



Pervasive Computing - Week 2

Introduction to MATLAB®

1. Learning Outcomes

In this lab, you will:

- Become familiar with the MATLAB programming environment.
- Learn basic Matrix manipulation.
- Use some useful built-in MATLAB functions.
- Learn how to visualise and manipulate data.

1.2 Tasks


1. Introduction to the MATLAB environment and setting the current directory
2. A basic introduction on how to define and manipulate vectors in MATLAB. This is the most basic way that numbers are stored and accessed in MATLAB.
3. An introduction on how to define and manipulate matrices. We demonstrate how to create matrices and how to access parts of a matrix.
4. The power of MATLAB is that the basic operations defined in linear algebra can be carried out with similar notation and a minimal number of programming steps.
5. We introduce the basic loop construct used in MATLAB. We show how to define a for loop and provide an example of how it can be used to solve a problem.
6. A general overview of the basic plotting commands is given.

1.3 Introduction

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. If you would like to learn more about the power of MATLAB, visit the [Onramp MATLAB Academy](#).

Open MATLAB from the start menu. The main MATLAB environment has 4 main windows. A screen shot of the MATLAB environment is shown in **Figure 1**. The Current Folder window shows the files available in the current folder within the directory. These are the files that can be called or loaded from the MATLAB Command Window. The Command Window is where code is typed and executed. The Workspace window is where variables are stored and viewed, and the Command History window shows commands which have been typed previously in the Command Window.

Note: To dock the Command History window, as shown in Figure 1: Click on the HOME tab > Layout > Command History > Docked

- Access the complete product documentation by clicking the help icon .

2.2 Getting Started

As you work in MATLAB, you issue commands that create variables and call functions. For example, create a variable named `a` by typing this statement at the command line:

```
>> a = 1
```

MATLAB adds variable `a` to the workspace and displays the result in the Command Window.

```
a =
```

```
1
```

Create a few more variables.

```
>> b = 2
```

```
b =
```

```
2
```

```
>> c = a + b
```

```
c =
```

```
3
```

```
>> d = cos(a)
```

```
d =
```

```
0.5403
```

When you do not specify an output variable, MATLAB uses the variable `ans`, short for *answer*, to store the results of your calculation.

```
>> sin(a)
```

```
ans =
```

```
0.8415
```

If you end a statement with a semicolon, MATLAB performs the computation, but suppresses the display of output in the Command Window.

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

You can recall previous commands by pressing the up- and down-arrow keys, ↑ and ↓. Press the arrow keys either at an empty command line or after you type the first few characters of a command. For example, to recall the command `b = 2`, type `b`, and then press the up-arrow key.

2.3 Matrices and Arrays

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional **arrays**, no matter what type of data. A **matrix** is a two-dimensional array often used for linear algebra.

2.3.1 Array Creation

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

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

2.4 Matrices and Arrays Operators

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 (i.e. interchange the row and column index of each element), 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	-0.0000	-0.0000
0.0000	1.0000	-0.0000
0.0000	-0.0000	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 =
```

1.0000000000000000	-0.0000000000000000	-0.0000000000000000
0.0000000000000001	0.9999999999999999	-0.0000000000000000

```
0.0000000000000002      -0.0000000000000003      1.0000000000000000
```

Reset the display to the shorter format using

```
>> format short
```

Note: `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
```

2.4.1 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
```

Note: MATLAB® is case sensitive, so `A` and `a` are *not* the same variable.

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

2.5 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)
```

Index in position 2 exceeds array bounds (must not exceed 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.

2.6 Workspace Variables

The *workspace* contains variables that you create within or imported into MATLAB® from data files or other programs. For example, these statements create variables A and B in the workspace.

```
>> A = magic(4);  
>> B = rand(3,5,2);
```

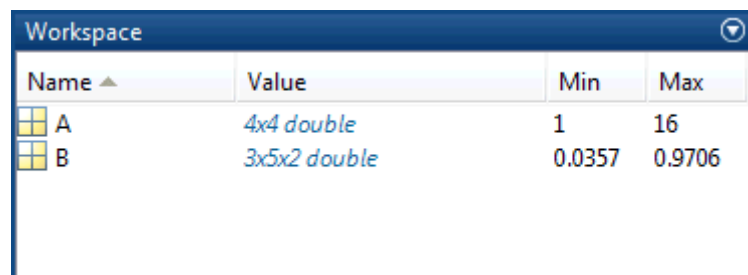
You can view the contents of the workspace using `whos`.

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	4x4	128	double	
B	3x5x2	240	double	

Note: `whos` lists in alphabetical order the name, sizes and types of all variables currently active in the workspace. Find newly created variables A and B at the top of the list.

The variables also appear in the Workspace pane on the desktop.



Workspace variables do not persist after you exit MATLAB. Save your data for later use with the `save` command,

```
>> save myfile.mat
```

Saving preserves the workspace in your current working folder in a compressed file with a `.mat` extension, called a MAT-file.

To clear all the variables from the workspace, use the `clear` command.

Restore data from a MAT-file into the workspace using `load`.

```
>> load myfile.mat
```

2.7 Text and Characters

When you are working with text, enclose sequences of characters in single quotes. You can assign text 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 character arrays with square brackets, just as you concatenate numeric arrays.

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

To convert numeric values to characters, 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'
```

2.8 Calling Functions

MATLAB® provides a large number of functions that perform computational tasks. Functions are equivalent to *subroutines* or *methods* in other programming languages.

To call a function, such as `max`, enclose its input arguments in parentheses:

```
>> A = [1 3 5];
```

```
>> max(A)

ans =

     5
```

If there are multiple input arguments, separate them with commas:

```
>> B = [10 6 4];

>> max(A,B)

ans =

    10     6     5
```

Return output from a function by assigning it to a variable:

```
>> maxA = max(A)

maxA =

     5
```

When there are multiple output arguments, enclose them in square brackets:

```
>> [maxA, location] = max(A)

maxA =

     5

location =

     3
```

Enclose any character inputs in single quotes:

```
>> disp('hello world')

hello world
```

To call a function that does not require any inputs and does not return any outputs, type only the function name, such as:

```
>> clc
```

The `clc` function clears the Command Window.

2.9 Programming and Scripts

The simplest type of MATLAB® program is called a **script**. A script is a file with a `.m` extension that contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

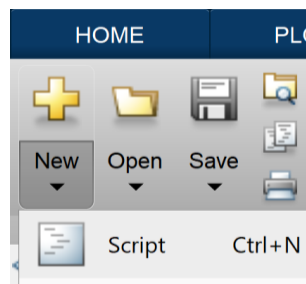
2.9.1 Creating a Script

To create a script, use the `edit` command,

```
>> edit myFirstScript.m
```

A dialogue box will then pop up asking would you like to create this new script.

Or, you can create a new untitled script by clicking on New Script in the toolstrip. Or using the keyboard short cut Ctrl+N.

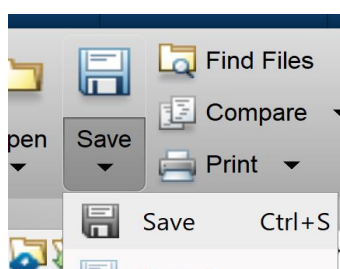


Whenever you write code, it is a good practice to add comments that describe the code. Comments allow others to understand your code, and can refresh your memory when you return to it later. Add comments using the percent (%) symbol.

```
% next we will create loops within our scripts
```

2.9.2 Saving Scripts

To save a script to the current folder, click on Save under the Editor tab of the toolstrip. Or using the keyboard shortcut Ctrl+S.



2.9.3 Running Scripts

To run a script, you type its name at the command line. Or you can run scripts from the Editor tab by pressing the **Run** button, .

2.9.4 Script Locations

Remember, MATLAB looks for scripts and other files in certain places. To run a script, the file must be in the current folder or in a folder on the **search path**.

If you want to store and run programs in another folder, add it to the search path. Select the folder in the Current Folder browser, right-click, and then select **Add to Path**.

2.9.5 Loops and Conditional Statements

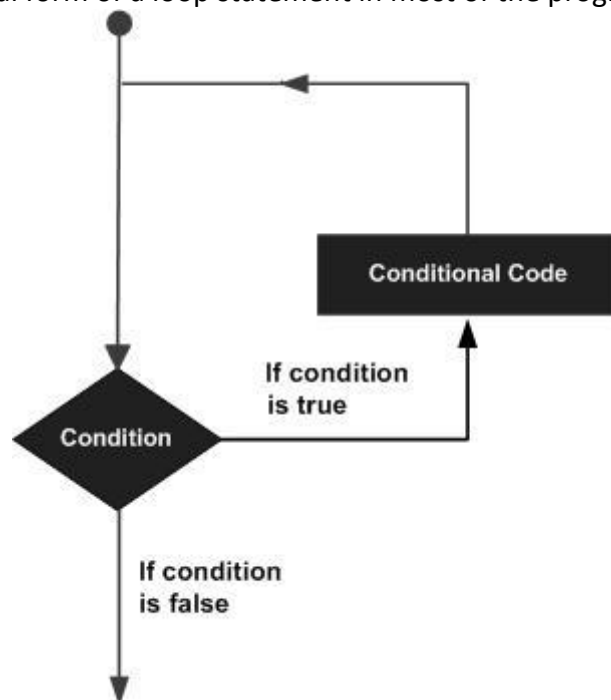
Within a script, you can loop over sections of code and conditionally execute sections using the keywords `for`, `while`, `if`, and `switch`.

2.10 Loop Types

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –



MATLAB provides following types of loops to handle looping requirements.

2.10.1 While Loop

The while loop repeatedly executes statements while condition is true.

The syntax of a `while` loop in MATLAB is –

```
while <expression>
    <statements>
end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true.

An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

While Loop Example 1

Create a script file such as whileLoop1.m and type the following code –

```
a = 10;

% while loop execution
while( a < 20 )

    fprintf('value of a: %d\n', a);

    a = a + 1;

end
```

When you run the file, it displays the following result –

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

2.10.2 For Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. The syntax of a for loop in MATLAB is:

```
for index = values
    <program statements>
    ...
end
```

values has one of the following forms

Sr.No.	Format & Description
1	<i>initval:endval</i> increments the index variable from <i>initval</i> to <i>endval</i> by 1, and repeats execution of <i>program statements</i> until <i>index</i> is greater than <i>endval</i> .

2	<i>initval:step:endval</i> increments <i>index</i> by the value <i>step</i> on each iteration, or decrements when <i>step</i> is negative.
3	<i>valArray</i> creates a column vector <i>index</i> from subsequent columns of array <i>valArray</i> on each iteration. For example, on the first iteration, <i>index</i> = <i>valArray</i> (:,1). The loop executes for a maximum of <i>n</i> times, where <i>n</i> is the number of columns of <i>valArray</i> , given by <i>numel(valArray, 1, :)</i> . The input <i>valArray</i> can be of any MATLAB data type, including a string, cell array, or struct.

For Loop Example 1

Create a script file such as `forLoop1.m` and type the following code -

```
for a = 10:20
    fprintf('value of a: %d\n', a);
end
```

When you run the file, it displays the following result -

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

For Loop Example 2

Create another script file and type the following code -

```
for a = 1.0: -0.1: 0.0
    disp(a)
end
```

When you run the file, it displays the following result -

```
1
0.9000
0.8000
0.7000
0.6000
0.5000
```



```
0.4000
0.3000
0.2000
0.1000
0
```

For Loop Example 3

Create a script file and type the following code –

```
for a = [24,18,17,23,28]
    disp(a)
end
```

When you run the file, it displays the following result –

```
24
18
17
23
28
```

For Loop Example 4

Create a script named `calcmean.m` that uses a for loop to calculate the mean of five random samples and the overall mean.

```
nsamples = 5;
npoints = 50;

for k = 1:nsamples
    currentData = rand(npoints,1);
    sampleMean(k) = mean(currentData);
end
overallMean = mean(sampleMean)
```

When you run the file, it displays the following result –

```
overallMean =
    0.4910
```

Now, modify the for loop so that you can view the results at each iteration. Display text in the Command Window that includes the current iteration number, and remove the semicolon from the assignment to `sampleMean`.

```

for k = 1:nsamples
    iterationString = ['Iteration #',int2str(k)];
    disp(iterationString)
    currentData = rand(npoints,1);
    sampleMean(k) = mean(currentData)
end
overallMean = mean(sampleMean)

```

When you run the script, it displays the intermediate results, and then calculates the overall mean.

```

>> calcmean
Iteration #1

sampleMean =

    0.4866

Iteration #2...

```

In the Editor, add conditional statements to the end of `calcmean.m` that displays a different message depending on the value of `overallMean`.

```

if overallMean < .49
    disp('Mean is less than expected')
elseif overallMean > .51
    disp('Mean is greater than expected')
else
    disp('Mean is within the expected range')
end

```

Run `calcmean` and verify that the correct message displays for the calculated `overallMean`. For example:

```

overallMean =

    0.5334

Mean is greater than expected

```

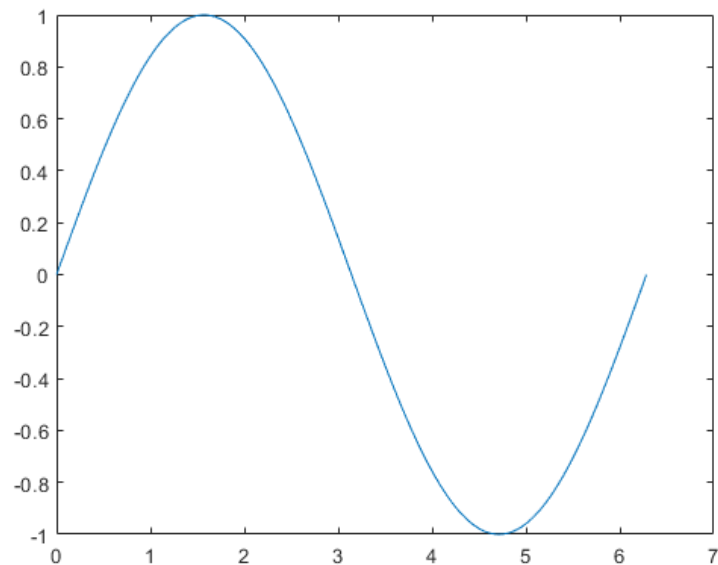
2.11 2-D and 3-D Plots

The MATLAB figure window displays plots. See [Types of MATLAB Plots](#) for a full description of the figure window. It also discusses the various interactive tools available for editing and customizing MATLAB graphics.

2.11.1 Line Plots

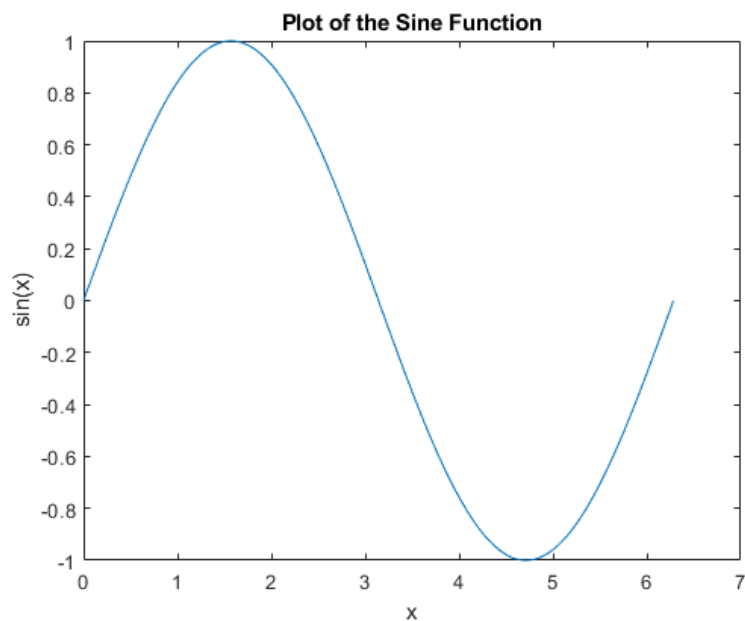
To create two-dimensional line plots, use the `plot` function. Create a new script and plot the value of the sine function from 0 to 2π :

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



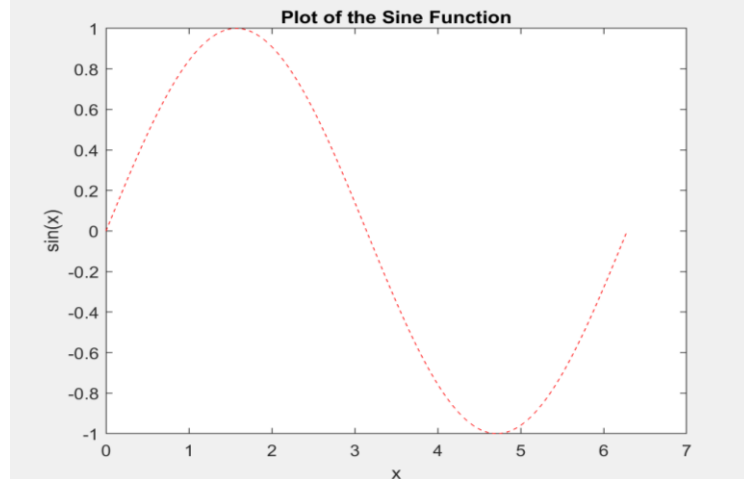
You can label the axes and add a title. Update your script by adding -

```
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. Update your plot function to -

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



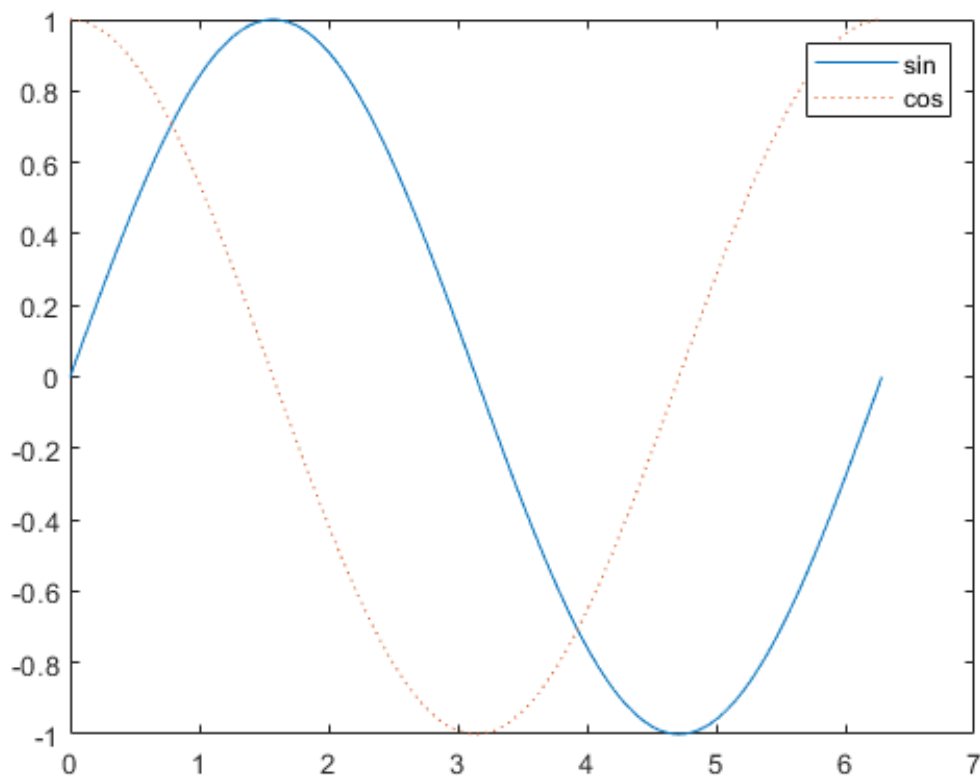
'r--' is a **line specification**. Each specification can include characters for the line color, style, and marker. A marker is a symbol that appears at each plotted data point, such as a +, o, or *. For example, 'g:*' requests a dotted green line with * markers.

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

Create a new script and type the following -

```
x = 0:pi/100:2*pi;  
y = sin(x);  
plot(x,y)  
  
hold on  
  
y2 = cos(x);  
plot(x,y2,':')  
legend('sin','cos')  
  
hold off
```



2.12 3-D Plots

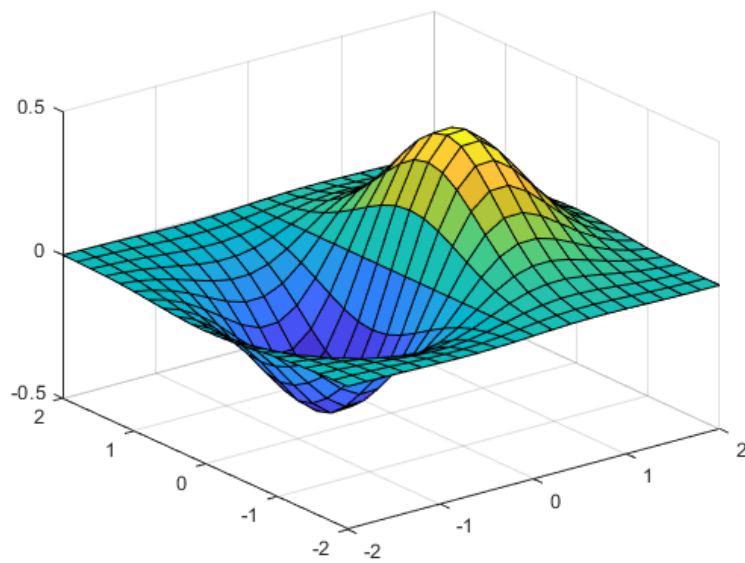
Three-dimensional plots typically display a surface defined by a function in two variables, $z = f(x,y)$.

To evaluate z , first create a set of (x,y) points over the domain of the function using `meshgrid`.

```
[X,Y] = meshgrid(-2:.2:2);  
Z = X .* exp(-X.^2 - Y.^2);
```

Then, create a surface plot.

```
surf(X,Y,Z)
```



Both the `surf` function and its companion `mesh` display surfaces in three dimensions. `surf` displays both the connecting lines and the faces of the surface in color. `mesh` produces wireframe surfaces that color only the lines connecting the defining points.

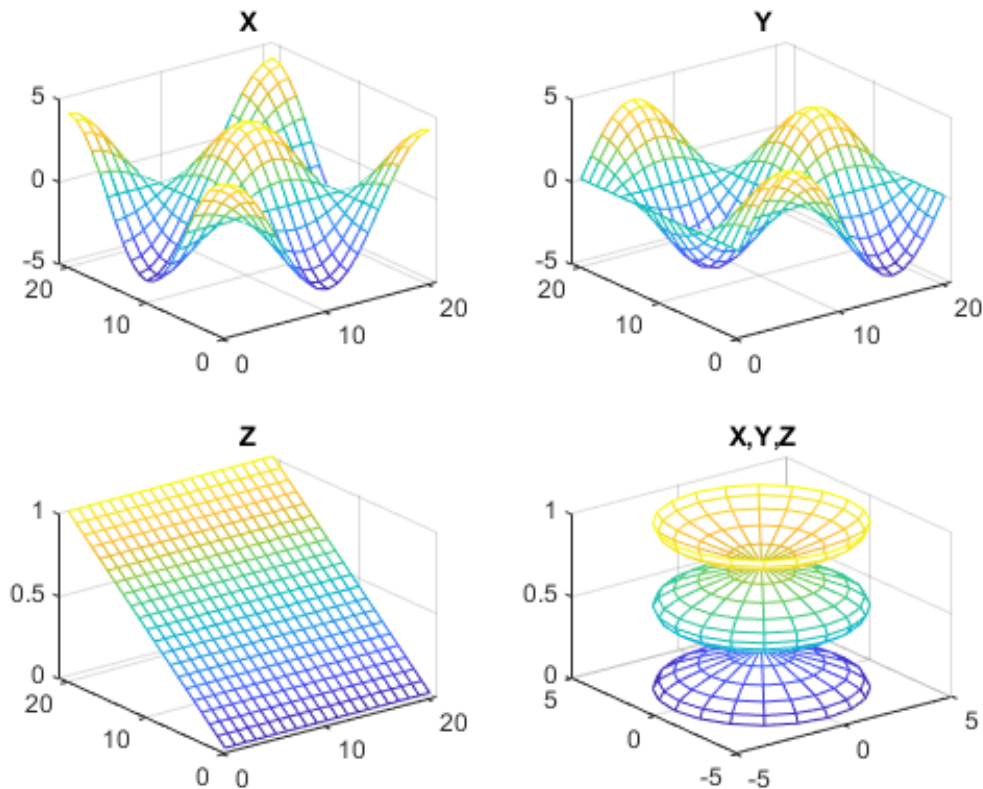
2.13 Subplots

You can display multiple plots in different subregions of the same window using the `subplot` function.

The first two inputs to `subplot` indicate the number of plots in each row and column. The third input specifies which plot is active.

Create a new script and create four plots in a 2-by-2 grid within a figure window.

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X); title('X');
subplot(2,2,2); mesh(Y); title('Y');
subplot(2,2,3); mesh(Z); title('Z');
subplot(2,2,4); mesh(X,Y,Z); title('X,Y,Z');
```



2.14 Plotting Imported Data

After you import data into the MATLAB® workspace, it is a good idea to plot the data so that you can explore its features. An exploratory plot of your data enables you to identify discontinuities and potential outliers, as well as the regions of interest.

This example uses sample data in *count.csv*, a comma-delimited text file. The file consists of three sets of hourly traffic counts, recorded at three different town intersections over a 24-hour period. Each data column in the file represents data for one intersection.

2.14.1 Load Data from CSV File

Import data into the workspace using the `load` function.

```
>> load count.csv
```

Loading this data creates a 24-by-3 matrix called `count` in the MATLAB workspace.

Next, get the size of the data matrix.

```
>> [n,p] = size(count)
```

```
n =
    24
```

```
p =
     3
```

`n` represents the number of rows, and `p` represents the number of columns.

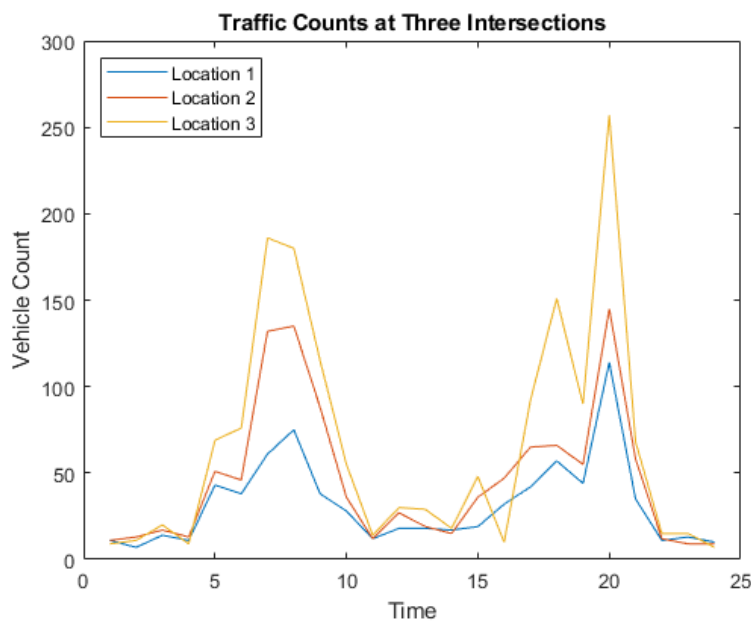
2.14.2 Plot the count.dat Data

Create a time vector, t , containing integers from 1 to n .

```
>> t = 1:n;
```

Plot the data as a function of time and annotate the plot.

```
>> plot(t,count),  
>> legend('Location 1','Location 2','Location  
3','Location','NorthWest')  
>>xlabel('Time'), ylabel('Vehicle Count')  
>> title('Traffic Counts at Three Intersections')
```



2.14.3 Functions for Calculating Descriptive Statistics

Use the following MATLAB® functions to calculate the descriptive statistics for your data.

Note: For matrix data, descriptive statistics for each column are calculated independently.

Statistics Function Summary

Function	Description
max	Maximum value
mean	Average or mean value
median	Median value
min	Smallest value
mode	Most frequent value
std	Standard deviation
var	Variance, which measures the spread or dispersion of the values

The following examples apply MATLAB functions to calculate descriptive statistics:

This example shows how to use MATLAB functions to calculate the maximum, mean, and standard deviation values for a 24-by-3 matrix called `count`. MATLAB computes these statistics independently for each column in the matrix.

Find the maximum, mean and standard deviation of each column of `count` -

```
% Find the maximum value in each column
>> mx = max(count)

% Calculate the mean of each column
>> mu = mean(count)

% Calculate the standard deviation of each column
>> sigma = std(count)
```

The results are -

```
mx =
    114    145    257

mu =
    32.0000    46.5417    65.5833

sigma =
    25.3703    41.4057    68.0281
```

To get the row numbers where the maximum data values occur in each data column, specify a second output parameter `indx` to return the row index. For example, try:

```
>> [mx,indx] = max(count)
```

The results are -

```
mx =
    114    145    257

indx =
    20    20    20
```

Here, the variable `mx` is a row vector that contains the maximum value in each of the three data columns. The variable `indx` contains the row indices in each column that correspond to the maximum values.

To find the minimum value in the entire `count` matrix, 24-by-3 matrix into a 72-by-1 column vector by using the syntax `count(:)`. Then, to find the minimum value in the single column, use the following syntax:

```
>> min(count(:))

ans =
```

Next, subtract the mean from each column of the matrix by using the following syntax:

```
% Get the size of the count matrix - as previously done
>> [n,p] = size(count)

% Compute the mean of each column - as previously done
>> mu = mean(count)

% Create a matrix of mean values by
% replicating the mu vector for n rows
>> MeanMat = repmat(mu,n,1)

% Subtract the column mean from each element
% in that column
>> x = count - MeanMat

% completed!
```

2.15: Next Tasks:

Try loading in some of the other sample [Datasets from MATLAB](#). Experiment with plotting these Datasets and calculating some descriptive statistics.

Experiment with the various [types of Plots](#) in MATLAB.

Visit [Mathworks website](#) and create an account using your Ulster University Email address.