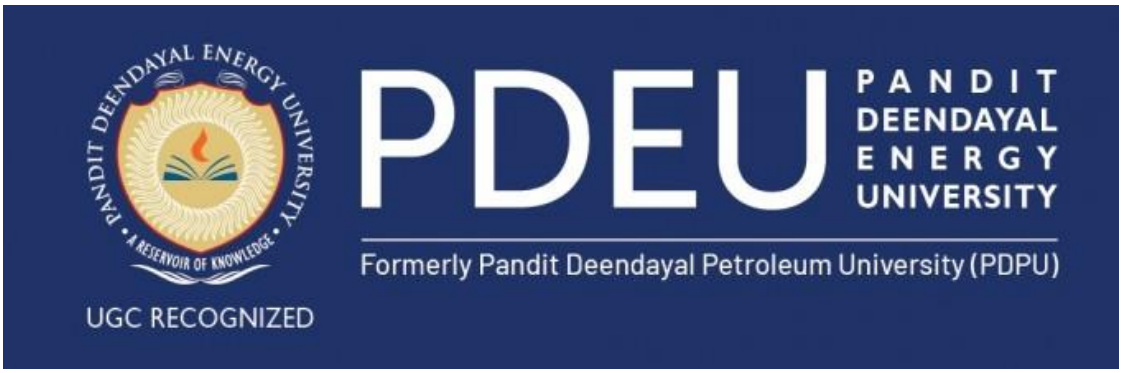


Computational Engineering LAB
Laboratory Manual

(B.Tech. Mechanical Engineering)

School of Technology

Pandit Deendayal Energy University

Gandhinagar Gujarat

INDEX

Contents

LAB 1: Introduction to Numerical Methods

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,

$$F = ma = m\frac{dv}{dt}$$

And *net* $F = mg\text{-}F_{drag}$

$$m\frac{dv}{dt} = mg - F_{drag}$$

$$\frac{dv}{dt} = g - \frac{F_{drag}}{m} = g - \frac{C_d V^2}{m}$$

Where, v = velocity of falling man, g is acceleration due to gravity = 9.81, cd = drag coefficient = 0.25, m = mass of the bungee jumper = 68.1 kg, t = time.

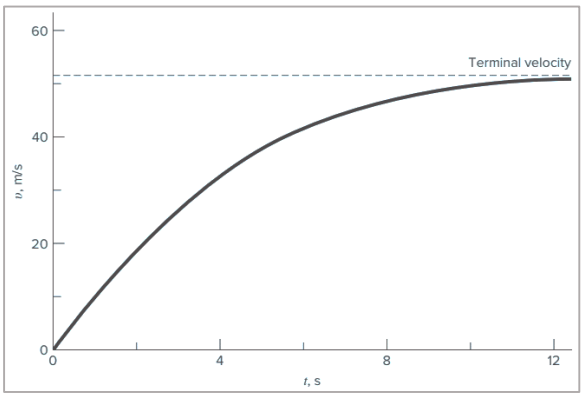
$$\frac{dv}{dt} = g - \frac{c_d}{m}v^2$$

if the jumper is initially at rest (υ = 0 at t = 0), calculus can be used to solve Equation with varying t values. (for solution-https://www.youtube.com/watch?v=VxNCIut1huw)

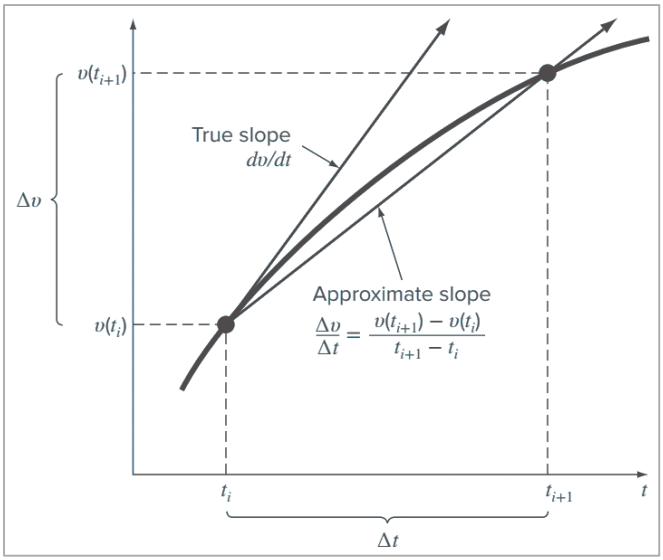
$$v(t) = \sqrt{\frac{gm}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

t, s	v, m/s
0	0
2	18.7292
4	33.1118
6	42.0762
8	46.9575
10	49.4214
12	50.6175
∞	51.6938

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.



$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}$$

$$v(t_{i+1}) = v(t_i) + \left[g - \frac{c_d}{m} v(t_i)^2\right](t_{i+1} - t_i)$$

$$v_{i+1} = v_i + \frac{dv_i}{dt}\Delta t$$

We can now see that the differential equation has been transformed into an equation that can be used to determine the velocity algebraically at t_{i+1} using the slope and previous values of v and t. If you are given an initial value for velocity at some time t_i, you can easily compute velocity at a later time t_{i+1}. This new value of velocity at t_{i+1} can in turn be employed to extend the computation to velocity at t_{i+2} and so on. Thus at any time along the way,

New value = old value + slope × step size

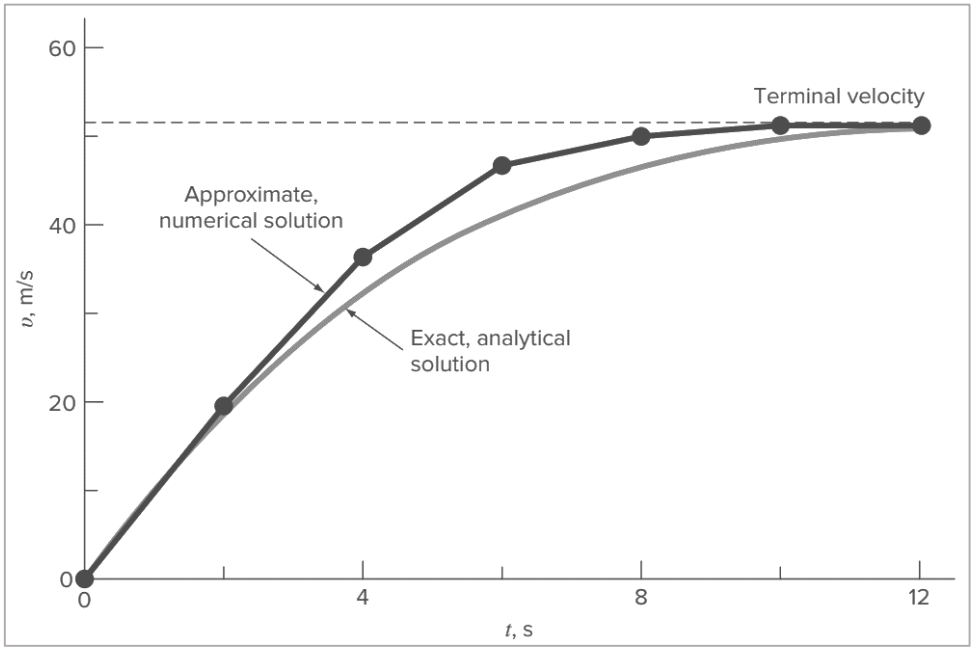
This approach is formally called **Euler's method**. Employ a step size of 2 s for the calculation, and initial condition as (υ = 0 at t = 0).

Solution. At the start of the computation ($t_0 = 0$), the velocity of the jumper is zero. Using this information and the parameter values from Example 1.1, Eq. (1.12) can be used to compute velocity at $t_1 = 2$ s:

$$v = 0 + \left[9.81 - \frac{0.25}{68.1}(0)^2\right] \times 2 = 19.62 \text{ m/s}$$

For the next interval (from $t = 2$ to 4 s), the computation is repeated, with the result

$$v = 19.62 + \left[9.81 - \frac{0.25}{68.1}(19.62)^2\right] \times 2 = 36.4137 \text{ m/s}$$



Few other real physical problems,

Assignment -

Any numerical as suggested by Faculty.

Sample numerical Problem-

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),

$$\frac{dT}{dt} = -k(T - T_a)$$

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. Also validate the numerical results/plot with that of analytical solution.

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 –

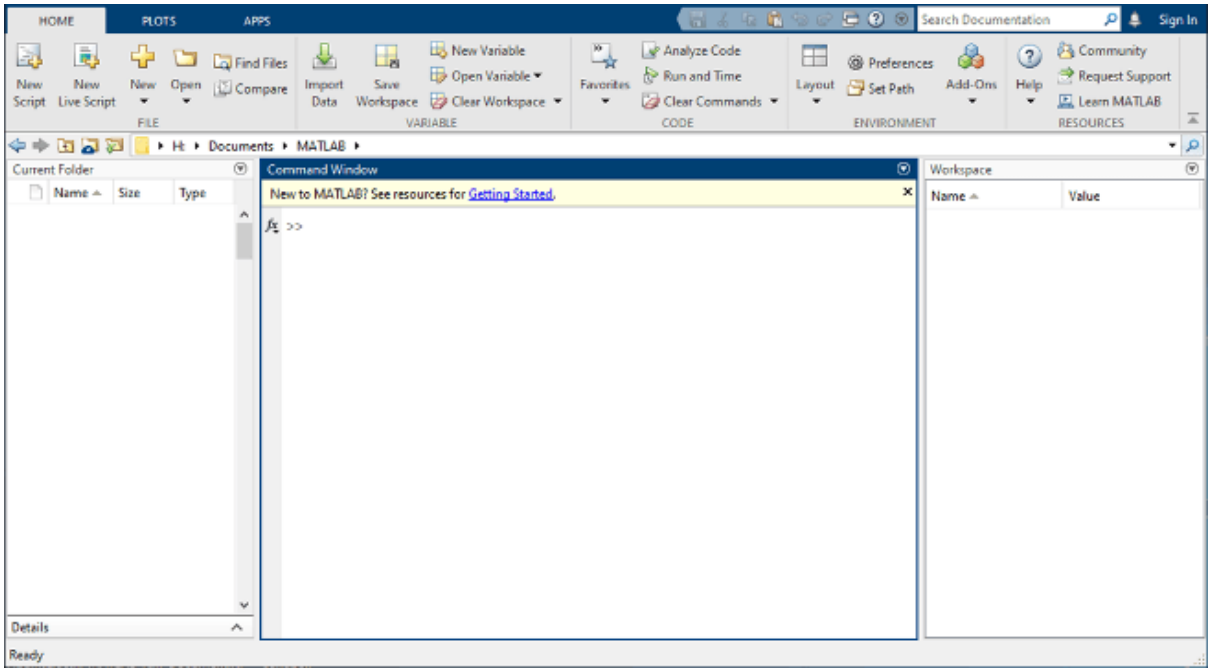
- Dealing with Matrices and Arrays
- 2-D and 3-D Plotting and graphics
- Linear Algebra
- Algebraic Equations
- Non-linear Functions
- Statistics
- Data Analysis
- Calculus and Differential Equations
- Numerical Calculations
- Integration
- Transforms
- Curve Fitting
- Various other special functions

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.

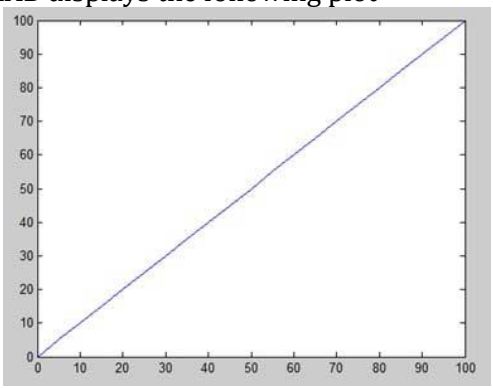
Special Variables and Constants
MATLAB supports the following special variables and constants –

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)
```

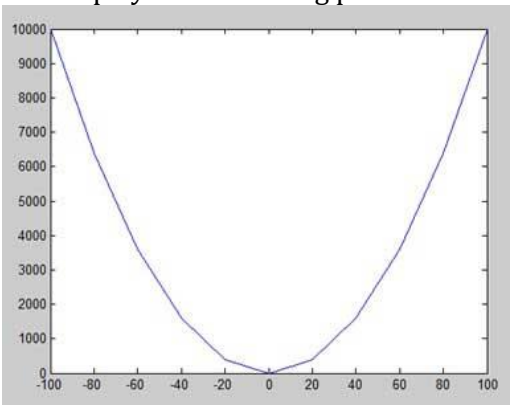
When you run the file, MATLAB displays the following plot –



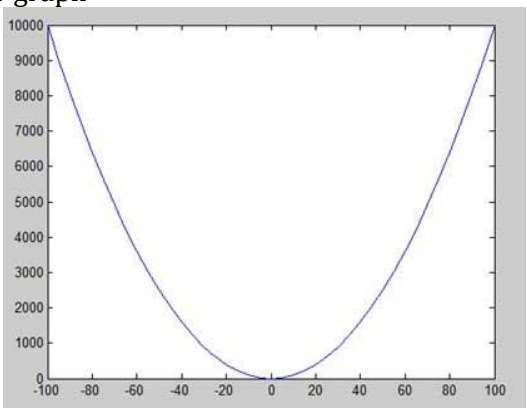
Let us take one more example to plot the function $y = x^2$. In this example, we will draw two graphs with the same function, but in second time, we will reduce the value of increment. Please note that as we decrease the increment, the graph becomes smoother.
Create a script file and type the following code –

```
x = [1 2 3 4 5 6 7 8 9 10];
x = [-100:20:100];
```

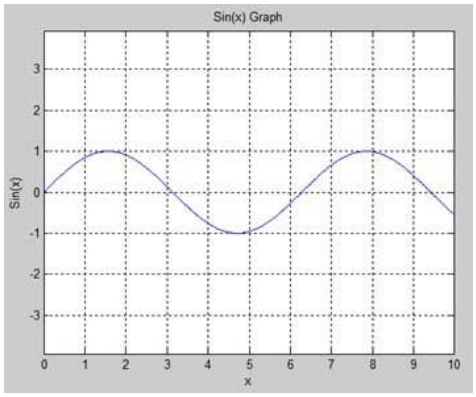
y = x.^2;
plot(x, y)
When you run the file, MATLAB displays the following plot –



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

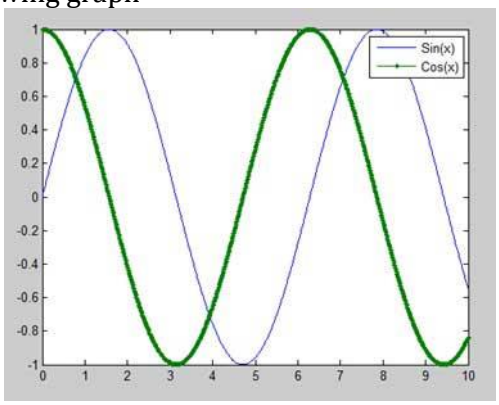
You can draw multiple graphs on the same plot. The following example demonstrates the concept –

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 –

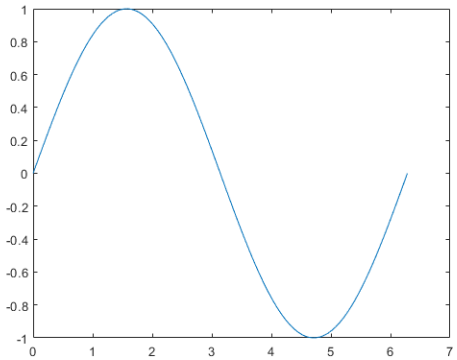


2-D and 3-D Plots

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 2π:

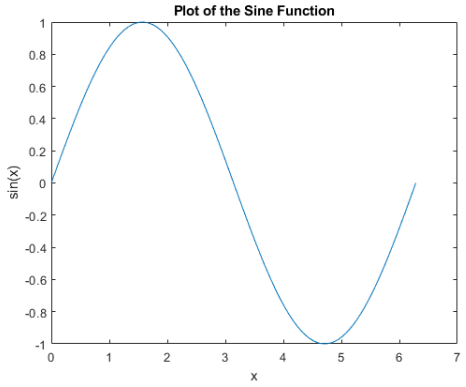
```
x = linspace(0,2*pi,50);
y = sin(x);
plot(x,y)
```



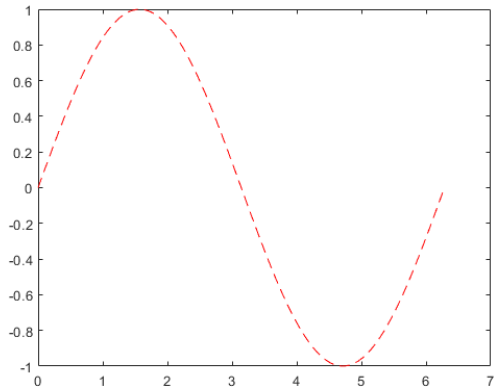
You can label the axes and add a title.

```
xlabel("x")
```

ylabel("sin(x)")
title("Plot of the Sine Function")

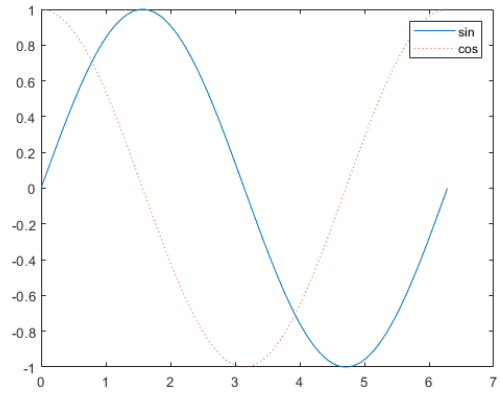


By adding a third input argument to the plot function, you can plot the same variables using a red dashed line.
plot(x,y,"r--")



"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
```

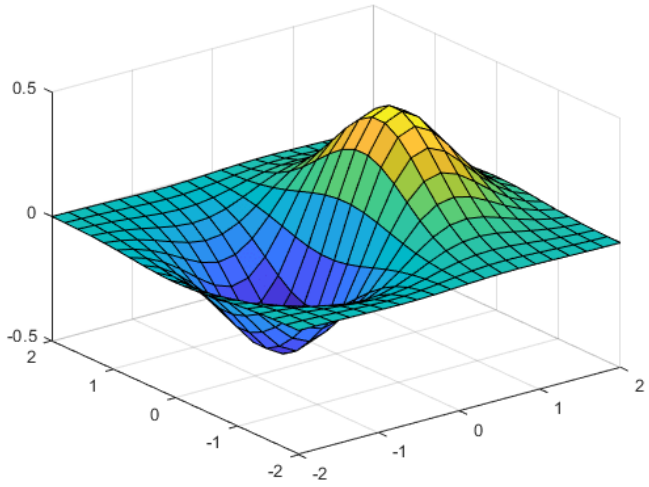


3-D Plots

Three-dimensional plots typically display a surface defined by a function in two variables, $z=f(x,y)$. For instance, calculate $z=xe^{-x2-y2}$ 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, 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, 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 -

One numerical as suggested by Faculty.

Sample numerical.

1. Draw the functions

$$y_1 = x^2$$
$$y_2 = x^3$$
$$y_3 = e^x$$
$$y_4 = \sin(x) * \cos(x)$$

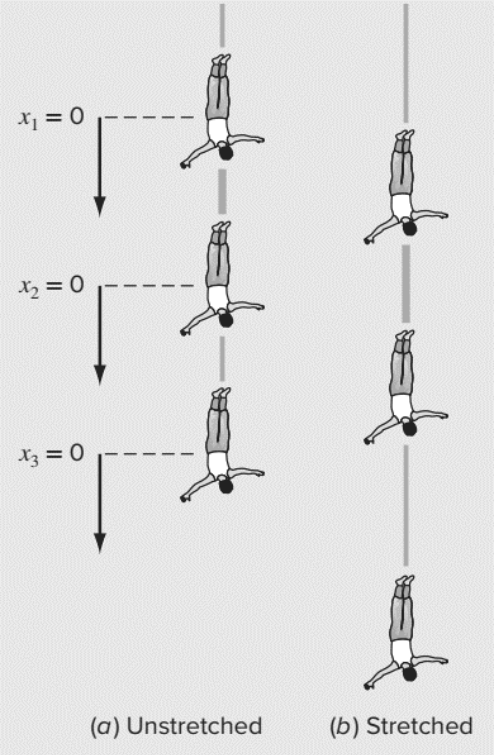
   on same plot for the range of x from -5 to +5. Also label the plots aesthetically.

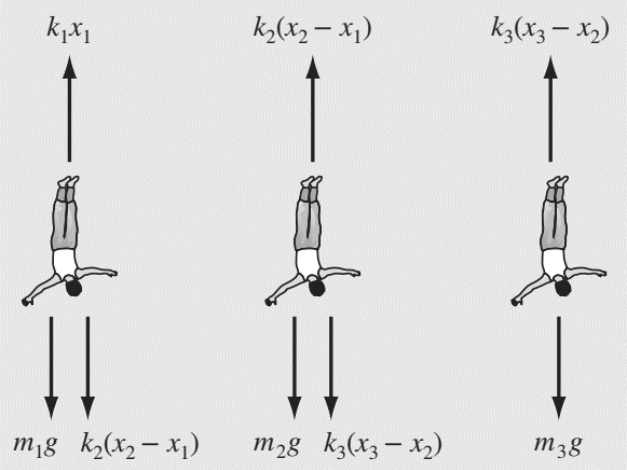
2. Make a MATLAB program that sums number from 1 to 100 using FOR loop statement.
3. Write a MATLAB code to plot 'tan' function with x domain as -2π to 2π with 150 number of points. (Hint – when you plot, you will notice the plot is not aesthetic. To make your plot aesthetic, use – xlim and ylim functions.
   Example - xlim([-2*pi 2*pi]); ylim([-10 10]);

LAB 3: Systems of Linear Algebraic Equations

Objective: To represent a system of linear algebraic equations in matrix form and obtain the solution.

Theory: Suppose that three jumpers are connected by bungee cords. Being held in place vertically so that each cord is fully extended but unstretched. We can define three distances, x_1, x_2, and x_3, as measured downward from each of a shows them their unstretched positions. After they are released, gravity takes hold and the jumpers will eventually come to the equilibrium positions shown in Suppose that you are asked to compute the displacement of each of the jumpers. If we assume that each cord behaves as a linear spring and follows Hooke's law, free-body diagrams can be developed for each jumper as depicted-





Free body diagram (using Hooke's law)

Using Newton's second law, force balances can be written for each jumper:

$$m_1 \frac{d^2x_1}{dt^2} = m_1 g + k_2(x_2 - x_1) - k_1x_1$$

$$m_2 \frac{d^2x_2}{dt^2} = m_2 g + k_3(x_3 - x_2) + k_2(x_1 - x_2)$$

$$m_3 \frac{d^2x_3}{dt^2} = m_3 g + k_3(x_2 - x_3)$$

where m_i = the mass of jumper i (kg), t = time (s), k_j = the spring constant for cord j (N/m), x_i = the displacement of jumper i measured downward from the equilibrium position (m), and g = gravitational acceleration (9.81 m/s^2). Because we are interested in the steady-state solution, the second derivatives can be set to zero. Collecting terms gives

$$\begin{aligned}(k_1 + k_2)x_1 \qquad - k_2x_2 \qquad &= m_1g\\ -k_2x_1 + (k_2 + k_3)x_2 - k_3x_3 &= m_2g\\ -k_3x_2 + k_3x_3 &= m_3g\end{aligned}$$

Thus, the problem reduces to solving a system of three simultaneous equations for the three unknown displacements. Because we have used a linear law for the cords, these equations are linear algebraic equations. MATLAB provides two direct ways to solve systems of linear algebraic equations. The most efficient way is to employ the backslash, or "left-division," operator as in

```
>> x = A\b
```

The second is to use matrix inversion:

```
>> x = inv(A)*b
```

However, the matrix inverse solution is less efficient than using the backslash.

Use MATLAB to solve the bungee jumper problem described. The parameters for the problem are

Jumper	Mass (kg)	Spring Constant (N/m)	Unstretched Cord Length (m)
Top (1)	60	50	20
Middle (2)	70	100	20
Bottom (3)	80	50	20

The system of equations

$$\begin{aligned}(k_1 + k_2)x_1 \qquad - k_2x_2 \qquad &= m_1g\\ -k_2x_1 + (k_2 + k_3)x_2 - k_3x_3 &= m_2g\\ -k_3x_2 + k_3x_3 &= m_3g\end{aligned}$$

are written as,

$$\begin{bmatrix} 150 & -100 & 0 \\ -100 & 150 & -50 \\ 0 & -50 & 50 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 588.6 \\ 686.7 \\ 784.8 \end{Bmatrix}$$

Start up MATLAB and enter the coefficient matrix and the right-hand-side vector:

```
>> K = [150 -100 0;-100 150 -50;0 -50 50]

K =
   150  -100     0
  -100   150   -50
     0   -50    50

>> mg = [588.6; 686.7; 784.8]

mg =
  588.6000
  686.7000
  784.8000
```

Employing left division yields

```
>> x = K\mg

x =
   41.2020
   55.9170
   71.6130
```

Alternatively, multiplying the inverse of the coefficient matrix by the right-hand-side vector gives the same result:

```
>> x = inv(K)*mg

x =
   41.2020
   55.9170
   71.6130
```

Because the jumpers were connected by 20-m cords, their initial positions relative to the platform is

```
>> xi = [20;40;60];
```

Thus, their final positions can be calculated as

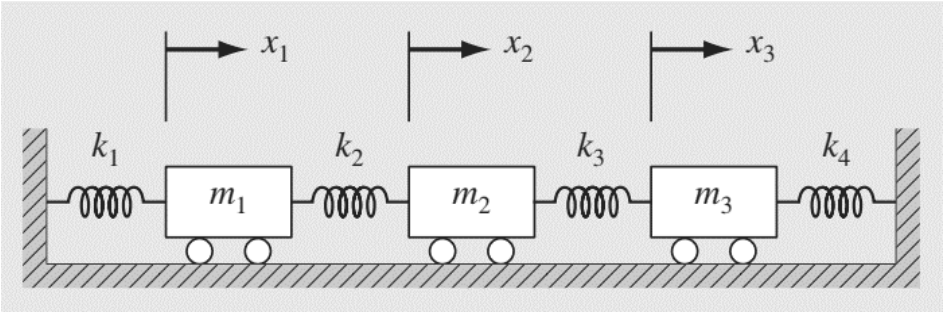
```
>> xf = x+xi

xf =
   61.2020
   95.9170
  131.6130
```

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

Consider the three mass-four spring system in Fig. Determining the equations of motion from $\Sigma F_x = ma_x$ for each mass using its free-body diagram results in the



Derive the governing equation for acceleration for all the three blocks?

where $k_1 = k_4 = 10$ N/m, $k_2 = k_3 = 30$ N/m, and $m_1 = m_2 = m_3 = 1$ kg. The three equations can be written in matrix form:

$$0 = \{\text{Acceleration vector}\} + [k/m \text{ matrix}]\{\text{displacement vector } x\}$$

At a specific time where $x_1 = 0.05$ m, $x_2 = 0.04$ m, and $x_3 = 0.03$ m, this forms a tridiagonal matrix. Use MATLAB to solve for the acceleration of each mass.

LAB 4: Eigenvalue Problems

Objective: To obtain the eigenvalue of a dynamical system.

Theory: Previously, have dealt with methods for solving sets of linear algebraic equations of the general form

$$[A]\{X\} = [B]$$

Such systems are called nonhomogeneous because of the presence of the vector {B} on the right-hand side of the equality. In contrast, a homogeneous linear algebraic system has a right-hand side equal to zero:

$$[A]\{X\} = [0]$$

At face value, this equation suggests that the only possible solution would be the trivial case for which all x's = 0. Graphically this would correspond to two straight lines that intersected at zero. Eigenvalue problems associated with engineering are typically of the general form,

$$[A]\{X\} = \lambda\{X\},$$

$$[A]- \lambda[I]\{X\} = 0,$$

Where the parameter λ is the eigenvalue. Thus, rather than setting the x's to zero, we can determine the value of λ that drives the left-hand side to zero! One way to accomplish this is based on the fact that, for nontrivial solutions to be possible, the determinant of the matrix must equal zero:

$$\det|[A]- \lambda[I]|=0$$

Expanding the determinant yields a polynomial in λ, which is called the characteristic polynomial. The roots of this polynomial are the solutions for the eigenvalues.

$$\begin{aligned}(a_{11} - \lambda)x_1 + a_{12}x_2 &= 0\\ a_{21}x_1 + (a_{22} - \lambda)x_2 &= 0\end{aligned}$$

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = \lambda^2 - (a_{11} + a_{22})\lambda - a_{12}a_{21}$$

$$\begin{matrix}\lambda_1 \\ \lambda_2\end{matrix} = \frac{(a_{11} - a_{22}) \pm \sqrt{(a_{11} - a_{22})^2 - 4a_{12}a_{21}}}{2}$$

Problem Statement. Use the polynomial method to solve for the eigenvalues of the following homogeneous system:

$$\begin{aligned}(10 - \lambda)x_1 - 5x_2 &= 0\\ -5x_1 + (10 - \lambda)x_2 &= 0\end{aligned}$$

$$\begin{matrix}\lambda_1 \\ \lambda_2\end{matrix} = \frac{20 \pm \sqrt{20^2 - 4(1)75}}{2} = 15, 5$$

We can now substitute either of these values back into the system and examine the result. For $\lambda_1 = 15$, we obtain

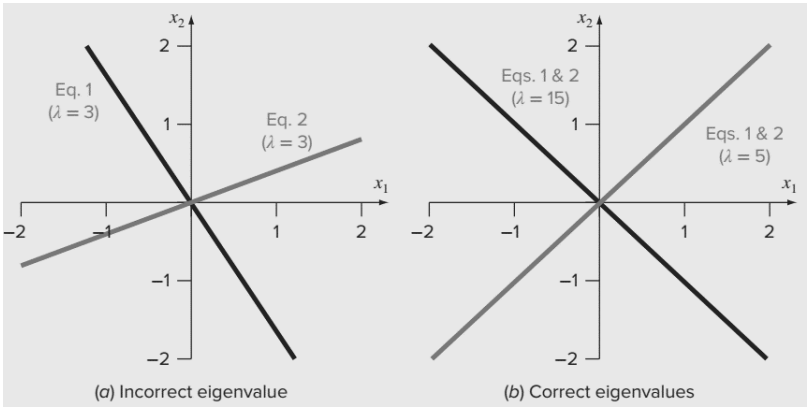
$$-5x_1 - 5x_2 = 0$$
$$-5x_1 - 5x_2 = 0$$

In essence as we move toward a correct eigenvalue the two lines rotate until they lie on top of each other. Mathematically, this means that there are an infinite number of solutions. But solving either of the equations yields the interesting result that all the solutions have the property that x 1 = −x2. Although at first glance this might appear trivial, it's actually quite interesting as it tells us that the ratio of the unknowns is a constant. This result can be expressed in vector form as,

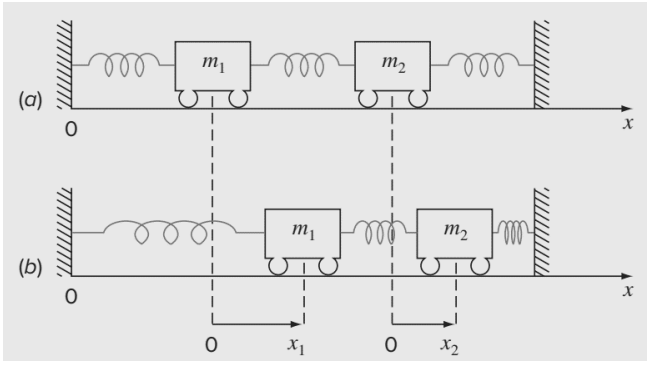
$$\{x\} = \begin{Bmatrix} -1 \\ 1 \end{Bmatrix}$$

which is referred to as the *eigenvector* corresponding to the eigenvalue $\lambda = 15$. In a similar fashion, substituting the second eigenvalue, $\lambda_2 = 5$, gives

$$5x_1 - 5x_2 = 0$$
$$-5x_1 + 5x_2 = 0$$



Example- A two mass–three spring system with frictionless rollers vibrating between two fixed walls. The position of the masses can be referenced to local coordinates with origins at their respective equilibrium positions (a). As in (b), positioning the masses away from equilibrium creates forces in the springs that on release lead to oscillations of the masses.



Newton's second law can be employed to develop a force balance for each mass:

$$m_1 \frac{d^2x_1}{dt^2} = -kx_1 + k(x_2 - x_1)$$

$$m_2 \frac{d^2x_2}{dt^2} = -k(x_2 - x_1) - kx_2$$

where x_i is the displacement of mass i away from its equilibrium position from vibration theory, it is known that solutions can take the form,

$$x_i = X_i \sin(\omega t)$$

where X_i = the *amplitude* of the oscillation of mass i (m) and ω = the *angular frequency* of the oscillation (radians/time), which is equal to

$$\omega = \frac{2\pi}{T_p}$$

where T_p = the *period* (time/cycle). Note that the inverse of the period is called the *ordinary frequency* f (cycles/time). If time is measured in seconds, the unit for f is the cycles/s, which is referred to as a *Hertz* (Hz).

Equation can be differentiated twice and substituted into Eq. After collection of terms, the result in,

$$\left(\frac{2k}{m_1} - \omega^2\right) X_1 - \frac{k}{m_1} X_2 = 0$$

$$-\frac{k}{m_2} X_1 + \left(\frac{2k}{m_2} - \omega^2\right) X_2 = 0$$

Comparison of Eq. with general form of eigenvalue Eq. indicates that at this point, the solution has been reduced to an eigenvalue problem—where, for this case, the eigenvalue is the square of the frequency.

If $m_1 = m_2 = 40$ kg and k = 200 N/m,

$$(10 - \lambda)x_1 - 5x_2 = 0$$
$$-5x_1 + (10 - \lambda)x_2 = 0$$

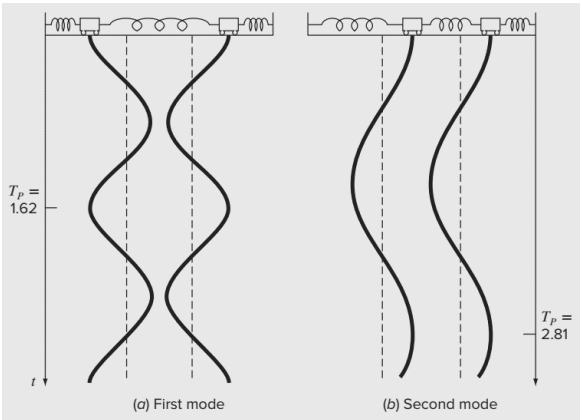


Figure - The principal modes of vibration of two equal masses connected by three identical springs between fixed walls.

As might be expected, MATLAB has powerful and robust capabilities for evaluating eigenvalues and eigenvectors. The function `eig`, which is used for this purpose, can be employed to generate a vector of the eigenvalues as in

```
>> e = eig(A)
```

where `e` is a vector containing the eigenvalues of a square matrix `A`. Alternatively, it can be invoked as

```
>> [V,D] = eig(A)
```

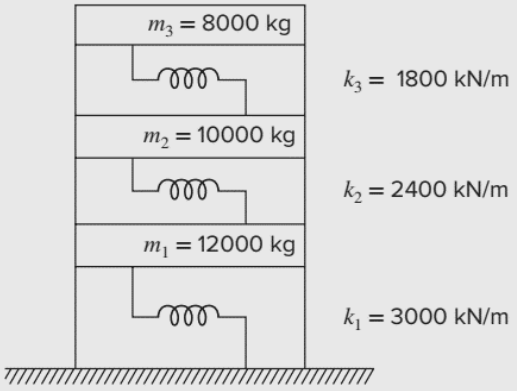
where `D` is a diagonal matrix of the eigenvalues and `V` is a full matrix whose columns are the corresponding eigenvectors.

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

A three-story building modeled as a mass-spring system.



Engineers and scientists use mass-spring models to gain insight into the dynamics of structures under the influence of disturbances such as earthquakes. Use MATLAB to determine the eigenvalues and eigenvectors for this system. Graphically represent the modes of vibration for the structure by displaying the amplitudes versus height for each of the eigenvectors. Normalize the amplitudes so that the translation of the third floor is one.

LAB 5: Non-linear Equations

Objective: To obtain the numerical solution of any non-linear equation.

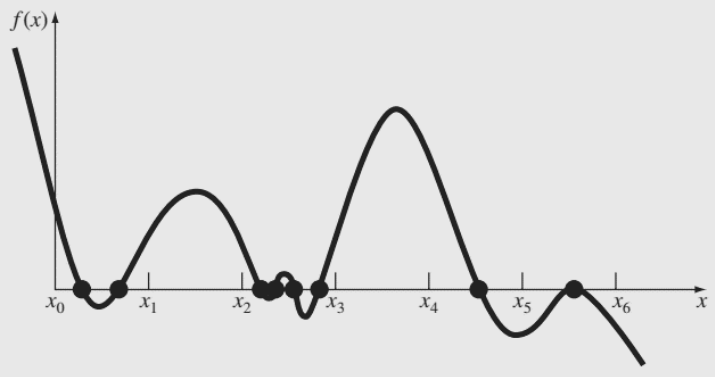
Theory: Root of the equation f(x) = 0 is to make a plot of the function and observe where it crosses the x axis. The two major classes of methods available are distinguished by the type of initial guess. They are

1. Bracketing methods - As the name implies, these are based on two initial guesses that "bracket" the root that is, are on either side of the root.
2. Open methods - These methods can involve one or more initial guesses, but there is no need for them to bracket the root.

f (x) changed sign on opposite sides of the root. In general, if f(x) is real and continuous in the interval from x_l to x_u and f(x_l) and f(x_u) have opposite signs, that is,

$$f(x_l)\, f(x_u) < 0$$

then there is at least one real root between x_l and x_u.



The Bisection method –

Example - To determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is 9.81 m/s2

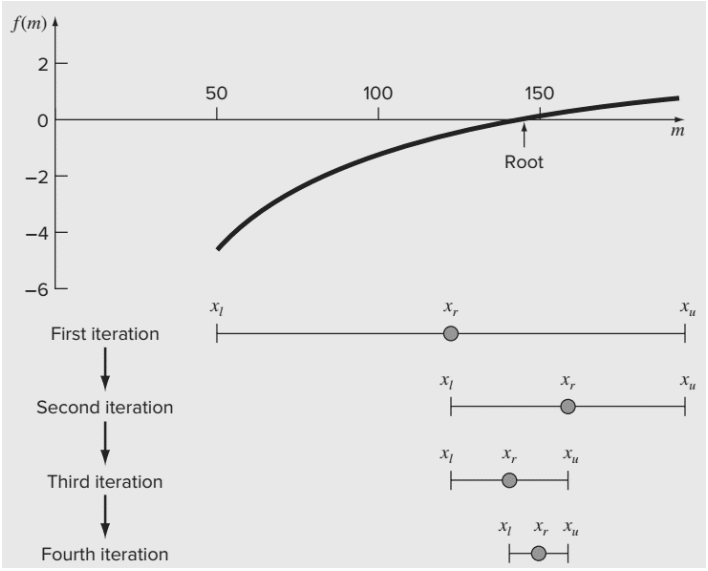
$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}\, t\right) - v(t)$$

We can see that the function changes sign between values of 50 and 200. The plot obviously suggests better initial guesses, say 140 and 150, but for illustrative purposes let's assume we don't have the benefit of the plot and have made conservative guesses. Therefore, the initial estimate of the root x_r lies at the midpoint of the interval,

$$x_r = \frac{50 + 200}{2} = 125$$

$$|\varepsilon_t| = \left|\frac{142.7376 - 125}{142.7376}\right| \times 100\% = 12.43\%$$

$$f(50)\,f(125) = -4.579(-0.409) = 1.871$$



At this point, the new interval extends from $x_l = 125$ to $x_u = 200$. A revised root estimate can then be calculated as

$$x_r = \frac{125 + 200}{2} = 162.5$$

which represents a true percent error of $|\varepsilon_t| = 13.85\%$. The process can be repeated to obtain refined estimates. For example,

$$f(125)\,f(162.5) = -0.409(0.359) = -0.147$$

Therefore, the root is now in the lower interval between 125 and 162.5. The upper bound is redefined as 162.5, and the root estimate for the third iteration is calculated as

$$x_r = \frac{125 + 162.5}{2} = 143.75$$

which represents a percent relative error of $\varepsilon_t = 0.709\%$. The method can be repeated until the result is accurate enough to satisfy your needs.

Stopping criteria,

$$|\varepsilon_a| = \left|\frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}}\right| 100\%$$

Iteration	x_l	x_u	x_r	$\lvert\varepsilon_a\rvert$ (%)	$\lvert\varepsilon_t\rvert$ (%)
1	50	200	125		12.43
2	125	200	162.5	23.08	13.85
3	125	162.5	143.75	13.04	0.71
4	125	143.75	134.375	6.98	5.86
5	134.375	143.75	139.0625	3.37	2.58
6	139.0625	143.75	141.4063	1.66	0.93
7	141.4063	143.75	142.5781	0.82	0.11
8	142.5781	143.75	143.1641	0.41	0.30

Other methods – False position, Newton-Raphson method etc.

Example- Find the roots of equation – $y(x) = e^{0.2x} - e^{-0.8x} - 2$ using bisection method. Take the initial guesses as root may lie between 0 and 4.

MATLAB routine for finding roots is as follows-

```
clc;close all; clear all;
x = linspace(0,10,50);
y = exp(0.2*x)-exp(-0.8*x)-2;
xl =0;
xu =4;
err= 0.01;
 iter = 1;
while abs(xu-xl)> err
    xr = (xl+xu)/2;
    yl = exp(0.2*xl)-exp(-0.8*xl)-2;
    yr = exp(0.2*xr)-exp(-0.8*xr)-2;
    yu = exp(0.2*xu)-exp(-0.8*xu)-2;
   if yl*yr <0
       xu=xr;
   else
       xl=xr;
   end
  iter = iter+1;
end
fprintf('root of equation = %f',xr);
plot(x,y);
hold on;
hline = refline([0 0]);
```

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

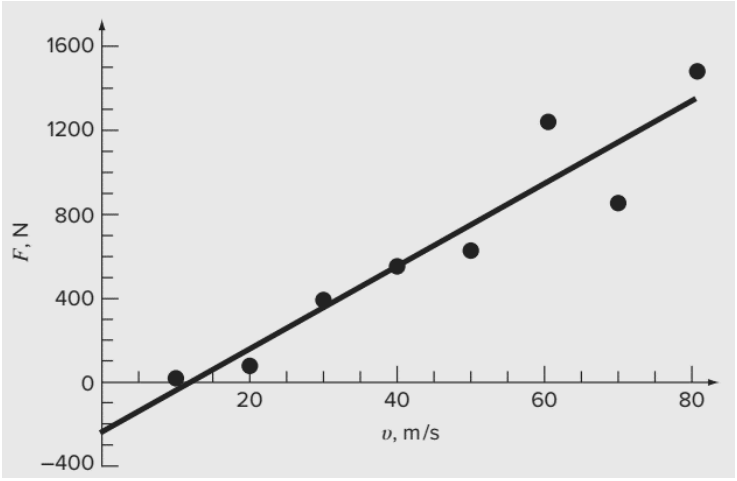
Using Bisection method, locate the root of –

$$F(x) = x^{10} - 1, \text{ between } x = 0 \text{ to } 1.3$$

LAB 6: Polynomial Approximation

Objective: To obtain the linear and polynomial approximation (curve fitting) for a given data set.

Theory: Given a set of points (x_i, y_i) for i=0,1,2,..., n, where the x_i are distinct values of the independent variable and the y_i are corresponding values of some function f, Either Approximate the value of y at some value of x not listed among the x_i or Determine a function g that in some sense approximate the data.



The mathematical expression for the straight line is

$$y = a_0 + a_1x + e$$

where a_0 and a_1 are coefficients representing the intercept and the slope, respectively, and e is the error, or residual, between the model and the observations, which can be represented by rearranging Eq. as

$$e = y - a_0 - a_1x$$

Thus, the residual is the discrepancy between the true value of y and the approximate value, $a_0 + a_1x$, predicted by the linear equation. The approach is to minimize the sum of the squares of the residuals as,

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1x_i)^2$$

This criterion, which is called *least square method*, has a number of advantages, including that it yields a unique line for a given set of data. To determine values for a_0 and a_1, Eq. is differentiated with respect to each unknown coefficient:

$$\frac{\partial S_r}{\partial a_0} = -2\sum (y_i - a_0 - a_1x_i)$$

$$\frac{\partial S_r}{\partial a_1} = -2\sum [(y_i - a_0 - a_1x_i)x_i]$$

all summations are from i = 1 to n. Setting these derivatives equal to zero will result in a minimum S_r. If this is done, the equations can be expressed as,

$$0 = \sum y_i - \sum a_0 - \sum a_1 x_i$$

$$0 = \sum x_i y_i - \sum a_0 x_i - \sum a_1 x_i^2$$

Upon rearranging,

$$n \quad a_0 + \left(\sum x_i\right) a_1 = \sum y_i$$

$$\left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 = \sum x_i y_i$$

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2}$$

$a_0 = \bar{y} - a_1 \bar{x}$ where $\bar{y}$ and $\bar{x}$ are the means of y and x, respectively.

Problem - Fit a straight line to the values in Table given below,

x_i	y_i
10	25
20	70
30	380
40	550
50	610
60	1,220
70	830
80	1,450

Solution – Following values needs to be calculated-

i	x_i	y_i	x_i^2	$x_i y_i$
1	10	25	100	250
2	20	70	400	1,400
3	30	380	900	11,400
4	40	550	1,600	22,000
5	50	610	2,500	30,500
6	60	1,220	3,600	73,200
7	70	830	4,900	58,100
8	80	1,450	6,400	116,000
Σ	360	5,135	20,400	312,850

The means can be computed as

$$\bar{x} = \frac{360}{8} = 45 \qquad \bar{y} = \frac{5{,}135}{8} = 641.875$$

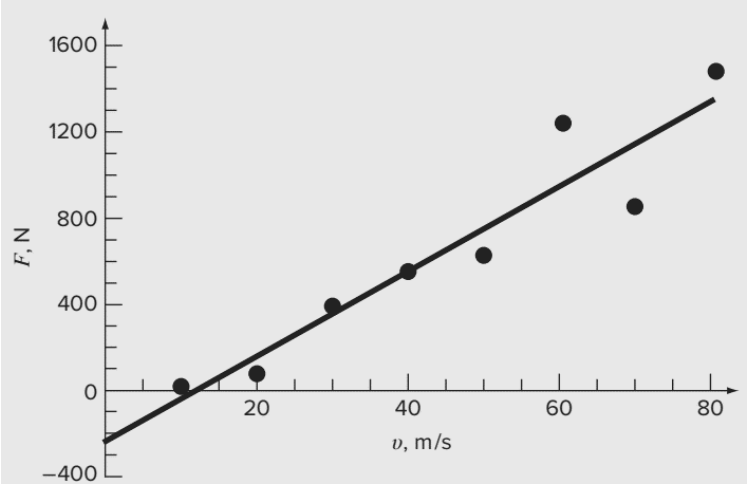
The slope and the intercept can then be calculated with Eqs. (14.15) and (14.16) as

$$a_1 = \frac{8(312{,}850) - 360(5{,}135)}{8(20{,}400) - (360)^2} = 19.47024$$

$$a_0 = 641.875 - 19.47024(45) = -234.2857$$

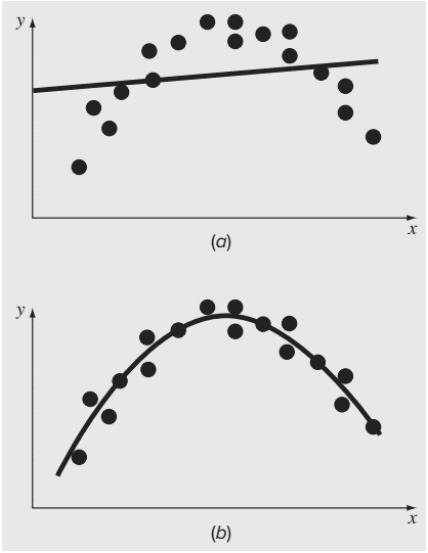
Using force and velocity in place of y and x, the least-squares fit is

$$F = -234.2857 + 19.47024v$$



In MALTAB, first define X vector and Y vector. Then type cftools (or open curve fitting tool from app menu). Explore the linear/polynomial of various degrees and observe the goodness of fit and r values.

Polynomial Regression-



The least-squares procedure can be readily extended to fit the data to a higher-order polynomial. For example, suppose that we fit a second-order polynomial or quadratic:

$$y = a_0 + a_1 x + a_2 x^2 + e$$

For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \tag{15.2}$$

To generate the least-squares fit, we take the derivative of Eq. (15.2) with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2 \sum \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$(n)a_0 + \left(\sum x_i\right) a_1 + \left(\sum x_i^2\right) a_2 = \sum y_i$$

$$\left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 = \sum x_i y_i$$

$$\left(\sum x_i^2\right) a_0 + \left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 = \sum x_i^2 y_i$$

Polynomial Regression

Problem Statement. Fit a second-order polynomial to the data in the first two columns of Table 15.1.

TABLE 15.1 Computations for an error analysis of the quadratic least-squares fit.

x_i	y_i	$(y_i - \bar{y})^2$	$(y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$
0	2.1	544.44	0.14332
1	7.7	314.47	1.00286
2	13.6	140.03	1.08160
3	27.2	3.12	0.80487
4	40.9	239.22	0.61959
5	61.1	1272.11	0.09434
$\sum$	152.6	2513.39	3.74657

Solution. The following can be computed from the data:

$m = 2$ $\quad \sum x_i = 15$ $\quad \sum x_i^4 = 979$

$n = 6$ $\quad \sum y_i = 152.6$ $\quad \sum x_i y_i = 585.6$

$\bar{x} = 2.5$ $\quad \sum x_i^2 = 55$ $\quad \sum x_i^2 y_i = 2488.8$

$\bar{y} = 25.433$ $\quad \sum x_i^3 = 225$

Therefore, the simultaneous linear equations are

$$\begin{bmatrix} 6 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 152.6 \\ 585.6 \\ 2488.8 \end{Bmatrix}$$

```
a =
    2.4786
    2.3593
    1.8607
```

Therefore, the least-squares quadratic equation for this case is

$$y = 2.4786 + 2.3593x + 1.8607x^2$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

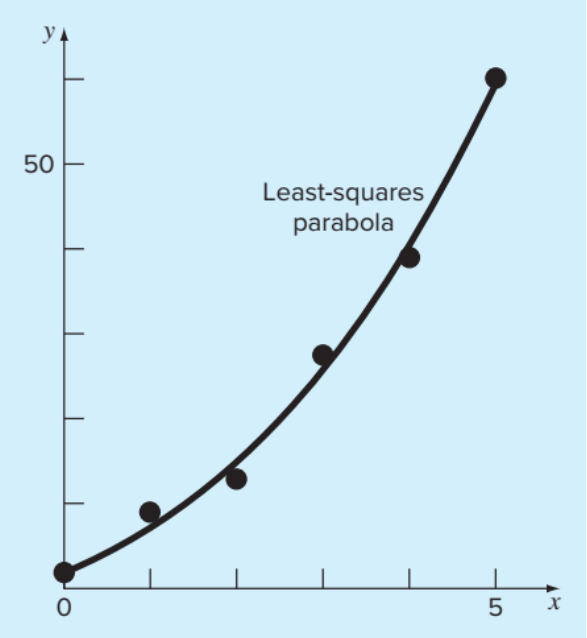
The most general definition of the coefficient of determination is

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

The coefficient of determination is

$$r^2 = \frac{2513.39 - 3.74657}{2513.39} = 0.99851$$

and the correlation coefficient is $r = 0.99925$



MATLAB routine for finding curve fitting (using polyfit function) –

```
clear all; close all; clc;
X = 10:10:80;
Y = [ 25 70 380 550 610 1220 830 1450];
p = polyfit(X,Y,1); % for linear regression
% p = polyfit(X,Y,2) % for qudratic regression
% p = polyfit(X,Y,3) % for cubic regression
scatter(X,Y,'r*');
% make straight line using coefficients found using polyfit function
Y1 = p(2) + p(1)*X;
hold on;
plot(X,Y1,'-b');
```

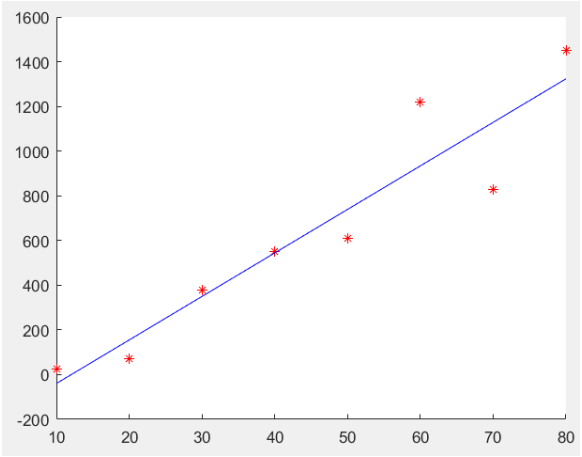


Figure – linear regression of the given data

Alternatively, you can use inbuilt curve fitting tool in MATLAB.

Introducing the Curve Fitting App

You can fit curves and surfaces to data and view plots with the Curve Fitting app.

- Create, plot, and compare multiple fits.
- Use linear or nonlinear regression, interpolation, smoothing, and custom equations.
- View goodness-of-fit statistics, display confidence intervals and residuals, remove outliers, and assess fits with validation data.
- Automatically generate code to fit and plot curves and surfaces, or export fits to the workspace for further analysis.

Fit a Curve

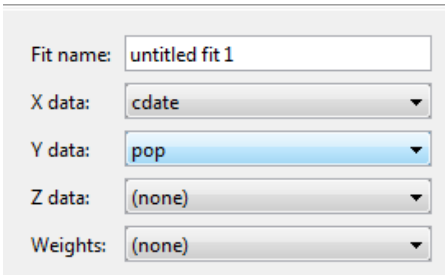
1. Load some example data at the MATLAB® command line:

   load census.mat

2. Open the Curve Fitting app by entering:

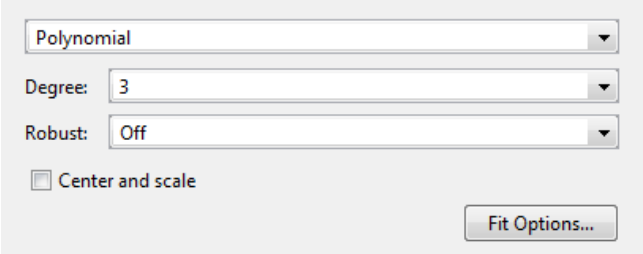
   cftool

   Alternatively, click **Curve Fitting** on the **Apps** tab.
3. Select **X data** and **Y data**. For details, see Selecting Data to Fit in Curve Fitting App.

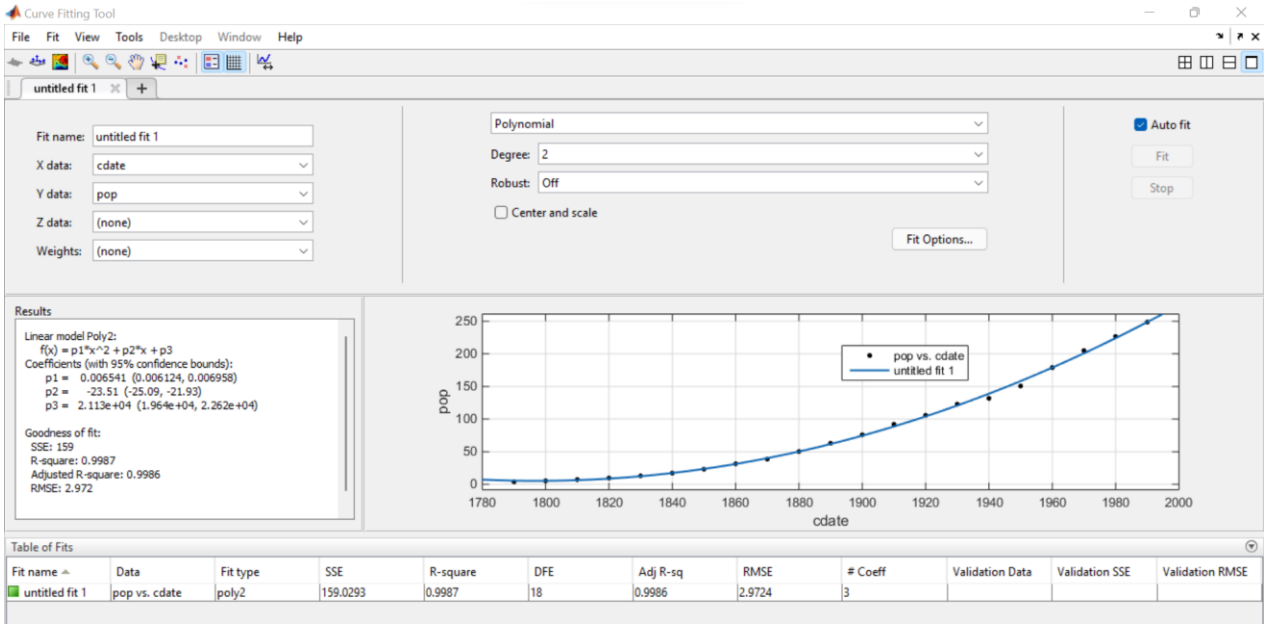


The Curve Fitting app creates a default polynomial fit to the data.

4. Try different fit options. For example, change the polynomial **Degree** to 3 to fit a cubic polynomial.



5. Select a different model type from the fit category list, e.g., **Smoothing Spline**. For information about models you can fit, see Model Types for Curves and Surfaces.



Assignment -

One numerical as suggested by Faculty.

Sample numerical.

Q. Suppose we have xdata and ydata points observed in any experiment as follows –

x	0	2	4	6	9	11	12	15	17	19
y	5	6	7	6	9	8	8	10	12	12

Using hand calculations, show the linear regression for the above data also using curve fitting toolbox in MATLAB, evaluate the goodness of fits with linear and polynomial fit with 2nd degree.

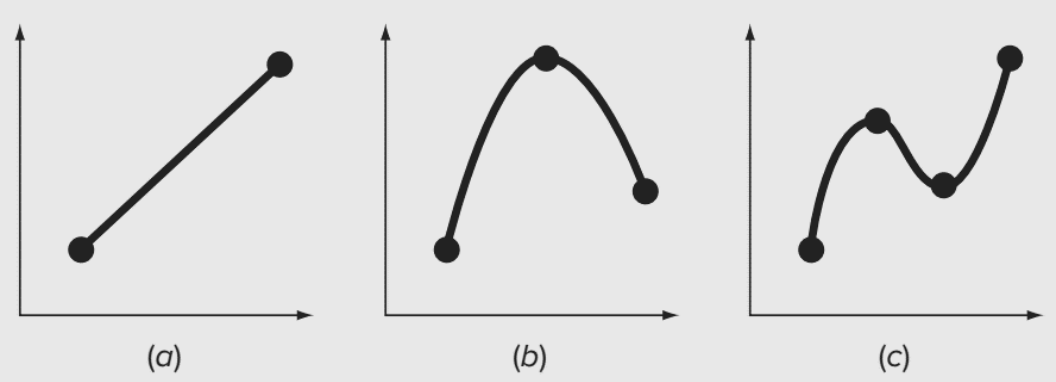
LAB 7: Polynomial Interpolation

Objective: To obtain the linear/polynomial interpolation using Newton/Lagrange's polynomial.

Theory: You will frequently have occasion to estimate intermediate values between precise data points. The most common method used for this purpose is polynomial interpolation. The general formula for an (n – 1)th-order polynomial can be written as-

$$f(x) = a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1}$$

For n data points, there is one and only one polynomial of order (n – 1) that passes through all the points. For example, there is only one straight line (i.e., a first-order polynomial) that connects two points. Similarly, only one parabola connects a set of three points. Polynomial interpolation consists of determining the unique (n–1) order polynomial that fits n data points. This polynomial then provides a formula to compute intermediate values.



Examples of interpolating polynomials: (*a*) first-order (linear) connecting two points, (*b*) second-order (quadratic or parabolic) connecting three points, and (*c*) third-order (cubic) connecting four points.

A straightforward way for computing the coefficients, is based on the fact that n data points are required to determine the n coefficients.

Problem Statement. Suppose that we want to determine the coefficients of the parabola, $f(x) = p_1x^2 + p_2x + p_3$, that passes through the last three density values from Table 17.1:

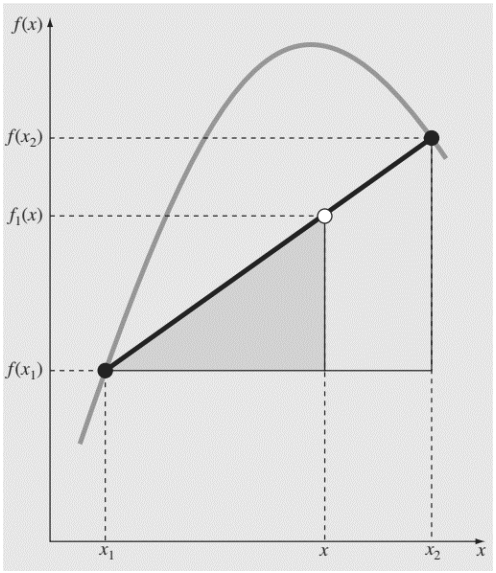
$$x_1 = 300 \quad f(x_1) = 0.616$$
$$x_2 = 400 \quad f(x_2) = 0.525$$
$$x_3 = 500 \quad f(x_3) = 0.457$$

This results in,

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{Bmatrix}$$

Such matrices are very ill-conditioned. That is, their solutions are very sensitive to round off errors. There are methods called, Newton's interpolating polynomial and Lagrange's interpolating polynomial, we can get rid of above problem.

The simplest form of interpolation is to connect two data points with a straight line. This technique, called linear interpolation, is depicted graphically in Fig.



Using similar triangles,

$$\frac{f_1(x) - f(x_1)}{x - x_1} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

which can be rearranged to yield

$$f_1(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)$$

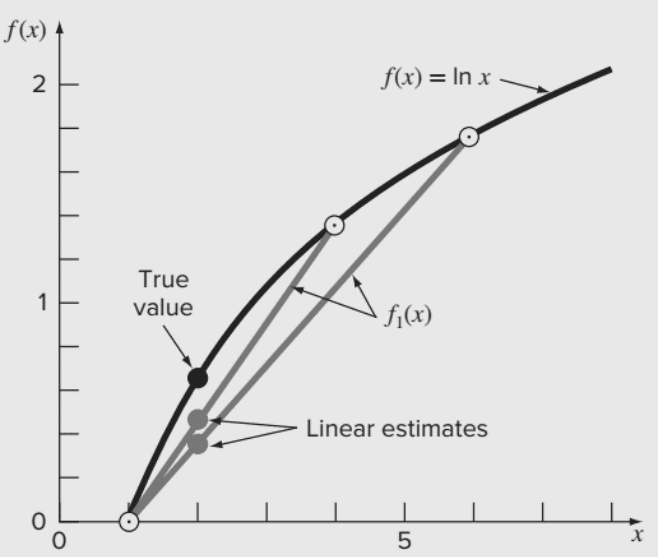
Problem Statement. Estimate the natural logarithm of 2 using linear interpolation. First, perform the computation by interpolating between ln 1 = 0 and ln 6 = 1.791759. Then, repeat the procedure, but use a smaller interval from ln 1 to ln 4 (1.386294). Note that the true value of ln 2 is 0.6931472.

Solution. We use Eq. (17.5) from $x_1 = 1$ to $x_2 = 6$ to give

$$f_1(2) = 0 + \frac{1.791759 - 0}{6 - 1}(2 - 1) = 0.3583519$$

which represents an error of $\varepsilon_t = 48.3\%$. Using the smaller interval from $x_1 = 1$ to $x_2 = 4$ yields

$$f_1(2) = 0 + \frac{1.386294 - 0}{4 - 1}(2 - 1) = 0.4620981$$



17.2.2 Quadratic Interpolation

The error in Example 17.2 resulted from approximating a curve with a straight line. Consequently, a strategy for improving the estimate is to introduce some curvature into the line connecting the points. If three data points are available, this can be accomplished with a second-order polynomial (also called a quadratic polynomial or a parabola). A particularly convenient form for this purpose is

$$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) \tag{17.6}$$

A simple procedure can be used to determine the values of the coefficients. For b_1, Eq. (17.6) with $x = x_1$ can be used to compute

$$b_1 = f(x_1) \tag{17.7}$$

Equation (17.7) can be substituted into Eq. (17.6), which can be evaluated at $x = x_2$ for

$$b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{17.8}$$

Finally, Eqs. (17.7) and (17.8) can be substituted into Eq. (17.6), which can be evaluated at $x = x_3$ and solved (after some algebraic manipulations) for

$$b_3 = \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1} \tag{17.9}$$

Problem Statement. Employ a second-order Newton polynomial to estimate ln 2 with the same three points used in Example 17.2:

$$x_1 = 1 \qquad f(x_1) = 0$$

$$x_2 = 4 \qquad f(x_2) = 1.386294$$

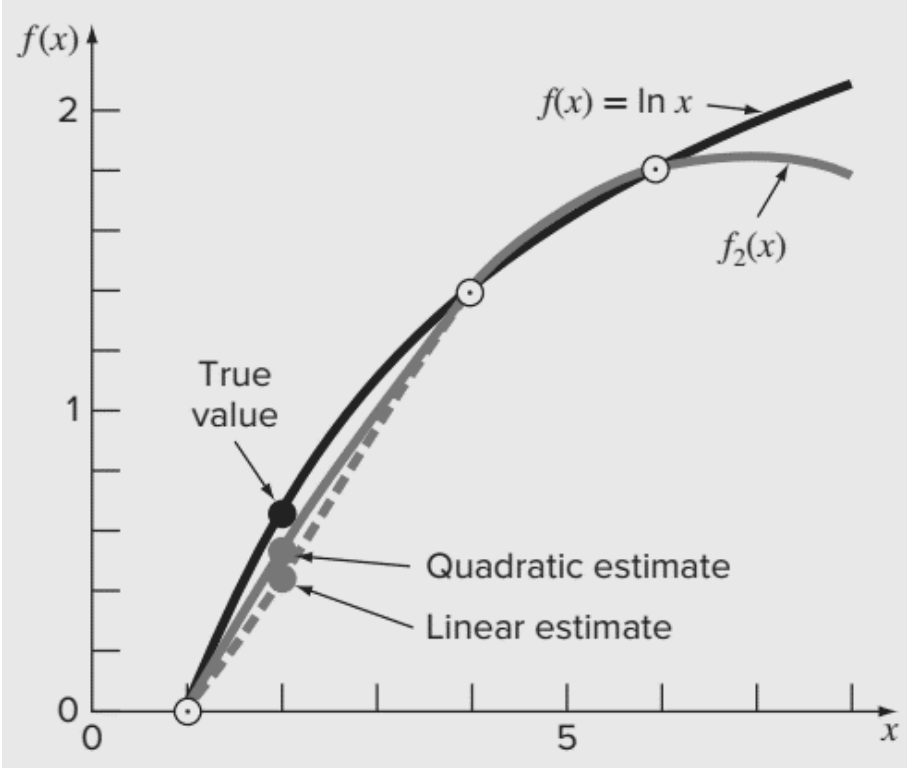
$$x_3 = 6 \qquad f(x_3) = 1.791759$$

Solution. Applying Eq. (17.7) yields

$$b_1 = 0$$

Equation (17.8) gives

$$b_2 = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$



and Eq. (17.9) yields

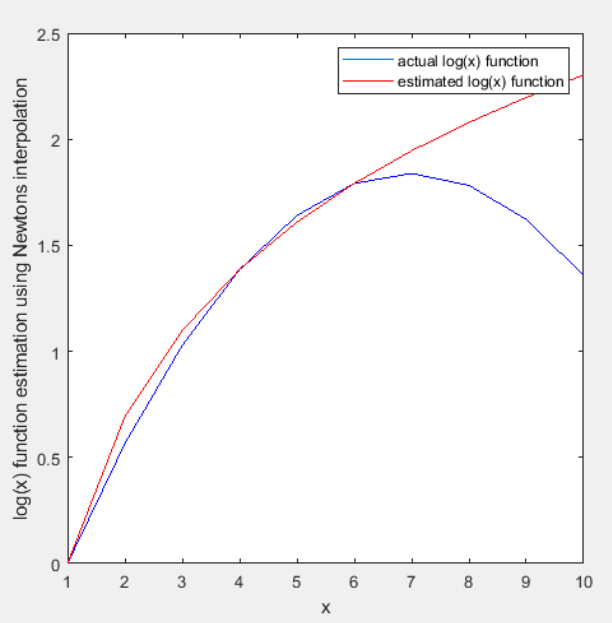
$$b_3 = \frac{\dfrac{1.791759 - 1.386294}{6 - 4} - 0.4620981}{6 - 1} = -0.0518731$$

Substituting these values into Eq. (17.6) yields the quadratic formula

$$f_2(x) = 0 + 0.4620981(x - 1) - 0.0518731(x - 1)(x - 4)$$

MATLAB Routine –

```
clc;clear all; close all;
x=1:1:10;
y = (-0.0518731*x.^2) + (0.7214635)*(x) - 0.6695904;
y1=log(x);
plot (x,y,'b');
hold on;
plot(x,y1,'r')
xlabel('x'); ylabel('log(x) function estimation using Newtons interpolation')
legend('actual log(x) function','estimated log(x) function')
```



LAGRANGE INTERPOLATING POLYNOMIAL

Suppose we formulate a linear interpolating polynomial as the weighted average of the two values that we are connecting by a straight line:

$$f(x) = L_1 f(x_1) + L_2 f(x_2) \tag{17.19}$$

where the L's are the weighting coefficients. It is logical that the first weighting coefficient is the straight line that is equal to 1 at x_1 and 0 at x_2:

$$L_1 = \frac{x - x_2}{x_1 - x_2}$$

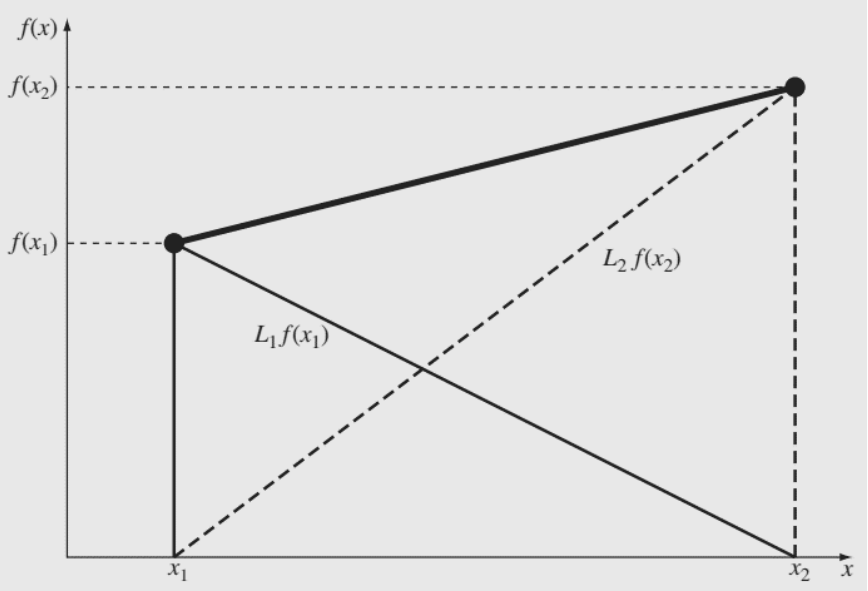
Similarly, the second coefficient is the straight line that is equal to 1 at x_2 and 0 at x_1:

$$L_2 = \frac{x - x_1}{x_2 - x_1}$$

Substituting these coefficients into Eq. (17.19) yields the straight line that connects the points (Fig. 17.8):

$$f_1(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2) \tag{17.20}$$

where the nomenclature $f_1(x)$ designates that this is a first-order polynomial. Equation (17.20) is referred to as the *linear Lagrange interpolating polynomial.*



A visual depiction of the rationale behind Lagrange interpolating polynomials. The figure shows the first-order case. Each of the two terms of Eq. (17.20) passes through one of the points and is zero at the other. The summation of the two terms must, therefore, be the unique straight line that connects the two points.

The same strategy can be employed to fit a parabola through three points. For this case three parabolas would be used with each one passing through one of the points and equaling zero at the other two. Their sum would then represent the unique parabola that connects the three points. Such a second-order Lagrange interpolating polynomial can be written as

$$f_2(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3) \tag{17.21}$$

Notice how the first term is equal to $f(x_1)$ at x_1 and is equal to zero at x_2 and x_3. The other terms work in a similar fashion.

where

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

where n = the number of data points and $\prod$ designates the "product of."

Problem Statement. Use a Lagrange interpolating polynomial of the first and second order to evaluate the density of unused motor oil at $T = 15$ °C based on the following data:

$x_1 = 0 \qquad f(x_1) = 3.85$

$x_2 = 20 \qquad f(x_2) = 0.800$

$x_3 = 40 \qquad f(x_3) = 0.212$

Solution. The first-order polynomial [Eq. (17.20)] can be used to obtain the estimate at $x = 15$:

$$f_1(x) = \frac{15 - 20}{0 - 20} 3.85 + \frac{15 - 0}{20 - 0} 0.800 = 1.5625$$

In a similar fashion, the second-order polynomial is developed as [Eq. (17.21)]

$$f_2(x) = \frac{(15 - 20)(15 - 40)}{(0 - 20)(0 - 40)} 3.85 + \frac{(15 - 0)(15 - 40)}{(20 - 0)(20 - 40)} 0.800 + \frac{(15 - 0)(15 - 20)}{(40 - 0)(40 - 20)} 0.212 = 1.3316875$$

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

17.4 Given the data

x	1	2	2.5	3	4	5
$f(x)$	0	5	7	6.5	2	0

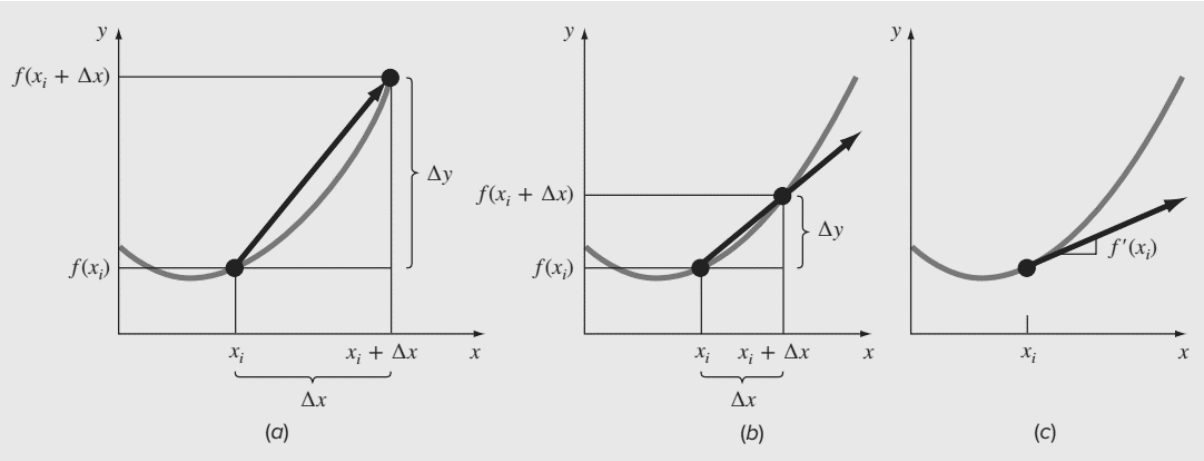
(a) Calculate $f(3.4)$ using Newton's interpolating polynomials of order 1 through 3. Choose the sequence of the points for your estimates to attain the best possible accuracy. That is, the points should be centered around and as close as possible to the unknown.

(b) Repeat **(a)** but use the Lagrange polynomial.

LAB 8: Numerical Differentiation

Objective: To numerically obtain the derivative of any function.

Theory: Calculus is the mathematics of change. Because engineers and scientists must continuously deal with systems and processes that change, calculus is an essential tool of our profession. Mathematically, the derivative, which serves as the fundamental vehicle for differentiation, represents the rate of change of a dependent variable with respect to an independent variable.



The graphical definition of a derivative: as Δx approaches zero in going from (*a*) to (*c*), the difference approximation becomes a derivative.

$$\frac{\Delta y}{\Delta x} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$$

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$$

Law	Equation	Physical Area	Gradient	Flux	Proportionality
Fourier's law	$q = -k\frac{dT}{dx}$	Heat conduction	Temperature	Heat flux	Thermal Conductivity
Fick's law	$J = -D\frac{dc}{dx}$	Mass diffusion	Concentration	Mass flux	Diffusivity
Darcy's law	$q = -k\frac{dh}{dx}$	Flow through porous media	Head	Flow flux	Hydraulic Conductivity
Ohm's law	$J = -\sigma\frac{dV}{dx}$	Current flow	Voltage	Current flux	Electrical Conductivity
Newton's viscosity law	$\tau = \mu\frac{du}{dx}$	Fluids	Velocity	Shear Stress	Dynamic Viscosity
Hooke's law	$\sigma = E\frac{\Delta L}{L}$	Elasticity	Deformation	Stress	Young's Modulus

Methodology to find derivative numerically-

Using Taylor series expansion,

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \cdots$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''(x_i)}{2!}h + O(h^2)$$

Also,

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

Similarly,

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

Table for different order of accuracy-

First Derivative	Error
$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$	$O(h)$
$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$	$O(h^2)$
Second Derivative	
$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$	$O(h)$
$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2}$	$O(h^2)$
Third Derivative	
$f'''(x_i) = \frac{f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)}{h^3}$	$O(h)$
$f'''(x_i) = \frac{-3f(x_{i+4}) + 14f(x_{i+3}) - 24f(x_{i+2}) + 18f(x_{i+1}) - 5f(x_i)}{2h^3}$	$O(h^2)$
Fourth Derivative	
$f''''(x_i) = \frac{f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)}{h^4}$	$O(h)$
$f''''(x_i) = \frac{-2f(x_{i+5}) + 11f(x_{i+4}) - 24f(x_{i+3}) + 26f(x_{i+2}) - 14f(x_{i+1}) + 3f(x_i)}{h^4}$	$O(h^2)$

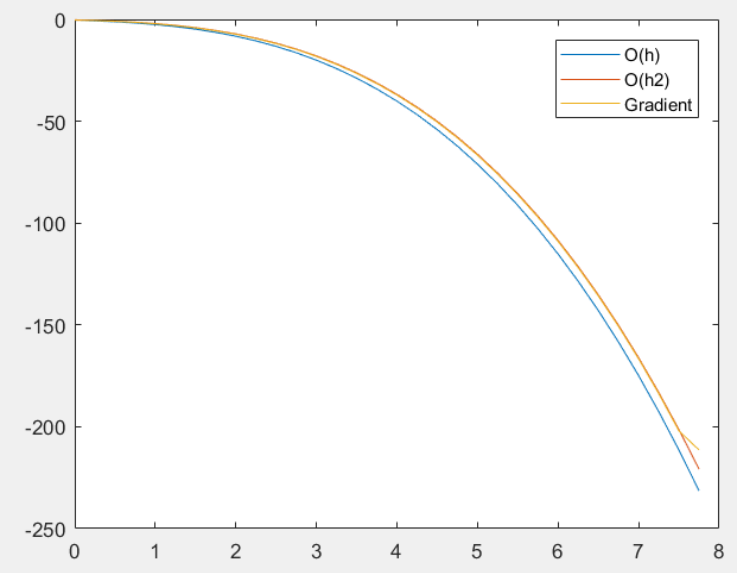
Problem Statement.

Estimate the derivative of for *O(h), O(h²)* for forward differencing-

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$$

at $x = 0.5$ using finite-differences and a step size of $h = 0.25$.

```
clear all;clc;
xl = input('Enter lower limit of X: ');xu = input('Enter upper limit of X: ');
h = input('Enter step size of X: ');n = xu-xl/h;
i = 1;
x(1) = xl;
while x<xu
    a = x(i)+ 2*h;
    b = x(i)+ h;
    c = x(i);
    df1(i) = (f(b)-f(c))/h;
    df2(i) = (-f(a) + 4*f(b) - 3*f(c))/(2*h);
    y(i) = f(c);
    xp(i) = x(i);
    x(i+1) = x(i) +h;
    i = i+1;
end
dfg = gradient(y,0.25);
plot(xp(),df1())
hold on
plot(xp(),df2())
hold on
plot(xp(),dfg())
legend('O(h)','O(h2)','Gradient')
function fun = f(x)
    fun = -0.1*(x^4) -0.15*(x^3) -0.5*(x^2) -0.25*x +1.2;
end
```



Assignment -

One numerical as suggested by Faculty.

Sample numerical.

Develop an M-file to obtain first-derivative estimates for unequally spaced data. Test it with the following data:

x	0.6	1.5	1.6	2.5	3.5
$f(x)$	0.9036	0.3734	0.3261	0.08422	0.01596

where $f(x) = 5e^{-2x}\ x$. Compare your results with the true derivatives.

LAB 9: Numerical Integration

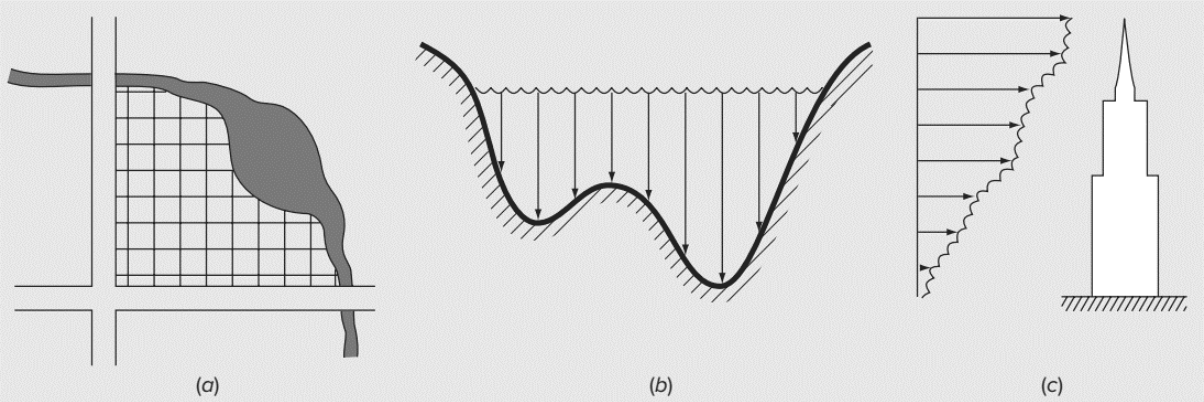
Objective: To numerically obtain the integration of any given function.

Theory: Mathematically, definite integration is represented by

$$I = \int_a^b f(x)\,dx$$

which stands for the integral of the function $f(x)$ with respect to the independent variable x, evaluated between the limits $x = a$ to $x = b$.

The meaning of integration is the total value, or summation, of a f (x) dx over the range x = a to b. In fact, the symbol ∫ is actually a stylized capital S that is intended to signify the close connection between integration and summation.



Examples of how integration is used to evaluate areas in engineering and scientific applications. (*a*) A surveyor might need to know the area of a field bounded by a meandering stream and two roads. (*b*) A hydrologist might need to know the cross-sectional area of a river. (*c*) A structural engineer might need to determine the net force due to a nonuniform wind blowing against the side of a skyscraper.

Method to numerically evaluate integration-

TRAPEZOIDAL RULE –

The *Newton-Cotes formulas* are the most common numerical integration schemes. They are based on the strategy of replacing a complicated function or tabulated data with a polynomial that is easy to integrate:

$$I = \int_a^b f(x)\,dx \cong \int_a^b f_n(x)\,dx$$

where $f_n(x)$ = a polynomial of the form

$$f_n(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n$$

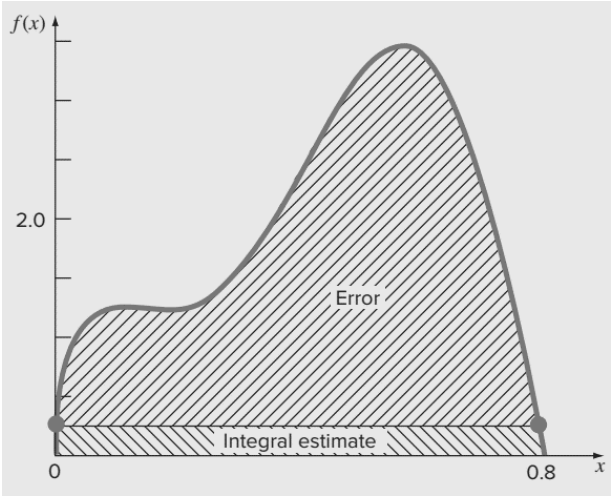
The trapezoidal rule is the first of the Newton-Cotes closed integration formulas.

$$I = \int_a^b \left[f(a) + \frac{f(b) - f(a)}{b - a} (x - a) \right] dx$$

The result of the integration is

$$I = (b - a) \frac{f(a) + f(b)}{2} \qquad I = \text{width} \times \text{average height}$$

which is called the *trapezoidal rule*.

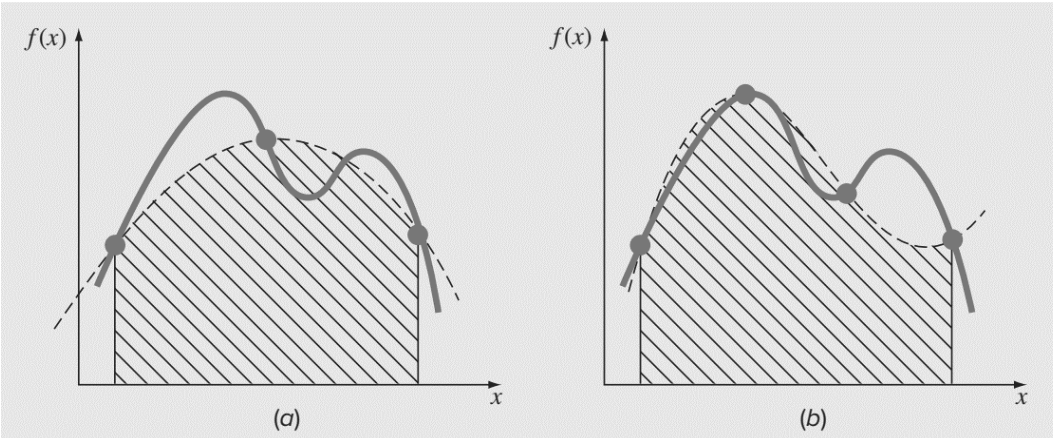


Error estimate

Simpson's 1/3 Rule –

Simpson's 1/3 rule corresponds to the case where the polynomial

(*a*) Graphical depiction of Simpson's 1/3 rule: It consists of taking the area under a parabola connecting three points. (*b*) Graphical depiction of Simpson's 3/8 rule: It consists of taking the area under a cubic equation connecting four points.



$$I = \int_{x_0}^{x_2} \left[\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \right.$$

$$\left. + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2) \right] dx$$

$$I = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)]$$

where, for this case, $h = (b - a)/2$. This equation is known as *Simpson's 1/3 rule*.

$$I = (b - a) \frac{f(x_0) + 4f(x_1) + f(x_2)}{6}$$

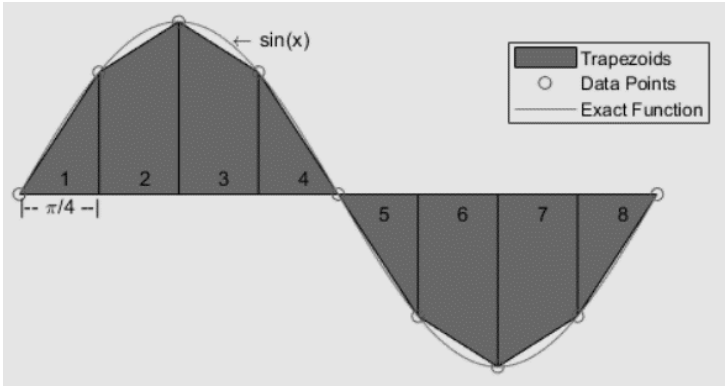
where $a = x_0$, $b = x_2$, and x_1 = the point midway between a and b, which is given by $(a + b)/2$.

Problem Statement. Use Eq. (19.23) to integrate

$$f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$$

from $a = 0$ to $b = 0.8$ using analytical, Trapezoidal and 1/3 simpson rule.

Segments (n)	Points	Name	Formula	Truncation Error
1	2	Trapezoidal rule	$(b - a) \frac{f(x_0) + f(x_1)}{2}$	$-(1/12)h^3 f''(\xi)$
2	3	Simpson's 1/3 rule	$(b - a) \frac{f(x_0) + 4f(x_1) + f(x_2)}{6}$	$-(1/90)h^5 f^{(4)}(\xi)$
3	4	Simpson's 3/8 rule	$(b - a) \frac{f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)}{8}$	$-(3/80)\, h^5 f^{(4)}(\xi)$



`trapz` performs numerical integration via the trapezoidal method. This method approximates the integration over an interval by breaking the area down into trapezoids with more easily computable areas. For example, here is a trapezoidal integration of the sine function using eight evenly-spaced trapezoids:

For an integration with `N+1` evenly spaced points, the approximation is

$$\int_a^b f(x)dx \approx \frac{b-a}{2N}\sum_{n=1}^{N}(f(x_n)+f(x_{n+1}))$$

$$= \frac{b-a}{2N}[f(x_1)+2f(x_2)+\ldots+2f(x_N)+f(x_{N+1})]\ ,$$

where the spacing between each point is equal to the scalar value $\frac{b-a}{N}$. By default MATLAB® uses a spacing of 1.

If the spacing between the `N+1` points is not constant, then the formula generalizes to

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{n=1}^{N}(x_{n+1}-x_n)[f(x_n)+f(x_{n+1})]\ ,$$

where $a = x_1 < x_2 < \ \ldots\ < x_N < x_{N+1} = b$, and $(x_{n+1}-x_n)$ is the spacing between each consecutive pair of points.

MATLAB code for evaluating area under the sin curve.

```
clc;clear all; close all;
X = 0:pi/100:pi;
Y = sin(X);
Q = trapz(X,Y);
```

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

19.4 Evaluate the following integral:

$$\int_{-2}^{4}(1-x-4x^3+2x^5)\,dx$$

(a) analytically, **(b)** single application of the trapezoidal rule, **(c)** composite trapezoidal rule with $n = 2$ and 4, **(d)** single application of Simpson's 1/3 rule, **(e)** Simpson's 3/8 rule, and

LAB 10: Ordinary Differential Equation

Objective: To numerically obtain the governing ODE of any physical system and solve it numerically.

Theory: equations, which are composed of an unknown function and its derivatives, are called differential equations. When the function involves one independent variable, the equation is called an *ordinary differential equation* (or ODE). Consider an ordinary differential equations of the form,

$$\frac{dy}{dt} = f(t, y)$$

Euler's Method –

New value = old value + slope × step size

in mathematical terms,

$$y_{i+1} = y_i + \phi h$$

where the slope ϕ is called an increment function. According to this equation, the slope estimate of ϕ is used to extrapolate from an old value yi to a new value yi+1 over a distance h. This formula can be applied step by step to trace out the trajectory of the solution into the future.

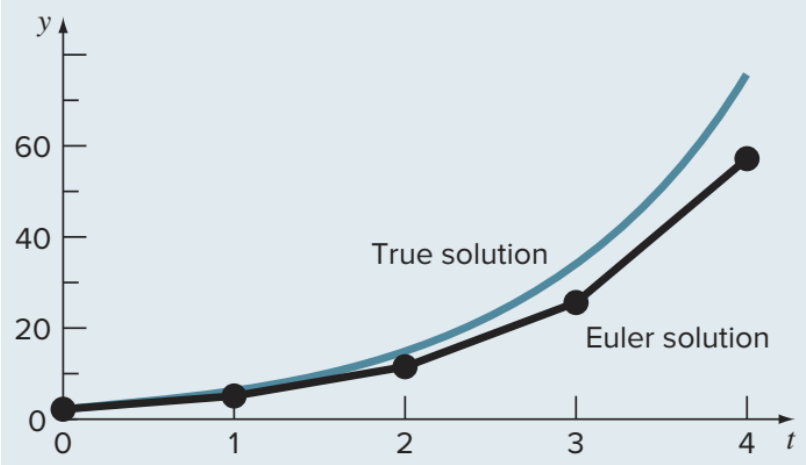
The first derivative provides a direct estimate of the slope at t_i

$\phi = f(t_i, y_i)$
where $f(t_i, y_i)$ is the differential equation evaluated at t_i and y_i.
$y_{i+1} = y_i + f(t_i, y_i)h$
This formula is referred to as *Euler's method* (or the Euler-Cauchy or point-slope method). A new value of y is predicted using the slope (equal to the first derivative at the original value of t) to extrapolate linearly over the step size h.

Problem Statement. Use Euler's method to integrate $y' = 4e^{0.8t} - 0.5y$ from $t = 0$ to 4 with a step size of 1. The initial condition at $t = 0$ is $y = 2$. Note that the exact solution can be determined analytically as

$$y = \frac{4}{1.3}(e^{0.8t} - e^{-0.5t}) + 2e^{-0.5t}$$

t	y_{true}	y_{Euler}	$\lvert\varepsilon_t\rvert$ (%)
0	2.00000	2.00000	
1	6.19463	5.00000	19.28
2	14.84392	11.40216	23.19
3	33.67717	25.51321	24.24
4	75.33896	56.84931	24.54



Assignment Problem –

Assignment –

Read about Modified Euler method and Attempt a numerical based on it.

One numerical as suggested by Faculty.

Sample numerical.

Solve the following initial value problem over the interval from $t = 0$ to where $y(0) = 1$. Display all your results on the same graph.

$$\frac{dy}{dt} = yt^2 - 1.1y$$

(a) Analytically.
(b) Using Euler's method with $h = 0.5$ and 0.25.

LAB 11: Partial Differential Equation

Objective: To numerically obtain the governing PDE of any physical system and solve it numerically.

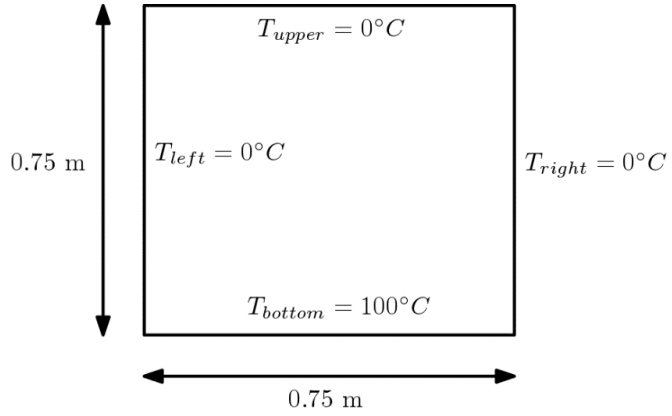
Theory:

Theory - Consider a 2D metal plate with dimensions 0.75x0.75 m2 is kept at 100 degree C at one of it end and rest of the ends are initially at 0 degree C as shown in the figure.

The transient Fourier law of heat conduction for 2D metal plate is given by,

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

Where T(x,y,t) is the temperature at any location and time and x, y, t are space and time coordinates. Assume the thermal conductivity of plate is K = 1 W/m degree C.



Take the spatial discretization as delta x = delta y = 0.025 m.

Using explicit finite difference method, the discrete equation can be obtained using Taylor series expansion as below,

$$\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} = k\left(\frac{T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}}{\Delta x^2} + \frac{T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n}}{\Delta y^2}\right)$$

Suppose $\Delta x = \Delta y = \Delta h$

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \frac{k\Delta t}{\Delta h^2}(T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n})$$

Take the spatial discretization as $\Delta x = \Delta y = \Delta h = 0.025$ m and $\Delta t = 0.00015$, thermal conductivity of plate is K = 1 W/m degree C.

MATLAB program-

```
% Explicit method for 2D heat equation in flat plate
close all; clear all; clc;
L =0.75;
dx = 0.05;
```

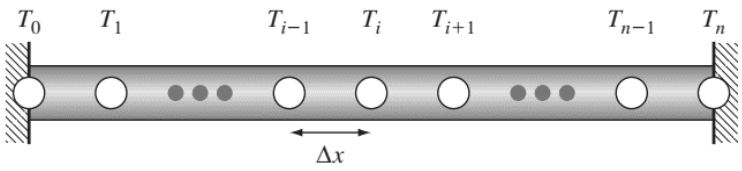
```
dy=0.05;
N = L/dx +1;
x = linspace(0,L,N);
y = linspace(0,L,N);
dt = 0.00015;
epsilon = 1e-4;
T_new = zeros(N,N);
 %boundary condition
      T_new(1,:) =100;
      T_new(:,1) =0;
      T_new(N,:)=0;
      T_new(:,N)=0;

error =1;iter=0;
while(error > epsilon)
    iter=iter+1;
      T = T_new;% update T every iteration
   for i=2:N-1
      for j =2:N-1
         T_new(i,j) = dt*((T(i+1,j)-2*T(i,j)+T(i-1,j))/dx^2 +(T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2) +
T(i,j);

      end
   end
   error = max(max(abs(T-T_new)));

figure(2);
hold on;
pause(0.001);
%     contourf(T_new);
    shading flat;colorbar;
    contourf(x,y,T_new,'ShowText','on')
colorbar;
xlabel('x meter');ylabel('y meter')

end
```

Assignment -

One numerical as suggested by Faculty.

Sample numerical.

Consider 1D heat conduction in metal rod as below.



1D heat conduction governing equation is given below,

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2}\right)$$

Write down the discretized equation is explicit finite difference method and obtain the Temperature profile T(x,t). Take length of rod is 0.75 m, Δx= 0.025 m, $\Delta t = 0.00015,$ thermal conductivity of plate is K = 1 W/m degree C.