Workbook

APPLIED NUMERICAL METHODS
(MT – 471)

Name _____

Roll No _____

Batch _____

Discipline _____

Semester\Terms _____

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Workbook

APPLIED NUMERICAL METHODS
(MT – 471)

Final Year

Prepared by

Noor Fatima Siddiqui

Approved by

Chairman
Department of Computer Science & Information Technology

Department of Computer Science & Information Technology
NED University of Engineering & Technology

APPLIED NUMERICAL METHODS
(MT – 471)

I N D E X

LAB # 1

+ - / * ^	Basic algebraic operations
Clear, clc, cla	Clear variables from the workspace
%	Beginning of a comment line
format	
diary	Start saving commands in a file named diary
.* ./ .^	Element by element algebraic operators
linspace (x_0, x_1, N)	Create an array of N equally spaced values between x_0 and x_1
x :Δx : x_1	Create an array of values between x_0 and x_1 spaced by Δx.
inv( )	Compute Inverse of a square matrix
det()	Compute determinant of a square matrix
'	Transpose of a Matrix
Rank()	Compute rank of Matrix
fopen	Command for opening a File
help	Asking help from Matlab
lookfor	Returns all the commands related to that keyword

MATLAB as a Calculation Engine

The basic arithmetic operators are + - * / ^ and these are used in conjunction with brackets: ( ). The symbol ^ is used to get exponents (powers): 2^4=16.

Type the commands shown following the MATLAB prompt:

```
>> 2 + 3/4*5
ans =
5.7500
```

Is this calculation 2 + 3/(4*5) or 2 + (3/4)*5?

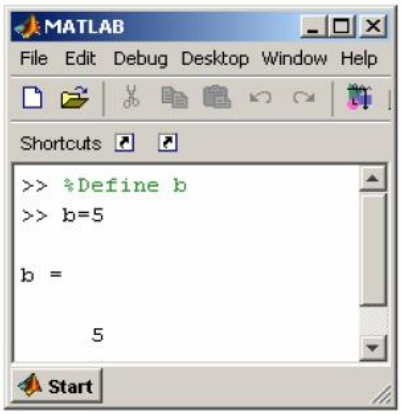
Matlab works according to the priorities:

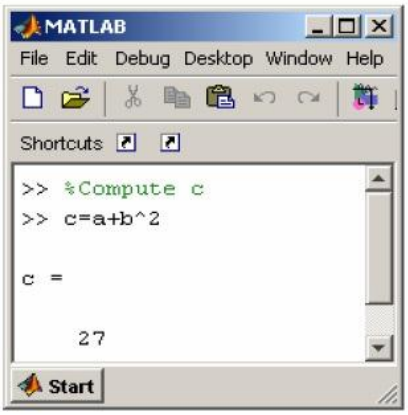
1. quantities in brackets,
2. powers 2 + 3^2 = 2 + 9 = 11,
3. * /, working left to right (3*4/5=12/5),
4. + -, working left to right (3+4-5=7-5),

Thus, the earlier calculation was for **2 + (3/4)*5** by priority 3 mentioned above.

MATLAB can be used as a calculator and can store the inputs or the outputs into variables. If a semicolon ";" is placed at the end of a statement, the value of the variable is calculated but not displayed on the screen. To clear variables from the workspace, use the command *clear*.

P





- To display the current value of a variable, double-click on its name in command window.
- To avoid retyping previously used command, use up and down arrows.

Numbers & Formats

Matlab recognizes several different kinds of numbers

Type	Examples
Integer	$1362, -217897$
Real	$1.234, -10.76$
Complex	$3.21 - 4.3i \quad (i = \sqrt{-1})$
Inf	Infinity (result of dividing by 0)
NaN	**N**ot **a** **N**umber, $0/0$

The "**e**" notation is used for very large or very small numbers:

`-1.3412e+03` $= -1.3412 \times 10^{3} = -1341.2$

`-1.3412e-01` $= -1.3412 \times 10^{-1} = -0.13412$

All computations in MATLAB are done in double precision, which means about 15 significant figures.

Command	Example of Output
`>>format short`	`31.4162`(4–decimal places)
`>>format short e`	`3.1416e+01`
`>>format long e`	`3.141592653589793e+01`
`>>format short`	`31.4162`(4–decimal places)
`>>format bank`	`31.42`(2–decimal places)

The format (how Matlab prints numbers) is controlled by the **format** command. Type **help format** for full list. Should you wish to switch back to the default format then **format** will suffice. The command **format compact** is also useful in that it suppresses blank lines in the output thus allowing more information to be displayed.

MATLAB **Variables Names**

Legal variable names:

- Begin with one of a-z or A-Z
- Have remaining characters chosen from a-z, A-Z, 0-9, or
- Have a maximum length of 31 characters
- Should not be the name of a built-in variable, built-in function, or user-defined function

Examples:

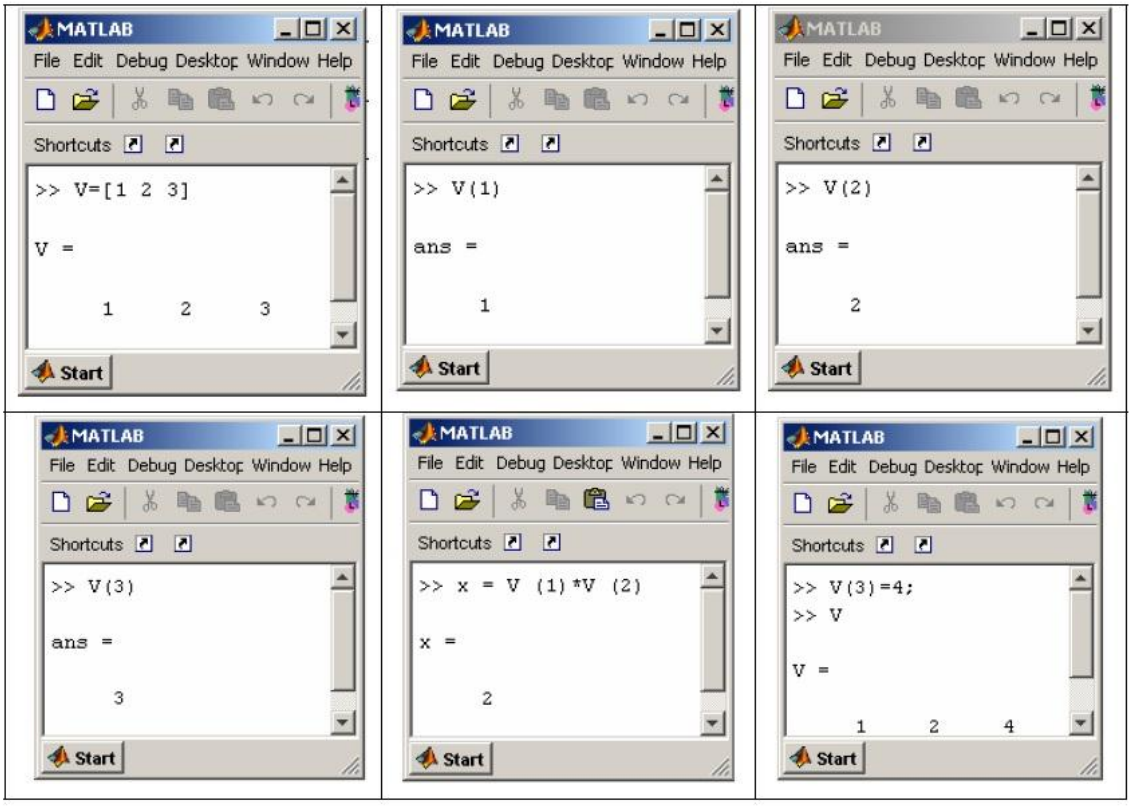
xxxxxxxxx
pipeRadius
widgets_per_baubble
mySum
mysum

Note: mySum and mysum are *different* variables. MATLAB is *case sensitive.*

LAB # 2

Vector Operations

Create the vector $V = [1 \quad 2 \quad 3]$. Display each of the elements. Multiply the first and second elements. Change the third element to 4 and display the new vector on the screen.



Element by Element Algebraic Operations

.* ./ .^	Element by element algebraic operations

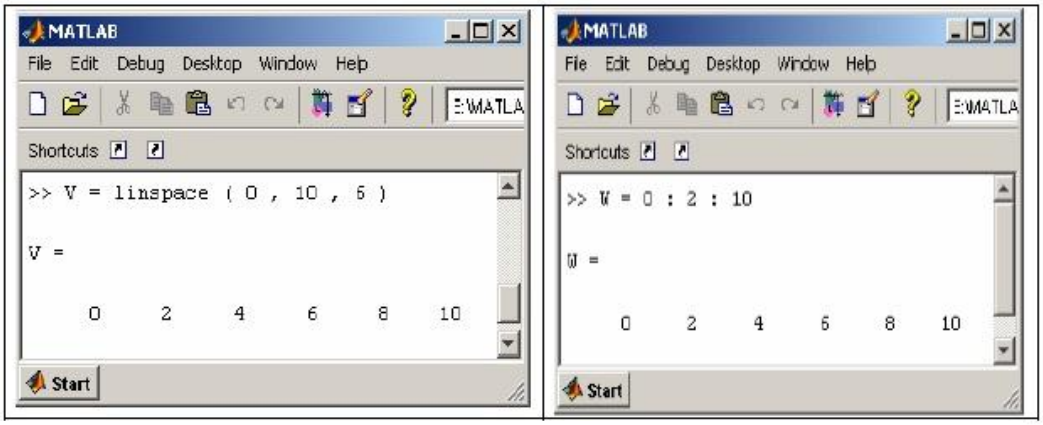
- Define two vectors $x = [1 \quad -2 \quad 5 \quad 3]$ and $y = [4 \quad 1 \quad 0 \quad -1]$.
- Multiply x and y-- element by element.
- Compute the squares of the elements of x.

```
Shortcuts
>> % Define x and y
>> x=[1 -2 5 3];
>> y=[4 1 0 -1];
>> % Compute z
>> z = x . * y
??? z = x . * y
            |
Error: Missing variable or function.

>> z = x .* y

z =

     4    -2     0    -3

>> % Compute the squares of the elements of x
>> x .^2
??? Error: "x" was previously used as a variable,
conflicting with its use here as the name of a function.

>> x.^2

ans =

     1     4    25     9
```

- A period is needed for element-by-element operations with arrays in front of ***, / , and ^** operators.
- A period is NOT needed in front of + and - .

Creating Arrays with Equally Spaced Elements

linspace (x_0, x_1, N)	Create an array of N equally spaced values between x_0 and x_1
$x : \Delta x : x_1$	Create an array of values between x_0 and x_1 spaced by Δx.

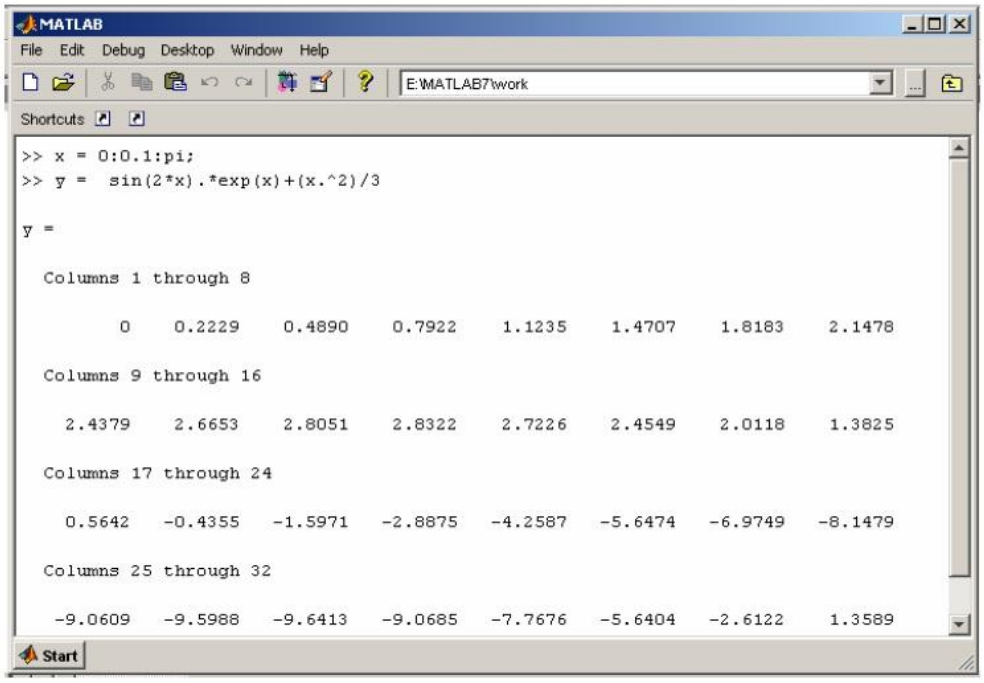


Evaluating Functions using Vector Operations

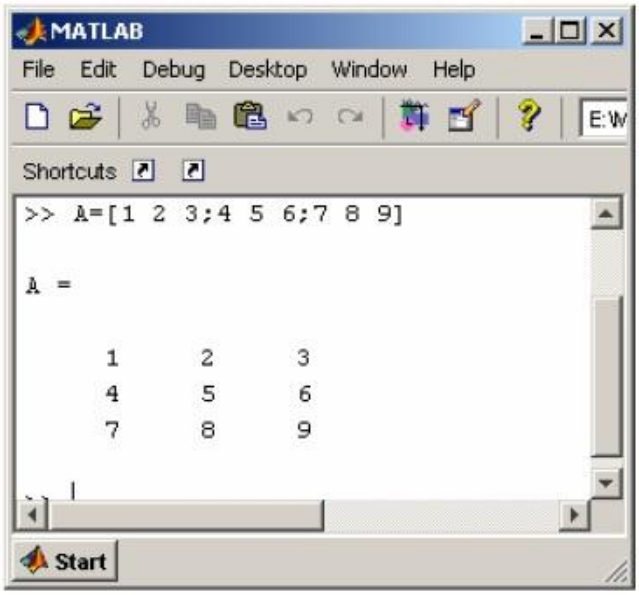
Evaluate the function y given below between $x = 0$ and $x = \pi$:

$$y = e^x \sin 2x + \frac{x^2}{3}$$

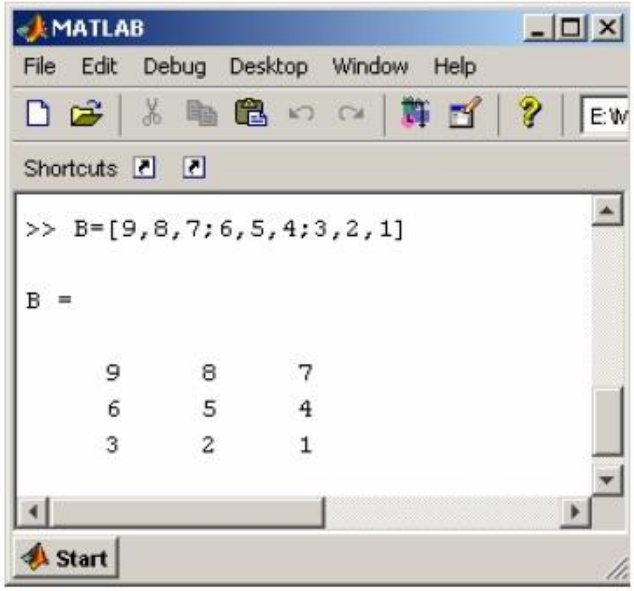
To do so, first create a vector x containing equally spaced elements in the domain of the function. Then compute the corresponding vector y. Each element of y will be equal to the value of the function evaluated at the corresponding elements of x.



Entering matrices into Matlab

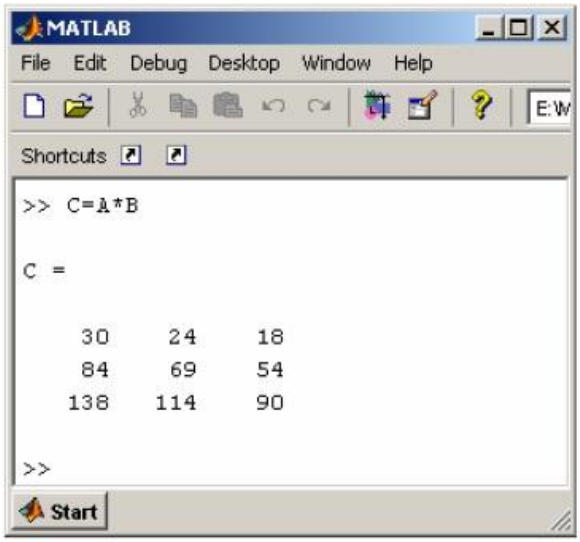


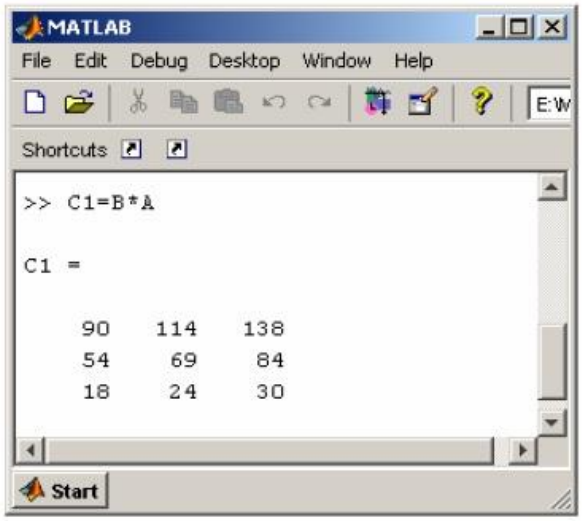
The elements in a row can also be separated by commas (,) rather than spaces



Matrix Multiplication

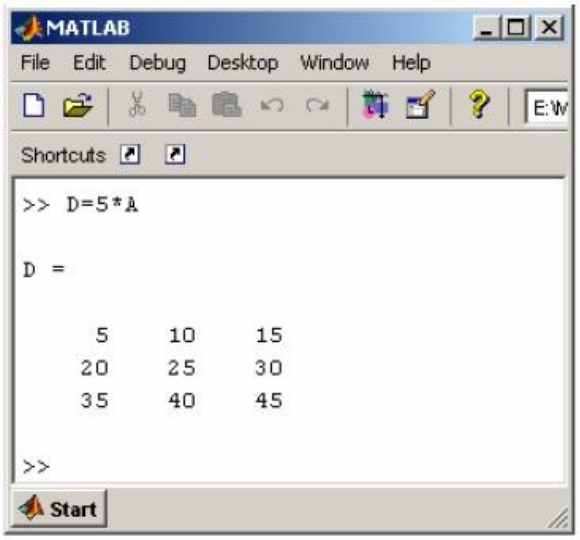
Matrices are multiplied by using (*) operator.



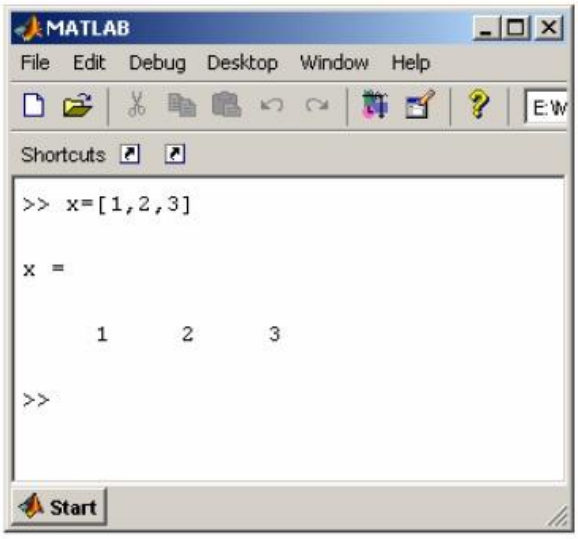


✍ A*B is not necessarily equal to B*A because matrix multiplication is not commutative.

Multiplying Matrix by a Scalar.

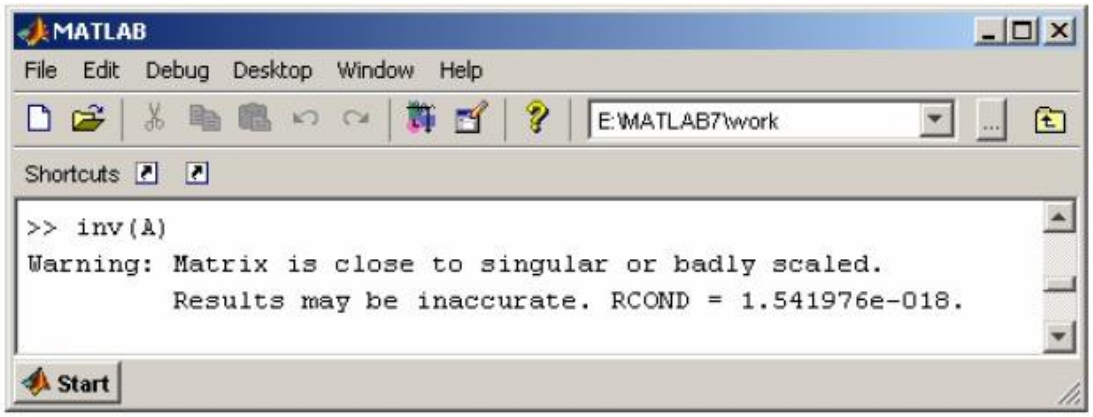


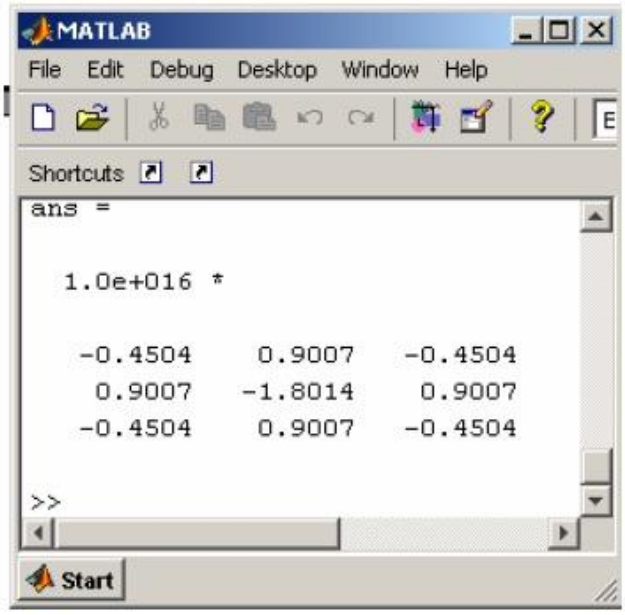
Creating Row Vector



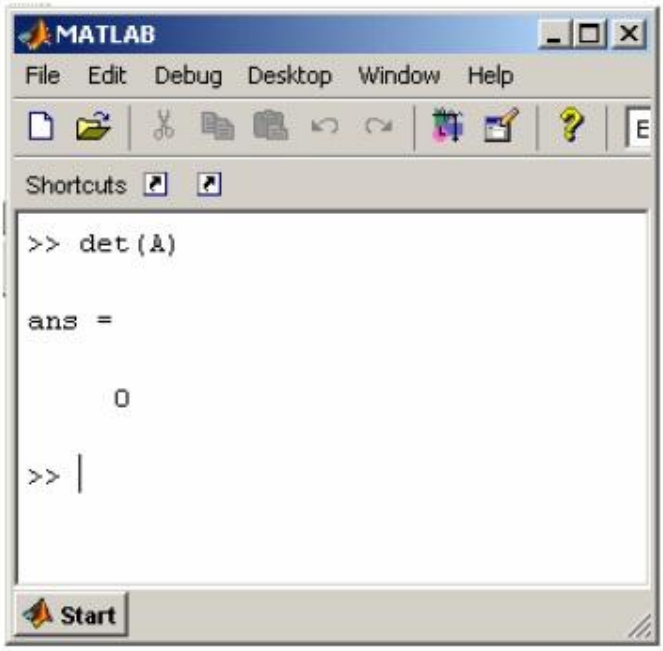
Inverse of a Matrix

Inverse of a square matrix can be obtained using inv( ) function.

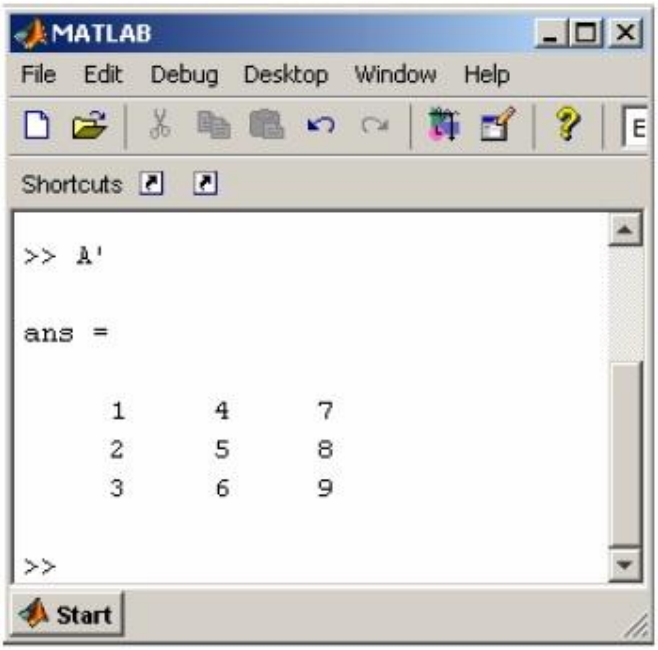




Determinant of a Matrix



Transpose of a Matrix



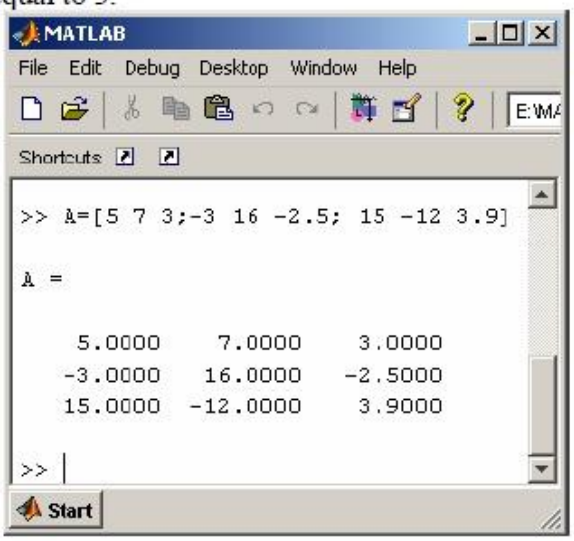
Solving System of Linear Equation

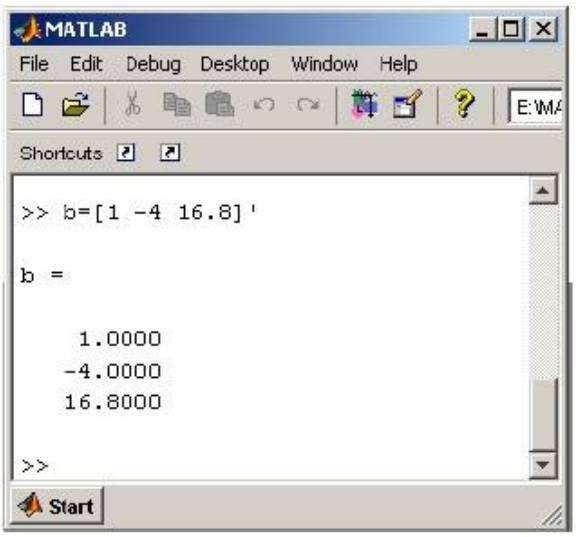
5x +7y +3z = 1---------- (01)

-3x + 16y -2.5z = - 4 ---------- (02)

15x – 12y +3.9z = 16.8 ---------- (03)

The equation **AX=b** has a solution if the **rank of A** and rank of the augmented matrix[A b]are both equal to 3.





LAB # 3

plot	Create a scatter/line plot in 2-D
title	Add a title to the current figure
xlabel, ylabel	Add *x* and *y* labels to the current figure
legend	Add a legend to the current figure
hold on / hold off	Plot multiple curves in the same figure
subplot	Make several plots in the same window
Grid on/off	
axis([XMIN XMAX YMIN YMAX])	sets scaling for the x- and y-axes on the current plot.
zoom	
axis square, normal, tight, equal	Axis control commands of a plot.
semilogx	Compute determinant of a square matrix
semilogy	Transpose of a Matrix
loglog	Compute rank of Matrix
Polar	Command for opening a File
Help	Asking help from Matlab
comet	Returns all the commands related to that

Plotting in X & Y(2D Plots)

plot **title** **xlabel, ylabel**	Create a scatter/line plot in 2-D Add a title to the current figure Add *x* and *y* labels to the current figure
legend	Add a legend to the current figure
hold on, hold off **Subplot(m,n,p)**	Plot multiple curves in the same figure Breaks the figure window into an m-by-n matrix of small windows, selects the p^{th} window for the current plot
Grid on/off	

The default is to plot solid lines. A solid white line is produced by

» plot(x, y, 'w-')

The third argument is a string whose 1st character specifies the color (optional) and the second the line style. The options for colors and styles are:

Line Styles & Colors in Matlab Plots

Color		Symbol		Line	
y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

The displacement of an object falling under the force of gravity is given by:

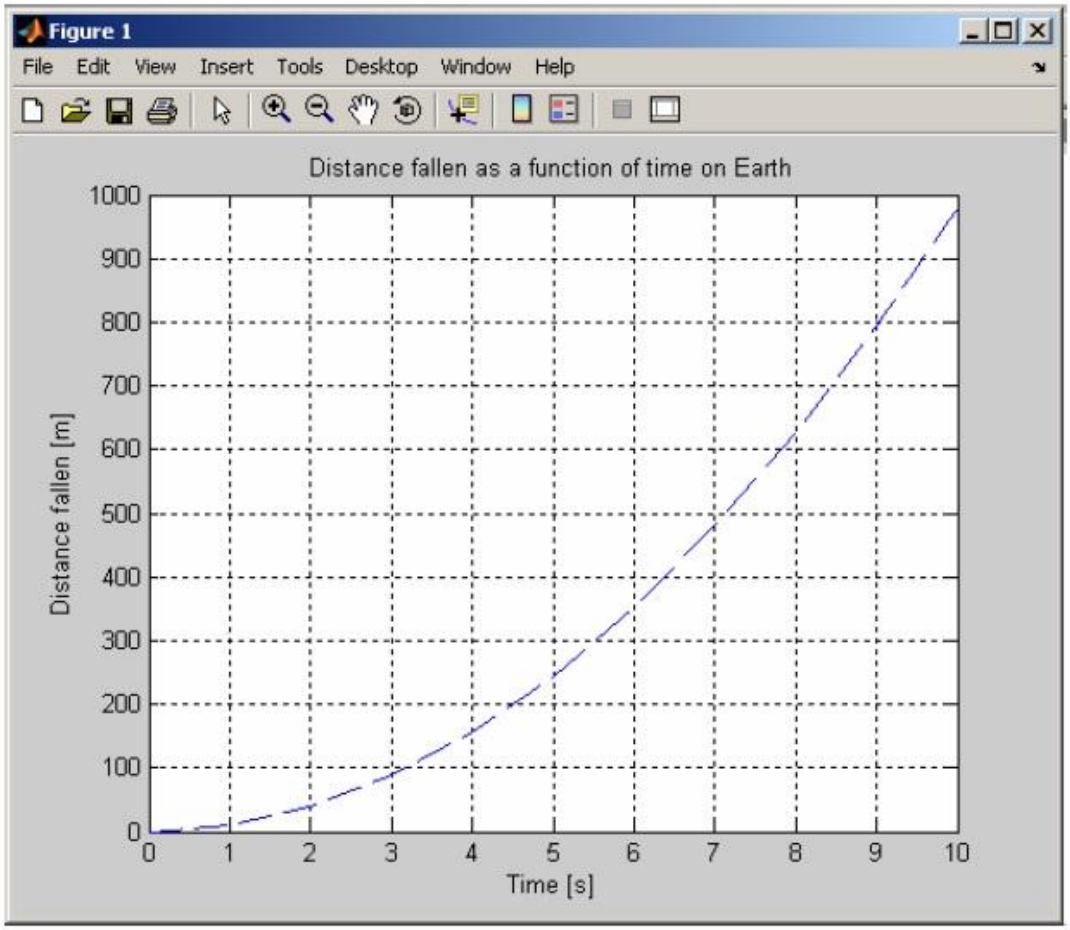
$$y = g\,t^2$$

where g is the acceleration of gravity [m/s^2], y is the distance traveled [m], and t is time [s].The following table gives the values of g for three different planets:

Planet	g
Earth	9.80 m/s^2
Jupiter	24.79 m/s^2
Mercury	3.72 m/s^2

i) Plot the distance fallen as a function of time near the surface of the Earth between 0 and 10 seconds, with an increment of 1 second. Add a title and x and y labels.

```
% Define time values
t = 0:1:10;

% Define acceleration due to  gravity
g = 9.80;

% Compute displacement y
y = g * t.^2;

% Plot
plot(t,y,'b--')

% Create labels
grid on
xlabel('Time [s]')
ylabel('Distance fallen [m]')
title('Distance fallen as a function of time on Earth')
```



✍ You can define the color and the style of a curve when using the *plot* command. In the example below, 'b- -' stands for blue dashed line. Other possible options are ":" for a dotted line, "-." for a dash-dotted line, "-" for a solid line. The colors are designated by the first letter of the desired color (b,r,g,y… k = black). If the style or the color is not specified, MATLAB uses its default settings. See *help plot* for additional information.

✍ You can zoom in and out, draw arrows, add titles, labels, legends, or write text directly in the figure using the menu bar.

LAB # 4

Plotting Multiple Functions on Same Figure in one Plot

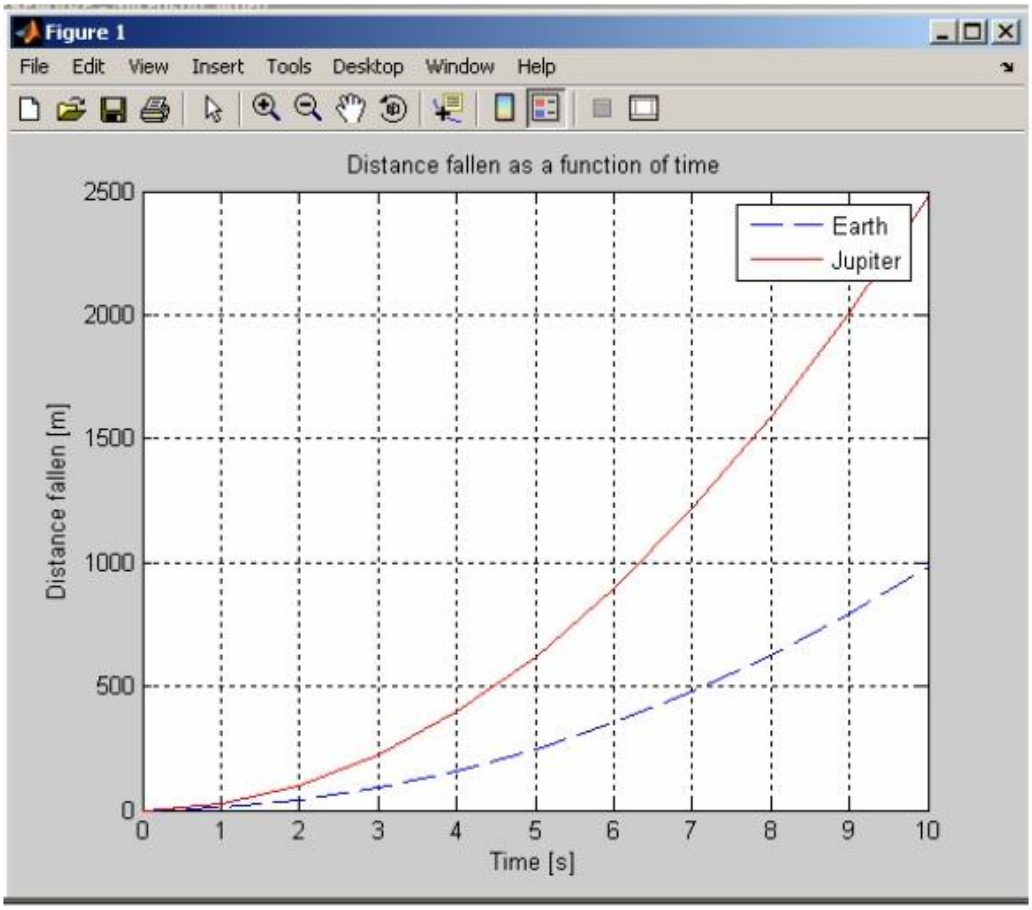
A call to plot clears the graphics window before plotting the current graph. This is not convenient if we wish to add further graphics to the figure at some later stage. To stop the window being cleared:

```
>> plot(x,y,'w-'), hold on
>> plot(x,z,'gx'), hold off
```

ii) Plot the displacement curve for both the Earth and Jupiter in the same figure and add a legend.

[Code continued from previous one]

```
% Both functions are to be plotted in the same figure
hold on

% Compute displacement for Jupiter
g = 24.79;
y = g * t.^2;

% Plot displacement for Jupiter
    plot(t,y,'r-')

% Create labels
  title('Distance fallen as a function of time')
  xlabel('Time [s]')
  ylabel('Distance fallen [m]')

% Add a legend
  legend('Earth','Jupiter')

% End of the hold command
  hold off
```



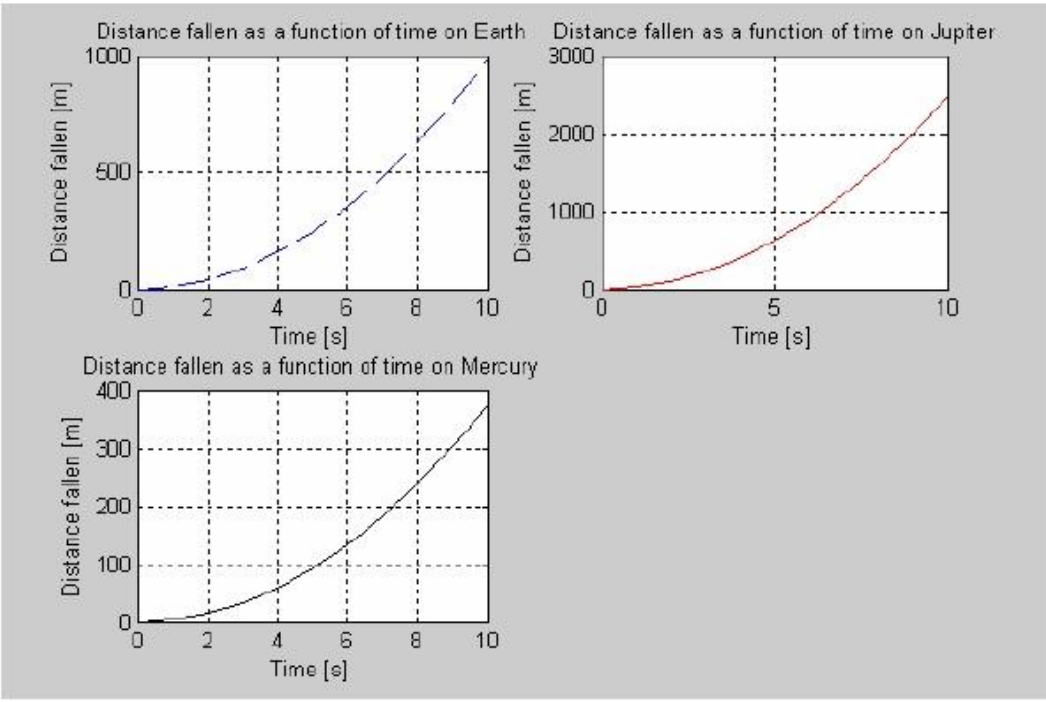
- Multiple functions can be plotted on the same figure using the *hold on* and *hold off* commands.
- Several graphs may be drawn on the same figure as in **plot(x,y,'w-',x,cos(3*pi*x),'g--')**
- A descriptive legend may be included with **legend('Sin curve','Cos curve')**
- The legend command will give a list of line-styles, as they appeared in the plot command, followed by a brief description.
- Matlab fits the legend in a suitable position, so as not to conceal the graphs whenever possible. The legend may be moved manually by dragging it with the mouse.

Plotting Multiple Functions on Same Figure in Separate Plots

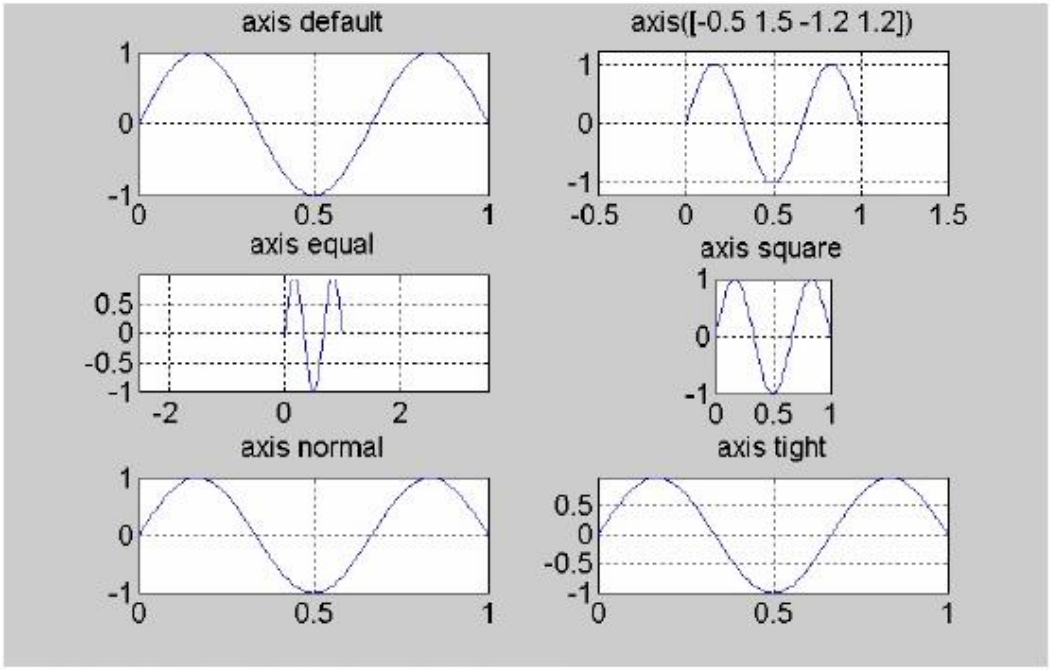
The three curves in the previous example can also be plotted on the same figure in three different plots using the *subplot* command.

iii) Plot the curves for Earth and Jupiter using *subplot*.

```
%Compute displacement for Earth
    t = 0:1:10;
    g = 9.80;
    y = g*t.^2;
% Plot
    subplot (2,2,1)
    plot (t,y,'b--')
    xlabel ('Time [s]')
    ylabel('Distance fallen [m]')
    title ('Distance fallen as a function of time on Earth')
    grid on

% Define displacement for Jupiter
    g=24.79;
    y = g*t.^2;
% Plot
    subplot(2,2,2)
    plot(t,y,'r-')
    xlabel('Time [s]')
    ylabel('Distance fallen [m]')
    title('Distance fallen as a function of time on Jupiter')
    grid on

% Define displacement for Mercury
    g=3.72;
    y = g*t.^2;
% Plot
    subplot(2,2,3)
    plot(t,y,'k-')
    xlabel('Time [s]')
    ylabel('Distance fallen [m]')
    title('Distance fallen as a function of time on Mercury')
    grid on
```



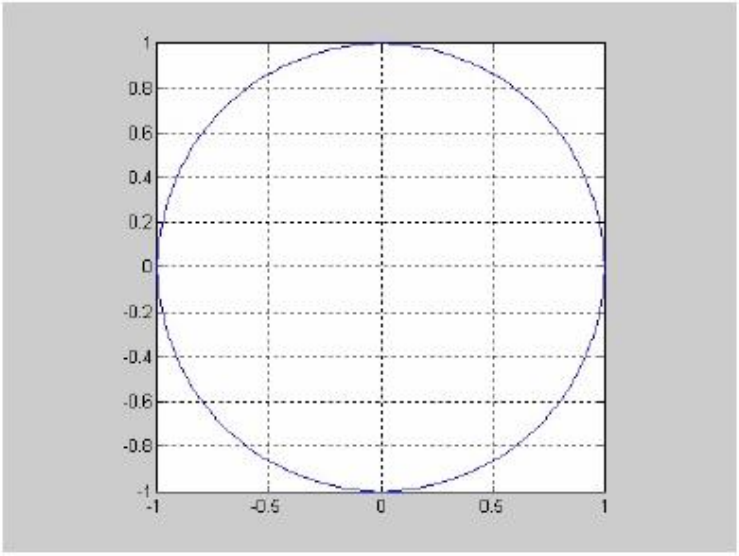
subplot(221) (or subplot(2,2,1)) specifies that the window should be split into a 2 x 2 array and we select the first sub-window for plotting the data.



It is easy to graph a curve ($f(t)$, $g(t)$) in 2D-space.

```
>> t = (0:2*pi/100:2*pi)';
>> plot(cos(t),sin(t))
>> axis('square')
>> grid
```

Note the use of the `axis('square')` command to make the circle appear round instead of elliptic).



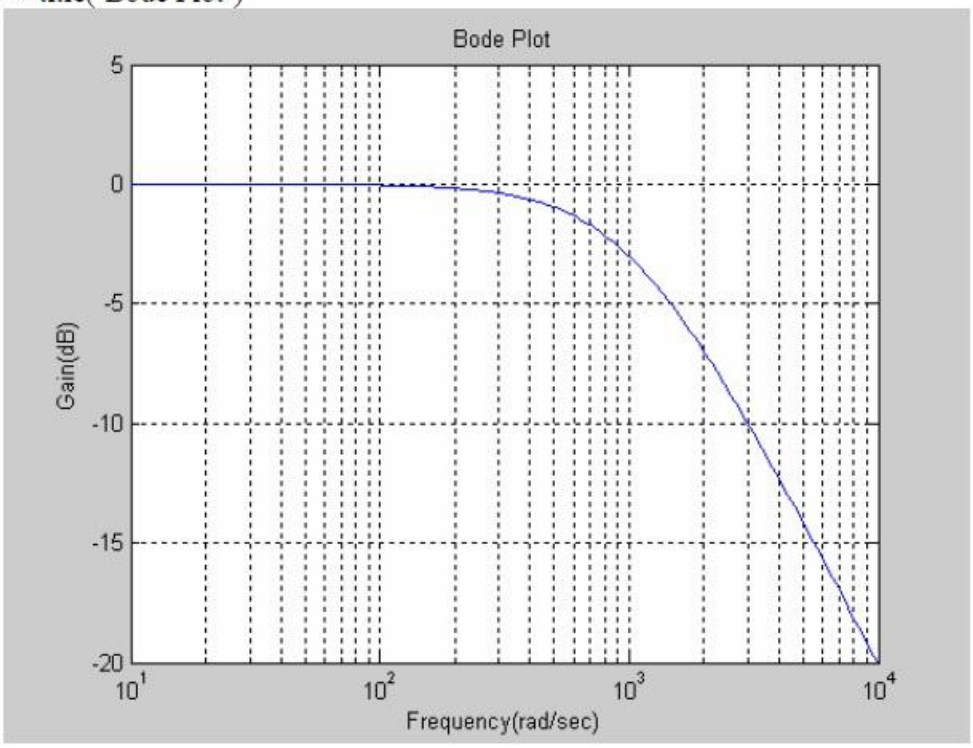
LAB # 5

2D Plot Types

semilogx

The semilogx command works just like the Plot command except that the x-axis is a log scale.

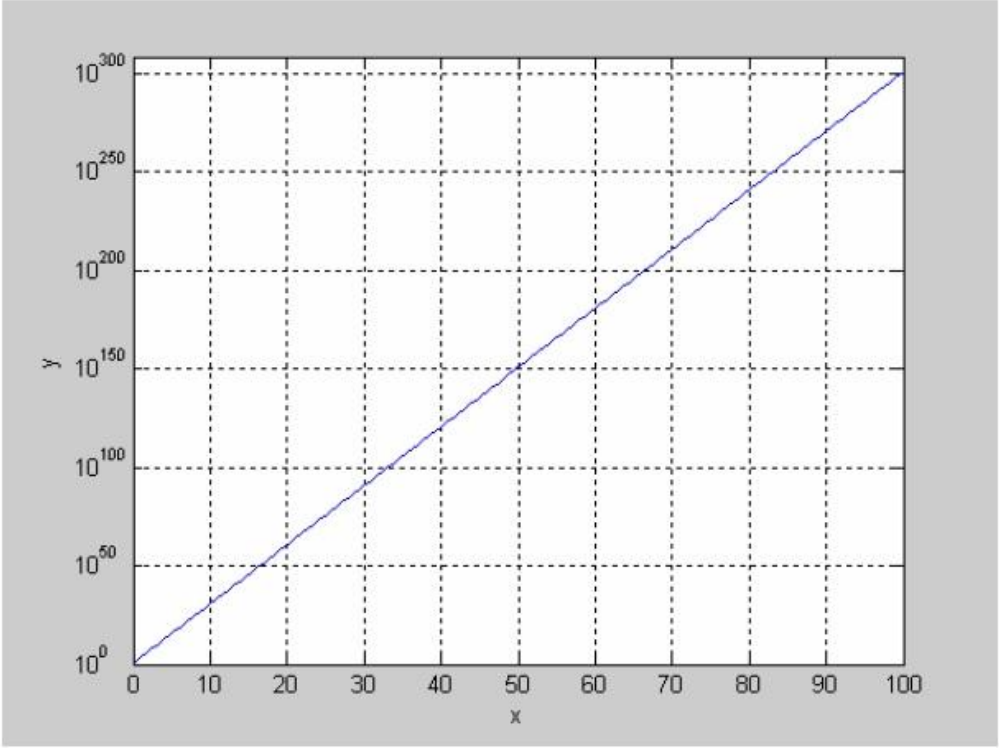
```
>>omega=logspace(1,4,200)
>>magnitude=20*log10(abs(1000./(i*omega+1000)))
>>semilogx(omega,magnitude)
>>axis([10,10000,-20 5])
>>grid on
>>ylabel('Gain(dB)')
>> xlabel('Frequency(rad/sec)');
>>title('Bode Plot')
```



semilogy

Plotting $y=5x10^{3x}$

```
x=linspace(0,100,200)
y=5.*10.^(3.*x);
semilogy(x,y)
ylabel('y')
xlabel('x')
grid
```

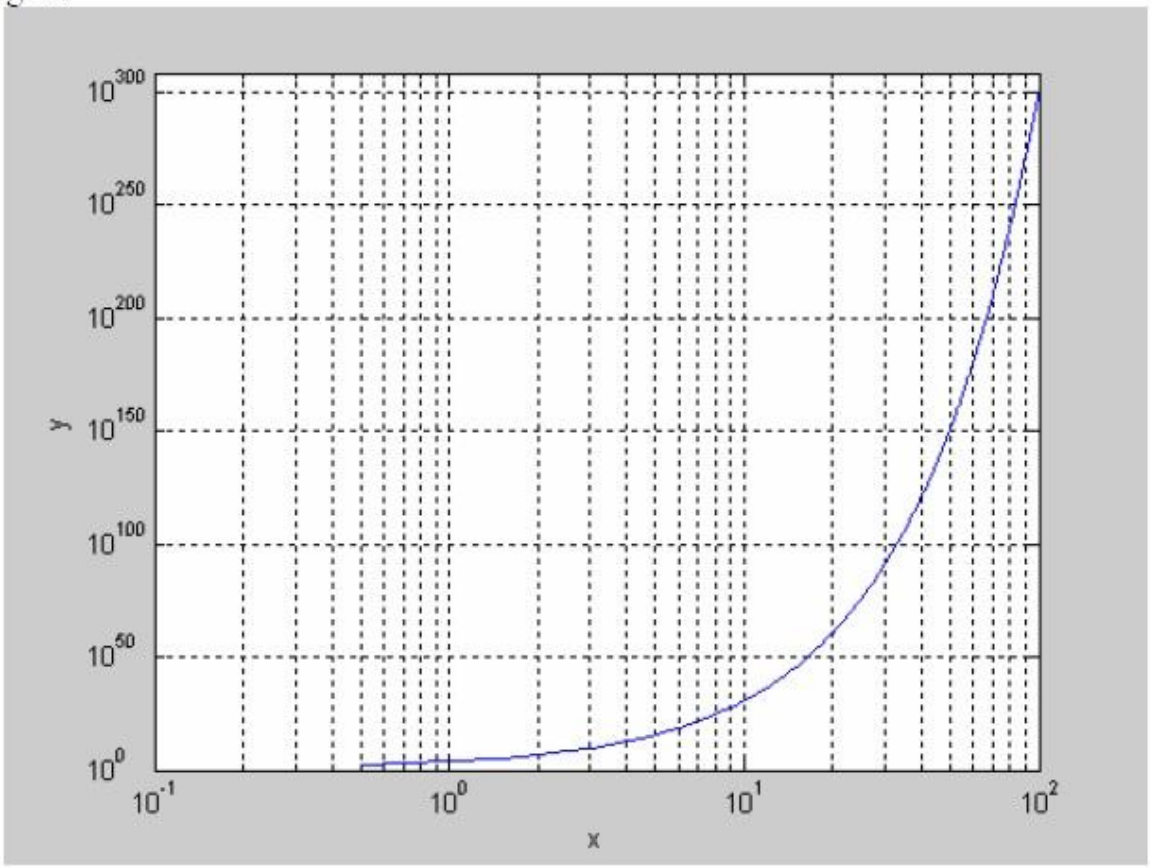


Note that the y-axis is now a log scale that is factors of 10 are equally spaced on the axis.

loglog

Plotting $y=5x10^{3x}$

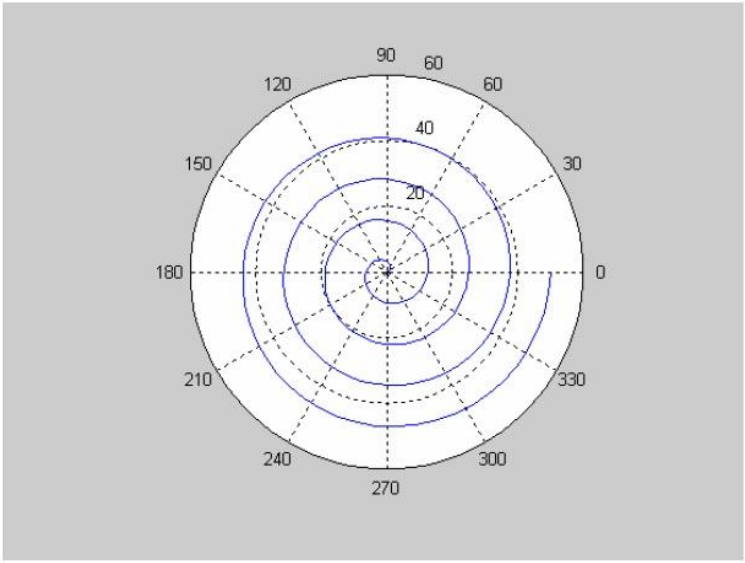
```
x=linspace(0,100,200)
y=5.*10.^(3.*x);
loglog(x,y)
ylabel('y')
xlabel('x')
grid
```



polar

A polar plot is used for data with polar coordinates.Let's plot a linear spiral r=2θ

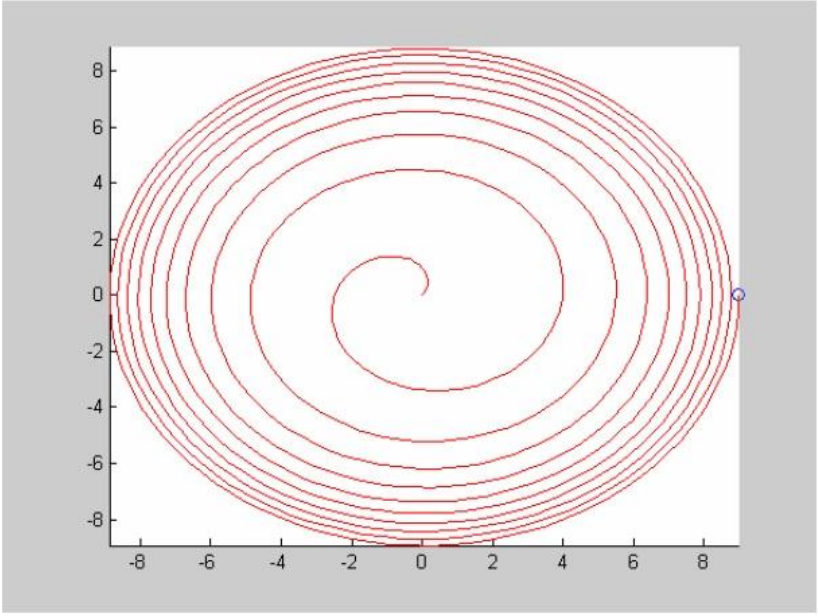
```
theta=linspace(0,8*pi,200)
r=2*theta;
polar(theta,r)
grid
```



comet

Comet function displays a plot and shows its trajectory as an animation.

```
theta=linspace(1,20*pi,1000)
r=5*log10(theta);
x=r.*cos(theta);
y=r.*sin(theta);
comet(x,y)
```



LAB # 6

Scripts & Functions

General

- For bigger tasks one wants to reuse code
- MATLAB program-files have extension ".m"
- Two possibilities: script-files and function-files
- Functions/scripts are called/executed by typing in filename (without ".m")

Script-Files

- Contain a series of MATLAB commands
- Commands are executed as if they were typed in
- All variables are global ⇒ interact with workspace
- Problem: must have matching variable-names / name overlapping

Function-Files

Structure

- Begin with declaration of function
  `function [out1, out2, ...] = funname(in1, in2, ...)`
  - `in1, in2, ...`: input parameter
  - `out1, out2, ...`: output parameter
- Followed by some rows of comments (comment starts after "%") which are displayed when entering ">> `help funname`'
- Body of function follows
- After Body subfunctions may follow

General

- Used variables are local
- Subfunctions are not visible outside ⇒ can structure code
- Name file as "`funname.m`"

Flow Control

Like programming languages MATLAB has branching and loops

Branching

- Mostly branching is done using `if`:

```
if condition1
     program1
elseif condition2
     program2
else
     program3
end
```

- A condition is as normal something like
  `(count < limit) & ((height - offset) >= 0)`
- Check operators for possible conditions/concatenations
- Be careful when using matrices in conditions (`A ~= B`)
- MATLAB also supports `switch`

Loops

- For-loop: Executes block of code specified number of times

```
for variable = expression
    statements
end
```

- While-loop: Repeatedly executes statements while condition is true

```
while condition
    statements
end
```

- Example:
  `for i = [1,2,4,5]; A(i) = i; end`
- More Flow Control for loops like `break`, `continue`, etc

Example

- x = 0:.1:1;
- y = [x; exp(x)];
- fprintf(' x exp(x) \n'), fprintf(' %6.2f %12.8f \n',y)

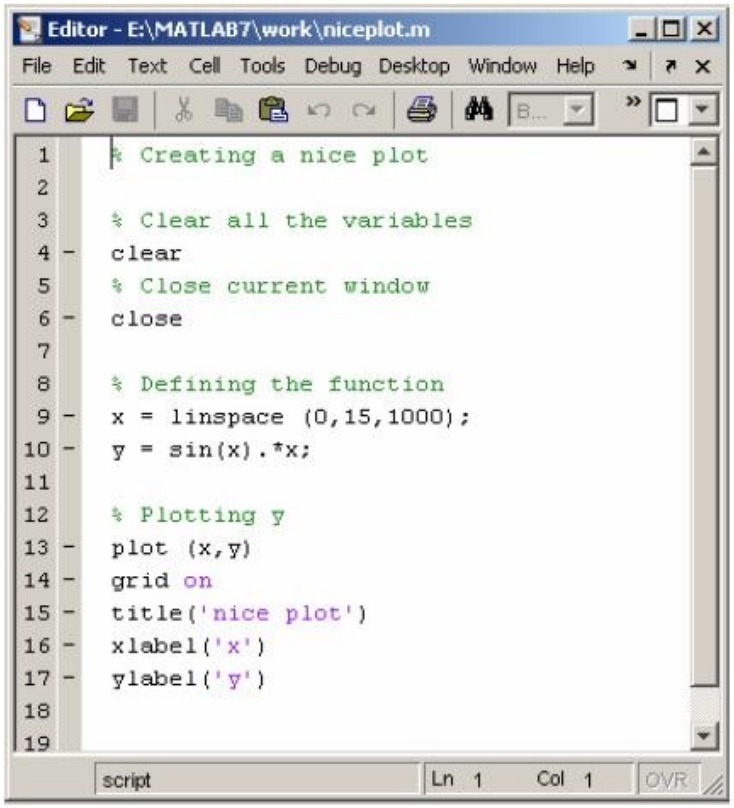
Notice, how the fprintf command is repeated for each column in y.

Example:

```
x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);
```

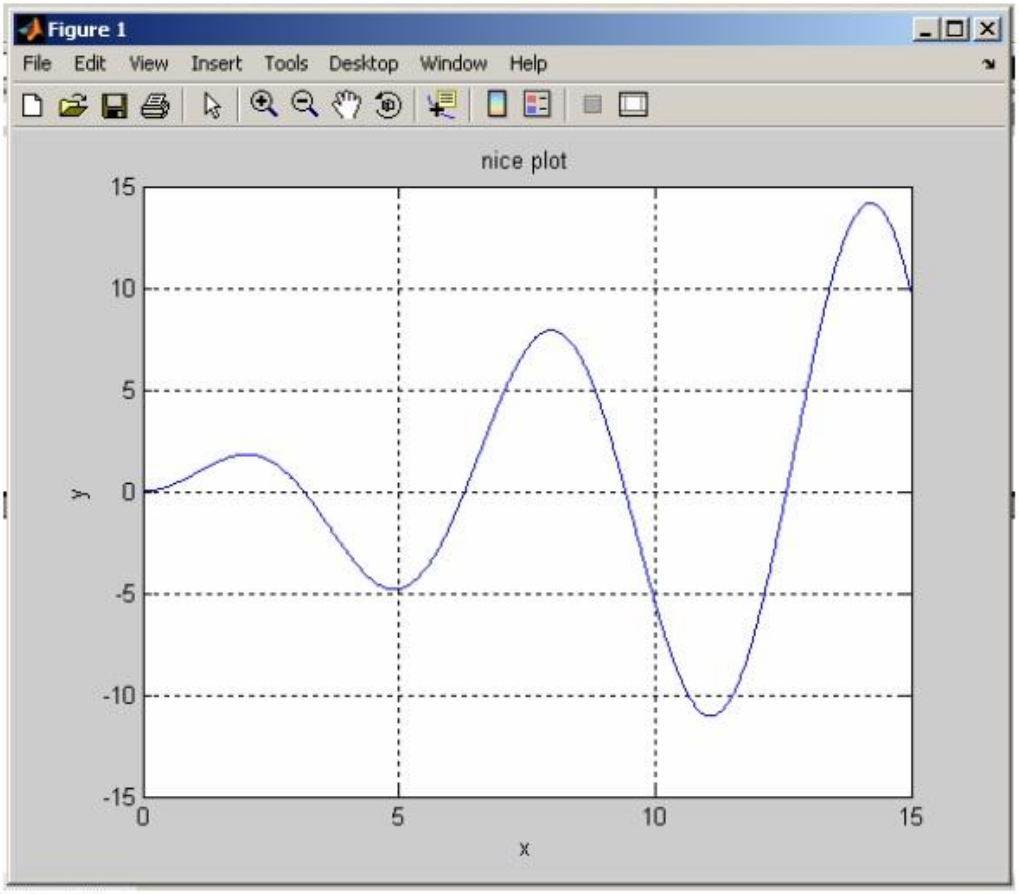
Example:

Open a new script (or .m) file using edit. Write Matlab commands to plot the function $y = x \sin x$, between 0 and 15 using 1000 points. Include labels and a title. Save your file as **niceplot.m.**



LAB # 7

In the command window, select the appropriate directory and execute the script by typing ***niceplot*** in the command window.



- It is a good idea to always start your script with a *clear* command. This will help eliminate a lot of unexpected errors associated with using previously computed values when you execute the script multiple times

- To insert comments within a script, start each line with a % symbol. The comments will appear green.

- If you type *help niceplot* in the command window, the comments contained at the beginning of the file will be displayed. This is how all help information is stored and displayed for existing MATLAB functions

Writing Functions in Matlab

When entering a command such as roots, plot, or step into Matlab what you are really doing is running an m-file with inputs and outputs that have been written to accomplish a specific task. These types of m-files are similar to subroutines in programming languages in that they have inputs (parameters which are passed to the m-file), outputs (values which are returned from the m-file), and a body of commands, which can contain local variables. Matlab calls these m-files functions. You can write your own functions using the function command.

The new function must be given a filename with a '.m' extension. This file should be saved in the same directory as the Matlab software, or in a directory, which is contained in MATLAB's search path. The first line of the file should contain the syntax for this function in the form:

function [output1,output2] = filename(input1,input2,input3)

A function can input or output as many variables as are needed. The next few lines contain the text that will appear when the help filename command is evoked. These lines are optional, but must be entered using % in front of each line in the same way that you include comments in an ordinary m-file. Finally, below the help text, the actual text of the function with all of the commands is included

Functions can be rather tricky to write, and practice will be necessary to successfully write one that will achieve the desired goal. Below is a simple example of what the function, add.m, might look like.

```
function [var3] = add(var1,var2)
%Add is a function that adds two numbers
var3 = var1+var2;
```

If you save these three lines in a file called "**add.m**" in the Matlab directory, then you can use it by typing at the command line:

```
>>y = add(3,8)
```

Obviously, most functions will be more complex than the one demonstrated here. This example just shows what the basic form looks like.

LAB # 8

Inline Function

From time to time you may need a quick function definition that is temporary—you don't care if the function is around tomorrow. You can avoid writing M-files for these functions using a special syntax. For example:

```
>> sc = inline('sin(x) + cos(x)')
sc = Inline function: sc(x) = sin(x) + cos(x)
>> sc([0 pi/4 pi/2 3*pi/4 pi])
ans =
1.0000 1.4142 1.0000 0.0000 -1.0000
```

You can also define functions of more than one variable, and name the variables explicitly:

```
>> w = inline('cos(x - c*t)','x','t','c')
w =
Inline function: w(x,t,c) = cos(x - c*t)
>> w(pi,1,pi/2)
ans =
6.1232e-17
```

Function of function

In many cases you need to use the name of a function as an argument to another function. For example, the function `fzero` finds a root of a scalar function of one variable. So we could say

```
>> fzero('sin',3)

ans =
    3.1416

>> fzero('exp(x)-3*x',1)

ans =
    0.6191
```

If you need to find the root of a more complicated function, or a function with a parameter, then you can write it in an M-file and pass the name of that function. Say we have

```
function f = demo(x,a)
exp(x) - a*x;
```

Then we can use

```
>> fzero(@demo,1,[],3)

ans =
    0.6191
```

If – elseif – else statements

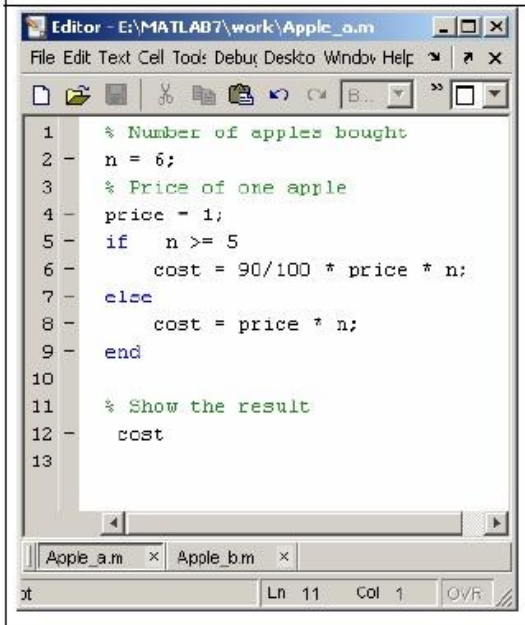
Command :

```
if      condition
        Statement
elseif  condition
        statement
else
        statement
end
```

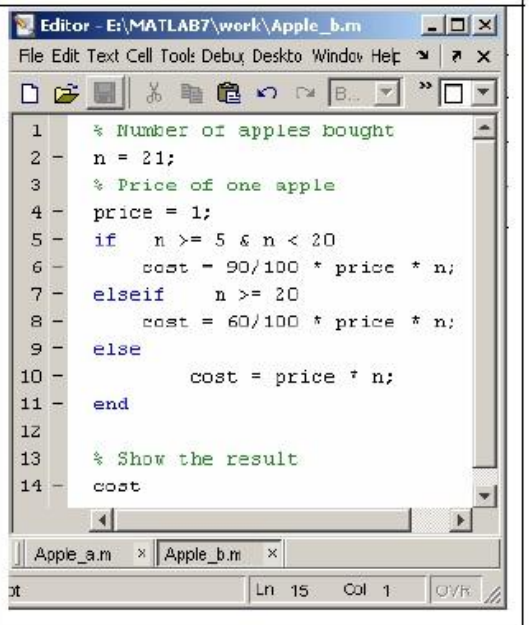
a) Suppose you are buying apples at a supermarket. If you buy more than 5 apples, you get a 10% discount. Write a program computing the amount owed given the price per apple (assume it to be $1) and the number of apples bought.

b) Suppose now the store offers a super discount of 40% if you buy more than 20 apples. Modify the code in part a) to implement this new condition using the *elseif* statement.

(a)



(b)



for loops

Command :

for *variable = initial value : increment : final value*
statement
statement
end

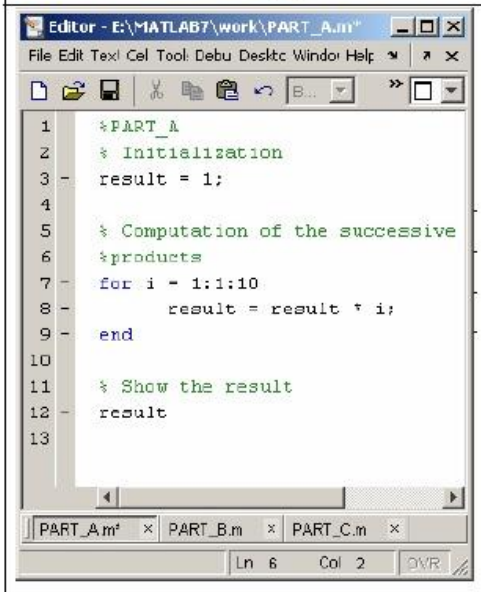
a) Compute 10!, i.e. the product of all the integers from 1 to 10.

b) Compute the sum of all even numbers from 0 to 100.

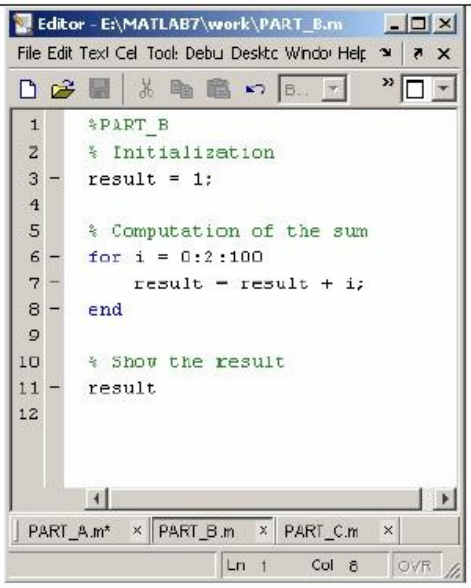
c) Define the following piecewise function between $x=-3$ and $x=5$ using 1000 points and plot it.

$$\begin{cases} y(x)=x & x<0 \\ y(x)=\sin \pi x & 0\le x\le 1 \\ y(x)=\ln x & x>1 \end{cases}$$

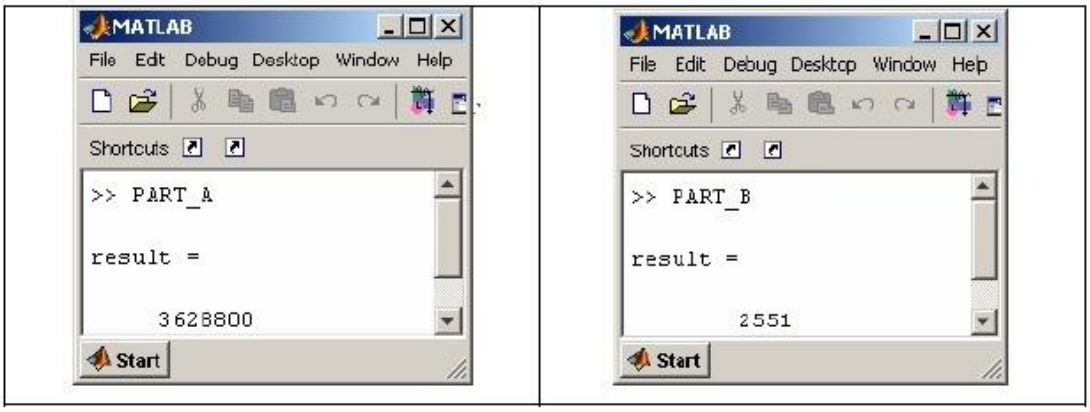
PART_A



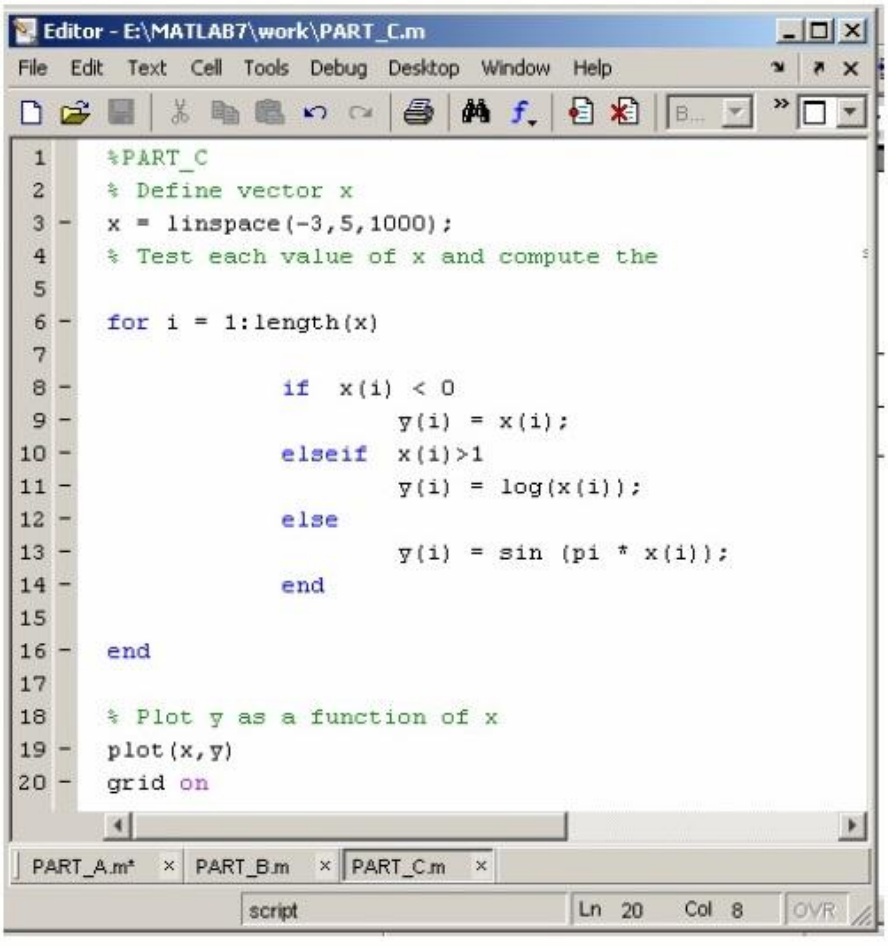
PART_B



LAB # 9


MATLAB
File Edit Debug Desktop Window Help
Shortcuts
>> PART_A
result =
3628800
Start
MATLAB
File Edit Debug Desktop Window Help
Shortcuts
>> PART_B
result =
2551
Start

PART_C


Editor - E:\MATLAB7\work\PART_C.m
File Edit Text Cell Tools Debug Desktop Window Help
%PART_C
% Define vector x
x = linspace(-3,5,1000);
% Test each value of x and compute the
for i = 1:length(x)
if x(i) < 0
y(i) = x(i);
elseif x(i)>1
y(i) = log(x(i));
else
y(i) = sin (pi * x(i));
end
end
% Plot y as a function of x
plot(x,y)
grid on
PART_A.m*
PART_B.m
PART_C.m
script
Ln 20 Col 8
OVR


MATLAB
File Edit Debug Desktop Window Help
Shortcuts
>> PART_C
>>
Start

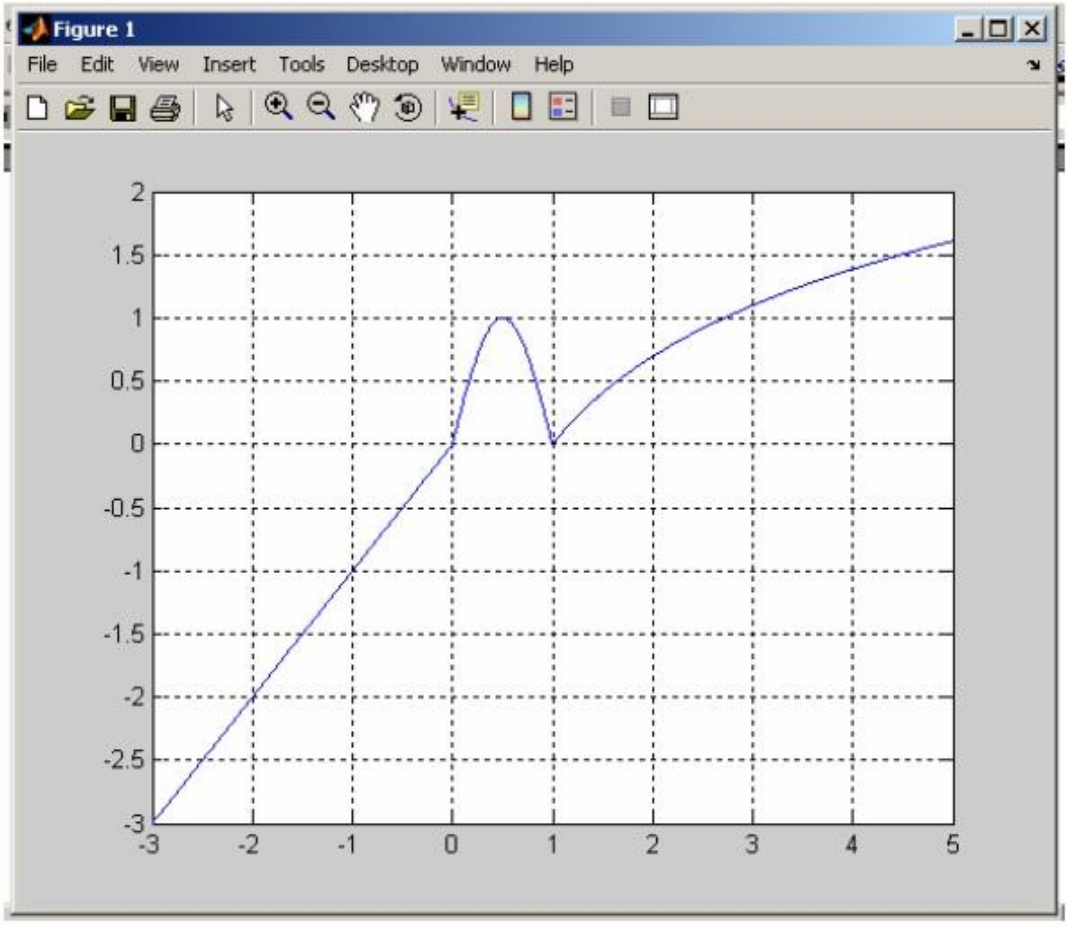


Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3 -2 -1 0 1 2 3 4 5

While loops

Command(s):

while *condition*
statement
end

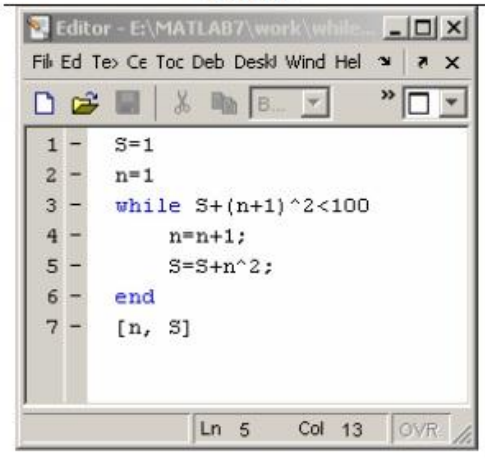
abs (x) returns the absolute value of *x*
sqrt (x) returns the square root of *x*

There are some occasions when we want to repeat section of MATLAB code until some logical condition is satisfied, but we cannot tell in advance how many times we have to go around the loop. This we can do with a **while…end** construct.

What is the greatest value of n that can be used in the sum

$1^2 + 2^2 + \ldots + n^2$ and get a value less than 100?

CODE



RESULT

% This function computes the value of n factorial using for loop

```
function [product]=factorialforloop(n)
product=1;
for i=1:n
   product=product*i;
end
```

% This function computes the value of n factorial using if statements

```
function [product]=factorialifstatement(n)
product=1;
i=1;
while (1)
   product=product*i;
   i=i+1;
   if i>n
      break
   end
end
```

% This function computes the value % of n factorial using while loop

```
function [product]=factorialwhileloop(n)
product=1;
i=1;
while i<=n
   product=product*i;
   i=i+1;
end
```

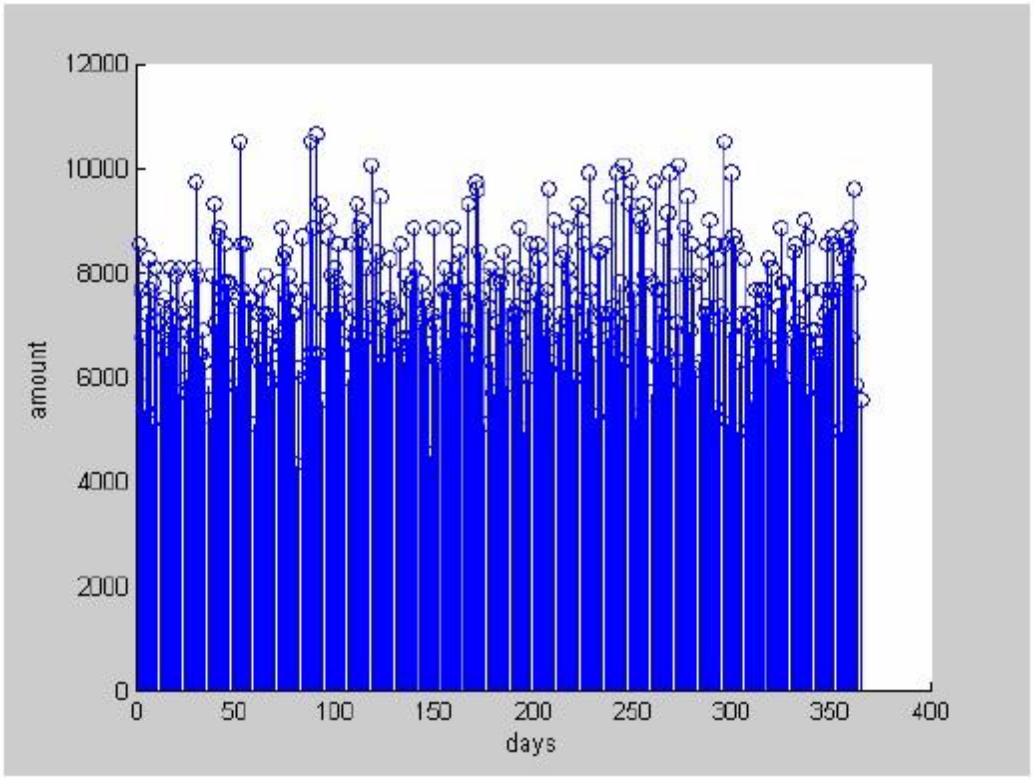
Random Numbers, Histogram, Mean and Stem Plot

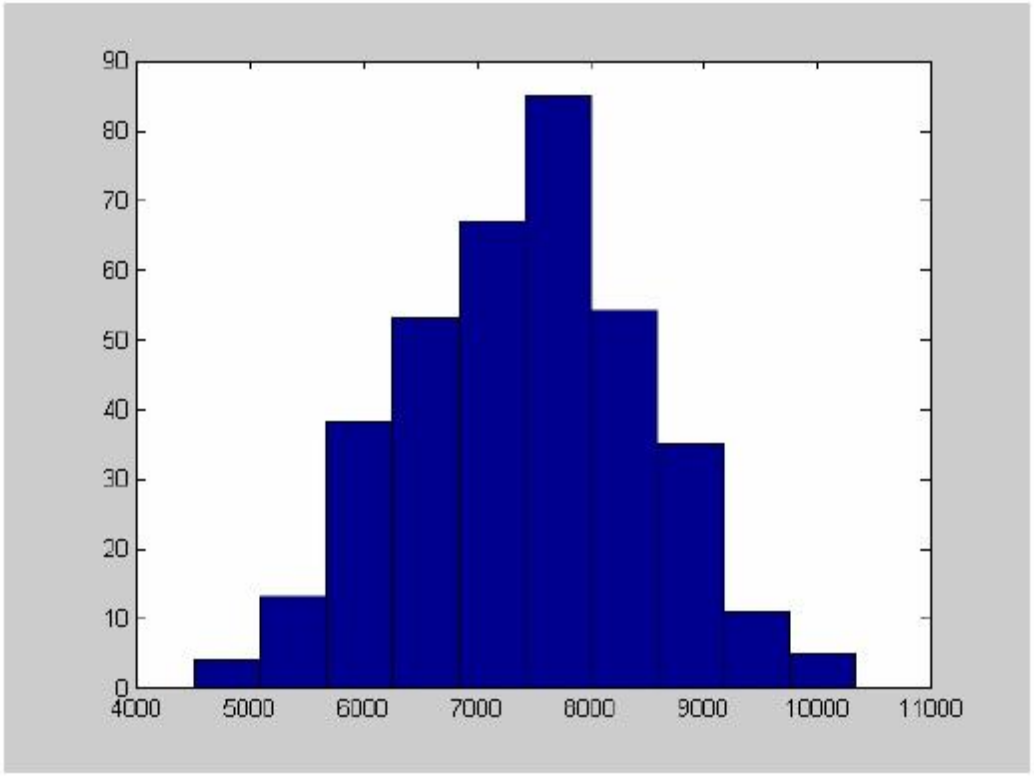
Command(s) :

mean	Compute the average of the components of a vector
stem	Create a stem plot

Highway patrol uses radar to measure the speed of the cars on a freeway. The speed limit is 55 mph. The officers pull over all cars whose speed is greater than 80 mph and write a ticket for $150 each. If 8000 vehicles are observed on the highway each day, on the average, how much money does the highway patrol collect each day? Compute the average over one full year and assume A = 10 in the **randn** command.

```
clear all
close all
% the average of the daily amount is done over a year
for i = 1:365
% create a vector whose components are the speed of  each car  taking the freeway
x = randn(8000,1)*10 + 55;
% Initialize the amount collected each day
S(i) = 0;
% test the speed of each vehicle. If it's greater than 80, add  150 to S
   for k = 1:length(x)
      if x(k) > 80
         S(i) = S(i) + 150;
      end
   end
end

% Compute the average of vector S containing the amount
%collected each day during a year
Savg = mean(S)
% Plot the amount of money collected each day
stem(S)
xlabel('days')
ylabel('amount')
```


amount
days
0
2000
4000
6000
8000
10000
12000
0
50
100
150
200
250
300
350
400


0
10
20
30
40
50
60
70
80
90
4000
5000
6000
7000
8000
9000
10000
11000

LAB # 10

Commands

[meshgrid]	Creates X and Y arrays for 3-D plots Create a two-dimensional grid of coordinate values
[mesh]	Create wireframe parametric surfaces specified by X, Y, and Z
[meshc]	Draws the mesh surface together with the contour plot beneath the plotted surface
[surf]	Creates a shaded surface using Z for the color data as well as surface height
[surfc]	surfc(...) is the same as surf(...) except that a contour plot is drawn beneath the surface.
[surface]	Create a 3D surface on the domain defined by **meshgrid**
[shading]	SHADING controls the color shading of SURFACE. SHADING **FLAT** sets the shading of the current graph to flat. SHADING **INTERP** sets the shading to interpolate. SHADING **FACETED** sets the shading to faceted, which is the default.
[colorbar]	Adds a Colorbar
[colormap()]	COLORMAP(MAP) sets the current figure's colormap to MAP. Default is jet. Others are copper, hsv, spring, Autumn, winter, summer etc.
[contour()]	CONTOUR(Z) is a contour plot of matrix Z treating the values in Z as heights above a plane. A contour plot is the level curves of Z for some values V. The values V are chosen automatically
[contourf()]	The same as CONTOUR(...) except that the contours are filled. Areas of the data at or above a given level are filled.
[plot3()]	A three-dimensional analogue of plot().
[sphere()]	sphere command. creates 3 n+1 by n+1 matrices that can then be used in the **mesh** or **surf** commands to create unit spheres.
[cylinder()]	
[surfnorm()]	surfnorm(X,Y,Z) or surfnorm(Z) plots the surface with the normals emanating from it.

In order to create a graph of a surface in 3D-space (or a contour plot of a surface), it is necessary to evaluate the function on a regular rectangular grid. This can be done using the meshgrid command.

First, create 1D vectors describing the grids in the x- and y-directions:

```
>> x = (0:2*pi/20:2*pi)';
>> y = (0:4*pi/40:4*pi)';
```

Next, ``spread'' these grids into two dimensions using `meshgrid`:

```
>> [X,Y] = meshgrid(x,y);
>> whos
  Name        Size              Bytes  Class
  X          41x21              6888  double array
  Y          41x21              6888  double array
  x          21x1                168  double array
  y          41x1                328  double array
```

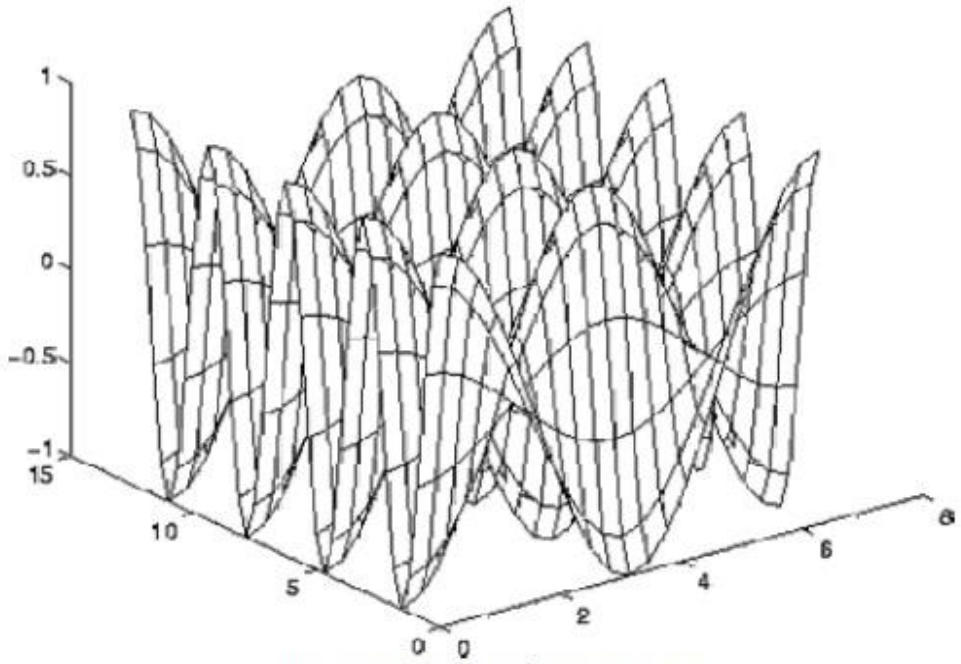
Grand total is 1784 elements using 14272 bytes

The effect of meshgrid is to create a vector X with the x-grid along each row, and a vector Y with the y-grid along each column. Then, using vectorized functions and/or operators, it is easy to evaluate a function $z = f(x,y)$ of two variables on the rectangular grid:

```
>> z = cos(X).*cos(2*Y);
```

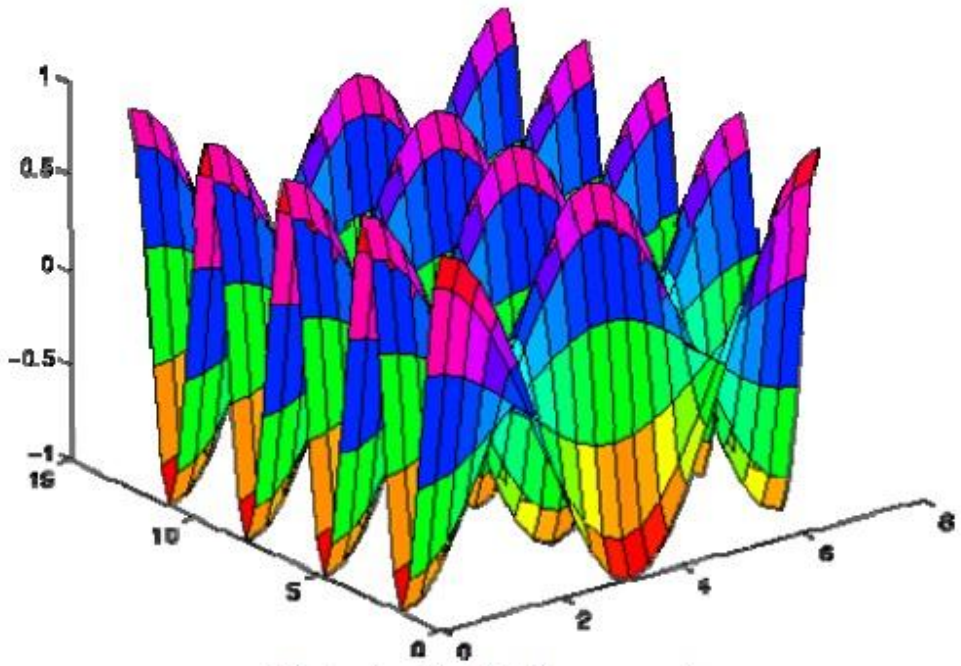
Having created the matrix containing the samples of the function, the surface can be graphed using either the **mesh** or the **surf** commands.

>> mesh(x,y,z)



Plot using the **mesh** command

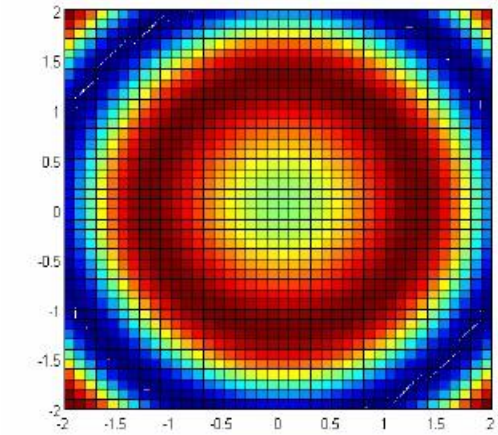
>> surf(x,y,z)



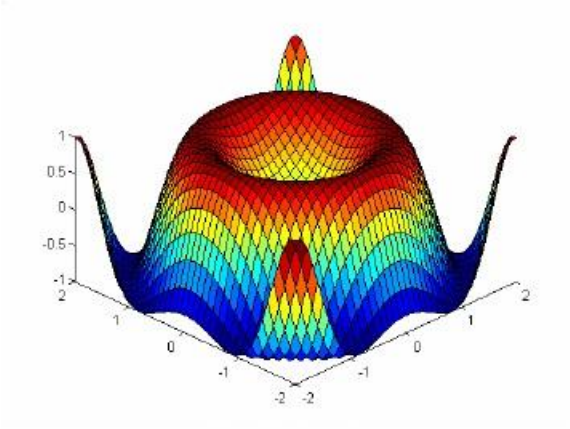
Plot using the **surf** command

The difference is that surf shades the surface, while mesh does not.

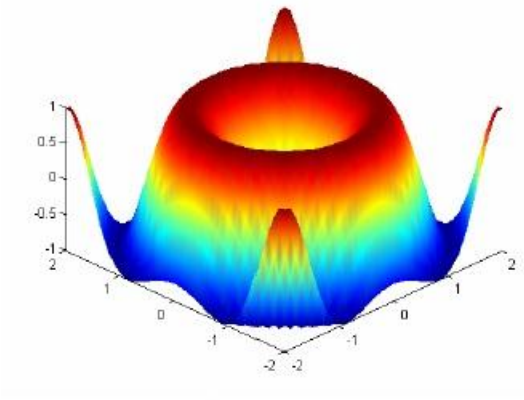
```
[x,y]=meshgrid(-2:0.1:2,-2:0.1:2);  z=sin(x.^2+y.^2); surface(x,y,z) pause
view(-45,45)  pause
shading('interp'); pause
grid on;  pause colorbar
```



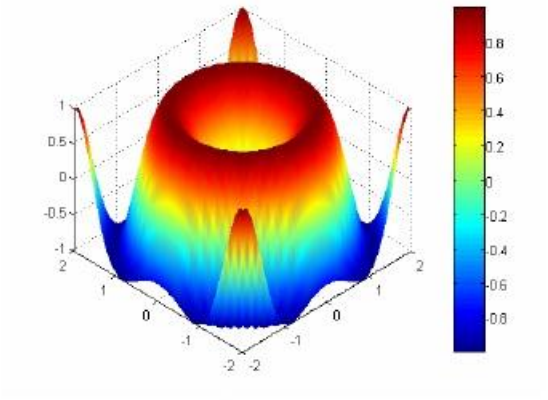
surf(x,y,z)



view(-45,45)



shading('interp');



grid on; colorbar

Example-I:

Plot the function $z(x,y)=\dfrac{\sin\left(x^2+y^2\right)}{x^2+y^2+10^{-16}}$ for x ranging between -3 and 3, and y ranging between –4 and 4. Use an increment of 0.1 in both directions. Include labels and a title. Once the figure appears on the screen, rotate it using one of the menu buttons. Use the command **shading('interp')** to remove the discrete mesh. Add a colorbar to the figure.

The 10^{-16} in the denominator is needed to avoid division by zero when $x=y=0$ and to get the correct value in the limit: $\lim\limits_{\substack{x\to 0\\ y\to 0}}\dfrac{\sin\left(x^2+y^2\right)}{x^2+y^2}=1$

clear; close all; [x,y] = meshgrid(-3:0.1:3,-4:0.1:4); z = sin(x.^2+y.^2)./(x.^2+y.^2+10^-16);

plot3(x,y,z) grid on

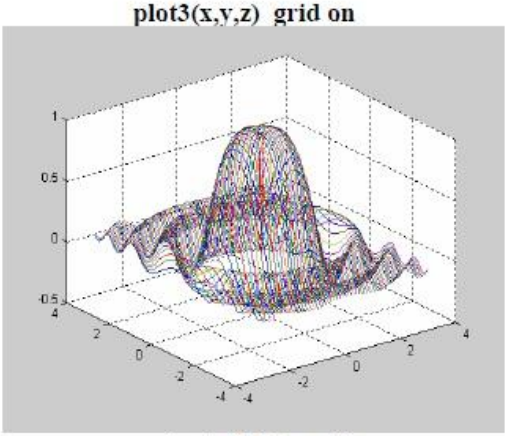
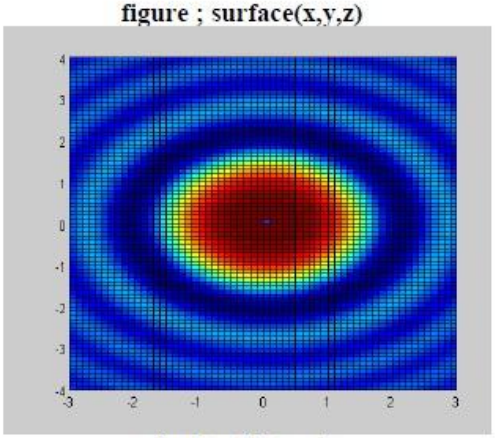
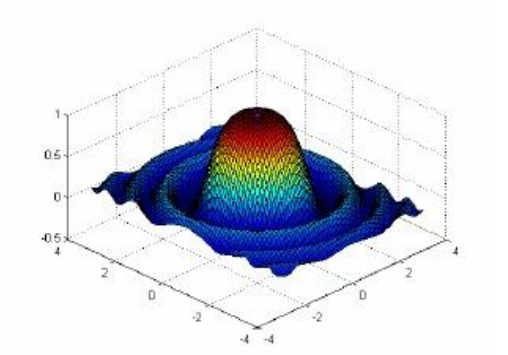


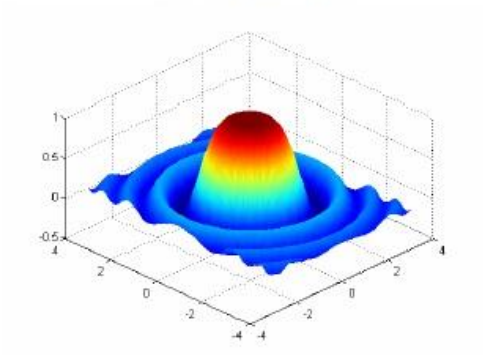
figure ; surface(x,y,z)



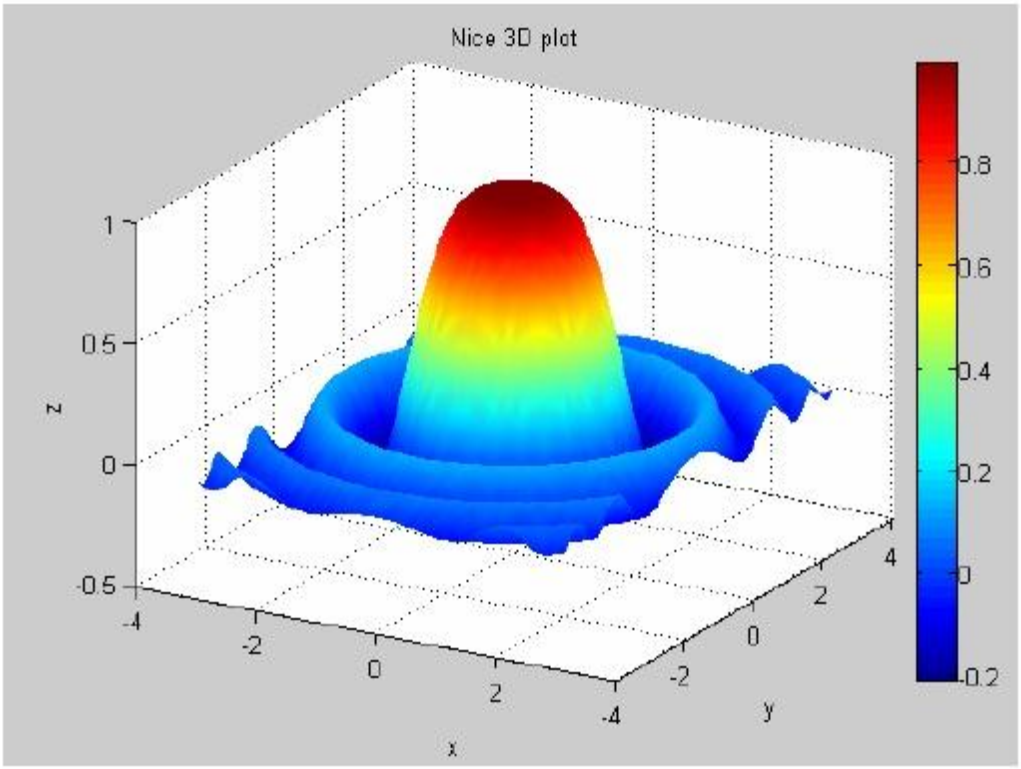
view(-45,45), grid



shading('interp')



```
shading('interp')
colorbar
xlabel('x')
ylabel('y')
zlabel('z')
title('Nice 3D plot')
```



Example-II

Consider the following function :

$$z(x,y) = y\cos(x)\sin(2y)$$

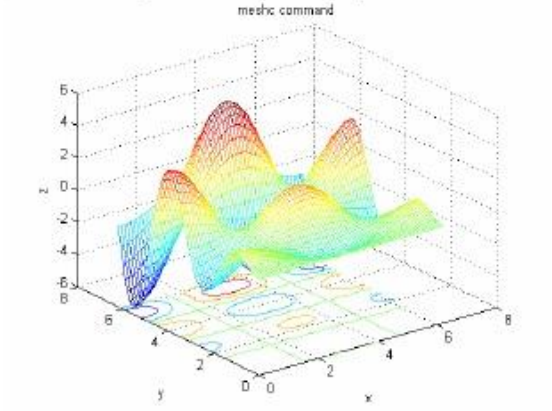
a) Plot the function z for x&y ranging between 0 and 2π, with an increment of 0.1 in both directions using the **meshc** command. This command draws the mesh surface together with the contour plot beneath the plotted surface.

b) Make a contour plot of the function above using the **contour** command. This will only display the level curves in a horizontal plane. Include labels and a title.

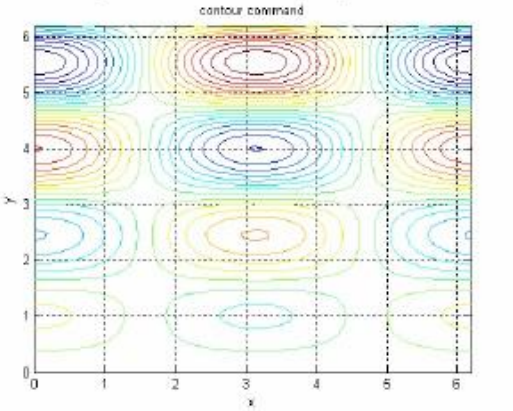
✍ Using the contour command, you can specify the number of levels you would like to be displayed. The default number is 10.

```
close all ; clear
% Define the grid
[x,y]=meshgrid(0:0.1:2*pi);
% Define the function of two variables
z = y.*cos(x).*sin(2*y);
```

```
% Plot both surface and level curves
    meshc(x,y,z)
    grid on
    xlabel('x')
    ylabel('y')
    zlabel('z')
    title('meshc command')
```




```
% Draw 20 level curves
    figure
    contour(x,y,z,20)
    grid on
    xlabel('x')
    ylabel('y')
     title('contour command')
```

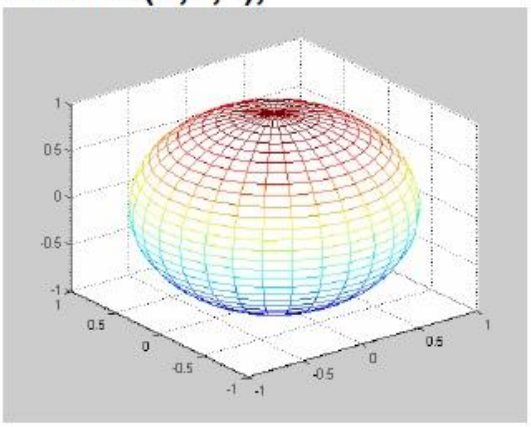


3D Sphere

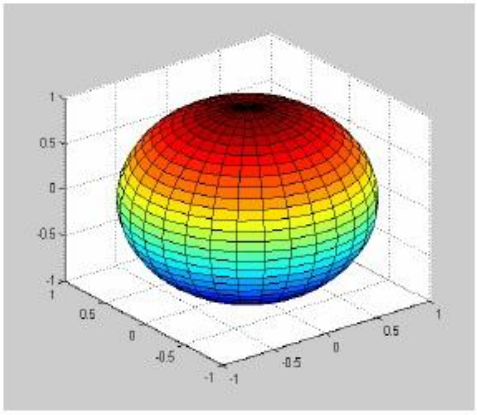
If you want to create a unit sphere, you can use the **sphere** command. It will create 3 n+1 by n+1 matrices that can then be used in the **mesh** or **surf** commands to create unit spheres.

In the sphere command 'n' is designated by the number inside the parentheses following the command sphere. Function **surf** is used to visualize data as shaded surface.

>>[X,Y,Z] = sphere(30);
>> mesh(X,Y,Z);



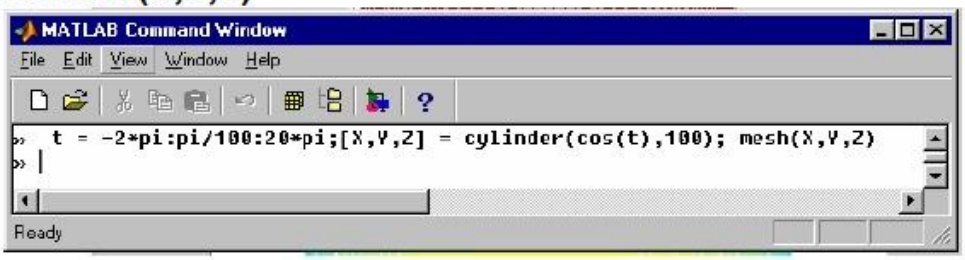
>> surf(X,Y,Z)

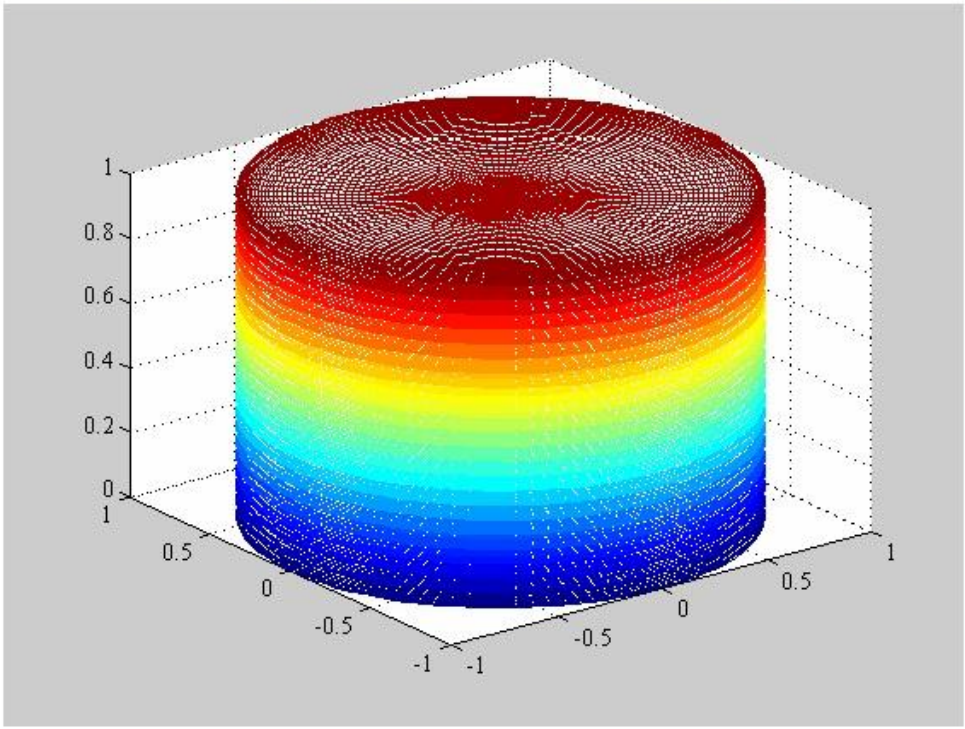


3D Cylinder

You can also create a unit cylinder by using the cylinder command. You must state a vector **R**(defines the radius of the cylinder along the z-axis) and number **N**(specifying number of points used to define circumference of the cylinder) in the parentheses following **cylinder**.

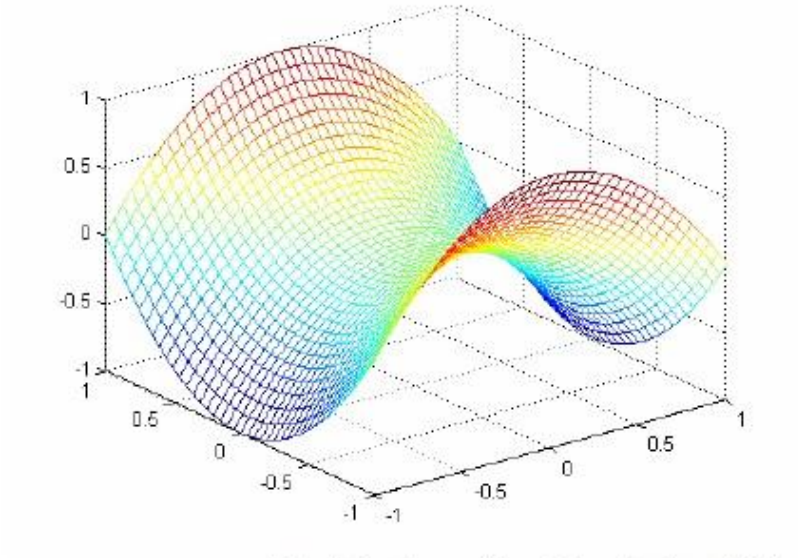
```
>>t = -2*pi:pi/100:2*pi;
>>[X,Y,Z] = cylinder(cos(t),20);
>>mesh(X,Y,Z)
```





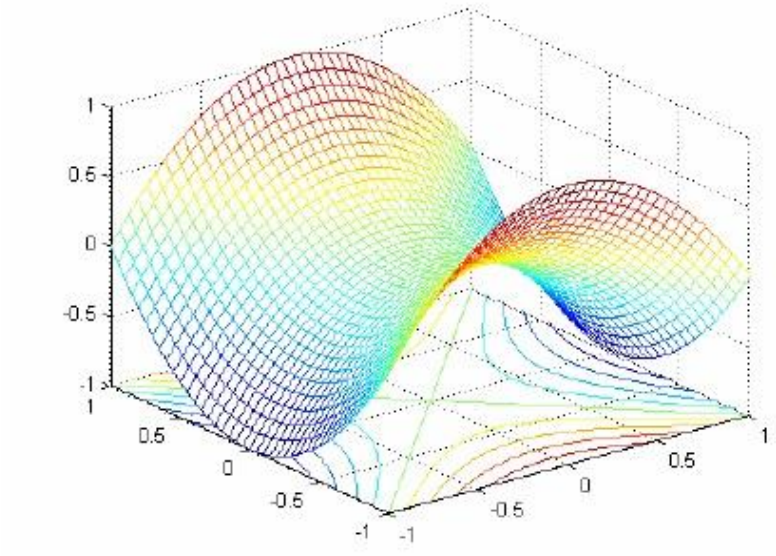
Plotting the Hyperbolic Paraboloid $z = y^2 - x^2$

```
x=-1:0.05:1; y=x; [xi,yi]=meshgrid(x,y); zi=yi.^2-xi.^2;
mesh(xi,yi,zi)
```



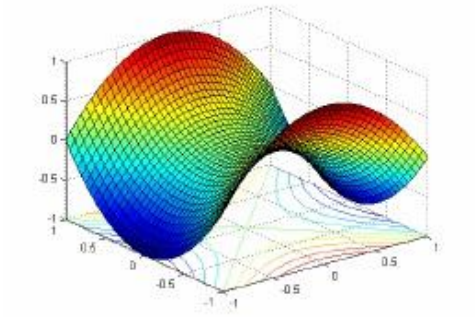
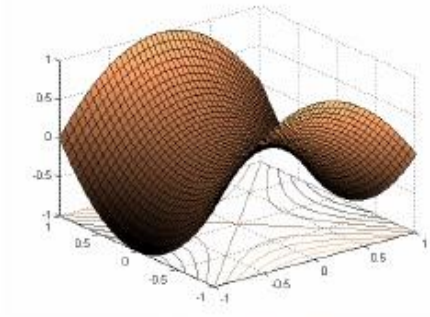
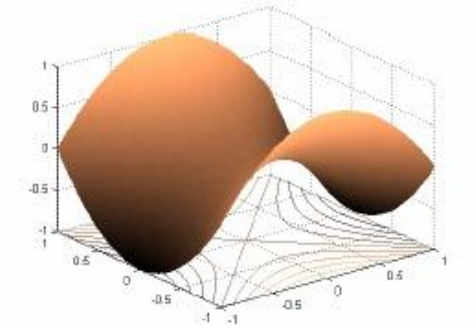
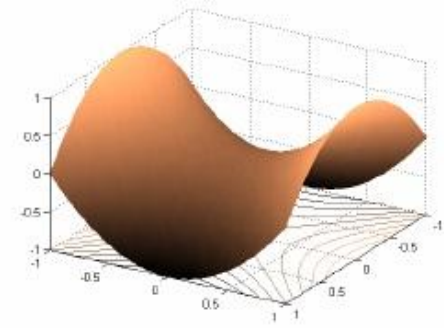
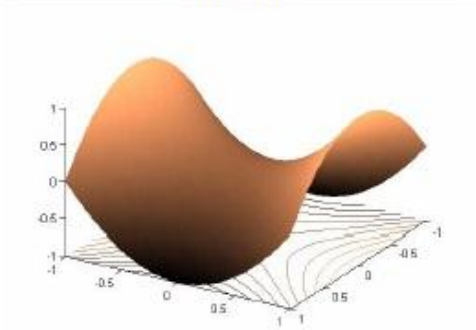
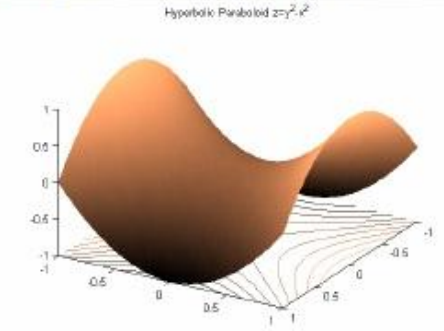
Mesh Surface without the Contour Plot

```
meshc(xi,yi,zi)
```

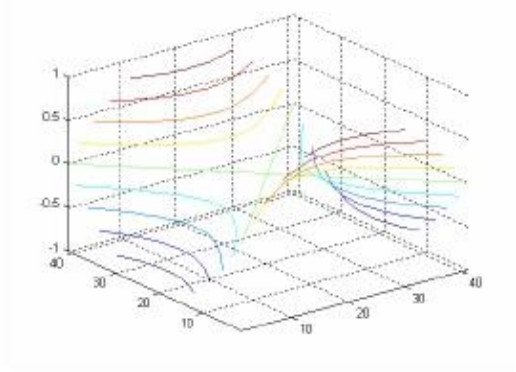
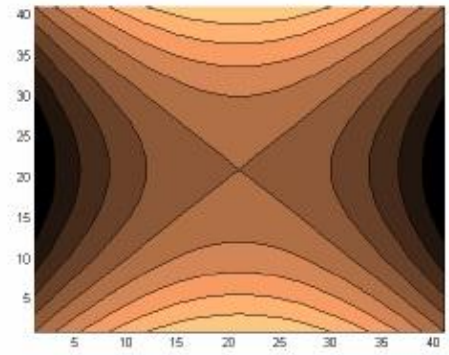
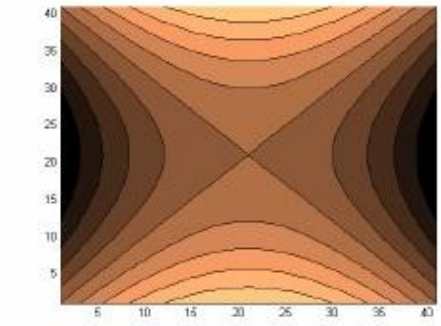
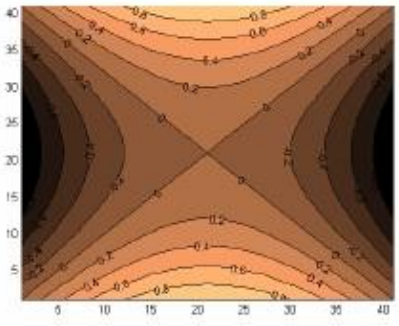
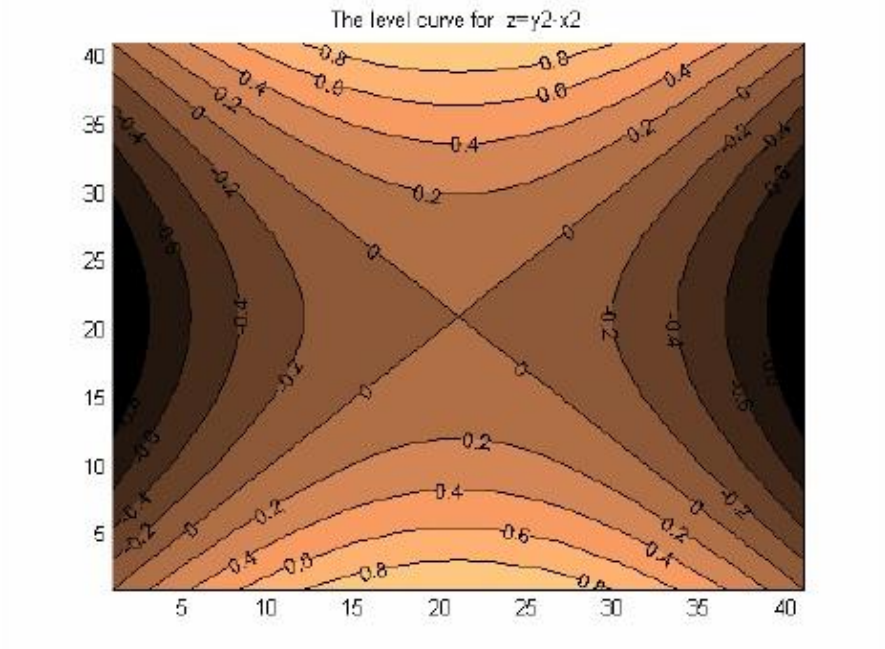


Mesh Surface together with the Contour Plot

x=-1:0.05:1; y=x; [xi,yi]=meshgrid(x,y); zi=yi.^2-xi.^2;

surfc(xi,yi,zi);	colormap copper;
	
shading interp;	**view([25,15,20]);**
	
grid off;	**title('Hyperbolic Paraboloid z=y2-x2')**
	

MESHC(...) is the same as MESH(...) except that a contour plot is drawn beneath the mesh. Because CONTOUR does not handle irregularly spaced data, this routine only works for surfaces defined on a rectangular grid. The matrices or vectors X and Y define the axis limits only.

contour3(zi);	contourf(zi);
	
[c,h]=contour(zi,'k-');	clabel(c,h);
	
title('The level curve for z=y2-x2')	
	

LAB # 11

Polynomials

Polynomials in MATLAB are represented as row arrays. The elements of the row array represent the coefficients of the polynomial. Many computational activities in engineering and science require a polynomial. Suppose we have a function $y=x^2+2x+1$. In MATLAB this polynomial is represented as [1 2 1].

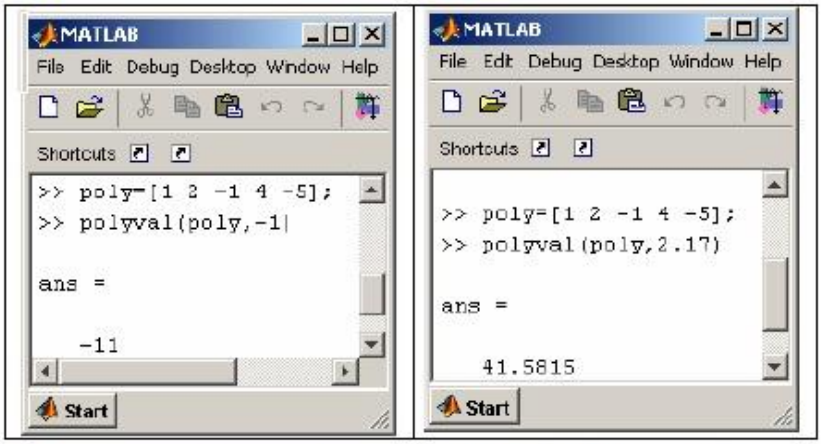
A polynomial of an arbitrary order is represented by

$$a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots. \; a_n x + a_{n+1}$$

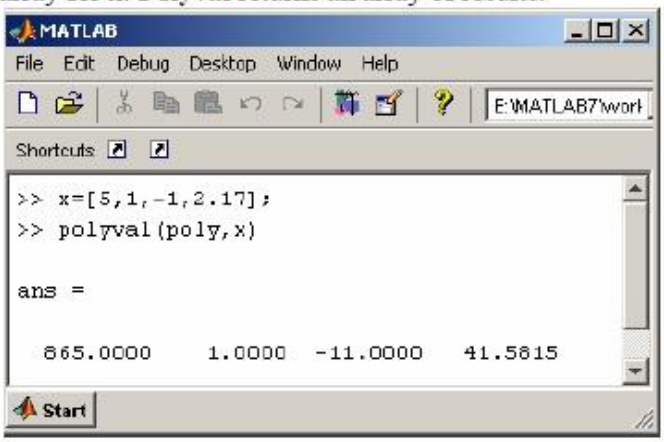
In MATLAB, this polynomial is represented as $[a_1\ a_2\ a_3 + \ldots + a_n\ a_{n+1}]$. The length of the row array is **n+1**.

Evaluating Polynomials

How do we can evaluate a polynomial for an arbitrary value of x (say x=5).
MATLAB provides the polyval function for evaluating polynomials.



A polynomial can be evaluated at all the desired individual points at the same time by creating a row array for x. **Polyval** returns an array of results.

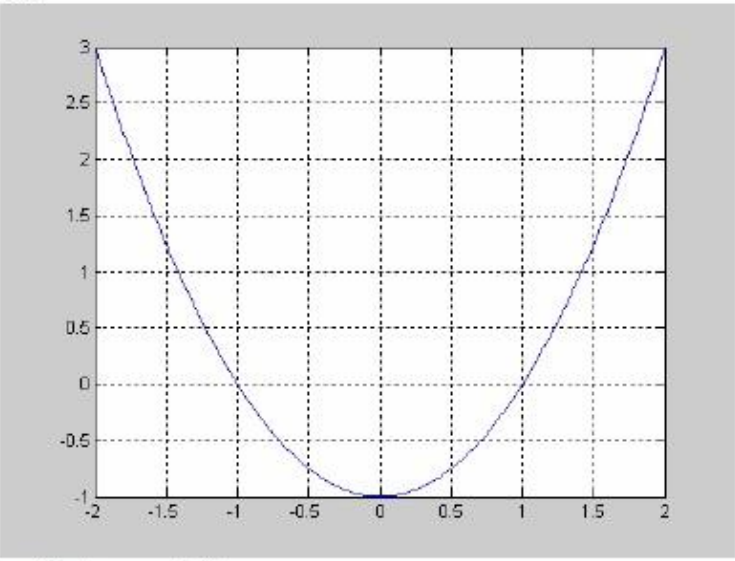


In this case Polyval command returns an array of results.

Plotting Polynomials

After knowing how to represent polynomials in MATLAB, we can easily plot them.

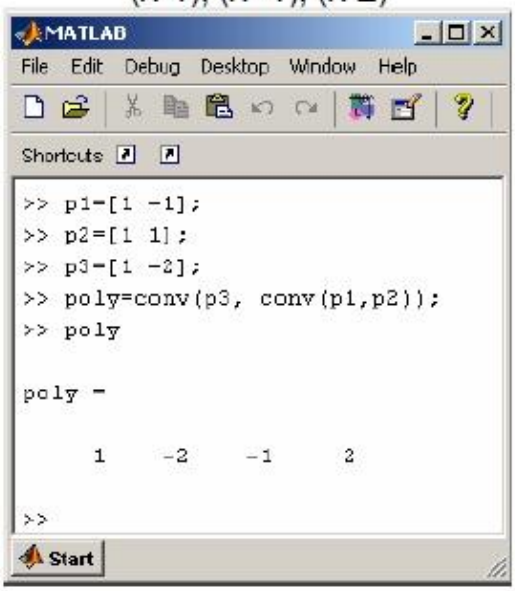
```
>>poly=[1  0 -1] %y=(x+1)(x-1)=x^2-1
>>x=linspace(-2,2,100);
>>y=polyval(poly,x);
>>plot(x,y)
>>grid
```



Multiplying Polynomials

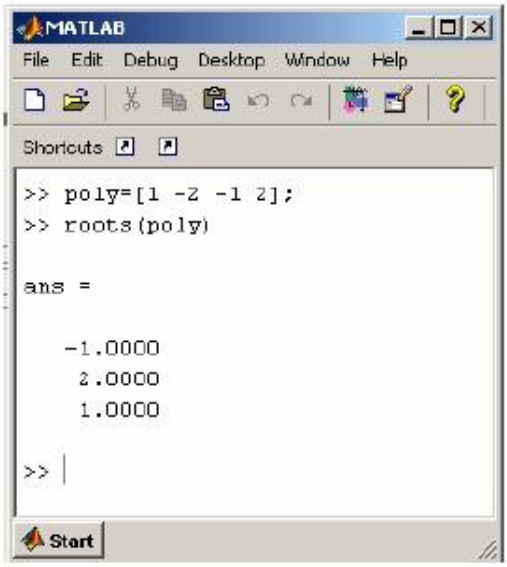
The MATLAB function conv multiplies two polynomials and returns the resultant polynomial as a row array. Suppose, we want to multiply the following three simple 1st order polynomials to get the resultant 3rd order polynomial.

(x-1), (x+1), (x-2)



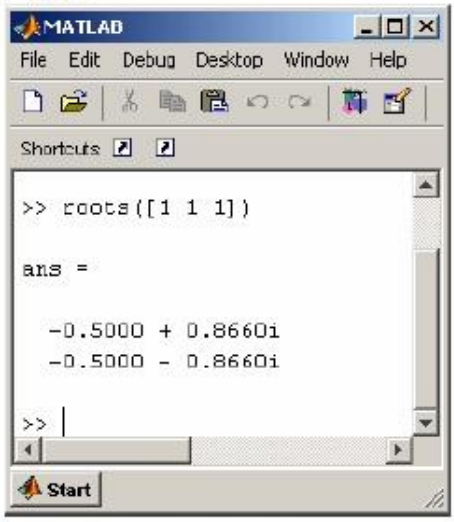
Roots of Polynomials

Consider the polynomial $y=f(x)=x^3-2x^2-x+2$. MATLAB provides a function called roots, Which finds the solution x to the equation y=f(x)=0 where f(x) is a polynomial.



Complex Roots of Polynomials

The roots function can also find the complex roots of a polynomial.
Consider solving $y=f(x)=x^2+x+1$.



This result says that the solution to $y=f(x)=x^2+x+1$ is

$$x= -0.5000 + 0.8660i$$

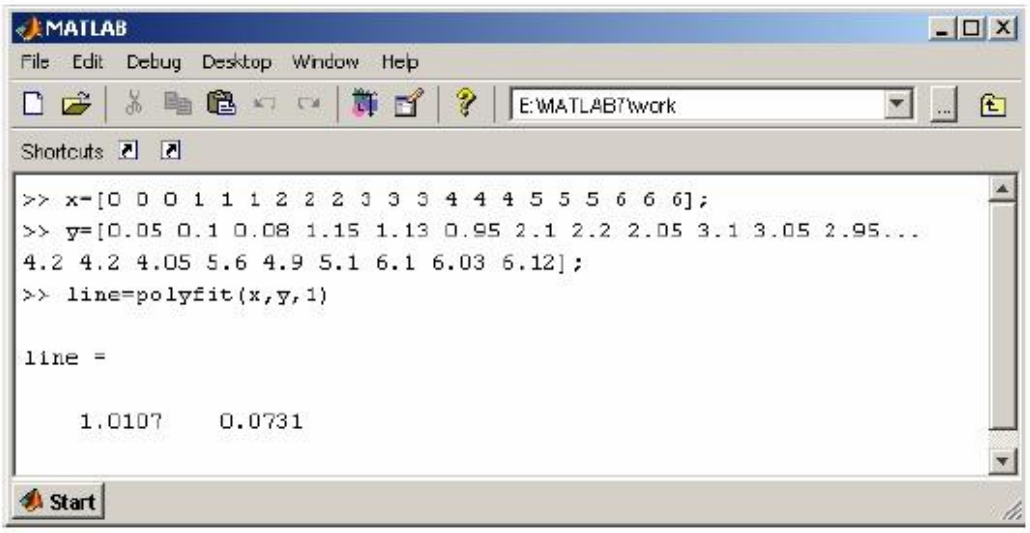
&

$$x= -0.5000 - 0.8660i$$

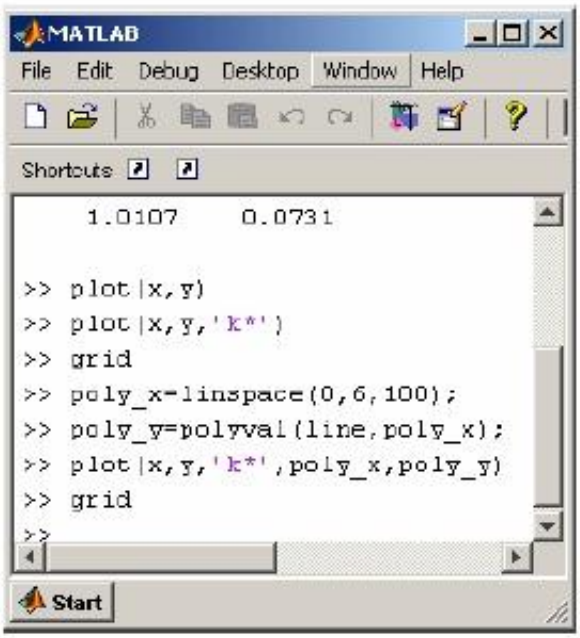
Polynomial Curve fitting

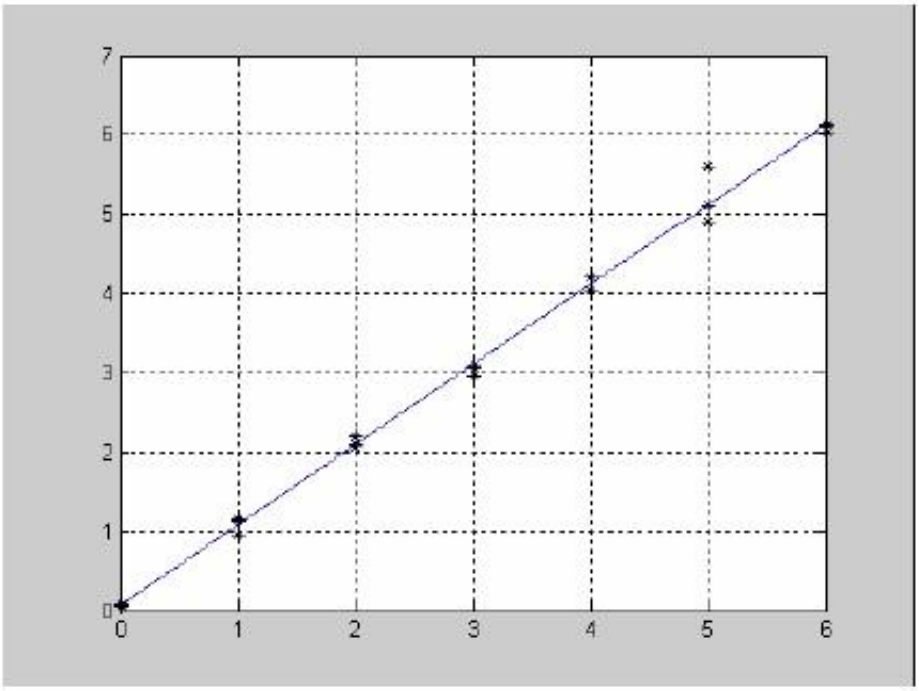
S.No.	Input (x)	Output (y)
1	0	0.05
2	0	0.1
3	0	0.08
4	1	1.15
5	1	1.13
6	1	.95
7	2	2.1
8	2	2.2
9	2	2.05
10	3	3.1
11	3	3.05
12	3	2.95
13	4	4.2
14	4	4.2
15	4	4.05
16	5	5.6
17	5	4.9
18	5	5.1
19	6	6.1
20	6	6.03
21	6	6.12

In many engineering applications we need to find a polynomial to represent the given data set. We can use this polynomial in modelling the system under consideration. The MATLAB function **polyfit**, fits an n^{th} order polynomial to a given data set.Consider the data set given on left. We will use polyfit to fit a curve to these data. The equation of the fitted curve will be verified graphically.



The equation representing the data is $y=1.0107x+0.0731$.To verify the equation with the given data set enter the following commands.

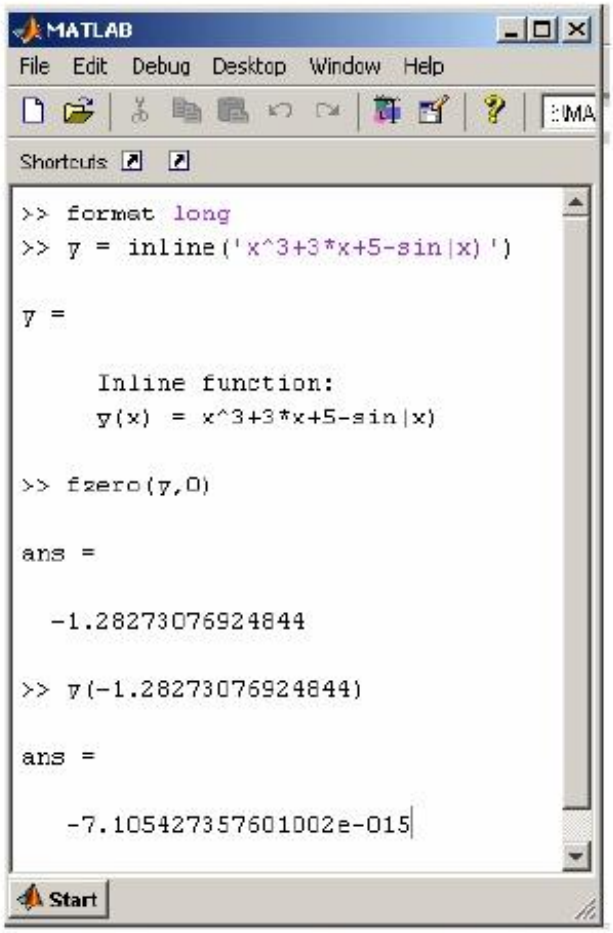

MATLAB
File Edit Debug Desktop Window Help
Shortcuts
1.0107 0.0731
>> plot(x,y)
>> plot(x,y,'k*')
>> grid
>> poly_x=linspace(0,6,100);
>> poly_y=polyval(line,poly_x);
>> plot(x,y,'k*',poly_x,poly_y)
>> grid
>>
Start


0
1
2
3
4
5
6
7

Roots of Equation

Consider the problem of solving $x^3+3x+5= \sin(x)$. We are required to know the value of x that makes this equation true. The equation can be re stated as $x^3+3x+5 -\sin(x)=0$.

The MATLAB function **fzero** finds the roots of an equation.. **There appears to be a root close to x=0.(How?)**



At x=-1.28273076924844, value of function y is such a small number that we can call it zero..

LAB # 12

Numerical Integration of a function

Numerical Integration is most useful for finding integrals of functions that can't be integrated mathematically.

Example-I: Integrate x^2+2x+1 from 0 to 1.

```
function y=NUMINT(x)
y=x.^2+2*x+1-log(x)
```

Or

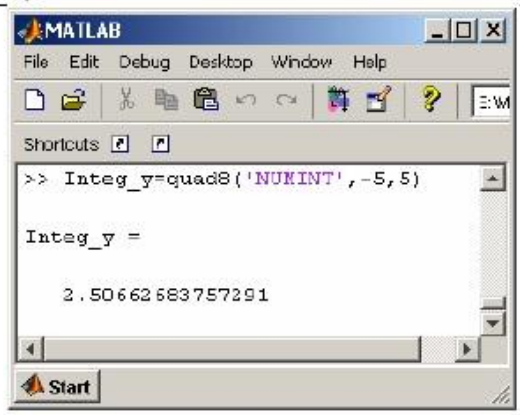
```
function y=NUMINT(x)
%y=x.^2+2*x+1-log(x)
y=x.*(x+2)+1
```

>> Integ_y=quad8('NUMINT',0,1)

```
y =
  Columns 1 through 2
   1.00000000000000   1.05793318891372
  Columns 3 through 4
   1.19192179527508   1.39021391475144
  Columns 5 through 6
   1.62917960675006   1.90939445477740
  Columns 7 through 8
   2.25000000000000   2.61854405376385
  Columns 9 through 10
   2.97082039324994   3.31577394178878
  Columns 11 through 12
   3.64141153805826   3.88658043600016
  Column 13
   4.00000000000000
y =
  Columns 1 through 2
   1.19192179527508   1.62917960675006
  Columns 3 through 4
   2.25000000000000   2.97082039324994
  Column 5
   3.64141153805826
Integ_y =
   2.33333333333333
```

Example-II: Integrate $e^{(-x^2/2)}$ from -5 to 5.

```
function y=NUMINT(x)
y=exp(-0.5*x.*x);
```

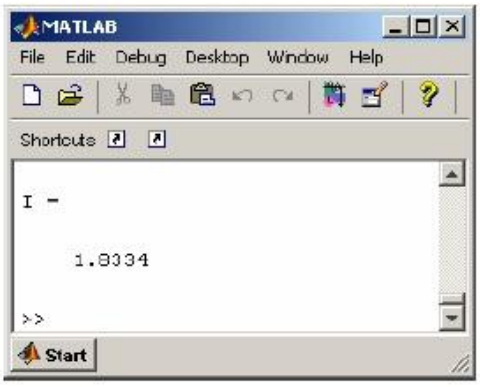


Trapz

Computes the integral of a function over a given domain using the trapezoidal method

Compute the integral of the function $f(x) = x^2 + x + 1$ for $0 \le x \le 1$ using the trapezoidal method. Compare the result to the exact analytical solution.

```
clear
close
x = linspace(0,1,100)

y = x .^2+x+1;
I = trapz(x,y)
```



Analytical result : **I** = 11/6 = 1.8333

LAB # 13

Solving Ordinary Differential Equations

In Matlab, there are a set of ODE solvers, namely **ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb.**General information regarding Matlab ODE solvers are given below:

ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a *one-step* solver – in computing y(t n), it needs only the solution at the immediately preceding time point, y(t n–1). In general, ode45 is the best function to apply as a "first try" for most problems.

ode23 is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness. Like ode45, ode23 is a one-step solver.

ode113 is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate. Ode113 is a *multistep* solver – it normally needs the solutions at several preceding time points to compute the current solution.

The above algorithms are intended to **solve non-stiff systems**. If they appear to be unduly slow, try using one of the **stiff solvers** below.

ode15s is a variable order solver based on the numerical differentiation formulas, NDFs. Optionally, it uses the backward differentiation formulas, BDFs (also known as Gear's method) that are usually less efficient. Like ode113, ode15s is a multistep solver. If you suspect that a problem is stiff or if ode45 has failed or was very inefficient, try ode15s.

ode23s is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than ode15s at crude tolerances. It can solve some kinds of stiff problems for which ode15s is not effective.

ode23t is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.

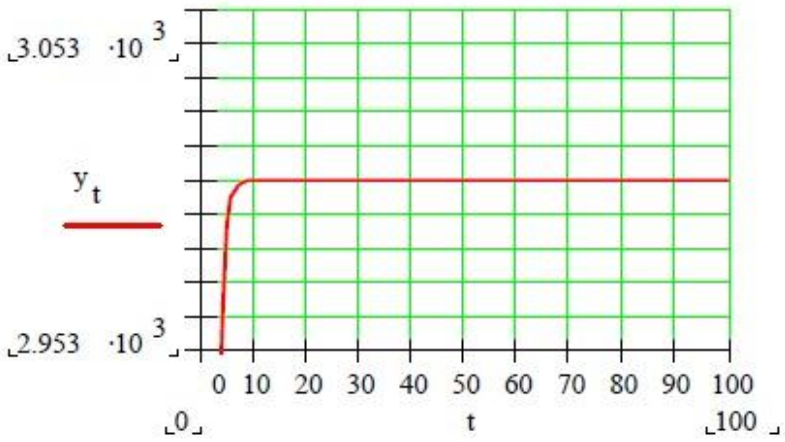
ode23tb is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like ode23s, this solver may be more efficient than ode15s at crude tolerances.

Stiffness is a special problem in the solution of ODEs. A stiff system is one which involves rapidly changing components together with slowly changing ones. The rapidly changing component is actually transient which dies out quickly, after which the system becomes dominated by the slowly varying components. Both individual and systems of ODEs can be stiff. Although the transient phenomena exist for only a short part of the integration interval, they can dictate the time step for the entire solution. An example of single stiff ODE is given below:

$$dy/dt = -1000y + 3000 - 2000e^{-t}$$

for y(0)=0 the analytical solution can be developed as

$$y = 3 - 0.998e^{-1000t} - 2.002e^{-t}$$



The above figure shows that the solution is initially dominated by the fast exponential term(e^{-1000t}). After a very short period t<0.005, this transient dies out and the solution becomes dictated by the slow exponential ($002e^{-t}$).

Syntax: **[T,Y] = ode45 ('Filename',tspan,y0)**

Where

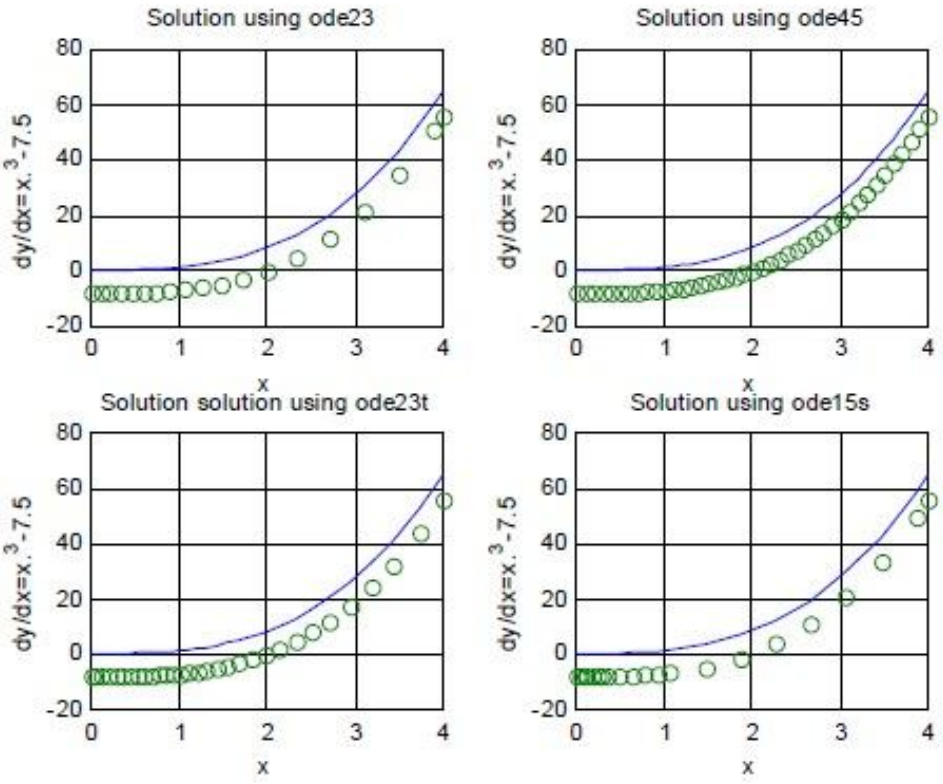
Filename	Name of the file containing the ODE(s) to be solved
tspan	A vector specifying the interval of integration [t0 tfinal]. To obtain solutions at specific times (all increasing or all decreasing), use tspan = [t0,t1, ..., tfinal].
y0	A vector of initial conditions.

RKCOMP.m

```
tspan=[0 4]
y0=0.5

[x,num_y]=ode23('g1',tspan,y0);
an1_y=x.^3-7.5;
subplot(2,2,1)
plot(x,num_y,x,an1_y,'o')
title('Solution using ode23 ')
xlabel('x'); ylabel('dy/dx=x.^3-7.5');
grid on;

[x,num_y]=ode45('g1',tspan,y0);
an1_y=x.^3-7.5;
subplot(2,2,2)
plot(x,num_y,x,an1_y,'o')
title('Solution using ode45');
xlabel('x'), ylabel('dy/dx=x.^3-7.5');
grid on;

[x,num_y]=ode23t('g1',tspan,y0);
an1_y=x.^3-7.5;
subplot(2,2,3)
plot(x,num_y,x,an1_y,'o')
title('Solution solution using ode23t')
xlabel('x'); ylabel('dy/dx=x.^3-7.5');
grid on;

[x,num_y]=ode15s('g1',tspan,y0);
an1_y=x.^3-7.5;
subplot(2,2,4)
plot(x,num_y,x,an1_y,'o')
title('Solution using ode15s')
xlabel('x')
ylabel('dy/dx=x.^3-7.5')
grid
```

g1.m

```
function dy = g1(x,y)
dy = 3*x^2;
```



Exercises:

Write Matlab programs to solve the following differential equations using 4th order Runge-Kutta method for the range and increment Δx shown. Also compare the accuracy of the solution by comparing the computed value with the values computed from the corresponding true solutions.

Problem	Differential equation	Boundary condition	Δx	Range	True solution
a.	$(1+x^2)\frac{dy}{dx} - xy = 0$	$y = 2 \ at \ x = 1$	0.25	$1 \le x \le 2$	$y = \sqrt{2(1+x^2)}$
b.	$x + y\frac{dy}{dx} - 2x^3 - 2xy^2 = 0$	$y = 1 \ at \ x = 0$	0.1	$0 \le x \le 1$	$\ln(x^2 + y^2) - 2x^2 = 0$

LAB # 14

MATLAB has the capability to manipulate expressions symbolically. There are tools to perform algebraic operations, differentiate and integrate functions, solve systems of equations, and solve ordinary differential equations. MATLAB adapted these tools from the software program Maple developed at the University of Waterloo, Canada.

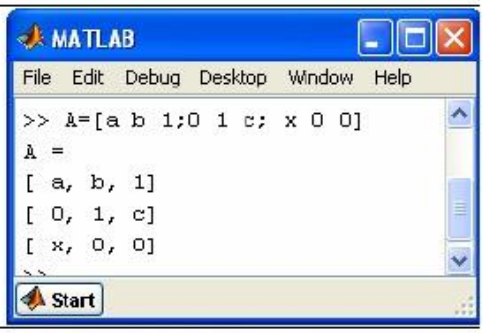
Creating symbolic expressions

Variables can be declared symbolic variables with the command for example we can declare variables x; y; z; a; b; c, etc. as symbolic variable using MATLAB **syms** command

```
» syms x y z a b c
```

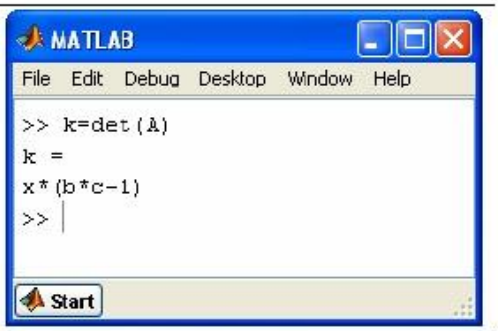
This command is a short cut for the more elaborate command sym('x', 'y','z', 'a', 'b', ' c') , or even more deliberately, x = sym('x'),y = sym('y'),etc .We can then define expressions using these variables and these expressions can be manipulated symbolically. For example a matrix A can be defined by

> A = [a b 1; 0 1 c; x 0 0]



Since A is a symbolic expression, we can calculate its determinant in terms of the variables a; b; c; x with the usual MATLAB command.

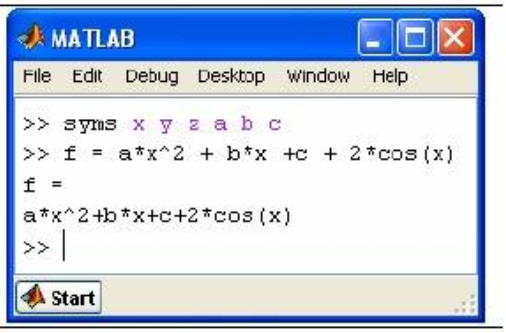
>> d = det(A)



Symbolic Functions

A function f(x) can be defined in terms of a symbolic expression.Suppose we want to express f(x)=ax2+bx+c+2cos(x) as a symbolic expression.

>> syms x y z a b c
>> f = a*x^2 + b*x +c + 2*cos(x)



Notice that we haven't used array operators in defining the function. Actually, the array operators .^, .*, ./ cannot be used in symbolic expressions because

symbolic expressions are not applied directly to vectors and matrices.

Evaluating symbolic expressions

Next, how do we specify the values of the parameters in the expression, and how do we evaluate the symbolically defined function at a point? This is done using the substitution command subs.

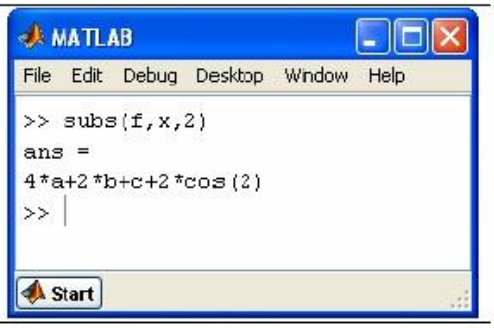
The syntax is **subs(F, Old, New)**

Where,

- **Old** is a symbolic variable, a string representing a variable name, or a string (quoted) expression.
- **New** is a symbolic or numeric variable or expression.
- **F** is the symbolic expression to be evaluated

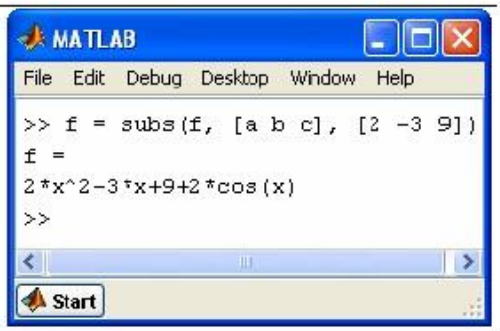
For example, if we wish to evaluate the function f at x = 2, we cannot execute f(2)to get f at x=2, as we did in case of using MATLAB calculation engine. To evaluate the symbolic function f defined above at x = 2, (leaving in the parameters a; b; c,) we will use MATLAB **subs** command from the Symbolic Toolbox:

```
>> subs(f,x,2)
```



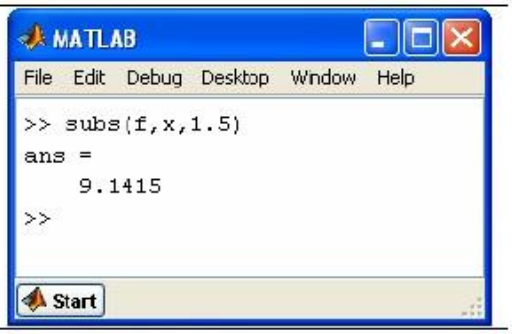
The result is still a symbolic expression. Now specify the values of the parameters a, b, c as a=2; b=3; c=9, and use subs command to evaluate f in terms of x only.

>> g = subs(f, [a b c], [2 -3 9])



We still have a symbolic expression depending on variable x. Now let us try to evaluate this function at a particular point, say x = 1.5. For this, we make another substitution **subs(g,x,-1.5)** with the answer 9.1415.

>> subs(f,x,-1.5)



The result is a real number now. Off course, many variations are possible.

Symbolic Summations

MATLAB symbolic engine is capable of doing symbolic summations, even those with infinitely many terms. The "**symsum**" command is used for symbolic summation. Consider the symbolic sum $\sum_{k=m}^{n} f(k)$, where f(k) is a function of an integer k. The syntax to calculate this using MATLAB Symbolic Toolbox is:

>>syms k
>>symsum(f,k,m,n).

Summation expression often involves factorial (!). Since (!) is also a numeric command, we need to tell MATLAB

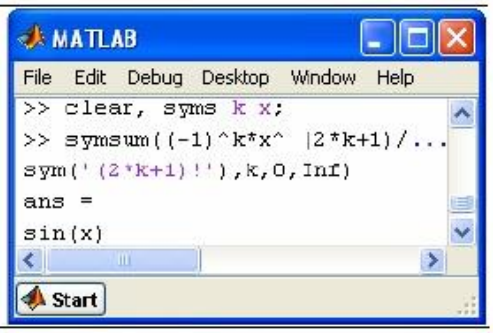
to use the symbolic form by including the term involving in the factorial. For example the expression (1+2*k)!, is translated as,

>>sym('(1+2*k)!')

Computing Infinite sums are particularly interesting in MATLAB symbolic toolbox. The following solutions for the three summation expression illustrate the necessary concepts of symbolic summation using MATLAB Symbolic Toolbox.

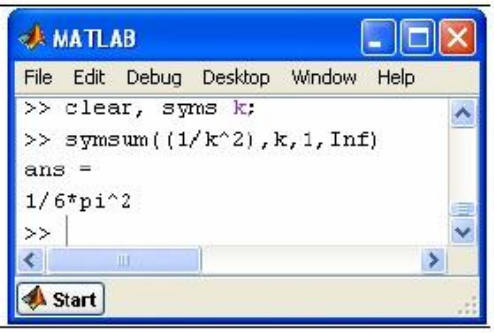
1. Evaluate $\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+2)!}$

```
>> clear, syms k x;
>>symsum((-1)^k*x^ (2*k+1)/...
sym('(2*k+1)!'),k,0,Inf)
```



2. Evaluate $\sum_{k=1}^{\infty}\frac{1}{k^2}$

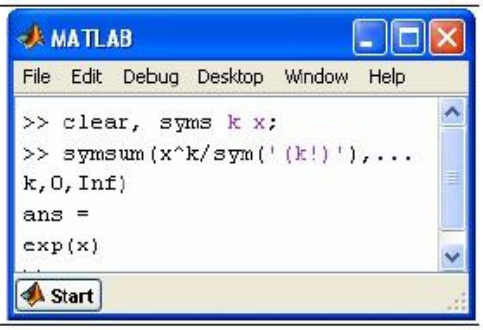
```
>> clear, syms k;
>> symsum((1/k^2),k,1,Inf)
```



3. Evaluate $\sum_{k=0}^{\infty}\frac{x^k}{k!}$

LAB # 15

```
>>clear, syms k x;
>>symsum(x^k/sym('(k!)'),k,0,I nf)
```



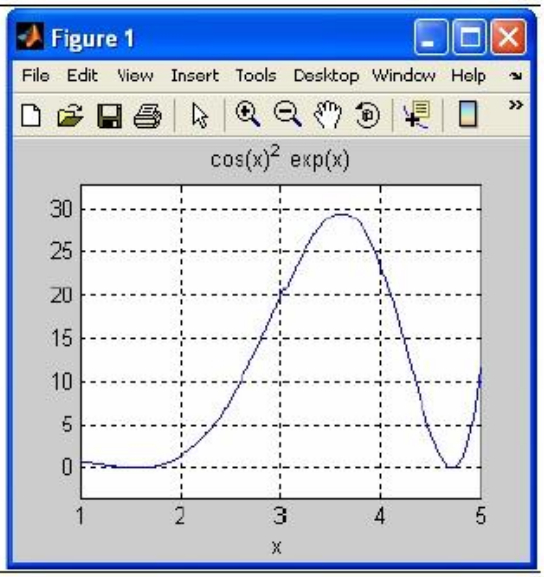
GRAPHING EQUATIONS AND FUNCTIONS

MATLAB symbolic engine has commands for plotting symbolic equations and functions. Most of these graphing commands start with "ez" pronounced as easy. These commands include ezplot, ezsurf and ezmesh for 2D and 3D plotting.

2D Plots

The command ezplot is used primarily to graph functions which are defined symbolically. If f is defined by a symbolic expression and we wish to graph it on the interval [1; 5], we can do it with the one command, ezplot(f, [1,5]). For example

```
>> syms x
>> f = cos(x)^2*exp(x)
>> ezplot(f, [1,5])
>>grid on
```



As another example consider the equation for a circle of radius 2, i.e. $x^2+y^2 = 4$.

Step-1: Move everything to the left-hand side of the equation, i.e. $x^2 + y^2 - 4 = 0$

Step-2: Clear the workspace and define x and y as symbolic.

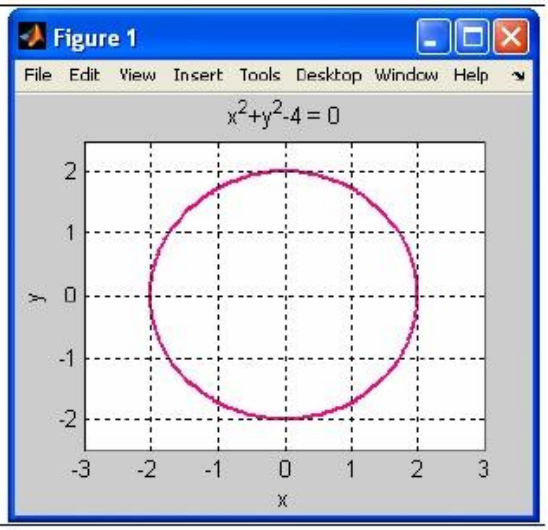
```
clear; syms x y;
```

Step-3: Define a variable equal to the left-hand ide of the equation, inside single quotation marks.

```
LHS = 'x^2+y^2 - 4';
```

Step-4: Use ezplot. **ezplot(LHS,[-3,3,-2.5,2.5])**

```
>> clear;
>>syms x y;
>>LHS = 'x^2+y^2 - 4';
>>ezplot(LHS)
>>axis([-3,3,-2.5,2.5])
>> grid on
```



The [-3,3,-2,2] tells MATLAB that the x scale is to go from -3 to +3, and the y scale is to go from -2.5 to +2.5. Note that we have assigned $x^2 + y^2 - 4$ to the character variable 'LHS' (see the Workspace window or type >> whos).

You can edit the figure in the Figure window. First save it as "circle.fig" using File/Save As so you can retrieve it later from the Command window, without having to recreate it. Here are some things you can do in the Figure window:

- Click on the arrow icon and then double-click on any portion of the figure you'd like to edit. In this way

you can change the color and thickness of the curve, add points, change the axes, edit the title.

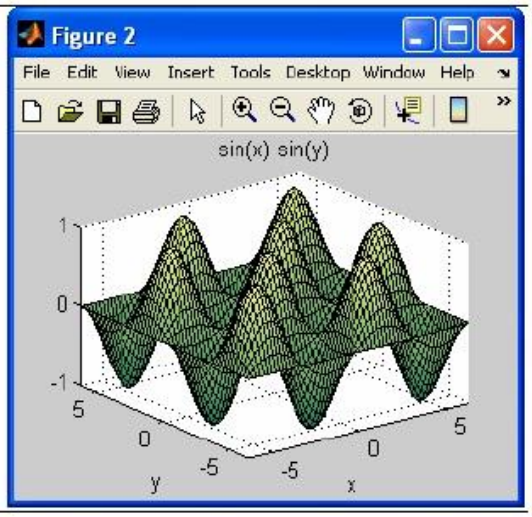
- Edit / Copy Figure to place it on the Windows Clipboard. Then you can paste it into a Word document, for example.
- While in the Figure window, press the Alt and Print Screen keys at the same time, to place the entire figure window on the clipboard. This can also be pasted into a Word document.

3D Plots

Three-dimensional plots can be created using the **ezmesh** and **ezsurf** commands. For these commands an explicit form must be used, i.e. z = f(x, y).

It is the **f(x, y)** that is entered into the command. The code below illustrates some possibilities.

```
>>clear
>>syms V x y
>>V=sin(x)*sin(y);
>>figure(2);
>>ezsurf(V);
>>colormap summer
```



In this example the function $V(x,y)=\sin(x)\cos(x)$ is plotted.

The 1st line clear variables and functions from memory.

The 2nd line defines the variables V, x, y as symbolic variables.

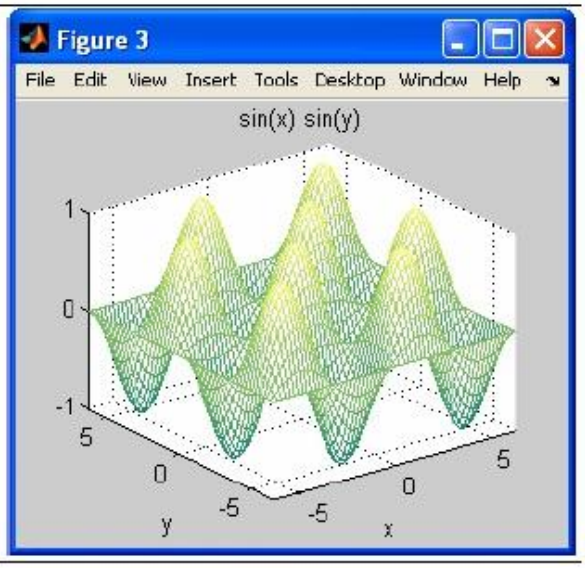
The 3rd line creates a blank figure window titled Figure 2

The 4th line is a symbolic implementation of MATLAB **surf** command. The MATLAB Surf command plots the colored parametric surface of the function V. Surf(V) uses x = 1:n and y = 1:m. The height, V, defined over a geometrically rectangular grid of x and y.

The 5th line selects and implements a colormap scheme for the plotted figure. In this case the selected colormap scheme is summer.

MATLAB provides many other colormap schemes like jet, bone, colorcube, cool, copper, flag, grey, hot, hsv, lines, pink, prism, spring, white and winter.

```
>>clear
>>syms V x y
>>V=sin(x)*sin(y)
>>figure(3);
>>ezmesh(V);
>>colormap summer
```



In this example the function $V(x,y)=\sin(x)\cos(x)$ is plotted once again. However this time we used **ezmesh** instead of **ezsurf** command. Ezmesh is a symbolic implementation of MATLAB mesh command. It gives 3D-mesh surfaces.

Note the difference in results produced by ezsurf and ezmesh commands.

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Information Technology
MS-471 – Applied Numerical Methods
Lab # 1 (Spring 2011)

Name: _____ Roll #: _____

Due on: _____

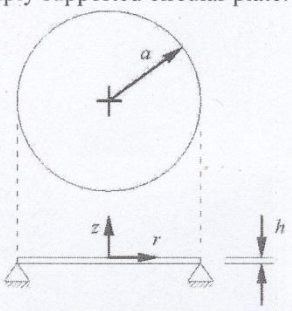
1. Plot sinθ versus θ and cosθ versus θ for 60 points in the interval $0 \le \theta \le 2\pi$. For sinθ curve connect the points with a dashed line and label the points with circles. For cosθ curve connect the points with a firm line and label the points with diamonds. Label the axes. Plot the two curves i) in the same plot, and ii) as different plots in the same figure window.

2. Write a function to plot the displacement of a cantilevered beam under a uniform load. In addition to creating the plot, the function should return the maximum deflection and angle between the horizontal and the surface of the beam at its tip. The formulas for the displacement y and tip angle θ are

$$y = -\frac{wx^2}{24EI}(6L^2 - 4Lx + x^2) \qquad \text{and} \qquad \theta = \frac{1}{6}\frac{wL^3}{EI}$$

where w is the load per unit length, E is Young 's modulus for the beam, I is the moment of inertia of the beam, and L is the length of the beam.

3. The sketch below depicts a simply supported circular plate.



When the plate is exposed to a uniform pressure load, p, in the z-direction, the z-direction displacement of the plate as a function of radial position is

$$w = \frac{pa^4}{64D}\left[1 - \left(\frac{r}{a}\right)^2\right]\left[\frac{5+\nu}{1+\nu} - \left(\frac{r}{a}\right)^2\right]$$

where a is the plate radius, D is the flexural rigidity,

$$D = \frac{Eh^3}{12(1-\nu^2)}$$

h is the plate thickness, E is Young's modulus, and ν is Poisson's ratio for the plate material. Write an m-file function to evaluate $w(r, p, a, h, E, \nu)$. Test your function by evaluating and plotting $w(r)$ for a steel plate (ν = 0.3, E = 200 GPa) of radius a = 10cm and thickness h = 0.05 cm under an applied pressure of 300 kPa.

4. Sutherland's viscosity law is used to represent the variation of gas viscosity with temperature,

$$\frac{\mu}{\mu_o} = \frac{(T/T_o)^{3/2}(T_o S)}{T+S}$$

where μ is the dynamic viscosity, T is the absolute temperature, μ_o and T_o are reference values of μ and T, respectively, and S is a constant (with units of temperature)that depends on the gas. For air, $T_o = 273$ K, $\mu_o = 1.71 \times 10^{-5}$ kg / (m ·s), and S =110 .4 K
Write a script m-file that evaluates μ_{air} at 30°C. In your expression for μ, use variables, not the numerical constants, for μ_o, T_o, and S.

5. Convert the script written in the preceding exercise to a function. The function should have one input parameter, T, and one output parameter, μ.

6. Modify the function from the preceding exercise so that it accepts a vector of T values as well as a scalar T values.

7. Write a function that evaluates μ_{air} in the range $0 \leq T \leq 230^{\circ}$ C using the function written in the preceding exercise. Plot the Sutherland model data against the shown below.

```
  0     1.720e-5
 20     1.817e-5
 40     1.911e-5
 60     2.002e-5
 80     2.091e-5
100     2.177e-5
127     2.294e-5
177     2.493e-5
227     2.701e-5
```