressions) :

```
>> 3 +  ( (4 ^ 2) / 8 )

ans =
```

```
     5
```

1-D arrays:

MATLAB also allows one to create one dimensional arrays (ie. a vector):

```
>> my_row_vector = [ 1 , 2 , 3 ]

my_row_vector =

     1     2     3
```

When defining a vector, the , used to separate each element may be omitted and a blank can be used to separate each element:

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

my_row_vector =

     1     2     3
```

Note that the size of variable `my_row_vector` is reported as 1 x 3 array in the Workspace. A column vector may be created as:

```
>> my_column_vector = [ 1 ; 2 ; 3 ]

my_column_vector =

     1
     2
     3
```

Note that the semicolon (;) is used to separate each of the three elements which comprise the 3 x 1 array.

A useful operator when working with arrays is the transpose operator (‘). We may obtain a column vector from a given row vector by taking the transpose of the row vector (or vice versa).

```
>> row = [ 4 5 6 ]

row =

     4     5     6

>> col = row’
```

```
col =

     4
     5
     6
```

Colon Operator:

MATLAB can create an array consisting of a range of numbers by using the colon operator:

```
>> colon_array_example = [ 1 : 10 ]

colon_array_example =

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

The expression `[ 1 : 10 ]` will result in a 10 element array consisting of the values from 1 to 10 inclusive. There is an alternative in which one may specify the step size between elements:

```
>> step_size_example = [ 1 : 2 : 10 ]

step_size_example =

     1     3     5     7     9
```

The first element is set to 1, the second element is set to 1 + step size = 3, the third element is set to 3 + step size = 5, etc. The general form of the expression is:

```
[ start_value : step_size : end_value ]
```

Accessing array elements:

To access a particular element of an array, the index of the element is used within `()`. For example:

```
>> my_row = [ -56 234 12 ]

my_row =

   -56   234    12
```

To access the first element of the my_row array, we can do:

```
>> element = my_row(1)

element =
```

```
   -56
```

NOTE: In MATLAB **array indices start from 1** (and not 0 as in the C and C++ programming languages). Using an index value of 0 will result in an error:

```
 >> element = my_row(0)
Attempted to access my_row(0); index must be a positive integer or
logical.
```

Element-by-Element vector operators:

When working with vectors, MATLAB allows for element-by-element operations. For the `+` and `-` operators this is quite intuitive:

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

A =

     1     2     3

>> B = [ 4 5 6 ]

B =

     4     5     6

>> C = A + B

C =

     5     7     9
```

In this example, we are adding two 1 x 3 row vectors and assigning the result to a third row_vector called C and the elements of C will be:

C(1) = A(1) + B(1)
C(2) = A(2) + B(2)
C(3) = A(3) + B(3)

The `-` operator works in a similar fashion:

```
>> A = [ 5 6 7 ]

A =
```

```
     5     6     7

>> B = [ 1 2 3]

B =

     1     2     3

>> C = A - B

C =

     4     1     6
```

When performing the element-by-element multiplication and division and power, we use the `.` operator followed by the specified arithmetic operator. For example:

```
.*
./
.^
```

The following shows typical use of the various element-by-element operators:

```
>> A = [ 2 3 4 ]

A =

     2     3     4

>> B = [ 5 6 7 ]

B =

     5     6     7

>> C = A.*B

C =

    10    18    28
```

In the above example, the elements of C will be:

C(1) = A(1) * B(1)
C(2) = A(2) * B(2)
C(3) = A(3) * B(3)

The `.^` operator is similarly defined, using the same values for A and B, the result of:

```
>> A = [ 2 3 4 ]

A =

     2     3     4

>> B = [ 5 6 7 ]

B =

     5     6     7


>> C = A .^ B

C =

          32         729       16384
```

The `./` operator is also similarly defined:

```
>> A = [ 8 32  64 ]

A =

     8    32    64

>> B = [ 4 2 4 ]

B =

     4     2     4

>> C = A ./ B

C =

     2    16    16
```

Scalar and vector operators:

MATLAB allows to multiply a vector by a scalar, or to divide a vector by a scalar, or to raise every element of a vector by a scalar :

```
>> A = [ 2 , 3, 4 ]

A =

     2     3     4

>> A = A * 2

A =

     4     6     8

>> A = [ 2 3 4 ]

A =

     2     3     4

>> A = A / 2

A =

    1.0000    1.5000    2.0000

>> A = [ 2 3 4 ]

>> A = A .^ 2

A =

     4    16    64
```

Matrices:

MATLAB allows for two-dimensional (and also higher dimensions) arrays:

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

my_array =

     1     2     3
     4     5     6
```

my_array is said to be a 2x3 array. It contains two rows and three columns. To access a particular element within the array, we use the notation `array_name(row_index, column_index)` as in:

```
>> element = my_array(1,1)

element =

     1
```

Array Multiplication:

Consider the following two array declarations:

```
A =

     1     2     3
     4     5     6

>> B = [ 7 8 9
10 11 12 ]

B =

     7     8     9
    10    11    12
```

We have already seen how we can perform the element-by-element array multiplication of these two arrays by using the .* operator:

```
>> C = A .* B

C =

     7    16    27
    40    55    72
```

What would happen if we were to try to multiply these two arrays together using the * operator?

```
>> C = A * B
Error using  *
Inner matrix dimensions must agree.
```

Array multiplication is defined differently than element-by-element array multiplication. Indeed, when MATLAB performs matrix multiplication using the * operator, it follows the rules of array

multiplication as defined by linear algebra. Furthermore, there are certain restrictions which are placed on the sizes of the two arrays which are to be multiplied. In general, if `mick` is an (m x n) array and if `keith` is an (n x k) array, then the resulting array found by multiplying `mick` by `keith` will be of size (m x k) . Stated in other words, the number of columns in the first array must be equal to the number of rows in the second array:

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

mick =

     1     2     3
     4     5     6

>> keith = [ 7 8 ; 9 10 ; 11 12 ]

keith =

     7     8
     9    10
    11    12

>> stones = mick * keith

stones =

    58    64
   139   154
```

Complex Numbers:

MATLAB can work with complex numbers of the form : `real_part + j * imaginary_part` and can perform complex arithmetic directly:

```
>> complex1 = 2 + 3i

complex1 =

   2.0000 + 3.0000i
>> complex2 = 3 + 5j

complex2 =

   3.0000 + 5.0000i
```

Note that we may use either the symbol "`i`" or "`j`" to denote the imaginary part of the complex number. To add these two complex numbers, we simply perform:

```
>> complex_sum = complex1 + complex2

complex_sum =

   5.0000 + 8.0000i
```

MATLAB uses the rules of complex arithmetic to perform addition, subtraction, multiplication and division of complex numbers:

```
>> complex_product = complex1 * complex2

complex_product =

  -9.0000 +19.0000i

>> complex_division = complex1 / complex2

complex_division =

   0.6176 - 0.0294i
```

We can even define a vector (or an array) of complex numbers:

```
>> complex_vector = [ complex1 ,   23 + 34j,  complex2 ]

complex_vector =

   2.0000 + 3.0000i  23.0000 +34.0000i   3.0000 + 5.0000i
```

Plotting:

MATLAB provides very powerful graphing capabilities which allows for the creation of either two-dimensional or three-dimensional plots. Suppose we wish to plot the equation of the straight lien defined by the equation:

$y = 3x$

We know this line will have a slope of three and intercept the x-axis at 0. Let us use MATLAB plot this line. First we define the points on the x-axis using a row-vector:

```
>> X = [ 0 , 1, 2 ,3 ]

X =
```

```
     0     1     2     3
```

Next, we compute the y-value for every x value by multiplying every element in the X vector by 3 and assigning the result to another row vector called Y:

```
>> Y = 3 * X

Y =

     0     3     6     9
```

Now, we are ready to use the MATLAB plot function:

```
>> plot(X,Y)
```

The plot function will plot the elements of the second argument (Y in this example) as a function of the elements in the first argument (X) . In other words, the elements of Y will be plotted on the vertical y-axis positioned at a location on the x-axis as specified by the elements of vector X.

Figure 3 shows the results of the plot command.

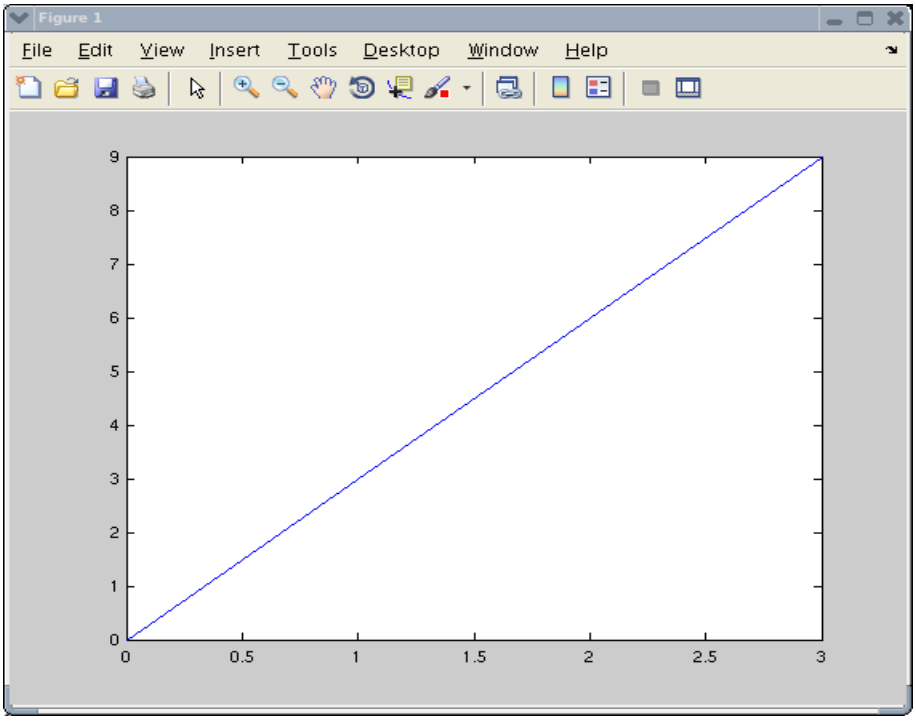


Figure 3: Plot of Y = 3X

There is a variation of the plot command called `stem` which does not connect the data points, but simply indicates them with circles and vertical lines (refer to Figure 4) :

```
>> stem(X,Y)
```

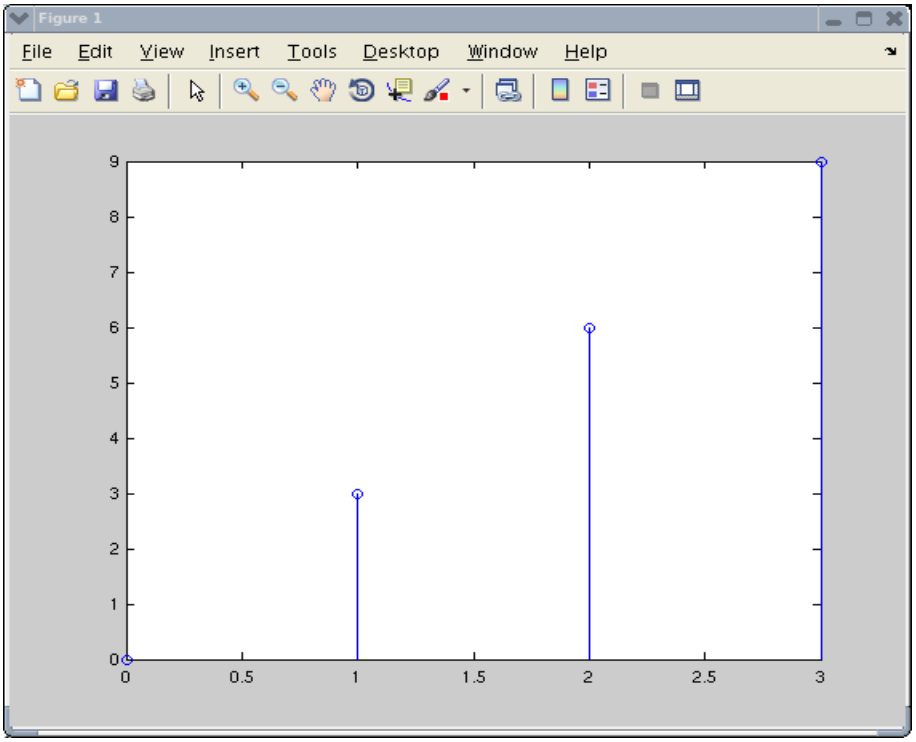


Figure 4: Results of stem(X,Y).

Descriptive labels to the x and y axes and a title may be added to plots by using the `xlabel`, `ylabel`, and `title` commands:

```
>> xlabel('X')
>> ylabel('Y = 3X')
>> title('A plot of the function y = 3x from x = 0 to x  = 3')
```

M-file Scripts:

MATLAB commands may be saved in a text file, such a file is called a MATLAB script M-file. The script may be loaded into MATLAB and the commands within the file will then be performed. A script may be created using any text editor such as Windows Notepad, or gedit, vi, emacs, etc. if you are using Linux. MATLAB has a built-in editor as well which may be accessed by selecting File -> New -> Script . The editor window will appear as shown in Figure 5.

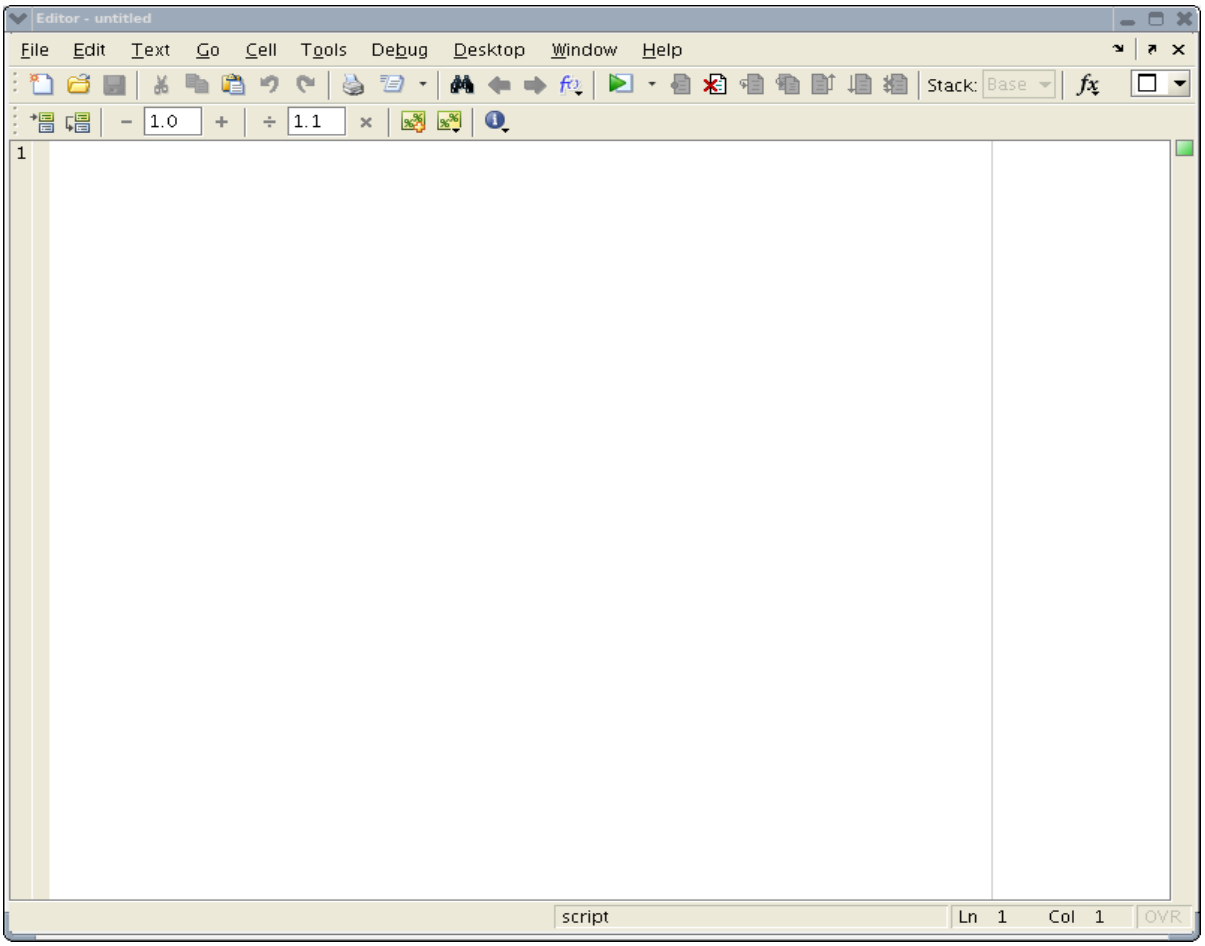


Figure 5: MATLAB text editor window.

Enter the following script and save it by selecting File -> Save and specify a file name such as “sample.m” in the Select File for Save As window.

```
% This is a comment
% Sample M-file script
% Ted Obuchowicz
% April 19, 2012

% Clear the workspace of any variables

clear
% define the input vector
X = [ 1 2 3 4];
%                              2
% define the output as y = x
Y = X .^2;
```

```
plot(X,Y)

% end of sample script
```

Note that this example script introduces a new MATLAB command called `clear`. The `clear` command clears the workspace of any variables (which may be present from some earlier MATLAB commands entered).

Once you have saved the file, exit from the editor by selecting File -> Close Editor

A script file may be loaded into MATLAB by typing in the name (the filename extension .m is not necessary and may be omitted) of the script from the MATLAB command prompt in the Command window:

```
>> sample
```

An alternative method of running a script is to select it with the mouse by navigating to its location in the Current folder window and right clicking and selecting Run from the pop-up menu.

Using the Help Command:

MATLAB has a built-in help utility. To learn more about a certain command simply type help followed by the command:

```
>> help plot
 plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, disconnected
    line objects are created and plotted as discrete points vertically at
    X.

    plot(Y) plots the columns of Y versus their index.
    If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
    In all other uses of plot, the imaginary part is ignored.

    Various line types, plot symbols and colors may be obtained with
    plot(X,Y,S) where S is a character string made from one element
    from any or all the following 3 columns:

           b     blue          .     point              -     solid
           g     green         o     circle             :     dotted
           r     red           x     x-mark             -.    dashdot
```

(the rest of the help output deleted for sake of brevity).

To obtain a list of help topics , simply type help:

```
>> help
HELP topics:

toolbox/local                   - General preferences and configuration infor-
mation.
matlab/general                  - General purpose commands.
matlab/ops                      - Operators and special characters.
matlab/lang                     - Programming language constructs.
matlab/elmat                    - Elementary matrices and matrix manipulation.
matlab/randfun                  - Random matrices and random streams.
matlab/elfun                    - Elementary math functions.
matlab/specfun                  - Specialized math functions.
matlab/matfun                   - Matrix functions - numerical linear algebra.
matlab/datafun                  - Data analysis and Fourier transforms.
matlab/polyfun                  - Interpolation and polynomials.
matlab/funfun                   - Function functions and ODE solvers.
matlab/sparfun                  - Sparse matrices.
matlab/strfun                   - Character strings.
matlab/iofun                    - File input and output.
matlab/timefun                  - Time and dates.
matlab/datatypes                - Data types and structures.
matlab/verctrl                  - Version control.
matlab/codetools                - Commands for creating and debugging code
matlab/helptools                - Help commands.
matlab/hds                      - (No table of contents file)
matlab/guide                    - Graphical user interface design environment
matlab/datamanager              - (No table of contents file)
matlab/graph2d                  - Two dimensional graphs.
```

Questions

In order to answer the following set of questions, a some self-study is required. Use the online help facility available in MATLAB to learn how to use the new commands introduced in each of the following questions. Create script files for each question. Include your name and ID as comments in the script files. For each question, submit as part of the written lab report the script file and any required plots.

Question 1:

Familiarize yourself with the `subplot` and `hold` command. Using the `subplot` and `plot` commands, plot the volume of a sphere as a function of it's radius for values of radius from 1 to 10. On the same graph plot the surface area of a sphere as a function of the radius (from radius = 1 to 10). Use different colors for the two plots (`help plot` will give details on how to select colors).

Use the following formulas:

Volume of sphere = (4/3) * pi * radius^3

Surface area of a sphere = 4 * pi * radius^2

Make use of the built-in MATLAB function called `pi` which returns the value of pi. The plots for the sphere's volume and surface area are to be plotted in the left-hand pane of the plot window.

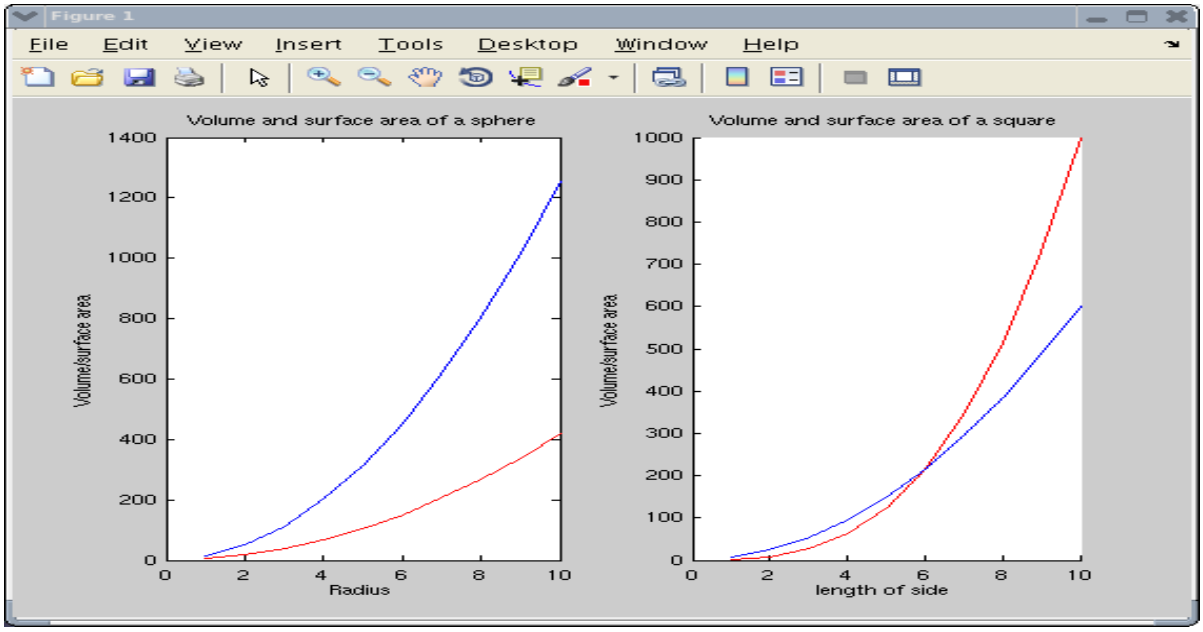
Repeat the above two plots, but this time plot the volume of a square and surface area of a square as a function of its length from 1 to 10. These plots are to be graphed in right-hand pane of the plot window.

Volume of a square = length of side^3

Surface area of a square = 6 * length of side^2

Label each axes as appropriate and include titles for the plots.

The plots should be similar to those shown below:



Question 2:

MATLAB has two useful functions when working with arrays: `zeros` and `ones`. To create a 10 element row_vector consisting of all zeros, the zeros function can be used as in:

```
>> Y1 = zeros(1,10)

Y1 =

     0     0     0     0     0     0     0     0     0     0
```

We can then use array element notation to set a few of these elements to some other value (1 for example):

```
>> Y1(3) = 1

Y1 =

     0     0     1     0     0     0     0     0     0     0
```

The `ones` function can be used to create a row vector consisting off all 1s :

```
>> Y2 = ones(1,10)

Y2 =

     1     1     1     1     1     1     1     1     1     1
```

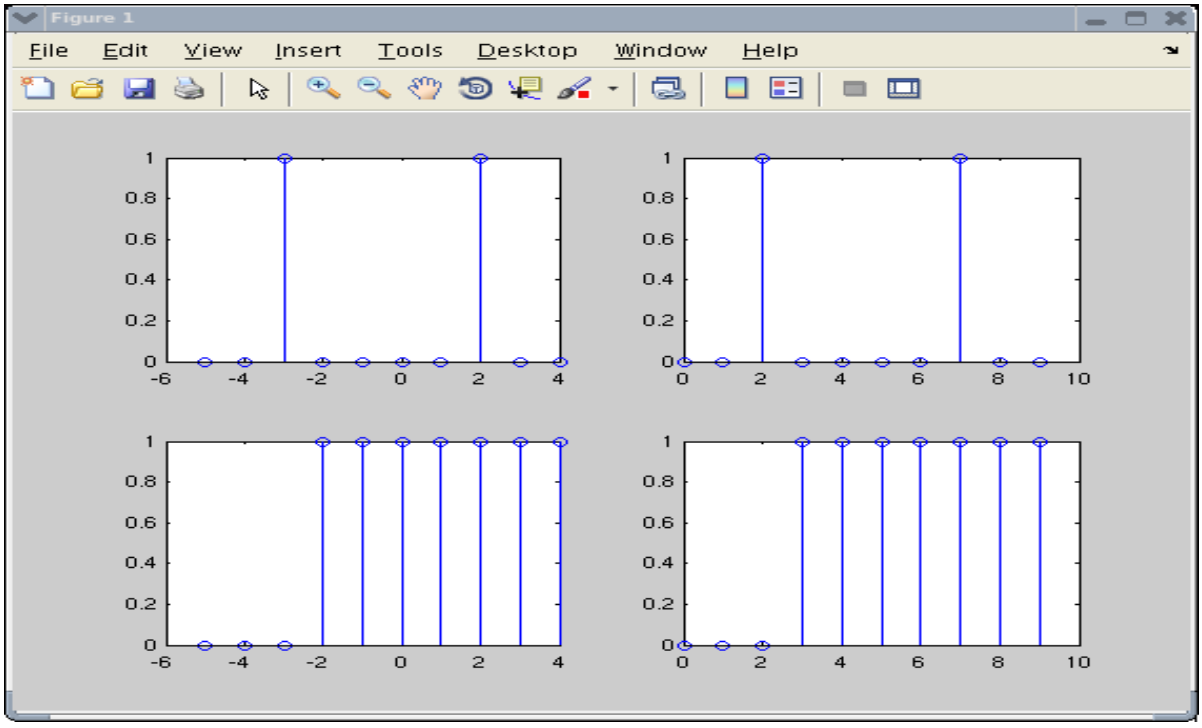
We can use the colon operator to specify a range of indices as in:

```
>> Y2(6:10) = 0

Y2 =

     1     1     1     1     1     0     0     0     0     0
```

The notation `Y2(6:10) = 0` is a convenient shorthand notation for:

```
Y2(6) = 0
Y2(7) = 0
Y2(8) = 0
Y2(9) = 0
Y2(10) = 0
```

Using the `zeros` and `ones` functions, create two different row_vectors: the first vector is to consist of all zeros except for two elements (which two are to be 1 is left up to you). The second vector is to consist of the first n elements being 0, and the remaining elements all equal to 1. The value of n is arbitrary. Plot in one window stem plots of these two vectors as a function of x as x varies from -5 to +4 in the left hand pane, and as x varies from 0 to 9 in the right half pane. Refer to the following figure for details:

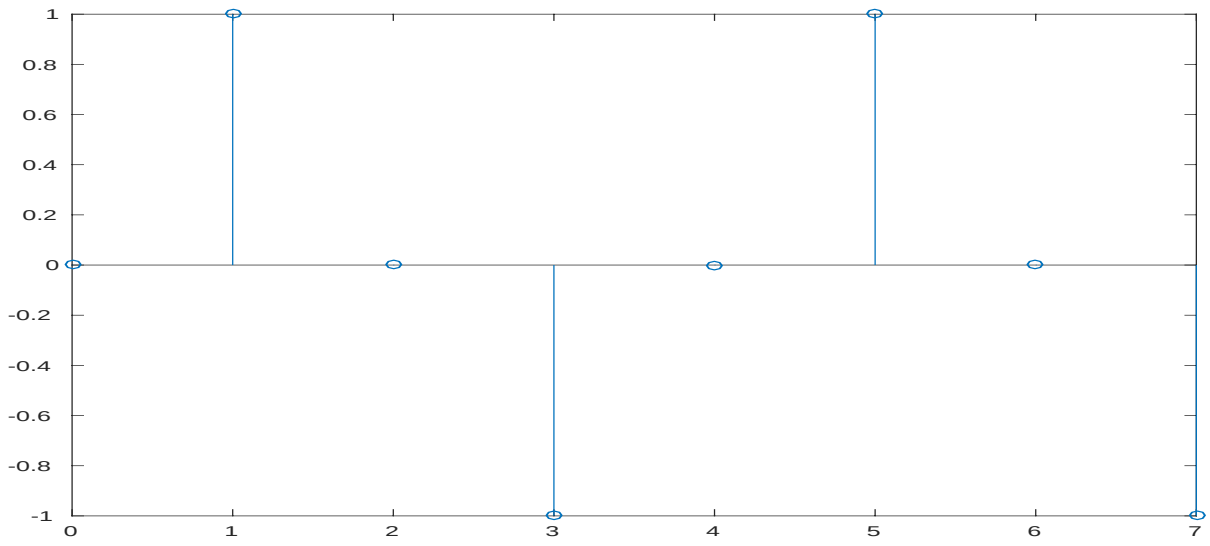


Question 3:

Consider the following MATLAB code:

```
>> n = [ 0 : 7 ] ;
>> N = 4 ;
>> x = sin ( (2*pi)/N * n );
>> stem(n ,x ) ;
```

The value of N defines the number of samples per period, and n defines the total number of data points stored. In the above example, we are sampling the discrete time sinusoidal signal 4 times every period and storing these sampled values into a vector consisting of 8 data points. The resulting stem plot is:



With more samples per period, the sampled signal more closely resembles a sine wave.
You can easily see the effect by doing:

```
>> N = 16 ;
>> n = [ 0 : 31  ] ;
>> x = sin ( (2*pi)/N * n );
>> stem(n,x);
```

(i) A signal X[n] is said to be periodic (with period T) if X[n + T] = x[n]. This question investigates the effect how floating point numbers are stored internally within a computer. Sample the signal x = cos((2*pi)/1024 * n) over 4 periods and then determine whether x[n] = x[n + 1024]. For example, you can compute the difference of x[1] - x[1+1024] to see if it is equal to 0.0. It would be useful to use the:

```
>> format long
```

command so that doubles are displayed to their full 15 decimal place accuracy.

(ii) Instead of using the MATLAB constant, `pi`, use the value of `3.14` and compare your results with those obtained in part (i) .

(iii) Determine (using MATLAB) whether the signals:

x1 = cos(pi/4* n + pi/3)

and

x2 = cos(9*pi/4.*n + pi/3)

are equal to each other.

Question 4:

MATLAB has the capability of reading information stored in ASCII text files and assigning this information to variables. Create an ASCII text file consisting of the following 100 numbers:

```
 1   2   3   4   5   6   7   8   9  10
11  12  13  14  15  16  17  18  19  20
21  22  23  23  24  26  27  28  29  30
31  32  33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48  49  50
51  52  53  54  55  56  57  58  59  60
61  62  62  64  65  66  67  68  69  70
71  72  72  74  75  76  77  78  79  80
81  82  83  84  85  86  87  88  89  90
91  92  93  94  95  96  97  98  99 100
```

Save the file with filename “my_big_array.txt”

Use the `load` command to read in these 100 numbers and store the values into an array called `my_big_array`.

References

1. *MATLAB Programming*, David C. Kuncicky, Pearson Education Inc., 2004.

Recommended texts

The following is a list of useful MATLAB programming texts:

1. *MATLAB Programming*, David C. Kuncicky, Pearson Education Inc., ISBN 0-13-035127-X 2004.

2. *MATLAB for Engineering Applications*, William J. Palm III, McGraw-Hill, ISBN 0-07-047330-7, 1999.

3. *MATLAB: A practical introduction to programming and problem solving*, Stormy Attaway, Butterworth-Heinemann, ISBN 978-0-12-385081-2.

4. *MATLAB 5 for Engineers*, Adrian Biran and Moshe Breiner, Addison-Wesley, ISBN 0-201-36043-8, 1999.

T. Obuchowicz
May 2, 2016