

**Computational Engineering LAB**

**Laboratory Manual**

(B.Tech. Mechanical Engineering)

School of Technology

Pandit Deendayal Energy University

Gandhinagar Gujarat

**INDEX**

Contents

[**LAB 1:** Introduction to numerical methods 3](#_Toc123640329)

[**LAB 2:** Basics of computational programming and software 7](#_Toc123640330)

**1. Introduction**

**2. Basics of computational programming and software**

**3. Systems of Linear Algebraic Equations**

**4. Eigenvalue problems**

**5. Non-linear Equations**

**6. Polynomial Approximation**

**7. Polynomial Interpolation**

**8. Numerical Differentiation**

**9. Numerical Interpolation**

**10. Ordinary Differential Equations**

**11. Introduction of Partial Differential Equations**

**12. Case Study - I**

**13. Case Study - II**

# **LAB 1:** Introduction to numerical methods

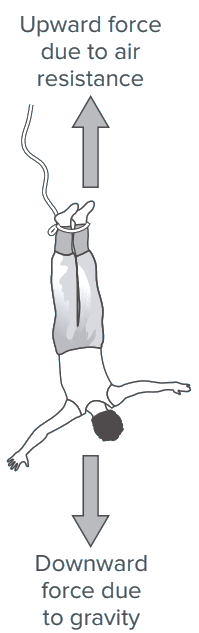
**Objective:** To understand basics of numerical methods for engineers.

**Theory:**

**Numerical methods** are techniques by which mathematical problems are formulated so that they can be solved with arithmetic and logical operations. The implementation of a numerical method with an appropriate convergence check in a programming language is called a numerical algorithm.

In the pre–computer era, the time and work of implementing such calculations seriously limited their practical use. However, with the advent of fast, inexpensive digital computers, the role of numerical methods in engineering and scientific problem solving has exploded.

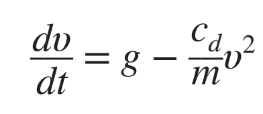
Example – Suppose a bungee-jumper is falling freely as shown in the figure-

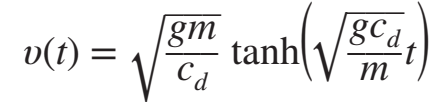
The governing equation based on newton’s laws are given as below,

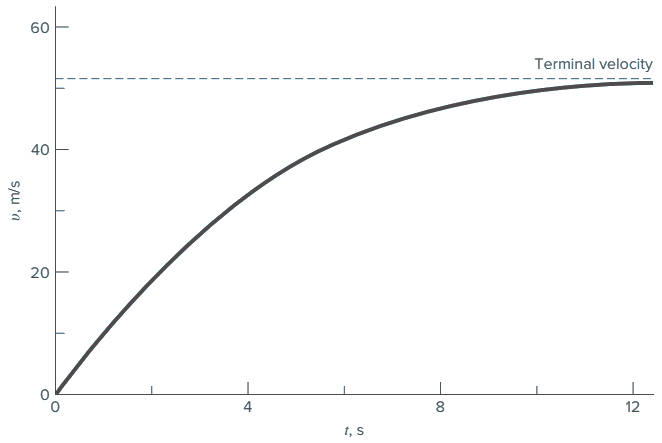
And *net F = mg-Fdrag*

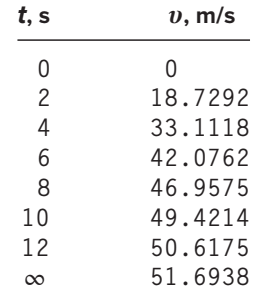
*= mg- Fdrag*

Where, v = velocity of falling man, g is acceleration due to gravity, cd = drag coefficient, m = mass of the bungee jumper, t = time.



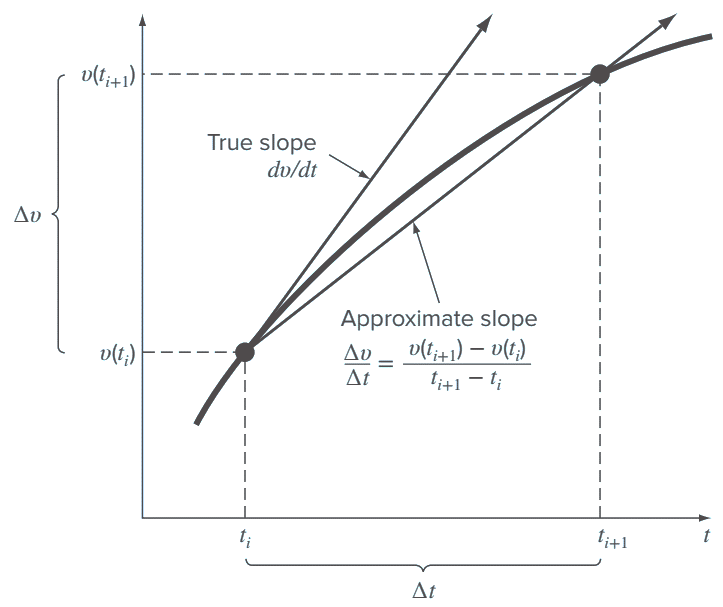
if the jumper is initially at rest (υ = 0 at t = 0), calculus can be used to solve Equation with varying t values.

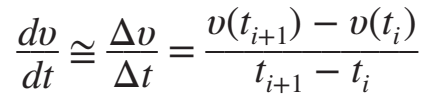


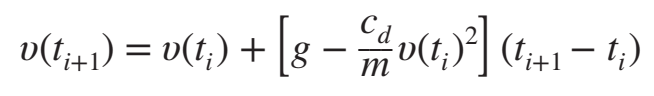


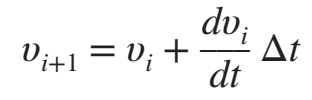
Plot the Solution 🡪

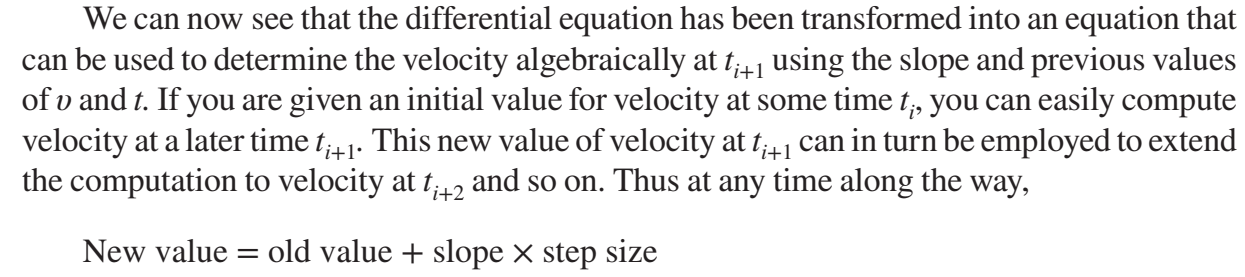
Alternatively, we can solve above problem numerically. The use of a finite difference to approximate the first derivative of υ with respect to t.



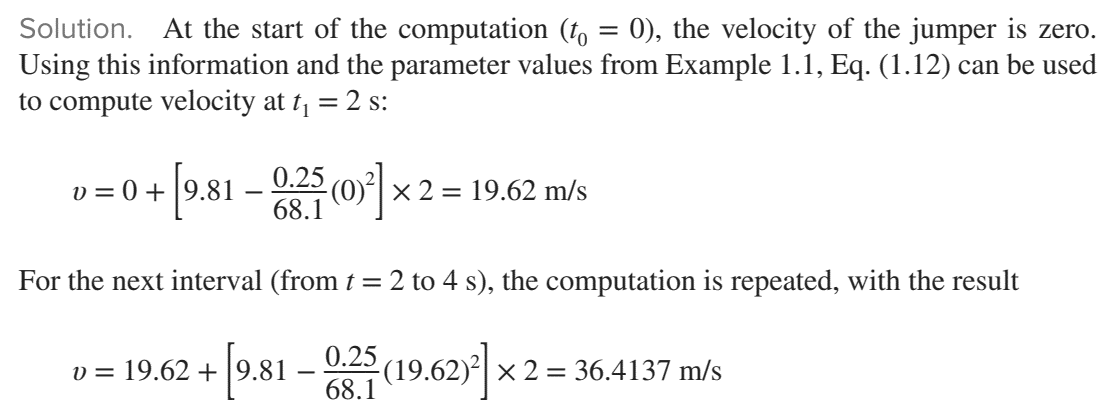


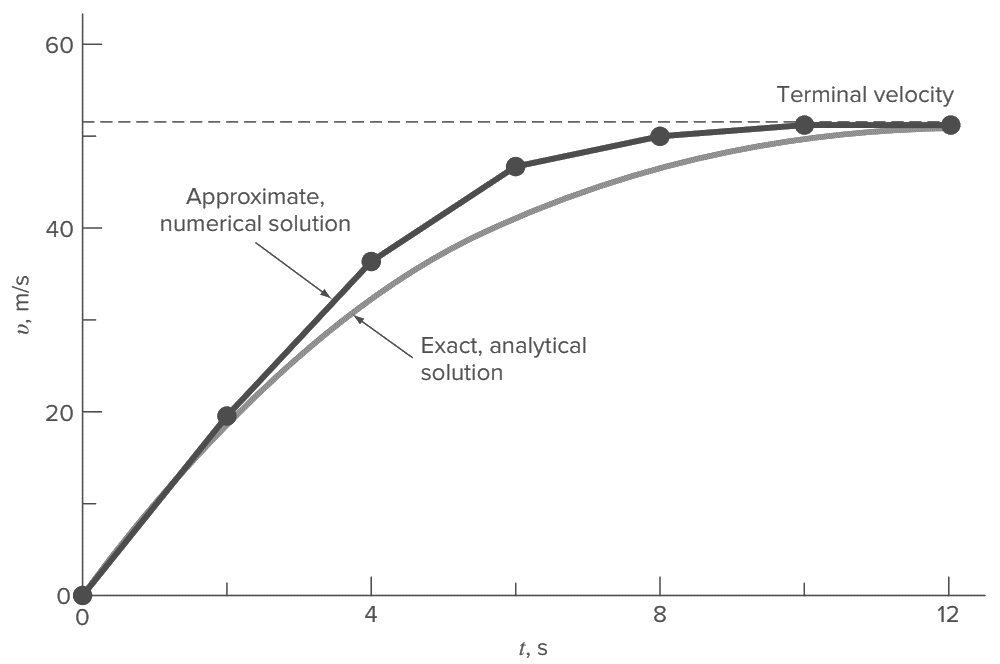




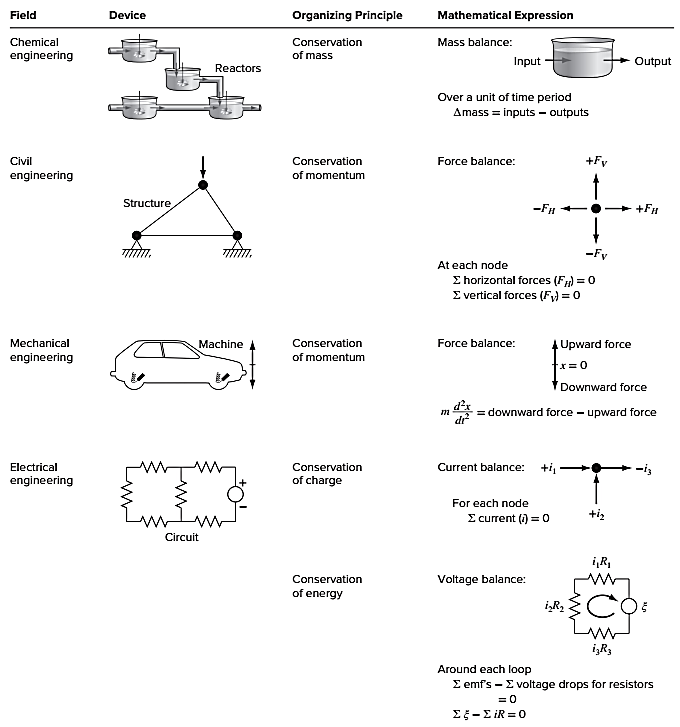


This approach is formally called **Euler’s method**. Employ a step size of 2 s for the calculation, and initial condition as (υ = 0 at t = 0).

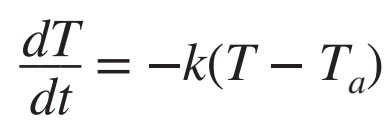




Few other real physical problems,



**Assignment 1** - Newton’s law of cooling says that the temperature of a body changes at a rate proportional to the difference between its temperature and that of the surrounding medium (the ambient temperature),



where T = the temperature of the body (°C), t = time (min), k = the proportionality constant (per minute), and Ta = the ambient temperature (°C).

Suppose that a cup of coffee originally has a temperature of 70 °C. Use Euler’s method to compute the temperature from t = 0 to 20 min using a step size of 2 min if T a = 20 °C and k = 0.019/min.

# **LAB 2:** Basics of computational programming and software

**Objective:** Understand the basics of computational programming and software.

**Theory:** The primary objective is to provide basic overview of MATLAB software.

MATLAB (an abbreviation of "**MAT**rix **LAB**oratory") is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Uses of MATLAB software −

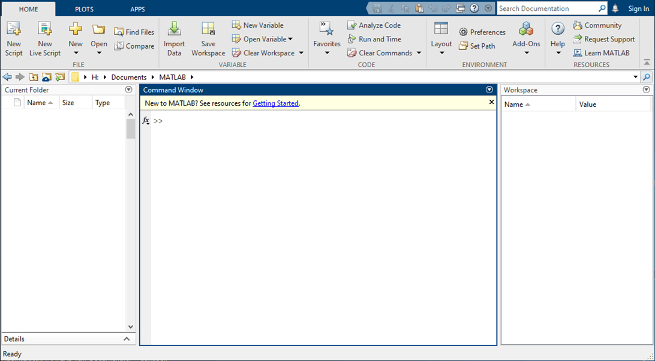
* Dealing with Matrices and Arrays
* 2-D and 3-D Plotting and graphics
* Linear Algebra
* Algebraic Equations
* Non-linear Functions
* Statistics
* Data Analysis
* Calculus and Differential Equations
* Numerical Calculations
* Integration
* Transforms
* Curve Fitting
* Various other special functions

Application of MATLAB

MATLAB is widely used as a computational tool in science and engineering encompassing the fields of all engineering streams−

* Signal Processing and Communications
* Image and Video Processing
* Control Systems
* Test and Measurement
* Computational fluid dynamics
* Computational Biology, finance
* Solid mechanics, Finite element method

When you start MATLAB®, the desktop appears in its default layout.



The desktop includes these panels:

**Current Folder** — Access your files.

**Command Window** — Enter commands at the command line, indicated by the prompt (>>).

**Workspace** — Explore data that you create or import from files.

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

a = 1

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

a =

1

Create a few more variables.

b = 2

b =

2

c = a + b

c =

3

d = cos(a)

d =

0.5403

Note that the argument in the trigonometric functions should be in radians !!

Example – sin(45 degree) must be converted to🡪 sin (45\*pi/180 radians).

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

sin(a)

ans =

0.8415

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

e = a\*b;

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

**clc-** Clear Command

Syntax clc

Description - clc clears all the text from the Command Window, resulting in a clear screen.

**clear -** Remove items from workspace, freeing up system memory.

clear removes all variables from the current workspace, releasing them from system memory.

**close** - Close one or more figure.

Syntax - close

close(fig), close all

Description - close - closes the current figure.

Commonly used Operators and Special Characters

MATLAB supports the following commonly used operators and special characters −

|  |  |
| --- | --- |
| **Operator** | **Purpose** |
| **+** | Plus; addition operator. |
| **-** | Minus; subtraction operator. |
| **\*** | Scalar and matrix multiplication operator. |
| **.\*** | Array multiplication operator. |
| **^** | Scalar and matrix exponentiation operator. |
| **.^** | Array exponentiation operator. |
| **\** | Left-division operator. |
| **/** | Right-division operator. |
| **.\** | Array left-division operator. |
| **./** | Array right-division operator. |
| **:** | Colon; generates regularly spaced elements and represents an entire row or column. |
| **( )** | Parentheses; encloses function arguments and array indices; overrides precedence. |
| **[ ]** | Brackets; enclosures array elements. |
| **.** | Decimal point. |
| **…** | Ellipsis; line-continuation operator |
| **,** | Comma; separates statements and elements in a row |
| **;** | Semicolon; separates columns and suppresses display. |
| **%** | Percent sign; designates a comment and specifies formatting. |
| **\_** | Quote sign and transpose operator. |
| **.\_** | Nonconjugated transpose operator. |
| **=** | Assignment operator. |

Special Variables and Constants

MATLAB supports the following special variables and constants −

|  |  |
| --- | --- |
| **Name** | **Meaning** |
| **ans** | Most recent answer. |
| **eps** | Accuracy of floating-point precision. |
| **i,j** | The imaginary unit √-1. |
| **Inf** | Infinity. |
| **NaN** | Undefined numerical result (not a number). |
| **pi** | The number π |

To plot the graph of a function, you need to take the following steps −

Define **x**, by specifying the **range of values** for the variable **x**, for which the function is to be plotted

Define the function, **y = f(x)**

Call the **plot** command, as **plot(x, y)**

Following example would demonstrate the concept. Let us plot the simple function **y = x** for the range of values for x from 0 to 100, with an increment of 5.

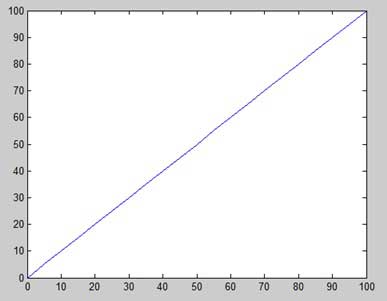
Create a script file and type the following code −

x = [0:5:100];

y = x;

plot(x, y)

When you run the file, MATLAB displays the following plot −



Let us take one more example to plot the function y = x2. In this example, we will draw two graphs with the same function, but in second time, we will reduce the value of increment. Please note that as we decrease the increment, the graph becomes smoother.

Create a script file and type the following code −

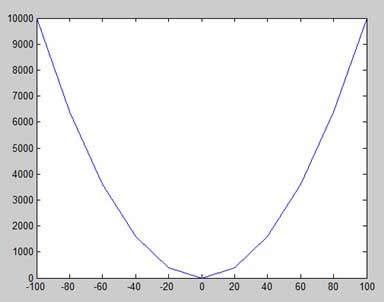
x = [1 2 3 4 5 6 7 8 9 10];

x = [-100:20:100];

y = x.^2;

plot(x, y)

When you run the file, MATLAB displays the following plot −



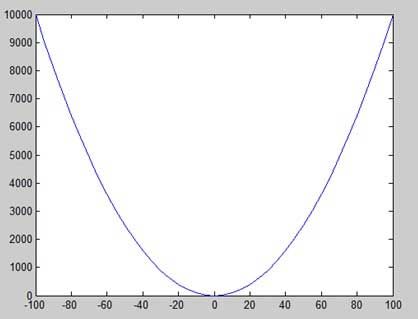
Change the code file a little, reduce the increment to 5 −

x = [-100:5:100];

y = x.^2;

plot(x, y)

MATLAB draws a smoother graph −



Adding Title, Labels, Grid Lines and Scaling on the Graph

MATLAB allows you to add title, labels along the x-axis and y-axis, grid lines and also to adjust the axes to spruce up the graph.

The **xlabel** and **ylabel** commands generate labels along x-axis and y-axis.

The **title** command allows you to put a title on the graph.

The **grid on** command allows you to put the grid lines on the graph.

The **axis equal** command allows generating the plot with the same scale factors and the spaces on both axes.

The **axis square** command generates a square plot.

Example

Create a script file and type the following code −

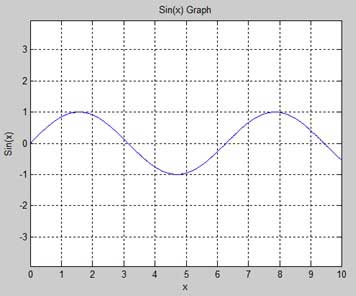
x = [0:0.01:10];

y = sin(x);

plot(x, y), xlabel('x'), ylabel('Sin(x)'), title('Sin(x) Graph'),

grid on, axis equal

MATLAB generates the following graph −



Drawing Multiple Functions on the Same Graph

You can draw multiple graphs on the same plot. The following example demonstrates the concept −

Example

Create a script file and type the following code −

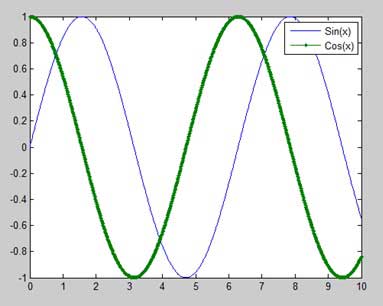
x = [0 : 0.01: 10];

y = sin(x);

g = cos(x);

plot(x, y, x, g, '.-'), legend('Sin(x)', 'Cos(x)')

MATLAB generates the following graph −



2-D and 3-D Plots

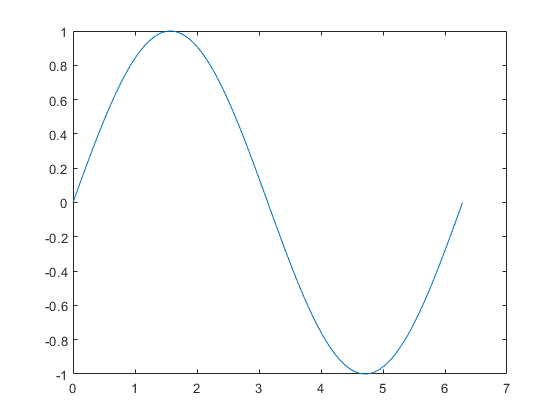
**Line Plots**

To create two-dimensional line plots, use the plot function. For example, plot the sine function over a linearly spaced vector of values from 0 to 2*π*:

x = linspace(0,2\*pi);

y = sin(x);

plot(x,y)

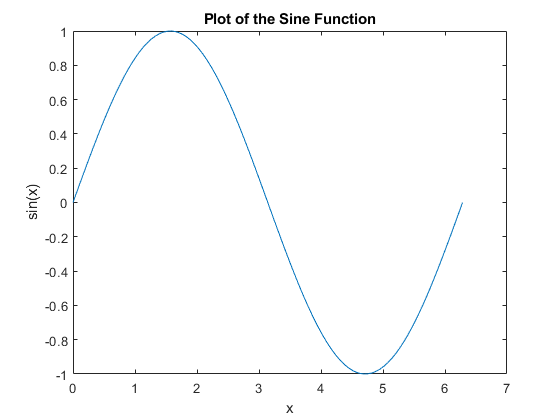


You can label the axes and add a title.

xlabel("x")

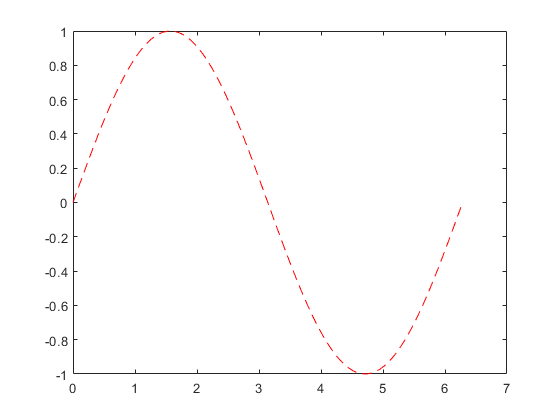
ylabel("sin(x)")

title("Plot of the Sine Function")



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

plot(x,y,"r--")



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

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

To add plots to an existing figure, use hold on. Until you use hold off or close the window, all plots appear in the current figure window.

x = linspace(0,2\*pi);

y = sin(x);

plot(x,y)

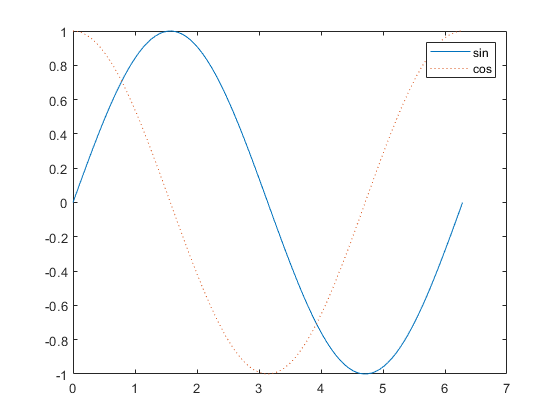
hold on

y2 = cos(x);

plot(x,y2,":")

legend("sin","cos")

hold off



**3-D Plots**

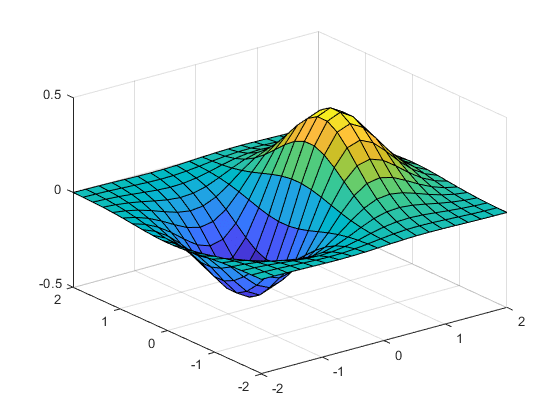
Three-dimensional plots typically display a surface defined by a function in two variables, *z*=*f*(*x*,*y*). For instance, calculate *z*=*xe*−*x*2−*y*2 given row and column vectors x and y with 20 points each in the range [-2,2].

x = linspace(-2,2,20);

y = x';

z = x .\* exp(-x.^2 - y.^2);

Then, create a surface plot as - surf(x,y,z)



**for**

for loop to repeat specified number of times

Syntax

for *index* = *values*

*statement*

*end*

Description

for *index* = *values*, *statements*, end executes a group of statements in a loop for a specified number of times. *values* has one of the following forms:

*initVal*:*endVal* — Increment the *index* variable from *initVal* to *endVal* by 1, and repeat execution of *statements* until *index* is greater than *endVal*.

*initVal*:*step*:*endVal* — Increment *index* by the value *step* on each iteration, or decrements *index* when *step* is negative.

Example –Create an identity matrix using FOR loop command.

A = zeros(4,4);

for i = 1:4

A(i,i) = 1;

end

**while**

while loop to repeat when condition is true

Syntax

while *expression*

*statements*

end

Description

while *expression*, *statements*, end evaluates an [expression](https://in.mathworks.com/help/matlab/ref/while.html#bub68r7-4), and repeats the execution of a group of statements in a loop while the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example - Use a while loop to calculate factorial(10).

n = 10;

f = n;

while n > 1

n = n-1;

f = f\*n;

end

disp(['n! = ' num2str(f)])

n! = 3628800

**if, elseif, else**

Execute statements if condition is true

Syntax

if *expression*

*statements*

elseif *expression*

*statements*

else

*statements*

end

Description

if *expression*, *statements*, end evaluates an [expression](https://in.mathworks.com/help/matlab/ref/if.html#bt_csfy), and executes a group of statements when the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

The elseif and else blocks are optional. The statements execute only if previous expressions in the if...end block are false. An if block can include multiple elseif blocks.

Examples

Use if, elseif, and else for Conditional Assignment

Create a tridiagonal matrix using if else assignment.

First create a matrix of 1s.

nrows = 4;

ncols = 6;

A = ones(nrows,ncols);

Loop through the matrix and assign each element a new value. Assign 2 on the main diagonal, -1 on the adjacent diagonals, and 0 everywhere else.

for c = 1:ncols

for r = 1:nrows

if r == c

A(r,c) = 2;

elseif abs(r-c) == 1

A(r,c) = -1;

else

A(r,c) = 0;

end

end

end

A

A = 4×6

2 -1 0 0 0 0

-1 2 -1 0 0 0

0 -1 2 -1 0 0

0 0 -1 2 -1 0

Arrays in MATLAB

In this section, we will discuss some functions that create some special arrays. For all these functions, a single argument creates a square array, double arguments create rectangular array.

The **zeros()** function creates an array of all zeros −

For example −

zeros(5)

MATLAB will execute the above statement and return the following result −

ans =

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

The **ones()** function creates an array of all ones −

For example −

ones(4,3)

MATLAB will execute the above statement and return the following result −

ans =

1 1 1

1 1 1

1 1 1

1 1 1

The **eye()** function creates an identity matrix.

For example −

eye(4)

MATLAB will execute the above statement and return the following result −

ans =

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

The **rand()** function creates an array of uniformly distributed random numbers on (0,1) −

For example −

rand(3, 5)

MATLAB will execute the above statement and return the following result −

ans =

0.8147 0.9134 0.2785 0.9649 0.9572

0.9058 0.6324 0.5469 0.1576 0.4854

0.1270 0.0975 0.9575 0.9706 0.8003

**Multidimensional Arrays**

An array having more than two dimensions is called a multidimensional array in MATLAB. Multidimensional arrays in MATLAB are an extension of the normal two-dimensional matrix.

Generally to generate a multidimensional array, we first create a two-dimensional array and extend it.

For example, let's create a two-dimensional array a.

a = [7 9 5; 6 1 9; 4 3 2]

MATLAB will execute the above statement and return the following result −

a =

7 9 5

6 1 9

4 3 2

The array *a* is a 3-by-3 array; we can add a third dimension to *a*, by providing the values like −

a(:, :, 2)= [ 1 2 3; 4 5 6; 7 8 9]

MATLAB will execute the above statement and return the following result −

a =

ans(:,:,1) =

0 0 0

0 0 0

0 0 0

ans(:,:,2) =

1 2 3

4 5 6

7 8 9

**Assignment 2:**

1. Draw the functions

on same plot for the range of x from -5 to +5. Also label the plots aesthetically.

1. Make a MATLAB program that sums number from 1 to 100 using FOR loop statement.
2. Write a MATLAB code to plot ‘tan’ function with x domain as -2𝜋 to 2𝜋 with 150 number of points. (Hint – when you plot, you will notice the plot is not aesthetic. To make your plot aesthetic, use – xlim and ylim functions.

Example - xlim([-2\*pi 2\*pi]); ylim([-10 10]);

# **LAB 3:** Systems of Linear Algebraic Equations

**Objective:** Understand the basics of computational programming and software.

**Theory:**