

# **LAB MANUAL OF APPLIED MATHEMATICS LAB USING SCILAB**

*ETMA 252*



*Maharaja Agrasen Institute of Technology  
PSP Area, Plot No. 1, Sector 22, Rohini, Delhi, 110086*

## **MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

### **VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

### **MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

#### **Engineering Hardware - Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

#### **Life - Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

#### **Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

#### **Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

#### **Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

#### **Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.

---

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY  
COMPUTER SCIENCE and ENGINEERING DEPARTMENT  
VISION**

To inculcate **Critical thinkers of Innovative Technology**

**MISSION**

1. To provide an excellent learning environment across the computer science discipline to inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities which enable them to become successful entrepreneurs in this globalized world.
2. To nurture an excellent learning environment that helps students to enhance their problem solving skills and to prepare students to be lifelong learners by offering a solid theoretical foundation with applied computing experiences and educating them about their professional, and ethical responsibilities.
3. To establish Industry-Institute Interaction, making students ready for the industrial environment and be successful in their professional lives.
4. To promote research activities in the emerging areas of technology convergence.
5. To build engineers who can look into technical aspects of an engineering solution thereby setting a ground for producing successful entrepreneur.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction to Scilab</b>   | <b>7</b>  |
| 1.1      | Overview . . . . .  | 8         |
| 1.2      | Become Familiar With Scilab . . . . .                                       | 8         |
| 1.3      | The Menu Bar . . . . .  | 9         |
| 1.4      | Pre-defined mathematical variables and operators . . . . .                  | 9         |
| 1.5      | Booleans . . . . .  | 10        |
| 1.6      | Complex Numbers . . . . .   | 10        |
| 1.7      | Strings . . . . .   | 10        |
| 1.8      | Matrices . . . . .  | 10        |
| 1.9      | Looping and Branching . . . . .   | 11        |
| 1.10     | Programming in Scilab . . . . .   | 15        |
| 1.10.1   | Editor . . . . .  | 15        |
| 1.11     | Functions . . . . .   | 16        |
| 1.12     | The Graphic Window . . . . .  | 17        |
| <b>2</b> | <b>Lab Objective and Requirements</b>                                       | <b>18</b> |
| 2.1      | Lab Objectives . . . . .  | 19        |
| 2.2      | Course Outcome: . . . . .   | 19        |
| 2.3      | Lab Requirements . . . . .  | 19        |
| 2.4      | List of Experiments as Prescribed by G.G.S.I.P.U . . . . .                  | 20        |
| 2.5      | List of Experiments Beyond the Syllabus Prescribed by G.G.S.I.P.U . . . . . | 20        |
| 2.6      | Format of The Lab Records to be Prepared by the Students . . . . .          | 21        |
| 2.7      | Marking Scheme For The Practical Examination . . . . .                      | 23        |

---

|          |  |           |
|----------|--|-----------|
| <b>3</b> | <b>Matrix Operations</b>   | <b>25</b> |
| 3.1      | Matrix Addition . . . . .  | 26        |
| 3.2      | Matrix Multiplication . . . . .  | 30        |
| 3.3      | Matrix Transpose . . . . .   | 34        |
| 3.4      | Viva Questions . . . . .   | 37        |
| <b>4</b> | <b>Inverse of a <math>3 \times 3</math> Matrix Using Gauss Jordan Method</b>         | <b>38</b> |
| 4.1      | Viva Questions . . . . .   | 43        |
| <b>5</b> | <b>Eigen Values and Eigen Vectors of a <math>2 \times 2</math> Matrix</b>            | <b>44</b> |
| 5.1      | Viva Questions . . . . .   | 48        |
| <b>6</b> | <b>Measures of Central Tendency</b>  | <b>49</b> |
| 6.1      | Measures of Central Tendency (Ungrouped Data) . . . . .                              | 50        |
| 6.2      | Measures of Central Tendency (Grouped Data) . . . . .                                | 55        |
| 6.3      | Viva Questions . . . . .   | 58        |
| <b>7</b> | <b>Curve fitting</b>   | <b>59</b> |
| 7.1      | Fitting a Straight Line . . . . .  | 60        |
| 7.2      | Fitting a Parabola . . . . .   | 63        |
| 7.3      | Viva Questions . . . . .   | 66        |
| <b>8</b> | <b>Plotting of 2D Graphs Using Scilab</b>  | <b>67</b> |
| 8.0.1    | A simple graph plotting raw data . . . . .   | 69        |
| 8.0.2    | Generation of square wave . . . . .  | 69        |
| 8.0.3    | Unit Step function I . . . . .   | 70        |
| 8.0.4    | Unit Step function II . . . . .  | 71        |
| 8.1      | Viva Questions . . . . .   | 72        |
| <b>9</b> | <b>Numerical Methods for Finding Roots of Algebraic and Transcendental Equations</b> | <b>73</b> |
| 9.1      | Bisection Method . . . . .   | 74        |
| 9.2      | Newton-Raphson Method . . . . .  | 77        |
| 9.3      | Viva Questions . . . . .   | 80        |

---

|  |           |
|--|-----------|
| <b>10 Numerical Integration for Solving Definite Integrals</b>                           | <b>81</b> |
| 10.1 The Trapezoidal Rule . . . . .  | 82        |
| 10.2 Simpson's $1/3^{rd}$ Rule . . . . .   | 86        |
| 10.3 Simpson's $3/8^{th}$ Rule . . . . .   | 90        |
| 10.4 Viva Questions . . . . .  | 93        |
| <b>11 Numerical Solution of Ordinary Differential equations using Euler's Method</b>     | <b>94</b> |
| 11.1 Euler's Method . . . . .  | 95        |
| 11.2 Viva Questions . . . . .  | 98        |
| <b>12 Numerical Solution of Ordinary Differential Equations Using Runge-Kutta Method</b> | <b>99</b> |
| 12.1 Runge-Kutta method of $4^{th}$ Order . . . . .                                      | 100       |
| 12.2 Viva Questions . . . . .  | 103       |

# Chapter 1

## Introduction to Scilab



---

# *Introduction to Scilab*

## **1.1 Overview**

Scilab is a programming language associated with a rich collection of numerical algorithms covering many aspects of scientific computing problems.

From the software point of view, Scilab is an interpreted language. This generally allows to get faster development processes, because the user directly accesses to a high level language, with a rich set of features provided by the library. The Scilab language is meant to be extended so that user-defined data types can be defined with possibly overloaded operations. Scilab users can develop their own module so that they can solve their particular problems. The Scilab language allows to dynamically compile and link other languages such as Fortran and C and in this way, external libraries can be used as if they were a part of Scilab built-in features. Scilab is also a numerical computation software that anybody can freely download. Available under Windows, Linux and Mac OS X, Scilab can be downloaded at the following address <http://www.scilab.org/>.

An online help is provided in many local languages.

From a scientific point of view, Scilab comes with many features. At the very beginning of Scilab, features were focused on linear algebra. But rapidly, the number of features extended to cover many areas of scientific computing such as matrices, statistics, Ordinary differential equations, signal processing, interpolation, approximation, linear, quadratic and non linear optimization and many more.

## **1.2 Become Familiar With Scilab**

The Scilab workspace consists of several windows:

- The console for making calculations
- The editor for writing programs

- 
- The graphics windows for displaying graphics
  - The embedded help

## 1.3 The Menu Bar

The following options in menu are particularly useful:

### Applications

- The command history allows you to find all the commands from previous sessions to the current session.
- The variables browser allows you to find all variables previously used during the current session.

### Editing

- Preferences (in Scilab menu under Mac OS X) allow you to set and customize colors, fonts and font size in the console and in the editor, which is very useful for screen projection.
- Clicking on Clear Console clears the entire content of the console. In this case, the command history is still available and calculations made during the session remain in memory. Commands that have been erased are still available through the keyboards arrow keys.

## 1.4 Pre-defined mathematical variables and operators

In Scilab, several mathematical variables are pre-defined variables, which name begins with a percent % character. The variables which have a mathematical meaning are summarized in the given table:

|     |                                 |
|-----|---------------------------------|
| %i  | the imaginary number $i$        |
| %e  | Euler's constant $e$            |
| %pi | the mathematical constant $\pi$ |

---

Also apart from usual operators for summation, subtraction, multiplication and division, comparison operators are as given:

|  |   |
|--|---|
| <code>a&amp;b</code>                         | logical and                               |
| <code>a b</code>                             | logical or                                |
| <code>~a</code>                              | logical not                               |
| <code>a==b</code>                            | true if the two expressions are equal     |
| <code>a~=b</code> or <code>a&lt;&gt;b</code> | true if the two expressions are different |
| <code>a&lt;b</code>                          | true if a is lower than b                 |
| <code>a&gt;b</code>                          | true if a is greater than b               |
| <code>a&lt;=b</code>                         | true if a is lower or equal to b          |
| <code>a&gt;=b</code>                         | true if a is greater or equal to b        |

## 1.5 Booleans

Boolean variables can store true or false values. In Scilab, true is written with `%t` or `%T` and false is written with `%f` or `%F`.

## 1.6 Complex Numbers

Scilab provides complex numbers, which are stored as pairs of floating point numbers. The predefined variable `%i` represents the mathematical imaginary number  $i$  which satisfies  $i^2 = -1$ .

## 1.7 Strings

Strings can be stored in variables, provided that they are delimited by double quotes. The concatenation operation is available from the `+` operator. They are many functions which allow to process strings, including regular expressions.

## 1.8 Matrices

There is a simple and efficient syntax to create a matrix with given values. The following is the list of symbols used to define a matrix:

- square brackets `"[` and `]"` mark the beginning and the end of the matrix
- commas `","` separate the values on different columns
- semicolons `";"` separate the values of different rows

---

The following syntax can be used to define an  $m \times n$  matrix, where blank spaces are optional (but make the line easier to read) and "..." are designating intermediate values:  $A = [a_{11}, a_{12}, \dots, a_{1n}; a_{21}, a_{22}, \dots, a_{2n}; \dots; a_{m1}, a_{m2}, \dots, a_{mn}]$

### Use of colon operator in matrices

Following table depicts use of colon operator in matrices:

|          |  |
|----------|--|
| A        | the whole matrix                                     |
| A(:,j)   | the whole matrix                                     |
| A(i:j,k) | the elements at rows from $i$ to $j$ , at column $k$ |
| A(i,j:k) | the elements at row $i$ , at columns from $j$ to $k$ |
| A(i,:)   | the row $i$  |
| A(:,j)   | the column $j$                                       |

### Matrix and usual operators

Following table shows various operator used in matrix operations:

|         |                         |    |                               |
|---------|-------------------------|----|-------------------------------|
| +       | addition                | .+ | elementwise addition          |
| -       | subtraction             | .- | elementwise subtraction       |
| *       | multiplication          | .* | elementwise multiplication    |
| /       | right division          | ./ | elementwise right division    |
| \       | left division           | .\ | elementwise left division     |
| ^ or ** | power i.e. $x^y$        | .^ | elementwise power             |
| '       | transpose and conjugate | .' | transpose (but not conjugate) |

## 1.9 Looping and Branching

In this section, we describe how to make conditional statements

### The if statement

The if uses a boolean variable to perform its choice: if the boolean is true, then the statement is executed. A condition is closed when the end keyword is met. In the following script, we display the string "Hello!" if the condition %t, which is always true, is satisfied.

```
if ( %t ) then
disp ( " Hello ! " )
end
```

The previous script produces:

Hello !

If the condition is not satisfied, the else statement allows to perform an alternative

---

statement, as in the following script:

```
if ( %f ) then
disp ( " Hello ! " )
else
disp ( " Goodbye !" )
end
```

The previous script produces:

Goodbye !

In order to get a boolean, any comparison operator can be used, e.g. `==`, `>`, etc or any function which returns a boolean. In the following session, we use the `==` operator to display the message "Hello !".

```
i = 2
if ( i == 2 ) then
disp ( " Hello ! " )
else
disp ( " Goodbye !" )
end
```

### **The select statement**

The select statement allows to combine several branches in a clear and simple way. Depending on the value of a variable, it allows to perform the statement corresponding to the case keyword. There can be as many branches as required. In the following script, we want to display a string which corresponds to the given integer *i*.

```
i = 2
select i
case 1
disp ( " One " )
case 2
disp ( " Two " )
case 3
disp ( " Three " )
```

---

```
else
disp (" Other ")
end
```

The previous script prints out "Two", as expected. The else branch is used if all the previous case conditions are false.

### **The for statement**

The for statement allows to perform loops, i.e. to perform a given action several times. Most of the time, a loop is performed over an integer value, which goes from a starting to an ending index value. We will see, at the end of this section, that the for statement is in fact much more general, as it allows to loop through the values of a matrix. In the following Scilab script, we display the value of  $i$ , from 1 to 3.

```
for i = 1 : 3
disp (i)
end
```

The previous script produces the following output.

- 1.
- 2.
- 3.

In the previous example, the loop is performed over a matrix of floating point numbers containing integer values. Indeed, we used the colon ":" operator in order to produce the vector of index values [1, 2, 3]. The following session shows that the statement 1:3 produces all the required integer values into a row vector.

```
-- > i = 1 : 3
i = 1.2.3.
```

### **The while statement**

The while statement allows to perform a loop while a boolean expression is true. At the beginning of the loop, if the expression is true, the statements in the body of the loop are executed. When the expression becomes false (an event which must occur at certain time), the loop is ended. In the following script, we compute the sum of the numbers  $i$  from 1 to 10 with a while statement.

---

```
s = 0
i = 1
while ( i <= 10 )
s = s + i
i = i + 1
end
```

At the end of the algorithm, the values of the variables  $i$  and  $s$  are:

$s = 55$  and  $i = 11$ .

### **The break and continue statement**

The break statement allows to interrupt a loop. Usually, we use this statement in loops where, if some condition is satisfied, the loops should not be continued further. In the following example, we use the break statement in order to compute the sum of the integers from 1 to 10. When the variable  $i$  is greater than 10, the loop is interrupted.

```
s = 0
i = 1
while ( %t )
if ( i > 10 ) then
break
end
s = s + i
i = i + 1
end
```

At the end of the algorithm, the values of the variables  $i$  and  $s$  are:

$s = 55$  and  $i = 11$

The continue statement allows to go on to the next loop, so that the statements in the body of the loop are not executed this time. When the continue statement is executed, Scilab skips the other statements and goes directly to the while or for statement and evaluates the next loop. In the following example, we compute the sum  $s = 1 + 3 + 5 + 7 + 9 = 25$ . The `modulo( $i$ , 2)` function returns 0 if the number  $i$  is even. In this situation, the script increments the value of  $i$  and use

---

the continue statement to go on to the next loop.

```
s = 0
i = 1
while ( i <= 10)
if ( modulo ( i, 2) == 0 ) then
i = i + 1
continue
else
s = s + i
i = i + 1
end
end
```

If the previous script is executed, the final values of the variables  $i$  and  $s$  are:

$s = 25$  and  $i = 11$ .

## 1.10 Programming in Scilab

Scilab is an interpreted language, where the commands in a program are depicted exactly as they should be typed in the scilab editor.

Scilab calculates only with numbers. All calculations are done with matrices, although this may go unnoticed. In this section, we present the basic features of the language showing how to create a real variable, and what elementary mathematical functions can be applied to a real variable. If Scilab provided only these features, it would only be a super desktop calculator. Fortunately, it is a lot more and this is the subject of the remaining sections.

### 1.10.1 Editor

Typing directly into the console has two disadvantages:

- It is not possible to save the commands.
- It is not easy to edit multiple lines of instructions in the console window.

---

The editor is the appropriate tool to run multiple instructions.

### **Opening the editor:**

To open the editor from the console, click on the first icon in the toolbar or on Applications '*SciNotes*' in the menu bar. The editor opens with a default file named '*Untitled1*'. Writing or typing in the editor is just like as any word processor.

In the text editor, opening and closing parentheses, end loops, function and test commands are added automatically.

### **Saving the program**

Any file can be saved by clicking on File > Save as

The extension `.sce` at the end of a file name will launch automatically Scilab when opening it (except under Linux and Mac OS X).

### **Executing the Program**

By clicking on Execute in the menu bar, three options are available:

- Execution file with no echo (Ctrl Shift E under Windows and Linux, Cmd Shift E under Mac OS X): the file is executed without writing the program in the console (saving the file first is mandatory).
- Execute file with echo (Ctrl L under Windows and Linux, Cmd L under Mac OS X): rewrite the file into the console and executes it.
- Execute until the caret, with echo (Ctrl E under Windows and Linux, Cmd E under Mac OS X): rewrite the selection chosen with the mouse into the console and executes it or execute the file data until the caret position defined by the user.

## **1.11 Functions**

When several commands are to be executed, it may be more convenient to write these statements into a file with Scilab editor. These are called as SCRIPT files. To execute the commands written in such a script file, the `exec` function can be used, followed by the name of the script file. These file generally have the extension `.sce` or `.sci`, depending on the content. Files having the `.sci` extension contain Scilab

---

functions and/or user defined functions and executing them loads the functions into Scilab environment (but does not execute them), whereas Files having the .sce extension contains both Scilab functions and executable statements. Please remember that the convention of naming the extension as .sce and .sci are not RULES, but a convention followed by the scilab community. The most simple calling sequence of a function is the following:

```
outvar = myfunction ( invar )
```

where the following list presents the various variables used in the syntax:

- myfunction is the name of the function
- invar is the name of the input arguments
- outvar is the name of the output arguments

## 1.12 The Graphic Window

A graphics window opens automatically when making any plot. It is possible to plot curves, surfaces and also sequences of points on scilab interface.

## Chapter 2

### Lab Objective and Requirements



---

## 2.1 Lab Objectives

The purpose of this course is to introduce the students with Scilab applications in Mathematical operations. The students will be able to enhance their analyzing and problem solving skills and use the same for writing programs in Scilab.

## 2.2 Course Outcome:

At the end of the course, the students will be able to:

C154.1: write, compile and debug programs in Scilab.

C154.2: use conditional expressions and looping statement to solve problems associated with conditions and repetitions.

C154.3: implement the programs using arithmetic and relational operators.

C154.4: understand the concept of inbuilt and user defined functions.

C154.5: understand and solve matrices operations effectively.

C154.6: able to choose programming components that efficiently solve computing problems in Mathematics .

## 2.3 Lab Requirements

### Hardware Details :

Intel i3/C2D Processor/2 GB RAM/500GB HDD/MB/LAN Card/ Key Board/  
Mouse/CD Drive/15" Color Monitor/ UPS - 33 Nos  
LaserJet Printer - 1 No

---

**Software Detail :**

**Operating System:** BSDs (e.g., FreeBSD), Linux, macOS, Windows

Open source software: Scilab 5.5.2

## **2.4 List of Experiments as Prescribed by G.G.S.I.P.U**

Paper Code: ETMA 252

Paper: Applied Mathematics Lab

1. Solution of algebraic and transcendental equation.
2. Algebra of matrices: Addition, multiplication, transpose etc.
3. Inverse of a system of linear equations using Gauss-Jordan method.
4. Numerical Integration.
5. Solution of ordinary differential equations using Runge-Kutta Method.
6. Solution of Initial value problem.
7. Calculation of eigen values and eigen vectors of a matrix.
8. Plotting of Unit step function and square wave function.

## **2.5 List of Experiments Beyond the Syllabus Prescribed by G.G.S.I.P.U**

1. Measures of Central Tendency (Ungrouped Data)
2. Measures of Central Tendency (Grouped Data)
3. Curve Fitting (Fitting a Straight Line)
4. Curve Fitting (Fitting a Parabola)

---

## 2.6 Format of The Lab Records to be Prepared by the Students

The front page of the lab record prepared by the students should have a cover page as displayed below.

### *NAME OF THE LAB*

Font should be (Size 20", italics bold, Times New Roman)

Faculty name

Font should be (12", Times Roman)

Student name

Roll No.:

Semester:

Group:

Font should be (12", Times Roman)



Maharaja Agrasen Institute of Technology, PSP Area,  
Sector – 22, Rohini, New Delhi – 110086

Font should be (18", Times Roman)

---

The second page in the record should be the index as displayed below.

**INTRODUCTION TO PROGRAMMING PRACTICAL RECORD**

PAPER CODE : ETMA 252

Name of the student :

University Roll No. :

Branch :

Section/ Group :

Experiments according to the list provided by GGSIPU and Beyond the syllabus

| Exp. no | Experiment Name | Date of performance | Date of checking | Remarks | Marks (10) |
|---------|-----------------|---------------------|------------------|---------|------------|
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |
|         |                 |                     |                  |         |            |

---

## 2.7 Marking Scheme For The Practical Examination

There will be two practical exams in each semester

- i. Internal Practical Exam
- ii. External Practical Exam

### INTERNAL PRACTICAL EXAM

It is taken by the respective faculty of the batch

**MARKING SCHEME FOR THIS EXAM IS:**

Total Marks: 40

Division of 10 marks per practical is as follows:

| <b>Rubrics for : Laboratory (General)</b> |  |            |   |   |
|---|--|------------|---|---|
| Serial No.                                | Experiment Component (LAC)             | Max. Marks | Grading Rubrics   |   |
|   |  |            | 2 marks   | 1 mark  |
| 1   | Practical Performance                  | 2          | Completeness of practical, exhibits proficiency in using different types of Inputs.   | Incomplete practical, unformatted, lacks comments, Demonstrates no Proficiency.   |
| 2   | Output and Validation                  | 2          | Output is free of errors and output is obtained. Demonstrates excellent understanding of the concepts relevant to the Experiment.                                 | Output contains few logical errors and/or no output is obtained. Demonstrates partial understanding of the Concepts relevant to the experiment. |
| 3   | Attendance and Viva Questions Answered | 4          | 1. Four marks for answering more than 75% questions.<br>2. Two marks for answering more than 50% questions.<br>3. One mark for answering less than 50% questions. |   |
| 4   | Timely Submission of Lab Record        | 2          | On time submission  | Late submission   |

Each experiment will be evaluated out of 10 marks. At the end of the semester average of 8 best performed practical will be considered as marks out of 40.

---

## EXTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

### MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 60

Division of 60 marks is as follows

- a. Sheet filled by the student: 20
- b. Viva Voice: 15
- c. Experiment performance: 15
- d. File submitted: 10

NOTE:

Internal marks + External marks = Total marks given to the students  
(40 marks)                      (60 marks)                      (100 marks)

Experiments given in exam to be performed can be from any section of the lab.

# Chapter 3

## Matrix Operations

---

# *Matrix Operations*

## **3.1 Matrix Addition**

- $(A + B)_{i,j} = A_{i,j} + B_{i,j}$
- Order of the matrices must be the same
- Matrix addition is commutative
- Matrix addition is associative

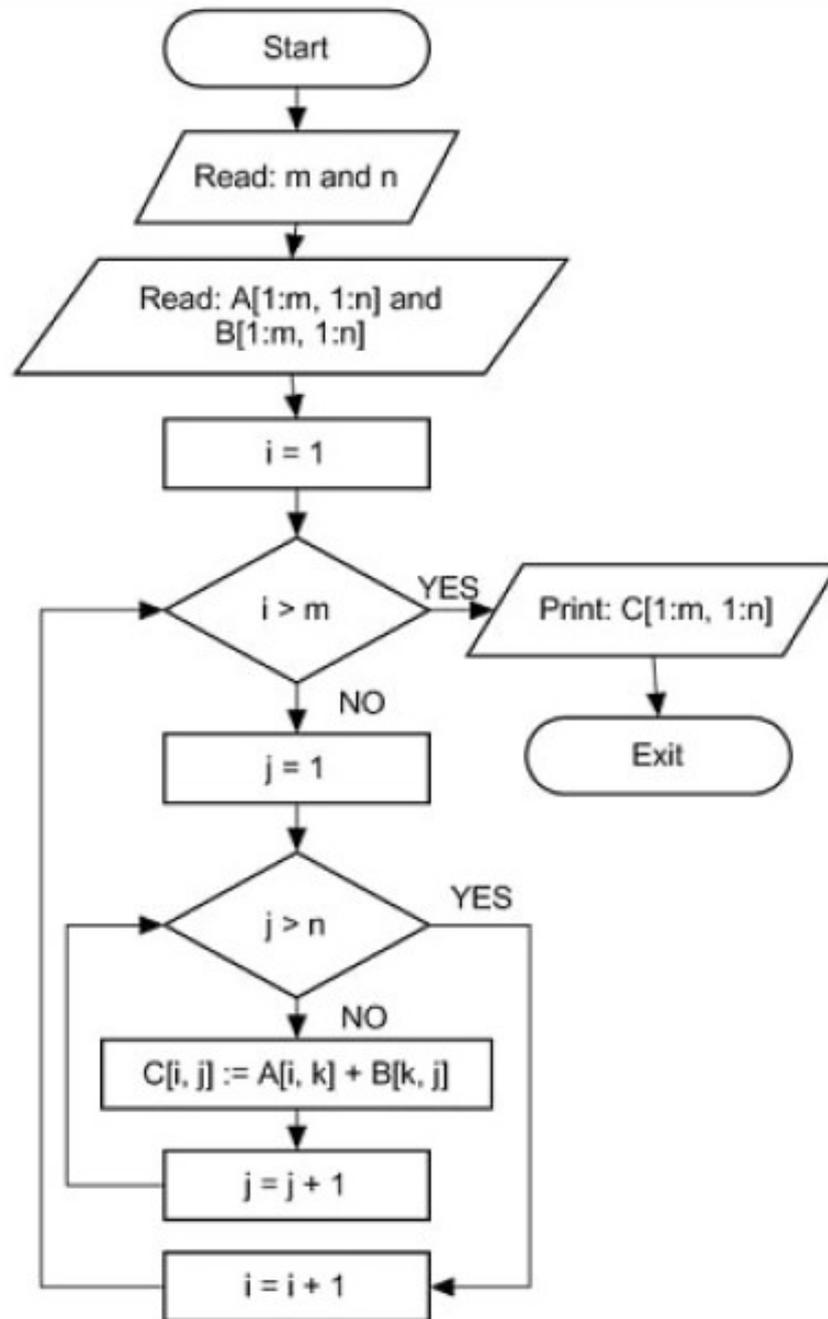
### **Matrix Addition Algorithm:**

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrices row wise using loops
4. Add the corresponding elements of the matrices using nested loops
5. Print the resultant matrix as console output
6. Stop

---

## Flow Chart for Matrix Addition

---



---

### Scilab Code for Matrix Addition

```
clc
m=input("enter number of rows in the Matrices: ");
n=input("enter number of columns in the Matrices: ");
disp('enter the first Matrix')
for i=1:m
for j=1:n
A(i,j)=input('\ ');
end
end
disp('enter the second Matrix')
for i=1:m
for j=1:n
B(i,j)=input('\ ');
end
end
for i=1:m
for j=1:n
C(i,j)=A(i,j)+B(i,j);
end
end
disp('The first matrix is')
disp(A)
disp('The Second matrix is')
disp(B)
disp('The sum of the two matrices is')
disp(C)
```

---

## Matrix Addition using functions

```
// Save file as addition.sce
clc
function [ ]=addition(m, n, A, B)
C=zeros(m,n);
C=A+B;
disp('The first matrix is')
disp (A)
disp('The Second matrix is')
disp (B)
disp('The sum of two matrices is')
disp (C)
endfunction
```

---

## 3.2 Matrix Multiplication

- $A_{m \times n} \times B_{n \times p} = AB_{m \times p}$

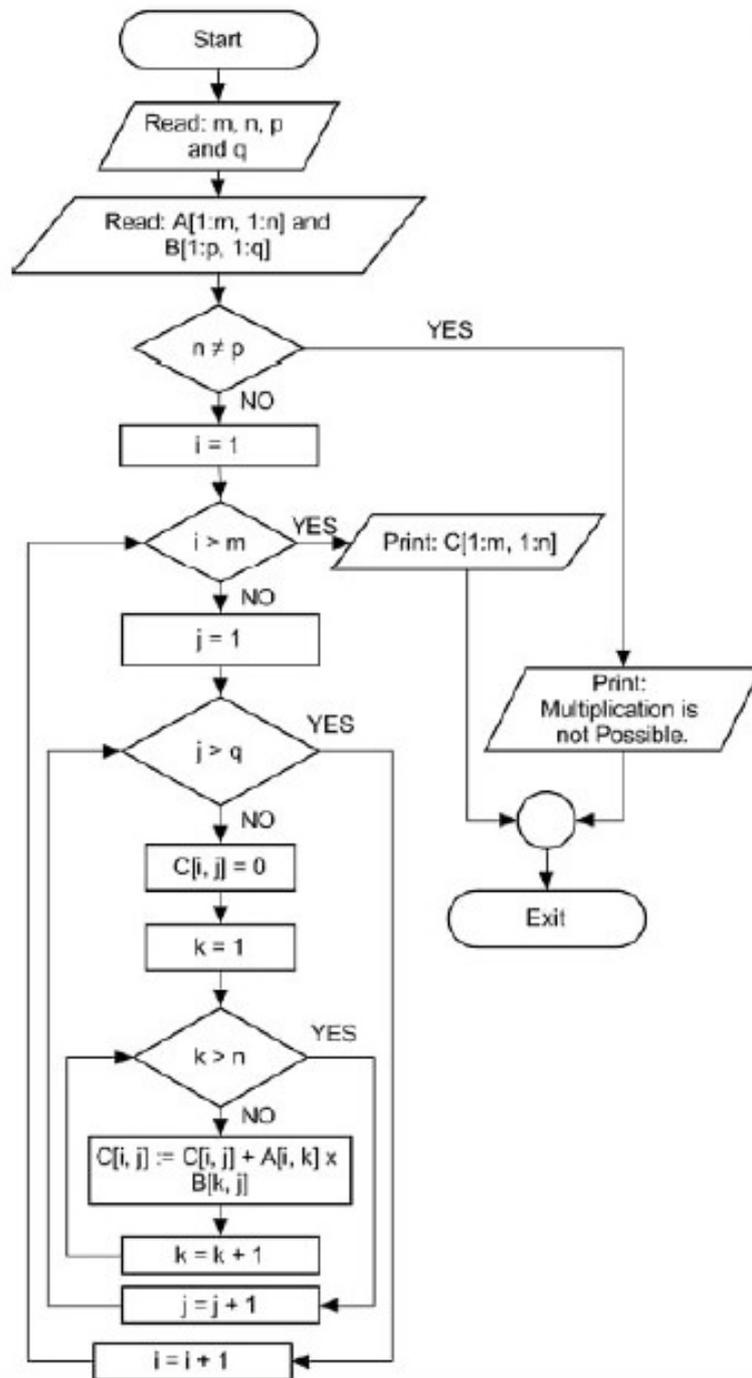
Where  $(AB)_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \dots + A_{in}B_{nj}$

- The number of columns in the first matrix must be equal to the number of rows in the second matrix
- Matrix multiplication is not commutative

### Matrix Multiplication Algorithm:

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrices row wise using loops
4. Check the number of rows and column of first and second matrices
5. If number of rows of first matrix is equal to the number of columns of second matrix, go to step 6. Otherwise, print 'Matrices are not conformable for multiplication' and go to step 3
6. Multiply the matrices using nested loops
7. Print the product in matrix form as console output
8. Stop

## Flow Chart for Matrix Multiplication



---

## Scilab Code for Matrix Multiplication

```
clc
m=input("Enter number of rows in the first Matrix: ");
n=input("Enter number of columns in the first Matrix: ");
p=input("Enter number of rows in the second Matrix: ");
q=input("Enter number of columns in the second Matrix: ");
if n==p
disp('Matrices are conformable for multiplication')
else
disp('Matrices are not conformable for multiplication')
break;
end
disp('enter the first Matrix')
for i=1:m
for j=1:n
A(i,j)=input('\ ');
end
end
disp('enter the second Matrix')
for i=1:p
for j=1:q
B(i,j)=input('\ ');
end
end
C=zeros(m,q);
for i=1:m
for j=1:q
for k=1:n
C(i,j)=C(i,j)+A(i,k)*B(k,j);
end
end
end
```

---

```
end
disp('The first matrix is')
disp(A)
disp('The Second matrix is')
disp(B)
disp('The product of the two matrices is')
disp(C)
```

### **Matrix Multiplication using functions**

```
// Save file as multiplication.sce
clc
function [ ] = multiplication(m, n, p, q, A, B)
C=zeros(m,n);
if n==p
disp('Matrices are conformable for multiplication')
else
disp('Matrices are not conformable for multiplication')
break;
end
C=A*B
disp('The first matrix is')
disp (A)
disp('The Second matrix is')
disp (B)
disp('The multiplication of two matrices is')
disp (C)
endfunction
```

---

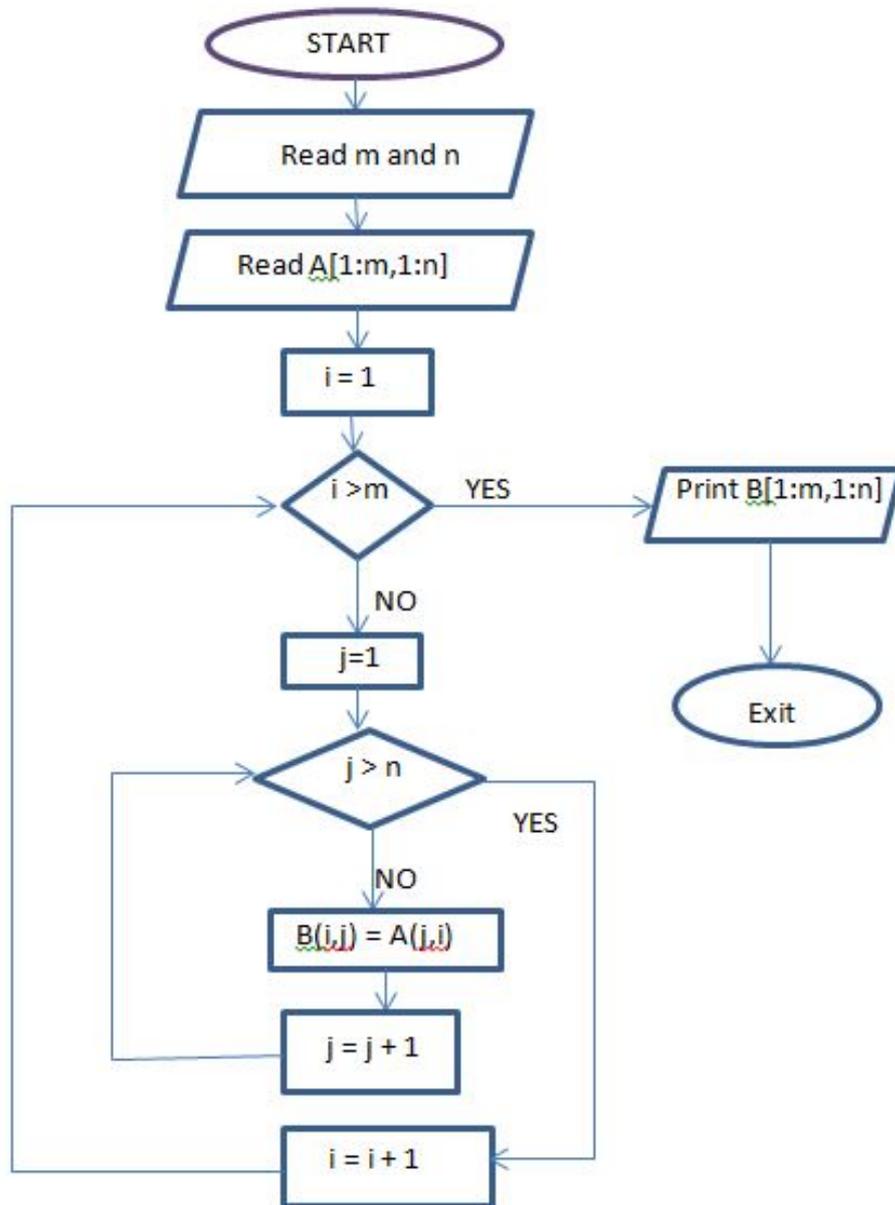
### 3.3 Matrix Transpose

- The transpose of an  $m \times n$  matrix  $A$  is  $n \times m$  matrix  $A^T$
- Formed by interchanging rows into columns and vice versa
- $(A^T)_{i,j} = A_{j,i}$

#### Matrix Transpose Algorithm:

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrix by row wise using loop
4. Interchange rows to columns using nested loops
5. Print the transposed matrix as console output
6. Stop

## Flow Chart for Matrix Transpose



---

### Scilab Code for Matrix Transpose

```
clc
m=input("Enter number of rows in the Matrix: ");
n=input("Enter number of columns in the Matrix: ");
disp('Enter the Matrix')
for i=1:m
for j=1:n
A(i,j)=input('\ ');
end
end
B=zeros(n,m);
for i=1:n
for j=1:m
B(i,j)=A(j,i)
end
end
disp('Entered matrix is')
disp(A)
disp('Transposed matrix is')
disp(B)
```

#### **Matrix Transpose using functions**

```
// Save file as transpose.sce
function [ ]=transpose(m, n, A)
B=zeros(m,n);
B=A'
disp('The matrix is')
disp (A)
disp('Transposed matrix is')
disp (B)
endfunction
```

---

### 3.4 Viva Questions

1. What do you mean by an array?
2. What is difference between 1-Dimensional array and 2-Dimensional array?
3. What will be the order of matrix  $BA$  formed by multiplication of matrices  $A$  and  $B$  of orders  $2 \times 3$  and  $3 \times 2$  respectively?
4. Define transpose of a matrix ? What will be the order of matrix  $A^T$ , if the order of matrix  $A$  is  $4 \times 4$
5. Can you perform term by term multiplication of matrices  $A$  and  $B$  of orders  $2 \times 3$  each? If yes, how to perform this using Scilab commands?
6. Differentiate between *for* loop and *while* loop.

## Chapter 4

# Inverse of a $3 \times 3$ Matrix Using Gauss Jordan Method

---

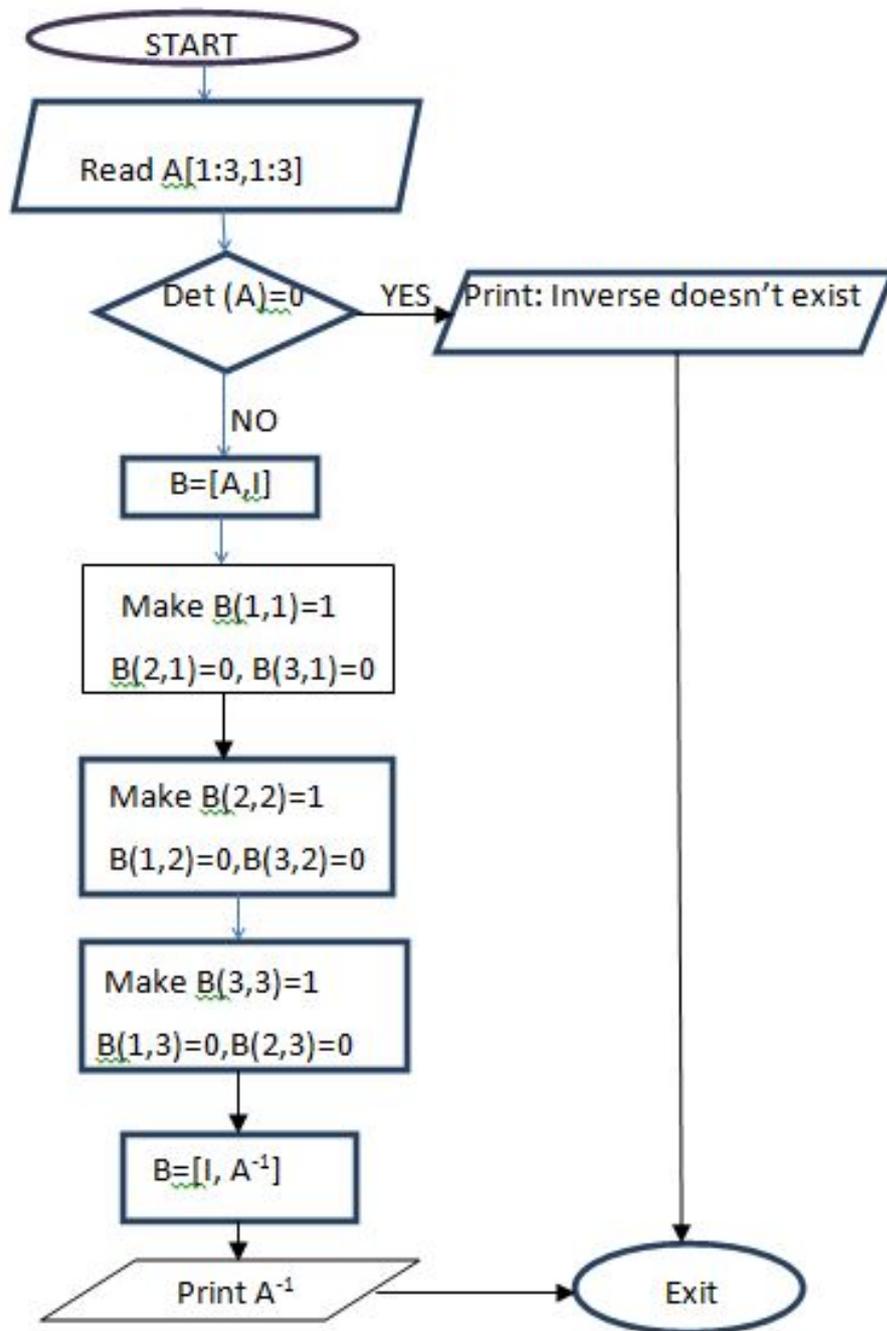
# Matrix Inverse

- The inverse of a square matrix  $A$  denoted by  $A^{-1}$  is such that  $AA^{-1} = I$
- Inverse of a square matrix exists if and only if matrix is non singular

## Matrix Inverse Algorithm:

1. Start
2. Enter the elements of the  $3 \times 3$  matrix row wise using loop
3. Check  $Det(A)$ . If  $Det(A) = 0$ , Print 'Inverse does not exist' else go to step 4.
4. Make an augmented matrix  $B = [A, I]$ , where  $I$  is unit matrix.
5. Make element  $B(1, 1) = 1$  and using this make  $B(2, 1) = 0$  and  $B(3, 1) = 0$  using row/column transformations.
6. Make element  $B(2, 2) = 1$  and using this make  $B(1, 2) = 0$  and  $B(3, 2) = 0$ .
7. Make element  $B(3, 3) = 1$  and using this make  $B(1, 3) = 0$  and  $B(2, 3) = 0$ .
8. Thus final augmented matrix is in the form  $B[I, A^{-1}]$ . Print  $A^{-1}$
9. Stop

## Flow Chart for Matrix Inverse



---

## Scilab Code for Matrix Inverse using Gauss Jordan Method

```
clc
disp('Enter a 3 by 3 matrix row-wise, make sure that diagonal elements are non
-zeros')
for i=1:3
for j=1:3
A(i,j)=input('\');
end
end
disp('Entered Matrix is')
disp(A)
if det(A)==0
disp('Matrix is singular, Inverse does not exist')
break;
end
//Taking the augmented matrix
B=[A eye(3,3)]
disp('Augumented matrix is:')
disp(B)
// Making B(1,1)=1
B(1,:) = B(1,+)/B(1,1);
//Making B(2,1) and B(3,1)=0
B(2,:) = B(2,:) - B(2,1)*B(1,);
B(3,:) = B(3,:) - B(3,1)*B(1,);
//Making B(2,2)=1 and B(1,2), B(3,2)=0
B(2,:) = B(2,+)/B(2,2);
B(1,:) = B(1,:) - B(1,2)*B(2,);
B(3,:) = B(3,:) - B(3,2)*B(2,);
//Making B(3,3)=1 and B(1,3), B(2,3)=0
B(3,:) = B(3,+)/B(3,3);
B(1,:) = B(1,:) - B(1,3)*B(3,);
```

---

```

B(2,:) = B(2,:) - B(2,3)*B(3,:);
disp('Augmented matrix after row operations is:')
disp(B)
B(:,1:3)=[ ]
disp('Inverse of the Matrix is')
disp(B)

```

### **Matrix Inverse using functions**

```
// Save file as transpose.sce
```

```

function [ ]=inverse(m, A)
C=zeros(m,m);
B=det(A)
if B==0
disp('Matrix is singular, Inverse does not exist')
break;
end
C=inv(A)
disp('The matrix is')
disp (A)
disp('Inverse of given matrix is:')
disp (C)
endfunction

```

---

## 4.1 Viva Questions

1. Does the inverse of a rectangular matrix exist? If no, why?
2. What is a singular matrix? Discuss inverse of a singular matrix exist?
3. Discuss the algorithm for finding inverse of a  $3 \times 3$  matrix using Gauss Jordan method.
4. How would you represent second row of a  $3 \times 3$  matrix in scilab?

## Chapter 5

# Eigen Values and Eigen Vectors of a $2 \times 2$ Matrix

---

# *Eigen Values and Eigen Vectors*

- Eigenvalues are a special set of scalars associated with a  $n \times n$  matrix.
- These are roots of the characteristic equation  $|A - \lambda I = 0|$ , hence also known as characteristic roots or latent roots.
- $X$  is a column vector associated with an eigen value  $\lambda$  such that  $AX = \lambda X$

## **Algorithm to find eigen values and eigen vectors of a $2 \times 2$ matrix**

1. Start
2. Enter the elements of the  $2 \times 2$  matrix  $A$ , row wise using loop
3. Find  $Trace(A)$  and  $Det(A)$  as the 2 eigen values  $\lambda_1, \lambda_2$  are roots of the characteristic equation  $\lambda^2 - trace(A) + det(A) = 0$

4. Find  $\lambda_1$  and  $\lambda_2$

5. Find corresponding eigen vectors  $X_1$  and  $X_2$  as follows:

If  $a_{1,2} \neq 0$ , calculate  $X_1$  and  $X_2$  from the first row

If  $a_{2,1} \neq 0$ , calculate  $X_1$  and  $X_2$  from the second row

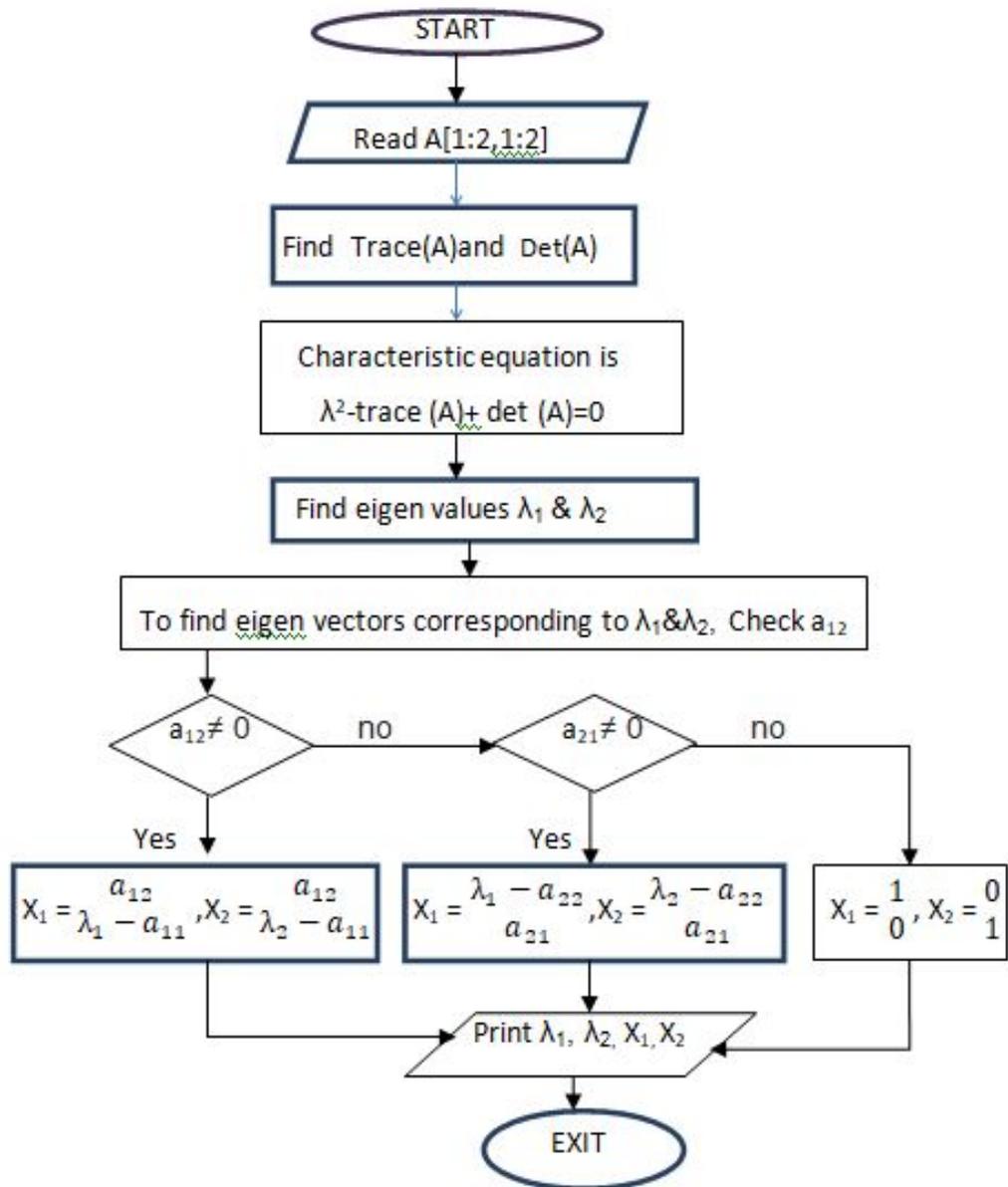
If  $a_{1,2} = 0$  and  $a_{2,1} = 0$ ,

$$X_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

6. Stop

## Flow Chart for Eigen Values and Vectors



---

### Scilab Code for Eigen Values of a $2 \times 2$ Matrix

```
disp('Enter the 2 by 2 Matrix row-wise')
for i=1:2
for j=1:2
A(i,j)=input('\ ');
end
end
b=A(1,1)+A(2,2);
c=A(1,1)*A(2,2)-A(1,2)*A(2,1);
// characteristic equation is  $\lambda^2 - trace(A) + det(A) = 0$ , here  $\lambda_1 \equiv e1, \lambda_2 \equiv e2$ 
e1 = (b + sqrt(b^2 - 4 * c))/2;
e2 = (b - sqrt(b^2 - 4 * c))/2;
if A(1, 2)~=0
X1 = [A(1,2); e1-A(1,1)];
X2 = [A(1,2); e2-A(1,1)];
elseif A(2, 1)~=0
X1 = [e1-A(2,2); A(2,1)];
X2 = [e2-A(2,2); A(2,1)];
else
X1 = [1; 0];
X2 = [0; 1];
end
disp('First Eigen value is:');
disp(e1)
disp('First Eigen vector is:');
disp (X1)
disp('Second Eigen value is:');
disp(e2)
disp('Second Eigen vector is:');
disp (X2)
```

---

## 5.1 Viva Questions

1. What are characteristic roots and characteristic equation of a matrix?
2. Discuss eigen values and eigen vectors of matrices? How many eigen values exist for a  $3 \times 2$  matrix?
3. Can a matrix have a zero eigen value? If yes, what kind of matrix it is?. If  $A$  is a matrix with eigen values 1, 2, 3, then What would be the eigen values of matrices  $A^2$ ,  $A^T$  and  $5A$ ?
4. Discuss the algorithm for finding eigen values and vectors for a  $2 \times 2$  matrix.

# Chapter 6

## Measures of Central Tendency

---

## 6.1 Measures of Central Tendency (Ungrouped Data)

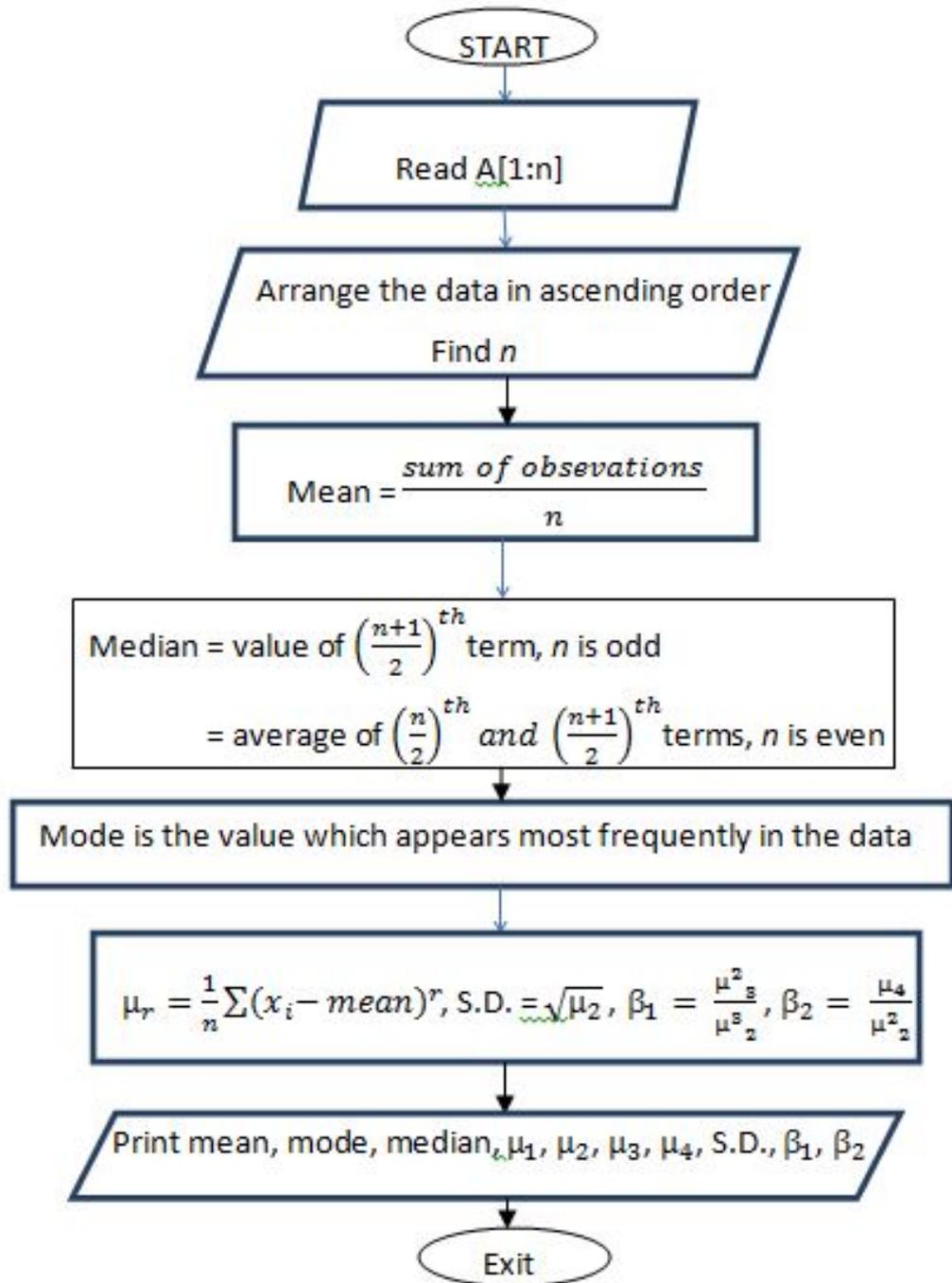
- Arithmetic mean or average is the sum of a collection of numbers divided by the number of observations,  $\text{Mean}(\bar{x}) = \sum \frac{x}{n}$
- The median is the middle score for a set of the data that has been arranged in ascending or descending order of magnitude.  
If  $n$  is odd, Median = Value of  $\frac{(n+1)^{th}}$  observation  
If  $n$  is even, Median = Average value of  $\frac{n^{th}}$  and  $(\frac{n}{2} + 1)^{th}$  observations
- The mode of a set of data is that value which appears most frequently in the set.
- The  $r^{th}$  moment about any point  $a$  of a distribution is denoted by  $\mu_r'$  and is given by  $\mu_r' = \frac{1}{n} \sum (x_i - a)^r$ , where  $n$  is the number of observations  
In particular  $r^{th}$  moment about mean  $\bar{x}$  is given by  $\mu_r = \frac{1}{n} \sum (x_i - \bar{x})^r$
- $\mu_0 = 1, \mu_1 = 0, \mu_2 = \sigma^2 = \text{variance}$
- Skewness denotes the opposite of symmetry  
$$\beta_1 = \frac{\mu_3'^2}{\mu_2'^3}$$
$$\beta_2 = \frac{\mu_4'}{\mu_2'^2}$$

### Algorithm to find mean, median, mode and moments of ungrouped data

1. Start
2. Arrange the data in ascending order
3. Find number of observations ( $n$ )
4. Evaluate  $Mean = \frac{\text{sum of observations}}{n}$

- 
5. Compute Median = Value of  $\frac{(n+1)^{th}}{2}$  observation, if  $n$  is odd,  
= Average value of  $\frac{n}{2}^{th}$  and  $(\frac{n}{2} + 1)^{th}$  observations, if  $n$  is even
  6. For finding mode of given data:
    - i* Arrange the data in ascending order
    - ii* Find the difference between the adjacent elements
    - iii* Find the indices at which the value is non zero
    - iv* Locate the position where there is largest gap between the non zero indices
    - v* Term at next position gives mode
  7. Find  $r^{th}$  moment about mean  $\bar{x}$  using  $\mu_r = \frac{1}{N} \sum (x_i - \bar{x})^r$
  8. Evaluate S.D. =  $\sqrt{\mu_2}$ ,  $\beta_1 = \frac{\mu_3}{\mu_2^{3/2}}$ ,  $\beta_2 = \frac{\mu_4}{\mu_2^2}$
  9. Stop

Flow Chart for Central Tendencies (Ungrouped Data)



---

```

//Scilab Code to find mean, mode, median, moments, skewness and kurtosis
of linear data
clc
function [ ]= moments(A)
B=gsort(A);
n = length(B);
meanA = sum(B)/n;
if pmodulo(n,2)==0
medianA =((B(n/2)+B(n/2 +1)))/2;
else medianA = B((n+1)/2);
end
C = diff(B)
//C= diff(B) calculates differences between adjacent elements of B along the first
array dimension whose size exceeds 1:
//If B is a vector of length n, then C = diff(B) returns a vector of length n-1.
The elements of C are the differences between adjacent elements of B.
//C = [B(2)-B(1)   B(3)-B(2)..... B(m)-B(m-1)]
D = find(C)
//D = find(C) finds the indices(positions), where value is non zero
E = diff(D)
[m k] = max(E) // maximum 'm' at  $k^{th}$  position
modeA = B(D(k)+1)
printf('Mean of the given data is : %f \n \n', meanA);
printf('Median of the given data is : %f \n \n', medianA);
printf('Mode of the given data is : %f \n \n', modeA);
printf('First moment about the mean(M1)= %f \n \n', 0);
for i=1:n
X(i)=A(i)-meanA;
end
M2 = sum(X.*X)/n;
M3 = sum(X.*X.*X)/n;

```

---

```
M4 = sum(X.*X.*X.*X)/n;
printf('Second moment about the mean(M2)= %f \n \n', M2);
printf('Third moment about the mean(M3)= %f \n \n', M3);
printf('Fourth moment about the mean(M4)= %f \n \n', M4);
sd= sqrt (M2);
printf('Standard deviation: %f \n \n', sd);
Csk= (meanA - modeA)/sd;
printf('Coefficient of skewness: %f \n \n', Csk);
Sk= (M3)^2/(M2)^3;
printf('Skewness: %f \n \n', Sk);
Kur= M4/(M2)^2;
printf('Kurtosis: %f \n \n', Kur);
endfunction
```

---

## 6.2 Measures of Central Tendency (Grouped Data)

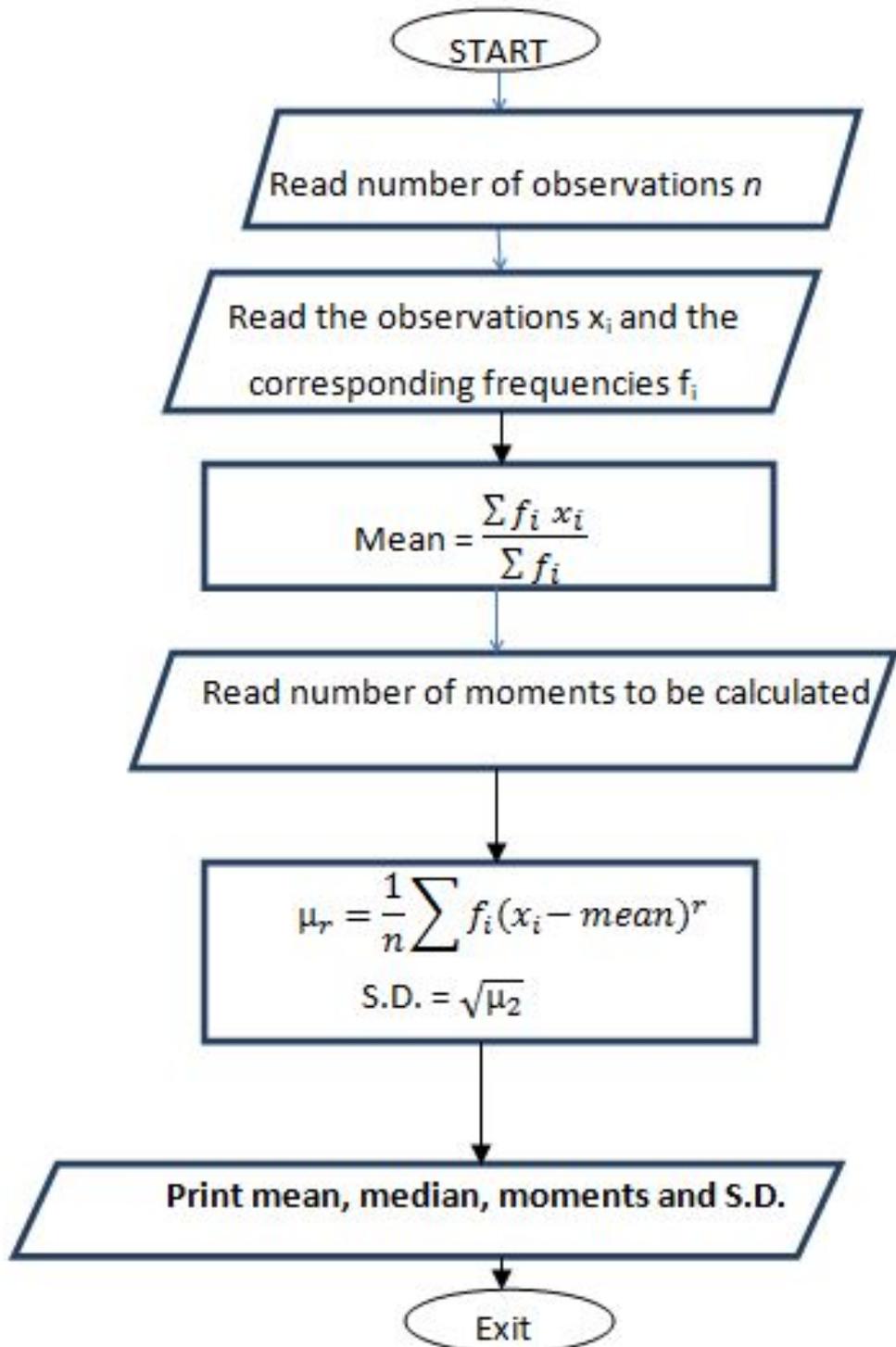
- $Mean = \frac{\sum f_i x_i}{\sum f_i}$
- The  $r^{th}$  moment about any point  $a$  of a distribution is denoted by  $\mu_r'$  and is given by  $\mu_r' = \frac{1}{N} \sum f_i (x_i - a)^r$ , where  $N = \sum f_i$   
In particular  $r^{th}$  moment about mean  $\bar{x}$  is given by  $\mu_r = \frac{1}{N} \sum f_i (x_i - \bar{x})^r$

### Algorithm to find mean, median, mode and moments of ungrouped data

1. Start
2. Enter the number of observations ( $n$ )
3. Input the observations using loop
4. Input frequency of each observation using loop
5. Compute sum of all frequencies  $\sum f_i$
6. Compute  $\sum f_i x_i$
7. Find  $mean = \frac{\sum f_i x_i}{\sum f_i}$
8. Enter how many moments to be calculated ( $r$ )
9. Compute  $r$  moments about mean in a loop
10. Evaluate S.D. =  $\sqrt{\mu_2}$
11. Print Mean, Moments and Standard Deviation
12. Stop

---

Flow Chart for Central Tendencies(Grouped Data)



---

### Scilab code for Mean and Moments of Grouped Data

```
clc
n=input('Enter the no. of observations:');
disp('Enter the values of xi');
for i=1:n
x(i)=input('\');
end;
disp('Enter the corresponding frequencies fi:');
sum=0;
for i=1:n
f(i)=input('\');
sum=sum+f(i);
end;
r=input('How many moments to be calculated:');
sum1=0
for i=1:n
sum1=sum1+f(i)*x(i);
end
A=sum1/sum; //Calculate the average
printf('Average=%f \n',A);
for j=1:r
sum2=0;
for i=1:n y(i)=f(i) * (x(i) - A)^ j;
sum2=sum2+y(i);
end
M(j)=(sum2/sum); //Calculate the moments
printf('Moment about mean M(%d)=%f \n',j,M(j));
end
sd=sqrt(M(2)); //Calculate the standard deviation
printf('Standard deviation=%f \n',sd);
```

---

## 6.3 Viva Questions

1. What are the most common measures of central tendencies?
2. How would you differentiate grouped and ungrouped data?
3. Why do we calculate median or mode, when mean is the most common measure of central tendency?
4. Name some measures of dispersion. Which is the most common measure?
5. What does a data with large value of standard deviation interpret? Can standard deviation have a negative value?
6. Define mean and variance for an ungrouped data. What will be mean and variance of a linear set 3, 3, 3, 3,3 ? If all the above values are multiplied by 2, what would the new mean and variance be?
7. Define moments associated with a data set. What is the significance of moments?

# Chapter 7

## Curve fitting

---

## 7.1 Fitting a Straight Line

Let  $y = ax + b$  be the straight line to be fitted to the given set of data points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , then normal equations are:

$$\sum y = a \sum x + nb \quad \dots (1)$$

$$\sum xy = a \sum x^2 + b \sum x \quad \dots (2)$$

### Algorithm to fit a straight line to given set of data points

1. Start
2. Find number of pairs of data points ( $n$ ) to be fitted
3. Input the  $x$  values and  $y$  values.
4. Find  $\sum x$ ,  $\sum y$ ,  $\sum x^2$  and  $\sum xy$
5. Solve the system of equations given by (1) and (2) using matrix method, where

$$A = \begin{bmatrix} \sum x & n \\ \sum x^2 & \sum x \end{bmatrix}$$

$$B = \begin{bmatrix} \sum y \\ \sum xy \end{bmatrix}$$

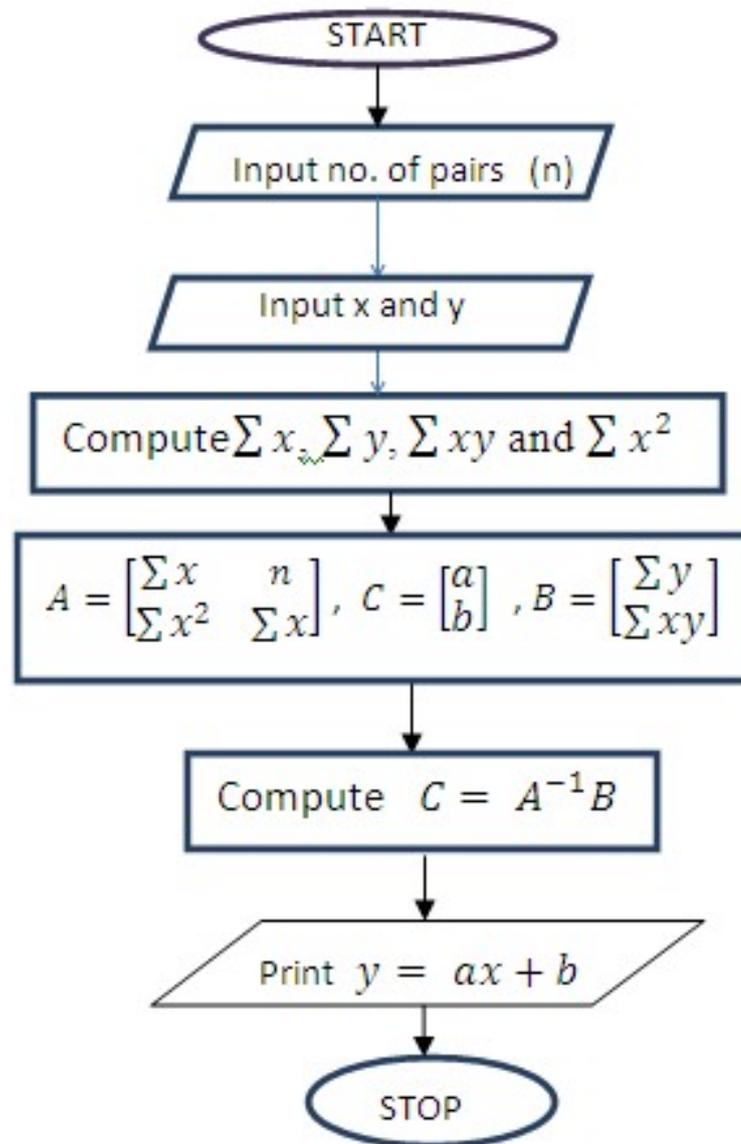
$$C = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$C = A^{-1}B$$

6. Required line of best fit is  $y = ax + b$
7. Stop

---

Flow Chart for fitting a straight line to given set of data points



---

```

//Scilab Code for fitting a straight line to given set of data points (x,y)
clc;
n =input('Enter the no. of pairs of values (x,y):')
disp('Enter the values of x:')
for i=1:n
x(i)=input('    ')
end
disp('Enter the corresponding values of y:')
for i=1:n
y(i)=input('    ')
end
sumx=0; sumx2=0; sumy=0; sumxy=0
for i=1:n
sumx=sumx+x(i);
sumx2=sumx2+x(i)*x(i);
sumy=sumy+y(i);
sumxy=sumxy+x(i)*y(i);
end
A=[sumx n; sumx2 sumx];
B=[sumy;sumxy];
C=inv(A)*B
printf('The line of best fit is y =(%g)x+(%g)',C(1,1),C(2,1))

```

---

## 7.2 Fitting a Parabola

Let  $y = ax^2 + bx + c$  be the parabola to be fitted to the given set of data points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , then normal equations are:

$$\sum y = a \sum x^2 + b \sum x + nc \quad \dots (1)$$

$$\sum xy = a \sum x^3 + b \sum x^2 + c \sum x \quad \dots (2)$$

$$\sum x^2y = a \sum x^4 + b \sum x^3 + c \sum x^2 \quad \dots (3)$$

### Algorithm to fit a parabola to given set of data points

1. Start
2. Find number of pairs of data points ( $n$ ) to be fitted
3. Input the  $x$  values and  $y$  values.
4. Find  $\sum x$ ,  $\sum y$ ,  $\sum x^2$ ,  $\sum x^3$ ,  $\sum x^4$ ,  $\sum xy$  and  $\sum x^2y$
5. Solve the system of equations given by (1), (2) and (3) using matrix method,

where

$$A = \begin{bmatrix} \sum x^2 & \sum x & n \\ \sum x^3 & \sum x^2 & \sum x \\ \sum x^4 & \sum x^3 & \sum x^2 \end{bmatrix}$$

$$B = \begin{bmatrix} \sum y \\ \sum xy \\ \sum x^2y \end{bmatrix}$$

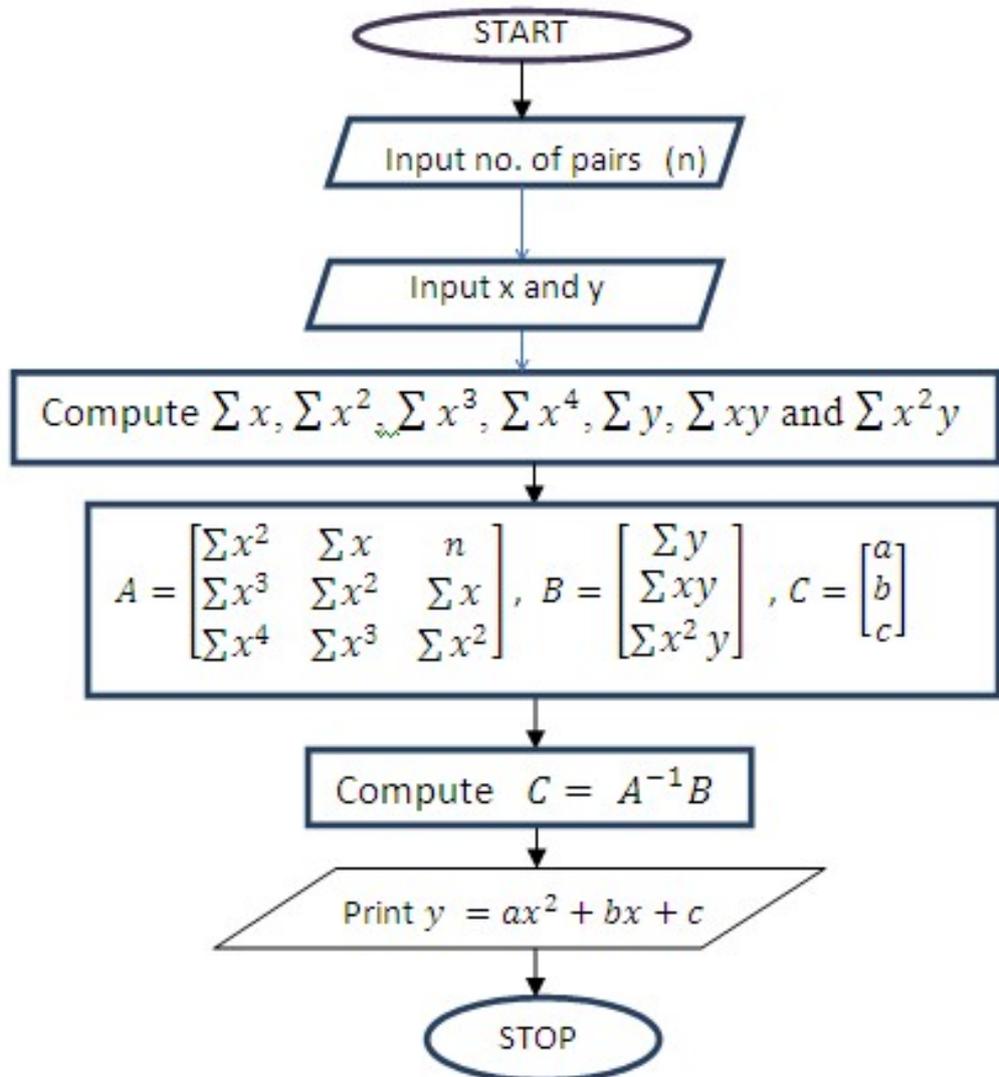
$$C = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$C = A^{-1}B$$

6. Required parabola to be fitted is  $y = ax^2 + bx + c$
7. Stop

---

Flow Chart for fitting a parabola to the given set of data points



---

```

//Scilab Code for fitting a parabola line to given set of data points (x,y)
clc;
n =input('Enter the no. of pairs of values (x,y):')
disp('Enter the values of x:')
for i=1:n
x(i)=input('    ')
end
disp('Enter the corresponding values of y:')
for i=1:n
y(i)=input('    ')
end
sumx=0; sumx2=0; sumx3=0; sumx4=0; sumy=0; sumxy=0; sumx2y=0;
for i=1:n
sumx=sumx+x(i);
sumx2=sumx2+x(i)*x(i);
sumx3=sumx3+x(i)*x(i)*x(i);
sumx4=sumx4+x(i)*x(i)*x(i)*x(i);
sumy=sumy+y(i);
sumxy=sumxy+x(i)*y(i);
sumx2y=sumx2y+x(i)*x(i)*y(i);
end
A=[sumx2 sumx n; sumx3 sumx2 sumx; sumx4 sumx3 sumx2];
B=[sumy;sumxy;sumx2y];
C=inv(A)*B
printf('The fitted parabola is y=(%g)x^ 2+(%g)x+(%g)',C(1,1),C(2,1),C(3,1))

```

---

### 7.3 Viva Questions

1. Define curve fitting.
2. Define the principle of least squares in terms of curve fitting.
3. What are the minimum number of points required to fit a curve?
4. Can we fit a straight line and parabola for same set of points?
5. How many normal equations are required for fitting a straight line and parabola respectively?
6. Define mean and variance for an ungrouped data. What will be mean and variance of a linear set 3, 3, 3, 3,3 ? If all the above values are multiplied by 2, what would the new mean and variance be?
7. Define moments associated with a data set. What is the significance of moments?

## Chapter 8

# Plotting of 2D Graphs Using Scilab

---

# Two Dimensional Graphs

The generic 2D multiple plot is

```
plot2di(x,y,<options>)
```

index of plot2d :  $i = \text{none}, 2, 3, 4$

For the different values of  $i$  we have:

$i = \text{none}$  : piecewise linear/logarithmic plotting

$i = 2$  : piecewise constant drawing style

$i = 3$  : vertical bars

$i = 4$  : arrows style

```
//Specifier Color
```

```
//r Red
```

```
//g Green
```

```
//b Blue
```

```
//c Cyan
```

```
//m Magenta
```

```
//y Yellow
```

```
//k Black
```

```
//w White
```

```
//Specifier Marker Type
```

```
//+ + + Plus sign
```

```
//o o o Circle
```

```
//* * * Asterisk
```

```
//... Point
```

```
//x x x Cross
```

```
//'square' or 's' Square
```

```
//'diamond' or 'd' Diamond
```

```
//^ Upward-pointing triangle
```

```
//v Downward-pointing triangle
```

```
//'pentagram' or 'p' Five-pointed star (pentagram)
```

---

## 8.0.1 A simple graph plotting raw data

```
clc
```

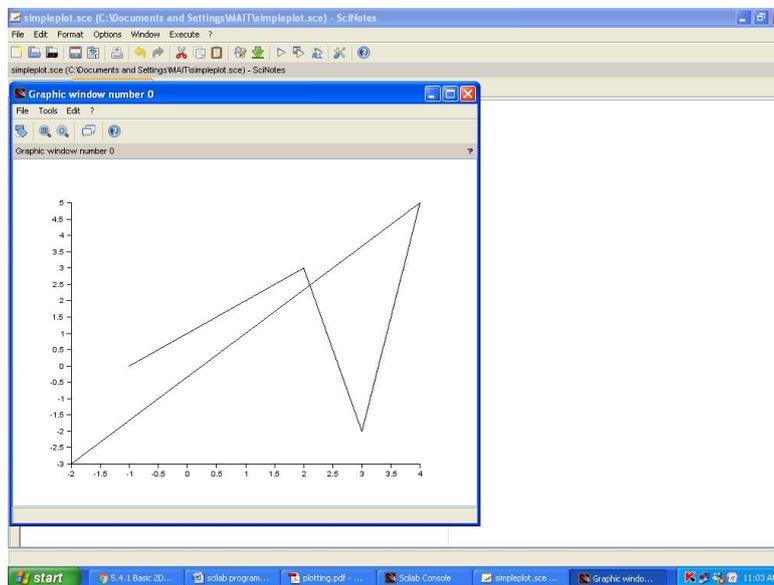
```
x = [1 -1 2 3 4 -2];
```

```
y = [2 0 3 -2 5 -3];
```

```
plot2d(x,y)
```

```
xlabel('x');
```

```
ylabel('y');
```



## 8.0.2 Generation of square wave

```
clc
```

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

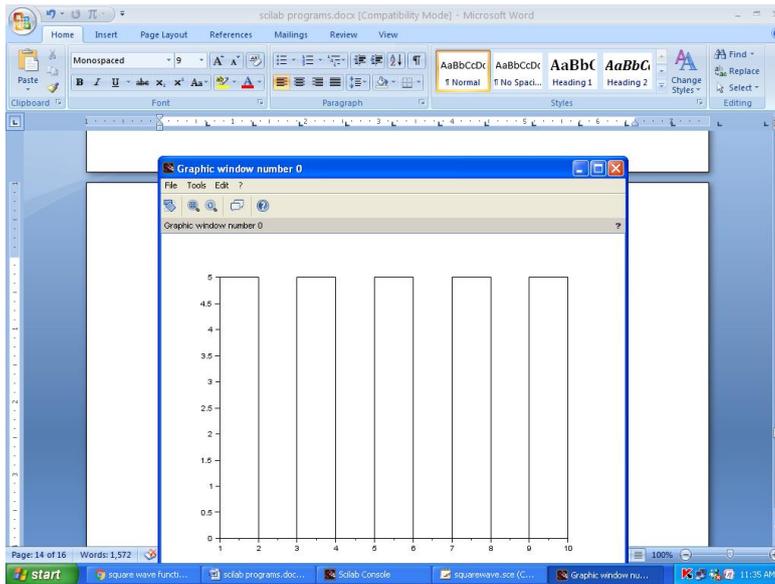
```
y = [5 0 5 0 5 0 5 0 5 0];
```

```
plot2d2(x,y)
```

```
xlabel('x');
```

```
ylabel('y');
```

```
title('Square Wave Function');
```



### 8.0.3 Unit Step function I

clc

```
x = [-1 -2 -3 -4 -5 0 1 2 3 4 5];
```

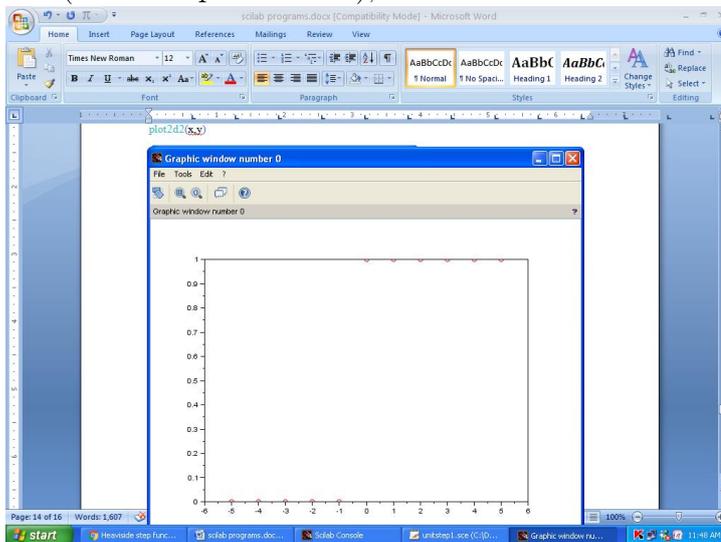
```
y = [0 0 0 0 0 1 1 1 1 1 1];
```

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

```
xlabel('x');
```

```
ylabel('y');
```

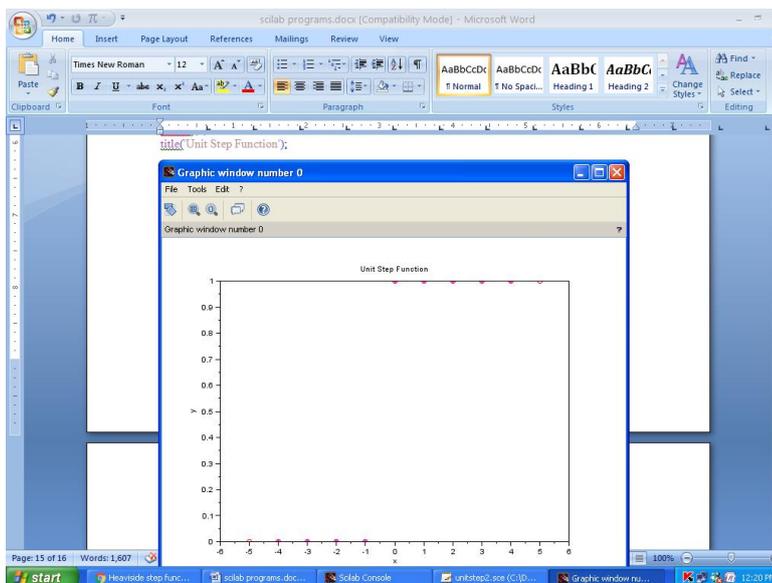
```
title('Unit Step Function');
```



---

## 8.0.4 Unit Step function II

```
function y=unitstep2(x)
y(find (x < 0)) = 0;
y(find (x >= 0)) = 1;
endfunction
clc
// define your independent values
x = [-4 : 1 : 4]';
// call your previously defined function
y = unitstep2(x);
// plot
plot(x, y, 'm*')
xlabel('x');
ylabel('y');
title('Unit Step Function');
```



---

## 8.1 Viva Questions

1. What are the differences between plot2d, plot2d1, plot2d2 and plot2d4 commands?
2. How do we provide title to a scilab plot?
3. How to specify line colour in scilab plot?
4. Define Square Wave function?
5. What is a Unit Step function?

## Chapter 9

# Numerical Methods for Finding Roots of Algebraic and Transcendental Equations

---

## 9.1 Bisection Method

Bisection method is used to find an approximate root in an interval by repeatedly bisecting into subintervals. It is a very simple and robust method but it is also relatively slow. Because of this it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods. The method is also called the interval halving method, binary search method or the dichotomy method. This scheme is based on the intermediate value theorem for continuous functions.

Let  $f(x)$  be a function which is continuous in the interval  $(a, b)$ . Let  $f(a)$  be positive and  $f(b)$  be negative. The initial approximation is  $x_1 = \frac{a+b}{2}$

Then to find the next approximation to the root, one of the three conditions arises:

1.  $f(x_1) = 0$ , then we have a root at  $x_1$ .
2.  $f(x_1) < 0$ , then since  $f(x_1)f(a) < 0$ , the root lies between  $x_1$  and  $a$ .
3.  $f(x_1) > 0$ , then since  $f(x_1)f(b) < 0$ , the root lies between  $x_1$  and  $b$ .

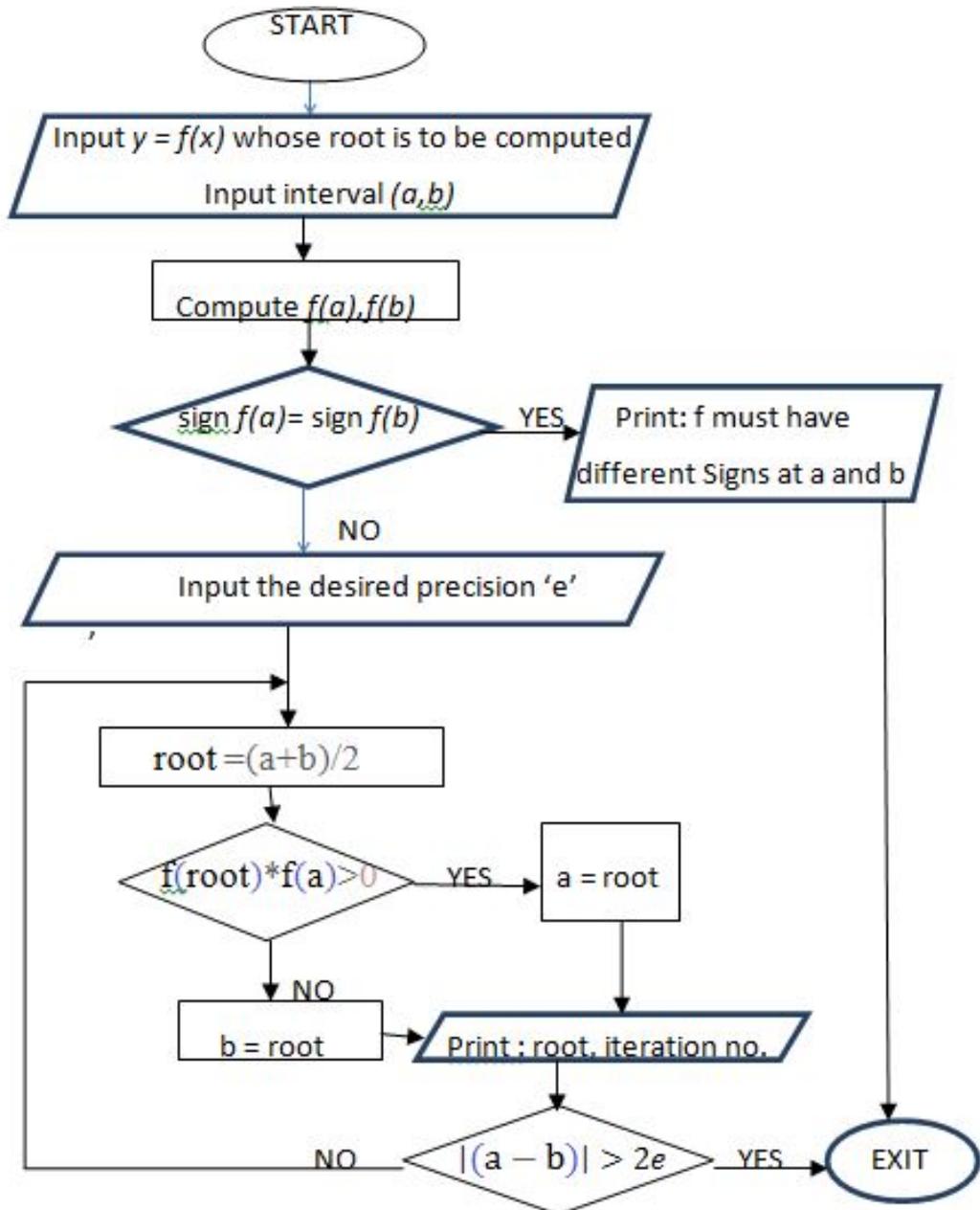
We can further divide this subinterval into two halves to get a new subinterval which contains the root. We can continue the process till we get desired accuracy.

### **Bisection Method Algorithm:**

1. Start
2. Read  $y = f(x)$  whose root is to be computed.
3. Input  $a$  and  $b$  where  $a, b$  are end points of interval  $(a, b)$  in which the root lies.
4. Compute  $f(a)$  and  $f(b)$ .
5. If  $f(a)$  and  $f(b)$  have same signs, then display function must have different signs at  $a$  and  $b$ , exit. Otherwise go to next step.
6. Input  $e$  and set iteration number counter to zero. Here  $e$  is the absolute error i.e. the desired degree of accuracy.
7.  $root = (a + b)/2$

8. If  $f(\text{root}) * f(a) > 0$ , then  $a = \text{root}$  else  $b = \text{root}$
9. Print root and iteration number
10. If  $|a - b| > 2e$ , print the root and exit otherwise continue in the loop
11. Stop

**Flow Chart for Bisection Method**



---

```

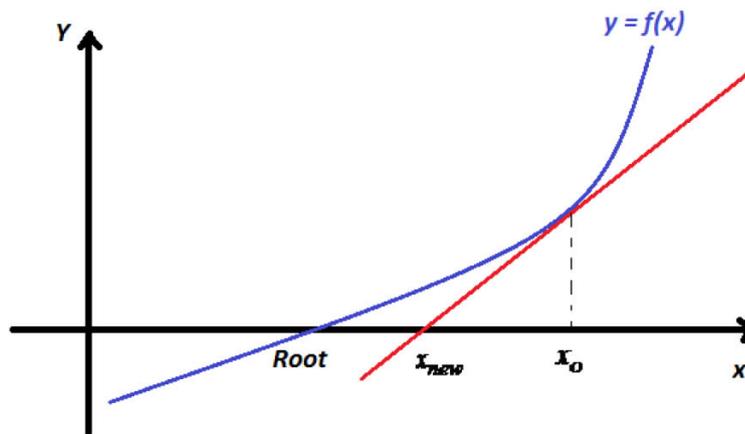
//Scilab Code for Bisection method
clc
deff('y = f(x)', 'y = x^3 + x^2 - 3 * x - 3')
a =input("enter initial interval value: ");
b =input("enter final interval value: ");
//compute initial values of f(a) and f(b)
fa = f(a);
fb = f(b);
if sign(fa) == sign(fb)
// sanity check: f(a) and f(b) must have different signs
disp('f must have different signs at the endpoints a and b')
error
end
e=input(" answer correct upto : ");
iter=0;
printf('Iteration \t a \t\t b \t \t root \t \t f(root)\n')
while abs(a - b) > 2 * e
root = (a + b)/2
printf(' %i\t\t %f \t %f \t %f \t %f \n' ,iter,a,b,root,f(root))
iff (root) * f(a) > 0
a = root
else
b = root
end
iter=iter+1
end
printf('\n \n The solution of given equation is %f after %i Iterations',root,iter - 1)

```

---

## 9.2 Newton-Raphson Method

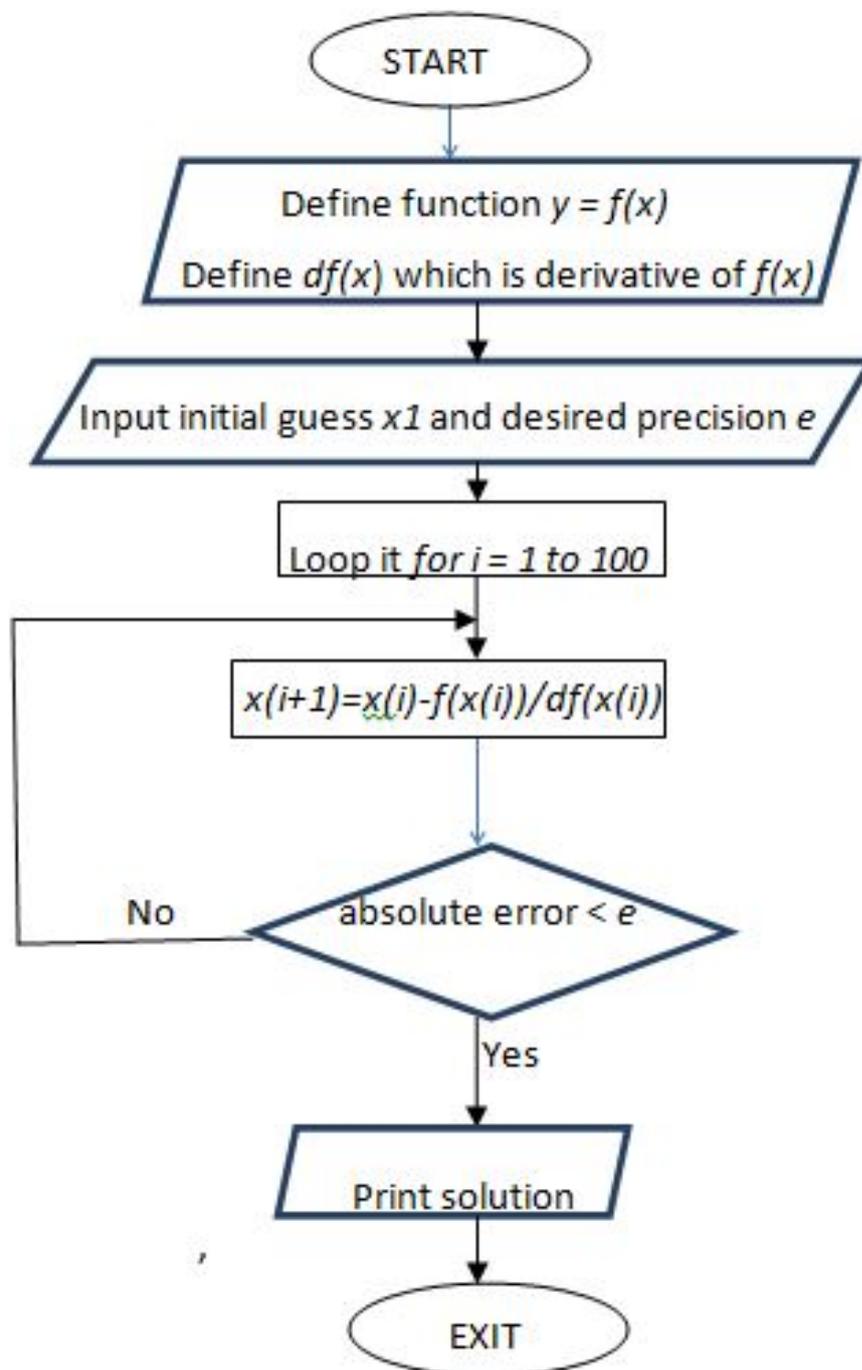
Newton -Raphson method named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots of a real-valued function. The Newton-Raphson method in one variable is implemented as follows:



Let  $x_0$  be an approximate root of the equation  $f(x) = 0$ . Next approximation  $x_1$  is given by  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$  and  $n^{th}$  approximation  $x_1$  is given by  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

### Newton Raphson Method Algorithm:

1. Start
2. Enter the function  $f(x)$  and its first derivative  $f'(x)$
3. Take an initial guess root say  $x_1$  and error precision  $e$ .
4. Use Newtons iteration formula to get new better approximate of the root, say  $x_2$ . Repeat the process for  $x_3, x_4 \dots$  till the actual root of the function is obtained, fulfilling the tolerance of error.



---

### Scilab code for Newton Raphson Method

```
clc
deff('y = f(x)', 'y = x^3 + x^2 - 3 * x - 3')
deff('y = df(x)', 'y = 3 * x^2 + 2 * x - 3')
x(1)=input('Enter Initial Guess:');
e= input(" answer correct upto : ");
for i = 1 : 100
x(i + 1) = x(i) - f(x(i))/df(x(i));
err(i) = abs((x(i + 1) - x(i))/x(i));
if err(i) < e
break;
end
end
printf('the solution is %f',x(i))
```

---

### 9.3 Viva Questions

1. Why do we numerical iterative methods for solving equations? Which are better : Direct or iterative?
2. What do you mean by root of an equation? How many real roots an equation may have between  $a$  and  $b$ , if  $f(a)$  and  $f(b)$  are having same signs?
3. What are algebraic and transcendental equations? Name some numerical methods for solving these equations.
4. If an equation has three real roots, can we interpret how many times it will intersect  $X$  and  $Y$  axis?
5. State giving reasons which of the Bisection method and Newton Raphson methods converges faster.
6. Why Newton Raphson method is more useful when the graph of  $f(x)$  is nearly vertical while crossing  $X$  - axis? What is the initial approximation  $x_0$  in Newton Raphson method in a given interval  $(a,b)$ ?
7. What is order of convergence for Newton Raphson method? Does NR method always converge? What are the limitations of Newton Raphson method?

## Chapter 10

# Numerical Integration for Solving Definite Integrals

Numerical integration is the approximate computation of an integral using numerical techniques. The numerical computation of an integral is sometimes called quadrature. There are a wide range of methods available for numerical integration.

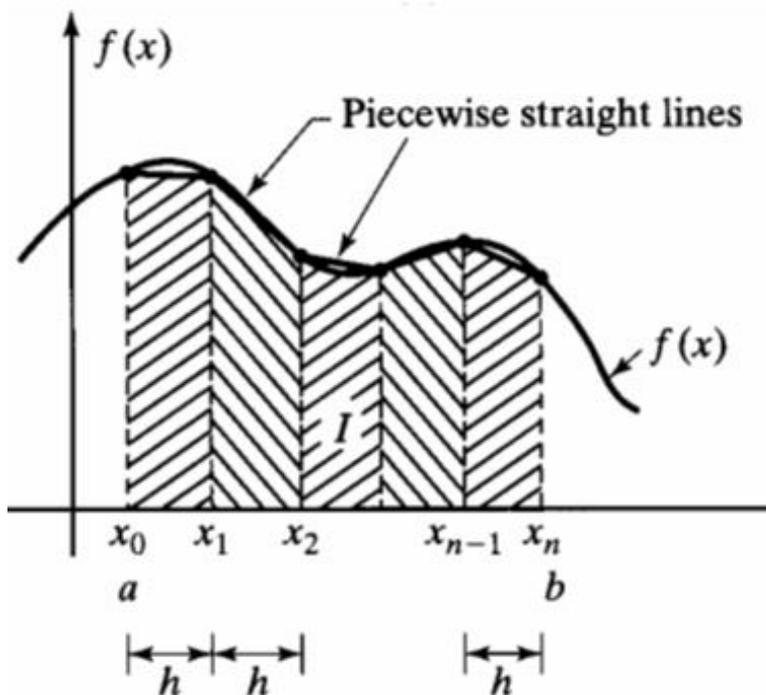
---

## 10.1 The Trapezoidal Rule

The trapezoidal rule uses trapezoids to approximate the area under a graph of a function.

$$\int_a^b f(x)dx = \frac{h}{2}(f_0 + f_n + 2 * \sum_{i=1}^{n-1} f_i)$$

where  $h = \Delta x$ , which can be calculated by  $h = (b - a)/n$ , where  $n$  is the number of segments required.

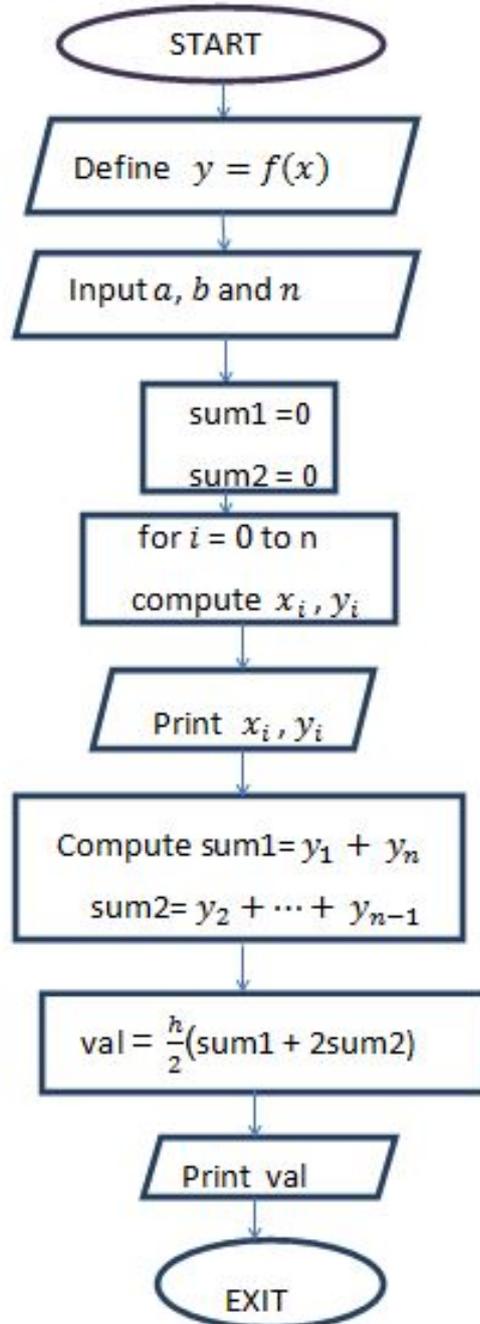


### Numerical Integration by Trapezoidal Rule Algorithm:

1. Start
2. Define and Declare function  $y = f(x)$  whose integral is to be computed.
3. Input  $a, b$  and  $n$ , where  $a, b$  are lower and upper limits of integral  $\int_a^b f(x)dx$  and  $n$  is number of trapezoids in which area is to be divided.
4. Initialize two counters  $sum1$  and  $sum2$  to zero.
5. Compute  $x(i) = a + i * h$  and  $y(i) = f(x(i))$  in a loop for all  $n + 1$  points dividing  $n$  trapezoids.

- 
6.  $sum1 = y(1) + y(n)$  and  $sum2 = y(2) + \dots + y(n - 1)$
  7.  $val = \frac{h}{2}(sum1 + 2sum2)$
  8. Print value of integral.
  9. Stop

#### Flow Chart for Trapezoidal Rule

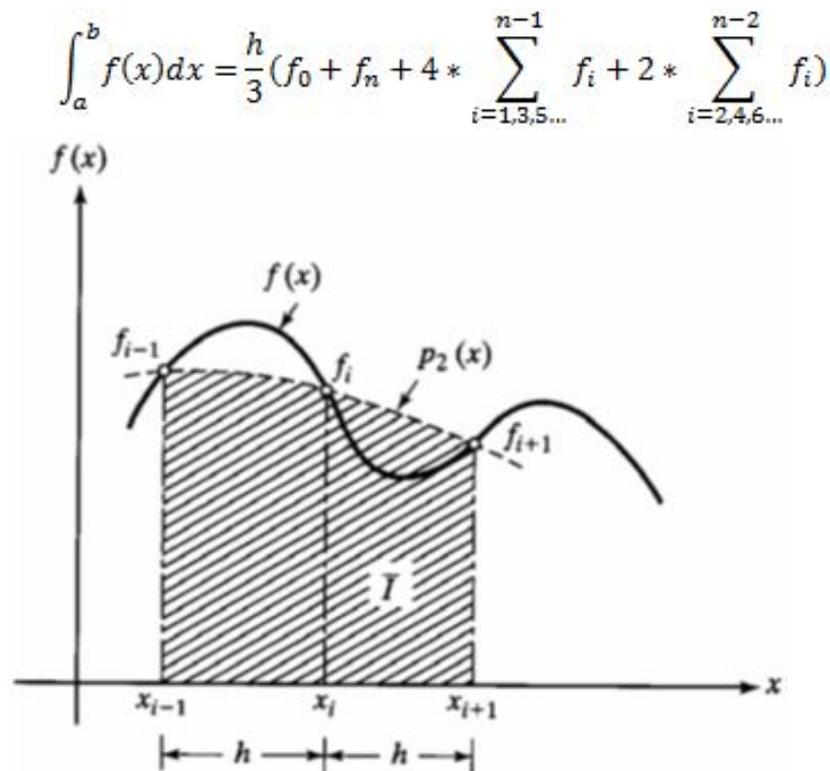


---

```
// Scilab code for Trapezoidal Rule
clc
deff('y = f(x)', 'y = x/(x^2 + 5)');
a=input("Enter Lower Limit: ")
b=input("Enter Upper Limit: ")
n=input("Enter number of sum intervals: ")
h = (b - a)/n
sum1 = 0
sum2 = 0
for i = 0 : n
x = a + i * h
y = f(x)
disp([xy])
if (i == 0)|(i == n)
sum1 = sum1 + y
else
sum2 = sum2 + y
end
end
val = (h/2) * (sum1 + 2 * sum2)
disp(val, "Value of integral by Trapezoidal Rule is:")
```

## 10.2 Simpson's $1/3^{rd}$ Rule

The Simpson's  $1/3^{rd}$  rule is similar to the trapezoidal rule, though it approximates the area using a series of quadratic functions instead of straight lines. It is used if the number of segments is even.

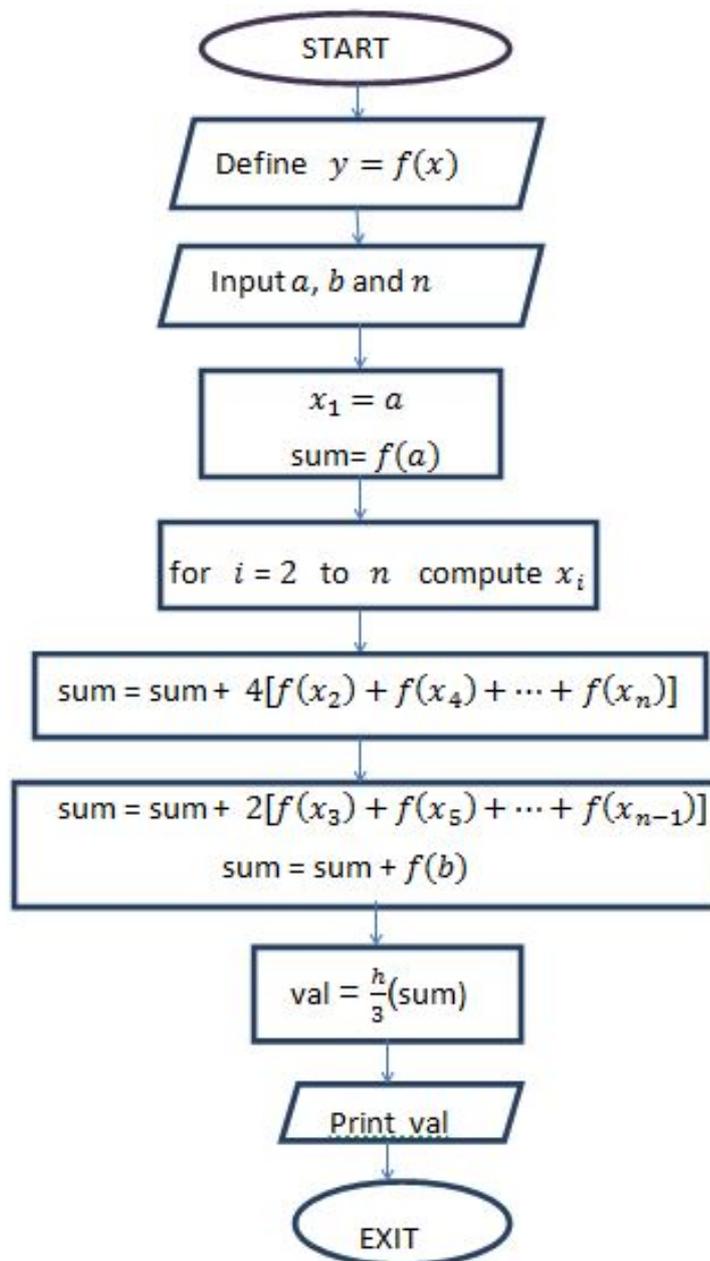


### Numerical Integration by Simpson's $1/3^{rd}$ Rule Algorithm:

1. Start
2. Define and Declare function  $y = f(x)$  whose integral is to be computed.
3. Input  $a$ ,  $b$  and  $n$ , where  $a$ ,  $b$  are lower and upper limits of integral  $\int_a^b f(x)dx$  and  $n$  is number of intervals in which area is to be divided. Note that  $n$  must be even.
4. Put  $x_1 = a$  and initialize  $sum = f(a)$
5. Compute  $x(i) = x(i - 1) + h$

- 
6.  $sum = sum + 4[f(x_2) + f(x_4) + \dots + f(x_n)]$
  7.  $sum = sum + 2[f(x_3) + f(x_5) + \dots + f(x_{n-1})]$
  8.  $sum = sum + f(b)$
  9.  $val = \frac{h}{3}(sum)$
  10. Print value of integral.
  11. Stop

**Flow Chart for Simpson's  $1/3^{rd}$  Rule**



---

```

// Scilab Code for Simpsons 1/3rd Rule
clc
deff('y = f(x)', 'y = x/(x^2 + 5)');
a=input("Enter Lower Limit: ")
b=input("Enter Upper Limit: ")
n=input("Enter number of sum intervals: ")
h = (b - a)/n
x(1) = a;
sum = f(a);
for i = 2 : n
x(i) = x(i - 1) + h;
end
for j = 2 : 2 : n
sum = sum + 4 * f(x(j));
end
for k = 3 : 2 : n
sum = sum + 2 * f(x(k)); end
sum = sum + f(b);
val = sum * h/3;
disp(val, "Value of integral by Simpsons 1/3rd Rule is:")

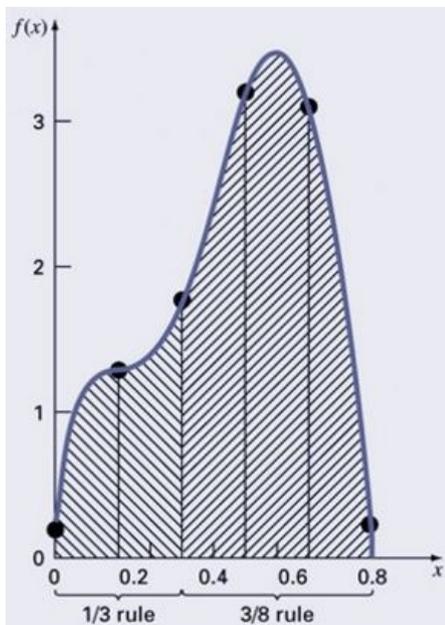
```

---

## 10.3 Simpson's $3/8^{th}$ Rule

The Simpson's  $3/8^{th}$  Rule rule is similar to the  $1/3$  rule. It is used when it is required to take 3 segments at a time. Thus number of intervals must be a multiple of 3.

$$\int_a^b f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + 3f(x_5) + 2f(x_6) + \dots + f(x_n)]$$

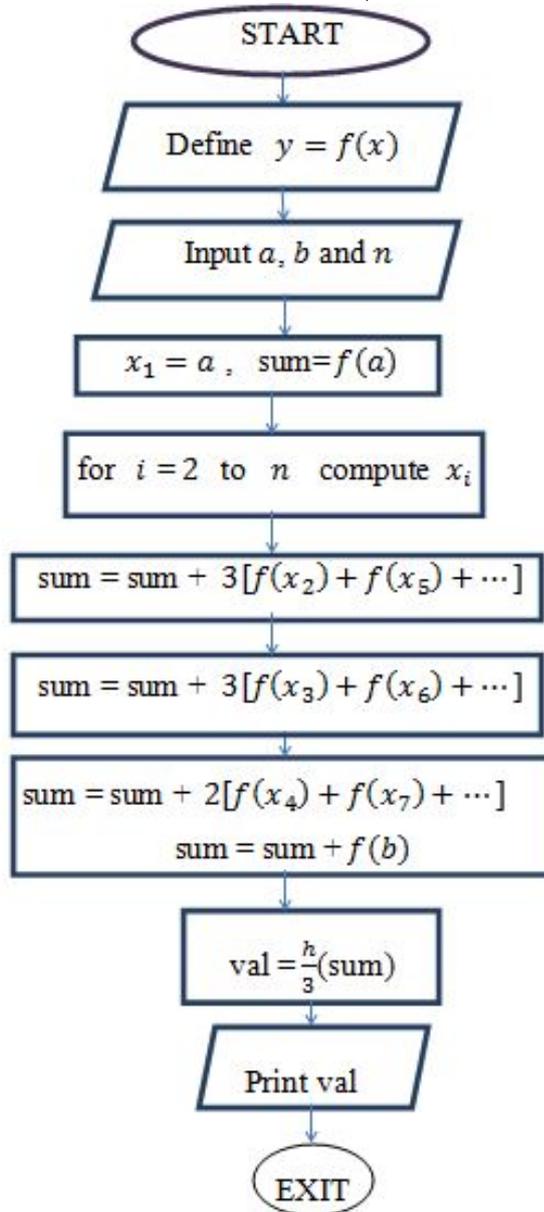


### Numerical Integration by Simpson's $3/8^{th}$ Rule Algorithm:

1. Start
2. Define and Declare function  $y = f(x)$  whose integral is to be computed.
3. Input  $a$ ,  $b$  and  $n$ , where  $a$ ,  $b$  are lower and upper limits of integral  $\int_a^b f(x)dx$  and  $n$  is number of intervals in which area is to be divided. Note that  $n$  must be a multiple of 3.
4. Put  $x_1 = a$  and initialize  $sum = f(a)$
5. Compute  $x(i) = x(i - 1) + h$
6.  $sum = sum + 3[f(x_2) + f(x_5) + \dots]$

- 
7.  $sum = sum + 3[f(x_3) + f(x_6) + \dots]$
  8.  $sum = sum + 2[f(x_4) + f(x_7) + \dots]$
  9.  $sum = sum + f(b)$
  10.  $val = \frac{3h}{8}(sum)$
  11. Print value of integral.
  12. Stop

Flow Chart for Simpson's  $3/8^{th}$  Rule



---

### Scilab Code for Simpson's $3/8^{\text{th}}$ Rule

```
clc
deff('y = f(x)', 'y = x/(x^2 + 5)');
a=input("Enter Lower Limit: ")
b=input("Enter Upper Limit: ")
n=input("Enter number of sum intervals: ")
h = (b - a)/n
x(1) = a;
sum = f(a);
for i = 2 : n
x(i) = x(i - 1) + h;
end
for j = 2 : 3 : n
sum = sum + 3 * f(x(j));
end
for k = 3 : 3 : n
sum = sum + 3 * f(x(k));
end
for l = 4 : 3 : n
sum = sum + 2 * f(x(l));
end
sum = sum + f(b);
val = sum * 3 * h/8;
disp(val, "Value of integral by Simpson's  $3/8^{\text{th}}$  Rule is:")
```

---

## 10.4 Viva Questions

1. What is numerical integration? Is numerical integration valid for indefinite integrals?
2. Explain geometrical significance of Trapezoidal rule.
3. Is Simpson's Method always better than Trapezoidal? Which one is more reliable?
4. What is the condition on number of intervals for Trapezoidal, Simpson's one third and Simpson's three eight rule? Raphson method?
5. What is The highest order of polynomial integrand for which Simpson's one third rule of integration is exact?

## Chapter 11

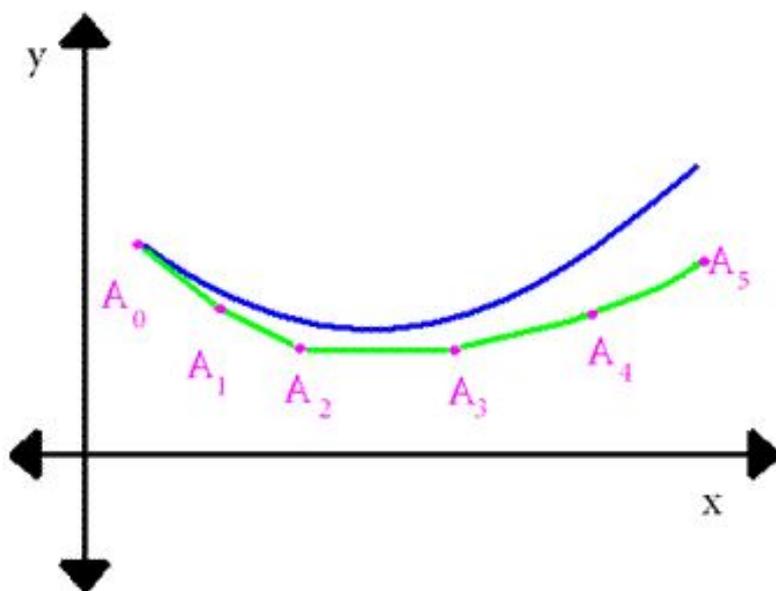
# Numerical Solution of Ordinary Differential equations using Euler's Method

The problem of finding a function  $y$  of  $x$  when we know its derivative and its value  $y_0$  at a particular point  $x_0$  is called an initial value problem.

---

## 11.1 Euler's Method

Euler's Method provides us with an approximation for the solution of a differential equation of the form  $\frac{dy}{dx} = f(x, y)$ ,  $y(x_0) = y_0$ . The idea behind Euler's Method is to use the concept of local linearity to join multiple small line segments so that they make up an approximation of the actual curve, as shown below.



The upper curve shows the actual graph of a function. The curve  $\widetilde{A_0 A_5}$  give approximations using Euler's method. Generally, the approximation gets less accurate the further we go away from the initial value. Better accuracy is achieved when the points in the approximation are chosen in small steps. Each next approximation is estimated from previous by using the formula  $y_{n+1} = y_n + hf(x_n, y_n)$

**Algorithm to solve an initial value problem using Euler's method:**

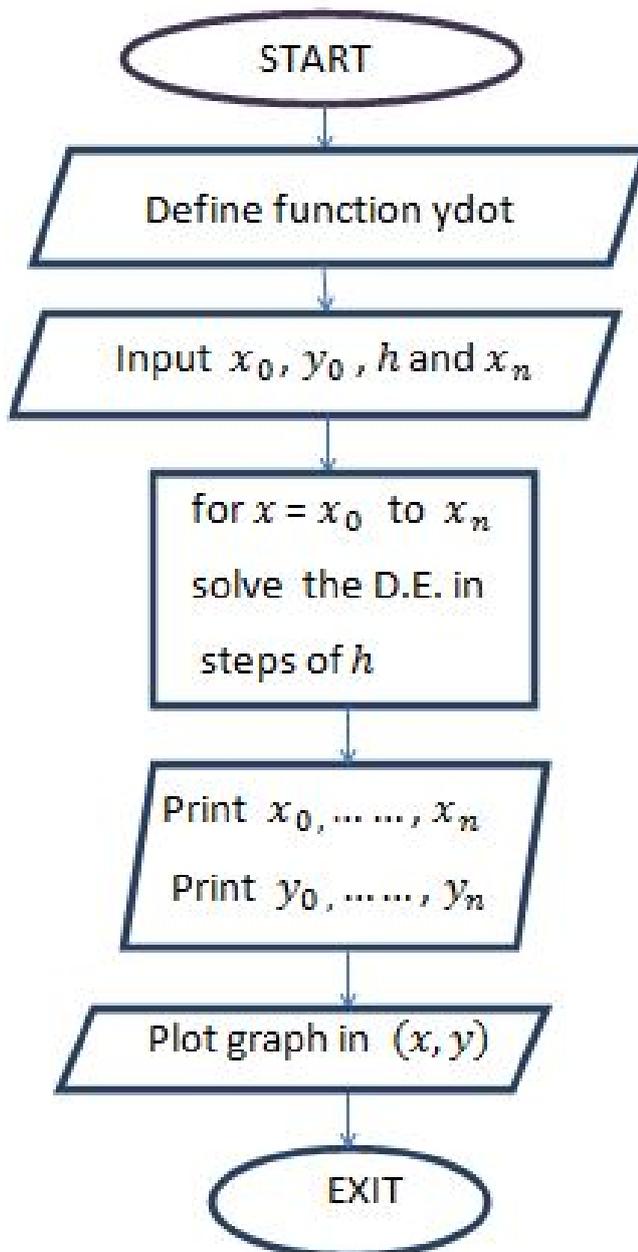
1. Start
2. Define and Declare function ydot representing  $\frac{dy}{dx} = f(x, y)$
3. Input  $x_0, y_0, h$  and  $x_n$  for the given initial value condition  $y(x_0) = y_0$ .  $h$  is

---

step size and  $x_n$  is final value of  $x$ .

4. For  $x = x_0$  to  $x_n$  solve the D.E. in steps of  $h$  using inbuilt function *ode*.
5. Print values of  $x_0$  to  $x_n$  and  $y_0$  to  $y_n$
6. Plot graph in  $(x, y)$
7. Stop

#### Flow Chart for Euler's Method



---

### Scilab Code for Solving Initial Value Problem using Euler's Method

```
// Solution of Initial value problem  $\frac{dy}{dx} = 2 - 2y - e^{-4x}$ ,  $y(0) = 1$ ,  $h = .1$ ,  $x_n = 1$ 
// If  $f$  is a Scilab function, its calling sequence must be  $ydot = f(x, y)$ 
//  $ydot$  is used for first order derivative  $\frac{dy}{dx}$ 
// ode is scilab inbuilt function to evaluate the D.E. in the given format
clc
function ydot = euler(x, y)
ydot= 2 - 2 * y - %e^(-4 * x)
endfunction
x0=input("Enter initial value x0: ")
y0=input("Enter initial value y0: ")
h=input("Enter step size h: ")
xn=input("Enter final value xn: ")
x = x0 : h : xn;
y=ode(y0, x0, x, euler)
disp(x, " x value:")
disp(y, " y value:")
plot (x, y)
```

---

## 11.2 Viva Questions

1. What is an initial value problem?
2. Which kind of differential equations can be solved using Euler's method?
3. Can we solve second order differential equation using Euler's method?
4. Explain geometrical significance of Euler's method.

## Chapter 12

# Numerical Solution of Ordinary Differential Equations Using Runge-Kutta Method

The problem of finding a function  $y$  of  $x$  when we know its derivative and its value  $y_0$  at a particular point  $x_0$  is called an initial value problem.

---

## 12.1 Runge-Kutta method of 4<sup>th</sup> Order

Euler's method is simple but not an appropriate method for integrating an ODE as the derivative at the starting point of each interval is extrapolated to find the next function value. The method has first-order accuracy.

In Fourth-order Runge-Kutta method, at each step the derivative is evaluated four times; once at the initial point; twice at trial midpoints; and once at a trial endpoint. The final function value is calculated from these derivatives as given below.

$$\begin{aligned}k_1 &= hf(x_n, y_n) \\k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\k_4 &= hf(x_n + h, y_n + k_3) \\y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}.\end{aligned}$$

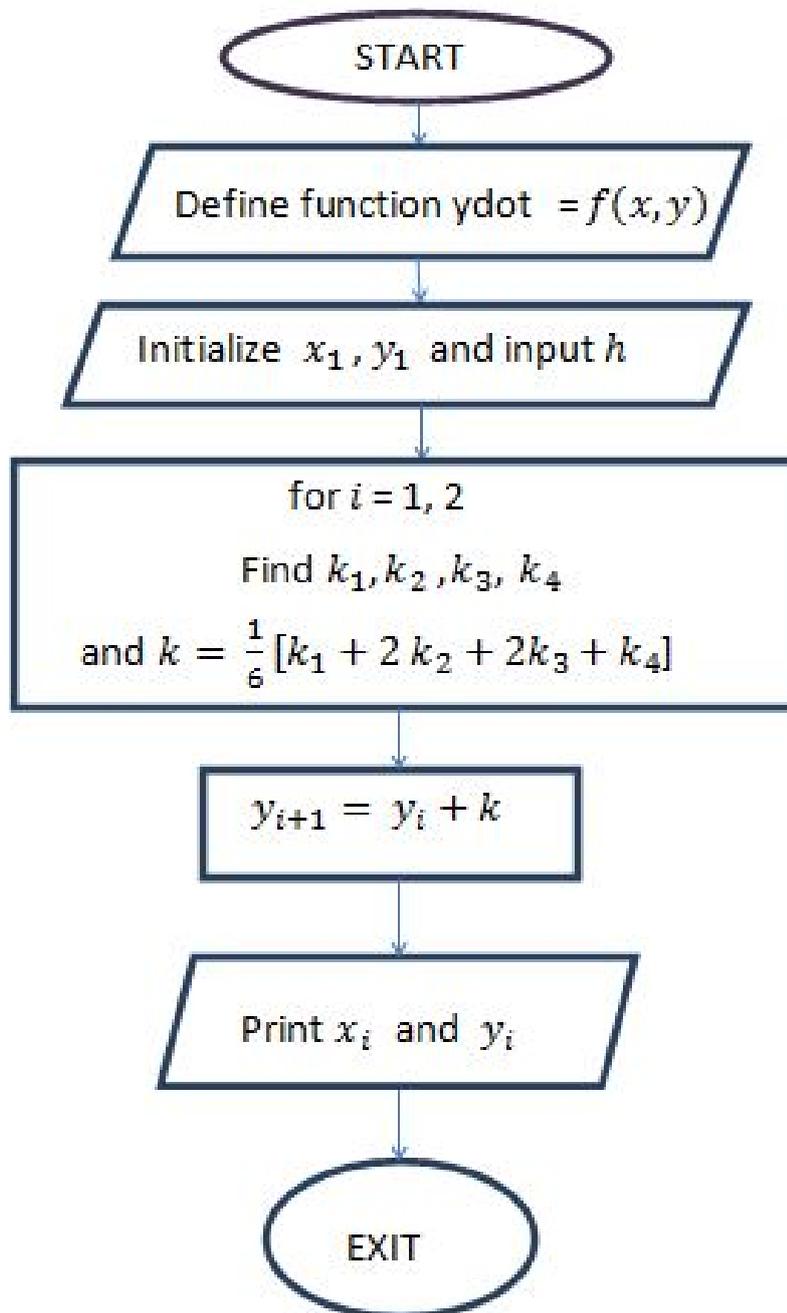
**Algorithm for solving initial value problem using RUNGE KUTTA METHOD :**

1. Start
2. Define and Declare function ydot representing  $\frac{dy}{dx} = f(x, y)$
3. Initialize values of  $x$  and  $y$  and input  $h$ , the step size.
4. Calculate 2 sets ( $i= 1,2$ ) of values for  $k_1, k_2, k_3$  and  $k_4$  and subsequently value of  $k = \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$ .
5. Finally  $y_{i+1} = y_i + k$
6. Print values of  $x_i$  and  $y_i$ .

---

7. Stop

Flow Chart for Runge Kutta Method



---

```

// Scilab Code for Runge Kutta Method
clc
function ydot = f(x, y)
ydot = x + y^2
endfunction

x1 = 0;
y1 = 1;

h=input("Enter step size h: ")
x(1) = x1;
y(1) = y1;
for i = 1 : 2
k1 = h * f(x(i), y(i));
k2 = h * f(x(i) + 0.5 * h, y(i) + 0.5 * k1);
k3 = h * f((x(i) + 0.5 * h), (y(i) + 0.5 * k2));
k4 = h * f((x(i) + h), (y(i) + k3));
k = (1/6) * (k1 + 2 * k2 + 2 * k3 + k4);
y(i + 1) = y(i) + k;
printf('\n The value of y at x=%f is %f ', i * h, y(i + 1))
x(i + 1) = x(1) + i * h;
end

```

---

## 12.2 Viva Questions

1. Which gives more accurate results Euler's method or Runge Kutta Method of fourth order?
2. Which kind of differential equations can be solved using Runge Kutta method?
3. Can we solve second order differentisl equation using Runge Kutta method?
4. Is there any connection between Euler's method and Runge Kutta Method?  
Explain