CpE 3203L – Digital Signal Processing

Lab Exercise 1 : Review of MATLAB

Name: _____ ID #: _____ Score: _____
Instructor: _____ Schedule: _____ Date: _____

Objectives:

- Review of the MATLAB environment and working within it
- Create MATLAB files or m-files, learn the programming language fundamentals

Outcomes:

- Solve matrix problems using MATLAB commands
- Create a graph of a signal
- Write MATLAB code and functions using m-files or scripts

I. Introduction to MATLAB

MATLAB® is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable you to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java®. You can use MATLAB for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

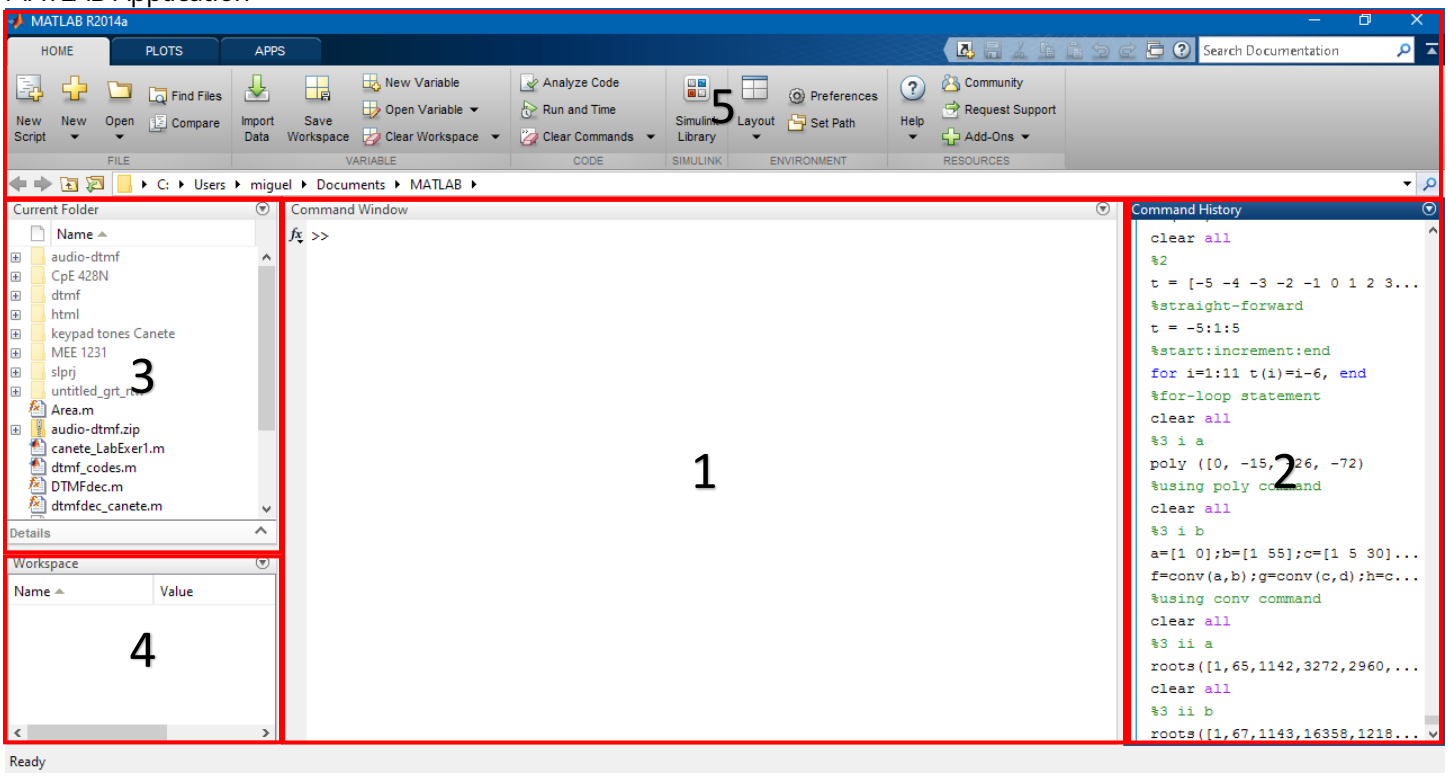
Key Features

- High-level language for numerical computation, visualization, and application development
- Interactive environment for iterative exploration, design, and problem solving
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations
- Built-in graphics for visualizing data and tools for creating custom plots
- Development tools for improving code quality and maintainability and maximizing performance
- Tools for building applications with custom graphical interfaces
- Functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET, and Microsoft® Excel®. †

In this course we will start with focusing on the basics of using MATLAB, learning how to interact within the workspace, using commands, creating simple programs, and using the Help feature. A lot of information and examples can be found in the Help documentation. Whenever you have difficulty with a command or feature, directly open the Help Documentation, it might be faster and easier than asking from google.

† Taken from Matlab Product Documentation.

MATLAB Application



The interface of MATLAB is divided into 5 general areas:

1. *Command Window*
   The command window is the primary way for users to interact with the application. It works very similar to the "cmd" for Windows OS and "terminal" for Linux. MATLAB commands are typed into this area and is marked by ">>" symbol. Any command you put into the area will be shown after this marker. Example.

   ```
   >> why
   >> help why
   ```

2. *Command History*
   From the name itself, this window shows all the previous commands that was typed into the Command Window. This is useful during times when users forget the commands from the previous sessions in MATLAB.

3. *Current Folder*
   Working with MATLAB will also involve working with files and directories. This window will show the working directory and serves as an internal file/folder explorer.

4. *Workspace*
   Variables used and generated in MATLAB will be shown here. The window will show the variable name and value. The icon will also indicate the variable type. Other applicable information will be shown here depending on the variable type.

5. *Toolstrip*
   Toolstrip is equivalent to the "Ribbon" in Microsoft Office applications. Main features and functions are placed here. There are 3 main tabs: HOME; PLOT and APPS. Home tab is for the file related, variable related, environment related, and others. The Plot tab is for ease of plotting the variable in the Workspace. Apps tab is were specialized functions can be found. MATLAB has an option to install other functions sets design for specific fields or topics, such as Image Processing and Digital Signal Processing among others.

It will not always look like the way it does for you and your seatmates current configuration but you can always change this around to however you like. Just make sure you remember what they are and how to use them. Configuration can be

changed in the HOME tab of the Toolstrip, under the 'Environment' section then 'Layout' button. There are predefined configuration you can also use.

II. MATLAB Basics

Working with MATLAB involves issuing commands thru the Command window to create variables and call functions. For example, to create a variable,

```
a = 1
```

This will create a variable that holds a number (default type is double) with a value of '1'. You should see the response of MATLAB to your command also in the same window. It will look something like this,

```
a =
      1
```

Let's create other variables,

```
b = 2
c = a + b
d = cos(a)
```

These variables and all others that you create will show up in the Workspace window. MATLAB can also store other types of data such as strings and floating point numbers. Test them out by creating your own variables.

When you do not specify an out variable, MATLAB uses the default `ans`, short for *answer*, to store the results of your calculations or commands.

```
sin(a)
ans =
      0.8415
```

If you end a command with a semicolon, the computation will be executed, but the display of the output in the command window will be suppressed.

```
e = a*b;
```

`cos(a)` and `sin(a)` are called functions in MATLAB, which are also mathematical functions themselves. Almost all of the usual math functions have an equivalent command in MATLAB. Try them out yourself. There are a lot of functions that are included with MATLAB, they range from the fundamental to the advanced level in mathematics and other topics such as, statistics, digital signal processing and, of course, control systems.

Array Creation

MATLAB is an abbreviation for "matrix laboratory'. Other programming languages work with numbers one at a time, MATLAB is designed to operate mainly on whole matrices and arrays. All MATLAB variables are multidimensional arrays, no matter what type of data.

To create and array with four elements in a single row, separate the elements with either a comma (,) or a space.

```
a = [1 2 3 4]
```

returns

```
a =
      1     2     3     4
```

This type of array is a *row vector*.

To create a matrix that has multiple rows, separate the rows with semicolons.

```
a = [1 2 3; 4 5 6; 7 8 10]
a =

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

Another way to create a matrix is to use a function, such as `ones`, `zeros`, or `rand`. For example, create a 5-by-1 column vector of zeros.

```
z = zeros(5,1)
z =

     0
     0
     0
```

```
     0
     0
```

Matrix and Array Operations

MATLAB allows you to process all of the values in a matrix using a single arithmetic operator or function.

```
a + 10
ans =

    11    12    13
    14    15    16
    17    18    20
sin(a)
ans =

    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
    0.6570    0.9894   -0.5440
```

To transpose a matrix, use a single quote (').

```
a'
ans =

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

You can perform standard matrix multiplication, which computes the inner products between rows and columns, using the * operator. For example, confirm that a matrix times its inverse returns the identity matrix:

```
p = a*inv(a)
p =

    1.0000         0   -0.0000
         0    1.0000         0
         0         0    1.0000
```

Notice that p is not a matrix of integer values. MATLAB stores numbers as floating-point values, and arithmetic operations are sensitive to small differences between the actual value and its floating-point representation. You can display more decimal digits using the format command:

```
format long
p = a*inv(a)
p =

   1.000000000000000                   0  -0.000000000000000
                   0   1.000000000000000                   0
                   0                   0   0.999999999999998
```

Reset the display to the shorter format using

```
format short
```

`format` affects only the display of numbers, not the way MATLAB computes or saves them.

To perform element-wise multiplication rather than matrix multiplication, use the .* operator:

```
p = a.*a
p =

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

The matrix operators for multiplication, division, and power each have a corresponding array operator that operates element-wise. For example, raise each element of a to the third power:

```
a.^3


ans =

           1           8          27
          64         125         216
         343         512        1000
```

Concatenation

Concatenation is the process of joining arrays to make larger ones. In fact, you made your first array by concatenating its individual elements. The pair of square brackets [] is the concatenation operator.

```
A = [a,a]
A =

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

Concatenating arrays next to one another using commas is called horizontal concatenation. Each array must have the same number of rows. Similarly, when the arrays have the same number of columns, you can concatenate vertically using semicolons.

```
A = [a; a]
A =

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

Array Indexing

Every variable in MATLAB is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 magic square A:

```
A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

```
A(4,2)
ans =
    14
```

Less common, but sometimes useful, is to use a single subscript that traverses down each column in order:

```
A(8)
ans =
    14
```

Using a single subscript to refer to a particular element in an array is called linear indexing.

If you try to refer to elements outside an array on the right side of an assignment statement, MATLAB throws an error.

```
test = A(4,5)
Attempted to access A(4,5); index out of bounds because size(A)=[4,4].
```

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

```
A(4,5) = 17
A =
    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form `start:end`. For example, list the elements in the first three rows and the second column of A:

```
A(1:3,2)
ans =
     2
    11
     7
```

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of A:

```
A(3,:)
ans =
     9     7     6    12     0
```

The colon operator also allows you to create an equally spaced vector of values using the more general form `start:step:end`.

```
B = 0:10:100
B =
     0    10    20    30    40    50    60    70    80    90   100
```

If you omit the middle step, as in `start:end`, MATLAB uses the default step value of 1.

Character Strings

A character string is a sequence of any number of characters enclosed in single quotes. You can assign a string to a variable.

```
myText = 'Hello, world';
```

If the text includes a single quote, use two single quotes within the definition.

```
otherText = 'You''re right'
otherText =
      You're right
```

`myText` and `otherText` are arrays, like all MATLAB variables. Their class or data type is char, which is short for character.

```
whos myText
  Name        Size            Bytes  Class    Attributes

  myText      1x12               24  char
```

You can concatenate strings with square brackets, just as you concatenate numeric arrays.

```
longText = [myText,' - ',otherText]
longText =
      Hello, world - You're right
```

To convert numeric values to strings, use functions, such as `num2str` or `int2str`.

```
f = 71;
c = (f-32)/1.8;
tempText = ['Temperature is ',num2str(c),'C']
tempText =

Temperature is 21.6667C
```

Help and Documentation

All MATLAB functions have supporting documentation that includes examples and describes the function inputs, outputs, and calling syntax. There are several ways to access this information from the command line:

Open the function documentation in a separate window using the `doc` command.

```
doc mean
```

Display function hints (the syntax portion of the function documentation) in the Command Window by pausing after you type the open parentheses for the function input arguments.

```
mean(
```

View an abbreviated text version of the function documentation in the Command Window using the help command.

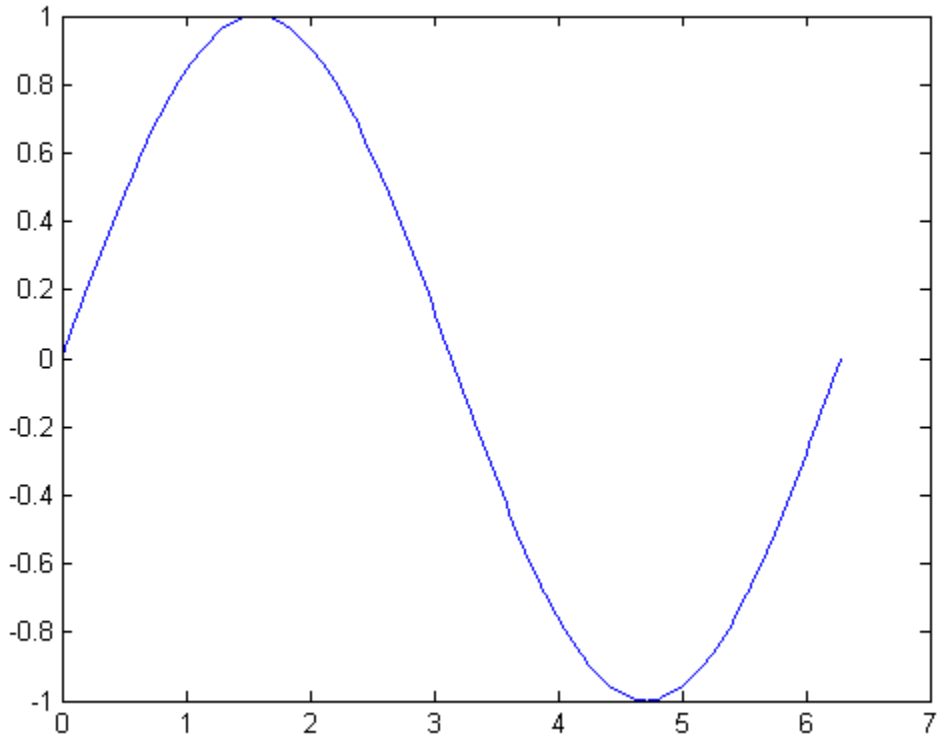
```
help mean
help sin
help max
help rand
help help
```

Access the complete product documentation by clicking the help icon.

Line Plots

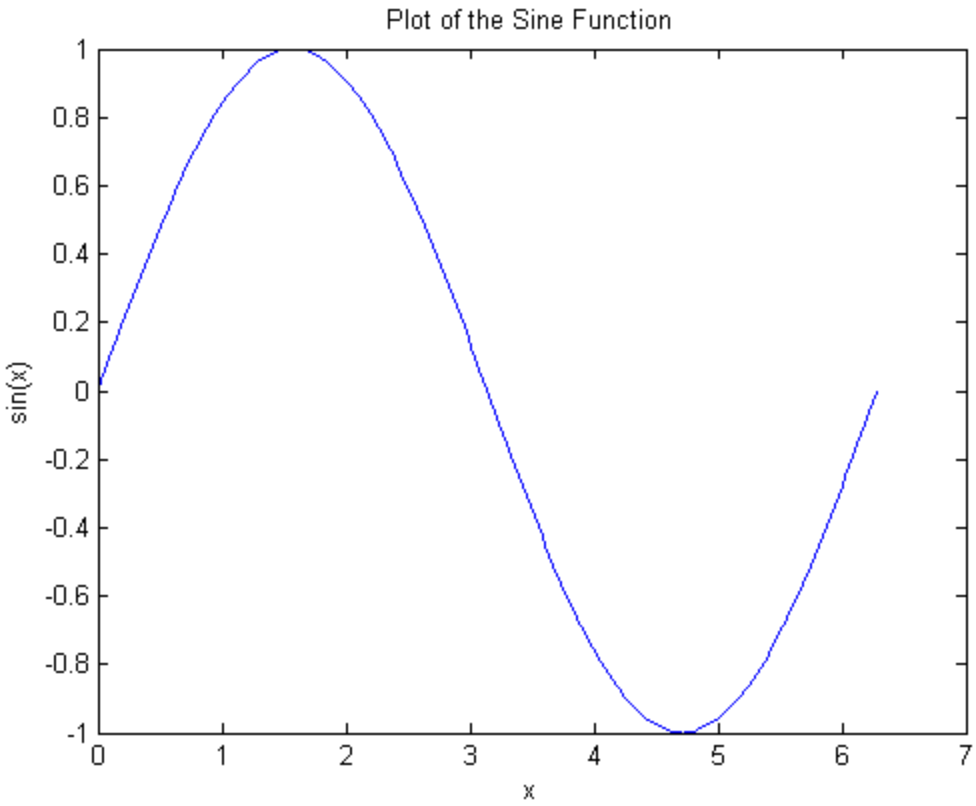
To create two-dimensional line plots, use the plot function. For example, plot the value of the sine function from 0 to :

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



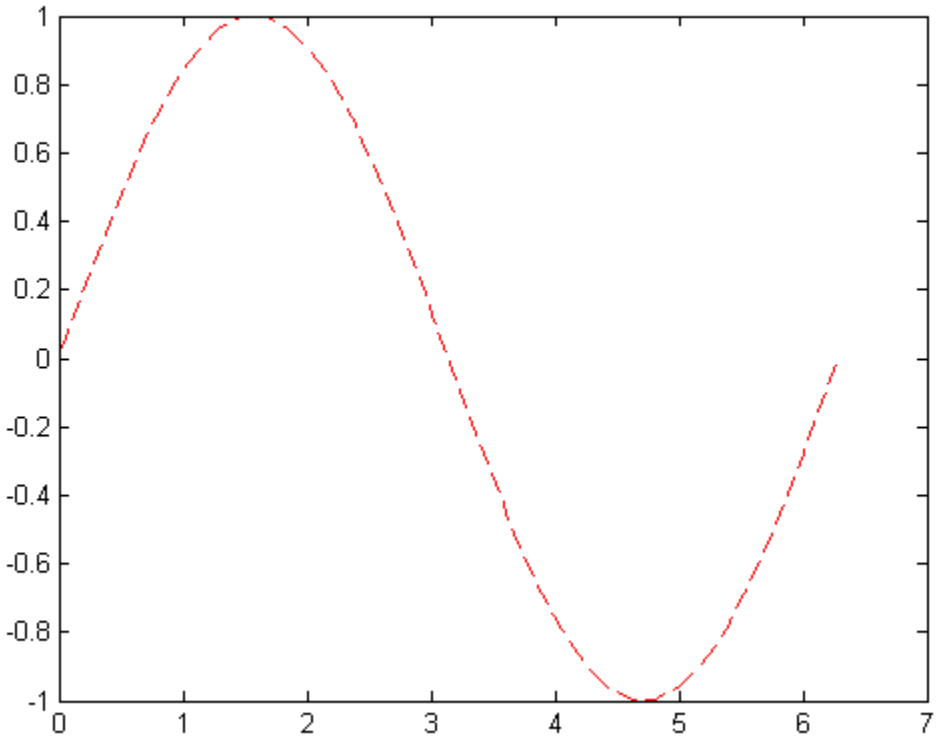
You can label the axes and add a title.

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```



By adding a third input argument to the plot function, you can plot the same variables using a red dashed line.

```
plot(x,y,'r--')
```



The 'r--' string is a *line specification*. Each specification can include characters for the line color, style, and marker. A marker is a symbol that appears at each plotted data point, such as a +, o, or *. For example, 'g:*' requests a dotted green line with * markers.

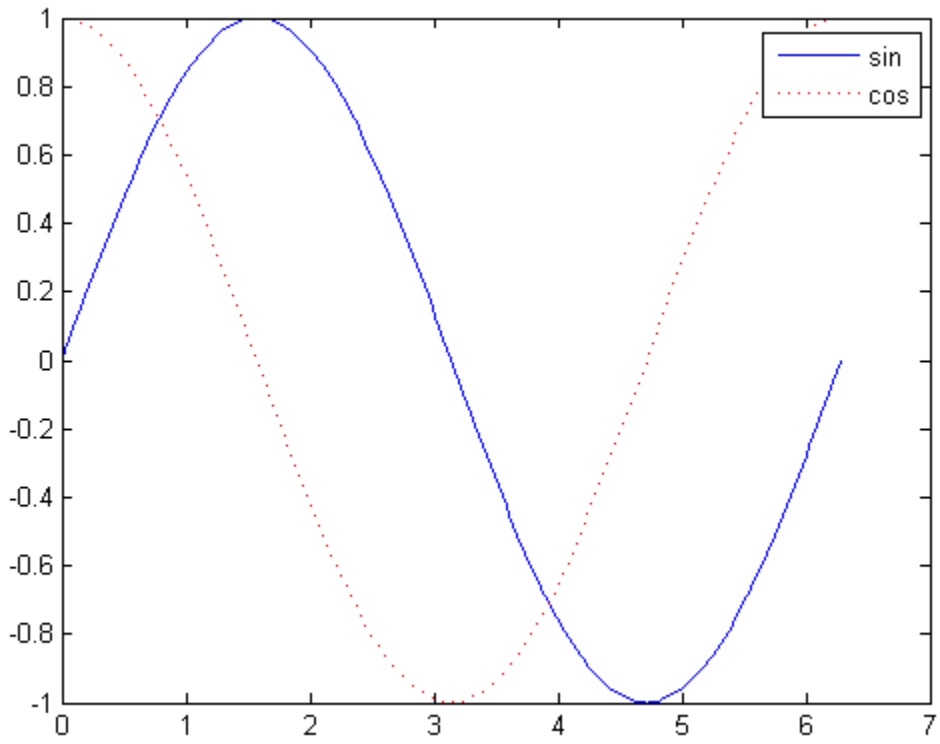
Notice that the titles and labels that you defined for the first plot are no longer in the current figure window. By default, MATLAB clears the figure each time you call a plotting function, resetting the axes and other elements to prepare the new plot.

To add plots to an existing figure, use `hold`.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)

hold on

y2 = cos(x);
plot(x,y2,'r:')
legend('sin','cos')
```



Until you use `hold off` or close the window, all plots appear in the current figure window.

Polynomials

MATLAB represents polynomials as row vectors containing coefficients ordered by descending powers. For example, consider the equation

$$p(x) = x^3 - 2x - 5$$

This is the celebrated example Wallis used when he first represented Newton's method to the French Academy. To enter this polynomial into MATLAB, use

```
p = [1 0 -2 -5];
```

The `polyval` function evaluates a polynomial at a specified value. To evaluate p at $s = 5$, use

```
polyval(p,5)
ans =
   110
```

It is also possible to evaluate a polynomial in a matrix sense. In this case $p(x) = x3 - 2x - 5$ becomes $p(X) = X\,3 - 2X - 5I$, where X is a square matrix and I is the identity matrix. For example, create a square matrix X and evaluate the polynomial p at X:

```
X = [2 4 5; -1 0 3; 7 1 5];
Y = polyvalm(p,X)

Y =
   377   179   439
   111    81   136
   490   253   639
```

The `roots` function calculates the roots of a polynomial:

```
r = roots(p)
r =
     2.0946
    -1.0473 +      1.1359i
    -1.0473 -      1.1359i
```

By convention, MATLAB stores roots in column vectors. The function poly returns to the polynomial coefficients:

```
p2 = poly(r)
p2 =
     1   8.8818e-16   -2   -5
```

`poly` and `roots` are inverse functions, up to ordering, scaling, and roundoff error.

Polynomial multiplication is done by using the `conv`, short for convolution.

```
p3 = conv([1 2 3],[2 0 5])
p3 =
     2     4    11    10    15
```

Executing commands automatically

MATLAB has a built-in text editor. This is used for editing text-based files that contain a series of MATLAB commands. You can save the file with the extension, '.m'. These files are also called 'm-files'. Try creating a file with the following commands,

```
clc %command for clearing the command window
clear %command for deleting all variables
disp('hello world'); %display universal programming string
matA = rand(4); %create a 4by4 random matrix
matA.*3
t = 0:0.5:10;
y = cos(t*pi); %MATLAB also has common constants in math
plot(t,y)
```

Save this file with the name "myFirstmFile.m". This will be saved in your current working directory which you should see in the Current Folder window. To execute the file, just type the filename in the Command Window

```
myFirstmFile.m
```

The results will be shown except for the commands were the output is suppressed.

Activity #1a

Execute the following commands in MATLAB. Create a file with the filename "Activity1a.txt" which contains the output of the commands to be executed on the command line.

1. Perform the following using MATLAB
   a. $A + B$, where
   $$A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -3 & 4 \end{bmatrix}$$
   $$B = \begin{bmatrix} 3 & -1 & 3 \\ 2 & 0 & 5 \end{bmatrix}$$
   b. $3A - 2B$, where
   $$A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$
   $$B = \begin{bmatrix} 1 & 3 \\ 0 & -4 \end{bmatrix}$$
   c. $5A - 2B$ using the matrices in item 1.a.

2. Perform $C(A + B)$, where
   $$A = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$
   $$B = \begin{bmatrix} 2 & -1 \\ 3 & 4 \end{bmatrix}$$
   $$C = \begin{bmatrix} 2 & -2 \\ 1 & 3 \\ 4 & -1 \end{bmatrix}$$

3. Using matrices in item 2, perform $CA + CB$

4. Create a plot within a single figure of the following signals with signal a in red line and signal b in dashed blue line, HINT: you can use the polyval command with a row vector
   a. $y = x^2 + 5x + 3$
   b. $y = x^3 + 4$

5. Create a plot using `subplot` command to show the two signals in item 4 in a single figure but two plots. Add an appropriate title for each plot in the figure.

6. Determine the roots of the following polynomials,
   a. $s^7 + 32s^6 + 8s^5 + 85s^4 + 4s^3 + s^2 + 3s + 1$
   b. $3s^5 - s^4 + 24s^3 + 9s^2 + 6s + 2$
   c. $s^3 + 77s^2 + 11s + 1$

III. Programming Language Basics

We have already learned that we can make a file that will run a group of commands instead of the 'one command: one output' usage pattern. Programming concepts are also implemented in MATLAB's language of computing. The essential ones being: Control Flow, Scripts or Functions and Debugging.

Control Flow

Control flow is the collective term used to include conditional and loop control statements. Let's start first with conditional statements. They enable you to select at run time which block of code to execute. The simplest conditional statement is and `if` statement. For example:

```
% Generate a random number
a = randi(100, 1);

%If it is even, divide by 2
If rem(a, 2) == 0
      Disp('a is even');
      b = a/2;
end
```

`if` statements can include alternate choices, using the optional keywords `elseif` or else. For example:

```
a = randi(100, 1);

if a < 30
    disp('small')
elseif a < 80
    disp('medium')
else
    disp('large')
end
```

Alternatively, when you want to test for equality against a set of known values, use a `switch` statement. For example:

```
 [dayNum, dayString] = weekday(date, 'long', 'en_US');

switch dayString
   case 'Monday'
      disp('Start of the work week')
   case 'Tuesday'
      disp('Day 2')
   case 'Wednesday'
      disp('Day 3')
   case 'Thursday'
      disp('Day 4')
   case 'Friday'
      disp('Last day of the work week')
   otherwise
      disp('Weekend!')
end
```

For both `if` and `switch`, MATLAB executes the code corresponding to the first true condition, and then exits the code block. Each conditional statement requires the end keyword.

In general, when you have many possible discrete, known values, `switch` statements are easier to read than `if` statements. However, you cannot test for inequality between `switch` and `case` values. For example, you cannot implement this type of condition with a switch:

```
yourNumber = input('Enter a number: ');

if yourNumber < 0
    disp('Negative')
elseif yourNumber > 0
    disp('Positive')
else
```

```
    disp('Zero')
end
```

With loop control statements, you can repeatedly execute a block of code. There are two types of loops:

- `for` statements loop a specific number of times, and keep track of each iteration with an incrementing index variable.
  For example, preallocate a 10-element vector, and calculate five values:
  ```
  x = ones(1,10);
  for n = 2:6
      x(n) = 2 * x(n - 1);
  end
  ```

- `while` statements loop as long as a condition remains true.
  For example, find the first integer `n` for which `factorial(n)` is a 100-digit number:
  ```
  n = 1;
  nFactorial = 1;
  while nFactorial < 1e100
      n = n + 1;
      nFactorial = nFactorial * n;
  end
  ```

Each loop requires the `end` keyword.

It is a good idea to indent the loops for readability, especially when they are nested (that is, when one loop contains another loop):

```
A = zeros(5,100);
for m = 1:5
    for n = 1:100
        A(m, n) = 1/(m + n - 1);
    end
end
```

You can programmatically exit a loop using a `break` statement, or skip to the next iteration of a loop using a `continue` statement. For example, count the number of lines in the help for the `magic` function (that is, all comment lines until a blank line):

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
       break
    elseif ~strncmp(line,'%',1)
       continue
    end
    count = count + 1;
end
fprintf('%d lines in MAGIC help\n',count);
fclose(fid);
```

TIP: If you inadvertently create an infinite loop (a loop that never ends on its own), stop execution of the loop by pressing Ctrl+C..

Control flow can be implemented on scripts and functions but not directly on the Command Window. Program files can be scripts that simply execute a series of MATLAB statements, or they can be functions that also accept input arguments and produce output. Both scripts and functions contain MATLAB code, and both are stored in text files with a '.m' extension. However, functions are more flexible and more easily extensible.

Relational Operators

These operators are used compare operands quantitatively with operators like "less than" and "not equal to." The following table provides a summary.

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

The relational operators compare corresponding elements of arrays with equal dimension. They always operate element-by-element. In this code example, the resulting matrix shows where an element of `A` is equal to the corresponding element of `B`.

```
A = [2 7 6;9 0 5;3 0.5 6];
B = [8 7 0;3 2 5;4 -1 7];

A == B
ans =
     0     1     0
     0     0     1
     0     0     0
```

These operators are also applicable to the control flow statements in the previous section. Recall your experience in other standard programming languages, such as C programming. The concepts for checking conditions can still be applied to MATLAB.

Scripts vs. Functions

Program files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that also accept input arguments and produce output. Both scripts and functions contain MATLAB code, and both are stored in text files with a `.m` extension. However, functions are more flexible and more easily extensible.

To create a file in MATLAB, you can use the shortcut key "`ctrl+n`", this will open a blank .m file in the Editor window. You can also use the Toolstrip to create a blank .m file or a specific type of file. This can be accessed in the HOME tab of the Toolstrip under the FILE section.

For example, create a script in a file named `triarea.m` that computes the area of a triangle:

```
b = 5;
h = 3;
a = 0.5*(b.* h)
```

After you save the file, you can call the script from the command line:

```
triarea
a =
    7.5000
```

To calculate the area of another triangle using the same script, you could update the values of `b` and `h` in the script and rerun it. Each time you run it, the script stores the result in a variable named `a` that is in the base workspace.

However, instead of manually updating the script each time, you can make your program more flexible by converting it to a function. Replace the statements that assign values to b and h with a function declaration statement. The declaration includes the function keyword, the names of input and output arguments, and the name of the function.

```
function a = triarea(b,h)
a = 0.5*(b.* h);
```

After you save the file, you can call the function with different base and height values from the command line without modifying the script:

```
a1 = triarea(1,5)
a2 = triarea(2,10)
a3 = triarea(3,6)

a1 =
    2.5000
a2 =
    10
a3 =
     9
```

Functions have their own workspace, separate from the base workspace. Therefore, none of the calls to the function `triarea` overwrite the value of `a` in the base workspace. Instead, the function assigns the results to variables `a1`, `a2`, and `a3`.

Activity #1b

Create a function that corresponds to each items using MATLAB. Save each function inside a folder ("Activity1b"). Use appropriate function names as your filenames for each item below. Create a script named "run_all.m" that will execute all the functions below with appropriate parameters and arguments based on the function.

- Fibonacci
  Write a function that accepts two integer values. It should display the Fibonacci numbers that exist within the range of the numbers given.

```
>>Fibonacci(1,10)
      1 1 2 3 5 8
```

- Area
  Create a function that accepts multiple input parameters. Choose your own formulas for calculating the area of standard shapes (i.e. circle, pentagon, square, triangle, etc.). And it should be able to solve atleast 5 shapes.

```
>>AreaOfLife('circle',10)
      314.1592
```

- Palindrome
  Create a function that determines if the input string is a palindrome or not. A palindrome is a word that can be spelled in reverse and still read the same word.
  Palindrome: racecar, pop, anna, aibohphobia, etc
  Not a palindrome: you, Palindrome

- Plot
  A function that accepts a string that identifies the signal that will be plotted. The function must have the following prototype:

  function BasicPlotter(type, param_vector, time_vector)

  type – is a string that determines the signal or mathematical function type. BasicPlotter must be able to plot sin, cos, sinc and normal (normal distribution) functions.

  param_vector – is a vector of arbitrary length depending on the function type e.g. f for frequency of sin and cos.

  time_vector – is a time series vector starting from any number increasing at equal intervals until the final value e.g. `-1 : 0.01 : 1`