Functions

By a **functions** we shall mean self-designed MATLAB script m-files which contains a few lines of codes meant for executing a certain process. For instance, writing a function file for solving a system of linear equations we have looked at briefly in Unit 2. Throughout our discussion in this lesson, when we are to consider writing algorithms can be classified into two types: direct and iterative methods. With the former, we obtain the solution in a finite number of steps, assuming no rounding errors, for example to solve a linear system with Gaussian elimination. Iterative methods involve the process involves generating a sequence of approximation that converge to the solution as the number of steps approaches infinity.

1. Writing function files

A basic function file is structured in such a way that it has an output array, function file and input array defined as

```
function [output]=function_name(input)
```

where input contains variable values defined by the user as to be used in the code while function name is the title of the file name for the script. However, it is not always the case that we specify the output. For example we can have a structure either as

```
function [output]=function_name(input)
        Code
end
```

or

```
function function_name(input)
        Code
       (output)
end
```

Sometimes inputs are specified either inside the code or at the command window. As such, the structure of the MATLAB function file changes and appears as

```
function function_name
        (input)
         Main code
        (output)
end
```

After the function declaration, say function [output]=function_name(input), in the m.file, it is possible to write a description of the function. This is done with the comment sign "%" before each line. Then, from the command window, we can read this information about program's description when we type "help<function name>".

MATLAB will automatically generate a display when commands are executed. However, in addition to this automatic display, MATLAB has several commands that can be used to generate displays or outputs. Two commands that are frequently used to generate output are: ***disp*** and ***fprintf***.

Example

In order to compute the (N+1)-term Taylor series approximation to e^x where $e^x \approx \sum_{k=0}^{N} \frac{x^N}{k!}$, we write a Matlab script file based on the following pseudo code;

Pseudo code

Input: $x, N > 0$

Output: $(N + 1)$-term Taylor series approximation to e^x

```
taylor=1; factorial=1; xpowk=1;
for k=1 to N, do
    factorial = factorial * 1;
    xpowk=xpowk*x;
    taylor=taylor + (xpowk/factorial);
end
```

Thus, in Matlab, we implement the following code;

```
function [taylor]=Taylorestimate(x,N)

taylor=1;
factorial=1;
xpowk=1;

for k=1:N
    factorial=factorial*k;
    xpowk=xpowk*x;
    taylor=taylor+(xpowk/factorial);
end
disp('The (N+1)-term Taylor series approx to
e^x is:')
end
```

With such a function file, to run, we type the main file line with inputs specified to get the results as

```
>> [taylor]=Taylorestimate(2,10)
The (N+1)-term Taylor series approx to e^x is:
taylor =

  7.3890
```

Example

The following is a MATLAB function file that evaluates $n!$, "n factorial" for any choice of an integer n

```
function f = factorial(n)
% FACTORIAL(N) returns the factorial of N.
% Compute a factorial value.
f = prod(1:n);
```

Such that for $n = 5$, we get on the command window

```
>> f = factorial(5)
f =
  120
```

2. Mathematical functions

A number of functions in MATLAB do not need to de defined or derived, however. For example, MATLAB already offers many predefined mathematical functions *"called **built-ins**"* for technical computing which contains a large set of mathematical functions. These functions can be accessed if we either type ***help elfun*** or ***help specfun*** in the command window to call up full lists of elementary or special functions, respectively.

Table 8 lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

Table 8: Elementary MATLAB built-in mathematical functions

cos(x)	cosine	abs(x)	absolute value
sin(x)	sine	sign(x)	signum function
tan(x)	tangent	max(x)	maximum value
acos(x)	arch cosine	min(x)	minimum value
asin(x)	arc sine	ceil(x)	round towards $+\infty$
atan(x)	arc tangent	floor(x)	round towards $-\infty$
sqrt(x)	square root	round(x)	round to nearest integer
log(x)	natural logarithm	rem(x)	remainder after division
log10(x)	standard logarithm	angle(x)	phase angle
		conj(g)	complex conjugate

Example

To evaluate the expression $y = e^{-a}\sin(x) + 10\sqrt{y}$, for $a = 5, x = 2$ and $y = 8$, we type the following statements in MATLAB;

```
>> a = 5;
>> x = 2;
>> y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y =
  28.2904
```

Example

To evaluate the expression $V = \ln x + e^{10}\sin(\pi/4) - \log_{10} x$.

```
>> x=7;
>> V=log(x)+exp(10)*sin(pi/4)-log10(x)
V =
  1.5576e+04
```

3. Plot functions

The very basic plot function imbedded in MATLAB displays a graph of one's interest in 2-dimensional. For example, the function plot can be used to produce a graph from two vectors x and y. Recall that a vector is an array of elements. This means by usage of vectors, we have two related arrays of elements whose plot can produce a graph.

During plotting, MATLAB is designed in such a way to plot one vector vs. another. The first one is treated as the abscissa (or x) vector and the second as the ordinate (or y) vector. These vectors have to be of the same length.

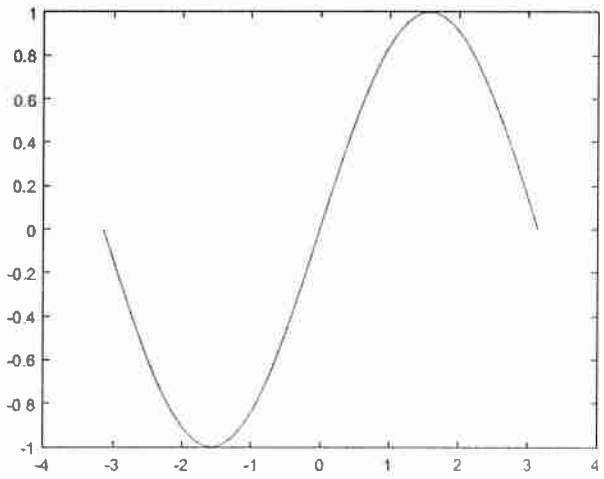
Besides, MATLAB can as well plot a vector vs. its own index. The index is therefore treated as the abscissa vector. Given a vector "***time***" and a vector "***dist***" we could say:

```
>> plot (time, dist)      % plotting versus time
>> plot (dist)            % plotting versus index
```

For instance, the following code

```
>> x=-pi:pi/100:pi;
>> y=sin(x);
>> plot(x,y)
```

produces a sine function graph whose x values range between -4 and 4, while y values range between -1 and 1.



Other than going just by mere plotting function, we also have MATLAB commands to annotate the basic plot, by either adding axis labels, titles, and legends. These are defined by

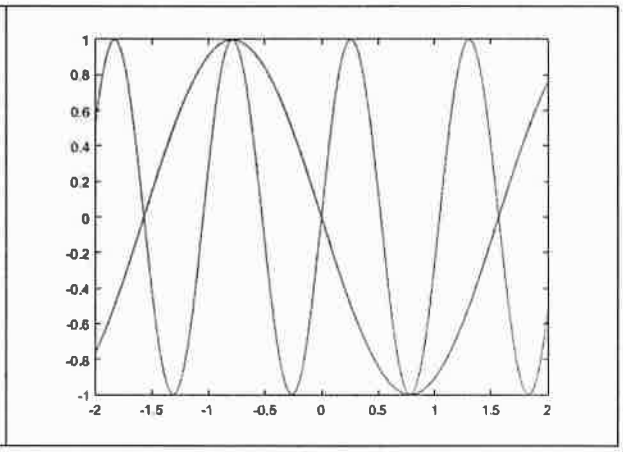
```
>> xlabel ('X-axis label')  % puts a label on x-axis
>> ylabel ('Y-axis label')  % puts a label on y-axis
>> title ('Title of my plot')  % puts a title on the plot
>> legend('Key features of my plot')  % highlights key features of the plot
```

For example, try running the following statements on the MATLAB command window and observe the differences in the outputs

```
>> x=-pi:0.1:pi; y=sin(x);
>> plot(x,y)
>> plot(x,y,'s-')
>> xlabel('x'); ylabel('y=sin(x)');
```

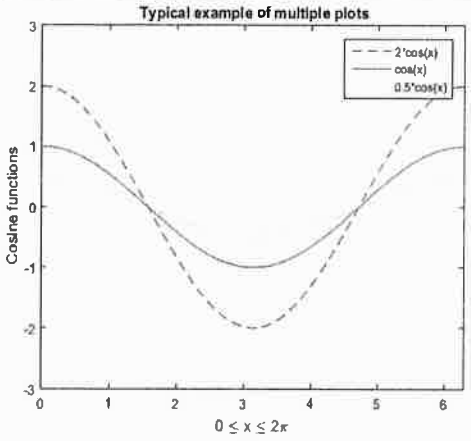
As another example, with a plot function file called aliasing, we can get

```
function aliasing

x= -2:0.01:2;
f_x=sin(-2*x);
g_x=sin(6*x);
% g_x=sin(-10*x);
plot(x,f_x);
hold on
plot(x,g_x);
hold off

end
```



We can also add colors to the plotted lines for easy of distinguishing one line from the other in a multiple plot. For example

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')
>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```



The color of a single curve is, by default, blue, but other colors are possible. In a plot function, the desired color is indicated by a third argument. For example, red is selected by ***plot(x,y,'r')***. Note the single quotes, ' ', around r. By default, MATLAB uses line style and color to distinguish the data sets plotted in the graph. However, we can change the appearance of these graphic components or add annotations to the graph to help explain our data for presentation. Some of the annotations are as presented in Table 9.

Table 9: Attributes for plot

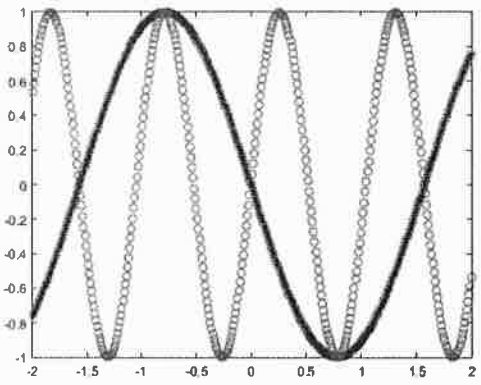
Symbol	Color	Symbol	Line style	Symbol	Marker
k	Black	–	Solid	+	Plus sign
r	Red	– –	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			s	Square
y	Yellow			d	Diamond

For example, it is possible to specify ***line styles***, ***colors***, and ***markers*** (e.g., circles, plus signs,...) using the plot command:

```
plot(x,y,'style_color_marker')
```

where style_color_marker is a triplet of values from Table 9. For example by specifying asterisk and circle style markers to our "aliasing" function, the plot looks change to

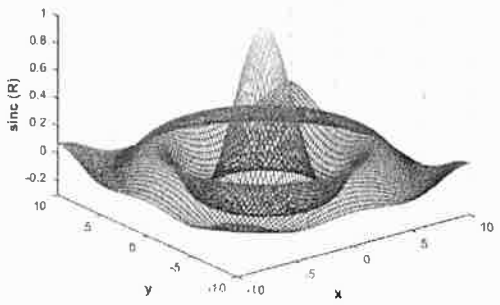
```
function aliasing
x= -2:0.01:2;
f_x=sin(-2*x);
g_x=sin(6*x);
% g_x=sin(-10*x);
plot(x,f_x,'*');
hold on
plot(x,g_x,'o');
hold off
end
```



Three-dimensional graphics can be produced using the imbedded functions surf, plot3 or mesh. For example, upon writing and executing the following meshgrid plot function in the command window

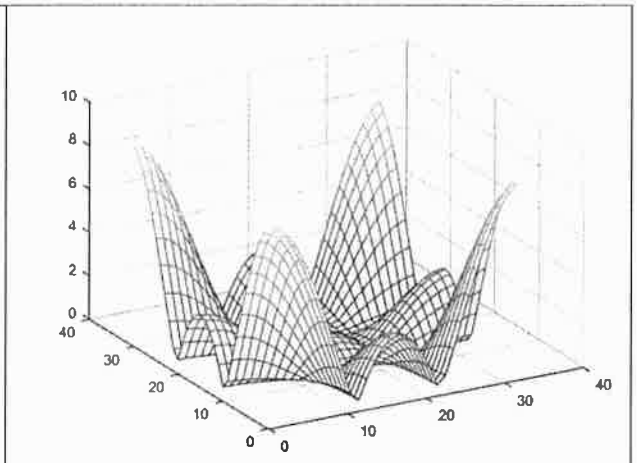
```
>> [X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);
>> f = sinc(sqrt((X/pi).^2+(Y/pi).^2));
>> mesh(X,Y,f);
>> axis([-10 10 -10 10 -0.3 1])
>> xlabel('{\bfx}')
>> ylabel('{\bfy}')
>> zlabel('{\bfsinc} ({\bfR})')
>> hidden off
```

we get an output



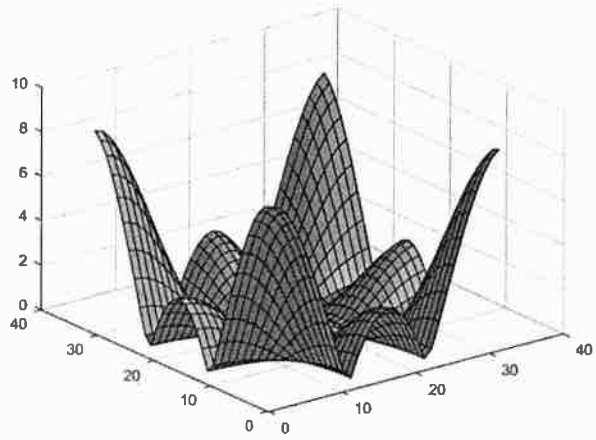
As another example, with *mesh* plot function we can get

```
>> A = zeros(32);
>> A(14:16,14:16) = ones(3);
>> F=abs(fft2(A));
>> mesh(F)
>> rotate3d on
```



Comparably, with *surf* plot function we can get

```
>> A = zeros(32);
>> A(14:16,14:16) = ones(3);
>> F=abs(fft2(A));
>> surfl(F)
>> rotate3d on
```



Activity 3 b

a) Plot the graph of the function $f(x) = \cos x + \sin \pi x$ for $x \in [-2\pi, 2\pi]$ with 0.1 sub-interval.

b) Write a function that shows an approximation for a factorial can be found using Stirling's formula:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

c) Write a MATLAB function which should be able to find the volume and surface area of a cylinder when it's closed at both ends.

d) Write a script file that calls a function to prompt the user and read in values for the hypotenuse and the angle (in radians), and then calls a function to calculate and return the lengths of sides a and b, and a function to print out all values in a nice sentence format.

e) Write a ***for loop*** function to estimate $\sqrt{2}$ when $t = [1, \cdots, 7]$ with reference to the recurrence equation

$$y_t = \frac{1}{2}(y_{t-1} + \frac{2}{y_{t-1}})$$

for $t = 1,2,3, \cdots, \quad y_0 = 3.$

Summary/Let Us Sum Up

To sum up, this unit was meant to introduce the student to the notion of counting, which is a key principle and very fundamental to most real life application problems. With principles of counting techniques, the SUM RULE is noted to be useful for counting events that are made of collection of independent events. On the other hand, the COMBINATORIAL RULE gives us a chance to look at problems of making choices in life, which if we want to obtain such related unique choices,