Laboratory Manual

Numerical Techniques Laboratory

EEE/ECE 282 (V3)

Department of Electrical & Electronic Engineering (EEE)
School of Engineering (SoE)
Brac University

Revision: July 2022

A. Course Objectives:

The objectives of this course are to:

a. Introduce an understanding of the core ideas and concepts of Numerical Methods.
b. Provide students with sound understanding and knowledge of programming and efficient coding to implement different numerical methods and concepts.

B. Course Outcomes, CO-PO-Taxonomy Domain & Level- Delivery-Assessment Tool:

Sl.	CO Description	POs	Bloom's taxonomy domain/level	Delivery methods and activities	Assessment tools
CO1	**Apply** numerical methods for various mathematical operations and tasks, such as roots of equations, curve fitting techniques, integration, differentiation, solution of linear and nonlinear system and differential equations.	a	Cognitive/ Apply	Lecture, Notes, Lab class	Assignment, Exam
CO2	**Evaluate** the applicability of common numerical methods for the solution of engineering problem.	b	Cognitive/ Evaluate	Lecture, Notes, Lab class	Assignment
CO3	**Use** appropriate simulation tools to perform experiments on various numerical methods.	e	Cognitive/ Apply Psychomotor/ Precision	Lab Class, Lectures, Tutorial	Assignment, Exam

C. Mark Distribution

Tentative Marks Distribution	
Assessment	**Weightage**
Attendance	*10*
Assignment	*25*
Lab Report	*20*
Simulation Exam	*25*
Final Quiz (MCQ)	*20*
Total	**100**

D. Course Materials

Text and Reference Books:

Sl.	Title	Author(s)	Publication Year	Edition	Publisher	ISBN
01	Numerical Methods for Engineers	Steven C. Chapra & Raymond C. Canale	2014	7th	McGraw-Hill Education	13: 978-0073397924
02	Elementary Numerical Analysis	Kendall Atkinson & Weimin Han	2003	3rd	Wiley	13: 978-0471433378

Lab Safety and Security Issues

1. *Laboratory Safety Rules (General Guidelines):*

The Department of EEE maintains general safety rules for laboratories. The guideline is attached in front of the door in each of the laboratories. The written rules are as follows.

a) Closed shoes must be worn that will provide full coverage of the feet and appropriate personnel clothing must be worn.
b) Always check if the power switch is off before plugging in to the outlet. Also, turn the instrument or equipment OFF before unplugging from the outlet.
c) Before supplying power to the circuit, the connections and layouts must be checked by the teacher.
d) Voltage equal or above 50V are always dangerous. Therefore, extra precautions must be taken as voltage level is increased.
e) Extension cords should be used only when necessary and only on a temporary basis.
f) Once the lab exercise is done, all equipment must be powered down and all probes, cords and other instruments must be returned to their proper position.
g) In case of fire, disconnect the electrical mains power source if possible.
h) Students must be familiar with the locations and operations of safety and emergency equipment like Emergency power off, Fire alarm switch and so on.
i) Eating, drinking, chewing gum inside electrical laboratories are strictly prohibited.
j) Do not use damaged cords or cords that become too hot or cords with exposed wiring and if something like that is found, inform the teacher/LTO right away.
k) No laboratory equipment can be removed from their fixed places without the teacher/LTO's authorization.
l) No lab work must be performed without the laboratory teacher/lab technical officer being present.

2. Electrical Safety

To prevent electrical hazards, there are symbols in front of the Electrical Distribution Board, High voltage three phase lines in the lab, Backup generator and substation. Symbols related to Arc Flash and Shock Hazard, Danger: High Voltage, Authorized personnel Only, no smoking etc. are posted in required places. Only authorized personnel are allowed to open the distribution boxes.

3. Electrical Fire:

If an electrical fire occurs, try to disconnect the electrical power source, if possible. If the fire is small, you are not in immediate danger, use any type of fire extinguisher except water to extinguish the fire. When in doubt, push in the Emergency Power Off button.

4. IMPORTANT:

Do not use water on an electrical fire.

Laboratory Plan

Class Week	Experiment Name	Page	Tentative schedule	Is this experiment used for any CO assessment?	
				Yes	No
1	Introductory Session	7-8	1st week		
2	Introduction of MATLAB: Familiarizing with MATLAB	9-21	2nd week		✔
3	Introduction of MATLAB: Programming in M-files	22-25	3rd week		✔
4	Numerical Differentiation Techniques	26-31	4th week	✔	
5	Numerical Integration Techniques	32-37	5th week	✔	
6	Curve Fitting: Linear & Polynomial Regression	38-47	6th Week	✔	
7	Curve Fitting: Linear & Polynomial Interpolation	48-56	7th Week	✔	
8	Solution of Simultaneous Linear Equations	67-65	8th Week	✔	
9	Solution to Non-Linear Equations	66-72	9th Week	✔	
10	Linear Ordinary Differential Equations	73-77	10th week	✔	
11	Exam		11th week	✔	
12	Project Submission		12th week		

Updated by:

1. **Tasfin Mahmud**
2. **Md. Mehedi Hasan Shawon**

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-00: Introductory Session

During the lab session of the first week the following topics should be discussed:

- Introducing the course materials, course outcomes, marks distribution, assessment plan
- Instructions regarding Group formation
- Lab Report submission guideline
- Software Installation procedure

1. **Introducing the course materials, course outcomes, marks distribution, assessment plan:**
   The list of experiments should be provided to the students and also an overview of the topics can be given. The course outcomes should be explained so that the students can have a clear understanding of what they are expected to to learn from this course. The complete marks distribution and the assessment plan should be discussed as well.

2. **Instructions:**
   The following instructions could be provided to the students for the group formation purpose:
   a. Each group should have at most three members (as this is a simulation-based lab).
   b. A group should be formed with the students having the same course code.
   c. All group members of a group should belong to the same section (exceptions can be made depending upon the situation).

3. **Lab Report submission guideline:**
   General guidelines for submitting the report could be as follows:
   a. Each group will submit one report.
   b. The hard copy of the report should be submitted before the next lab session (a soft copy should also be submitted on the online platform, if necessary).
   c. All group members should contribute equally. If any member is not cooperating, you may remove his/her name from the lab report and send an email to the instructor informing the issue.
   d. Late submission will not be accepted.
   e. Plagiarism will be treated harshly.

f. A cover page should be added with the report including the course code, course name, experiment no., experiment name, name and ID of each of the group members.
g. Comments should be used inside the code to make it understandable and a proper explanation of the workings should be also provided. Also, necessary screenshots of the outputs for each of the codes should be attached.
h. The report would be assessed in terms of comprehensiveness, cleanliness, and overall presentation.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-01: Introduction of MATLAB: Familiarizing of MATLAB

Objective

In this first lab, we will learn how to:

- Perform various mathematical operations on simple variables, vectors, matrices and complex numbers using MATLAB.
- Indexing array and allocating memory
- Familiarizing with special characters and functions, control flow and operators
- Generating polynomials and plotting sinusoids.

Software and Device requirements:

MATLAB software

Minimum PC specifications:

Windows/ Mac: Microsoft® Windows® 7 Professional, Enterprise, Ultimate or Home Premium (64-bit); Windows 8 (64-bit) (All Service Packs); Windows 10 (64-bit); Windows 2008 R2 Server; Windows 2012 Server (All Service Packs).
Ram: 2 GB
Processor: Intel® Pentium® 4 or AMD Athlon XP 2000 with multi-core CPU
Display resolutions: 1,024 x 768 display resolution with true color (16-bit color)

Procedure:

PART - A

1. Warm-Up:

MATLAB is a high-level programming language that has been used extensively to solve complex engineering problems.

MATLAB works with three types of windows on your computer screen. These are the

1. Command window
2. The Figure window
3. The Editor window

- The Figure window only pops up whenever you plot something.
- The Editor window is used for writing and editing MATLAB programs (called M-Files) and can be invoked in Windows from the pull-down menu after selecting File | New | M-file. In UNIX, the Editor window pops up when you type in the command window: edit filename ('filename' is the name of the file you want to create).
- The command window is the main window in which you communicate with the MATLAB interpreter. The MATLAB interpreter displays a command >> indicating that it is ready to accept commands from you.

• View the MATLAB introduction by typing

```
>> intro
```

at the MATLAB prompt. This short introduction will demonstrate some basic MATLAB commands.

• Explore MATLAB's help capability by trying the following:

```
>> help
>> help plot
>> help ops
>> help arith
```

• Type demo and explore some of the demos of MATLAB commands.

• You can use the command window as a calculator, or you can use it to call other MATLAB programs (Mfiles).

Say you want to evaluate the expression $a^3 + \sqrt{bd} - 4c$, where $a = 1.2$, $b = 2.3$, $c = 4.5$ and $d = 4$. Then in the command window, type:

```
>> a = 1.2;
>> b=2.3;
>> c=4.5;
>> d=4;
>> a^3+sqrt(b*d)-4*c

ans =
-13.2388
```

Note the semicolon after each variable assignment. If you omit the semicolon, then MATLAB echoes back on the screen the variable value.

2. Arithmetic Operations:

a) There are four different arithmetic operators:

- + addition

- - subtraction
- * multiplication
- / division (for matrices it also means inversion)

b) There are also three other operators that operate on an element by element basis:

- .* multiplication of two vectors element by element
- ./ division of two vectors, element-wise
- .^ raise all the elements of a vector to a power.

c)

Suppose that we have the vectors $x = [x_1, x_2, ..., x_n]$ and. Then

$$x.*y = [x_1 y_1, x_2 y_2, ..., x_n y_n]$$
$$x./y = [x_1 / y_1, x_2 / y_2, ..., x_n / y_n]$$
$$x.\hat{}\, p = [x_1{}^p, x_2{}^p, ..., x_n{}^p]$$

The arithmetic operators + and — can be used to add or subtract matrices, scalars or vectors.

By vectors we mean one-dimensional arrays and by matrices we mean multi-dimensional arrays. This terminology of vectors and matrices comes from Linear Algebra.

Example:

```
>> X=[1,3,4]
>> Y=[4,5,6]
>> X+Y
ans=
      5 8 10
```

d)

For the vectors X and Y the operator + adds the elements of the vectors, one by one, assuming that the two vectors have the same dimension. In the above example, both vectors had the dimension 1 × 3, i.e., one row with three columns.

An error will occur if you try to add a 1 × 3 vector to a 3 × 1 vector. The same applies for matrices.

e)

To compute the dot product of two vectors you can use the multiplication operator * For the above example, it is:

```
>> X*Y'
   ans =
         43
```

Note the single quote after Y. The single quote denotes the transpose of a matrix or a vector.

f)

To compute an element by element multiplication of two vectors (or two arrays), you can use the .* operator:

```
>> X .* Y
ans =
4 15 24
```

That is, X.*Y means [1×4, 3×5, 4×6] = [4 15 24]. The .* operator is used very often (and is highly recommended) because it is executed much faster compared to the code that uses for loops.

3. Complex numbers:

MATLAB also supports complex numbers. The imaginary number is denoted with the symbol i or j, assuming that you did not use these symbols anywhere in your program (that is very important!). Try the following:

```
>> z=3 + 4i % note that you do not need the '*' after 4
>> conj(z) % computes the conjugate of z
>> angle(z) % computes the phase of z
>> real(z) % computes the real part of z
>> imag(z) % computes the imaginary part of z
>> abs(z) % computes the magnitude of z
```

4. Array indexing:

In MATLAB, all arrays (vectors) are indexed starting with 1, i.e., y(1) is the first element of the array y. Note that the arrays are indexed using parenthesis (.) and not square brackets [.] as in C/C++. To create an array having as elements the integers 1 through 6, just enter:

```
>> x=[1,2,3,4,5,6]
```

Alternatively, you can use the : notation,

```
>> x=1:6
```

The colon(":") notation above creates a vector starting from 1 to 6, in steps of 1. If you want to create a vector from 1 to 6 in steps of say 2, then type:

```
>> x=1:2:6
Ans =
1 3 5
```

Try the following code:

```
>> a=2:4:17
>> b=20:-2:0

>> a=2:(1/10):4
```

Extracting or inserting numbers in a vector can be done very easily. To concatenate an array, you can use the [ ] operator, as shown in the example below:

```
>> x=[1:3 4 6 100:110]
```

To access a subset of the array, try the following:

```
>> x(3:7)
>> length(x) % gives the size of the array or vector
>> x(2:2:length(x))
```

5. Allocating memory:

You can allocate memory for one-dimensional arrays (vectors) using the zeros command. The following command allocates memory for a 100-dimensional array:

```
>> Y=zeros(100,1);
>> Y(30)

ans =
0
```

Similarly, you can allocate memory for two-dimensional arrays (matrices). The command

```
>> Y=zeros(4,5)
```

defines a 4 by 5 matrix. Similar to the zeros command, you can use the command ones to define a vector containing all ones,

```
>> Y=ones(1,5)
ans=
1 1 1 1 1
```

6. Special characters and functions

Symbol Meaning

- pi ð(3.14...)
- sqrt indicates square root e.g., sqrt(4)=2
- ˆ indicates power(e.g., 3ˆ2=9)
- abs Absolute value | .| e.g., abs(-3)=3
- ; Indicates the end of a row in a matrix. It is also used to suppress printing on the screen (echo o.)
- % Denotes a comment. Anything to the right of % is ignored by the MATLAB interpreter and is considered as comments
- ' Denotes transpose of a vector or matrix. It's also used to define strings, e.g.,str1='DSP';

Some special functions are given below:

`length(x)` - gives the dimension of the array x
`size(x)` - Finds the size of array x.

Example:

```
>> x=1:10;
```

```
>> length(x)
ans =
10
```

The function `size` works similarly. But the output is slightly different.

Example:

```
>> x=2:8;
>> size(x)
ans =
1   7
```

Here, 1 in the answer is the number of rows in the vector x, while 7 denotes the number of columns.

PART - B

1. Control flow:

MATLAB has the following flow control constructs:

- if statements
- switch statements

- for loops
- while loops
- break statements

These if, for, switch and while statements need to be terminated with an end statement.

a If

) The general form of a simple **if-else** statement is:

```
if relation
   statements
end
```

The general form of a simple **if-elseif-else** statement is:

```
if relation
     statements
elseif relation
     statements
else
     statements
end
```

Statement syntax	Example

`if <case>`
`  <do this>`
`elseif <case>`
`  <do this>`
`else`
`  <do this>`
`end`
	`if x > 10`
	`  y = y + 1;`
	`elseif x > 5`
	`  y = y - 1;`
	`else`
	`  y = y - 4;`
	`end`

Example:

```
x=-3;
if x>0
      str='positive'
  elseif x<0
      str='negative'
    elseif x= = 0
       str='zero'
else
   str='error'
end
```

What is the value of "str" after execution of the above code?

b) While statement:

The general form of a while loop is:

```
while relation
      statements
 end
```

Statement syntax	Example

`while <case>`
`  <do this>`
`end`
	`while x < 20`
	`  y = y/3;`
	`  x = x + 1;`
	`end`

Example:

```
x=-10;
while x<0
x=x+1;
end
```

What is the value of x after execution of the above loop?

c) For Loop:

Statement syntax	Example

`for <variable = statement>`
`  <do this>`
`end`
	`for n = 1:1:10`
	`  x(n) = 2*n;`
	`end`

Example:

```
X=0;
for i=1:10
X=X+1;
end
```

The above code computes the sum of all numbers from 1 to 10.

d) Break:

The break statement lets you exit early from a for or a while loop:

```
x=-10;
while x < 0
      x=x+2;
  if x = = -2
  break;
  end
end
```

MATLAB supports the following relational and logical operators:

2. Relational Operators:

Symbol	Meaning
<=	Less than equal
<	Less than
>=	Greater than equal
>	Greater than
==	Equal
~=	Not equal

3. Logical Operators:

Symbol	Meaning
&	AND
|	OR
~ (tilde)	NOT

4. Polynomials:

Polynomials in MATLAB are represented by arrays. The usual representation is that the elements in an array are the coefficients of the polynomial starting from the highest order term to the lowest order term. If a term is not present then its coefficient is entered as 0. An example is:

$y = 5x^3 + 9x + 1$ is represented by

```
>> y = [5 0 9 1]
```

The other way to represent polynomials is via their roots. In MATLAB these are also represented by arrays. The polynomial:

$y = (x+3)(x-5)(x+9)$ is represented by

```
>> y = [-3 5 -9];
```

Two very useful commands are roots and poly.

- The command roots() will factorize a polynomial into its roots and return the roots in an array.
- The function poly() does the opposite. It takes in the roots of a polynomial and returns the coefficients of the polynomial.

5. Generating sinusoidal waves:

Example

To generate a sine wave $y = 5\sin(2\pi 10t)$

```
f = 10;
A = 5;
t = 0:0.001:1;
y = A*sin(2*pi*f*t)
```

6. Plotting:

You can plot arrays using MATLAB's function plot. The function plot (.) is used to generate line plots. The function stem(.) plots every point of the array without connecting them with a smooth line. More generally, plot(X,Y) plots vector Y versus vector X. Various line types, plot symbols and colors may be obtained using plot(X,Y,S) where S is a character string indicating the color of the line, and the type of line (e.g., dashed, solid, dotted, etc.). Examples for the string S include:

```
r red + plus -- dashed
g green * star
b blue s square
```

- You can insert x-labels, y-labels and title to the plots, using the functions `xlabel(.)`, `ylabel(.)` and `title(.)` respectively.
- To plot two or more graphs on the same figure, use the command **subplot**.
- For instance, to show the above sine wave, type: `>> plot(t,y)`
- Again, using the stem command: `>> stem(t,y)`
- To plot the two figures in the same plot, type:

```
>> subplot(2,1,1), plot(y)
>> subplot(2,1,2), stem(y)
```

The (m,n,p) argument in the subplot command indicates that the figure will be split in m rows and n columns. The 'p' argument takes the values 1, 2, . . . , m× n. In the example above, m = 2, n = 1,and, p = 1 for the top figure and p = 2 for the bottom figure.

To get more help on plotting, type: help plot or help subplot.

5. Class Task:

1. Define two 4 x 5 matrices, X and Y where X should consist of all different elements and Y should consist of all ones. Find out S, R, T, Q, V and W where, S is the sum of X and Y, R is the difference of X and Y T is the element by element product of X and Y Q is the element by element division of Y by X V is equal to X 2Y 3 W is the normal multiplication of X and Y matrices Store the numbers in the third column of V in another variable using MATLAB command (not manually)
2. Write the above code with nested if statements (without using elseif)
3. Initialize a variable with 6 and another with 21. When the 1st variable is greater than 5 then k equals that var. Otherwise, for values of the 1st var less than 1, k is 5 times of the 1 st var plus the 2nd var and for all other conditions k = any value of your choice.
4. Find all powers of 2 below 10000 and store them in an array.
5. Find 1 1 m for values of m from 1 to 100
6. Find n e 1 for values of n, starting from 100, ending at 0 with an increment of -2.

6. Assignment:

No. 1:

- Put the digits of your id's in a row vector.
- Define another vector which has elements 1 to 8.
- Multiply the two vectors of step 1 & 2 element by element to get the vector element_mult.
- Multiply the vector of step 1 & transpose of the vector of step 2 normally to get the vector norm_mult.
- Now multiply element_mult & norm_mult to find final_value.
- Find the square root of the final value and define this vector as root_fv
- Obtain the size of the vector element_mult to get the vector sz
- Use the last element of the id vector as the power of each element of root_fv. Thus find the vector p.

  Those having 0 as the last digit of id should use the 2nd last digit instead.
- Define a vector sum, which is the summation of vector p and the 2nd element of the size vector sz.
- Find the mean of the vector sum. Display all the values.

No. 2:

- Define a vector t which has values from 0 to 80 ms with an interval of 0.001
- Make two sine and one cosine waves with frequencies 15 & 30 & 60 and amplitudes 10, 2 & 5 respectively.
- Plot the two sine waves in two different figure windows.
- Again plot the two sine waves in the same figure window.
- Using the subplot command, plot all of them on the same window, but different plots.
- Using the subplot command, plot all the sine waves on the same figure and cosine in a different plot.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-02: Introduction of MATLAB: Programming in M-files

---

Objective

This part is intended to illustrate and generate M-files, also known as function files, in MATLAB.

Software Devices and Requirements

MATLAB software

Minimum PC specifications:

Windows/ Mac: Microsoft® Windows® 7 Professional, Enterprise, Ultimate or Home Premium (64-bit); Windows 8 (64-bit) (All Service Packs); Windows 10 (64-bit); Windows 2008 R2 Server; Windows 2012 Server (All Service Packs).
Ram: 2 GB
Processor: Intel® Pentium® 4 or AMD Athlon XP 2000 with multi-core CPU
Display resolutions: 1,024 x 768 display resolution with true color (16-bit color)

Procedure

MATLAB programming is done using M-files, i.e., files that have the extension .m. These files are created using a text editor. To open the text editor, go to the File pull-down menu, choose New, then M-file. After you type in the program, save it, and then call it from the command window to execute it.

FUNCTION FILES:

A function file is also an m-file, but here all the variables are local. A function file starts with a function definition line as:

```
function[output variables]=name of the function(input variables)
```

Examples:

Function definition line	file name
`function [r,h,f] = motion(x,y,t)`	`motion.m`
`function [theta] = angleTH(x,y)`	`angleTH.m`
`function [ ] = circle(r)`	`circle.m`

!!!!!!!!Caution!!!!!!!!

1. The 1st word in the function definition line function must be typed in all lower case. A common mistake is to type Function.
2. Here the function name must be the same as the file name without the m extension.

Example

: `function [r,h,f] = motion(x,y,t)` is the function definition line

`[a,b,c]=motion (x1,y1,z1);` The input variables x1, y1 and z1 must be defined before executing this command

`[a,b,c]=motion (rx, ry,[0:100]);` The input variables rx, ry must be defined before executing this command, the 3rd variable t is specified in the call statement.

`[a,b,c]=motion (2, 3.5,0.001]);` all input variables are specified in the call statement

`[radius,h]=motion(rx,ry);` call with partial list of input & output variables. The 3rd input variable must be assigned a default value inside the function if it is required in calculations. The output corresponds to the 1st two elements of the output list of motion.

Say for instance that you want to write a program to compute the average (mean) of a vector x. The program should take as input the vector x and return the average of the vector.

Steps: 1. You need to create a new file, called "average.m". Open the text editor by going to the File pull down menu, choose New, then M-file. Type the following in the empty file:

```
function y=average(x)
L=length(x);
sum=0;
for i=1:L
sum=sum+x(i);
end
y=sum/L; % the average of x
```

Remarks:

y — is the output of the function "average" x — is the input array to the function "average"

average — is the name of the function. It's best if it has the same name as the filename.

MATLAB files always need to have the extension .m

2. From the Editor pull-down menu, go to File | Save, and enter: average.m for the filename.
3. Go to the Command window to execute the program by typing:

```
>> x=1:100;
>> y=average(x)
ans =
50.5000
```

Note that the function average takes as input the array x and returns one number, the average of the array.
In general, you can pass more than one input argument and can have the function return multiple values. You can declare the function average, for instance, to return 3 variables while taking 4 variables as input with the following statement:

```
function [y1, y2, y3]=average(x1,x2,x3,x4)
```

In the command window it has to be invoked as:

```
>> [y1, y2, y3]=average(x1,x2,x3,x4)
```

Assignments

1) Calculate factorials:

- Write a function **factorial** to compute the **factorial n*!*** for any integer n.
- The input should be the number ***n*** and output should be ***n!.***
- You might have to use a ***for*** loop or a ***while*** loop to do this.

2) Find the sum of a geometric series:

- Write a function to compute the sum of a geometric series $1+r+r^2+r^3+.....+r^n$ for a given r & n.
- The input of the function must be r & n and the output must be the sum of the series.

3) Convert temperature:

- Write a function that outputs a table showing Celsius temperature and their corresponding Fahrenheit temperature.
- The input of the function should be two variables ti & tf, specifying the lower & the upper range of the table in Celsius.
- The output should be a two column matrix; the 1st column showing the temperature in Celsius from ti to tf in the increments of $1^{\circ}C$ and the 2nd column showing and the corresponding temperatures in Fahrenheit.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-03: Numerical Differentiation Techniques

Introduction:

We are familiar with the analytical method of finding the derivative of a function when the functional relation between the dependent variable y and the independent variable x is known. However, in practice, most often functions are defined only by tabulated data, or the values of y for specified values of x can be found experimentally. Also in some cases, it is not possible to find the derivative of a function by analytical method. In such cases, the analytical process of differentiation breaks down and some numerical processes have to be invented. The process of calculating the derivatives of a function by means of a set of given values of that function is called numerical differentiation. This process consists in replacing a complicated or an unknown function by an interpolation polynomial and then differentiating this polynomial as many times as desired.

Forward Difference Formula:

All numerical differentiation are done by expansion of Taylor series

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + \frac{f'''(x)h^3}{6} + \ldots\ldots\ldots\ldots\ldots\ldots(1)$$

From (1)

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \ldots\ldots\ldots\ldots\ldots\ldots(2)$$

Where, $O(h)$ is the truncation error, which consists of terms containing h and higher order in terms of h.

Lab Task 1:

Write the MATLAB code for numerical differentiation using Forward Difference Method.
Given $f(x) = e^x$, find $f'(1)$ using $h=10^{-1}, 10^{-2},,,,$ upto 10^{-6}. Find out the error in each case by comparing the calculated value with exact value.

Function generating code for Forward Difference method-

```
function value = forward_difference(f,x,h,o,i)
%f = derivative function
%x = the value at which the derivative has to be evaluated
%h = divided difference
%o = Order of the derivative
%i = takes the values either 1 or 2; 2 denotes more accurate
results

    if(o == 1 && i == 1)
        value = (f(x+h)-f(x))/h;
    elseif(o == 1 && i == 2)
        value = (-f(x + 2*h) + 4*f(x+h) - 3*f(x))/(2*h);
    elseif(o == 2 && i == 1)
        value = (f(x+2*h)-2*f(x+h)+f(x))/(h*h);
    elseif(o == 2 && i == 2)
        value = (-f(x+3*h)+ 4*f(x+2*h)-5*f(x+h)+2*f(x))/(h*h);
    elseif(o == 3 && i == 1)
        value = (f(x+3*h) - 3*f(x+2*h) + 3*f(x+h) - f(x))/(h^3);
    elseif(o == 3 && i == 2)
        value = (-3*f(x+4*h) + 14*f(x+3*h) - 24*f(x+2*h) +
18*f(x+h) - 5*f(x))/(2*(h^3));
    elseif(o == 4 && i == 1)
        value = (f(x+4*h) - 4*f(x+3*h) + 6*f(x+2*h) - 4*f(x+h) +
f(x))/(h^4);
    elseif(o == 4 && i == 2)
        value = (-2*f(x+5*h) + 11*f(x+4*h) - 24*f(x+3*h) +
26*f(x+2*h) - 14*f(x+h) + 3*f(x))/(h^4);
    end

end
```

Central Difference Formula (of order O (h^2)):

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + \frac{f'''(c_1)h^3}{6} \ldots\ldots\ldots (3)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)h^2}{2} - \frac{f'''(c_2)h^3}{6} \ldots\ldots\ldots (4)$$

Using (3) and (4)

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \ldots\ldots\ldots (5)$$

Where, $O(h^2)$ is the truncation error, which consists of terms containing h^2 and higher order terms of h.

Lab Task 2:

Write the MATLAB code for numerical differentiation using Central Difference Method.
Given f(x) = sin (cos (1/x)) find $f'(1\sqrt{2})$ using $h=10^{-1}, 10^{-2},,,,$ up to 10^{-6}. Find out the error in each case by comparing the calculated value with exact value.

Function generating code for Central Difference method-

```
function value = central_difference(f,x,h,o,i)
%f = derivative function
%x = the value at which the derivative has to be evaluated
%h = divided difference
%o = Order of the derivative
%i = takes the values either 1 or 2; 2 denotes more accurate
results

    if(o == 1 && i == 1)
        value = (f(x+h)-f(x-h))/(2*h);
    elseif(o == 1 && i == 2)
        value = (-f(x + 2*h) + 8*f(x+h) - 8*f(x-h) +
f(x-2*h))/(12*h);
    elseif(o == 2 && i == 1)
        value = (f(x+h)-2*f(x)+f(x-h))/(h*h);
```

```
    elseif(o == 2 && i == 2)
        value = (-f(x+2*h) + 16*f(x+h) - 30*f(x) + 16*f(x-h) -
f(x-2*h))/(12*h*h);
    elseif(o == 3 && i == 1)
        value = (f(x+2*h) - 2*f(x+h) + 2*f(x-h) -
f(x-2*h))/(2*(h^3));
    elseif(o == 3 && i == 2)
        value = (-f(x+3*h) + 8*f(x+2*h) - 13*f(x+h) + 13*f(x-h) -
8*f(x-2*h) + f(x-3*h))/(8*(h^3));
    elseif(o == 4 && i == 1)
        value = (f(x+2*h) - 4*f(x+h) + 6*f(x) - 4*f(x-h) +
f(x-2*h))/(h^4);
    elseif(o == 4 && i == 2)
        value = (-f(x+3*h) + 12*f(x+2*h) + 39*f(x+h) + 56*f(x) -
39*f(x-h) + 12*f(x-2*h) + f(x-3*h))/(6*(h^4));
    end

end
```

Central Difference Formula (of order O (h^4)):

Using Taylor series expansion it can be shown that

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4) \ldots\ldots\ldots(6)$$

Here the truncation error reduces to h^4

Richardson's Extrapolation:

We have seen that

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

Which can be written as

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} + Ch^2$$

Or, $f'(x) \approx D_0(h) + Ch^2$ ……………………………….(7)

If step size is converted to 2h

$$f'(x) \approx D_0(2h) + 4Ch^2 \text{ ………………………….…….(8)}$$

Using (7) and (8)

$$f'(x) \approx \frac{4D_0(h) - D_0(2h)}{3} = \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} \text{ ……………….(9)}$$

Equation (9) is same as equation (6)

The method of obtaining a formula for $f'(x)$ of higher order from a formula of lower order is called extrapolation. The general formula for Richardson's extrapolation is

$$f'(x) = D_k(h) + O(h^{2k+2}) = \frac{4^k D_{k-1}(h) - D_{k-1}(2h)}{4^k - 1} + O(h^{2k+2}) \text{ ……(10)}$$

Practice Problems (Experiment - 03)

Problem-1: Derive the expressions of Forward Divided Difference (FDD), Backward Divided Difference (BDD), Central Divided Difference (CDD) of 1st order derivatives.

Problem-2: Write the MATLAB functions for FDD, BDD & CDD of 1st order derivatives using the expressions you derived in the previous problem.

Problem-3: For the given function:

$$f(x) = \frac{2x\sin(3x) + e^{-2x}}{x^{0.5}}$$

Determine the value of the 1st derivative of the function numerically for $x = 3.5$ with best possible accuracy.
Also, compare the numerical results of the derivatives with the analytical (by using the formulae of differentiation) results.

Problem-4: The following data are provided for the *displacement*, s of a moving object as a function of *time*, t:

$s\,(meter)$	0	4	8	12	16	20
t $(second)$	0	34.7	61.8	82.8	99.2	112.0

Determine the *velocity*, v of the object for each of the time instants. [Just to recall: $v = \frac{ds}{dt}$]

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-04: Numerical Integration Techniques

Introduction

There are two cases in which engineers and scientists may require the help of numerical integration techniques. (1) Where experimental data is obtained whose integral may be required and (2) where a closed form formula for integrating a function using calculus is difficult or so complicated as to be almost useless. For example the integral

$$\Phi(t) = \int_0^{t^3} \frac{1}{e^t - 1}\, dt.$$

Since there is no analytic expression for $\Phi(x)$, numerical integration techniques must be used to obtain approximate values of $\Phi(x)$.

Formulae for numerical integration called quadrature are based on fitting a polynomial through a specified set of points (experimental data or function values of the complicated function) and integrating (finding the area under the fitted polynomial) this approximating function. Any one of the interpolation polynomials studied earlier may be used.

Some of the Techniques for Numerical Integration Trapezoidal Rule

Assume that the values of a function $f(x)$ are given at x_1, x_1+h, x_1+2h ……x_1+nh and it is required to find the integral of $f(x)$ between x_1 and x_1+nh. The simplest technique to use would be to fit **straight lines** through $f(x_1)$, $f(x_1+h)$ ……and to determine the **area** under this approximating function as shown in Fig 7.1.

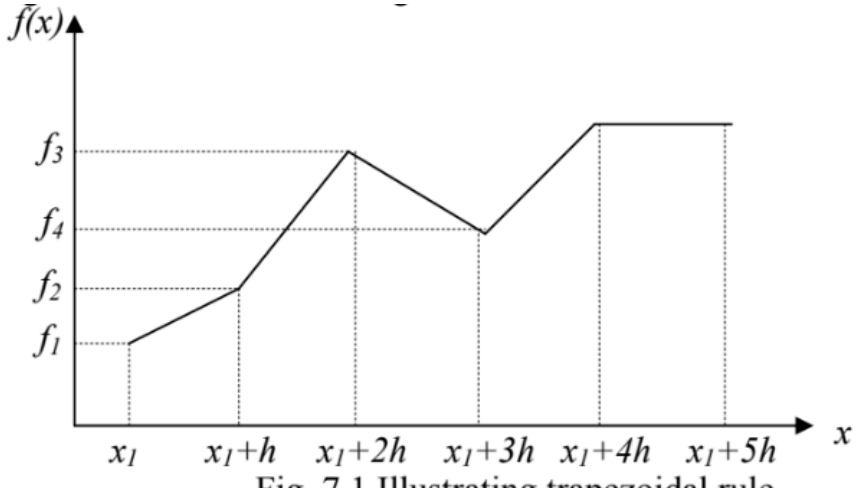


Fig. 7.1 Illustrating trapezoidal rule

For the first two points we can write:

$$\int_{x_1}^{x_1+h} f(x)dx = \frac{h}{2}(f_1 + f_2)$$

This is called first-degree Newton-Cotes formula.
From the above figure it is evident that the result of integration between x_1 *and* x_1+nh is nothing but the sum of areas of some trapezoids. In equation form this can be written as:

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1}^{n} \frac{(f_i + f_{i+1})}{2} h$$

The above integration formula is known as *Composite Trapezoidal rule.*
The composite trapezoidal rule can explicitly be written as:

$$\int_{x_1}^{x_1+nh} f(x)dx = \frac{h}{2}(f_1 + 2f_2 + 2f_3 + ...2f_n + f_{n+1})$$

Simpson's 1/3 Rule
This is based on approximating the function *f(x)* by fitting **quadratics** through sets of **three** points. For only three points it can be written as:

$$\int_{x_1}^{x_1+2h} f(x)dx = \frac{h}{3}(f_1 + 4f_2 + f_3)$$

This is called second-degree Newton-Cotes formula.
It is evident that the result of integration between x_1 *and* x_1+nh can be written as

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1,3,5,\ldots,n-1} \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2})$$

$$= \frac{h}{3}(f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + 4f_6 + \ \ldots\ 4f_n + f_{n+1})$$

In using the above formula it is implied that *f* is known at an **odd number of points** (***n*+1 is odd,** where n is the no. of subintervals).

Simpson's 3/8 Rule
This is based on approximating the function *f(x)* by fitting **cubic** interpolating polynomial through sets of **four** points. For only four points it can be written as:

$$\int_{x_1}^{x_1+3h} f(x)dx = \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + f_4)$$

This is called third-degree Newton-Cotes formula.
It is evident that the result of integration between x_1 *and* x_1+nh can be written as

$$\int_{x_1}^{x_1+nh} f(x)dx = \sum_{i=1,4,7,\ldots,n-2} \frac{h}{3}(f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3})$$

$$= \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + 2f_4 + 3f_5 + 3f_6 + 2f_7 + \ \ldots\ + 2f_{n-2} + 3f_{n-1} + 3f_n + f_{n+1})$$

In using the above formula it is implied that *f* is known at (*n+1*) points where **n is divisible by 3**.
An algorithm for integrating a **tabulated function** using composite trapezoidal rule:
Remarks: $f_1, f_2, \ldots\ldots\ldots, f_{n+1}$ are the tabulated values at $x_1, x_{1+h}, \ldots\ldots\ldots x_{1+nh}$ *(n+1* points*)*

1 Read *h*
2 *for i* = 1 to *n* +1 Read f_i *endfor*
3 *sum* ← $(f_1 + f_{n+1}) / 2$

```
4        for j = 2 to n do
5         sum ← sum + f_j
         endfor
6        int egral ← h . sum
7   write int egral  stop
```

Ex. 1. Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.8 using composite trapezoidal rule with (a) h=0.2, (b) h=0.4 and (c) h=0.6

Table 7.1

X	*f(x)*	*X*	*f(x)*
1.6	4.953	2.8	16.445
1.8	6.050	3.0	20.086
2.0	7.389	3.2	24.533
2.2	9.025	3.4	29.964
2.4	11.023	3.6	36.598
2.6	13.468	3.8	44.701

The data in Table 7.1 are for $f(x) = e^x$. Find the true value of the integral and compare this with those found in (a), (b) and (c).

Ex. 2. (a) Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.6 using Simpson's composite 1/3 rule.

(b) Integrate the function tabulated in Table 7.1 over the interval from x=1.6 to x=3.4 using Simpson's composite 3/8 rule.

An algorithm for integrating a **known function** using composite trapezoidal rule:

If *f(x)* is given as a closed form function such as $f(x) = e^{-x}\cos x$ and we are asked to integrate it from x_1 *to* x_2, we should decide first what h should be. Depending on the value of h we will have to evaluate the value of *f(x)* inside the program for $x=x_1+nh$ where n=0,1, 2,….n and $n = (x_2 - x_1) / h$.

```
1    h = (x_2 − x_1) / n
2    x ← x_1
3    sum ← f(x)
4    for i = 2 to n do
5          x ← x + h
6          sum ← sum + 2 f(x)
     endfor
7    x ← x_2
8    sum ← sum + f(x)
9    int egral ← (h/2) . sum
10   write int egral
     stop
```

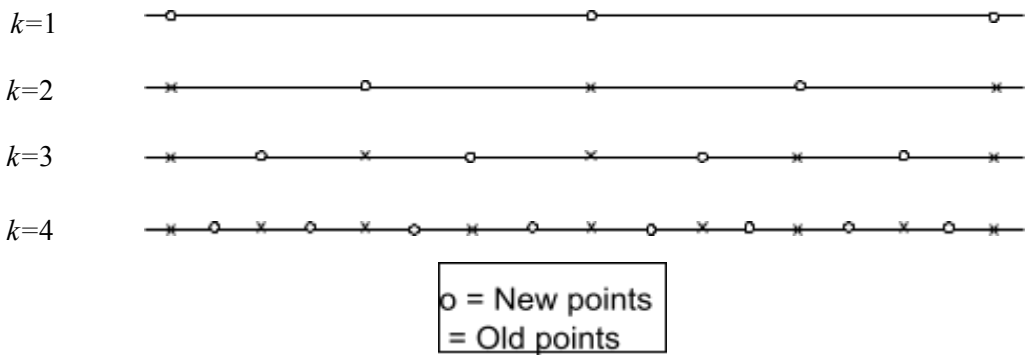
Ex. 3. (a) Find (approximately) each integral given below using the composite trapezoidal rule with $n = 12$.

$$(i)\ \int_{-1}^{1} (1+x^2)^{-1} dx \qquad (ii)\ \int_{0}^{4} x^2 e^{-x} dx$$

Adaptive Integration

When *f(x)* is a known function we can choose the value for *h* arbitrarily. The problem is that we do not know a priori what value to choose for *h* to attain a desired accuracy (for example, for an arbitrary *h* sharp picks of the function might be missed). To overcome this problem, we can start with two subintervals $h = h_1 = (x_2 - x_1) / 2$ and apply either trapezoidal or Simpson's 1/3 rule. Then we let $h_2 = h_1 /2$ and apply the formula again, now with four subintervals and the results are compared. If the new value is sufficiently close, the process is terminated. If the 2nd result is not close enough to the first, *h* is halved again and the procedure is repeated. This is continued until the last result is close enough to its predecessor. This form of numerical integration is termed as adaptive integration.

The no. of computations can be reduced because when *h* is halved, all of the old points at which the function was evaluated appear in the new computation and thus repeating evaluation can be avoided. This is illustrated below.



An algorithm for adaptive integration of a **known function** using trapezoidal rule:

1 Read x_1, x_2 , e Remark: The allowed error in integral is e

2 $h \leftarrow x_2 - x_1$

3 $S_1 \leftarrow (f(x_1) + f(x_2)) / 2$

4 $I_1 \leftarrow h \,.\, S_1$

5 $i \leftarrow 1$

Repeat

6 $x \leftarrow x_1 + h /2$

7 *for* $j = 1$ to i *do*

```
8                   S1 ← S1 + f(x)
9                       x ← x + h
                endfor
10              i ← 2i
11              h ← h /2
12              I0 ← I1
13              I1 ← h . S1
14          until I1 − I0 ≤ e . I1
15          write I1, h, i
            stop
```

Ex. 4. Evaluate the integral of xe^{-2x_2} between x=0 and x=2 using a tolerance value sufficiently small as to get an answer within 0.1% of the true answer, 0.249916.

Ex. 5. Evaluate the integral of $\sin^2(16x)$ between $x = 0$ and $x = \pi/2$. Why the result is erroneous? How can this be solved? (The correct result is $\pi / 4$)

Practice Problem (Experiment - 04)

Problem-1: Write the MATLAB functions implementing Trapezoidal, Simpson's 1/3 & Simpson's 3/8 Rule to determine the integral result of any function numerically.

Problem-2: Determine the integral of the given function using:

$$\int_{0.5}^{2} \frac{1 + 5\sin(x^2)}{(x+1)^{\frac{3}{2}}}\, dx$$

i. Trapezoidal Rule
ii. Simpson's 1/3 Rule
iii. Simpson's 3/8 Rule

Find the Analytical result of the integral as well. Calculate the Relative Percentage Error for each of the cases.
[You may use calculator to get the analytical value of the integral]

Problem-3: The following data are provided for the $velocity, v$ of a moving object as a function of $time, t$:

$t\ (sec.)$	0	1	2	3	4
$v\ (m/s)$	0	2.15	3.26	4.18	4.82

Determine the $displacement, s$ of the object for the given period of time with best possible accuracy. Which numerical approach would you choose?
[Just to recall: $s = \int v\ dt$]

Problem-4: Consider the data-points once again:

$t\ (sec.)$	0	1	2	3	4	5	6
$v\ (m/s)$	0	2.15	3.26	4.18	4.82	5.61	4.77

Determine the $displacement, s$ again. Which numerical approach would you choose now to have better accuracy?

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-05: Curve Fitting: Linear & Polynomial Regression

Introduction:

Data is often given for discrete values along a continuum. However we may require estimates at points between the discrete values. Then we have to fit curves to such data to obtain intermediate estimates. In addition, we may require a simplified version of a complicated function. One way to do this is to compute values of the function at a number of discrete values along the range of interest. Then a simpler function may be derived to fit these values. Both of these applications are known as **curve fitting**.

There are two general approaches of curve fitting that are distinguished from each other on the basis of the amount of error associated with the data. First, where the data exhibits a significant degree of error, the strategy is to derive a single curve that represents the general trend of the data. Because any individual data may be incorrect, we make no effort to intersect every point. Rather, the curve is designed to follow the pattern of the points taken as a group. One approach of this nature is called ***least squares regression***.

Second, where the data is known to be very precise, the basic approach is to fit a curve that passes directly through each of the points. The estimation of values between well-known discrete points from the fitted exact curve is called ***interpolation***.

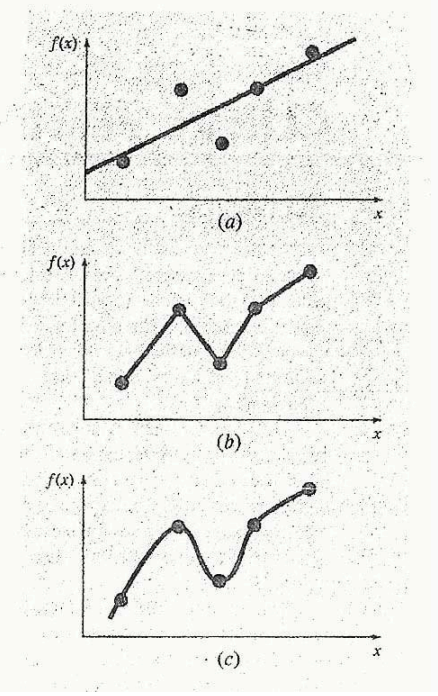


Figure 1: (a) Least squares linear regression (b) linear interpolation (c) curvilinear interpolation

Least squares Regression:

Where substantial error is associated with data, polynomial interpolation is inappropriate and may yield unsatisfactory results when used to predict intermediate values. A more appropriate strategy for such cases is to derive an approximating function that fits the shape or general trend of the data without necessarily matching the individual points. Now some criterion must be devised to establish a basis for the fit. One way to do this is to derive a curve that minimizes the discrepancy between the data points and the curve. A technique for accomplishing this objective is called least squares regression, where the goal is to minimize the sum of the square errors between the data points and the curve. Now depending on whether we want to fit a straight line or other higher order polynomial, regression may be linear or polynomial. They are described below.

Linear regression:

The simplest example of least squares regression is fitting a straight line to a set of paired observations: (x1, y1), (x2, y2), , , ,(xn, yn). The mathematical expression for straight line is

$y_m=a_0+a_1x$

Where a_0 and a_1 are coefficients representing the intercept and slope and y_m is the model value. If y_0 is the observed value and e is error or residual between the model and observation then

$e=y_0-y_m=y_0 - a_0 - a_1x$

Now we need some criteria such that the error e is minimum and also we can arrive at a unique solution (for this case a unique straight line). One such strategy is to minimize the sum of the square errors. So sum of square errors

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_{i,observed} - y_{i,\mathrm{model}})^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2 \quad \text{.....1}$$

To determine the values of a_0 and a_1, equation (1) is differentiated with respect to each coefficient.

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

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

Setting these derivatives equal to zero will result in a minimum Sr. If this is done, the equations can be expressed as

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

Now realizing that $\sum a_0 = n a_0$, we can express the above equations as a set of two simultaneous linear equations with two unknowns a_0 and a_1.

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

from where

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$
$$a_0 = \bar{y} - a_1 \bar{x}$$

Where $\bar{y}$ and $\bar{x}$ are the means of y and x respectively

Lab Task 1:

Fit a straight line to the x and y values of table 1

Table 1

x	y
1	0.5
2	2.5
3	2.0
4	4.0
5	3.5
6	6.0
7	5.5

Ans: a_0=0.071142857, a_1=0.8392857

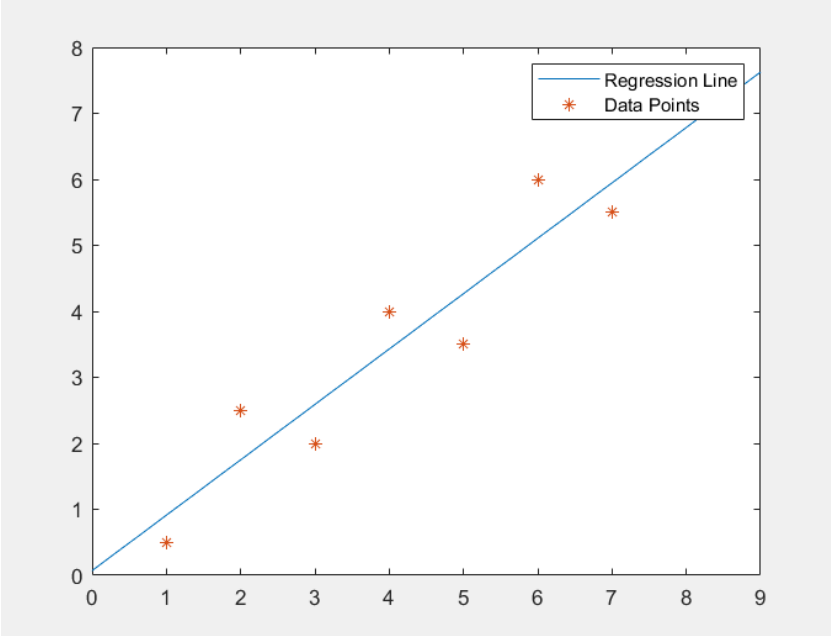
Code-

```
clear all
close all
clc

%Define the data
xa = [1,2,3,4,5,6,7];
ya = [.5,2.5,2.0,4,3.5,6,5.5];

[a1,a0] = linear_regression(xa,ya);%Implements simple least square linear regression
% [a,b1] = linear_regression_using_log(xa,ya);%Implements a linearized version of the regression line y=b1*[x^(a)]
%Find the approximated values for all the data points

% Y = [];
% for i = 1:0.1:20
%     c = a0 + a1 * i;
%     Y = [Y c];
% end

%y = contains true function poinsts
%Y = contains approximated function points using the regression line
xp = 0:0.1:9;

yp=a0 + a1 * xp;
plot(xp,yp,xa,ya,'*');
legend('Regression Line', 'Data Points');
```

Code for linear regression function-

```
function  [a1, a0]= linear_regression(x,y)

   n = length(x);
   sum_x = sum(x);
   sum_y = sum(y);
   square_x = sum(x.^2);
   sum_xy = sum(x.*y);

   a1 = ((n * sum_xy) - (sum_x * sum_y)) / ((n * square_x) - (sum_x*sum_x));
%     a1=((n*sum_xy)-(sum_x*sum_y))/((n*square_x)-(sum_x*sum_x));
   mean_y = sum_y / n;
   mean_x = sum_x / n;
   a0 = mean_y - a1 * mean_x;

end
```

Output-



Polynomial Regression:

In some cases, we have some engineering data that cannot be properly represented by a straight line. We can fit a polynomial to these data using polynomial regression.

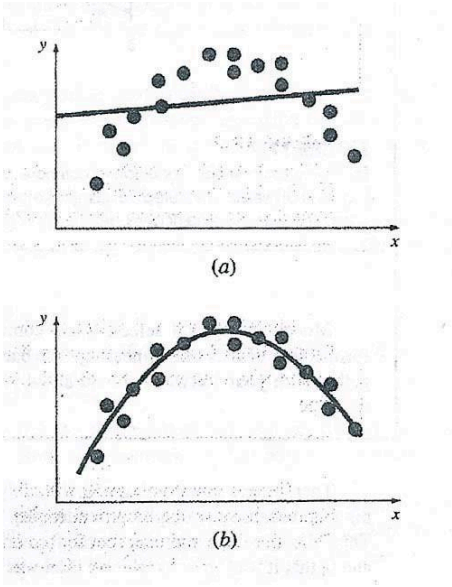


Figure 2: (a) Data that is ill-suited for linear least squares regression (b) indication that a parabola is preferable

The least squares procedure can be readily extended to fit the data to a higher order polynomial. For example, we want to fit a second order polynomial

$y_m=a_0 + a_1x+ a_2x^2$

For this case the sum of the squares of residuals is

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2$$

Taking derivative of equation (2) with respect to unknown coefficients a_0, a_1 and a_2

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

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

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

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

$$na_0 + (\sum x_i)a_1 + (\sum x_i^2)a_2 = \sum y_i$$
$$(\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i^3)a_2 = \sum x_i y_i$$
$$(\sum x_i^2)a_0 + (\sum x_i^3)a_1 + (\sum x_i^4)a_2 = \sum x_i^2 y_i$$

Now a_0, a_1 and a_2 can be calculated using matrix inversion.

Lab Task 2: Fit a second order polynomial to the data given in table 2

Table 2

x	y
0	2.1
1	7.7
2	13.6
3	27.2
4	40.9
5	61.1

Ans: a_0=2.47857, a_1=2.35929, a_2=1.86071

Code-

```
clear all
close all
clc

 %Define Data Points

 xa = [0,1,2,3,4,5];

 ya = [2.1,7.7,13.6,27.2,40.9,61.1];

 xp = 0:0.1:7;


[a0,a1,a2] = second_order_regression(xa,ya);%Call second order regression function

Y = a0 + a1 * xp + a2 * xp.*xp;

%Get the approximated function values using linear regression
```

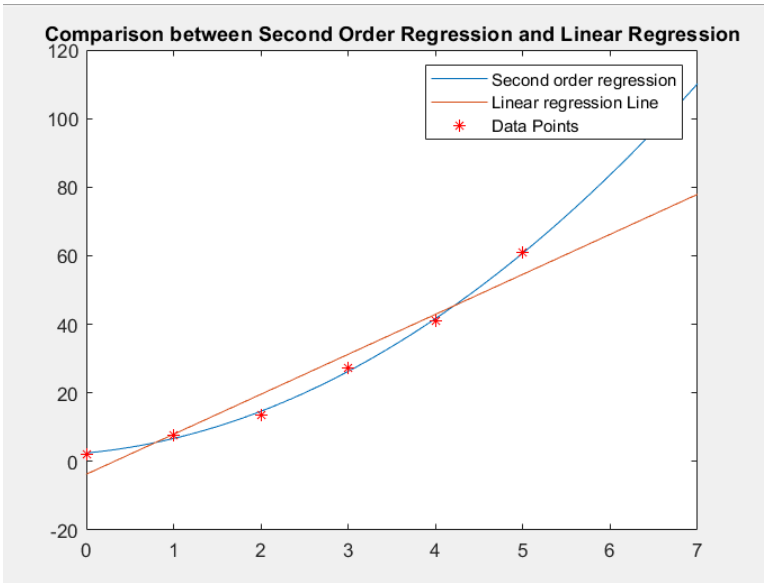
```
[A1,A0] = linear_regression(xa,ya);

 Y1 = A0 + A1 * xp;

%Plot the results

 plot(xp,Y,xp,Y1,xa,ya,'*r');

legend('Second order regression','Linear regression Line','Data Points');

title('Comparison between Second Order Regression and Linear Regression');
```

Function generating code for second order regression-

```
  function [a0,a1,a2] = second_order_regression(x,y)
   n = length(x);
   sum_x = sum(x);
   sum_y = sum(y);
   square_x = sum(x.*x);
   cube_x = sum(x.^3);
   sum_xy = sum(x.*y);
   sum_x2y = sum(x.*x.*y);
   x_4 = sum(x.^4);


   A = zeros(3,3);
   A(1,1) = n;
   A(1,2) = sum_x;
   A(1,3) = square_x;
   A(2,1) = sum_x;
   A(2,2) = square_x;
   A(2,3) = cube_x;
   A(3,1) = square_x;
   A(3,2) = cube_x;
   A(3,3) = x_4;

   C = inv(A);
   b = [sum_y; sum_xy; sum_x2y];
```

```
    P = (C*b);

    a0 = P(1);
    a1 = P(2);
    a2 = P(3);
end
```

Output-



Practice Problem (Experiment 05)

Problem-1: The following data shows the relationship between the viscosity of SAE 70 oil and temperature.

Temperature (℃)	26.67	93.33	148.89	315.56
Viscosity (N.s/*m*2)	1.35	0.085	0.012	0.00075

(a) Plot the data-points in MATLAB and decide whether a straight line could fit them.

(b) After taking the log of the data, use linear regression to find the equation of the line that best fits the data.

Problem-2: The data below represents the bacterial growth in a liquid culture over a number of days:

Day	0	4	8	12	16	20
Amount (x 106)	67	84	98	125	149	185

Find a best-fit equation to the data trend. Try several possibilities:

(a) linear (1st order),

(a) parabolic (2nd order),

(b) predict the number of bacteria after 40 days for both of the fitted equations.

Problem-3: Consider the following data table:

x	0.4	0.8	1.2	1.6	2	2.3
y	800	975	1500	1950	2900	3600

To fit these data points, which type of equation do you think would be suitable?

Find the equation accordingly.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-06: Curve Fitting: Linear & Polynomial Interpolation

---

Introduction:

Forming a polynomial:

A polynomial, p(x) of degree n in MATLAB is stored as a row vector, **p**, of length n+1. The components represent the coefficients of the polynomial and are given in the descending order of the power of x, that is

p = [a_n a_{n-1} a_1 a_0]

is interpreted as

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

In MATLAB the following commands are used to evaluate a polynomial:
polyval, **poly**, **roots**, **conv** etc.

Lab Task 1: Construct a polynomial such that C(x)= A(x)*B(x) Where A(x)= $3x^2+2x-4$ and B(x)= $2x^3-2$ Also find the roots of A(x), B(x) and C(x).

Interpolation:

In the mathematical subfield of numerical analysis, **interpolation** is a method of constructing new data points from a discrete set of known data points.

In engineering and science one often has a number of data points, as obtained by sampling or some experiment, and tries to construct a function which closely fits those data points. This is called curve fitting. Interpolation is a specific case of curve fitting, in which the function must go exactly through the data points.

Definition:

Given a sequence of *n distinct* numbers x_k called **nodes** and for each x_k a second number y_k, we are looking for a function *f* so that

$$f(x_k) = y_k \ , \ k = 1, \ldots, n$$

A pair x_k,y_k is called a **data point** and *f* is called the **interpolant** for the data points.

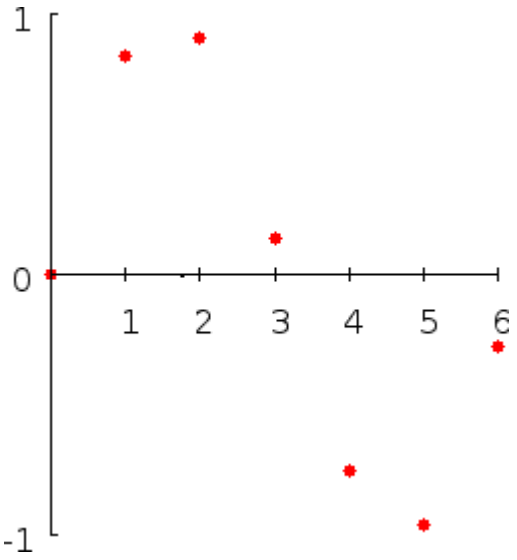
For example, suppose we have a table like this, which gives some values of an unknown function *f*. The data are given in the table:

Table 1

x	*f(x)*
0	0
1	0.8415
2	0.9093
3	0.1411
4	-0.7568
5	-0.9589
6	-0.2794

The plot can be shown as:

.

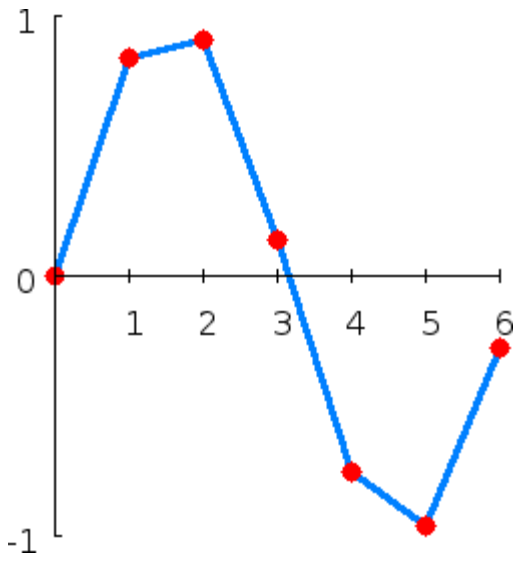


What value does the function have at, say, $x = 2.5$? Interpolation answers questions like this.

Types of interpolation:

A. Linear interpolation

One of the simplest methods is linear interpolation. Consider the above example of determining $f(2.5)$. We join the data points by linear interpolation and get the following plot:



Noe we can get $f(2.5)$. Since 2.5 is midway between 2 and 3, it is reasonable to take $f(2.5)$ midway between $f(2) = 0.9093$ and $f(3) = 0.1411$, which yields 0.5252.

Generally, linear interpolation takes two data points, say (x_a,y_a) and (x_b,y_b), and the interpolant is given by

$$f(x) = \frac{x - x_b}{x_a - x_b} y_a - \frac{x - x_a}{x_a - x_b} y_b$$

This formula can be interpreted as a weighted mean.

Linear interpolation is quick and easy, but it is not very precise.

Lab Task 2. Write a MATLAB code to implement Linear Interpolation and Plot the curve corresponding to table1 using Linear Interpolation.

Code-

```
% Data points from Table 1
x = [0, 1, 2, 3, 4, 5, 6];
fx = [0, 0.8415, 0.9093, 0.1411, -0.7568, -0.9589, -0.2794];

% Value of x for which you want to interpolate
x_interpolation = 2.5;

% Performing linear interpolation using Matlab's builtin function interp1
f_interpolated = interp1(x, fx, x_interpolation, 'linear');

% Displaying the resulted values
fprintf('Resulting Interpolated value at x = %.2f: %.4f\n', x_interpolation, f_interpolated);
% For .2f, the values will be showed after two decimal points, and for .4f,
% four decimal points will be displayed
```

Output-

```
Resulting Interpolated value at x = 2.50: 0.5252
>>
```

B. Polynomial interpolation

Polynomial interpolation is a generalization of linear interpolation. Note that the linear interpolant is a linear function. We now replace this interpolant by a polynomial of higher degree.

Consider again the problem given above. The following sixth-degree polynomial goes through all the seven points:

$$f(x) = -0.0001521x^6 - 0.003130x^5 + 0.07321x^4 - 0.3577x^3 + 0.2255x^2 + 0.9038x$$

Substituting $x = 2.5$, we find that $f(2.5) = 0.5965$.

Generally, if we have n data points, there is exactly one polynomial of degree $n-1$ going through all the data points. The interpolation error is proportional to the distance between the data points to the power n.

However, polynomial interpolation also has some disadvantages. Calculating the interpolating polynomial is relatively very computationally expensive Furthermore, polynomial interpolation may not be so exact after all, especially at the endpoints.

a. Lagrange Polynomial:

The Lagrange interpolating polynomial is the polynomial $P(x)$ of degree $(n-1)$ that passes through the n points $(x_1, y_1 = f(x_1))$, $(x_2, y_2 = f(x_2))$, ..., $(x_n, y_n = f(x_n))$, and is given by

$$P(x) = \sum_{j=1}^{n} P_j(x),$$

where

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}.$$

Written explicitly,

$$P(x) = \frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)} y_1 + \frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)} y_2 + \cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})} y_n.$$

When constructing interpolating polynomials, there is a tradeoff between having a better fit and having a smooth well-behaved fitting function. The more data points that are used in the interpolation, the higher the degree of the resulting polynomial, and therefore the greater oscillation it will exhibit between the data points. Therefore, a high-degree interpolation may be a poor predictor of the function between points, although the accuracy at the data points will be "perfect."

For $n = 3$ points,

$$P(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} y_3$$

Note that the function $P(x)$ passes through the points (x_i, y_i), as can be seen for the case $n = 3$,

$$\begin{aligned}
\frac{(x_1 - x_2)(x_1 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_1 - x_1)(x_1 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_1 - x_1)(x_1 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 &= y_1 \\
\frac{(x_2 - x_2)(x_2 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_2 - x_1)(x_2 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_2 - x_1)(x_2 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 &= y_2 \\
\frac{(x_3 - x_2)(x_3 - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x_3 - x_1)(x_3 - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x_3 - x_1)(x_3 - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 &= y_3.
\end{aligned}$$

Algorithm for the Lagrange Polynomial: To construct the Lagrange polynomial

$$P(x) = \sum_{k=0}^{n} y_k L_{n,k}(x)$$

of degree n, based on the n+1 points (x_k, y_k) for $k = 0, 1, \cdots, n$. The Lagrange coefficient polynomials for degree *n* are:

$$L_{n,k}(x) = \frac{(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}$$

for $k = 0, 1, \cdots, n$.

So, for a given *x* and a set of *(N+1)* data pairs, (x_i, f_i), *i= 0, 1, N:*

Set SUM=0

DO FOR *i*=0 to *N*

Set *P*=1

DO FOR *j*=0 to *N*

IF *j~=i*

Set *P=P*(x-x(j))/(x(i)-x(j))*

End DO(*j*)

Lab Task 3. Construct Lagrange interpolating Polynomials for the data points given

in table -1 and Plot the curve.

Function generating code for Lagrange Polynomial-

```
function sum = lagrange_polynomial(x,y)

   n = length(x);
   sum = 0;

   for i = 1:n
      num = 1;
      den = 1;
      for j = 1:n
         if (i~= j)
%              num = conv(num,poly(x(j)));
            num = conv(num,[1 -x(j)]);
            den = den * (x(i) - x(j));
         end
      end
         c = num / den;
         c1 = c* y(i);
         sum = sum + c1;
   end
end
```

Code for Plotting Lagrange Polynomial-

```
clear all
close all
clc

%x = input('X:');
%y = input('Y:');
%Define Data Points
x = [0,1,2,3,4,5,6];
y = [0,0.8415,0.9093,0.1411,-0.7568,-0.9589,-0.2794];

P = lagrange_polynomial(x,y);%lagrange polynomial coefficients
% Y = [];

%Get the approximated values using lagrange polynomial
% for i = x(1):0.1:x(end)
```

```
%     y1 = polyval(P,i);
%     Y = [Y y1];
% end

s = x(1):0.1:x(end);
Y = polyval(P,s);

%Plot the results

plot(s,Y,'linewidth',3),hold on
plot(x,y,'*r','linewidth',3),hold off
```

Note: Run the code for generating the function in a separate tab, and keeping it aside the scripted plotting execution code will be needing to execute in Matlab.

Practice Problems (Experiment - 06)

Problem-1: The following data come from a table that was measured with high precision. Use the best numerical method (for this type of problem) to determine y at x = 3.5 using MATLAB. Note that a polynomial will yield an exact value.

x	0	1.8	5	6	8.2	9.2	12
y	26	16.415	5.375	3.5	2.015	2.54	8

Problem-2: The following data define the sea-level concentration of dissolved Oxygen for fresh water as a function of temperature:

T (℃)	0	8	16	24	32	40
Oxygen (mg/L)	14.621	11.843	9.870	8.418	7.305	6.413

Estimate Oxygen concentration using MATLAB for T = 27℃, considering:

(a) Newton's interpolation,

(b) Note that the exact result is 7.986 mg/L. Calculate the percentage error for both of the cases.

Problem-3: Generate eight equally-spaced points from the function using MATLAB:

$$f(t) = sin^2(t)$$

from t = 0 to 2π. Fit these data with a seventh-order interpolating polynomial.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-07: Solution of Simultaneous Linear Equations

Objective

Systems of linear algebraic equations occur often in diverse fields of science and engineering and are an important area of study. In this experiment we will be concerned with the different techniques of finding the solution of a set of n linear algebraic equations in n unknowns.

Concept of linear equations and their solution

A set of linear algebraic equations looks like this:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots a_{1N}x_N = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots a_{2N}x_N = b_2 \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \\ &a_{M1}x_1 + a_{M2}x_2 + \ldots a_{MN}\ x_N = b_M \end{aligned} \qquad (1)$$

Here the N unknowns x_j , $j = 1, 2, \ldots, N$ are related by M equations. The coefficients a_{ij} with $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$ are known numbers, as are the *right-hand side* quantities b_i, $i = 1, 2, \ldots, M$.

Existence of solution

If $N = M$ then there are as many equations as unknowns, and there is a good chance of solving for a unique solution set of x_j's. Analytically, there can fail to be a unique solution if one or more of the M equations is a linear combination of the others (This condition is called *row degeneracy*), or if all equations contain certain variables only in exactly the same linear combination(This is called *column degeneracy*). (For square matrices, a row degeneracy implies a column degeneracy, and vice versa.) A set of equations that is degenerate is called *singular*.
Numerically, at least two additional things can go wrong:

- While not exact linear combinations of each other, some of the equations may be so close to linearly dependent that rounding errors in the machine renders them linearly dependent at some stage in the solution process. In this case your numerical procedure will fail, and it can tell you that it has failed.
- Accumulated round off errors in the solution process can swamp the true solution. This problem particularly emerges if N is too large. The numerical procedure does not fail algorithmically. However, it returns a set of x's that are wrong, as can be discovered by direct substitution back into the original equations. The closer a set of equations is to being singular, the more likely this is to happen.

Matrices

Equation (1) can be written in matrix form as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \qquad (2)$$

Here the raised dot denotes matrix multiplication, **A** is the matrix of coefficients, **x** is the column vector of unknowns and **b** is the right-hand side written as a column vector,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Finding Solution

There are so many ways to solve this set of equations. Below are some important methods.

(1) Using the backslash and pseudo-inverse operator

In MATLAB, the easiest way to determine whether Ax = b has a solution, and to find such a the solution when it does, is to use the backslash operator. Exactly what **A** \ **b** returns is a bit complicated to describe, but if there is a solution to **A** · **x** = **b**, then **A** \ **b** returns one.
Warnings: (1) **A** \ **b** returns a result in many cases when there is no solution to **A** · **x** = **b**. (2) **A** \ **b** sometimes causes a warning to be issued, even when it returns a solution. This means that you can't just use the backslash operator: you have to check that what it returns is a solution. (In any case, it's just good common sense to check numerical computations as you do them.) In MATLAB this can be done as follows:

Using backslash operator:
x = A\b;

You can also use the pseudo-inverse operator:
x=pinv(A)*b; % it is also guaranteed to solve Ax = b, if Ax = b has a solution. As

with the backslash operator, you have to check the result.

(2) Using Gauss-Jordan Elimination and Pivoting

To illustrate the method let us consider three equations with three unknowns:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \quad \text{(A)}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = a_{24} \quad \text{(B)}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = a_{34} \quad \text{(C)}$$

Here the quantities b_i, $i = 1, 2, \ldots, M$'s are replaced by a_{iN+1}, where i=1,2, ….M for simplicity of understanding the algorithm.

The First Step is to eliminate the first term from Equations (B) and (C). (Dividing (A) by a_{11} and multiplying by a_{21} and subtracting from (B) eliminates x_1 from (B) as shown below)

$$(a_{21} - \frac{a_{11}}{a_{11}}a_{21})x_1 + (a_{22} - \frac{a_{12}}{a_{11}}a_{21})x_2 + (a_{23} - \frac{a_{13}}{a_{11}}a_{21})x_3 = (a_{24} - \frac{a_{14}}{a_{11}}a_{21})$$

Let, $\frac{a_{21}}{a_{11}} = k_2$, then

$$(a_{21} - k_2 a_{11})x_1 + (a_{22} - k_2 a_{12})x_2 + (a_{23} - k_2 a_{13})x_3 = (a_{24} - k_2 a_{14})$$

Similarly multiplying equation (A) by $\frac{a_{31}}{a_{11}} = k_3$ and subtracting from (C), we get

$$(a_{31} - k_3 a_{11})x_1 + (a_{32} - k_3 a_{12})x_2 + (a_{33} - k_3 a_{13})x_3 = (a_{34} - k_3 a_{14})$$

Observe that $(a_{21} - k_2 a_{11})$ and $(a_{31} - k_3 a_{11})$ are both zero.

In the steps above it is assumed that a_{11} is not zero. This case will be considered later in this experiment.

The above elimination procedure is called triangularization.

Algorithm for triangularization n equations in n unknowns:

```
1       for i = 1 to n and j = 1 to (n +1) in steps of 1 do read a_ij endfor
2       for k = 1 to (n −1) in steps of 1 do
3               for i = (k +1) to n in steps of 1 do
4               u ← a_ik / a_kk
5               for j = k to (n +1) in steps of 1 do
6                       a_ij ← a_ij − u a_kj endfor
                endfor
        endfor
```

The reduced equations are:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24}$$

$$a_{32}x_2 + a_{33}x_3 = a_{34}$$

The next step is to eliminate a_{32} from the third equation. This is done by multiplying second equation by

$u = a_{32} / a_{22}$ and subtracting the resulting equation from the third. So, same algorithm can be used.

Finally the equations will take the form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24}$$

$$a_{33}x_3 = a_{34}$$

The above set of equations are said to be in triangular (Upper) form.

From the above upper triangular form of equations, the values of unknowns can be obtained by back substitution as follows:

$$x_3 = a_{34} / a_{33}$$

$$x_2 = (a_{24} - a_{23}x_3) / a_{22}$$

$$x_2 = (a_{14} - a_{12}x_2 - a_{13}x_3) / a_{11}$$

Algorithmically, the back substitution for n unknowns is shown below:

```
1       x_n ← a_n(n+1) / a_nn
2       for i = (n −1) to 1 in step of -1 do
3       sum ← 0
4               for j = (i + 1) to n in steps of 1 do
5               sum ← sum + a_ij x_j endfor
6       x_i ← (a_i(n+1) − sum) / a_ii
        endfor
```

Lab Task 1. Given the simultaneous equations shown below, solve for x_1, x_2, x_3, using Gaussian Elimination.

$$2x_1\ 3x_2\ 5x_3\ 23$$
$$3x_1\ 4x_2\ x_3\ 14$$
$$6x_1\ 7x_2\ 2x_3\ 26$$

Code:

```
clear all
close all
clc

A=[2 3 5; 3 4 1; 6 7 2]; %coefficient matrix
b=[23; 14; 26]; %constant matrix

%here B=augmented upper triangular matrix; x=solutions
[B,x] = naive_gaussian_elimination(A,b)
```

Function generating code for the Naive Gaussian Elimination:

```
function [A,x] = naive_gaussian_elimination(A,b)

    %Form the augmented matrix
    A = [A b];
    [m,n] = size(A);%size of the matrix; m=row, n=column

    for i = 1:m-1%Denotes the number of steps for converting into
upper triangular matrix.
        for j = i+1:m
            A(j,i)/A(i,i)
            A(j,:) = A(j,:) - (A(i,:)/A(i,i)) * A(j,i);
            A
        end
    end

    %Back Substitution
    x = zeros(1,n-1);
    x(n-1) = A(m,n)/A(m,n-1);
    for i = n-2:-1:1
        sum  = 0;
        for j = i+1:n-1
            sum = sum + A(i,j) * x(j);
        end
        x(i) = (A(i,n) - sum)/A(i,i);
    end
end
```

Pivoting

In the triangularization algorithm we have used,

$$u \leftarrow a_{ik} / a_{kk}$$

Here it is assumed that a_{kk} is not zero. If it happens to be zero or nearly zero, the algorithm will lead to no results or meaningless results. If any of the a_{kk} is small it would be necessary to reorder the equations. It is noted that the value of a_{kk} would be modified during the elimination process and there is no way of predicting their values at the start of the procedure. The elements a_{kk} are called pivot elements. In the elimination procedure the pivot should not be zero or a small number. In fact for maximum precision the pivot element should be the largest in absolute value of all the elements below it in its column, i.e. up as the maximum of all a_{mk} where, $m \geq k$ a_{kk} should be picked.

So, during the Gauss elimination, a_{mk} elements should be searched and the equation with the maximum value of a_{mk} should be interchanged with the current position. For example if during elimination we have the following situation:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ 0.3x_2 + 4x_3 &= 5 \\ -8x_2 + 3x_3 &= 6 \end{aligned}$$

As $-8 > 0.3$, 2nd and 3rd equations should be interchanged to yield:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ -8x_2 + 3x_3 &= 6 \\ 0.3x_2 + 4x_3 &= 5 \end{aligned}$$

It should be noted that interchange of equations does not affect the solution.
The algorithm for picking the largest element as the pivot and interchanging the equations is called pivotal condensation.

Algorithm for pivotal condensation

```
1       max ← |a_kk|
2       p ← k
3       for m = (k+1) to n in steps of 1 do
4               if (|a_mk| > max) then
5               max ← |a_mk|
6               p ← m
7               endif
        endfor
8       if (p ~= k)
9               for q = k to (n+1) in steps of 1 do
```

10 $temp \leftarrow a_{kq}$

11 $a_{kq} \leftarrow a_{pq}$

12 $a_{pq} \leftarrow temp$

endfor

endif

Lab Task 2. Modify the MATLAB program of Naive Gaussian Elimination to include pivotal condensation and singularity check.

Lab Task 3. Try to solve the following systems of equations (i) Gaussian elimination, (ii) Gaussian elimination with pivoting

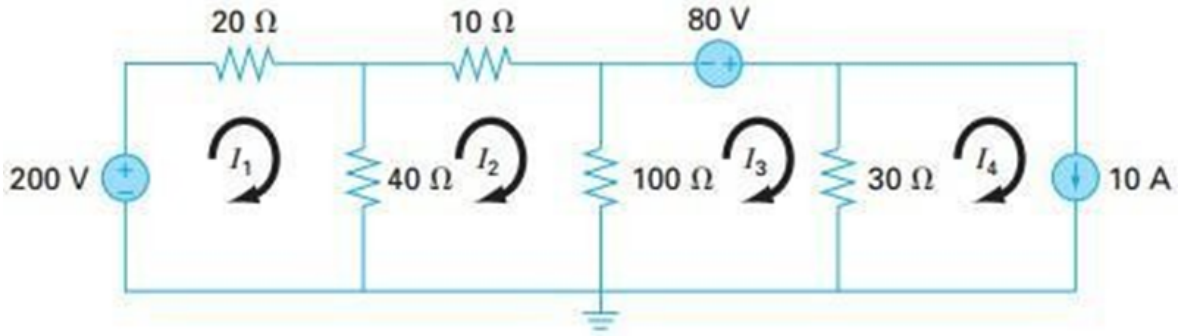
(A)
$$\begin{aligned} 2x_1 + 4x_2 - 6x_3 &= -4 \\ x_1 + 5x_2 + 3x_3 &= 10 \\ x_1 + 3x_2 + 2x_3 &= 5 \end{aligned}$$

(B)
$$\begin{aligned} x_1 + x_2 + 6x_3 &= 7 \\ -x_1 + 2x_2 + 9x_3 &= 2 \\ x_1 - 2x_2 + 3x_3 &= 10 \end{aligned}$$

(C)
$$\begin{aligned} 4x_1 + 8x_2 + 4x_3 &= 8 \\ x_1 + 5x_2 + 4x_3 - 3x_4 &= -4 \\ x_1 + 4x_2 + 7x_3 + 2x_4 &= 10 \\ x_1 + 3x_2 - 2x_4 &= -4 \end{aligned}$$

Practice Problem (Experiment 07)

Problem-1: **(Lab Report) (Use Naive Gauss Elimination)**



Find the equations of the mesh currents using Mesh Analysis first. After doing so, determine the value of the mesh currents using suitable Numerical technique.

Problem-2: (Lab Report) (Use Naive Gauss Elimination)

Consider the following vectors:

$$\vec{A} = 2\vec{i} - 3\vec{j} + a\vec{k}$$

$$\vec{B} = b\vec{i} + \vec{j} - 4\vec{k}$$

$$\vec{C} = 3\vec{i} + c\vec{j} + 2\vec{k}$$

Vector $\vec{A}$ is perpendicular to $\vec{B}$ as well as to $\vec{C}$. It is also given that $\vec{B}.\vec{C} = 2$. Use a suitable Numerical technique to find the values of a, b & c.

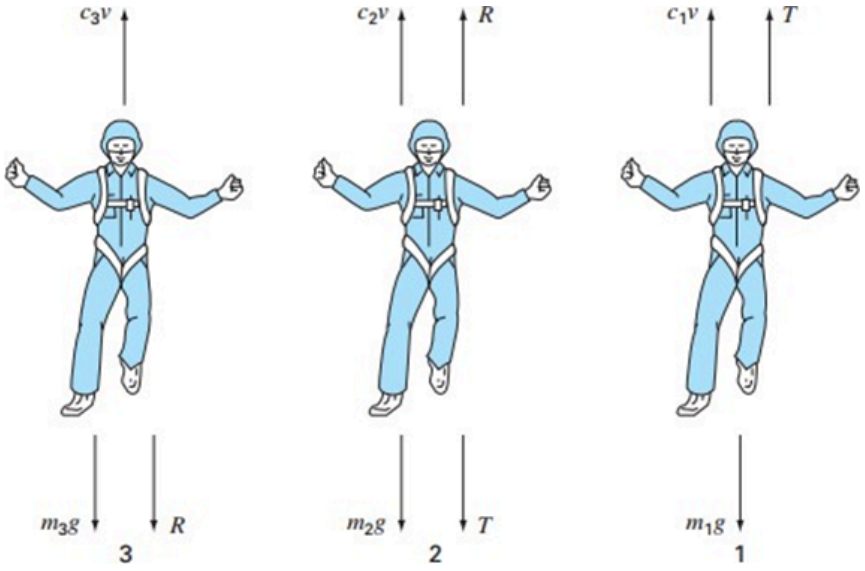
Problem-3: (Lab Report) (Use Naive Gauss Elimination)

Suppose that a team of three parachutists is connected by a weightless cord while free-falling at a velocity of 5 m/s.

Calculate the tension (T & R) in each section of cord and the acceleration (a) of the team, given the following:

Parachutist	Mass, kg	Drag Coefficient, kg/s
1	70	10
2	60	14
3	40	17

[Hint: Consider each of the parachutists according to the following figure:



Applying Newton's 2nd Law, the following equations can be formed:

$$m1g - T - c1v = m1a$$

$$m_2g + T - c_2v - R = m_2a$$

$$m3g - c3v + R = m3a$$

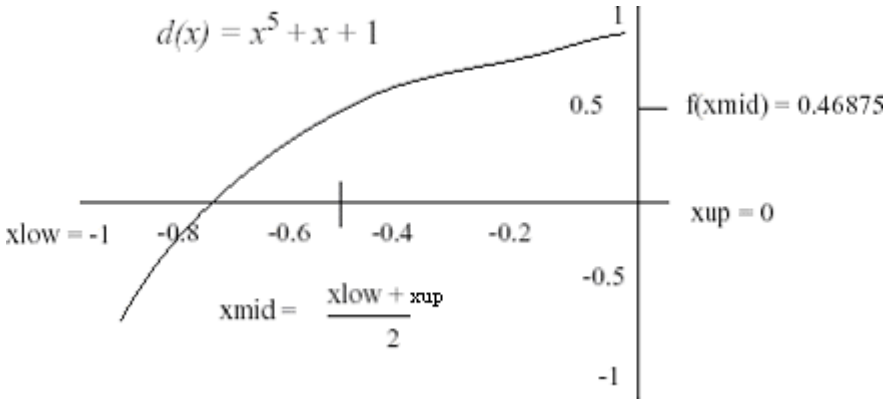
where, $m1$, $m2$, $m3$ are the masses and $c1$, $c2$, $c3$ are the drag coefficients of the three parachutists.

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-08: Solution to Non-Linear Equations

Bisection method:

The Bisection method is one of the simplest procedures for finding the root of a function in a given interval.

The procedure is straightforward. The approximate location of the root is first determined by finding two values that bracket the root (a root is bracketed or enclosed if the function changes sign at the endpoints). Based on these a third value is calculated which is closer to the root than the original two values. A check is made to see if the new value is a root. Otherwise a new pair of brackets is generated from the three values, and the procedure is repeated.



Consider a function $d(x)$ and let there be two values of x , x_{low} and x_{up} ($x_{up} > x_{low}$), bracketing a root of $d(x)$.

Steps:

1. The first step is to use the brackets x_{low} and x_{up} to generate a third value that is closer to the root. This new point is calculated as the mid-point between x_{low} and x_{up} namely $x_{mid} = \frac{x_{low} + x_{up}}{2}$. The method therefore gets its name from this bisecting of two values. It is also known as interval halving method.
2. Test whether x_{mid} is a root of $d(x)$ by evaluating the function at x_{mid}.
3. If x_{mid} is not a root,
   a. If $d(x_{low})$ and $d(x_{mid})$ have opposite signs i.e. $d(x_{low}) \cdot d(x_{mid})$

<0, root is in left half of interval.

b. If $d(x_{low})$ and $d(x_{mid})$ have same signs i.e. $d(x_{low}) \cdot d(x_{mid})$ >0, root is in right half of interval.

4. Continue subdividing until interval width has been reduced to a size ≤ ε

where ε = selected x tolerance

Algorithm: Bisection Method

```
Input xLower, xUpper, xTol
yLower = f(xLower) (* invokes fcn definition *)
xMid = (xLower + xUpper)/2.0
yMid = f(xMid)
iters = 0 (* count number of iterations *)
While ( (xUpper - xLower)/2.0 > xTol )
iters = iters + 1
if( yLower * yMid > 0.0) Then xLower = xMid
Else xUpper = xMid
Endofif
xMid = (xLower + xUpper)/2.0
yMid = f(xMid)
Endofwhile
Return xMid, yMid, iters (* xMid = approx to root *)
```

Exercise 1. Find the real root of the equation $d(x) = x^5 + x + 1$ using Bisection Method. x_{low} = -1, x_{up} =0 and ε = selected x tolerance $=10^{-4}$.

Note: For a given x tolerance (epsilon), we can calculate the number of iterations directly. The number of divisions of the original interval is the smallest value of n that satisfies: $\frac{x_{up} - x_{low}}{2^n} \langle \varepsilon \text{ or } 2^n \rangle \frac{x_{up} - x_{low}}{\varepsilon}$

Thus $n \rangle \log 2\left(\frac{x_{up} - x_{low}}{\varepsilon}\right)$.

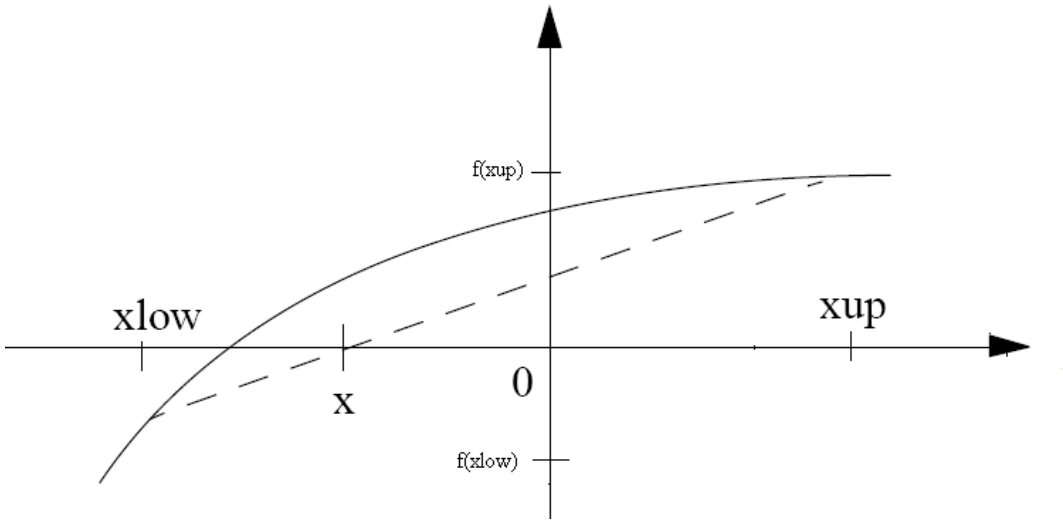
In our previous example, x_{low} = -1, x_{up} =0 and ε = selected x tolerance $=10^{-4}$.

So we have $n = 14$.

False-Position Method (Regula Falsi)

A shortcoming of the bisection method is that, in dividing the interval from x_{low} to x_{up} into equal halves, no account is taken of the magnitude of $f(x_{low})$ and $f(x_{up})$. For example, if $f(x_{low})$ is much closer to zero than $f(x_{up})$, it is likely that the root is closer to x_{low} than to x_{up}. An alternative method that exploits this graphical insight is to join $f(x_{low})$ and $f(x_{up})$ by a straight line. The intersection of this line with the x axis represents an improved estimate of the root. The fact that the replacement of the curve by

a straight line gives the false position of the root is the origin of the name, method of false position, or in Latin, Regula Falsi. It is also called the Linear Interpolation Method.



Using similar triangles, the intersection of the straight line with the x axis can be estimated as

$$\frac{f(x_{low})}{x - x_{low}} = \frac{f(x_{up})}{x - x_{up}}$$

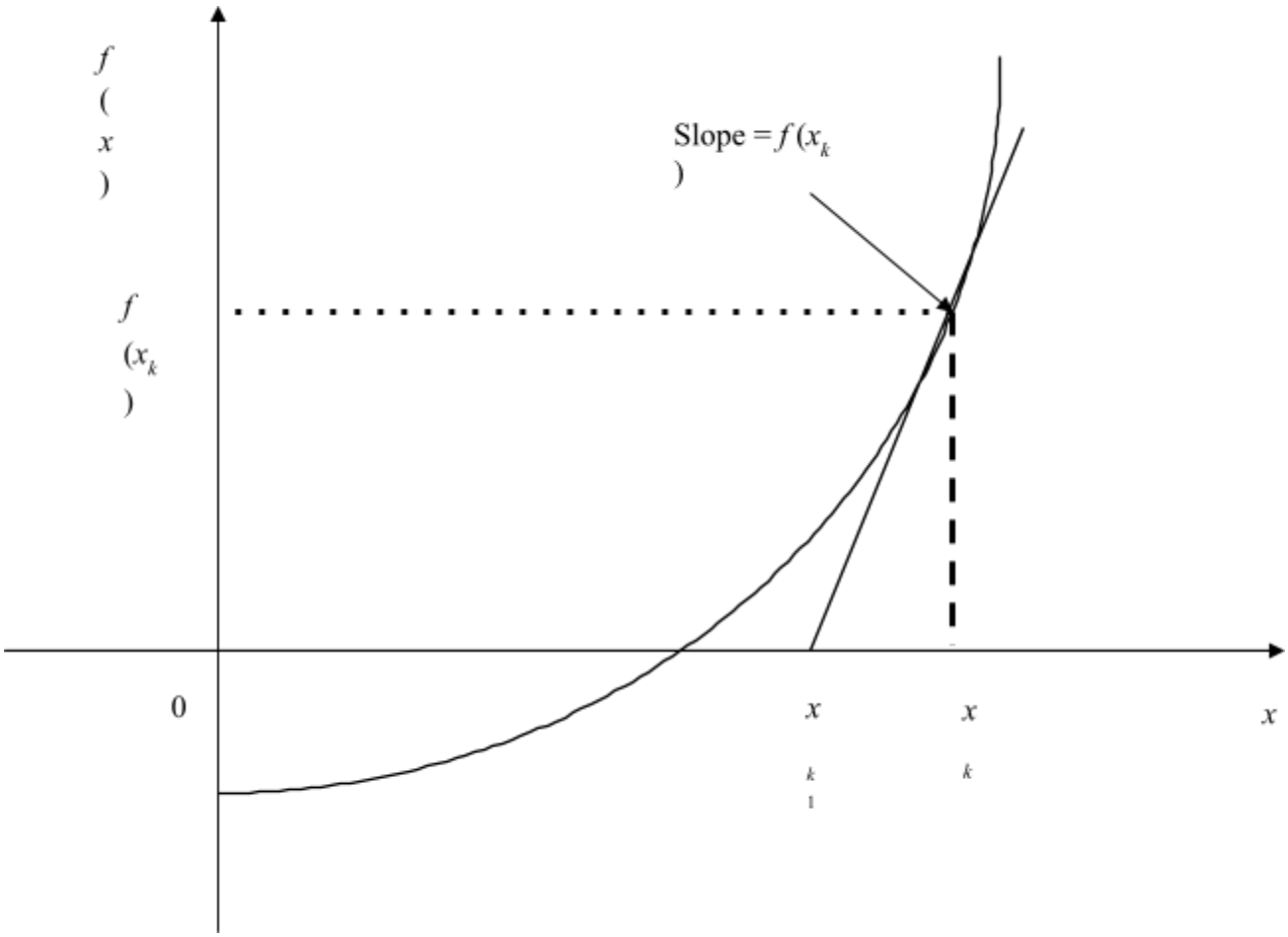
That is $x = x_{up} - \dfrac{f(x_{up})(x_{low} - x_{up})}{f(x_{low}) - f(x_{up})}$

This is the False Position formulae. The value of x then replaces whichever of the two initial guesses, x_{low} or x_{up}, yields a function value with the same sign as $f(x)$. In this way, the values of x_{low} and x_{up} always bracket the true root. The process is repeated until the root is estimated adequately.

Newton Raphson Method:

If $f(x)$, $f'(x)$ and $f''(x)$ are continuous near a root x , then this extra information regarding the nature of $f(x)$ can be used to develop algorithms that will produce sequences $\{x_k\}$ that converge faster to x than either the bisection or false position method. The Newton-Raphson (or simply Newton's) method is one of the most useful and best-known algorithms that relies on the continuity of $f'(x)$ and $f''(x)$.

The attempt is to locate root by repeatedly approximating $f(x)$ with a linear function at each step. If the initial guess at the root is x_k, a tangent can be extended from the point $[x_k, f(x_k)]$. The point where this tangent crosses the x axis usually represents an improved estimate of the root.



The Newton-Raphson method can be derived on the basis of this geometrical interpretation. As in the figure, the first derivative at x is equivalent to the slope:

$$f'(x_k) = \frac{f(x_k) - 0}{x_k - x_{k+1}}$$

which can be rearranged to yield

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

which is called the Newton Raphson formulae.

So the Newton-Raphson Algorithm actually consists of the following steps:

1. Start with an initial guess 0 x and an x-tolerance ε.

2. Calculate $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad k = 0,1,2,\ldots$

Algorithm - Newton's Method

Input x0, xTol
iters = 1
dx = -f(x0)/fDeriv(x0) (* fcns f and fDeriv *)
root = x0 + dx
While (Abs(dx) > xTol)
dx = -f(root)/fDeriv(root)
root = root + dx
iters = iters + 1
End of while
Return root, iters

Exercise 2. Use the Newton Raphson method to estimate the root of $f(x) = e^{-x} - 1$, employing an initial guess of $x = 0$. The tolerance is = 10^{-8} .

Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-09: Linear Ordinary Differential Equations

Introduction:

In mathematics, a differential equation is an equation in which the derivatives of a function appear as variables. Differential equations have many applications in physics and chemistry, and are widespread in mathematical models explaining biological, social, and economic phenomena.

Differential equations are divided into two types:

a. An Ordinary Differential Equation (ODE) only contains a function of one variable, and derivatives in that variable.

b. A Partial differential Equation (PDE) contains multivariate functions and their partial derivatives.

The order of a Differential equation is that of the highest derivative that it contains. For instance, a first-order Differential equation contains only first derivatives.

A linear differential equation of order n is a differential equation written in the following form:

$$a_n(x)\frac{d^n y}{dx^n} + a_{n-1}(x)\frac{d^{n-1} y}{dx^{n-1}} + \ldots\ldots\ldots + a_1(x)\frac{dy}{dx} + a_0(x)y = f(x)$$

where, $a_n(x) \neq 0$.

Initial value problem:

A problem in which we are looking for the unknown function of a differential equation where the values of the unknown function and its derivatives at some point are known is called an initial value problem (in short IVP).

If no initial conditions are given, we call the description of all solutions to the differential equation the general solution.

Methods of solving the Ordinary Differential Equations:

- **Euler's Method:**

Let $y'(x) = f(x, y(x))$

$$y(x_0) = y_0$$

Here $f(x, y)$ is a given function, y_0 is a given initial value for y at $x = x_0$. The unknown in the problem is the function $y(x)$.

Our goal is to determine (approximately) the unknown function $y(x)$ for $x > x_0$. We are told explicitly the value of $y(x_0)$, namely y_0. Using the given differential equation, we can also determine exactly the instantaneous rate of change of y at x_0. The basic idea is simple. This differential equation tells us how rapidly the variable y is changing and the initial condition tells us where y starts

$y'(x_0) = f(x_0, y(x_0)) = f(x_0, y_0)$

If the rate of change of $y(x)$ were to remain $f(x_0, y_0)$ for all time, then $y(x)$ would be exactly $y_0 + f(x_0, y_0)(x - x_0)$. The rate of change of $y(x)$ does not remain $f(x_0, y_0)$ for all time, but it is reasonable to expect that it remains close to $f(x_0, y_0)$ for x close to x_0. If this is the case, then the value of $y(x)$ will remain close to $y_0 + f(x_0, y_0)(x - x_0)$ for x close to x_0. So pick a small number h and define

$x_1 = x_0 + h$

$y_1 = y_0 + f(x_0, y_0)(x_1 - x_0) = y_0 + f(x_0, y_0)h$

By the above argument

$y(x_1) \approx y_1$

Now we start over. We now know the approximate value of y at x_1. If $y(x_1)$ were exactly y_1, then the instantaneous rate of change of y at x_1 would be exactly $f(x_1, y_1)$. If this rate of change were to persist for all future value of x, $y(x)$ would be exactly $y_1 + f(x_1, y_1)(x - x_1)$

As $y(x_1)$ is only approximately y_1 and as the rate of change of $y(x)$ varies with x, the rate of change of $y(x)$ is only approximately $f(x_1, y_1)$ and only for x near x_1. So we approximate $y(x)$ by $y_1 + f(x_1, y_1)(x - x_1)$ for x bigger than, but close to, x_1. Defining $x_2 = x_1 + h = t_0 + 2h$

$y_2 = y_1 + f(x_1, y_1)(x_2 - x_1) = y_1 + f(x_1, y_1)h$

We have $y(x_2) \approx y_2$

We just repeat this argument. Define, for $n = 0, 1, 2, 3$.....

$x_n = x_0 + nh$

Suppose that, for some value of n, we have already computed an approximate value y_n for $y(x_n)$. Then the rate of change of $y(x)$ for x close to x_n is $f(x, y(x)) \approx f(x_n, y(x_n)) \approx y(x_n, y_n)$

and, again for x near x_n, $y(x) = y_n + f(x_n, y_n)(x - x_n)$ Hence

$y(x_{n+1}) \approx y_{n+1} = y_n + f(x_n, y_n)h$

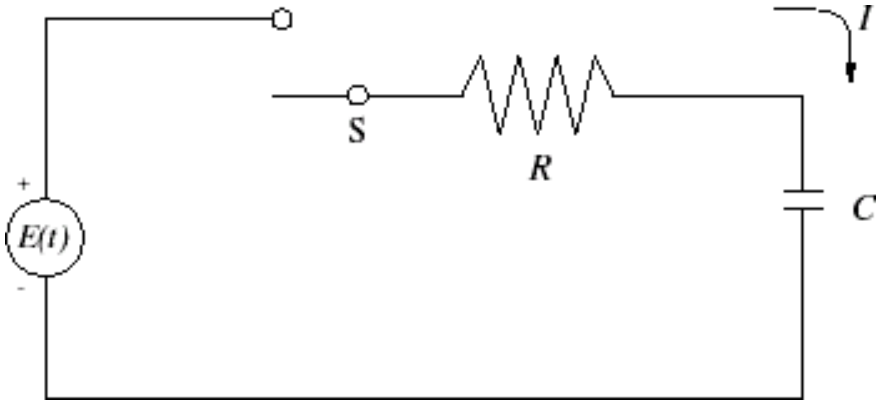
This algorithm is called Euler's Method. The parameter h is called the step size.

Exercise 1. Use Euler's method to solve the IVP $y' = \frac{x-y}{2}$ on $[0, 3]$ with $y(0) = 1$.

Compare solutions for $h = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$

The exact solution is $y = 3e^{-x/2} - 2 + x$

Exercise 2. Consider the following circuit:



In this circuit, $R = 20K\Omega, C = 10\mu F, E = 117\ V$ and $Q(0) = 0$. Find

Q for $t = 0$ to $t = 3sec$.

- **The Improved Euler's Method**

Euler's method is one algorithm that generates approximate solutions to the initial value problem

$$y'(x) = f(x, y(x))\ y(x_0) = y_0$$

In applications, $f(x, y)$ is a given function and x_0 and y_0 are given numbers. The function $y(x)$ is unknown. Denote by $\phi(x)$ the exact solution for this initial value problem. In other words $\phi(x)$ is the function that obeys the following relation exactly.

$\phi'(x) = f(x, \phi(x))$

$\phi(x_0) = y_0$

Fix a step size h and define $x_n = x_0 + nh$. We now derive another algorithm that generates approximate values for ϕ at the sequence of equally spaced values x_0, x_1, x_2..... We shall denote the approximate values y_n with $y_n = \phi(x_n)$

By the fundamental theorem of calculus and the differential equation, the exact solution obeys

$$\phi(x_{n+1}) = \phi(x_n) + \int_{xn}^{x_{n+1}} \phi'(x)dx = \phi(x_n) + \int_{x_n}^{x_{n+1}} f(x, \phi(x))dx$$

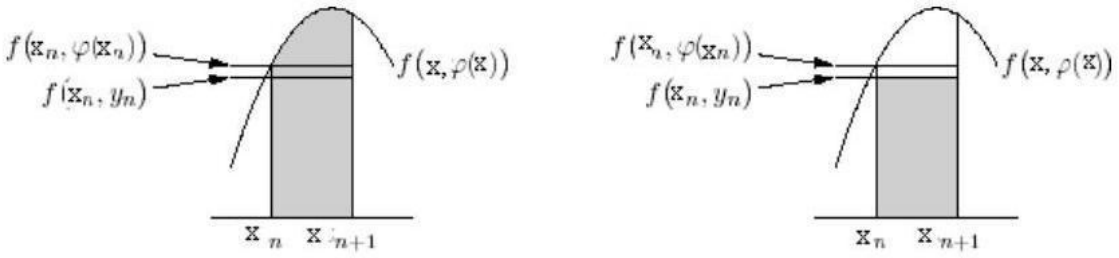
Fix any n and suppose that we have already found y_0, y_1, y_2.....y_n for computing y_{n+1} will be of the form:

$$y_{n+1} = y_n + \ approximate\ value\ for \int_{x_n}^{x_{n+1}} f(x, \phi(x))dx$$

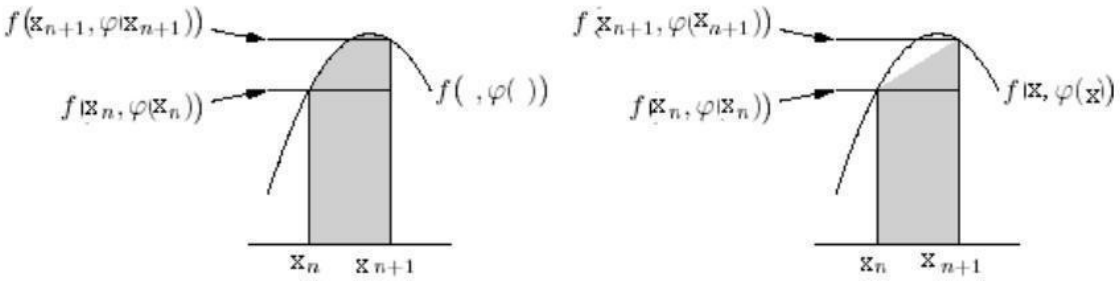
In fact Euler's method is of precisely this form. In Euler's method, we approximate $f(x, \phi(x))$ for $x_n \le x \le x_{n+1}$ by the constant $f(x_n, y_n)$. Thus Euler's approximate value for

$$\int_{x_n}^{x_{n+1}} f(x, \phi(x))dx = \int_{x_n}^{x_{n+1}} f(x_n, y_n)dx = f(x_n, y_n)h$$

The area of the complicated region $0 \le y \le f(x, \phi(x)); x_n \le x \le x_{n+1}$ (represented by the shaded region under the parabola in the left half of the figure below) is approximated by the area of the rectangle $0 \le y \le f(x_n, y_n); x_n \le x \le x_{n+1}$ (the shaded rectangle in the right half of the figure below).



Our second algorithm, the improved Euler's method, gets a better approximation by attempting to approximate by the trapezoid on the right below rather than the rectangle on the right above. The exact area of this trapezoid is the length h of the base multiplied by



the average, $\frac{1}{2}\left[f\left(x_n, \phi\left(x_n\right)\right) + f\left(x_{n+1}, \phi\left(x_{n+1}\right)\right)\right]$

we do not know $\phi\left(x_n\right)$ or $\phi\left(x_{n+1}\right)$ exactly. Recall that we have already found $y_0, y_1, y_2, \ldots\ldots, y_n$ and are in the process of finding y_{n+1}. So we already have an approximation for $\phi\left(x_n\right)$, namely y_n, but not for $\phi\left(x_{n+1}\right)$. Improved Euler uses $\phi\left(x_{n+1}\right) \approx \phi\left(x_n\right) + \phi'\left(x_n\right)h \approx y_n + f\left(x_n, y_n\right)h$

in approximating $\frac{1}{2}\left[f\left(x_n, \phi\left(x_n\right)\right) + f\left(x_{n+1}, \phi\left(x_{n+1}\right)\right)\right]$. Altogether Improved Euler's approximate value for $\int_{x_n}^{x_{n+1}} f(x, \phi(x))dx = \frac{1}{2}\left[f\left(x_n, y_n\right)\right) + f\left(x_{n+1}, y_n + f\left(x_n, y_n\right)h\right)]h$ so that the improved Euler's method algorithm is

$y\left(x_{n+1}\right) \approx y_{n+1} = y_n + \frac{1}{2}\left[f\left(x_n, y_n\right)\right) + f\left(x_{n+1}, y_n + f\left(x_n, y_n\right)h\right)]h$

The general step is:

$p_{n+1} = y_n + hf\left(x_n, y_n\right),\ x_{n+1} = x_n + h, y_{n+1} = y_n + \frac{h}{2}\left[f\left(x_n, y_n\right)\right) + f\left(x_{n+1}, p_{n+1}\right)]$

Exercise 3. Solve exercise 1 and 2 using Improved Euler's method.