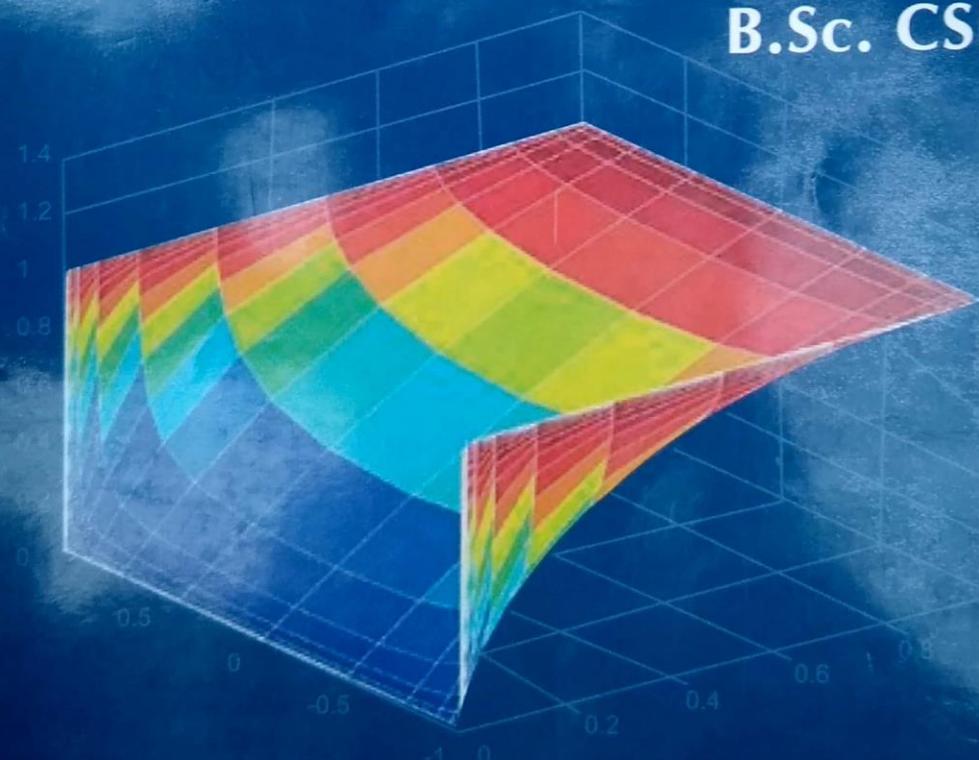




NUMERICAL METHOD

with
Practical Approach

B.Sc. CSIT



Arjun Singh Saud

Numerical Method (CSC 204)

Course Title: Numerical Method ----- Full Marks: 60 + 20 + 20

Course No.: CSC207 ----- Pass Marks: 24 + 8 + 8

Nature of the Course: Theory + Lab Credit Hrs:

Semester : III

Course Description:

This course contains the concepts of numerical method techniques for solving linear and nonlinear equations, interpolation and regression, differentiation and integration, and partial differential equations.

Course Objectives:

The main objective of the course is to provide the knowledge of numerical method techniques for mathematical modeling.

Course Contents:

Unit 1: Solution of Nonlinear Equations ----- (8 Hrs.)

- 1.1 Errors in Numerical Calculations, Sources of Errors, Propagation of Errors, Review of Taylor's Theorem
- 1.2 Solving Non-linear Equations by Trial and Error method, Half-Interval method and Convergence, Newton's method and Convergence, Secant method and Convergence, Fixed point iteration and its convergence, Newton's method for calculating multiple roots, Horner's method

Unit 2: Interpolation and Regression ----- (8 Hrs.)

- 2.1 Interpolation vs Extrapolation, Lagrange's Interpolation, Newton's Interpolation using divided differences, forward differences and backward differences, Cubic spline



interpolation

- 2.2 Introduction of Regression, Regression vs Interpolation, Least squares method, Linear Regression, Non-linear Regression by fitting Exponential and Polynomial

Unit 3: Numerical Differentiation and Integration ————— (8 Hrs.)

- 3.1 Differentiating Continuous Functions (Two-Point and Three-Point Formula), Differentiating Tabulated Functions by using Newton's Differences, Maxima and minima of Tabulated Functions
- 3.2 Newton-Cote's Quadrature Formulas, Trapezoidal rule, Multi-Segment Trapezoidal rule, Simpson's 1/3 rule, Multi-Segment Simpson's 1/3 rule, Simpson's 3/8 rule, Multi-Segment Simpson's 3/8 rule, Gaussian integration algorithm, Romberg integration

Unit 4: Solving System of Linear Equations ————— (8 Hrs.)

- 4.1 Review of the existence of solutions and properties of matrices, Gaussian elimination method, pivoting, Gauss-Jordan method, Inverse of matrix using Gauss-Jordan method
- 4.2 Matrix factorization and Solving System of Linear Equations by using Dolittle and Cholesky's algorithm
- 4.3 Iterative Solutions of System of Linear Equations, Jacobi Iteration Method, Gauss-Seidal Method
- 4.4 Eigen values and eigen vectors problems, Solving eigen value problems using power method.

Unit 5: Solution of Ordinary Differential Equations ————— (8 Hrs.)

- 5.1 Review of differential equations, Initial value problem, Taylor series method, Picard's method, Euler's method and its accuracy, Heun's method, Runge-Kutta methods
- 5.2 Solving System of ordinary differential equations, Solution of the higher order equations, Boundary value problems, Shooting method and its algorithm

Unit 6: Solution of Partial Differential Equations ————— (5 Hrs.)

- 6.1 Review of partial differential equations, Classification of partial differential equation, Deriving difference equations, Laplacian equation and Poisson's equation, engineering examples

Laboratory Works:

The laboratory exercise should consist program development and testing of non-linear equations, system of linear equations, interpolation, numerical integration and differentiation, linear algebraic equations, ordinary and partial differential equations. Numerical solutions using C or Matlab.

Text Books:

1. W. Cheney and D. Kincaid, "Numerical Mathematics and Computing", 7th Edition, Brooks/Cole Publishing Co, 2012
2. C.F. Gerald and P.O. Wheatley, "Applied Numerical Analysis", 9th Edition, Addison Wesley Publishing Company, New York, 2011

Reference Books:

1. E. Balagurusamy, "Numerical Methods", Tata McGraw-Hill Publishing Company Ltd., New Delhi, 1999
2. W.H. Press, B.P. Flannery et al., "Numerical Recipes: Art of Scientific Computing", 3rd Edition, Cambridge Press, 2007.
3. J. M. Mathews and K. Fink, "Numerical Methods using MATLAB", 4th Edition, Prentice Hall Publication, 2004

Table of Contents

Chapter 1 ERRORS AND MATHEMATICAL REVIEW

1.1 Introduction -----	2
1.2 Types Errors in Numerical Computation -----	2
1.2.1 True error -----	2
1.2.2 Relative Error -----	3
1.2.3 Approximate Error -----	4
1.2.4 Relative Approximate Error-----	5
1.3 Sources of Error-----	7
1.3.1 Round off Error-----	7
1.3.2 Truncation Error -----	7
1.4 Floating Point Representation -----	8
1.5 Propagation of Errors -----	13
1.6 Review of Taylor Theorem-----	14
1.6.1 Error in Taylor Series -----	16
1.7 Mean Value Theorem -----	18
1.8 Algorithm Analysis-----	19
1.8.1 Big Oh (O) notation-----	19
1.8.2 Big Omega (Ω) notation-----	19
1.8.3 Big Theta (Θ) notation-----	19
Exercise-----	20

Chapter 2 SOLVING NONLINEAR EQUATIONS

2.1 Solving Nonlinear Equations-----	22
2.2 Direct Analytical Method -----	22
2.3 Graphical Method-----	23
2.4 Trial and Error Method-----	23
2.5 Iterative Methods -----	24
2.5.1 Bisection Method (Half Interval Method)-----	24
2.5.2 Newton Raphson Method -----	32
2.5.3 The secant method-----	39
2.5.4 Fixed-point method -----	45
2.6 Synthetic Division-----	51
2.7 Remainder Theorem-----	53
2.8 Finding Multiple Roots by using Newton-Raphson Method -----	54
2.9 Horner's Method for Polynomial Evaluation -----	57
Exercise-----	60

Chapter 3 INTERPOLATION

3.1 What is Interpolation? -----	64
3.2 Lagrange Interpolation -----	64
3.3 Newton's Divided Difference Interpolation-----	69
3.3.1 Divided Difference Table -----	74
3.4 Newton's Forward and Backward Difference Interpolation -----	76
3.4.1 Newton Forward-Difference interpolation-----	77

3.4.2 Newton Backward-Difference interpolation -----	83
3.5 Spline Interpolation-----	88
3.5.1 Natural Cubic Splines-----	89
Exercise-----	96

Chapter **4 REGRESSION ANALYSIS**

4.1 Introduction -----	100
4.1.1 Concept of Regression -----	100
4.1.2 Interpolation vs. Regression-----	100
4.1.3 Parameter Estimation Methods-----	101
4.2 Least Squares Method-----	102
4.3 Linear Regression -----	102
4.4 Nonlinear Regression -----	106
4.4.1 Fitting Exponential Model -----	106
4.4.2 Fitting Exponential Model by Linearization -----	109
4.4.3 Fitting Polynomial Models -----	114
Exercise-----	120

Chapter **5 SOLUTION OF SYSTEM OF LINEAR ALGEBRAIC EQUATIONS**

5.1 System of Equations -----	124
5.1.1 Existence of Solution and Ill Conditioning -----	124
5.2 Solving System of Linear Equations-----	127
5.3 Direct Methods for Solving System of Linear Equations-----	127

5.3.1 Naive Gauss Elimination Method	127
5.3.2 Gauss Elimination with Partial and Complete Pivoting	135
5.3.3 Gauss-Jordan Method	141
5.4 Matrix Inversion	147
5.5 Matrix Factorization	151
5.5.1 Doolittle LU Decomposition	152
5.5.2 Cholesky Method	162
5.6 Iterative Methods for solving Systems of Linear Equations	170
5.6.1 Jacobi Iteration Method	171
5.6.2 Gauss Seidel Method	176
5.7 Ill-conditioning	182
5.8 Eigenvectors and Eigenvalues of a Matrix	182
5.8.1 Power Method	182
Exercise	191

Chapter **6** NUMERICAL DIFFERENTIATION

6.1 Introduction	194
6.2 Differentiating Continuous Functions	194
6.2.1 Two Point Forward Difference Formula	194
6.2.2 Two Point Backward Difference Formula	194
6.2.3 Three Point Formula	197
6.3 Differentiating Discrete (Tabulated) Functions	200
6.3.1 Derivatives Using Newton's Divided Difference Formula	203
6.3.2 Derivatives Using Newton's Forward Difference Formula	208

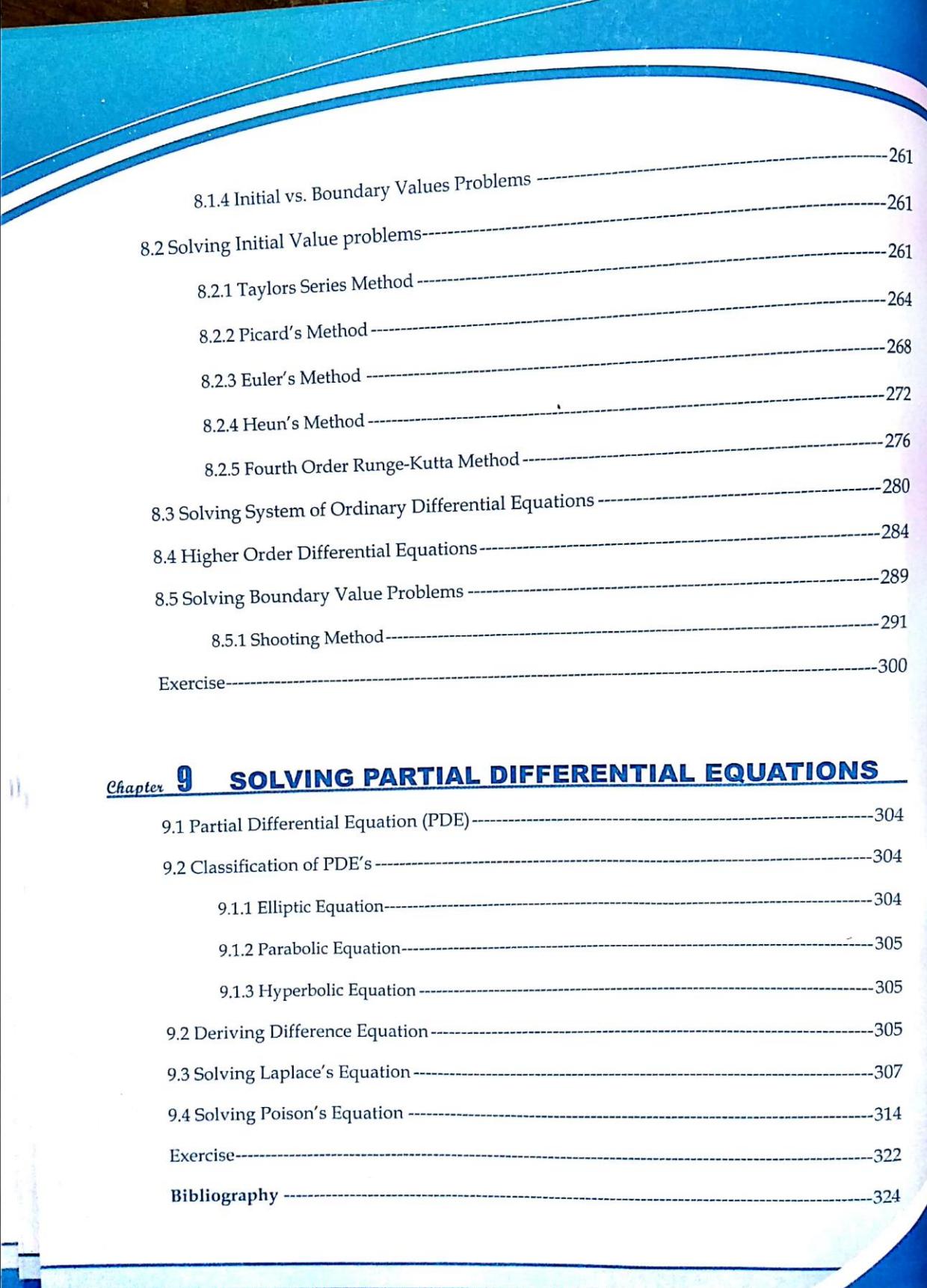
6.3.3 Derivatives Using Newton's Backward Difference Formula-----	213
6.4 Maxima and Minima of tabulated functions -----	218
Exercise-----	222

Chapter 7 NUMERICAL INTEGRATION

7.1 Introduction -----	226
7.2 Newton-Cotes Integration Formulae-----	227
7.3 A General Quadrature Formula for Equally Spaced Arguments -----	227
7.3.1 Trapezoidal Rule-----	228
7.3.2 Composite Trapezoidal Rule-----	230
7.3.3 Simpson's 1/3 Rule-----	233
7.3.4 Composite Simpson's 1/3 Rule-----	236
7.3.5 Simpson's 3/8 Rule-----	241
7.3.6 Composite Simpson's 3/8 Rule-----	244
7.4 Gaussian Integration -----	247
7.5 Romberg Integration -----	251
Exercise-----	257

Chapter 8 SOLVING ORDINARY DIFFERENTIAL EQUATIONS

8.1 Introduction -----	260
8.1.1 Ordinary vs. Partial differential equations -----	260
8.1.2 Order and Degree of Differential Equations -----	260
8.1.3 General vs. Particular Solutions of Differential Equations -----	261



8.1.4 Initial vs. Boundary Values Problems	261
8.2 Solving Initial Value problems	261
8.2.1 Taylors Series Method	261
8.2.2 Picard's Method	264
8.2.3 Euler's Method	268
8.2.4 Heun's Method	272
8.2.5 Fourth Order Runge-Kutta Method	276
8.3 Solving System of Ordinary Differential Equations	280
8.4 Higher Order Differential Equations	284
8.5 Solving Boundary Value Problems	289
8.5.1 Shooting Method	291
Exercise	300

Chapter 9 SOLVING PARTIAL DIFFERENTIAL EQUATIONS

9.1 Partial Differential Equation (PDE)	304
9.2 Classification of PDE's	304
9.1.1 Elliptic Equation	304
9.1.2 Parabolic Equation	305
9.1.3 Hyperbolic Equation	305
9.2 Deriving Difference Equation	305
9.3 Solving Laplace's Equation	307
9.4 Solving Poisson's Equation	314
Exercise	322
Bibliography	324

Chapter 1

ERRORS AND MATHEMATICAL REVIEW

Chapter Content

- Introduction
- Errors in Numerical Computation
- Sources of Error
- Floating Point Representation
- Propagation of Errors
- Review of Taylors Theorem
- Mean Value Theorem
- Algorithm Analysis

1.1 INTRODUCTION

Mathematical models are an integral part in solving scientific and engineering problems. Many times, these mathematical models are derived from engineering and science principles, while at other times the models may be obtained from experimental data. Mathematical models generally result in need of using mathematical procedures that include but are not limited to

- ▶ Nonlinear equations,
- ▶ Simultaneous linear equations,
- ▶ Curve fitting by interpolation or regression,
- ▶ Differentiation,
- ▶ Integration, and
- ▶ Differential equations.

These mathematical procedures may be suitable to be solved exactly as you must have experienced in the series of calculus courses you have taken, but in most cases, the procedures need to be solved approximately using numerical methods.

1.2 TYPES ERRORS IN NUMERICAL COMPUTATION

In any numerical analysis, errors will arise during the calculations. To be able to deal with the issue of errors, we need to identify where the error is coming from, followed by quantifying the error, and lastly minimize the error as per our needs.

1.2.1 True error

True error denoted by E_t , is the difference between the true value (also called the exact value) and the approximate value.

$$\text{True Error} = \text{True value} - \text{Approximate value}$$

Example

The derivative of a function $f(x)$ at a particular value of x can be approximately calculated by

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

For $f(x) = 7e^{0.5x}$ and $h = 0.3$, find the approximate value of $f'(2)$, the true value of $f'(2)$ and the true error.

Solution

Approximate value of derivative is calculated by using formula

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

Now,

For $x = 2$ and $h = 0.3$,

$$f'(2) \approx \frac{f(2+0.3) - f(2)}{0.3}$$

$$\begin{aligned}
 &= \frac{f(2.3) - f(2)}{0.3} \\
 &= \frac{7e^{0.5(2.3)} - 7e^{0.5(2)}}{0.3} \\
 &= \frac{22.107 - 19.028}{0.3} \\
 &= 10.265
 \end{aligned}$$

The exact value of $f'(2)$ can be calculated by using our knowledge of differential calculus.

$$\begin{aligned}
 f(x) &= 7e^{0.5x} \\
 f'(x) &= 7 \times 0.5 \times e^{0.5x} = 3.5e^{0.5x}
 \end{aligned}$$

So the true value of $f'(2)$ is

$$f'(2) = 3.5e^{0.5(2)} = 9.5140$$

Now, true error is calculated as

$$\begin{aligned}
 E_t &= \text{True value} - \text{Approximate value} \\
 &= 9.5140 - 10.265 = -0.75061
 \end{aligned}$$

The magnitude of true error does not show how bad the error is. If the function given in the above example were $f(x) = 7 \times 10^{-6} e^{0.5x}$, the true error in calculating $f'(2)$ with $h = 0.3$, would be $E_t = -0.75061 \times 10^{-6}$. This value of true error is smaller than true error with function $f(x) = 7e^{0.5x}$. Thus even if the two problems are similar and they use the same value of the function arguments, magnitude of error can be different. This brings us to the definition of relative true error.

1.2.2 Relative Error

Relative true error is denoted by E_r and is defined as the ratio between the true error and the true value.

$$\text{Relative True Error} = \frac{\text{True Error}}{\text{True Value}}$$

Example

The derivative of a function $f(x)$ at a particular value of x can be approximately calculated by

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

For $f(x) = 7e^{0.5x}$ and $h = 0.3$, find the relative true error at $x = 2$.

Solution

From above example,

$$\begin{aligned}
 E_t &= \text{True value} - \text{Approximate value} \\
 &= 9.5140 - 10.265 \\
 &= -0.75061
 \end{aligned}$$

Now, Relative true error is calculated as

$$\begin{aligned}
 E_r &= \frac{\text{True Error}}{\text{True Value}} \\
 &= \frac{-0.75061}{9.5140} \\
 &= -0.078895
 \end{aligned}$$

Relative true errors are also presented as percentages. For this example,

$$\begin{aligned}
 E_r &= -0.078895 \times 100\% \\
 &= -7.58895\%
 \end{aligned}$$

Absolute relative true errors may also need to be calculated. In such cases,

$$\begin{aligned}
 |E_r| &= -0.0758881 \\
 &= 0.0758895 \\
 &= 7.58895\%
 \end{aligned}$$

1.2.3 Approximate Error

True errors can be calculated only if true values are known. But mostly we do not have the luxury of knowing true values. In such cases we want to find the approximate values. When we are solving a problem numerically, we may only have access to approximate values. Approximate error is denoted by E_a and is defined as the difference between the present approximation and previous approximation.

Approximate Error = Present Approximation – Previous Approximation

Example

The derivative of a function $f(x)$ at a particular value of x can be approximately calculated by

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

For $f(x) = 7e^{0.5x}$ and at $x = 2$, find the following

- $f'(2)$ using $h = 0.3$
- $f'(2)$ using $h = 0.15$
- approximate error for the value of $f'(2)$

Solution

The approximate expression for the derivative of a function is

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

For $x = 2$ and $h = 0.3$,

$$\begin{aligned}f'(2) &\approx \frac{f(2+0.3) - f(2)}{0.3} \\&= \frac{f(2.3) - f(2)}{0.3} \\&= \frac{7e^{0.5(2.3)} - 7e^{0.5(2)}}{0.3} \\&= \frac{22.107 - 19.028}{0.3} \\&= 10.265\end{aligned}$$

Repeat the procedure with $h = 0.15$,

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

For $x = 2$ and $h = 0.15$,

$$\begin{aligned}f'(2) &\approx \frac{f(2+0.15) - f(2)}{0.15} \\&= \frac{f(2.15) - f(2)}{0.15} \\&= \frac{7e^{0.5(2.15)} - 7e^{0.5(2)}}{0.15} \\&= \frac{20.50 - 19.028}{0.15} \\&= 9.8799\end{aligned}$$

So the approximate error, E_a is

$$\begin{aligned}E_a &= \text{Present Approximation} - \text{Previous Approximation} \\&= 9.8799 - 10.265 \\&= -0.38474\end{aligned}$$

The magnitude of approximate error does not show how bad the error is. For $f(x) = 7 \times 10^{-6} e^{0.5x}$, the approximate error in calculating $f'(2)$ with $h = 0.15$ would be $E_a = -0.38474 \times 10^{-6}$. This value of approximate error is smaller. Thus even when the two problems are similar in that they use the same value of the function argument, magnitude of error is different. This brings us to the definition of relative approximate error.

1.2.4 Relative Approximate Error

Relative approximate error is denoted by E_{ra} and is defined as the ratio between the approximate error and the present approximation.

$$\text{Relative Approximate Error} = \frac{\text{Approximate Error}}{\text{Present Approximation}}$$

Example

The derivative of a function $f(x)$ at a particular value of x can be approximately calculated by

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

For $f(x) = 7e^{0.5x}$, find the relative approximate error in calculating $f'(2)$ using values from $h = 0.3$ and $h = 0.15$.

Solution

From above example, the approximate value of $f'(2) = 10.263$ using $h = 0.3$ and $f'(2) = 9.8800$ using $h = 0.15$.

$$\begin{aligned} E_a &= \text{Present Approximation} - \text{Previous Approximation} \\ &= 9.8799 - 10.265 \\ &= -0.38474 \end{aligned}$$

The relative approximate error is calculated as

$$\begin{aligned} E_{ra} &= \frac{\text{Approximate Error}}{\text{Present Approximation}} \\ &= \frac{-0.38474}{9.8799} \\ &= -0.038942 \end{aligned}$$

Relative approximate errors can also be presented as percentages. For this example,

$$\begin{aligned} E_{ra} &= -0.038942 \times 100\% \\ &= -3.8942\% \end{aligned}$$

Absolute relative approximate errors may also need to be calculated. In this example

$$\begin{aligned} |E_{ra}| &= |-0.038942| \\ &= 0.038942 \text{ or } 3.8942\% \end{aligned}$$



Did you know? While solving a mathematical model using numerical methods, how can we use relative approximate errors to minimize the error?

In a numerical method that uses iterative methods, a user can calculate relative approximate error E_{ra} at the end of each of the iteration. The user may pre-specify a minimum acceptable tolerance called the pre-specified tolerance, say ϵ . If the absolute relative approximate error E_{ra} is less than or equal to the pre-specified tolerance ϵ , that is, $|E_{ra}| \leq \epsilon$ then the acceptable error has been reached and no more iterations would be required.

Alternatively, one may pre-specify how many significant digits they would like to be correct in their answer. In that case, if one wants at least m significant digits to be correct in the answer, then you would need to have the absolute relative approximate error, $|E_{ra}| \leq 0.5 \times 10^{2-m} \%$.

1.3 SOURCES OF ERROR

Error in solving an engineering or science problem can arise due to several factors. First, the error may be in the modeling technique. A mathematical model may be based on using assumptions that are not acceptable. For example, one may assume that the drag force on a car is proportional to the velocity of the car, but actually it is proportional to the square of the velocity of the car. This can create huge errors in determining the performance of the car, no matter how accurate the numerical methods you may use are. Second, errors may arise from mistakes in programs themselves or in the measurement of physical quantities. But, in applications of numerical methods, the two errors we need to focus on are

1. Round off error
2. Truncation error.

1.3.1 Round off Error

A computer can only represent a number approximately. For example, a number like $1/3$ may be represented as 0.333333 on a PC. Then the round off error in this case is $1/3 - 0.333333 = 0.00000033$. There are also other numbers that cannot be represented exactly. For example, π and $\sqrt{2}$ are numbers that need to be approximated in computer calculations.

1.3.2 Truncation Error

Truncation error is defined as the error caused by truncating a mathematical procedure. For example, the Maclaurin series for e^x is given as

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

This series has an infinite number of terms but when using this series to calculate e^x , only a finite number of terms can be used. For example, if one uses three terms to calculate e^x , then

$$e^x \approx 1 + x + \frac{x^2}{2!}$$

the truncation error for such an approximation is

$$\begin{aligned} \text{Truncation error} &= e^x - \left(1 + x + \frac{x^2}{2!} \right), \\ &= \frac{x^3}{3!} + \frac{x^4}{4!} + \dots \end{aligned}$$

8 Numerical Methods with Practical Approach

But, to control truncation errors, we can use the concept of relative approximate error to see how many terms need to be considered. Assume that one is calculating $e^{1.2}$ using the Maclaurin series, then

$$e^{1.2} = 1 + 1.2 + \frac{1.2^2}{2!} + \frac{1.2^3}{3!} + \dots$$

Let us assume one wants the absolute relative approximate error to be less than 1%. In Table 1, we show the value of $e^{1.2}$, approximate error and absolute relative approximate error as a function of the number of terms.

n	$e^{1.2}$	E_a	$ E_{ra} \%$
1	1	-	-
2	2.2	1.2	54.546
3	2.92	0.72	24.658
4	3.208	0.288	8.9776
5	3.2944	0.0864	2.6226
6	3.3151	0.020736	0.62550

Using 6 terms of the series yields a $|E_{ra}| < 1\%$.

1.4 FLOATING POINT REPRESENTATION

To keep the relative error of similar order for all numbers, we may use a floating-point representation of the number. In floating-point representation, a number 256.78 is written as $+2.5678 \times 10^2$, 0.003678 is written as $+3.678 \times 10^{-3}$, and -256.789 is written as -2.56789×10^2 .

The general representation of a number in base-10 format is: $sign \times mantissa \times 10^{exponent}$ or for a number y ,

$$y = \sigma \times m \times 10^e$$

Where

σ = sign of the number, +1 or -1

m = mantissa, $1 \leq m < 10$

e = integer exponent

If we use the five spaces, then let us use four for the mantissa and the last one for the exponent. So the smallest number that can be represented is 1 but the largest number would be 9.999×10^9 . By using the floating-point representation, we lose in accuracy but we gain in the range of numbers that can be represented. For our example, the maximum number represented changed from 999.99 to 9.999×10^9 . In floating point representation, 256.78 can be represented as 2.568×10^2 and in the five spaces as

2	5	6	8	2
---	---	---	---	---

Another example, the number 576329.78 would be represented as 5.763×10^5 and in five spaces as

5	7	6	3	5
---	---	---	---	---

So, in representing 256.78 , the round off error created is $256.78 - 256.8 = -0.02$, and the relative error is

$$E_r = \frac{-0.02}{256.78} \times 100 = -0.0077888\%$$

Similarly, in representing 576329.78 , the round off error created is $576329.78 - 5.763 \times 10^5 = 29.78$, and the relative error is

$$E_r = \frac{29.78}{576329.78} \times 100 = 0.0051672\%$$

Thus from above it is clear that the errors are large for large numbers, but the relative errors are of the same order for both large and small numbers.

This floating-point format can also be used to represent binary numbers. A number y would be written as

$$y = \sigma \times m \times 2^e$$

Where

σ = sign of number (negative or positive – use 0 for positive and 1 for negative),

m = mantissa, $(1)_2 \leq m < (10)_2$, that is $(1)_{10} \leq m < (2)_{10}$, and

e = integer exponent.

Example

Represent $(54.75)_{10}$ in floating point binary format. Assuming that the number is written to a hypothetical word that is 9 bits long where the first bit is used for the sign of the number, the second bit for the sign of the exponent, the next four bits for the mantissa, and the next three bits for the exponent,

Solution

Binary representation is:

$$(54.75)_{10} = (110110.11)_2 = (1.1011011)_2 \times 2^5$$

The exponent 5 is equivalent in binary format as

$$(5)_{10} = (101)_2$$

Hence

$$(54.75)_{10} = (1.1011011)_2 \times 2^{101}$$

The sign of the number is positive, so the bit for the sign of the number will have zero in it. The sign of the exponent is positive. So the bit for the sign of the exponent will also have zero in it.

The mantissa $m = 1011$ (There are only 4 places for the mantissa, and the leading 1 is not stored as it is always expected to be there), and the exponent $e = 101$. We have the representation as

0	0	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---

Example

What number does the below given floating point format represent in base-10 format.

0	1	1	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---

Assume a hypothetical 9-bit word, where the first bit is used for the sign of the number, second bit for the sign of the exponent, next four bits for the mantissa and next three for the exponent.

Solution

The first bit is 0, so the number is positive.

The second bit is 1, so the exponent is negative.

The next four bits, 1011, are the magnitude of the mantissa, so

$$m = (1.1011)_2 = (1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4})_{10} = 1.6875$$

The last three bits, 110, are the magnitude of the exponent, so

$$e = (110)_2 = (1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)_{10} = 6$$

The number in binary format then is

$$(1.1011)_2 \times 2^{-(110)_2}$$

The number in base-10 format is

$$= 1.6875 \times 2^{-6}$$

$$= 0.026367$$

Example

A machine stores floating-point numbers in a hypothetical 10-bit binary word. It employs the first bit for the sign of the number, the second one for the sign of the exponent, the next four for the exponent, and the last four for the magnitude of the mantissa.

a. Find how 0.02832 will be represented in the floating-point 10-bit word.

b. What is the decimal equivalent of the 10-bit word representation of part (a)?

Solution

For the number, we have the integer part as 0 and the fractional part as 0.02832

Let us first find the binary equivalent of the integer part

$$\text{Integer part } (0)_{10} = (0)_2$$

Now we find the binary equivalent of the fractional part

Fractional part: .02832 × 2

$$\underline{0.05664 \times 2}$$

$$\underline{0.11328 \times 2}$$

$$\underline{0.22656 \times 2}$$

$$\underline{0.45312 \times 2}$$

$$\underline{0.90624 \times 2}$$

$$\underline{1.81248 \times 2}$$

$$\underline{1.62496 \times 2}$$

$$\underline{1.24992 \times 2}$$

$$\underline{0.49984 \times 2}$$

$$\underline{0.99968 \times 2}$$

$$\underline{1.99936}$$

Hence

$$\begin{aligned}(0.02832)_{10} &\cong (0.00000111001)_2 \\&= (1.11001)_2 \times 2^{-6} \\&\cong (1.1100)_2 \times 2^{-6}\end{aligned}$$

The binary equivalent of exponent is found as follows

Since

$$(6)_{10} = (110)_2$$

So

$$(0.02832)_{10} = (1.1100)_2 \times 2^{-(110)_2}$$

$$= (1.1100)_2 \times 2^{-(0110)_2}$$

The ten-bit representation bit by bit is

0	1	0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---

Converting the above floating point representation from part (a) to base 10 by following Example 2 gives

$$\begin{aligned}&(1.1100)_2 \times 2^{-(0110)_2} \\&= (1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4}) \times 2^{-(0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)} \\&= (1.75)_{10} \times 2^{-(6)_{10}} \\&= 0.02734375\end{aligned}$$



Did you know? How are numbers actually represented in floating point in a real computer?

In an actual typical computer, a real number is stored as per the IEEE-754 (Institute of Electrical and Electronics Engineers) floating-point arithmetic format. To keep the discussion short and simple, let us point out the salient features of the single precision format.

A single precision number uses 32 bits. A number y is represented as

$$y = \sigma \times (1.a_1 a_2 \cdots a_{23}) \cdot 2^e$$

where

σ = sign of the number (positive or negative)

a_i = entries of the mantissa, can be only 0 or 1, $i = 1, \dots, 23$

e = the exponent

The first bit represents the sign of the number (0 for positive number and 1 for a negative number).

The next eight bits represent the exponent. Note that there is no separate bit for the sign of the exponent. The sign of the exponent is taken care of by normalizing by adding 127 to the actual exponent. For example in the previous example, the exponent was 6. It would be stored as the binary equivalent of $127 + 6 = 133$. Why is 127 and not some other number added to the actual exponent? Because in eight bits the largest integer that can be represented is $(11111111)_2 = 255$, and halfway of 255 is 127. This allows negative and positive exponents to be represented equally. The normalized (also called biased) exponent has the range from 0 to 255, and hence the exponent e has the range of $-127 \leq e \leq 128$.

If instead of using the biased exponent, let us suppose we still used eight bits for the exponent but used one bit for the sign of the exponent and seven bits for the exponent magnitude. In seven bits, the largest integer that can be represented is $(1111111)_2 = 127$ in which case the exponent e range would have been smaller, that is, $-127 \leq e \leq 127$. By biasing the exponent, the unnecessary representation of a negative zero and positive zero exponent (which are the same) is also avoided.

Actually, the biased exponent range used in the IEEE-754 format is not 0 to 255, but 1 to 254. Hence, exponent e has the range of $-126 \leq e \leq 127$. So what are $e = -127$ and $e = 128$ used for? If $e = 128$ and all the mantissa entries are zeros, the number is $\pm\infty$ (the sign of infinity is governed by the sign bit), if $e = 128$ and the mantissa entries are not zero, the number being represented is Not a Number (NaN). Because of the leading 1 in the floating point representation, the number zero cannot be represented exactly. That is why the number zero (0) is represented by $e = -127$ and all the mantissa entries being zero.

The next twenty-three bits are used for the mantissa. The largest number by magnitude that is represented by this format is

$$(1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + \dots + 1 \times 2^{-22} + 1 \times 2^{-23}) \times 2^{127} = 3.40 \times 10^{38}$$

The smallest number by magnitude that is represented, other than zero, is

$$(1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + \dots + 0 \times 2^{-22} + 0 \times 2^{-23}) \times 2^{-126} = 1.18 \times 10^{-38}$$



Note: How are numbers represented in floating point in double precision in a computer?

In double precision IEEE-754 format, a real number is stored in 64 bits.

- ✓ The first bit is used for the sign,
- ✓ the next 11 bits are used for the exponent, and
- ✓ the rest of the bits, that is 52, are used for mantissa.

1.5 PROPAGATION OF ERRORS

If a calculation is made with numbers that are not exact, then the calculation itself will have an error. Propagation of error (or propagation of uncertainty) is the effect of variables uncertainties (or errors) on the uncertainty of a function based on them. When the variables are the values of experimental measurements they have uncertainties due to measurement limitations (e.g., instrument precision) which propagate to the combination of variables in the function. How do the errors in each individual number propagate through the calculations. Let's look at the concept via some examples.

Example

Find the bounds for the propagation error in adding two numbers. For example if one is calculating $X + Y$ where, $X = 1.5 \pm 0.05$ & $Y = 3.4 \pm 0.04$

Solution

By looking at the numbers, the maximum possible value of X and Y are $X = 1.55$ and $Y = 3.44$

Hence,

$$X + Y = 1.55 + 3.44 = 4.99 \text{ is the maximum value of } X + Y.$$

The minimum possible value of X and Y are $X = 1.45$ and $Y = 3.36$.

Hence,

$$X + Y = 1.45 + 3.36 = 4.81 \text{ is the minimum value of } X + Y.$$

Hence

$$4.81 \leq X + Y \leq 4.99.$$

One can find similar intervals of the bound for the other arithmetic operations of $X - Y$, $X * Y$, and X / Y . What if the evaluations we are making are function evaluations instead? How do we find the value of the propagation error in such cases? If f is a function of several variables $X_1, X_2, X_3, \dots, X_{n-1}, X_n$, then the maximum possible value of the error in f is

$$\Delta f \approx \left| \frac{\partial f}{\partial X_1} \Delta X_1 \right| + \left| \frac{\partial f}{\partial X_2} \Delta X_2 \right| + \dots + \left| \frac{\partial f}{\partial X_{n-1}} \Delta X_{n-1} \right| + \left| \frac{\partial f}{\partial X_n} \Delta X_n \right|$$

Example

Subtraction of numbers that are nearly equal can create unwanted inaccuracies. Using the formula for error propagation, show that this is true.

Solution

Let

$$z = x - y$$

Then

$$|\Delta z| = \left| \frac{\partial z}{\partial x} \Delta x \right| + \left| \frac{\partial z}{\partial y} \Delta y \right| = |(1)\Delta x| + |(-1)\Delta y| = |\Delta x| + |\Delta y|$$

So the absolute relative change is

$$\left| \frac{\Delta z}{z} \right| = \frac{|\Delta x| + |\Delta y|}{|x - y|}$$

As x and y become close to each other, the denominator becomes small and hence create large relative errors.

For example if

$$x = 2 \pm 0.001$$

$$y = 2.003 \pm 0.001$$

$$\left| \frac{\Delta z}{z} \right| = \frac{|0.001| + |0.001|}{|2 - 2.003|} = 0.6667 = 66.67\%$$

1.6 REVIEW OF TAYLOR THEOREM

Taylor's theorem is given by

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \dots \quad (1)$$

Provided all derivatives of $f(x)$ exist and are continuous between x and $x+h$. Equation (1) can also be written as:

$$f(h+x) = f(h) + f'(h)x + \frac{f''(h)}{2!}x^2 + \frac{f'''(h)}{3!}x^3 + \dots \quad (2)$$

If we put $h = 0$ in equation (2), we get

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots \quad (3)$$

This equation (3) is called Maclaurin series.

Example

Derive the Maclaurin series of $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$

Solution

Maclaurin series is simply a Taylor series for the point $h = 0$.

$$f(x) = \sin(x), \Rightarrow f(0) = 0$$

$$f'(x) = \cos(x), \Rightarrow f'(0) = 1$$

$$f''(x) = -\sin(x), \Rightarrow f''(0) = 0$$

$$f'''(x) = -\cos(x), \Rightarrow f'''(0) = -1$$

$$f''''(x) = \sin(x), \Rightarrow f''''(0) = 0$$

$$f'''''(x) = \cos(x), \Rightarrow f'''''(0) = 1$$

Using the Maclurians series now,

$$\begin{aligned} f(x) &= f(0) + f'(0)x + f''(0)\frac{x^2}{2!} + f'''(0)\frac{x^3}{3!} + f''''(0)\frac{x^4}{4!} + f'''''(0)\frac{x^5}{5!} + \dots \\ &= 0 + 1 \cdot x + 0 \frac{x^2}{2!} - 1 \frac{x^3}{3!} + 0 \frac{x^4}{4!} + 1 \frac{x^5}{5!} + \dots \\ &= x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots \end{aligned}$$

Thus we can find the approximate values of $\sin x$ by using the basic arithmetic operations of addition, subtraction, division, and multiplication. Therefore it is possible write a program to find value of $\sin x$ without using *math.h* header file.

Example

Find the value of $e^{0.25}$ using the first five terms of the Maclaurin series.

Solution

The first five terms of the Maclaurin series for e^x can be calculated as

$$\begin{aligned} f(x) &= e^x, & \Rightarrow & f(0) = 1 \\ f'(x) &= e^x, & \Rightarrow & f'(0) = 1 \\ f''(x) &= e^x, & \Rightarrow & f''(0) = 1 \\ f'''(x) &= e^x, & \Rightarrow & f'''(0) = 1 \\ f''''(x) &= e^x, & \Rightarrow & f''''(0) = 1 \end{aligned}$$

Thus, from Maclurians series

$$\begin{aligned} f(x) &= f(0) + f'(0)x + f''(0)\frac{x^2}{2!} + f'''(0)\frac{x^3}{3!} + f''''(0)\frac{x^4}{4!} + f'''''(0)\frac{x^5}{5!} + \dots \\ e^x &\approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \\ e^{0.25} &\approx 1 + 0.25 + \frac{0.25^2}{2!} + \frac{0.25^3}{3!} + \frac{0.25^4}{4!} \\ &= 1.2840 \end{aligned}$$

The exact value of $e^{0.25}$ up to 5 significant digits is also 1.2840.

Example

Find the value of $f(6)$ given that $f(4) = 125$, $f'(4) = 74$, $f''(4) = 30$, $f'''(4) = 6$ and all other higher derivatives of $f(x)$ at $x = 4$ are zero.

Solution

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2!} + f'''(x)\frac{h^3}{3!} + \dots$$

$$x = 4$$

$$h = 6 - 4 = 2$$

Since fourth and higher derivatives of $f(x)$ are zero at $x = 4$.

$$f(4+2) = f(4) + f'(4)2 + f''(4)\frac{2^2}{2!} + f'''(4)\frac{2^3}{3!}$$

$$\begin{aligned} f(6) &= 125 + 74(2) + 30\left(\frac{2^2}{2!}\right) + 6\left(\frac{2^3}{3!}\right) \\ &= 125 + 148 + 60 + 8 \\ &= 341 \end{aligned}$$

Note that to find $f(6)$ exactly, we only needed the value of the function and all its derivatives at some other point, in this case, $x = 4$. We did not need the expression for the function and all its derivatives. Taylor series application would be redundant if we needed to know the expression for the function, as we could just substitute $x = 6$ in it to get the value of $f(6)$.

1.6.1 Error in Taylor Series

As you have noticed, the Taylor series has infinite terms. Only in special cases such as a finite polynomial does it have a finite number of terms. So whenever you are using a Taylor series to calculate the value of a function, it is being calculated approximately.

The Taylor polynomial of order n of a function $f(x)$ with $(n+1)$ continuous derivatives in the domain $[x, x+h]$ is given by

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2!} + \dots + f^{(n)}(x)\frac{h^n}{n!} + R_n(x)$$

Where the remainder is given by

$$R_n(x) = \frac{(x-h)^{n+1}}{(n+1)!} f^{(n+1)}(c).$$

Where

$$x < c < x+h$$

That is, c is some point in the domain $(x, x+h)$.

Example

The Taylor series for e^x at point $x = 0$ is given by

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots$$

- What is the truncation (true) error in the representation of e^1 if only four terms of the series are used?
- Use the remainder theorem to find the bounds of the truncation error.

Solution

If only four terms of the series are used, then

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$$

$$\begin{aligned} e^1 &\approx 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} \\ &= 2.66667 \end{aligned}$$

The truncation (true) error would be the unused terms of the Taylor series, which then are

$$\begin{aligned} E_t &= \frac{x^4}{4!} + \frac{x^5}{5!} + \dots \\ &= \frac{1^4}{4!} + \frac{1^5}{5!} + \dots \approx 0.0516152 \end{aligned}$$

But is there any way to know the bounds of this error other than calculating it directly?

Yes,

$$f(x+h) = f(x) + f'(x)h + \dots + f^{(n)}(x) \frac{h^n}{n!} + R_n(x)$$

where

$$R_n(x) = \frac{(x-h)^{n+1}}{(n+1)!} f^{(n+1)}(c), \quad x < c < x+h, \text{ and}$$

c is some point in the domain $(x, x+h)$. So in this case, if we are using four terms of the Taylor series, the remainder is given by ($x=0, n=3$)

$$\begin{aligned} R_3(x) &= \frac{(0-1)^{3+1}}{(3+1)!} f^{(3+1)}(c) \\ &= \frac{1}{4!} f^{(4)}(c) = \frac{e^c}{24} \end{aligned}$$

Since

$$x < c < x+h$$

$$0 < c < 0+1$$

$$0 < c < 1$$

The error is bounded between

$$\frac{e^0}{24} < R_3(1) < \frac{e^1}{24}$$

$$\frac{1}{24} < R_3(1) < \frac{e}{24}$$

$$0.041667 < R_3(1) < 0.113261$$

So the bound of the error is less than 0.113261 which does concur with the calculated error of 0.0516152.

1.7 MEAN VALUE THEOREM

Suppose f is continuous on $[a, b]$ and differentiable on (a, b) . Draw the secant line joining the point $(a, f(a))$ to the point $(b, f(b))$. Clearly there's a point c where $a < c < b$ such that the tangent line to the graph of f at $x = c$ is parallel to, thus has the same slope as, the secant line. The slope of the secant line is $(f(b) - f(a))/(b - a)$, and that of the tangent line is $f'(c)$.

$$\text{So, } f'(c) = \frac{f(b) - f(a)}{b - a}$$

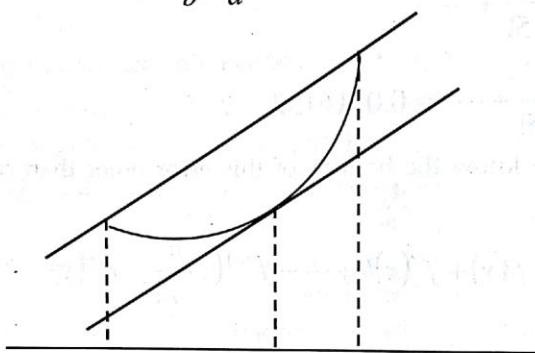


Figure 1.1: Graphical Interpretation of Mean Value Theorem

Example

Consider $f(x) = x^3 + 3x^2$ on the interval $-5 \leq x \leq 1$. Since f is a polynomial, f is continuous on $-5 \leq x \leq 1$ and differentiable on $-5 < x < 1$.

$$\frac{f(1) - f(-5)}{1 - (-5)} = \frac{4 + 50}{6} = 9$$

So we should be able to find a number c between -5 and 1 such that $f'(c) = 9$.

$$f'(x) = 3x^2 + 6x, \text{ so } f'(c) = 3c^2 + 6c.$$

Now,

Set $f'(c)$ equal to 9 and solve for c

We get,

$$\begin{aligned} 3c^2 + 6c &= 9 \\ \Rightarrow c^2 + 2c &= 3 \\ \Rightarrow c^2 + 2c - 3 &= 0 \\ \Rightarrow c = -3 \quad \text{or} \quad c &= 1 \end{aligned}$$

$c = 1$ is not in the interval $-5 < x < 1$ --- it's an endpoint. But $c = -3$ is. Therefore, $c = -3$ is a number satisfying the conclusion of the Mean Value Theorem.

1.8 ALGORITHM ANALYSIS

Complexity analysis of an algorithm is very hard if we try to analyze exact. we know that the complexity of an algorithm is the mathematical function of the size of the input. So if we analyze the algorithm in terms of bound (upper and lower) then it would be easier. For this purpose we need the concept of asymptotic notations.

1.8.1 Big Oh (O) notation

When we have only asymptotic upper bound then we use O notation. A function $f(x)=O(g(x))$ (read as $f(x)$ is big oh of $g(x)$) iff there exists two positive constants c and k such that for all $x \geq k$, $f(x) \leq c \cdot g(x)$. The above relation says that $g(x)$ is an upper bound of $f(x)$.

Example 1.15

Given that $f(n) = 3n^2 + 4n + 7$ & $g(n) = n^2$, then prove that $f(n) = O(g(n))$.

Proof

let us choose c and k values as 14 and 1 respectively then we can have

$$f(n) \leq c \cdot g(n), n \geq k \text{ as}$$

$$3n^2 + 4n + 7 \leq 14 \cdot n^2 \text{ for all } n \geq 1$$

The above inequality is trivially true

$$\text{Hence } f(n) = O(g(n))$$

1.8.2 Big Omega (Ω) notation

Big omega notation gives asymptotic lower bound. A function $f(x) = \Omega(g(x))$ (read as $f(x)$ is big omega of $g(x)$) iff there exists two positive constants c and k such that for all $x \geq k$, $c \cdot g(x) \leq f(x)$. The above relation says that $g(x)$ is a lower bound of $f(x)$.

Example 1.16

Given that $f(n) = 3n^2 + 4n + 7$ & $g(n) = n^2$, then prove that $f(n) = \Omega(g(n))$.

Proof

let us choose c and K values as 1 and 1, respectively then we can have

$$f(n) \geq c \cdot g(n), n \geq K \text{ as}$$

$$3n^2 + 4n + 7 \geq 1 \cdot n^2 \text{ for all } n \geq 1$$

The above inequality is trivially true

$$\text{Hence } f(n) = \Omega(g(n))$$

1.8.3 Big Theta (Θ) notation

When we need asymptotically tight bound then we use notation Big- Θ . A function $f(x) = \Theta(g(x))$ (read as $f(x)$ is big theta of $g(x)$) iff there exists three positive constants c_1, c_2 and k such that for all $x \geq k$, $c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)$. The above relation says that $f(x)$ is order of $g(x)$.

Example 1.17

Given that $f(n) = 3n^2 + 4n + 7$ & $g(n) = n^2$, then prove that $f(n) = \Theta(g(n))$.

Proof

Let us choose c_1, c_2 and k values as 14, 1 and 1 respectively then we can have,

$$f(n) \leq c_1 \cdot g(n), n \geq k \text{ as } 3n^2 + 4n + 7 \leq 14 \cdot n^2, \text{ and}$$

$f(n) \geq c_2 * g(n)$, $n \geq k$ as $3n^2 + 4n + 7 \geq 1 * n^2$ for all $n \geq 1$ (in both cases).

So

$c_2 * g(n) \leq f(n) \leq c_1 * g(n)$ is trivial.

Hence

$$f(n) = \Theta(g(n)).$$

EXERCISE

1. How true error is different from approximate error? Discuss each with suitable examples.
2. "While measuring errors it is better to use relative errors than absolute errors", do you agree with the statement? Justify your answer with sufficient facts.
3. What are the different sources of errors? Discuss round off and truncation errors with suitable examples.
4. Why floating point representation is necessary? How binary numbers are represented in floating point format? Explain with example.
5. Derive Maclurians Series from Taylor's series. Use Maclurians Series to calculate the value of $\cos(90^\circ)$.
6. Calculate the error bound in Taylor's series while calculating the value of $\sin(45^\circ)$ by using first five terms of the series.
7. Write a program in c/c++ language to calculate the value of $e^{0.5}$ without using *math.h* header file.
8. State mean value theorem and verify this by using at least two examples of your choice.



Chapter 2

SOLVING NONLINEAR EQUATIONS

Chapter Content

- Solving Nonlinear Equations
- Direct Analytical Method
- Graphical Method
- Trial and Error Method
- Iterative Methods
- Synthetic Division
- Remainder Theorem
- Roots of polynomial by using Newton-Raphson Method
- Horner's Method for Polynomial Evaluation

2.1 SOLVING NONLINEAR EQUATIONS

Equation whose graph does not form a straight line (linear) is called a Nonlinear Equation. In a nonlinear equation, the variables are either of degree greater than one or less than one, but never one. In a nonlinear system output is not directly proportional to its input; a linear system fulfills this condition. In other words, a nonlinear system is any problem where the variable(s) to be solved for cannot be written as a linear combination of independent components. A typical example is the equation $2x^2+x-6=0$. More generally, we will be concerned with any equation of the form $a_nx^n+a_{n-1}x^{n-1}+\dots+a_2x^2+a_1x+a_0=0$ where a_n, a_{n-1}, \dots, a_0 are known numbers and we are trying to find values of x. Besides this, a non-linear equation can also be algebraic equation or transcendental equation. An equation $y = f(x)$ is algebraic if it can be expressed in the form $f_n y^n + f_{n-1} y^{n-1} + \dots + f_1 y + f_0 = 0$, where f_i is an i^{th} order polynomial. For example, $2x+3xy-4y+5=0$ is an algebraic equation. Transcendental equations contain non-algebraic expressions like exponential, trigonometric, logarithmic and other functions. For example, $f(x)=e^{0.2x} \sin(3x-0.5)$ is a transcendental equation. Methods of solving linear equations can be categorized into four types:

- ❖ Direct Analytical Method
- ❖ Graphical Methods
- ❖ Trial and Error Methods
- ❖ Iterative Methods

2.2 DIRECT ANALYTICAL METHOD

In this method equations are solved by using derived formulas. We just need to supply the value of input parameters in the equation. For example, Solution of quadratic equation $ax^2 + bx + c = 0$ given by the formula give below:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Example

Solve the equation $2x^2 + 5x + 2 = 0$.

Solution

Here,

$$a=2, \quad b=5, \quad c=2$$

Now,

$$\text{First Root}(x_1) = \frac{-5 + \sqrt{5^2 - 4 \times 2 \times 2}}{2 \times 2} = \frac{-5 + \sqrt{25 - 16}}{4} = \frac{-5 + 3}{4} = -\frac{1}{2}$$

$$\text{Second Root}(x_2) = \frac{-5 - \sqrt{5^2 - 4 \times 2 \times 2}}{2 \times 2} = \frac{-5 - \sqrt{25 - 16}}{4} = \frac{-5 - 3}{4} = -2$$

2.3 GRAPHICAL METHOD

This is the simplest method to determine the root of an equation $f(x)=0$. The procedure is quite straightforward:

- ☞ Plot the function $f(x)$
- ☞ Observe when it crosses the x-axis, this point represents the value for which $f(x)=0$

This will provide only a rough approximation of the root. However, this rough approximation of the root can be used as a first guess for other more accurate techniques.

Example

Solve the equation $2x^2 + 5x + 2 = 0$ by using graphical method.

Solution

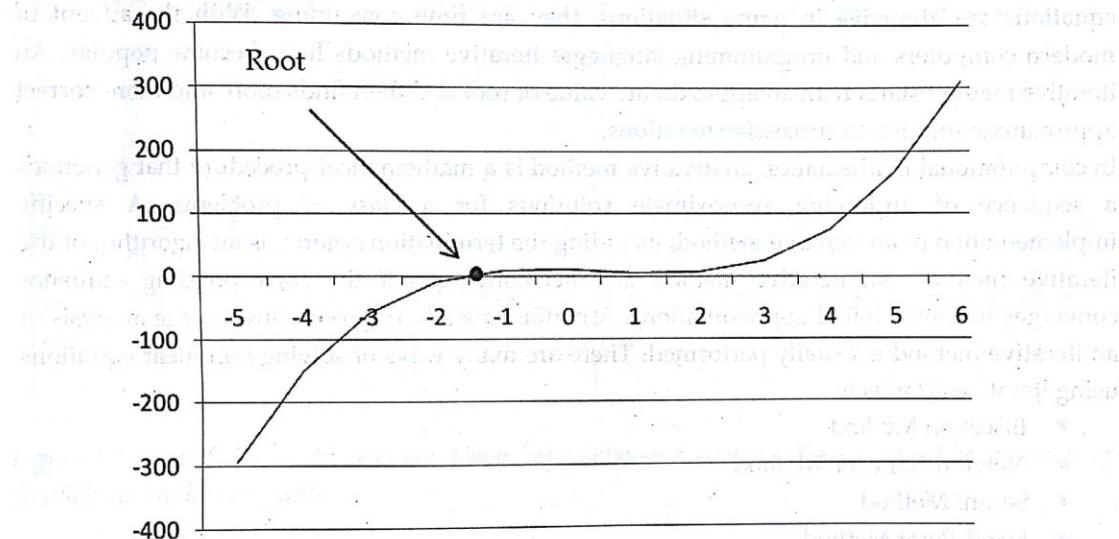


Figure 2.1: Plot of function $2x^2 + 5x + 2 = 0$

If the above figure, graph plotting the given function $2x^2 + 5x + 2 = 0$ intersects the x-axis nearly at $x = -1$. Thus one of the roots of the equation is -1 . But, since the equation is cubic, there exists three roots of the equation.

2.4 TRIAL AND ERROR METHOD

In this method we make a series of guesses for x and then evaluate $f(x)$ at that x if it is close to zero, it is one of the approximate root of the given nonlinear equation. Otherwise make another guess for x and repeat the same process again until the x for which $f(x) \approx 0$ is found.

Example

Solve the equation $f(x)=2x^2+x-6=0$

Solution

Step 1: Guess $x=0$	\Rightarrow	$f(x) = -6$
Step 2: Guess $x=1$	\Rightarrow	$f(x) = -3$
Step 3: Guess $x=2$	\Rightarrow	$f(x) = -4$
Step 4: Guess $x=1.5$	\Rightarrow	$f(x) = 0$
Step 5: Guess $x=-1$	\Rightarrow	$f(x) = -5$
Step 6: Guess $x=-2$	\Rightarrow	$f(x) = 0$

2.5 ITERATIVE METHODS

Although direct analytical method and graphical methods for finding roots of nonlinear equations are attractive in many situations, they are time consuming. With the advent of modern computers and programming languages iterative methods have become popular. An iterative method starts with an approximate value of root and then finds more and more correct approximate solution in successive iterations.

In computational mathematics, an iterative method is a mathematical procedure that generates a sequence of improving approximate solutions for a class of problems. A specific implementation of an iterative method, including the termination criteria, is an algorithm of the iterative method. An iterative method is called convergent if the corresponding sequence converges for given initial approximations. A mathematically rigorous convergence analysis of an iterative method is usually performed. There are many ways of solving nonlinear equations using iterative approach:

- ☞ Bisection Method
- ☞ Newton-Raphson Method
- ☞ Secant Method
- ☞ Fixed-Point Method
- ☞ False Position Method

2.5.1 Bisection Method (Half Interval Method)

One of the first numerical methods developed to find the root of a nonlinear equation $f(x)=0$ was the bisection method (also called Binary-Search method or half interval method). The method is based on the following theorem: An equation, $f(x)=0$ where $f(x)$ is a real continuous function, has at least one root between x_l and x_u if $f(x_l)f(x_u) < 0$.

Note that if $f(x_l)f(x_u) > 0$, there may or may not be any root between x_l and x_u . If $f(x_l)f(x_u) < 0$, then there may be more than one root between x_l and x_u . So the theorem only guarantees one root between x_l and x_u . Since the method is based on finding the root between two points, the method falls under the category of bracketing methods.

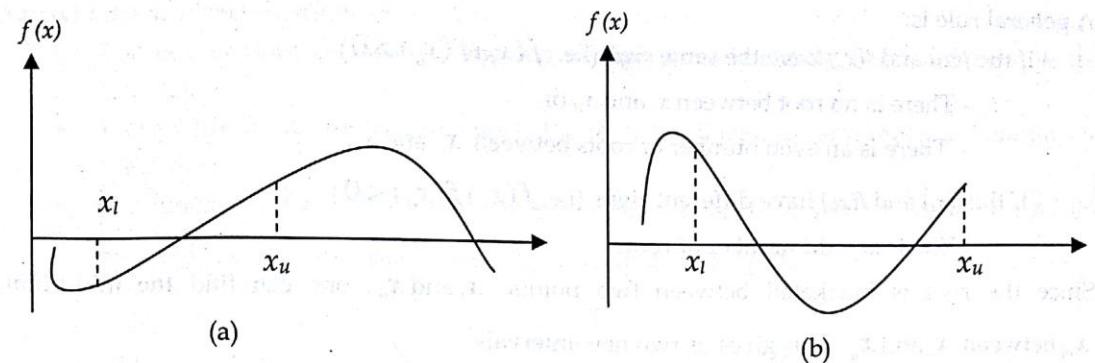


Figure 2.2: Existence of root between two points of real, continuous function (a) At least one root exists if the function changes sign. (b) Even number of roots may exist if the function does not change sign.

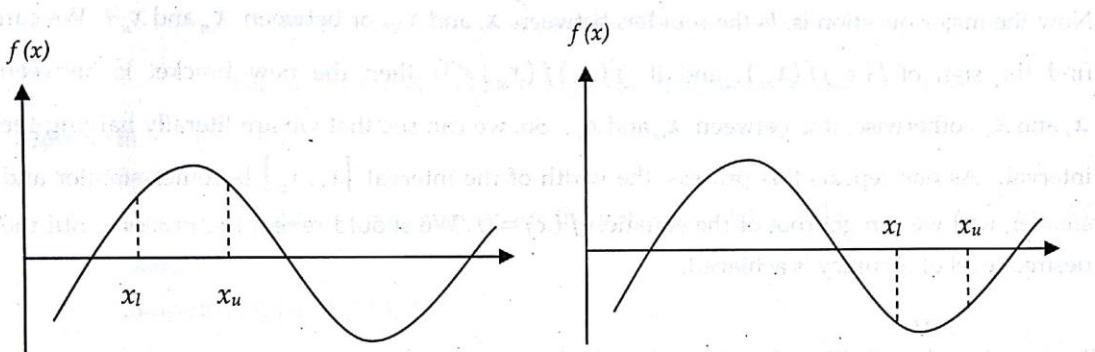


Figure 2.3: If the function $f(x)$ does not change sign between two points, there may not be any roots of $f(x)=0$ between the two points.

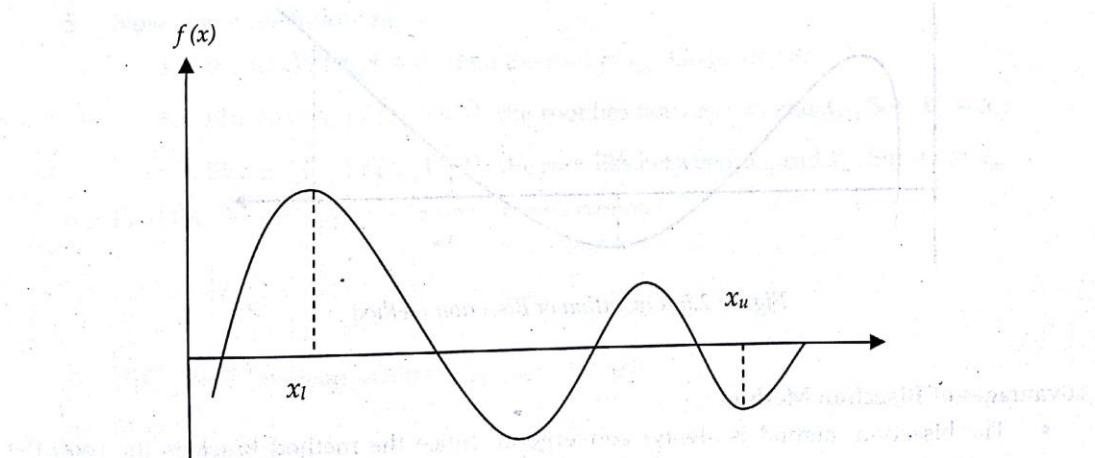


Figure 2.4: If the function $f(x)$ changes sign between two points, more than one root for $f(x)=0$ exists between the two points.

A general rule is:

- If the $f(x_l)$ and $f(x_u)$ have the same sign (i.e. $f(x_l)f(x_u) > 0$)
 - There is no root between x_l and x_u or
 - There is an even number of roots between x_l and x_u .
- If the $f(x_l)$ and $f(x_u)$ have different signs (i.e. $f(x_l)f(x_u) < 0$)
 - There is an odd number of roots.

Since the root is bracketed between two points, x_l and x_u , one can find the mid-point, x_m between x_l and x_u . This gives us two new intervals

1. x_l and x_m , and
2. x_m and x_u .

Now the major question is: Is the root lies between x_l and x_m , or between x_m and x_u ? We can find the sign of $f(x_l)f(x_m)$, and if $f(x_l)f(x_m) < 0$ then the new bracket is between x_l and x_m , otherwise, it is between x_m and x_u . So, we can see that we are literally halving the interval. As one repeats this process, the width of the interval $[x_l, x_u]$ becomes smaller and smaller, and we can get root of the equation $f(x) = 0$. We should repeat the process until the desired level of accuracy is achieved.

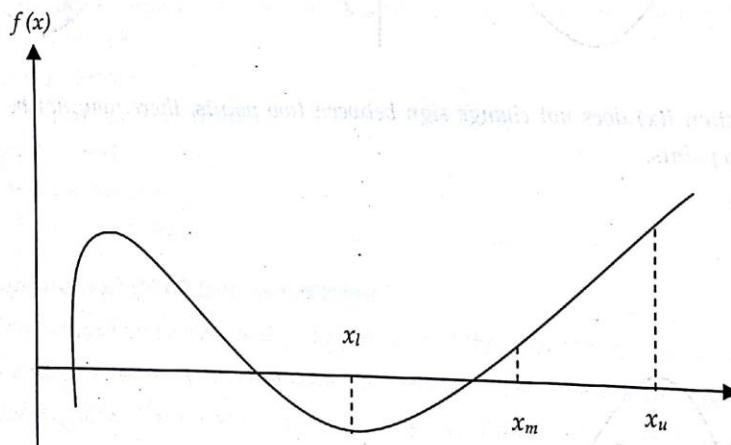


Figure 2.5: Operation of Bisection method

Advantages of Bisection Method

- The bisection method is always convergent. Since the method brackets the root, the method is guaranteed to converge.
- As iterations are conducted, the interval gets halved. So one can guarantee the decrease in the error in the solution of the equation.

Drawbacks of Bisection Method

- The convergence of bisection method is slow as it is simply based on halving the interval.
- If one of the initial guesses is closer to the root, it will take larger number of iterations to reach the root.
- If a function $f(x)$ is such that it just touches the x-axis it will be unable to find the lower guess, x_l , and upper guess, x_u , such that $f(x_l)f(x_u) < 0$

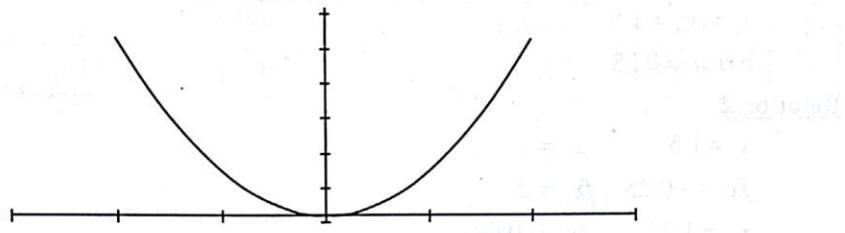


Figure 2.6: Function has a single root that cannot be bracketed.

Algorithm

- Start
- Choose x_l and x_u as two guesses for the root such that $f(x_l)f(x_u) < 0$ and stopping criterion E
- Compute $f(x_l)$ and $f(x_u)$
- Estimate the root, x_m of the equation $f(x) = 0$ as the mid-point between x_l and x_u as

$$x_m = \frac{x_l + x_u}{2}$$

- Now check the following
 - If $f(x_l)f(x_m) = 0$; then the root is x_m . Go to step 8.
 - Else If $f(x_l)f(x_m) < 0$, the root lies between x_l and x_m . Set $x_u = x_m$.
 - Else if $f(x_l)f(x_m) > 0$, the root lies between x_m and x_u . Set $x_l = x_m$.
- Find the absolute approximate relative error as

$$|E_{ra}| = \left| \frac{x_l - x_u}{x_u} \right|$$

- If $|E_{ra}| > E$, then go to Step 3, else go to step 8.
- Stop

Example

Solve the Equation $f(x) = 3x^2 - 6x + 2 = 0$

Solution

Assume Initial guess is $x_l = 1$ and $x_u = 2$

Iteration 1

$$x_l = 1 \quad x_u = 2$$

$$fx_l = -1 \quad fx_u = 2$$

$$x_m = 1.5 \quad fx_m = -0.25$$

since $fx_u \times fx_m < 0$

$$x_l = x_m = 1.5$$

$$\text{Error} = 0.25$$

Iteration 2

$$x_l = 1.5 \quad x_u = 2$$

$$fx_l = -0.25 \quad fx_u = 2$$

$$x_m = 1.75 \quad fx_m = 0.687$$

since $fx_l \times fx_m < 0$

$$x_u = x_m = 1.75$$

$$\text{Error} = 0.143$$

Iteration 3

$$x_l = 1.5 \quad x_u = 1.75$$

$$fx_l = -0.25 \quad fx_u = 0.687$$

$$x_m = 1.625 \quad fx_m = 0.171$$

since $fx_l \times fx_m < 0$

$$x_u = x_m = 1.625$$

$$\text{Error} = 0.076$$

Iteration 4

$$x_l = 1.5 \quad x_u = 1.625$$

$$fx_l = -0.25 \quad fx_u = 0.171$$

$$x_m = 1.562 \quad fx_m = -0.051$$

since $fx_u \times fx_m < 0$

$$x_l = x_m = 1.562$$

$$\text{Error} = 0.038$$

Since error is below the specified limit

$$\text{root} = 1.562$$

Above solution can also be shown in table as below

Iteration	x_l	x_u	x_m	$f(x_l)$	$f(x_u)$	$f(x_m)$	Error
1	1.000	2.000	1.500	-1.000	2.000	-0.250	0.500
2	1.500	2.000	1.750	-0.250	2.000	0.688	0.250
3	1.500	1.750	1.625	-0.250	0.688	0.172	0.143
4	1.500	1.625	1.563	-0.250	0.172	-0.051	0.077
5	1.563	1.625	1.594	-0.051	0.172	0.058	0.038
6	1.563	1.594	1.578	-0.051	0.058	0.003	0.020
7	1.563	1.578	1.570	-0.051	0.003	-0.024	0.010

Thus, root=1.57

Second Example

Using the bisection method solve $x^2 - 4\cos x = 0$, correct up to 2 significant figures.

Solution

Assume Initial guess is $x_l=1$ and $x_u=2$,

Iteration 1

$$x_l = 1 \quad x_u = 2$$

$$fx_l = -1.161 \quad fx_u = 5.664$$

$$x_m = 1.5 \quad fx_m = 1.97$$

since $fx_l \times fx_m < 0$

$$x_u = x_m = 1.5$$

$$Error = 0.333$$

Iteration 2

$$x_l = 1 \quad x_u = 1.5$$

$$fx_l = -1.161 \quad fx_u = 1.97$$

$$x_m = 1.25 \quad fx_m = 0.301$$

since $fx_l \times fx_m < 0$

$$x_u = x_m = 1.25$$

$$Error = 0.166$$

Iteration 3

$$x_l = 1 \quad x_u = 1.25$$

$$fx_l = -1.161 \quad fx_u = 0.301$$

$$x_m = 1.125 \quad fx_m = -0.459$$

since $fx_u \times fx_m < 0$

$$x_l = x_m = 1.125$$

$$Error = 0.10$$

Iteration 4

$$\begin{aligned}
 x_l &= 1.125 & x_u &= 1.25 \\
 f(x_l) &= -0.459 & f(x_u) &= 0.171 \\
 x_m &= 1.1875 & f(x_m) &= -0.0858 \\
 \text{since } f(x_u) \times f(x_m) &< 0 & \\
 x_l &= x_m = 1.1875 & \\
 \text{Error} &= 0.05
 \end{aligned}$$

Iteration 5

$$\begin{aligned}
 x_l &= 1.1875 & x_u &= 1.25 \\
 f(x_l) &= -0.0858 & f(x_u) &= 0.171 \\
 x_m &= 1.216 & f(x_m) &= 0.0906 \\
 \text{since } f(x_l) \times f(x_m) &< 0 & \\
 x_u &= x_m = 1.216 & \\
 \text{Error} &= 0.023
 \end{aligned}$$

Iteration 6

$$\begin{aligned}
 x_l &= 1.1875 & x_u &= 1.216 \\
 f(x_l) &= -0.0858 & f(x_u) &= 0.0906 \\
 x_m &= 1.201 & f(x_m) &= 0.002 \\
 \text{since } f(x_l) \times f(x_m) &< 0 & \\
 x_u &= x_m = 1.201 & \\
 \text{Error} &= 0.011
 \end{aligned}$$

Iteration 7

$$\begin{aligned}
 x_l &= 1.1875 & x_u &= 1.201 \\
 f(x_l) &= -0.0858 & f(x_u) &= 0.002 \\
 x_m &= 1.194 & f(x_m) &= -0.0419 \\
 \text{since } f(x_u) \times f(x_m) &< 0 & \\
 x_l &= x_m = 1.194 & \\
 \text{Error} &= 0.005
 \end{aligned}$$

Since error is below the specified limit
 root = 1.194

Above solution can also be shown in table as below

Iteration	x_l	x_u	x_m	$f(x_l)$	$f(x_u)$	$f(x_m)$	Error
1	1.000	2.000	1.500	-1.161	5.665	1.967	0.500
2	1.000	1.500	1.250	-1.161	1.967	0.301	0.333
3	1.000	1.250	1.125	-1.161	0.301	-0.459	0.200
4	1.125	1.250	1.188	-0.459	0.301	-0.086	0.100
5	1.188	1.250	1.219	-0.086	0.301	0.106	0.050
6	1.188	1.219	1.203	-0.086	0.106	0.010	0.026
7	1.188	1.203	1.195	-0.086	0.010	-0.038	0.013
8	1.195	1.203	1.199	-0.038	0.010	-0.014	0.006

Thus, root=1.199

// C program for Bisection Method

```
#include<stdio.h>
#include<conio.h>
int a0,a1,a2,a3;
float f(float x)
{
    return(a3*x*x*x+a2*x*x+a1*x+a0);
}
int main()
{
    float xl,xm,xu,fxl,fxm,fxu,E,Era;
    printf("Enter cofficients a3,a2,a1 and a0\n");
    scanf("%d%d%d%d",&a3,&a2,&a1,&a0);
    printf("Enter initial bracket and E\n");
    scanf("%f%f%f",&xl,&xu,&E);
    fxl=f(xl);fxu=f(xu);
    while(1)
    {
        fxl=f(xl);fxu=f(xu);
        xm=(xl+xu)/2;
        fxm=f(xm);
        if((fxl*fxm)==0)
            printf("Root=%f\n",xm);
        else if((fxl*fxm)<0)
            xu=xm;
        else
            xl=xm;
        Era=(xu-xl)/xu;
    }
}
```

```

if(Era<E)
{
    printf("Root=%f\n",xm);
    break;
}
getch();
}

```

Output

Enter coefficients a3, a2, a1 and a0

0 3 -6 2

Enter initial bracket and E

1 2 0.05

Root=1.562500

Convergence of Bisection Method

In bisection method interval is halved every iteration. After n^{th} iteration size of interval is reduced to

$$\Delta_n = \frac{(x_u - x_l)}{2^n}$$

Now we can say that maximum error after n^{th} iteration is $E_n = \pm \Delta_n$

$$\Rightarrow |E_n| = \frac{(x_u - x_l)}{2^n}$$

Similarly, after $(n+1)^{\text{th}}$ iteration maximum error is given by

$$E_{n+1} = \frac{(x_u - x_l)}{2^{n+1}}$$

$$E_{n+1} = \frac{(x_u - x_l)}{2 * 2^n}$$

$$E_{n+1} = \frac{E_n}{2} \quad (1)$$

This equation shows that error is halved after each iteration of bisection method therefore we can say that bisection method converges linearly.

2.5.2 Newton Raphson Method

Newton-Raphson method is based on the principle that if the initial guess of the root of $f(x)=0$ is at x_i , and if one draws the tangent to the curve at $f(x_i)$, the point x_{i+1} where the tangent crosses the x-axis is an improved estimate of the root

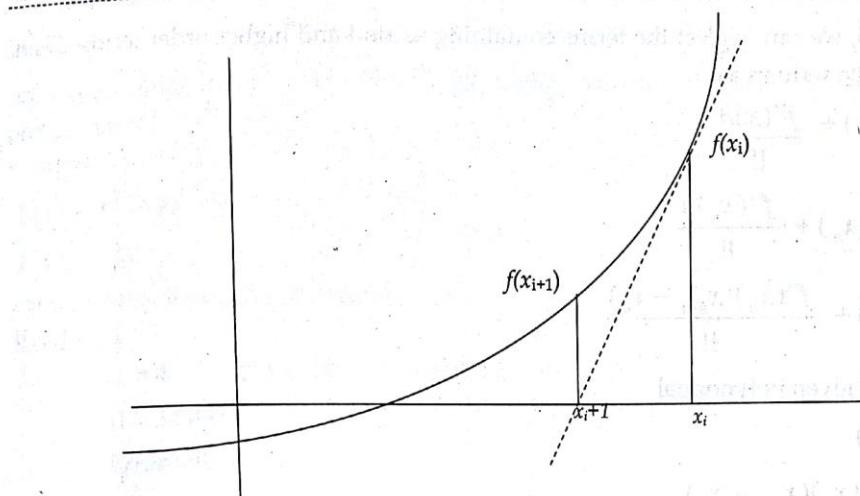


Figure 2.7: Geometrical interpretation of Newton-Raphson Method

Using the definition of the slope of a function, at $x = x_i$

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

If $f(x)$ have root at x_{i+1} , then $f(x_{i+1})=0$. Putting this in above equation, we get

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

This gives

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

This equation is called the Newton-Raphson formula for solving nonlinear equations of the form $f(x)=0$. So starting with an initial guess x_i , we can find the next guess x_{i+1} , by using the above equation. We can repeat this process until the root within a desirable tolerance is found.

Deriving Newton Raphson formula from Taylors Series

Let h is an small increment in x_n to get next estimate of root

$$x_{n+1} - x_n = h$$

$$x_{n+1} = x_n + h$$

We know that Taylors series can be written as

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

Since h is very small, we can neglect the terms containing second and higher order terms. Then, above equation can be written as:

$$f(x+h) = f(x) + \frac{f'(x)h}{1!}$$

$$f(x_n + h) = f(x_n) + \frac{f'(x_n)h}{1!}$$

$$f(x_{n+1}) = f(x_n) + \frac{f'(x_n)(x_{n+1} - x_n)}{1!}$$

If x_{n+1} is root of the given polynomial

$$\Rightarrow f(x_{n+1}) = 0$$

$$\Rightarrow f(x_n) + \frac{f'(x_n)(x_{n+1} - x_n)}{1!} = 0$$

Solving above equation to get the value of x_{n+1} we get

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{--- (1)}$$

This equation is called Newton Raphson formula

Algorithm

1. Start
2. Input initial guess x_0 and precision E
3. Evaluate $f(x)$ symbolically
4. Use x_0 to estimate the new value of the root x_1 as

$$x_i = x_0 - \frac{f(x_0)}{f'(x_0)}$$

5. Find the absolute relative approximate error, $|\epsilon_a|$ as

$$|\epsilon_a| = \left| \frac{x_1 - x_0}{x_1} \right|$$

6. Compare the absolute relative approximate error, $|\epsilon_a|$ with the pre-specified relative error tolerance, E .

If $|\epsilon_a| > E$

$$x_0 = x_1$$

go to step 3,

else

go to step 7

7. Terminate

Example

Solve the equation $x^2 + 4x - 9 = 0$ by using Newton Raphson Method. Assume Error precision is 0.1.

Solution

$$f(x) = x^2 + 4x - 9$$

$$f'(x) = 2x + 4$$

Let us assume that initial guess is 4.0

Iteration 1

$$x_0 = 4 \quad f(x) = 23 \quad f'(x) = 12$$

$$x_1 = 2.0833$$

$$\text{Error} = 0.92$$

Iteration 2

$$x_0 = 2.0833 \quad f(x) = 3.673 \quad f'(x) = 8.166$$

$$x_1 = 1.633$$

$$\text{Error} = 0.275$$

Iteration 3

$$x_0 = 1.633 \quad f(x) = 0.202 \quad f'(x) = 7.266$$

$$x_1 = 1.605$$

$$\text{Error} = 0.017$$

Since error is less than specified limit

Root=1.605

Above solution can also be shown in table as below

Iteration	x	$f(x)$	$f'(x)$	Error
1	4.000	23.000	12.000	
2	2.083	3.674	8.167	0.920
3	1.634	0.202	7.267	0.275
4	1.606	0.001	7.211	0.017
5	1.606	0.000	7.211	0.000

Thus, root=1.606

Second Example

Solve the equation $e^{-x} - 3x = 0$ by using Newton Raphson Method. Assume Error precision is 0.01.

Solution

$$f(x) = e^{-x} - 3x$$

$$f'(x) = -e^{-x} - 3$$

Let us assume that initial guess is $x_0=1$

Iteration 1

$$x_0 = 1 \quad f(x) = -2.632 \quad f'(x) = -3.367$$

$$x1 = 0.218$$

$$Error = 3.577$$

Iteration 2

$$x_0 = 0.218 \quad f(x) = 0.15 \quad f'(x) = -3.8$$

$$x1 = 0.2574$$

$$Error = 0.153$$

Iteration 3

$$x_0 = 0.254 \quad f(x) = 0.0023 \quad f'(x) = -3.773$$

$$x1 = 0.2576$$

$$Error = 0.0024$$

Since error is less than specified limit

$$\text{Root} = 0.2576$$

Above solution can also be shown in table as below

Iteration	x	f(x)	f'(x)	Error
1	1.000	-2.632	-3.368	
2	0.218	0.148	-3.804	3.577
3	0.257	0.001	-3.773	0.151
4	0.258	0.000	-3.773	0.001

Thus, root=0.258

```
//C program for Newton-Raphson method
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define F(x) (a3*x*x*x+a2*x*x+a1*x+a0)
#define FD(x) (3*a3*x*x+2*a2*x+a1) //derivative of F(x)
float a0,a1,a2,a3;
int main()
{
    float x0,x1,fx0,fdx0,E,Er;
    printf("Enter coefficients a3,a2,a1 and a0\n");
    scanf("%f%f%f%f",&a3,&a2,&a1,&a0);
    printf("Enter initial guess and E\n");
    scanf("%f%f",&x0,&E);
    while(1)
    {

```

```

fx0=F(x0);fdx0=FD(x0);
x1=x0-fx0/fdx0;
Er=(x1-x0)/x1;
if(fabs(Er)<E)
{
    printf("Root=%f\n",x1);
    break;
}
x0=x1;
if(i==50)
break;
}
getch();
}

```

Output

Enter coefficients a3,a2,a1 and a0

1 0 5 -3

Enter initial guess and E

2 0.05

Root=0.564102

Drawbacks of Newton-Raphson Method

- **Division by zero:** Consider the function $f(x) = 1 - x^2$. It has a maximum at $x=0$ and solutions of $f(x) = 0$ at $x = \pm 1$. If we start iterating from the $x_0=0$ (where the derivative is zero), x_1 will be undefined, since the tangent at $(0,1)$ is parallel to the x -axis:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{1}{0} = \text{undefined}$$

The same issue occurs if, instead of the starting point, any iteration point is stationary. Even if the derivative is small but not zero, the next iteration will be a far worse approximation.

- **Oscillations near local maximum and minimum:** For some functions, some starting points may enter an infinite cycle, preventing convergence. Let $f(x) = x^3 - 2x + 2$ and take 0 as the starting point. The first iteration produces 1 and the second iteration returns to 0 so the sequence will alternate between the two without converging to a root.

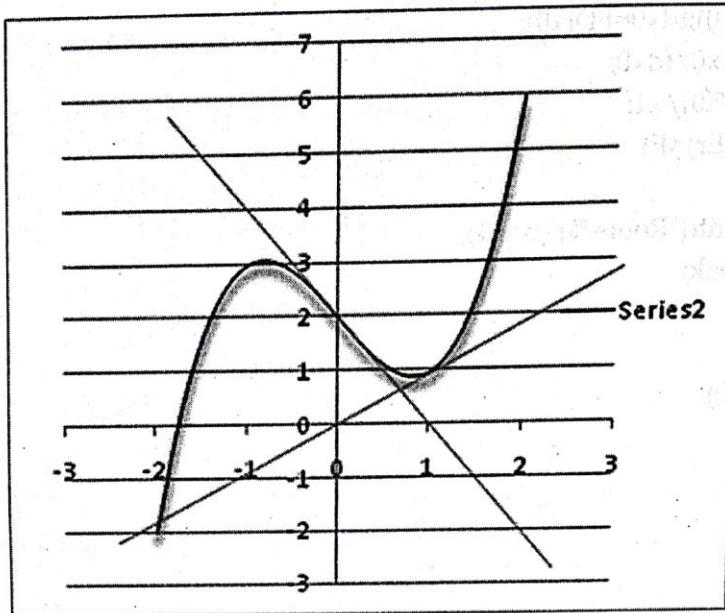


Figure 2.8: Oscillation of roots

Convergence of Newton-Raphson Method

Suppose x_r is root of $f(x)=0$ and x_n is estimated root of $f(x)=0$ such that $|x_r - x_n| = \delta \ll 1$. Then by Taylor's Series expansion:

$$f(x_r) = f(x_n + \delta) = f(x_n) + f'(x_n)(x_r - x_n) + R_1 \quad (1)$$

Where R_1 is Lagrange form of the Taylor series expansion remainder and is given by the equation

$$R_1 = \frac{1}{2!} f''(\varepsilon)(x_r - x_n)^2$$

Where ε is in between x_n and x_r . Since x_r is the root, equation (1), becomes:

$$0 = f(x_n) + f'(x_n)(x_r - x_n) + \frac{1}{2!} f''(\varepsilon)(x_r - x_n)^2 \quad (2)$$

Dividing equation (2) by $f'(x_n)$ and rearranging gives

$$\frac{f(x_n)}{f'(x_n)} + (x_r - x_n) = -\frac{f''(\varepsilon)}{2f'(x_n)}(x_r - x_n)^2 \quad (3)$$

We know that according to Newton Raphson formula x_{n+1} is given by

$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

Thus equation (3) can be written as

$$x_r - x_{n+1} = -\frac{f''(\varepsilon)}{2f'(x_n)}(x_r - x_n)^2 \quad (4)$$

Let $E_n = x_r - x_n$ and $E_{n+1} = x_r - x_{n+1}$, Equation (4) Can be written as

$$E_{n+1} = -\frac{f''(\varepsilon)}{2f'(x_n)} E_n^2$$

Taking absolute value of both sides gives

$$|E_{n+1}| = \frac{|f''(\varepsilon)|}{2|f'(x_n)|} E_n^2 \quad (5)$$

Equation (5) shows that the Newton-Raphson method have quadratic rate of convergence.

2.5.3 The secant method

The Newton-Raphson method of solving the nonlinear equation $f(x)=0$ is given by the recursive formula

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

From the above equation, it is clear that one of the drawbacks of the Newton-Raphson method is that you have to evaluate the derivative of the function. With availability of symbolic manipulators such as Maple, Mathcad, Mathematica and Matlab, this process has become more convenient. However, it is still can be a laborious process. To overcome this drawback, the derivative $f'(x)$ of the function $f(x)$ is approximated as

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \quad (2)$$

Substituting Equation (2) in (1), gives

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \quad (3)$$

The above equation is called the Secant method. This method requires two initial guesses, but unlike the bisection method, the two initial guesses do not need to bracket the root of the equation. The Secant method may or may not converge, but when it converges, it converges faster than the bisection method. However, since the derivative is approximated, it converges slower than Newton-Raphson method.

Derivation of Secant Method from Geometry

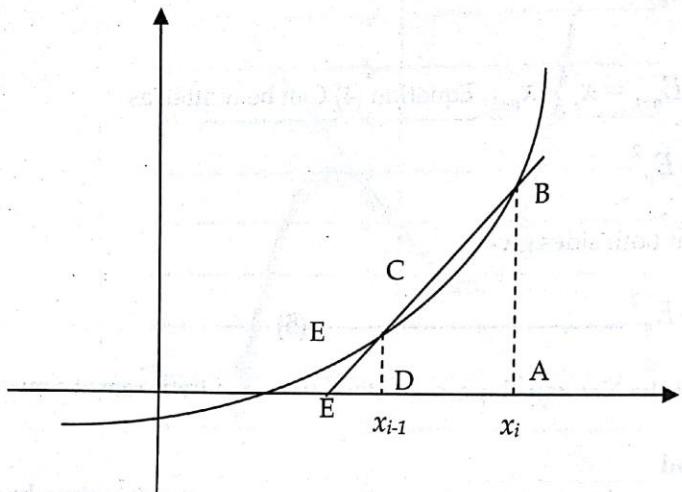


Figure: Geometrical representation of the Secant method.

The Secant method can be also derived from geometry shown in above figure. Let us consider two initial guesses x_i and x_{i-1} and draw a straight line between $f(x_i)$ and $f(x_{i-1})$ passing through the x -axis at x_{i+1} . Now, ABE and DCE are similar triangles and therefore following relation holds

$$\frac{AB}{AE} = \frac{DC}{DE}$$

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

Rearranging the terms we get

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

Adding and subtracting $f(x_i)x_i$ from numerator, we get

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

This equation is called secant formula.

Algorithm

1. Start
2. Input two initial guess (say x_0 and x_1) and precision.(Say E)
3. Evaluate $f(x_0)$ and $f(x_1)$
4. Estimate new value of the root x_2 as

$$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

5. Find the absolute relative approximate error, $|\epsilon_a|$ as

$$|\epsilon_a| = \left| \frac{x_2 - x_1}{x_2} \right|$$

6. Compare the absolute relative approximate error, $|\epsilon_a|$ with the pre-specified relative error tolerance, E.

If $|\epsilon_a| > E$,

$$x_0 = x_1$$

$$f(x_0) = f(x_1)$$

$$x_1 = x_2$$

go to step 3,

else

go to step 7

7. Terminate

Example

Solve the equation $2x^2 + 4x - 10 = 0$ with error precision 0.05

Solution

Let us assume that $x_0=2$ $x_1=6$, and specified precision is 0.05

Iteration 1

$$x_0 = 2 \quad f(x_0) = 6$$

$$x_1 = 6 \quad f(x_1) = 86$$

=>

$$x_2 = 1.7 \quad \text{Error} = 2.529$$

Iteration 2

$$x_0 = 6 \quad f(x_0) = 86$$

$$x_1 = 1.7 \quad f(x_1) = 2.58$$

=>

$$x_2 = 1.57 \quad \text{Error} = 0.828$$

Iteration 3

$$x_0 = 1.7 \quad f(x_0) = 2.58$$

$$x_1 = 1.57 \quad f(x_1) = 1.18$$

=>

$$x_2 = 1.46 \quad \text{Error} = 0.07534$$

42 Numerical Methods with Practical Approach

Iteration 4

$$x_0 = 57 \quad f(x_0) = 1.18$$

$$x_1 = 1.46 \quad f(x_1) = 0.55$$

=>

$$x_2 = 1.45 \quad Error = 0.00689$$

Since absolute value of Error is less than specified limit

=> Root=1.45

Above solution can also be shown in table as below

Iteration	x_0	x_1	$f(x_0)$	$f(x_1)$	Error
1	2.000	6.000	6.000	86.000	
2	6.000	1.700	86.000	2.580	2.529
3	1.700	1.567	2.580	1.179	0.085
4	1.567	1.455	1.179	0.055	0.077
5	1.455	1.450	0.055	0.001	0.004

Thus, root=1.45

Second Example

Solve the equation $\cos x + 2 \sin x + x^2 = 0$ with error precision 0.01
Solution

Let us assume that $x_0=0$ $x_1=2$, and specified precision is 0.01
Iteration 1

$$x_0 = 0 \quad f(x_0) = 1$$

$$x_1 = 2 \quad f(x_1) = 5.402$$

$$x_2 = -0.454 \quad Error = 5.402$$

Iteration 2

$$x_0 = 2 \quad f(x_0) = 5.402$$

$$x_1 = -0.454 \quad f(x_1) = 0.227$$

$$x_2 = -0.562 \quad Error = 0.192$$

Iteration 3

$$x_0 = -0.454 \quad f(x_0) = 0.227$$

$$x_1 = -0.562 \quad f(x_1) = 0.096$$

$$x_2 = -0.641 \quad Error = 0.123$$

Iteration 4

$$x_0 = -0.562 \quad f(x_0) = 0.096$$

$$x_1 = -0.641 \quad f(x_1) = 0.016$$

$$x_2 = -0.657 \quad Error = 0.024$$

Iteration 5

$$x_0 = -0.641 \quad f(x_0) = 0.016$$

$$x_1 = -0.657 \quad f(x_1) = 0.0017$$

$$x_2 = -0.659 \quad Error = 0.003$$

Since absolute value of Error is less than specified limit

Root=-0.659

Above solution can also be shown in table as below

Iteration	x_0	x_1	$f(x_0)$	$f(x_1)$	Error
1	0.000	2.000	1.000	5.402	
2	2.000	-0.454	5.402	0.227	5.402
3	-0.454	-0.562	0.227	0.096	0.192
4	-0.562	-0.641	0.096	0.016	0.123
5	-0.641	-0.657	0.016	0.002	0.024
6	-0.657	-0.659	0.002	0.000	0.003

Thus, root=1.45

//C program for Secant Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define F(x) (a3*x*x*x+a2*x*x+a1*x+a0)
float a0,a1,a2,a3;
int main()
{
    float x0,x1,x2,fx0,fx1,E,Er;
    printf("Enter cofficients a3,a2,a1 and a0\n");
    scanf("%f%f%f%f",&a3,&a2,&a1,&a0);
    printf("Enter two initial guesses and E\n");
    scanf("%f%f%f",&x0,&x1,&E);
    while(1)
    {
        fx0=F(x0);fx1=F(x1);
        x2=x1-(fx1*(x1-x0))/(fx1-fx0);
        Er=(x2-x1)/x2;
        if(fabs(Er)<E)
        {
            printf("Root=%f\n",x2);
            break;
        }
    }
}
```

```

x0=x1;
x1=x2;
}
getch();
}

```

Output

Enter coefficients a₃, a₂, a₁ and a₀

1 0 -1 -5

Enter two initial guesses and E

1 4 0.05

Root=1.890508

Convergence of Secant Method

Secant formula is given by

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \quad (1)$$

Let x_r be the actual root of $f(x)$ and e_i be the error estimate in i^{th} iteration
 \Rightarrow

$$x_{i+1} = e_{i+1} + x_r$$

$$x_i = e_i + x_r$$

$$x_{i-1} = e_{i-1} + x_r$$

Substitution these values in equation (1), we get

$$\begin{aligned} e_{i+1} &= e_i - \frac{f(x_i)(e_i - e_{i-1})}{f(x_i) - f(x_{i-1})} \\ e_{i+1} &= \frac{e_{i-1}f(x_i) - e_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} \end{aligned} \quad (2)$$

According to mean value theorem there exists at least one point say $x=t_i$ in the interval x_i and x_r such that

$$f'(t_i) = \frac{f(x_i) - f(x_r)}{x_i - x_r} \quad (3)$$

Since $f(x_r) = 0$ and $e_i = x_i - x_r$, equation (3) becomes

$$f'(t_i) = \frac{f(x_i)}{e_i}$$

$$f(x_i) = e_i f'(t_i)$$

Similarly,

$$f(x_{i-1}) = e_{i-1} f'(t_{i-1})$$

Substituting these values of $f(x_i)$ and $f(x_{i-1})$ in equation (2), we get

$$e_{i+1} = \frac{e_{i-1} e_i f'(t_i) - e_i e_{i-1} f'(t_{i-1})}{f(x_i) - f(x_{i-1})}$$

$$e_{i+1} = \frac{e_{i-1} e_i (f'(t_i) - f'(t_{i-1}))}{f(x_i) - f(x_{i-1})}$$

\Rightarrow

$$e_{i+1} \propto e_{i-1} e_i \quad \text{---(4)}$$

By definition of rate of convergence, the method is of order p iff

$$e_i \propto e^{p}_{i-1} \quad \text{---(5)}$$

Or

$$e_{i+1} \propto e^p_i \quad \text{---(6)}$$

From 4, 5 and 6, we get

$$e_i^p \propto e_{i-1} e_i$$

$$\Rightarrow e^p_i \propto e_{i-1} e^{p}_{i-1}$$

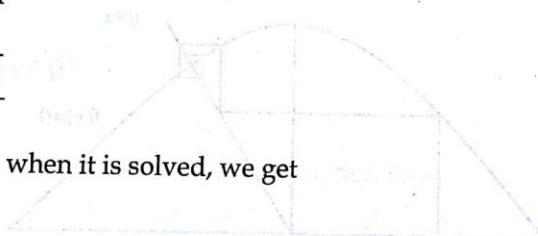
$$e_i \propto e_{i-1}^{(p+1)/p} \quad \text{---(7)}$$

Comparing equation (5) and (7), we get

$$p = (p+1)/p$$

$$p^2 - p - 1 = 0$$

$$p = \frac{1 \pm \sqrt{5}}{2}$$



This is called golden ratio, when it is solved, we get

$$p \approx 1.62$$

Thus,

$$e_i \propto e_{i-1}^{1.62}$$

This equation shows that convergence of secant method is superlinear

2.5.4 Fixed-point method

In fixed-point iteration method we rearrange the function $f(x)=0$ such that x is on the left-hand side of the equation. This can be expressed as below:

$$f(x)=0 \quad \text{---(1)}$$

is written as

$$x = g(x) \quad \text{---(2)}$$

Equation (2) called fixed point equation. This can be achieved by algebraic manipulation or by simply adding x to both sides of the original equation, example:

$$x^2 + 3x - 1 = 0 \Leftrightarrow x = \frac{1-x^2}{3}$$

$$\sin(x) = 0 \Leftrightarrow x = \sin(x) + x$$

Since equation (1) and equation (2) are similar, root of equation (2) is also root of equation (1). Root of equation (2) is given by point of intersection of straight line $y=x$ and the curve $x=g(x)$. This point of intersection is called fixed point of $g(x)$. Point p is a fixed point of the function $f(x)$ if and only if $f(p) = p$. For example, if function f is defined on the real numbers by $f(x) = x^2 - 3x + 4$ then 2 is a fixed point of f , because $f(2) = 2$.

Equation (2) provides a convenient way of predicting the next value of x on the basis of current value of x . If x_0 is the initial guess to the root, next approximated value of root can be calculated as

$$x_1 = g(x_0)$$

Similarly, further approximation can be given as

$$x_2 = g(x_1)$$

After generalizing this, we get

$$x_{n+1} = g(x_n) \quad \dots \dots \dots \quad (3)$$

This equation is called fixed point formula and gives rise to the sequence x_0, x_1, x_2, \dots which is hoped to converge to a point x (fixed point) as shown in figure given below:

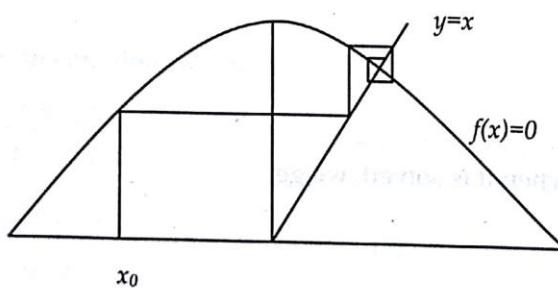


Figure 2.10: Working of Fixed Point method.

Algorithm

1. Start
2. Input initial guess (say x_0) and precision (Say E)
3. Convert $f(x) = 0$ to the form $x = g(x)$
4. Estimate new value of the root x_2 as

$$x_1 = g(x_0)$$

5. Find the absolute relative approximate error, $|E_a|$ as

$$|\varepsilon_a| = \left| \frac{x_1 - x_0}{x_1} \right|$$

6. Compare the absolute relative approximate error, $|\varepsilon_a|$ with the pre-specified relative error tolerance, E.

If $|\varepsilon_a| > E$,

$$x_0 = x_1$$

go to step 4

else

go to step 7

7. Terminate

Example

Solve the equation $x^2 - 6x + 8 = 0$ by assuming error precision 0.01.

Solution

Above equation can be written as $x = x^2 - 5x + 8$

Assume that initial value of guess is 1.

Iteration 1

$$x_0 = 1.00 \quad x_1 = 4.00$$

$$\text{Error} = 0.7500$$

Iteration 2

$$x_0 = 4.00 \quad x_1 = 4.00$$

$$\text{Error} = 0.0000$$

Since, Absolute value of error is less than specified limit

Root=4.00

Second Example

Solve the equation $1 + 1/2 \sin(x) - x = 0$ by assuming error precision 0.001.

Solution

Above equation can be written as $x = 1 + 1/2 \sin(x)$

Assume that initial value of guess is 0.

Iteration 1

$$x_0 = 0 \quad x_1 = 1$$

$$\text{Error} = 1$$

Iteration 2

$$x_0 = 1 \quad x_1 = 1.42$$

$$\text{Error} = 0.296$$

Iteration 3

$$x_0 = 1.42 \quad x_1 = 1.494$$

$$\text{Error} = 0.0492$$

Iteration 4

$$x_0 = 1.494 \quad x_1 = 1.4985$$

$$\text{Error} = 0.0027$$

Iteration 5

$$x_0 = 1.4985 \quad x_1 = 1.4986$$

$$\text{Error} = 0.000103$$

Since, Absolute value of error is less than specified limit

$$\text{Root}=1.4986$$

//C program for Fixed Point Iteration Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define G(x) (a3*x*x*x+a2*x*x+a1)/(-a0)
float a0,a1,a2,a3;
int main()
{
    float x0,x1,E,Er;
    printf("Enter coefficients a3,a2,a1 and a0\n");
    scanf("%f%f%f%f",&a3,&a2,&a1,&a0);
    printf("Enter initial guess and E\n");
    scanf("%f%f",&x0,&E);
    while(1)
    {
        x1=G(x0);
        Er=(x1-x0)/x1;
        if(fabs(Er)<E)
        {
            printf("Root=%f\n",x1);
            break;
        }
        x0=x1;
    }
    getch();
}
```

Output

Enter coefficients a3,a2,a1 and a0

1	0	-7	2
---	---	----	---

Enter initial guess and E

1	0.05
---	------

Root=0.289455

Convergence of Fixed-Point Method

Fixed point formula is given as

$$x_{n+1} = g(x_n)$$

Let x_f be the root of given equation

$$x_f = g(x_f)$$

Subtracting above two equations we get

$$x_f - x_{n+1} = g(x_f) - g(x_n) \quad \dots \dots \dots (1)$$

From mean value theorem there is at least one point $x=t$ in the interval (x_f, x_n) such that

$$g'(t) = \frac{g(x_f) - g(x_n)}{x_f - x_n}$$

$$\Rightarrow g(x_f) - g(x_n) = g'(t)(x_f - x_n)$$

Substituting this value in equation (1), we get

$$x_f - x_{n+1} = g'(t)(x_f - x_n) \quad \dots \dots \dots (2)$$

Since,

$$e_{n+1} = x_f - x_{n+1}$$

$$e_n = x_f - x_n$$

Equation (2) becomes

$$e_{n+1} = g'(t)e_n \quad \dots \dots \dots (3)$$

Equation (3) says that error decreases with each iteration only if $g'(t) < 1$. Hence we say that fixed point method converges only under the condition $g'(t) < 1$.

Analysis of Convergence of Fixed-Point Iteration from Taylor Series

We know that,

$$E_{n+1} = r - x_{n+1}$$

$$E_n = r - x_n$$

Where r is the exact value of the root, x_n is the value of the n^{th} iterate and E_n is the error after n^{th} iterations. Hence, fixed-point iteration can be written as

$$x_{n+1} = g(x_n)$$

$$r - E_{n+1} = g(r - E_n)$$

If E_n is small we may expand the function g in a Taylor series about r . Hence,

$$g(r - E_n) = g(r) - g'(r)E_n + \frac{g''(r)}{2}E_n^2 + \dots$$

$$r - E_{n+1} = g(r) - g'(r)E_n + \frac{g''(r)}{2}E_n^2 + \dots$$

Since r is a fixed-point it satisfies the relation $r = g(r)$, the leading term in the Taylor series gives

$$E_n \approx g'(r)E_n$$

Equation (3) says that error decreases with each iteration only if $g'(r) < 1$. Hence we say that fixed point method converges only under the condition $g'(r) < 1$.

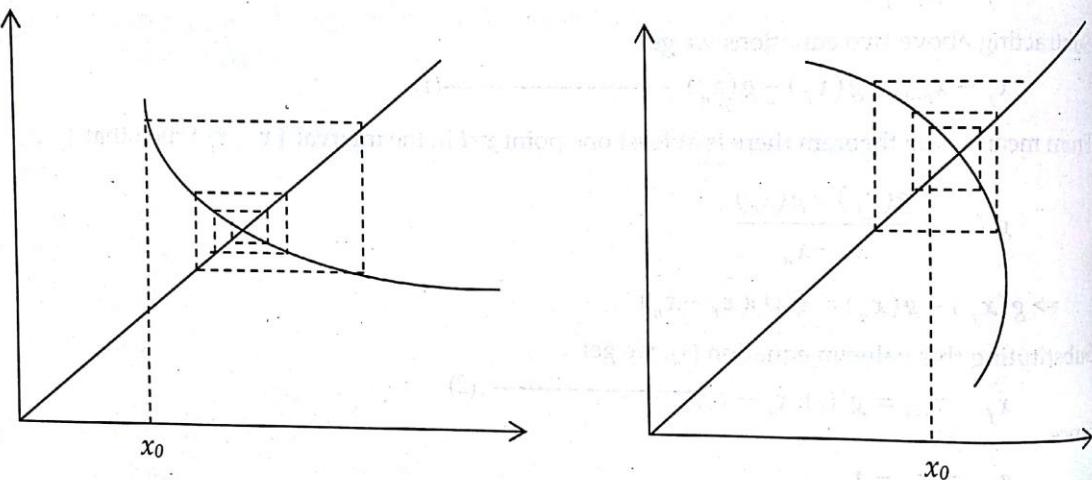


Figure 2.11: Convergence and divergence of Fixed point method

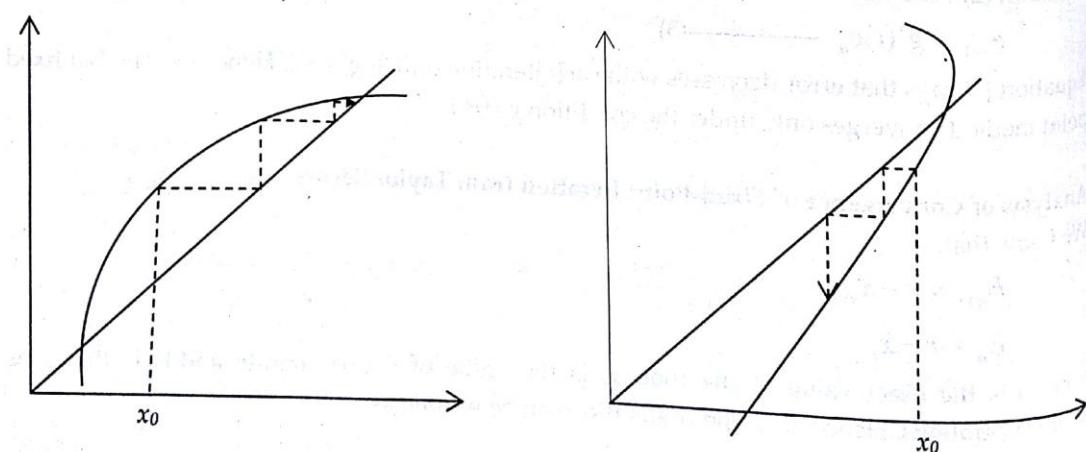


Figure 2.12: Convergence and Divergence of Fixed point method.

2.6 SYNTHETIC DIVISION

Synthetic division is a method of performing polynomial long division, with less writing and fewer calculations. It is mostly taught for division by binomials of the form $x-a$. The most useful aspect of synthetic division is that it allows us to calculate without writing variables and uses fewer calculations. As well, it takes significantly less space than long division. Most importantly, the subtractions in long division are converted to additions by switching the signs at the very beginning, preventing sign errors.

Step 1: Set up the synthetic division

When you write out the dividend make sure that you write it in descending powers and you insert 0's for any missing terms. For example, if you had the problem $(x^4 - 3x + 5) \div (x - 4)$, the polynomial, starts out with degree 4, then the next highest degree is 1. It is missing degrees 3 and 2. So if we were to put it inside a division box we would write it like this:

$$x - 4 \overline{)x^4 + 0x^3 + 0x^2 - 3x + 5}$$

When you set this up using synthetic division write c for the divisor $x - c$. Then write the coefficients of the dividend to the right, across the top. Include any 0's that were inserted in for missing terms.

$$\begin{array}{r|ccccc} 4 & 1 & 0 & 0 & -3 & 5 \\ & \hline & & & & \end{array}$$

Step 2: Bring down the leading coefficient to the bottom row

$$\begin{array}{r|ccccc} 4 & 1 & 0 & 0 & -3 & 5 \\ & \hline & 1 & & & \end{array}$$

Step 3: Multiply c by the value just written on the bottom row

$$\begin{array}{r|ccccc} 4 & 1 & 0 & 0 & -3 & 5 \\ & \hline & 4 & & & \\ & \hline & 1 & & & \end{array}$$

Step 4: Add the column created in step 3

$$\begin{array}{r|ccccc} 4 & 1 & 0 & 0 & -3 & 5 \\ & \hline & 4 & & & \\ & \hline & 1 & 4 & & \end{array}$$

Step 5: Repeat until done

$$\begin{array}{r|ccccc} 4 & 1 & 0 & 0 & -3 & 5 \\ & \hline & 4 & 16 & 64 & 244 \\ & \hline & 1 & 4 & 16 & 61 & 249 \end{array}$$

Step 6: Write out the answer

The numbers in the last row make up your coefficients of the quotient as well as the remainder. The final value on the right is the remainder. Working right to left, the next number is your constant, the next is the coefficient for x , the next is the coefficient for x squared, etc...

$$\text{Quotient} = x^3 + 4x^2 + 16x + 61$$

$$\text{Remainder} = 249$$

Let $p(x)$ be a polynomial of degree n . If we divide $p(x)$ by $(x - a)$, we get another polynomial $q(x)$, which is quotient, of degree $n-1$.

Assume

$$p(x) = a_n x^n + a^{n-1} x_{n-1} + \dots + a_0$$

And

$$q(x) = b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \dots + b_0$$

$$\Rightarrow$$

$$p(x) = (x - a)q(x)$$

$$a_n x^n + a^{n-1} x_{n-1} + \dots + a_0 = (x - a)(b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \dots + b_0)$$

Comparing the coefficients on both side, we get

$$b_{n-1} = a_n$$

$$b_{n-2} = a_{n-1} + a * b_{n-1}$$

:

$$b_0 = a_1 + a * b_1$$

Thus

$$b_n = 0 \quad \text{and}$$

$$b_{i-1} = a_i + a * b_i \quad i = n, \dots, 1$$

Algorithm

1. Input coefficients of dividend polynomial, say a_n, a_{n-1}, \dots, a_0
2. Input constant term of divisor polynomial of the form $x - a$
3. Set $b_n = 0$;
4. Repeat till $n > 0$

$$b_{n-1} = a_n + b_n * c$$
5. Remainder (R) = $a_0 + b_0 * c$
6. Print quotient polynomial and Remainder
7. Terminate

//C program for Synthetic Division

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```

int main()
{
    int a[100], b[100], r, c;
    int i, m, n;
    printf("Enter degree of polynomial\n");
    scanf("%d", &n);
    printf("Enter coefficients of dividend polynomial\n");
    for(i=n; i>=0; i--)
        scanf("%d", &a[i]);
    printf("Enter Constant term of divisor polynomial\n");
    scanf("%d", &c);
    b[n] = 0;
    m = n;
    while(m > 0)
    {
        b[m-1] = a[m] + b[m]*c;
        m--;
    }
    printf("Quotient:");
    m = n - 1;
    while(m >= 0)
    {
        if(b[m] != 0)
            printf("%d x^%d +", b[m], m);
        m--;
    }
    getch();
}

```

Output

Enter degree of polynomial
4

Enter coefficients of dividend polynomial

2 1 0 -6 4

Enter Constant term of divisor polynomial

2

Quotient: $2x^3 + 5x^2 + 10x^1 + 14$

2.7 REMAINDER THEOREM

Remainder Theorem states that if the polynomial $f(x)$ is divided by $x - c$, then the remainder is $f(c)$. This means that we can apply synthetic division and the last number is the remainder. This remainder is functional value of $f(x)$ at $x=c$. Thus The Remainder Theorem is useful for evaluating polynomials at a given value of x .

example

Given, $f(x) = 3x^3 + x^2 + x - 5$, use the Remainder Theorem to find $f(-2)$.

solution

We have to use synthetic division as described above. What is different is what are final answer is going to be. This time, we are looking for the functional value, so our answer will not be a quotient, but only the remainder.

$$\begin{array}{r} 3 & 1 & 1 & -5 \\ -2 & & & \\ \hline 3 & -5 & 11 & -27 \end{array}$$

\Rightarrow Remainder = -27

Thus: $f(-2) = -27$

2.8 FINDING MULTIPLE ROOTS BY USING NEWTON-RAPHSON METHOD

All real roots of polynomials can be found by repeatedly applying Newton's Method and synthetic division. Actually synthetic division is used to obtain polynomial of degree $n-1$ from polynomial of degree n . Root of polynomial can be found by using Newton's formula given below and initial guess .

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Suppose x_r is one of the root of polynomial of degree n . Now deflate the polynomial by dividing it by $x - x_r$ to get another polynomial of degree $n-1$ and use x_r as initial guess for finding next root. Continue this process until polynomial of degree one is achieved as quotient. This polynomial will be of the form $a_1x + a_0 = 0$. Now final root can be calculated as below:

$$x_r = -\frac{a_0}{a_1}$$

Algorithm

1. Start
2. Input degree and coefficient of polynomial
3. Input initial guess x_0 and error criterion E
4. While $n > 1$
 - ✓ Find root using Newton-Raphson algorithm, say n^{th} root is x_r
 - ✓ Divide the polynomial by $x - x_r$ to get the polynomial of degree $n-1$
 - ✓ Set $x_0 = x_r$
 - ✓ $n = n-1$
5. End While

6. $1^{\text{st}} \text{ root} = -\frac{a_0}{a_1}$

7. Terminate

Example

Find roots of equation $(x^4 - 2x^3 - 13x^2 + 38x - 24) = 0$ using newton raphson method
Fourth synthetic division.

Solution

Assume,

Initial guess (x_0) = 0.5 Error Limit (E) = 0.01\

Step1:

Use Newton Raphson method to get root

Thus, Fourth root of the equation =1.00

Step 2

Now, Use synthetic division deflate the polynomial by $(x-1)$, which gives the polynomial:

$$x^3 - x^2 - 14x + 24 = 0$$

Again use Newton Raphson method, with

Initial guess (x_0) = 1.00 Error Limit (E) = 0.01

Third root of the equation=2.00

Step 3

Again, Use synthetic division to deflate the polynomial by $(x-2)$, which gives the polynomial:

$$x^2 + x - 12 = 0$$

From Newton Raphson method, with

Initial guess (x_0) = 2.00 Error Limit (E) = 0.01

Second root of the equation=3.00

Again, Use synthetic division to deflate the polynomial by $(x-2)$, which gives the polynomial

$$x + 4 = 0$$

Now, first root of the polynomial is calculated using the formula given below:

$$\text{root} = -\frac{a_0}{a_1}$$

Thus, First Root = -4

//C Program for Calculating Multiple Roots Using Newton-Raphson Method

```

#include<stdio.h>
#include<conio.h>
#define F(x) (a[4]*x*x*x*x+a[3]*x*x*x+a[2]*x*x+a[1]*x+a[0])
#define FD(x) (4*a[4]*x*x*x+3*a[3]*x*x+2*a[2]*x+a[1])
float a[100],q[100];
int main()
{
    float x0,xr,fx0,fdx0,E,Er,c;
    int i,n,m;
    printf("Enter degree of polynomial\n");
    scanf("%d",&n);
    printf ("Enter coefficients of dividend polynomial\n");
    for(i=n;i>=0;i--)
        scanf("%f",&a[i]);
    printf("Enter initial guess and E\n");
    scanf("%f%f",&x0,&E);
    while(n>1)
    {
        while(1)
        {
            fx0=F(x0);fdx0=FD(x0);
            xr=x0-fx0/fdx0;
            Er=(xr-x0)/xr;
            if(fabs(Er)<E)
            {
                printf("Root%d=%f\n",n,xr);
                break;
            }
            x0=xr;
        }
        //Deflate the polynomial
        c=xr;
        q[n]=0;
        m=n-1;
        while(m>=0)
        {
            q[m]=a[m+1]+q[m+1]*c;
            m--;
        }
        for(i=n;i>=0;i--)
            a[i]=q[i];
    }
}

```

```

n=n-1;
x0=xr;
}
xr=-a[0]/a[1];
printf("Root%d=%f\n",n,xr);
getch();
}

```

Output

Enter degree of polynomial

4

Enter coefficients of dividend polynomial

1 -2 -13 38 -24

Enter initial guess and E

0.5 0.01

Root 4=1.000000

Root 3=2.000000

Root 2=3.000004

Root 1=-4.000004

2.9 HORNER'S METHOD FOR POLYNOMIAL EVALUATION

Horner's method is used for either of two things: As an algorithm for evaluating polynomials efficiently, or as a method for approximating the roots of a polynomial. Horner's Method is a way to optimize the task of evaluating a polynomial. The method splits the polynomial into its individual terms and solves them incrementally. This method separates the lowest degree term in the polynomial from the rest by grouping any terms with a higher degree into one unit with degree one. Thus, it represents any polynomial in the form $K*x+C$, where K is the group of all terms with a degree higher than one, and C is the term with degree zero. When the terms are grouped into K, their degrees decrease by one, as follows.

$$x^4 + 8x^3 + 6x^2 + 4x + 3 = K * x + 3 \quad \text{where } K = (x^3 + 8x^2 + 6x + 4)$$

This method can then be applied recursively inside the K to simplify the polynomial that results after grouping, by repeating these steps until there are no terms with degree higher than one and will yield an equation where there is no need to calculate x^n for any term. After applying this method to the sample polynomial we end up with:

$$x^4 + 8x^3 + 6x^2 + 4x + 3 \Rightarrow (((1)*x+8)*x+6)*x+4)*x+3$$

Generalizing this we can derive recursive relation for polynomial evaluation as below:

Given the polynomial

$$p(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_n x^n$$

where a_0, \dots, a_n are real numbers, we wish to evaluate the polynomial at a specific value of, say at x_0 . To accomplish this, we need to define given polynomial in terms of nested multiplication as below:

$$p(x) = (a_0 + x(a_1 + a_2 x + a_3 x^2 + \dots + a_{n-1} x^{n-1}))$$

$$p(x) = (a_0 + x(a_1 + x(a_2 + a_3 x + \dots + a_{n-2} x^{n-2})))$$

$$p(x) = (a_0 + x(a_1 + x(a_2 + x(a_3 + x(a_4 + \dots + x(a_{n-1} + x(a_n))))))$$

Now we can define a new sequence of constants as follows:

$$b_n = a_n$$

$$b_{n-1} = a_{n-1} + b_n x_0$$

Then b_0 is the value of $p(x_0)$

Algorithm

1. Start
2. Enter degree of polynomial, say n
3. Enter the value at which polynomial to be evaluated, x
4. Initially set $b_n = a_n$
5. While $n > 0$
 - $b_{n-1} = a_{n-1} + b_n * x$
6. End While
7. Display the value of b_0 , which is the value of polynomial at x
8. Terminate

Example

Evaluate the polynomial $p(x) = 3x^3 - 4x^2 + 5x - 6$ at $x=2$.

Solution

We know that

$$a_3 = 3, a_2 = -4, a_1 = 5, a_0 = -6$$

Now new sequence of constants can be determined by using recursive formula as below:

$$b_3 = a_3 = 3$$

$$b_2 = a_2 + b_3 * x = -4 + 3 * 2 = 2$$

$$b_1 = a_1 + b_2 * x = 5 + 2 * 2 = 9$$

$$b_0 = a_0 + b_1 * x = -6 + 9 * 2 = 12$$

Thus $p(2)=12$

//C program for Evaluating Polynomial using Horners Method

```
#include<stdio.h>
#include<conio.h>
#define P(x) (a[4]*x*x*x*x+a[3]*x*x*x+a[2]*x*x+a[1]*x+a[0])
float a[100],b[100];
int main()
{
    float x;
    int n,i;
    printf("Enter degree of polynomial\n");
    scanf("%d",&n);
    printf("Enter cofficients of dividend polynomial\n");
    for(i=n;i>=0;i--)
        scanf("%f",&a[i]);
    printf("Enter the value at which polynomial to be evaluated\n");
    scanf("%f",&x);
    b[n]=a[n];
    while(n>0)
    {
        b[n-1]=a[n-1]+b[n]*x;
        n--;
    }
    printf("Value of polynomial p(%f)=%f",x,b[0]);
    getch();
    return 0;
}
```

Output

Enter degree of polynomial
4

Enter cofficients of dividend polynomial
2 -1 3 -5 4

Enter the value at which polynomial to be evaluated
2

Value of polynomial p(2.00)=30.00

EXERCISE

1. Describe different types of non-linear equation with suitable examples.
2. Describe different techniques of solving non-linear equations briefly. Which method is more popular with advent of modern computers? Describe briefly about it.
3. Describe drawbacks of bisection method with their graphical explanation.
4. Prove that convergence of bisection method is linear.
5. Compare and contrast between Newton-Raphson method and Secant method.
6. Derive Newton-Raphson formula using Taylor's series
7. Describe the cases where Newton-Raphson method cannot find roots of non-linear equation.
8. What is fixed point? How fixed point method can be used to calculate roots of non-linear equations. Describe convergence criterion of fixed point method.
9. Why synthetic division is used. Write down algorithm of synthetic division. State and illustrate remainder theorem.
10. What is purpose of Horner's method? Illustrate significance of the method with suitable example.
11. Find the positive root of the following equation by bisection method
 - a) $x^3 - 4x - 9 = 0$
 - b) $e^x = 3x$
 - c) $x^3 + 3x - 1 = 0$
 - d) $x^3 + x^2 - 1 = 0$
 - e) $x^3 + 1.2x^2 = 4x + 4.8$
 - f) $3x = \cos x + 1$
 - g) $x e^x = 2$
 - h) $x \log_{10}x - 1.2$
 - i) $x^3 - 4x + 1 = 0$
 - j) $2x = 3 + \cos x$
12. Solve the following for positive root by Newton Raphson Method.
 - a) $2x^3 - 3x - 6 = 0$
 - b) $3x - \cos x - 1 = 0$
 - c) $x^3 - 6x - 4 = 0$
 - d) $x = \cos x$
 - e) $x \tan x = 1.28$
 - f) $\cos x = x^2$
 - g) $\cos x - x e^x = 0$
 - h) $x^4 - x - 9 = 0$
 - i) $x^3 + x + 1 = 0$
 - j) $x^3 + 2x^2 + 50x + 7 = 0$
13. Find the positive root of the following equation by Secant Method
 - a) $x^3 + x + 1 = 0$
 - b) $x^3 - 5x + 1 = 0$
 - c) $x^3 + x^2 - 1 = 0$
 - d) $x - \cos x = 0$

e) $x^3 - 2x + 5 = 0$

f) $x^3 - 5x + 1 = 0$

g) $x^3 + 3x - 1 = 0$

h) $e^x = 3x$

i) $2x = 3 + \cos x$

j) $\cos x - x e^x = 0$

Solve following non-linear equations using fixed point iteration

a) $\cos(x) - x e^x = 0$

b) $x^4 - x - 10 = 0$

c) $x - e^{-x} = 0$

d) $e^{-x} (x^2 - 5x + 2) + 1 = 0$

e) $x - \sin(x) - (1/2) = 0$

f) $e^{-x} = 3 \log(x)$

g) $\tan(x) - x - 1 = 0$



Chapter 3

INTERPOLATION

Chapter Content

- What is Interpolation
- Lagrange Interpolation
- Newton's Divided Difference Interpolation
- Interpolation with evenly spaced data
- Spline Interpolation

3.1 WHAT IS INTERPOLATION?

Given $n+1$ data points $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_n, y_n)$, interpolation is the process of finding an equation $y = f(x)$ that passes through all $n+1$ data points and using this equation to find value of y at x , $x_0 < x < x_n$. We can use interpolation to find value y at x that is not given in the set of data points. And extrapolation is the process of finding an equation $y = f(x)$ that passes through all $n+1$ data points and using this equation to find value of y at x , $x < x_0$ or $x > x_n$.

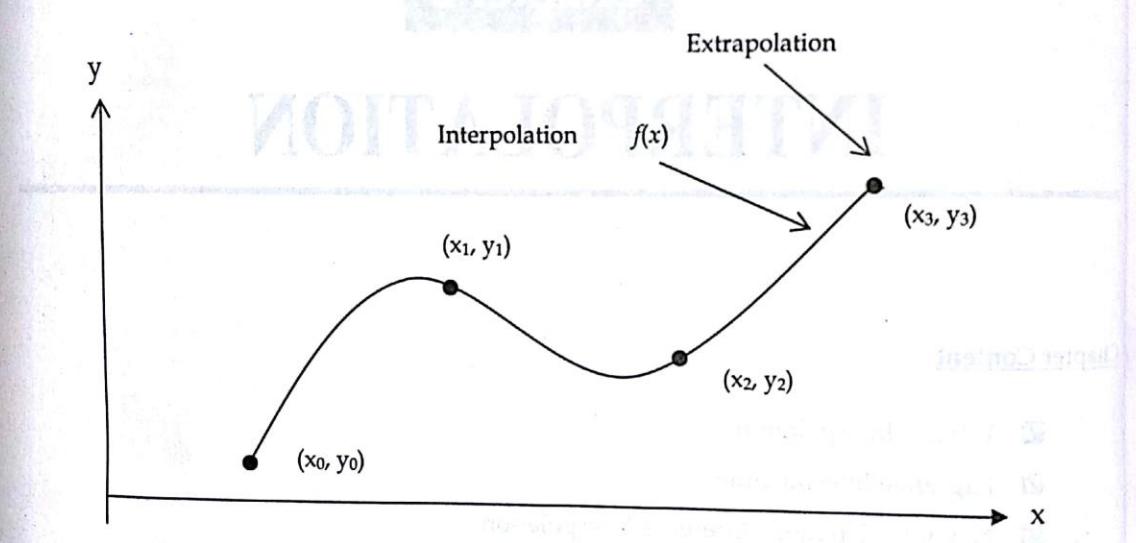


Figure 3.1 Interpolation and extrapolation

A polynomial is a common choice for an interpolating function because polynomials are easy to evaluate, differentiate, and integrate relative to other choices such as a trigonometric and exponential series.

3.2 LAGRANGE INTERPOLATION

A second order polynomial can be written in the form

$$p_2(x) = b_1(x - x_0)(x - x_1) + b_2(x - x_1)(x - x_2) + b_3(x - x_2)(x - x_0) \quad (1)$$

Let $(x_0, f_0), (x_1, f_1)$, and (x_2, f_2) are three interpolating points. Substituting these points in equation (1) we get

$$p_2(x_0) = f_0 = b_2(x_0 - x_1)(x_0 - x_2)$$

$$p_2(x_1) = f_1 = b_3(x_1 - x_2)(x_1 - x_0)$$

$$p_2(x_2) = f_2 = b_1(x_2 - x_0)(x_2 - x_1)$$

From above three equations we can calculate value of b_1 , b_2 , and b_3 . Formulae for calculating these values are given below

$$b_2 = \frac{f_0}{(x_0 - x_1)(x_0 - x_2)}$$

$$b_3 = \frac{f_1}{(x_1 - x_2)(x_1 - x_0)}$$

$$b_1 = \frac{f_2}{(x_2 - x_0)(x_2 - x_1)}$$

Substituting these values of b_1, b_2 , and b_3 in equation (1), we get

$$p_2(x) = \frac{f_0(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f_1(x - x_2)(x - x_0)}{(x_1 - x_2)(x_1 - x_0)} + \frac{f_2(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Above equation can be written as

$$p_2(x) = f_0 l_0 + f_1 l_1 + f_2 l_2 = \sum_{i=0}^2 f_i l_i(x) \quad (2)$$

Where,

$$l_i(x) = \prod_{j=0, j \neq i}^2 \frac{(x - x_j)}{(x_i - x_j)}$$

Generalizing equation (2) for $n+1$ points, we get the relation

$$p_n(x) = \sum_{i=0}^n f_i l_i(x) \quad (3)$$

Where,

$$l_i(x) = \prod_{j=0, j \neq i}^n \frac{(x - x_j)}{(x_i - x_j)}$$

Equation (3) is called Lagrange interpolation formula.

Algorithm

1. Start
2. Read number of points, say n
3. Read the value at which value is needed, say x
4. Read given data points
5. Calculate values of L_i as below:
 for $i=1$ to n
 for $j=1$ to n
 if ($j \neq i$)
 $L[i] = L[i] * ((x - x[j]) / (x[i] - x[j]))$
 End if
 End for

- End for
6. Calculate interpolated value at x as below
 - For $i=1$ to n
 - $v=v + f_x[i]*Lx[i]$
 - End for
 7. Print the interpolation value v at x
 8. Terminate

Example

The upward velocity of a rocket is given as a function of time in Table below.

Time(t)	Velocity (v)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Determine the value of the velocity at $t = 16$ seconds using a first order Lagrange polynomial.

Solution

For first order polynomial interpolation (also called linear interpolation), the velocity is given by

$$p_1(x) = \sum_{i=0}^1 l_i(x) f_i = l_0(x) f_0 + l_1(x) f_1$$

Since we want to find the velocity at $t = 16$, and we are using a first order polynomial, we need to choose the two data points that are closest to $t = 16$ that also bracket $t = 16$. These two points are $t_0 = 15$ and $t_1 = 20$. This gives

$$l_0(x) = \prod_{\substack{j=0 \\ j \neq 0}}^1 \frac{x - x_j}{x_0 - x_j} = \frac{x - x_1}{x_0 - x_1} = \frac{16 - 20}{15 - 20} = \frac{4}{5} = 0.8$$

$$l_1(x) = \prod_{\substack{j=0 \\ j \neq 1}}^1 \frac{x - x_j}{x_1 - x_j} = \frac{x - x_0}{x_1 - x_0} = \frac{16 - 15}{20 - 15} = 0.2$$

Thus,

$$\begin{aligned} p_1(16) &= l_0(x) f_0 + l_1(x) f_1 \\ &= 0.8 \times 362.78 + 0.2 \times 517.35 = 393.69 \end{aligned}$$

Second Example

The value of e^x is given in Table below:

X	e^x
0	1
1	2.7183
2	7.3891
3	20.0855

Determine the value of $e^{1.2}$ by using second order polynomial interpolation using Lagrange polynomial interpolation.

Solution

For second order polynomial interpolation (also called quadratic interpolation), the polynomial is given by

$$p_2(x) = \sum_{i=0}^2 l_i(x)f_i = l_0(x)f_0 + l_1(x)f_1 + l_2(x)f_2$$

Since we want to find value of $e^{1.2}$, and we are using a Second order polynomial, we need to choose the three data points that are closest to $x=1.2$ that also bracket $x=1.2$. The three data points are $x_0 = 0$ and $x_1 = 1$ and $x_2 = 2$. This gives

$$l_0(x) = \prod_{\substack{j=0 \\ j \neq 0}}^2 \frac{x - x_j}{x_0 - x_j} = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(1.2 - 1)(1.2 - 2)}{(0 - 1)(0 - 2)} = \frac{0.2 * -0.8}{2} = -0.08$$

$$l_1(x) = \prod_{\substack{j=0 \\ j \neq 1}}^2 \frac{x - x_j}{x_1 - x_j} = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(1.2 - 0)(1.2 - 2)}{(1 - 0)(1 - 2)} = \frac{1.2 * -0.8}{1 * -1} = 0.96$$

$$l_2(x) = \prod_{\substack{j=0 \\ j \neq 2}}^2 \frac{x - x_j}{x_2 - x_j} = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(1.2 - 0)(1.2 - 1)}{(2 - 0)(2 - 1)} = \frac{1.2 * 0.2}{2 * 1} = 0.12$$

Now,

$$\begin{aligned} p_2(1.2) &= l_0(x)f_0 + l_1(x)f_1 + l_2(x)f_2 \\ &= -0.08 * 1 + 0.96 * 2.7183 + 0.12 * 7.3891 = 3.41626 \end{aligned}$$

//C Program for Lagrange Interpolation

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,j;
    float x,l,v=0,ax[10],fx[10],L[10];
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter the value of x\n");
```

```

        scanf("%f",&x);
        for(i=0;i<n;i++)
        {
            printf("Enter the value of x and fx at i=%d\n", i);
            scanf("%f%f",&ax[i],&fx[i]);
        }
        for(i=0;i<n;i++)
        {
            l=1.0;
            for(j=0;j<n;j++)
            {
                if (j!=i)
                    l=l*((x-ax[j])/(ax[i]-ax[j]));
            }
            L[i]=l;
        }
        for(i=0;i<n;i++)
        {
            v=v+fx[i]*L[i];
        }
        printf("Interpolated value=%f",v);
        getch();
        return 0;
    }
}

```

Output

Enter the number of points

4

Enter the value of x

1

Enter the value of x and fx at i=0

-1 -8

Enter the value of x and fx at i=1

0 3

Enter the value of x and fx at i=2

2 1

Enter the value of x and fx at i=3

3 12

Interpolation value=2.00

3.3 NEWTON'S DIVIDED DIFFERENCE INTERPOLATION

Let us consider a polynomial of degree n of the form given below:

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0)(x - x_1)\dots(x - x_{n-1}) \quad \dots(1)$$

To construct the polynomial we need to find coefficients $a_0, a_1, a_2, \dots, a_n$. Let us suppose $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ are given interpolating points. Now, at $x = x_0$ equation (1) becomes

$$\begin{aligned} p(x_0) &= f(x_0) = a_0 \\ \Rightarrow a_0 &= f(x_0) \end{aligned}$$

Similarly, at $x = x_1$ equation (1) becomes

$$\begin{aligned} p(x_1) &= f(x_1) = a_0 + a_1(x_1 - x_0) \\ \Rightarrow a_1 &= \frac{f(x_1) - a_0}{(x_1 - x_0)} = \frac{f(x_1) - f(x_0)}{(x_1 - x_0)} \end{aligned}$$

at $x = x_2$ equation (1) becomes

$$\begin{aligned} p(x_2) &= f(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ \Rightarrow f(x_2) &= f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \end{aligned}$$

Giving

$$a_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

Note that a_0, a_1 , and a_2 are finite divided differences. a_0, a_1 , and a_2 are the first, second, and third finite divided differences, respectively. We denote the first divided difference by

$$f[x_0] = f(x_0)$$

The second divided difference by

$$f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

and the third divided difference by

$$f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

This leads us to writing the general form of the Newton's divided difference polynomial for $n+1$ data points, $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_n, y_n)$, as

$$p_n(x) = a_0 + a_1(x - x_0) + \dots + a_n(x - x_0)(x - x_1)\dots(x - x_{n-1})$$

Where,

$$a_0 = f[x_0]$$

$$a_1 = f[x_1, x_0]$$

$$a_2 = f[x_2, x_1, x_0]$$

\vdots

$$a_{n-1} = f[x_{n-1}, x_{n-2}, \dots, x_0]$$

$$a_n = f[x_n, x_{n-1}, \dots, x_0]$$

Where the definition of the m^{th} divided difference is

$$a_m = f[x_m, \dots, x_0] \\ = \frac{f[x_m, \dots, x_1] - f[x_{m-1}, \dots, x_0]}{x_m - x_0}$$

Now equation (1) can be written as

$$p_n(x) = f[x_0] + \sum_{i=1}^n f[x_0, x_1, \dots, x_n] \prod_{j=0}^{i-1} (x - x_j) \quad (2)$$

This equation (2) is called Newton's divided difference interpolation polynomial.

Algorithm

1. Start
2. Read number of points, say n
3. Read the value at which interpolated value is needed, say x
4. Read given data points
5. Calculate First divided differences as
For $i=0$ to $n-1$
 $dd[i]=fx[i]$
End For
6. Calculate second to n^{th} divided differences as
For $i=0$ to $n-1$
 for $j = n-1$ to $i+1$
 $dd[j] = \frac{dd[j] - dd[j-1]}{x[j] - x[j-1-i]}$
 End For
End for
7. Set $v=0$ and $p=1$

8. Calculated interpolated value as

For $i=0$ to $n-1$

For $j=0$ to $i-1$

$$p = p * (x - x_j)$$

End For

$$v = v + dd[i] * p$$

Reset $p=1$

End For

9. Print the interpolated value v

10. Terminate

Example

The upward velocity of a rocket is given as a function of time in table below.

Time(t)	Velocity(v)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Determine the value of the velocity at $t=16$ seconds with third order polynomial using Newton's divided difference polynomial method.

Solution

For a third order polynomial, the velocity is given by

$$p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2)$$

Since we want to find the velocity at $t=16$, and we are using a third order polynomial, we need to choose the four data points that are closest to $x=16$ that also bracket $t=16$ to evaluate it. The four data points are $x_0 = 10$, $x_1 = 15$, $x_2 = 20$, and $x_3 = 22.5$.

Then

$$x_0 = 10, \quad f(x_0) = 227.04$$

$$x_1 = 15, \quad f(x_1) = 362.78$$

$$x_2 = 20, \quad f(x_2) = 517.35$$

$$x_3 = 22.5, \quad f(x_3) = 602.97$$

Now, calculate values of divided differences as below:

$$a_0 = f[x_0] = f(x_0) = 227.04$$

$$a_1 = f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{362.78 - 227.04}{15 - 10} = 27.148$$

$$a_2 = f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$

$$a_3 = f[x_3, x_2, x_1, x_0] = \frac{f[x_3, x_2, x_1] - f[x_2, x_1, x_0]}{x_3 - x_0}$$

We know that,

$$f[x_2, x_1] = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{517.35 - 362.78}{20 - 15} = 30.914$$

and

$$f[x_1, x_0] = 27.148$$

=>

$$a_2 = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0} = \frac{30.914 - 27.148}{20 - 10} = 0.37660$$

Again,

$$f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1}$$

We know that

$$f[x_3, x_2] = \frac{f(x_3) - f(x_2)}{x_3 - x_2} = \frac{602.97 - 517.35}{22.5 - 20} = 34.248$$

$$f[x_2, x_1] = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{517.35 - 362.78}{20 - 15} = 30.914$$

$$\Rightarrow f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1} = \frac{34.248 - 30.914}{22.5 - 15} = 0.44453$$

Therefore,

$$a_3 = \frac{f[x_3, x_2, x_1] - f[x_2, x_1, x_0]}{x_3 - x_0} = \frac{0.44453 - 0.37660}{22.5 - 10} = 5.4347 \times 10^{-3}$$

Now, Third order polynomial can be written as

$$\begin{aligned} p(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) \\ &= 227.04 + 27.148(x - 10) + 0.3766(x - 10)(x - 15) + 0.0055347(x - 10)(x - 15)(x - 20) \end{aligned}$$

At $x = 16$,

$$\begin{aligned} p(16) &= 227.04 + 27.148(16 - 10) + 0.3766(16 - 10)(16 - 15) + 0.0055347(16 - 10)(16 - 15)(16 - 20) \\ &= 392.06 \end{aligned}$$

//C Program for Newton's Divided Difference

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,j;
    float v=0,p,xv,x[10],fx[10],a[10];
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter the value of x\n");
    scanf("%f",&xv);
    for(i=0;i<n;i++)
    {
        printf("Enter the value of x and fx at i=%d\n", i);
        scanf("%f%f",&x[i],&fx[i]);
    }
    for(i=0;i<n;i++)
        a[i]=fx[i];
    for(i=0;i<n;i++)
    {
        for(j=n-1;j>i;j--)
            a[j]=(a[j]-a[j-1])/(x[j]-x[j-i]);
    }
    v=0;
    for(i=0;i<n;i++)
    {
        p=1;
        for(j=0;j<=i-1;j++)
            p=p*(xv-x[j]);
        v=v+a[i]*p;
    }
    printf("Interpolation value=%f",v);
    getch();
    return 0;
}
```

Output

Enter the number of points

4

Enter the value of x

1.3

Enter the value of x and fx at i=0

0.5 0.4794

Enter the value of x and fx at i=1

1 0.8415

Enter the value of x and fx at i=2

1.5 0.9975

Enter the value of x and fx at i=3

2.0 0.9093

Interpolation value=0.962270

3.3.1 Divided Difference Table

We know that first order divided differences can be calculated from given interpolating points as below:

$$f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$f[x_2, x_1] = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

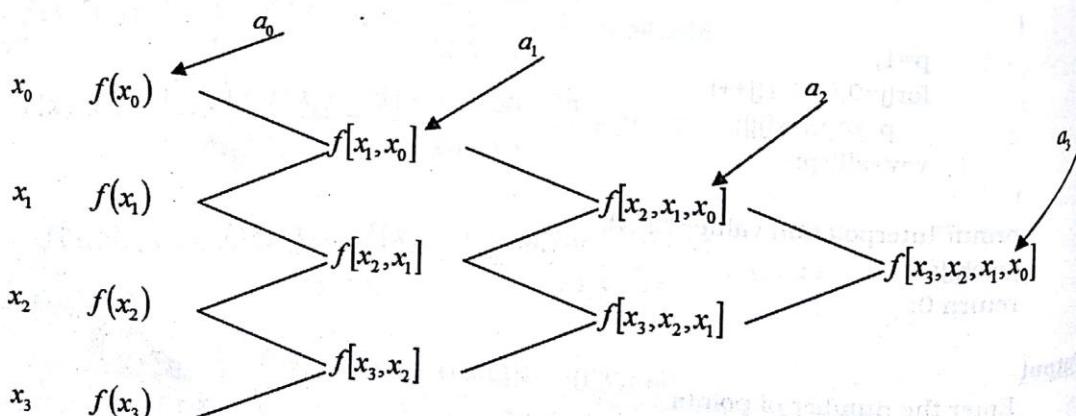
And second order divided difference can be calculated from first order divided differences as below:

$$f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$

Thus we can say that higher order divided differences can be calculated from lower order divided difference recursively by using the formula:

$$f[x_m, \dots, x_0] = \frac{f[x_m, \dots, x_1] - f[x_{m-1}, \dots, x_0]}{x_m - x_0}$$

Thus from above recursive formula divided difference table for cubic polynomial can be constructed as below:

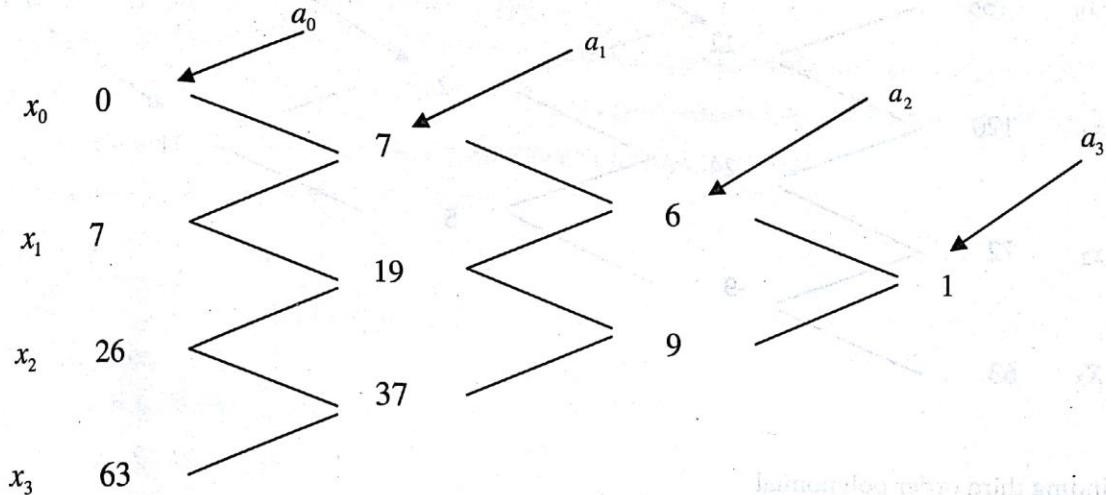


Example

Figure: Table of divided differences for a cubic polynomial.

Given the following data points, create the table of divided differences. Use the table to estimate the value of $f(1.8)$ by using second and third order polynomial.

x	1	2	3	4
$f(x)$	0	7	26	63

SolutionEvaluating $f(1.8)$ by using second order polynomial

We know that

$$p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

Therefore

$$f(1.8) = p_2(1.8) = 0 + 7 \times (1.8 - 1) + 6 \times (1.8 - 1)(1.8 - 2) = 0 + 5.6 - 0.96 = 4.64$$

Evaluating $f(1.8)$ by using third order polynomial

$$p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2)$$

Therefore

$$\begin{aligned} f(1.8) &= p_3(1.8) = 0 + 7 \times (1.8 - 1) + 6 \times (1.8 - 1)(1.8 - 2) + 1 \times (1.8 - 1)(1.8 - 2)(1.8 - 3) \\ &= 0 + 5.6 - 0.96 + 0.192 = 4.832 \end{aligned}$$

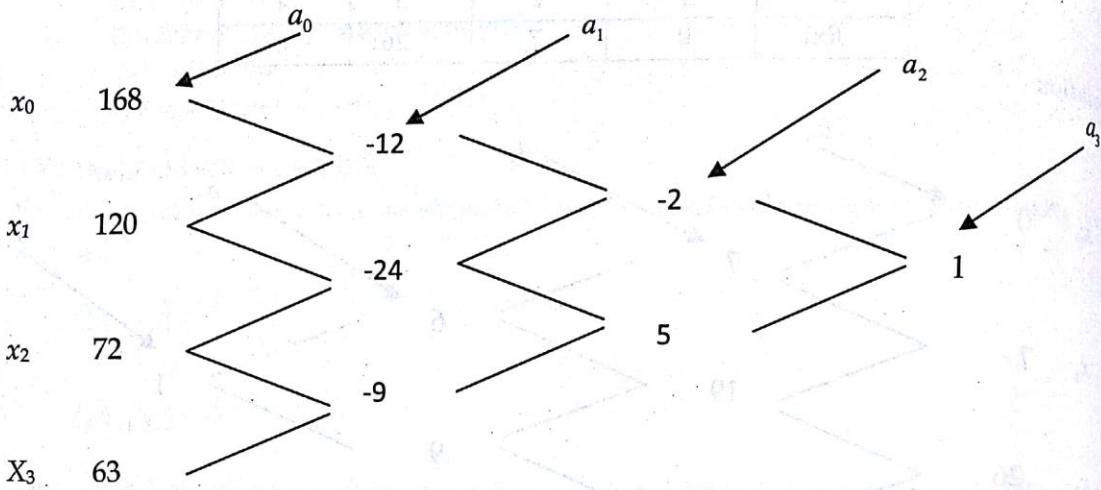
This example clearly shows that higher order polynomials give more accurate value than lower order polynomials

Second Example

Given the following data points, obtain the table of divided differences. Use the table to find third order Lagrange polynomial and use the polynomial to estimate the value of $f(8)$.

X	3	7	9	10
$f(x)$	168	120	72	63

Solution



Finding third order polynomial

We know that

$$p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2)$$

Therefore

$$\begin{aligned} f(x) &= p_3(x) = 168 + (-12) \times (x - 3) + (-2) \times (x - 3)(x - 7) + 1 \times (x - 3)(x - 7)(x - 9) \\ &= 168 - 12x + 36 - 2x^2 + 20x - 42 + x^3 - 19x^2 + 111x - 189 \\ &= x^3 - 21x^2 + 119x - 27 \end{aligned}$$

Thus, third order Lagrange polynomial is: $x^3 - 21x^2 + 119x - 27$

Evaluating $f(1.8)$ by using third order polynomial

$$f(x) = p_3(x) = x^3 - 21x^2 + 119x - 27$$

Put $x=8$

$$f(8) = p_3(8) = 8^3 - 21 \times 8^2 + 119 \times 8 - 27 = 93$$

3.4 NEWTON'S FORWARD AND BACKWARD DIFFERENCE INTERPOLATION

Newton's divided-difference formula can be expressed in a simplified form when x_0, x_1, \dots, x_n are arranged consecutively with equal spacing. Newton Forward and Backward interpolation formula is used only for equal intervals. Newton Forward interpolation formula is used mainly to interpolate the values of $f(x)$ near the beginning of a set of tabular values and also for extrapolating the value of $f(x)$ a short distance ahead of $f(x_0)$. This means, it is used to find the unknown values of y for some x which lies at the beginning of a set of tabular values. Newton Backward interpolation formula is used mainly to interpolate the values of $f(x)$ near the end of a set of tabular values and also for extrapolating the value of $f(x)$ at a short distance after $f(x_n)$.

This means, it is used to find the unknown values of $f(x)$ for some x which lies at the end of a set of tabular values.

3.4.1 Newton Forward-Difference interpolation

We know that Newton's divided difference polynomial is given by the formula

$$p_n(x) = f[x_0] + f[x_1, x_0](x - x_0) + \dots + f[x_n, x_{n-1}, \dots, x_1, x_0](x - x_0)(x - x_1)\dots(x - x_{n-1}).$$

Let us introduce the notation $h = x_{i+1} - x_i$, for each $i = 0, 1, \dots, n-1$

$$\text{Suppose, } x = x_0 + sh$$

$$\text{Since, } x_k = x_0 + kh$$

$$\Rightarrow x - x_k = (s - k)h$$

This gives,

$$x - x_0 = sh$$

$$x - x_1 = (s - 1)h$$

.....

.....

$$x - x_{n-1} = (s - n + 1)h$$

Now, Newton's interpolation divided difference formula becomes

$$\begin{aligned} p_n(x) &= p_n(x_0 + sh) = f[x_0] + f[x_0, x_1]sh + f[x_0, x_1, x_2]s(s-1)h^2 \\ &\quad + \dots + f[x_0, x_1, \dots, x_{n-1}, x_n]s(s-1)(s-2)\dots(s-n+1)h^n \\ &= \sum_{k=0}^n f[x_0, x_1, \dots, x_k]s(s-1)\dots(s-k+1)h^k \end{aligned}$$

Using binomial-coefficient notation,

$$\binom{s}{k} = \frac{s(s-1)\dots(s-k+1)}{k!}$$

We can express $p_n(x)$ compactly as

$$p_n(x) = p_n(x_0 + sh) = f[x_0] + \sum_{k=1}^n \binom{s}{k} k! h^k f[x_0, x_1, \dots, x_k] \quad (1)$$

Newton forward-difference formula is constructed by making use of the forward difference notation Δ . With this notation,

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{1}{h} \Delta f(x_0)$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{1}{2h} \left[\frac{1}{h} \Delta f(x_1) - \frac{1}{h} \Delta f(x_0) \right] = \frac{1}{2h^2} \Delta^2 f(x_0)$$

And, in general

$$f[x_0, x_1, \dots, x_k] = \frac{1}{k!h^k} \Delta^k f(x_0)$$

Now equation (1) can be written as

$$p_n(x) = p_n(x_0 + sh) = f[x_0] + \sum_{k=1}^n \binom{s}{k} \Delta^k f(x_0) \quad (2)$$

This equation is called Newton-Gregory forward difference formula

We can construct Newton's Forward differences by construction Newton's Forward difference table as below:

x_0	$f(x_0)$	$\Delta f(x_0)$		
x_1	$f(x_1)$		$\Delta^2 f(x_0)$	$\Delta^3 f(x_0)$
x_2	$f(x_2)$	$\Delta f(x_1)$		
x_3	$f(x_3)$	$\Delta f(x_2)$	$\Delta^2 f(x_1)$	

Figure Table of Forward differences for a cubic polynomial.

Algorithm for Newton's Forward Difference Interpolation

1. Start
2. Read number of data points, say n
3. Read the value at which interpolated value is needed, say x_p
4. Read n data points, say $x[i]$ and $f_x[i]$
5. Set $h=x[1]-x[0]$ and $s=(x_p-x[0])/h$
6. Calculate first forward differences as below
 For $i=0$ to $n-1$
 $fd[i]=fx[i]$
 End for
7. Calculate second to nth forward differences as below
 For $i=0$ to $n-1$
 For $j=n-1$ to $i+1$
 $fd[j] = fd[j] - fd[j-1]$
 End for
 End For
8. Set $v=fd[0]$ and Set $p=1$
9. Calculate interpolated value as below

For i=1 to n-1
 For k=1 to i
 $p = p * (s - k + 1)$

End For

$$v = v + \frac{fd[i] * p}{i!}$$

Reset p=1

End for

10. Print the interpolated value v at x_p

11. Terminate

Example

Construct Newton's forward-difference table for data points given in table below and then approximate the value of $f(1.1)$ by using Newton's forward difference formula and fourth divided difference.

x	1.0	1.3	1.6	1.9	2.2
$f(x)$	0.7651977	0.6200860	0.4554022	0.2818186	0.1103623

Solution

Here, $h = x_{i+1} - x_i = 0.3$

$$\text{Since, } x = x_0 + sh \Rightarrow s = \frac{x - x_0}{h} = \frac{1.1 - 1.0}{0.3} = \frac{1}{3}$$

x_i	$f(x_i)$	$\Delta f(x_0)$	$\Delta^2 f(x_0)$	$\Delta^3 f(x_0)$	$\Delta^4 f(x_0)$
1.0 <u>0.7651977</u>		-0.1451117			
1.3 0.6200860		-0.0195721			
1.6 0.4554022		-0.1646838	0.0106723		
1.9 0.2818186		-0.0088998		0.0003548	
2.2 0.1103623		-0.1735836		0.0110271	
		0.0021273			
		-0.1714563			

Now the Newton forward divided-difference formula can be used with the forward divided differences that have a solid underscore in above table.
 We know that,

$$P_n(x) = P_n(x_0 + sh) = f[x_0] + \sum_{k=1}^n \binom{s}{k} \Delta^k f(x_0)$$

Where,

$$\binom{s}{k} = \frac{s(s-1)\cdots(s-k+1)}{k!}$$

Thus,

$$\begin{aligned} p_4(x) &= f[x_0] + \binom{s}{1} \Delta f(x_0) + \binom{s}{2} \Delta^2 f(x_0) + \binom{s}{3} \Delta^3 f(x_0) + \binom{s}{4} \Delta^4 f(x_0) \\ &= f[x_0] + s \Delta f(x_0) + \frac{s(s-1)}{2!} \Delta^2 f(x_0) + \frac{s(s-1)(s-2)}{3!} \Delta^3 f(x_0) + \frac{s(s-1)(s-2)(s-3)}{4!} \Delta^4 f(x_0) \end{aligned}$$

Hence,

$$\begin{aligned} p_4(1.1) &= 0.7651977 + \frac{1}{3}(-0.1451117) + \frac{\frac{1}{3}x - \frac{2}{3}}{2}(-0.0195721) \\ &\quad + \frac{\frac{1}{3}x - \frac{2}{3}x - \frac{5}{3}}{6}(0.0106721) + \frac{\frac{1}{3}x - \frac{2}{3}x - \frac{5}{3}x - \frac{8}{3}}{24}(0.0003548) \\ &= 0.712 \end{aligned}$$

Second Example

Obtain the Newton's forward interpolating polynomial, $p_5(x)$ for the following tabular data and interpolate the value of the function at $x=0.0045$.

X	0	0.001	0.002	0.003	0.004	0.005
$f(x)$	1.121	1.123	1.1255	1.127	1.128	1.1285

Solution

$$\text{Here, } h = x_{i+1} - x_i = 0.001$$

$$\text{Since, } x = x_0 + sh \Rightarrow s = \frac{x - x_0}{h} = \frac{0.0045 - 0}{0.001} = 4.5$$

x_i	$f(x_i)$	$\Delta f(x_0)$	$\Delta^2 f(x_0)$	$\Delta^3 f(x_0)$	$\Delta^4 f(x_0)$	$\Delta^5 f(x_0)$
0	<u>1.121</u>					
0.001		<u>0.002</u>				
0.002			<u>0.0005</u>			
0.003				<u>-0.0015</u>		
0.004					<u>0.002</u>	
0.005						<u>-0.0025</u>

Now the Newton forward divided-difference formula can be used with the forward divided differences that have a solid underscore in above table.

We know that,

$$p_n(x) = p_n(x_0 + sh) = f[x_0] + \sum_{k=1}^n \binom{s}{k} \Delta^k f(x_0)$$

Where,

$$\binom{s}{k} = \frac{s(s-1)\cdots(s-k+1)}{k!}$$

Thus,

$$\begin{aligned} p_5(x) &= f[x_0] + \binom{s}{1} \Delta f(x_0) + \binom{s}{2} \Delta^2 f(x_0) + \binom{s}{3} \Delta^3 f(x_0) + \binom{s}{4} \Delta^4 f(x_0) + \binom{s}{5} \Delta^5 f(x_0) \\ &= f[x_0] + s \Delta f(x_0) + \frac{s(s-1)}{2!} \Delta^2 f(x_0) + \frac{s(s-1)(s-2)}{3!} \Delta^3 f(x_0) + \frac{s(s-1)(s-2)(s-3)}{4!} \Delta^4 f(x_0) + \\ &\quad \frac{s(s-1)(s-2)(s-3)(s-4)}{5!} \Delta^5 f(x_0) \end{aligned}$$

Hence,

$$\begin{aligned} p_5(x) = p_5(x_0 + sh) &= 1.121 + 0.002s + \frac{0.0005}{2}s(s-1) + \frac{-0.0015}{6}s(s-1)(s-2) + \\ &\quad \frac{0.002}{24}s(s-1)(s-2)(s-3) + \frac{-0.0025}{120}s(s-1)(s-2)(s-3)(s-4) \end{aligned}$$

Now, put $x=0.0045$

$$\begin{aligned} p_5(0.0045) = p_5(0 + 4.5 \times 0.001) &= 1.121 + 0.002 \times 4.5 + \frac{0.0005}{2} \times 4.5 \times 3.5 - \frac{0.0015}{6} \times 4.5 \times 3.5 \times 2.5 + \\ &\quad \frac{0.002}{24} \times 4.5 \times 3.5 \times 2.5 \times 1.5 - \frac{0.0025}{120} \times 4.5 \times 3.5 \times 2.5 \times 1.5 \times 0.5 = 1.1284 \end{aligned}$$

//C Program for Newton's Forward Difference Polynomial

```
#include<stdio.h>
#include<conio.h>
int fact(int n)
{
    if (n==1)
        return 1;
    else
        return(n*fact(n-1));
}
int main()
{
    int n,i,j,k;
```

82 Numerical Methods with Practical Approach

```
float v=0,p,x[10],fx[10],fd[10],h,s;
printf("Enter the number of points\n");
scanf("%d",&n);
printf("Enter the value at which interpolated value is needed\n");
scanf("%f",&xp);
for(i=0;i<n;i++)
{
    printf("Enter the value of x and fx at i=%d\n", i);
    scanf("%f%f",&x[i],&fx[i]);
}
h=x[1]-x[0];
s=(xp-x[0])/h;
for(i=0;i<n;i++)
{
    fd[i]=fx[i];
}
for(i=0;i<n;i++)
{
    for(j=n-1;j>i;j--)
    {
        fd[j]=(fd[j]-fd[j-1]);
    }
}
v=fd[0];

for(i=1;i<n;i++)
{
    p=1;
    for(k=1;k<=i;k++)
    {
        p=p*(s-k+1);
    }
    v=v+(fd[i]*p)/fact(i);
}
printf("Interpolation value=%f",v);
getch();
return 0;
}
```

Output

```
Enter the number of points
4
Enter the value at which interpolated value is needed
0.95
Enter the value of x and fx at i=0
0.9 0.7833
Enter the value of x and fx at i=1
```

1.0 0.8415

Enter the value of x and fx at i=2

1.1 0.8912

Enter the value of x and fx at i=3

1.2 0.9320

Interpolation value=0.813437

3.4.2 Newton Backward-Difference interpolation

If the interpolating nodes are reordered as x_n, x_{n-1}, \dots, x_0 , Newton's divided difference polynomial can be written as

$$p_n(x) = f[x_n] + f[x_n, x_{n-1}](x - x_n) + \dots + f[x_n, x_{n-1}, \dots, x_1, x_0](x - x_n)(x - x_{n-1}) \dots (x - x_1)$$

Let us introduce the notation $h = x_{i+1} - x_i$, for each $i = 0, 1, \dots, n-1$

$$\text{Suppose, } x = x_n + sh$$

$$\text{Since, } x_k = x_n + (k-n)h$$

$$\Rightarrow x - x_k = (s + n - k)h$$

This gives,

$$x - x_1 = (s + n - 1)h$$

$$x - x_2 = (s + n - 2)h$$

.....

$$x - x_{n-1} = (s + n - n + 1)h = (s + 1)h$$

$$x - x_n = (s + n - n)h = sh$$

Now, Newton's interpolation divided difference formula becomes

$$\begin{aligned} p_n(x) &= p_n(x_n + sh) = f[x_n] + f[x_n, x_{n-1}]sh + f[x_n, x_{n-1}, x_{n-2}]s(s+1)h^2 \\ &\quad \dots + f[x_n, x_{n-1}, \dots, x_2, x_1]s(s+1)\dots(s+n-2)(s+n-1)h^n \end{aligned}$$

Newton backward-difference formula is constructed by making use of the backward difference notation ∇ . With this notation,

$$f[x_n, x_{n-1}] = \frac{f[x_n] - f[x_{n-1}]}{x_n - x_{n-1}} = \frac{1}{h} \nabla f(x_n)$$

$$f[x_n, x_{n-1}, x_{n-2}] = \frac{f[x_n, x_{n-1}] - f[x_{n-1}, x_{n-2}]}{x_n - x_{n-2}} = \frac{1}{2h} \left[\frac{1}{h} \nabla f(x_n) - \frac{1}{h} \nabla f(x_{n-1}) \right] = \frac{1}{2h^2} \nabla^2 f(x_n)$$

And, in general

$$f[x_n, x_{n-1}, \dots, x_{n-k}] = \frac{1}{k!h^k} \nabla^k f(x_n)$$

Thus, interpolation divided difference formula can be written as

$$p_n(x) = p_n(x_n + sh) = f[x_n] + \nabla f(x_n)s + \frac{1}{2}\nabla^2 f(x_n)s(s+1) + \dots + \frac{1}{n!}\nabla^n f(x_n)s(s+1)\dots(s+n-1)$$

This equation is called Newton-Gregory backward difference formula

Example

Construct Newton's backward-difference table for data points given in table below and then approximate the value of $\sin(45)$ by using Newton's backward difference formula and fourth divided difference

$\sin(x)$	10	20	30	40	50
$f(x)$	0.1736	0.3420	0.5000	0.6428	0.7660

Solution

Here, $h = x_{i+1} - x_i = 10$

$$\text{Since, } x = x_n + sh \Rightarrow s = \frac{x - x_n}{h} = \frac{45 - 50}{10} = -\frac{1}{2}$$

x_i	$f(x_i)$	$\nabla f(x_n)$	$\nabla^2 f(x_n)$	$\nabla^3 f(x_n)$	$\nabla^4 f(x_n)$
10	0.1736				
20	0.3420	0.1684			
30	0.5000	-0.0104	0.1580	0.0048	
40	0.6428		-0.0152	-0.0004	0.1428
50	<u>0.7660</u>		<u>-0.0196</u>	<u>0.0044</u>	<u>0.1232</u>

We know that backward divided difference interpolation formula is given by

$$p_n(x) = p_n(x_n + sh) = f[x_n] + \nabla f(x_n)s + \frac{1}{2}\nabla^2 f(x_n)s(s+1) + \dots + \frac{1}{n!}\nabla^n f(x_n)s(s+1)\dots(s+n-1)$$

Thus

$$p_4(x) = p_4(x_n + sh) = f[x_n] + \nabla f(x_n)s + \frac{1}{2!}\nabla^2 f(x_n)s(s+1) + \frac{1}{3!}\nabla^3 f(x_n)s(s+1)(s+2) + \frac{1}{4!}\nabla^4 f(x_n)s(s+1)(s+2)(s+3)$$

Now, put $x=45$

$$\begin{aligned}
 p_4(45) &= p_4(50 - \frac{1}{2} \times 10) = 0.766 + 0.1232 \times (-\frac{1}{2}) + \frac{1}{2} \times (-0.0196) \times (-\frac{1}{2}) \times \frac{1}{2} + \\
 &\quad \frac{1}{6} \times 0.0044 \times (-\frac{1}{2}) \times \frac{1}{2} \times \frac{3}{2} + \frac{1}{24} (-0.0004) \times (-\frac{1}{2}) \times \frac{1}{2} \times \frac{3}{2} \times \frac{5}{2} = 0.70659
 \end{aligned}$$

Hence,

$$\sin(45) = p_4(45) = 0.70659$$

Second Example

The sale for the last five years is given in the table below. Find 4th order polynomial that passes through the given data and then use the polynomial to estimate the sales for the year 1979.

Year-x	1974	1976	1978	1980	1982
Sales-f(x)	40	43	48	52	57

Solution

$$\text{Here, } h = x_{i+1} - x_i = 2$$

$$\text{Since, } x = x_n + sh \Rightarrow s = \frac{x - x_n}{h} = \frac{1979 - 1982}{2} = -\frac{3}{2}$$

x_i	$f(x_i)$	$\nabla f(x_n)$	$\nabla^2 f(x_n)$	$\nabla^3 f(x_n)$	$\nabla^4 f(x_n)$
1974	40				
		3			
1976	43		2		
			5	-3	
1978	48			-1	5
			4	2	
1980	52			1	
			5		
1982	57				

We know that backward divided difference interpolation formula is given by

$$P_n(x) = p_n(x_n + sh) = f[x_n] + \nabla f(x_n)s + \frac{1}{2!} \nabla^2 f(x_n)s(s+1) + \dots + \frac{1}{n!} \nabla^n f(x_n)s(s+1)\dots(s+n-1)$$

Thus

$$\begin{aligned}
 p_4(x) &= p_4(x_n + sh) = f[x_n] + \nabla f(x_n)s + \frac{1}{2!} \nabla^2 f(x_n)s(s+1) + \frac{1}{3!} \nabla^3 f(x_n)s(s+1)(s+2) + \\
 &\quad \frac{1}{4!} \nabla^4 f(x_n)s(s+1)(s+2)(s+3)
 \end{aligned}$$

Thus, fourth order polynomial is

$$p_4(x) = p_4(x_n + sh) = 57 + 5s + \frac{1}{2}1 \times s(s+1) + \frac{1}{6}2s(s+1)(s+2) + \\ \frac{1}{24}5s(s+1)(s+2)(s+3)$$

Now, put $x=1979$

$$p_4(1979) = p_4(1982 + \left(-\frac{3}{2}\right) \times 2) = 57 + 5 \times \left(-\frac{3}{2}\right) + \frac{1}{2} \times \left(-\frac{3}{2}\right) \times \left(-\frac{1}{2}\right) + \frac{1}{6} \times 2 \times \left(-\frac{3}{2}\right) \times \left(-\frac{1}{2}\right) \times \left(\frac{1}{2}\right) + \\ \frac{1}{24} \times 5 \times \left(-\frac{3}{2}\right) \times \left(-\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(\frac{3}{2}\right) \\ = 57 - 7.5 + 0.375 + 0.125 + 0.1172 = 50.1172$$

Hence,

Estimated sales of year 1979 = 50.1172

Algorithm

1. Start
2. Read number of points, say n
3. Enter the value at which interpolated value is required, say x_p
4. Read n data points
5. Set $h=x[1]-x[0]$ and $s=(x_p-x[n-1])/h$
6. Calculate first backward differences as below:
 For $i=0$ to $n-1$
 $bd[i]=fx[i]$
 End for
7. Calculate 2nd to n^{th} backward differences as below:
 For $i=n-1$ to 1
 For $j=0$ to $i-1$
 $bd[j] = bd[j+1] - bd[j]$
 End for
8. Set $v=bd[n-1]$
9. Calculate interpolated value as below
 For $i=1$ to $n-1$
 For $k=1$ to i
 $p = p * (s+k-1)$
 End for
 $v = v + \frac{bd[n-i-1]*p}{i!}$
- Reset $p=1$
- End for

10. Print Interpolated value v at xp
11. Terminate

//C Program for Newton's Backward Difference Interpolation

```
#include<stdio.h>
#include<conio.h>
int fact(int n)
{
    if (n==1)
        return 1;
    else
        return(n*fact(n-1));
}
int main()
{
    int n,i,j,k;
    float v=0,p,xp,x[10],fx[10],bd[10],h,s;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter the value of x\n");
    scanf("%f",&xp);
    for(i=0;i<n;i++)
    {
        printf("Enter the value of x and fx at i=%d\n", i);
        scanf("%f%f",&x[i],&fx[i]);
    }
    h=x[1]-x[0];
    s=(xp-x[n-1])/h;
    for(i=0;i<n;i++)
        bd[i]=fx[i];
    for(i=n-1;i>0;i--)
    {
        for(j=0;j<i;j++)
            bd[j]=(bd[j+1]-bd[j]);
    }
    v=bd[n-1];
    for(i=1;i<n;i++)
    {
        p=1;
        for(k=1;k<=i;k++)
        {
            p=p*(s+k-1);
        }
        v=v+(bd[n-i]*p)/fact(i);
    }
    printf("Interpolation value=%f",v);
}
```

```

    getch();
    return 0;
}

```

Output

```

Enter the number of points
4
Enter the value at which Interpolated value is required
1.15
Enter the value of x and fx at i=0
0.9 0.7833
Enter the value of x and fx at i=1
1.0 0.8415
Enter the value of x and fx at i=2
1.1 0.8912
Enter the value of x and fx at i=3
1.2 0.9320
Interpolation value=0.912737

```

3.5 SPLINE INTERPOLATION

Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called spline. Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. Spline interpolation avoids the problem of Runge's phenomenon. Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree. This was shown by Runge when he interpolated data based on a simple function of

$$y = \frac{1}{1 + 25x^2}$$

On an interval of $[-1, 1]$. For example, take six equidistantly spaced points in $[-1, 1]$ and find y at these points as given in table below.

x	-1.0	-0.6	-0.2	0.2	0.6	1.0
$y = 1/(1 + 25x^2)$	0.038461	0.1	0.5	0.5	0.1	0.038461

Now through these six points, one can pass a fifth order polynomial

$$p_5(x) = 3.1378 \times 10^{-11} x^5 + 1.2019 x^4 - 3.3651 \times 10^{-11} x^3 - 1.7308 x^2 + 1.0004 \times 10^{-11} x + 5.6731 \times 10^{-1},$$

$$-1 \leq x \leq 1$$

On plotting the fifth order polynomial (See figure below) and the original function, one can see that the two do not match well. One may consider choosing more points in the interval $[-1, 1]$ to get a better match, but it diverges even more. Thus to information from more data points and also to keep the function true to the data behavior, spline interpolation is used.

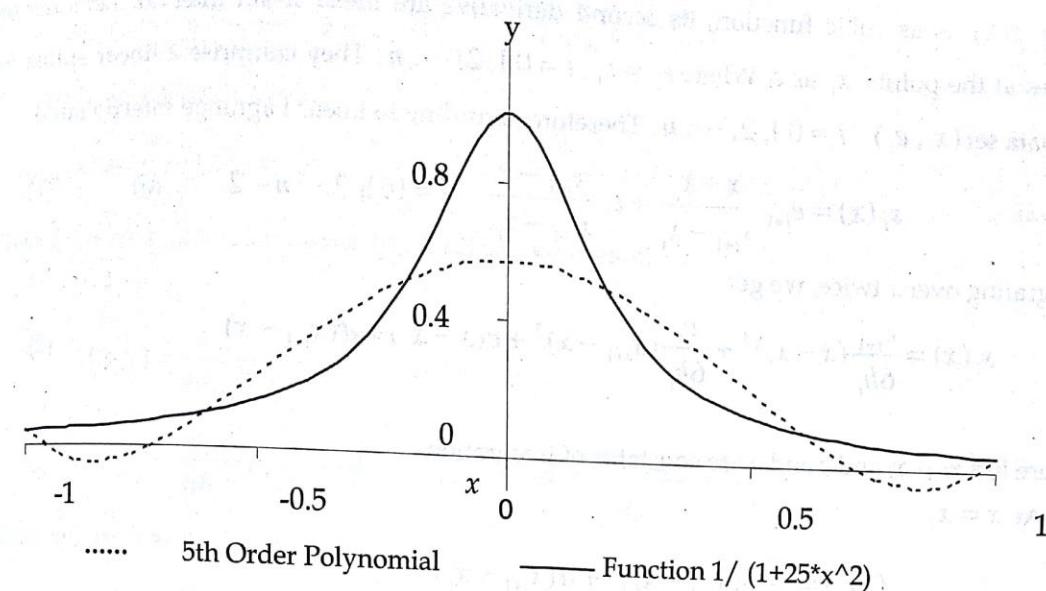


Figure: 3.2: Runge's Phenomenon

3.5.1 Natural Cubic Splines

Let $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ are $n+1$ given data points. A function s is called a spline of degree k on $x_0 < x_1 < x_2 < \dots < x_n$ if

- 1) $s \in [x_0, x_n]$,
- 2) $s^{(j)}, j = 0, 1, 2, \dots, k-1$ are all continuous functions on $[x_0, x_n]$ where $s^{(j)}$ is the j th derivative;
- 3) s is a polynomial of degree $\leq k$ on each interval $[x_i, x_{i+1}]$.

If $k = 3$, the spline is called a cubic spline. A cubic spline must satisfy the following conditions

1. $s(x)$ must interpolate all the points $x_0, x_1, x_2, \dots, x_n$. i.e, for each i

$$s(x_i) = f(x_i)$$

2. The function values at interior points must be equal.

$$s_i(x_i) = s_{i-1}(x_i)$$

3. The first derivatives at the interior points must be equal.

$$s'_i(x_i) = s'_{i-1}(x_i)$$

4. The second derivatives at the interior points must be equal.

$$s''_i(x_i) = s''_{i-1}(x_i)$$

5. The second derivatives at the end points are zero.

$$s''(x_0) = s''(x_n) = 0$$

Since $s(x)$ is a cubic function, its second derivative are linear in an interval. Let's denote values at the points x_i as e_i , where $e_i = s''_i, i = 0, 1, 2, \dots, n$. They comprise a linear spline for the data set $(x_i, e_i) \quad i = 0, 1, 2, \dots, n$. Therefore according to linear Lagrange interpolation

$$s_i''(x) = e_{i+1} \frac{x - x_i}{x_{i+1} - x_i} + e_i \frac{x_{i+1} - x}{x_{i+1} - x_i}, \quad i = 0, 1, 2, \dots, n-2 \quad (1)$$

Integrating over x twice, we get

$$s_i(x) = \frac{e_{i+1}}{6h_i} (x - x_i)^3 + \frac{e_i}{6h_i} (x_{i+1} - x)^3 + c(x - x_i) + d(x_{i+1} - x) \quad (2)$$

Where $h_i = x_{i+1} - x_i$ and c and d are constants of integration.

At $x = x_i$

$$\begin{aligned} f(x_i) &= \frac{e_i}{6h_i} (x_{i+1} - x_i)^3 + d(x_{i+1} - x_i) \\ \Rightarrow f(x_i) - \frac{e_i}{6h_i} h_i^3 &= dh_i \quad \Rightarrow d = \frac{f(x_i)}{h_i} - \frac{e_i h_i}{6} \end{aligned}$$

Similarly, at $x = x_{i+1}$

$$\begin{aligned} f(x_{i+1}) &= \frac{e_{i+1}}{6h_i} (x_{i+1} - x_i)^3 + c(x_{i+1} - x_i) \\ \Rightarrow f(x_{i+1}) - \frac{e_{i+1}}{6h_i} h_i^3 &= ch_i \quad \Rightarrow c = \frac{f(x_{i+1})}{h_i} - \frac{e_{i+1} h_i}{6} \end{aligned}$$

Which leads to,

$$s_i(x) = \frac{e_{i+1}}{6h_i} (x - x_i)^3 + \frac{e_i}{6h_i} (x_{i+1} - x)^3 + \left(\frac{f(x_{i+1})}{h_i} - \frac{e_{i+1} h_i}{6} \right) (x - x_i) + \left(\frac{f(x_i)}{h_i} - \frac{e_i h_i}{6} \right) (x_{i+1} - x) \quad (3)$$

Differentiating the above equation, we get

$$s'_i(x) = \frac{e_{i+1}}{2h_i} (x - x_i)^2 - \frac{e_i}{2h_i} (x_{i+1} - x)^2 + \left(\frac{f(x_{i+1})}{h_i} - \frac{e_{i+1} h_i}{6} \right) - \left(\frac{f(x_i)}{h_i} - \frac{e_i h_i}{6} \right)$$

and

$$s'_i(x_i) = -\frac{e_i}{2h_i} h_i^2 + \left(\frac{f(x_{i+1})}{h_i} - \frac{e_{i+1} h_i}{6} \right) - \left(\frac{f(x_i)}{h_i} - \frac{e_i h_i}{6} \right) = -\frac{e_{i+1} h_i}{6} - \frac{e_i h_i}{3} + b_i$$

Where

$$b_i = \frac{f(x_{i+1}) - f(x_i)}{h_i}, \quad i = 0, 1, \dots, n-2 \quad (4)$$

Similarly,

$$s''_{i-1}(x) = e_i \frac{x - x_{i-1}}{x_i - x_{i-1}} + e_{i-1} \frac{x_i - x}{x_i - x_{i-1}}, \quad i = 1, 2, \dots, n \quad (1)$$

Integrating over x twice, we get

$$s_{i-1}(x) = \frac{e_i}{6h_{i-1}}(x - x_{i-1})^3 + \frac{e_{i-1}}{6h_{i-1}}(x_i - x)^3 + c(x - x_{i-1}) + d(x_i - x) \quad (2)$$

Where $h_{i-1} = x_i - x_{i-1}$ and c and d are constants of integration.

At $x = x_{i-1}$

$$\begin{aligned} f(x_{i-1}) &= \frac{e_{i-1}}{6h_{i-1}}(x_i - x_{i-1})^3 + d(x_i - x_{i-1}) \\ \Rightarrow f(x_{i-1}) - \frac{e_{i-1}}{6h_{i-1}}h_{i-1}^3 &= dh_{i-1} \quad \Rightarrow \quad d = \frac{f(x_{i-1}) - \frac{e_{i-1}h_{i-1}}{6}}{h_{i-1}} \end{aligned}$$

Similarly, at $x = x_i$

$$\begin{aligned} f(x_i) &= \frac{e_i}{6h_{i-1}}(x_i - x_{i-1})^3 + c(x_i - x_{i-1}) \\ \Rightarrow f(x_i) - \frac{e_i}{6h_{i-1}}h_{i-1}^3 &= ch_{i-1} \quad \Rightarrow \quad c = \frac{f(x_i) - \frac{e_ih_{i-1}}{6}}{h_{i-1}} \end{aligned}$$

This leads to,

$$s_{i-1}(x) = \frac{e_i}{6h_{i-1}}(x - x_{i-1})^3 + \frac{e_{i-1}}{6h_{i-1}}(x_i - x)^3 + \left(\frac{f(x_i)}{h_{i-1}} - \frac{e_ih_{i-1}}{6}\right)(x - x_{i-1}) + \left(\frac{f(x_{i-1})}{h_{i-1}} - \frac{e_{i-1}h_{i-1}}{6}\right)(x_i - x) \quad (3)$$

Differentiating the above equation, we get

$$s'_{i-1}(x) = \frac{e_i}{2h_{i-1}}(x - x_{i-1})^2 - \frac{e_{i-1}}{2h_{i-1}}(x_i - x)^2 + \left(\frac{f(x_i)}{h_{i-1}} - \frac{e_ih_{i-1}}{6}\right) - \left(\frac{f(x_{i-1})}{h_{i-1}} - \frac{e_{i-1}h_{i-1}}{6}\right)$$

and

$$s'_{i-1}(x_i) = \frac{e_i}{2h_{i-1}}h_{i-1}^2 + \left(\frac{f(x_i)}{h_{i-1}} - \frac{e_ih_{i-1}}{6}\right) - \left(\frac{f(x_{i-1})}{h_{i-1}} - \frac{e_{i-1}h_{i-1}}{6}\right) = \frac{e_{i-1}h_{i-1}}{6} + \frac{e_ih_{i-1}}{3} + b_{i-1}$$

Where

$$b_{i-1} = \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}, \quad i = 1, 2, \dots, n-1$$

Since,

$$s'_i(x_i) = s'_{i-1}(x_i)$$

$$\Rightarrow -\frac{e_{i+1}h_i}{6} - \frac{e_ih_i}{3} + b_i = \frac{e_{i-1}h_{i-1}}{6} + \frac{e_ih_{i-1}}{3} + b_{i-1}$$

$$\Rightarrow -e_{i+1}h_i - 2e_ih_i + 6b_i = e_{i-1}h_{i-1} + 2e_ih_{i-1} + 6b_{i-1}$$

$$\Rightarrow 6b_i - 6b_{i-1} = e_{i+1}h_i + 2e_ih_i + e_{i-1}h_{i-1} + 2e_ih_{i-1}$$

$$\Rightarrow 6(b_i - b_{i-1}) = e_{i-1}h_{i-1} + 2e_i(h_{i-1} + h_i) + e_{i+1}h_i, \quad i = 1, 2, \dots, n-1$$

Which is a tri-diagonal system of equations in terms of the unknowns e_i 's. There are n unknowns and $n-2$ equations. Two additional conditions are needed to determine the cubic splines. When they are $e_0 = e_n = 0$, the resulting splines are called the *natural* cubic splines.

These equations can be rearranged as

$$e_0 = 0$$

$$h_{i-1}e_{i-1} + u_i e_i + h_i e_{i+1} = v_i \quad i = 1, 2, 3, \dots, n-1$$

$$e_n = 0$$

Where,

$$u_i = 2(h_{i-1} + h_i) \quad i = 1, 2, 3, \dots, n-1$$

$$v_i = 6(b_i - b_{i-1})$$

Above system of $n-1$ equations can be written in $(n-1) \times (n-1)$ matrix form as below:

$$\begin{bmatrix} u_1 & h_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ h_1 & u_2 & h_2 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & h_{n-3} & u_{n-2} & h_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & h_{n-2} & u_{n-1} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{n-2} \\ e_{n-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}$$

Algorithm for Cubic Spline Interpolation

1. Start
2. Read the number of data points, say n
3. Read the point at which value is required, say x_p
4. Read n data points, say $x[i]$ and $f[x][i]$
5. Calculate values of h and b as below:
For $i=0$ to $n-1$

$$h[i] = x[i+1] - x[i]$$

$$b[i] = \frac{f[x][i+1] - f[x][i]}{h[i]}$$

End for

6. Calculate coefficients u and v as below
For $i=1$ to $n-1$

$$u[i] = 2(h[i-1] + h[i])$$

$$v[i] = 6(b[i] - b[i-1])$$

End for

7. Construct matrix of order $(n-1) \times (n-1)$ by using values computed in step 5 & 6
8. Read RHS vector
10. Use Gauss-Elimination method to find value of $e_i, i = 1, 2, \dots, n-1$
11. Calculate the interpolated value at x_p (x_p lies between x_i and x_{i+1}) by using following formula
$$v = \frac{e[i+1]}{6h[i]}(x_p - x[i])^3 + \frac{e[i]}{6h[i]}(x[i+1] - x_p)^3 + \left(\frac{fx[i+1]}{h[i]} - \frac{e[i+1]h[i]}{6} \right)(x_p - x[i]) + \left(\frac{fx[i]}{h[i]} - \frac{e[i]h[i]}{6} \right)(x[i+1] - x_p)$$
12. Print the interpolated value v
13. Terminate

Example

Find all possible cubic splines for the following data set and then approximate the value of $f(6)$.

X	0	5	7	8	10
$f(x)$	0	2	-1	-2	20

Solution

To calculate the 2nd derivatives, first determine

$$h_0 = 5, h_1 = 2, h_2 = 1, h_3 = 2$$

$$b_0 = 0.4, b_1 = -1.5, b_2 = -1, b_3 = 11$$

Using above we can calculate u_i 's and v_i 's as

$$u_1 = 14, u_2 = 6, u_3 = 6$$

$$v_1 = -11.4, v_2 = 3, v_3 = 72$$

$$h_0 e_0 + u_1 e_1 + h_1 e_2 = v_1 \Rightarrow 14e_1 + 2e_2 = -11.4$$

$$h_1 e_1 + u_2 e_2 + h_2 e_3 = v_2 \Rightarrow 2e_1 + 6e_2 + e_3 = 3$$

$$h_2 e_2 + u_3 e_3 + h_3 e_4 = v_3 \Rightarrow e_2 + 6e_3 = 72$$

Then from equation, we get

$$\begin{pmatrix} 14 & 2 & 0 \\ 2 & 6 & 1 \\ 0 & 1 & 6 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} -11.4 \\ 3 \\ 72 \end{pmatrix}$$

The values of e_1, e_2 , and e_3 can be obtained by using the tri-diagonal solver. These values along with $e_0 = e_4 = 0$ can then be used to determine the splines. After solving above equations, we get

$$e_1 = -0.6245$$

$$e_2 = -1.3288$$

$$e_3 = 12.2214$$

Now,

$$s_0(x) = \frac{e_1}{6h_0}(x-x_0)^3 + \frac{e_0}{6h_0}(x_1-x)^3 + \left(\frac{f(x_1)}{h_0} - \frac{e_1 h_0}{6}\right)(x-x_0) + \left(\frac{f(x_0)}{h_0} - \frac{e_0 h_0}{6}\right)(x_1-x)$$

$$\Rightarrow s_0(x) = \frac{-0.6245}{6*5}(x-0)^3 + \left(\frac{2}{5} - \frac{-0.6245*5}{6}\right)(x-0)$$

$$\Rightarrow s_0(x) = -0.0208x^3 + (0.4 + 0.5204)x = -0.0208x^3 + 0.9204x \quad x \in [0, 5]$$

$$s_1(x) = \frac{e_2}{6h_1}(x-x_1)^3 + \frac{e_1}{6h_1}(x_2-x)^3 + \left(\frac{f(x_2)}{h_1} - \frac{e_2 h_1}{6}\right)(x-x_1) + \left(\frac{f(x_1)}{h_1} - \frac{e_1 h_1}{6}\right)(x_2-x)$$

$$\Rightarrow s_1(x) = \frac{-1.3288}{6*2}(x-5)^3 + \frac{-0.6245}{6*2}(7-x)^3 + \left(-\frac{1}{2} - \frac{-1.3288*2}{6}\right)(x-5) + \left(\frac{2}{2} - \frac{-0.6245*2}{6}\right)(7-x)$$

$$\Rightarrow s_1(x) = -0.1107(x^3 - 15x^2 + 75x - 125) - 0.0520(343 - 147x + 21x^2 - x^3) - 0.0570(x-5) + 1.208(7-x)$$

$$\Rightarrow s_1(x) = -0.0587x^3 + 0.5685x^2 - 1.9235x + 4.7425 \quad x \in [5, 7]$$

$$s_2(x) = \frac{e_3}{6h_2}(x-x_2)^3 + \frac{e_2}{6h_2}(x_3-x)^3 + \left(\frac{f(x_3)}{h_2} - \frac{e_3 h_2}{6}\right)(x-x_2) + \left(\frac{f(x_2)}{h_2} - \frac{e_2 h_2}{6}\right)(x_3-x)$$

$$\Rightarrow s_2(x) = \frac{12.2214}{6*1}(x-7)^3 + \frac{-1.3288}{6*1}(8-x)^3 + \left(-\frac{2}{1} - \frac{12.2214*1}{6}\right)(x-7) + \left(\frac{-1}{1} - \frac{-1.3288*1}{6}\right)(8-x)$$

$$\Rightarrow s_2(x) = 2.0369(x^3 - 21x^2 + 147x - 343) - 0.2215(512 - 192x + 24x^2 - x^3) - 4.0369(x-7) - 0.7785(8-x)$$

$$\Rightarrow s_2(x) = 2.2584x^3 - 48.0909x^2 + 338.6939x - 790.0344 \quad x \in [7, 8]$$

$$s_3(x) = \frac{e_4}{6h_3}(x-x_3)^3 + \frac{e_3}{6h_3}(x_4-x)^3 + \left(\frac{f(x_4)}{h_3} - \frac{e_4 h_3}{6}\right)(x-x_3) + \left(\frac{f(x_3)}{h_3} - \frac{e_3 h_3}{6}\right)(x_4-x)$$

$$\Rightarrow s_3(x) = \frac{12.2214}{6*2}(10-x)^3 + \frac{20}{2}(x-8) + \left(-\frac{2}{2} - \frac{12.2214*2}{6}\right)(10-x)$$

$$\Rightarrow s_3(x) = 1.01845(1000 - 300x + 30x^2 - x^3) + 10(x-8) - 5.0738(10-x)$$

$$\Rightarrow s_3(x) = -1.01845x^3 + 30.5535x^2 - 290.4612x + 887.712 \quad x \in [8, 10]$$

Now, Value of $f(6)$ can be estimated by using the cubic spline $s_1(x)$ because 6 lies the interval $[5, 7]$.

$$s_1(x) = -0.0587x^3 + 0.5685x^2 - 1.9235x + 4.7425 \quad x \in [5, 7]$$

$$\Rightarrow s_1(6) = f(6) = -0.0587x^3 + 0.5685x^2 - 1.9235x + 4.7425 = 0.9883$$

//C program for Spline Interpolation

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k,xp;
    float h[10],x[10],fx[10], u[10],v[10], b[10], m[10][10],r[10],e[10];
    float val,pivot,sum;
    printf("Enter Number of Points:\n");
}
```

```

scanf("%d",&n);
printf("Enter Sample points\n");
for(i=0;i<n;i++)
    scanf("%f%f",&x[i],&fx[i]);
printf("Enter Interpolation Point\n");
scanf("%d",&xp);
for(i=0;i<n-1;i++)
{
    h[i]=x[i+1]-x[i];
    b[i]=(fx[i+1]-fx[i])/h[i];
}
for(i=1;i<=n-1;i++)
{
    u[i]=2*(h[i-1]+h[i]);
    v[i]=6*(b[i]-b[i-1]);
}
for(i=1;i<n-1;i++)
{ //Construction of matrix in LHS
    m[i][i]=u[i];
    if(i!=1)
    {
        m[i][i-1]=h[i-1];
        m[i-1][i]=h[i-1];
    }
    r[i]=v[i]; //RHS Vector
}

for (k=1;k<n-2;k++) //Forward Elimination
{
    pivot=m[k][k];
    if (pivot<0.000001)
        printf("Method failed\n");
    else
        for(i=k+1;i<n-1;i++)
        {
            for(j=1;j<n-1;j++)
            {
                m[i][j]=m[i][j]-m[k][j]*m[i][k]/pivot;
            }
            r[i]=r[i]-r[k]*m[i][k]/pivot;
        }
    e[n-2]=r[n-2]/m[n-2][n-2];
}

```

```

for(i=n-3;i>=1;i--)
{
    sum=0;
    for(j=i+1;j<n-1;j++)
    {
        sum=sum+m[i][j]*e[j];
    }
    e[i]=(r[i]-sum)/m[i][i];
}

for(i=0;i<=n-1;i++) //Locating interval in which interpolation point lies
if(xp<=x[i])
    break;
i=i-1;

//calculation of interpolation Value
val=(e[i+1]/(6*h[i])*xp-x[i])*(xp-x[i])*(xp-x[i])+(e[i]/(6*h[i])*x[i+1]-xp)*(x[i+1]-xp)*(x[i+1]-xp))+(((fx[i+1]/h[i])-e[i+1]*h[i]/6))*(xp-x[i])+(((fx[i]/h[i])-e[i]*h[i]/6)*(x[i+1]-xp));

printf("Interpolated Value=%f\n",val);
getch();
}

```

Output

```

Enter Number of Points:
3
Enter Sample points
4   2
9   3
16  4
Enter Interpolation Point
7
Interpolated Value=2.622857

```

EXERCISE

1. Differentiate interpolation and extrapolation. Why polynomials are suitable for interpolation function?
2. Derive formula for Lagrange interpolation and then write down algorithm to calculate interpolated value at non-tabular point using it.
3. Can we use Newton divided differences with data that is not equi-spaced? Derive formula for Newton divided difference interpolation polynomial.
4. When it is suitable to used Newton forward and backward difference interpolation? Write down algorithm for each

5. Does using interpolating polynomial of higher degree achieve more correct results? If yes, justify your answer with counter example.
6. What is meant by spline? Define spline of degree k. explain about cubit splines also derive its formula.
7. What are divided differences? State the second order Newton's second order divided difference interpolation polynomial.
8. How divided difference table are useful? Construct a general divided difference table for four data points.
9. Using Lagrange's Interpolation Formula find $y(10)$ from the following table

X	5	6	9	11
Y	12	13	14	16

10. Use Lagrange Formula to fit a polynomial to the data given below & find y at $x = 1$.

X	-1	0	2	3
Y	-8	3	1	12

11. Using Lagrange's Interpolation Formula find $y(10)$ from the following table

X	1	2	5
Y	1	4	10

12. Find the cubic Lagrange Interpolation polynomial fitting the points $y(1) = -3$, $y(3) = 0$, $y(4) = 30$, $y(6) = 132$. Hence find $y(5)$.

13. Estimate the values of $y(0.95)$ using the Newton forward difference interpolation formula

X	0.9	1.0	1.1	1.2
$f(x) = \sin x$	0.7833	0.8415	0.8912	0.9320

14. Estimate the values of $y(1.8)$ using the Newton backward difference interpolation formula

X	0.5	1	1.5	2.0
Y	0.4794	0.8415	0.9975	0.9093

15. Develop cubic splines for the data given below and then predict $f(1.5)$

X	0	1	2	3
$f(x)$	1	-1	-1	0

16. Given the data points below, estimate the function value at $x=1.5$ using cubic splines

X	0	1	2	3
$f(x)$	1	3	4	7

□□□

Chapter 4

REGRESSION ANALYSIS

Chapter Content

- Introduction
- Least Squares Method
- Linear Regression Model
- Nonlinear Models for Regression

4.1 INTRODUCTION

4.1.1 Concept of Regression

Regression analysis is a form of predictive modeling technique which investigates the relationship between a dependent and independent variables. Dependent variables are also called target variable and independent variables are also called predictors. The goal of regression analysis is to express the dependent variable as a function of the independent variables. This technique is used for forecasting, time series modeling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression. Thus, regression analysis is an important tool for modeling and analyzing data. Here, we fit a curve or line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized. Once a regression analysis relationship is obtained, it can be used to predict values of the dependent variable. The duality of fit and the accuracy of prediction made depend upon the data used. Hence non-representative or improperly compiled data result in poor fits and predictions.

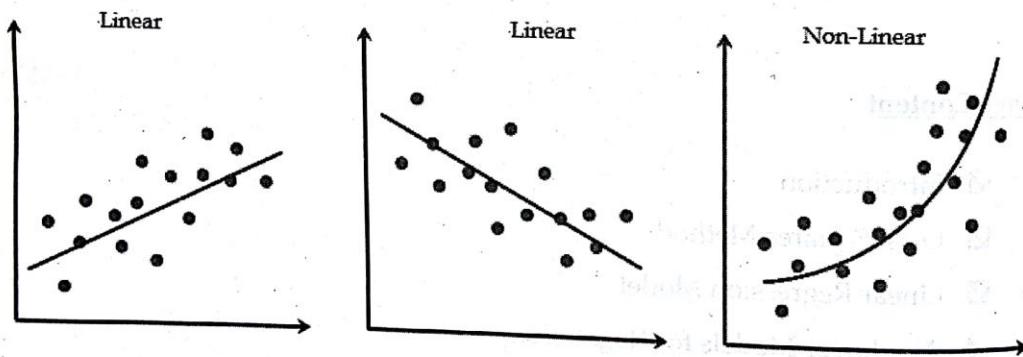


Figure: Linear and Non-linear Regression

4.1.2 Interpolation vs. Regression

In interpolation we are given with some data points, and we are supposed to find a curve which fits the input/output relationship perfectly. In case of interpolation, we don't have to worry about variance of the fitted curve. That means given the set of n data points (x_i, y_i) we look for the function $f(x)$ that satisfies the relation $y_i = f(x_i)$ for all given data points.

When we do regression, we look for a function that minimizes some cost, usually sum of squares of errors. We don't require the function to have the exact values at given points; we just want a good approximation. In general, we search for the function $y = f(x)$ that might not satisfy the relation $y_i = f(x_i)$ for any given data points, but the cost function $\sum (f(x_i) - y_i)^2$ will be the smallest possible of all the functions of given form.

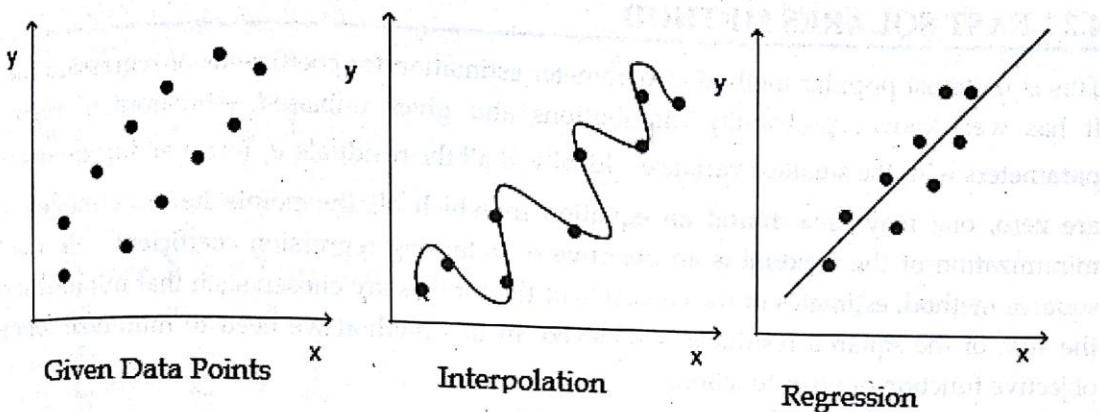


Figure: Interpolation vs. Regression

4.1.3 Parameter Estimation Methods

Parameter estimation is the process of estimating values of parameters of the model, based on the observed pairs of values and minimizing certain objective function. For example, consider the following variables and parameters:

Dependent Variable= y

Independent Variable= x

Parameters= a, b

Dependent variable is linear with parameters. That is, $y = ax + b$

Now, parameter estimation is used to find values of regression constants or parameters a and b by using given data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ such that some objective function or error function is minimized. A measure of goodness of fit of regression model is the magnitude of the residual (error), e_i at each of the n data points.

$$e_i = y_i - f(x_i), i = 1, 2, \dots, n$$

Some methods of parameter estimation are as below:

1. Minimize the sum of errors
i.e, minimize $\sum e_i = \sum y_i - f(x_i), i = 1, 2, \dots, n$
2. Minimize the sum of absolute values of errors
i.e, minimize $\sum |e_i| = \sum |y_i - f(x_i)|, i = 1, 2, \dots, n$
3. Minimize the sum of square of errors (Least square regression),
i.e, minimize $\sum e_i^2 = \sum (y_i - f(x_i))^2, i = 1, 2, \dots, n$

Usually first two methods are not used for parameter estimation because they do not yield unique line through given data set. The reason behind this is constants of the model can be chosen such that the average residual is zero without making individual residuals small. But the third method will yield unbiased parameters with the smallest variance.

4.2 LEAST SQUARES METHOD

This is the most popular method of parameter estimation for coefficients of regression models. It has well known probability distributions and gives unbiased estimators of regression parameters with the smallest variance. Ideally, if all the residuals e_i (error at data points (x_i, y_i)) are zero, one may have found an equation in which all the points lie on a model. Thus, minimization of the residual is an objective of obtaining regression coefficients. In the least squares method, estimates of the constants of the models are chosen such that minimization of the sum of the squared residuals is achieved. In this method we need to minimize following objective function or error function.

$$\sum e_i^2 = \sum (y_i - f(x_i))^2, i = 1, 2, \dots, n$$

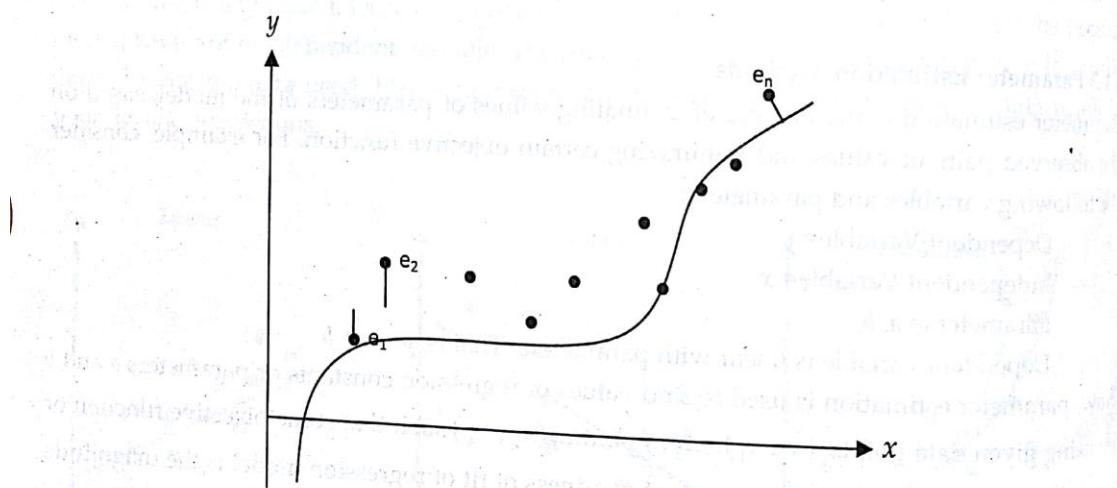


Figure: Errors at Different Data Points

4.3 LINEAR REGRESSION

Fitting a straight line is simplest approach of regression analysis, which is called linear regression. A Straight line can be represented by using the mathematical equation: $y = f(x) = a + bx$, where a and b are regression coefficients to be determined. Let the sum of squares of individual errors can be expressed as

$$E = \sum e_i^2 = \sum_{i=1}^n (y_i - f(x_i))^2 = \sum_{i=1}^n (y_i - a - bx_i)^2$$

We should choose regression coefficient a and b such that E is minimum. Necessary condition for E to be minimum is

$$\Rightarrow \frac{\partial E}{\partial a} = 0 \text{ and } \frac{\partial E}{\partial b} = 0$$

$$\frac{\partial E}{\partial a} = 2 \sum_{i=1}^n (y_i - a - bx_i)(-1) = 0 \text{ and}$$

$$\frac{\partial E}{\partial b} = 2 \sum_{i=1}^n (y_i - a - bx_i)(-x_i) = 0$$

Simplifying above equations, we get

$$-\sum_{i=1}^n y_i + \sum_{i=1}^n a + \sum_{i=1}^n bx_i = 0 \quad \text{and}$$

$$-\sum_{i=1}^n y_i x_i + \sum_{i=1}^n ax_i + \sum_{i=1}^n bx_i^2 = 0$$

Since,

$$\sum_{i=1}^n a = a + a + \dots + a = na$$

Above equations can be written as

$$na + b \sum_{i=1}^n x_i = \sum_{i=1}^n y_i \quad (1)$$

and

$$a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i \quad (2)$$

Solving the above Equations (1) and (2) gives

$$b = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

$$a = \frac{\sum_{i=1}^n y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n} = \bar{y} - b \bar{x}$$

$$\text{Where, } \bar{y} = \frac{\sum_{i=1}^n y_i}{n} \quad \text{and} \quad \bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

Algorithm

1. Start
2. Read number of points, say n
3. Read give data points, say x[i] and y[i]
4. Find summations of x, y, xy, and x² as below
for i=0 to n-1

```

sx=sx+x[i]
sy=sy+y[i]
sxy=sxy+x[i]*y[i],
sx2=sx2+x[i]*x[i]

```

End for

5. Calculate values of parameters as below:
 $b = ((n*sxy) - (sx*sy)) / ((n*sx2) - (sx*sx))$ and
 $a = (sy/n) - (b*sx/n)$
6. Display the equation $ax+b$
7. Terminate

Example

Fit a straight line that best fits the following set of data by using linear regression.

x	1	2	3	4	5
$f(x)$	3	5	7	10	12

Solution

x_i	y_i	x_i^2	$x_i y_i$
1	3	1	3
2	5	4	10
3	7	9	21
4	10	16	40
5	12	25	60
$\sum x_i = 15$	$\sum y_i = 37$	$\sum x_i^2 = 55$	$\sum x_i y_i = 134$

Now,

$$b = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2} = \frac{5 * 134 - 15 * 37}{5 * 55 - 15^2} = \frac{115}{50} = 2.3$$

$$a = \frac{\sum_{i=1}^n y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n} = \frac{37}{5} - 2.3 \frac{15}{5} = 7.4 - 2.3 * 3 = 0.5$$

Thus the straight line that best fits through given data points is: $y = 2.3x + 0.5$

Second Example

The values of y and their corresponding values of y are shown in the table below.

x	0	1	2	3	4
$y=f(x)$	2	3	5	4	6

- a) Find the least square regression line $y = ax + b$
- b) Estimate the value of y when $x = 10$.

Solution

x_i	y_i	x_i^2	$x_i y_i$
0	2	0	0
1	3	1	3
2	5	4	10
3	4	9	12
4	6	16	24
$\sum x_i = 10$	$\sum y_i = 20$	$\sum x_i^2 = 30$	$\sum x_i y_i = 49$

Now,

$$b = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - \left(\sum x_i \right)^2} = \frac{5 \times 49 - 10 \times 20}{5 \times 30 - 10^2} = \frac{45}{50} = 0.9$$

$$a = \frac{\sum y_i}{n} - b \frac{\sum x_i}{n} = \frac{20}{5} - 0.9 \times \frac{10}{5} = 4 - 1.8 = 2.2$$

Thus the least square regression line that best fits through given data points is:

$$y = 0.9x + 2.2 \quad (1)$$

Now put $x=10$ in above equation, we get

$$y = 0.9 \times 10 + 2.2 = 11.2$$

/*C program for Linear Regression

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,j,k;
    float a=0,b=0,x[10],y[10],sx=0,sy=0,sxy=0,sx2=0;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter the value of x and fx\n");
    for(i=0;i<n;i++)
    {
        scanf("%f%f",&x[i],&y[i]);
    }
    for(i=0;i<n;i++)
    {
        sx=sx+x[i];
        sy=sy+y[i];
    }
}
```

```

sxy=sxy+x[i]*y[i];
sx2=sx2+x[i]*x[i];
}
b=((n*sxy) - (sx*sy))/((n*sx2) - (sx*sx));
a=(sy/n) - (b*sx/n);
printf("Fitted line is:%f + %f x",a,b);
getch();
}

```

Output

Enter the number of points

4

Enter the value of x and fx

0	-1
2	5
5	12
7	20

Fitted line is:-1.138 + 2.897 x

4.4 NONLINEAR REGRESSION

Nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. Nonlinear regression model can be described in two ways

1. By Fitting Exponential Model
2. By Fitting Polynomial Model

4.4.1 Fitting Exponential Model

An exponential model is a model described by the equation: $y = ae^x$. In this equation the coefficients a and b are the constants of the exponential model. Nonlinear regression model can be obtained by fitting exponential through the given data points $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)$. When exponential model is fitted through the given data points, the residual at each data point x_i is given the equation given below:

$$e_i = y_i - ae^{bx_i} \quad (2)$$

And the sum of the square of the residuals is

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - ae^{bx_i})^2 \quad (3)$$

To find the constants a and b of the exponential model, we minimize E by differentiating with respect to a and b and equating the resulting equations to zero.

$$\frac{\partial E}{\partial a} = \sum_{i=1}^n 2(y_i - ae^{bx_i})(-e^{bx_i}) = 0$$

$$\frac{\partial E}{\partial b} = \sum_{i=1}^n 2(y_i - ae^{bx_i})(-ax_i e^{bx_i}) = 0$$

\Rightarrow

$$-\sum_{i=1}^n y_i e^{bx_i} + a \sum_{i=1}^n e^{2bx_i} = 0 \quad (4)$$

$$\sum_{i=1}^n y_i x_i e^{bx_i} - a \sum_{i=1}^n x_i e^{2bx_i} = 0 \quad (5)$$

From Equation (4), a can be written explicitly in terms of b as below:

$$a = \frac{\sum_{i=1}^n y_i e^{bx_i}}{\sum_{i=1}^n e^{2bx_i}} \quad (6)$$

Substituting Equation (6) in (5) gives

$$\sum_{i=1}^n y_i x_i e^{bx_i} - \frac{\sum_{i=1}^n y_i e^{bx_i}}{\sum_{i=1}^n e^{2bx_i}} \sum_{i=1}^n x_i e^{2bx_i} = 0 \quad (7)$$

This equation is still a nonlinear equation in b and can be solved best by numerical methods such as the bisection method or the secant method to find the value of b . By using equation (6) the value of a can be calculated.

Example

Below is given the relative intensity of radiation as a function of time.

$x(\text{time})$	0	1	3	5	7	9
$y(\text{intensity of radiation})$	1.0	0.891	0.708	0.562	0.447	0.355

If the level of the relative intensity of radiation is related to time via an exponential formula $y = ae^{bx}$, find the value of the regression constants a and b .

Solution

The value of b is given by solving the nonlinear Equation

$$f(b) = \sum_{i=1}^n y_i x_i e^{bx_i} - \frac{\sum_{i=1}^n y_i e^{bx_i}}{\sum_{i=1}^n e^{2bx_i}} \sum_{i=1}^n x_i e^{2bx_i} = 0 \quad (1)$$

and then the value of a is evaluated from

$$a = \frac{\sum_{i=1}^n y_i e^{bx_i}}{\sum_{i=1}^n e^{2bx_i}} \quad (2)$$

Equation (1) can be solved for b using bisection method. To estimate the initial guesses, we assume $b = -0.120$ and $b = -0.110$. We need to check whether these values first bracket the root of $f(b) = 0$. At $b = -0.120$

i	x_i	y_i	$y_i x_i e^{bx_i}$	$y_i e^{bx_i}$	e^{2bx_i}	$x_i e^{2bx_i}$
1	0	1	0.00000	1.00000	1.00000	0.00000
2	1	0.891	0.79205	0.79205	0.78663	0.78663
3	3	0.708	1.4819	0.49395	0.48675	1.4603
4	5	0.562	1.5422	0.30843	0.30119	1.5060
5	7	0.447	1.3508	0.19297	0.18637	1.3046
6	9	0.355	1.0850	0.12056	0.11533	1.0379
$n=6$			$\sum = 6.2501$	$\sum = 2.9062$	$\sum = 2.8763$	$\sum = 6.0954$

Now,

$$f(-0.120) = (6.2501) - \frac{2.9062}{2.8763} (6.0954) \\ = 0.091357$$

Similarly

$$f(-0.110) = -0.10099$$

Since

$$f(-0.120) \times f(-0.110) < 0,$$

the value of λ falls in the bracket of $[-0.120, -0.110]$. The next guess of the root then is

$$b = \frac{-0.120 + (-0.110)}{2} \\ = -0.115$$

Continuing with the bisection method, the root of $f(\lambda) = 0$ is found as $b = \lambda = -0.1150^3$. From Equation (2), a can be calculated as:

$$a = \frac{\sum_{i=1}^6 y_i e^{bx_i}}{\sum_{i=1}^6 e^{2bx_i}} = \frac{2.9373}{2.9378} = 0.99983$$

thus the regression formula is given by

$$y = 0.99983 e^{-0.11508x}$$

4.4.2 Fitting Exponential Model by Linearization

From above example it is clear that when we fit exponential model through given data set, iterative methods are required to estimate the values of the model parameters. This can be time consuming and tedious. The most common approach used to estimate such model is linearization, which can be done easily by calculating logarithms of both sides.

We know that, exponential model is given by

$$y = ae^{bx} \quad (1)$$

Taking natural log of both sides gives

$$\log y = \log(ae^{bx})$$

\Rightarrow

$$\log y = \log a + bx \quad (2)$$

This equation is similar to the form of linear equation $y = a + bx$. Thus we can evaluate parameters a and b by using the equation of linear regression model as below:

$$b = \frac{n \sum_{i=1}^n x_i \log y_i - \sum_{i=1}^n x_i \sum_{i=1}^n \log y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2} \quad (3)$$

And

$$\log a = \frac{\sum_{i=1}^n \log y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n} \quad (4)$$

$$\text{Let, } R = \frac{\sum_{i=1}^n \log y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n}$$

Now, equation (4) becomes

$$\log a = R$$

Taking antilog on both side we get,

$$a = e^R$$

(6)

Now, we can calculate values of regression coefficients a and b easily by using equations (3) and (6).

Algorithm Nonlinear Regression with Exponential Model

1. Start
2. Read number of data points, say n
3. Read give data points, say $x[i], y[i]$
4. Calculate needed summations as below:
 for $i=0$ to $n-1$
 $sx=sx+x[i]$
 $slgy=slgy+log(y[i])$
 $sxy=sxy+x[i]*log(y[i])$
 $sx2=sx2+x[i]*x[i]$
 End for
5. Calculate values of b and a by using following formulae
 $a=((n*sxy) - (sx*slgy))/((n*sx2) - (sx*sx))$
 $r=(slgy/n) - (b*sx/n)$
 $a=e^r$
6. Display the equation ae^{bx}
7. Terminate

Example

Below is given the relative intensity of radiation as a function of time.

$x(\text{time})$	0	1	3	5	7	9
$y=\text{intensity of radiation}$	1.00	0.891	0.708	0.562	0.447	0.355

If the level of the relative intensity of radiation is related to time via an exponential formula $y = ae^{bx}$, find the value of the regression constants a and b . Also estimate intensity of radiation at time $t=13$.

Solution

We know that exponential model is given by

$$y = ae^{bx}$$

=>

$$\log y = \log a + bx$$

This equation is similar in form to the linear equation $y = a + bx$. Thus we can evaluate parameters a and b by using the equation of linear regression model as below:

$$b = \frac{n \sum_{i=1}^n x_i \log y_i - \sum_{i=1}^n x_i \sum_{i=1}^n \log y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

And

$$\log a = \frac{\sum_{i=1}^n \log y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n}$$

Now calculate required summations as below:

i	x_i	y_i	$\log y_i$	$x_i \log y_i$	x_i^2
1	0	1	0.00000	0.00000	0.00
2	1	0.891	-0.11541	-0.1154	1.00
3	3	0.708	-0.34531	-1.0359	9.00
4	5	0.562	-0.57625	-2.8813	25.0
5	7	0.447	-0.80520	-5.6364	49.0
6	9	0.355	-1.0356	-9.3207	81.0
$n=6$	$\sum x_i = 25$		$\sum \log y_i = -2.878$	$\sum x_i \log y_i = -18.99$	$\sum x_i^2 = 165$

Now,

$$b = \frac{6 \times (-18.99) - 25 \times (-2.878)}{6 \times 165 - 25^2} = \frac{-41.99}{365} = -0.115$$

$$\log a = \frac{-2.878}{6} - (-0.115) \frac{25}{6} = -0.4796 + 0.4791 = -0.0005$$

Since

$$\log a = -0.0005$$

$$\Rightarrow a = e^{-0.0005} = 0.999$$

Thus the regression formula is

$$y = 0.999 \times e^{-0.115x}$$

Now, put $x=13$ in above equation, we get

$$y = 0.999 \times e^{-0.115 \times 13} = 0.224$$

Thus, intensity of radiation at time $t=13$ is 0.224.

Second Example

Fit the curve $y = ae^{bx}$ through the data given below.

X	-4	-2	0	1	2	4
$y=f(x)$	0.57	1.32	4.12	6.65	11	30.3

Solution

We know that exponential model is given by

$$y = ae^{bx}$$

$$\Rightarrow \log y = \log a + bx$$

This equation is similar in form to the linear equation $y = a + bx$. Thus we can evaluate parameters a and b by using the equation of linear regression model as below:

$$b = \frac{n \sum_{i=1}^n x_i \log y_i - \sum_{i=1}^n x_i \sum_{i=1}^n \log y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

And

$$\log a = \frac{\sum_{i=1}^n \log y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n}$$

Now, calculate required summations as below

i	x_i	y_i	$\log y_i$	$x_i \log y_i$	x_i^2
1	-4	0.57	-0.562	2.248	14
2	-2	1.32	0.277	-0.555	4
3	0	4.12	1.415	0	0
4	1	6.65	1.894	1.894	1
5	2	11.0	2.398	4.796	4
6	4	30.3	3.411	13.645	16
$n=6$	$\sum x_i = 1$		$\sum \log y_i = 8.835$	$\sum x_i \log y_i = 22.03$	$\sum x_i^2 = 41$

Now,

$$b = \frac{6 \times 22.03 - 1 \times 8.835}{6 \times 41 - 1} = \frac{123.345}{245} = 0.503$$

$$\log a = \frac{\sum_{i=1}^n \log y_i}{n} - b \frac{\sum_{i=1}^n x_i}{n}$$

$$\log a = \frac{8.835}{6} - 0.503 \times \frac{1}{6} = 1.472 - 0.084 = 1.388$$

Since

$$\log a = 1.388$$

$$\Rightarrow a = e^{1.388} = 4.006$$

Thus the regression formula then is

$$y = 4.006e^{0.503x}$$

//C program for exponential regression model (using linearization)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float a=0,b=0,r,x[10],y[10],sx=0,slgy=0,sxy=0,sx2=0;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter the value of x and fx");
    for(i=0;i<n;i++)
    {
        scanf("%f%f",&x[i],&y[i]);
    }
    for(i=0;i<n;i++)
    {
        sx=sx+x[i];
        slgy=slgy+log(y[i]);
        sxy=sxy+x[i]*log(y[i]);
        sx2=sx2+x[i]*x[i];
    }
    b=((n*sxy) - (sx*slgy))/((n*sx2) - (sx*sx));
    r=(slgy/n) - (b*sx/n);
    a=exp(r);
    printf("Fitted curve is:y=%fe^%fx",a,b);
    getch();
}
```

Output

Enter the number of points

5
Enter the value of x and fx
2 4.077
4 11.084
6 30.128
8 81.897
10 222.62

Fitted curve is: $y=1.4999e^{0.5x}$

4.4.3 Fitting Polynomial Models

Polynomial regression is a form of regression analysis in which the relationship between the independent x and the dependent variable y is modeled as an n^{th} degree polynomial in x . Let n data points are $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and we want to fit an m^{th} order polynomial through the given data points. General form of polynomial of degree m is given below:

$$y = a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m, m < n \quad (1)$$

Now, the residual at each data point is given by

$$e_i = y_i - a_0 - a_1 x_i - \dots - a_m x_i^m \quad (2)$$

The sum of the square of the residuals is given by

$$\begin{aligned} E &= \sum_{i=1}^n e_i^2 \\ &= \sum_{i=1}^n (y_i - a_0 - a_1 x_i - \dots - a_m x_i^m)^2 \end{aligned} \quad (3)$$

To find the constants of the polynomial regression model, we equate the derivatives with respect to a_i to zero as below:

$$\frac{\partial E}{\partial a_0} = \sum_{i=1}^n 2(y_i - a_0 - a_1 x_i - \dots - a_m x_i^m)(-1) = 0$$

$$\frac{\partial E}{\partial a_1} = \sum_{i=1}^n 2(y_i - a_0 - a_1 x_i - \dots - a_m x_i^m)(-x_i) = 0$$

.....

.....

$$\frac{\partial E}{\partial a_m} = \sum_{i=1}^n 2(y_i - a_0 - a_1 x_i - \dots - a_m x_i^m)(-x_i^m) = 0$$

This gives,

$$na_0 + a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 + \dots + a_m \sum_{i=1}^n x_i^m = \sum_{i=1}^n y_i$$

$$a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 + \dots + a_m \sum_{i=1}^n x_i^{m+1} = \sum_{i=1}^n x_i y_i$$

.....

$$a_0 \sum_{i=1}^n x_i^m + a_1 \sum_{i=1}^n x_i^{m+1} + a_2 \sum_{i=1}^n x_i^{m+2} + \dots + a_m \sum_{i=1}^n x_i^{2m} = \sum_{i=1}^n x_i^m y_i$$

Setting those equations in matrix form gives

$$\begin{bmatrix} n & \left(\sum_{i=1}^n x_i\right) & \dots & \left(\sum_{i=1}^n x_i^m\right) \\ \left(\sum_{i=1}^n x_i\right) & \left(\sum_{i=1}^n x_i^2\right) & \dots & \left(\sum_{i=1}^n x_i^{m+1}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(\sum_{i=1}^n x_i^m\right) & \left(\sum_{i=1}^n x_i^{m+1}\right) & \dots & \left(\sum_{i=1}^n x_i^{2m}\right) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \vdots \\ \sum_{i=1}^n x_i^m y_i \end{bmatrix} \quad (5)$$

The above system can be solved for a_0, a_1, \dots, a_m

Algorithm for Polynomial Regression

1. Start
2. Read number of points, say n
3. Read order of polynomial, say m
4. Read given data points, say $x[i]$ and $fx[i]$
5. Calculate required summations as below
for $i=1$ to $2m$
 for $j=0$ to $n-1$
 $sx[i] = sx[i] + pow(x[j], i)$
 End for
End for
for $i=0$ to m
 for $j=0$ to $n-1$
 $sxy[i] = y[j] * pow(x[j], i)$
 End for
End for
6. Construct RHS matrix of order $(m+1) \times (m+1)$
7. Construct LHS matrix of order $(m+1) \times 1$
8. Solve for coefficients a_0, a_1, \dots, a_m using gauss elimination method
9. Display the equation $y = a_0 + a_1x + a_2x^2 + \dots + a_mx^m$
10. Terminate

Example

Fit the quadratic curve through the following data and estimate value of y at $x=12$.

X	1	3	4	5	6	7	8	9	10
$f(x)$	2	7	8	10	11	11	10	9	8

Solution

General form of quadratic equation is $y = a_0 + a_1x + a_2x^2$. The coefficients a_0, a_1, a_2 of the quadratic equation can be found as from following metrics

$$\begin{bmatrix} n & \left(\sum_{i=1}^n x_i \right) & \left(\sum_{i=1}^n x_i^2 \right) \\ \left(\sum_{i=1}^n x_i \right) & \left(\sum_{i=1}^n x_i^2 \right) & \left(\sum_{i=1}^n x_i^3 \right) \\ \left(\sum_{i=1}^n x_i^2 \right) & \left(\sum_{i=1}^n x_i^3 \right) & \left(\sum_{i=1}^n x_i^4 \right) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \end{bmatrix}$$

Now calculate required summations as below

i	x	y	x^2	x^3	x^4	$x \times y$	$x^2 \times y$
1	1	2	1	1	1	2	2
2	3	7	9	27	81	21	63
3	4	8	16	64	256	32	128
4	5	10	25	125	625	50	250
5	6	11	36	216	1296	66	396
6	7	11	49	343	2401	77	539
7	8	10	64	512	4096	80	640
8	9	9	81	729	6561	81	729
9	10	8	100	1000	10000	80	800
	$\Sigma x = 53$	$\Sigma y = 76$	$\Sigma x^2 = 381$	$\Sigma x^3 = 3017$	$\Sigma x^4 = 25317$	$\Sigma xy = 489$	$\Sigma x^2y = 3547$

Thus we have following matrix

$$\begin{bmatrix} 9 & 53 & 381 \\ 53 & 381 & 3017 \\ 381 & 3017 & 25317 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 76 \\ 489 \\ 3547 \end{bmatrix}$$

Solving the above system of simultaneous linear equations, we get

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1.46 \\ 3.605 \\ -0.268 \end{bmatrix}$$

Thus the quadratic curve that passes through given data points is: $y = -1.46 + 3.605x - 0.268x^2$

Now, put $x=12$ in above equation, we get

$$y = -1.46 + 3.605 \times 12 - 0.268 \times 12^2 = 3.208$$

Example

Below is given the coefficient of thermal expansion as a function of temperature.

x = Temperature ($^{\circ}$ F)	y = Coefficient of thermal expansion
80	6.47×10^{-6}
40	6.24×10^{-6}
-40	5.72×10^{-6}
-120	5.09×10^{-6}
-200	4.30×10^{-6}
-280	3.33×10^{-6}
-340	2.45×10^{-6}

Fit the above data to $y = a_0 + a_1x + a_2x^2$

Solution

Since $y = a_0 + a_1x + a_2x^2$ is the quadratic relationship between the thermal expansion coefficient and the temperature, the coefficients a_0, a_1, a_2 are found as follows

$$\begin{bmatrix} n & \left(\sum_{i=1}^n x_i \right) & \left(\sum_{i=1}^n x_i^2 \right) & \left(\sum_{i=1}^n x_i^3 \right) & \left(\sum_{i=1}^n x_i^4 \right) \\ \left(\sum_{i=1}^n x_i \right) & \left(\sum_{i=1}^n x_i^2 \right) & \left(\sum_{i=1}^n x_i^3 \right) & \left(\sum_{i=1}^n x_i^4 \right) & \left(\sum_{i=1}^n x_i^5 \right) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \end{bmatrix}$$

18 Numerical Methods with Practical Approach

Now, calculate required summations as below

i	x	y	x^2	x^3	x^4	$x \times y$	$x^2 \times y$
1	80	6.47×10^{-6}	6.4×10^3	5.1200×10^5	4.096×10^7	5.1760×10^{-4}	4.1408×10^{-2}
2	40	6.24×10^{-6}	1.6×10^3	6.4000×10^4	2.56×10^6	2.4960×10^{-4}	9.9840×10^{-3}
3	-40	5.7200×10^{-6}	1.6×10^3	-6.4000×10^4	2.56×10^6	-2.2880×10^{-4}	9.1520×10^{-3}
4	-120	5.09×10^{-6}	1.44×10^4	-1.7280×10^6	2.0736×10^8	-6.1080×10^{-4}	7.3296×10^{-2}
5	-200	4.3×10^{-6}	4.0×10^4	-8.0000×10^6	1.60×10^9	-8.6000×10^{-4}	1.7200×10^{-1}
6	-280	3.33×10^{-6}	7.84×10^4	-2.1952×10^7	6.1466×10^9	-9.3240×10^{-4}	2.6107×10^{-1}
7	-340	2.45×10^{-6}	1.156×10^5	-3.9304×10^7	1.3363×10^{10}	-8.3300×10^{-4}	2.8322×10^{-1}
\sum	-8.6×10^2	3.36×10^{-5}	2.58×10^5	-7.0472×10^7	2.1363×10^{10}	-2.6978×10^{-3}	8.5013×10^{-1}

Thus we have following matrix

$$\begin{bmatrix} 7.0000 & -8.6000 \times 10^2 & 2.5800 \times 10^5 \\ -8.6000 \times 10^2 & 2.5800 \times 10^5 & -7.0472 \times 10^7 \\ 2.5800 \times 10^5 & -7.0472 \times 10^7 & 2.1363 \times 10^{10} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3.3600 \times 10^{-5} \\ -2.6978 \times 10^{-3} \\ 8.5013 \times 10^{-1} \end{bmatrix}$$

Solving the above system of simultaneous linear equations, we get

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 6.0217 \times 10^{-6} \\ 6.2782 \times 10^{-9} \\ -1.2218 \times 10^{-11} \end{bmatrix}$$

Thus, the polynomial regression model is

$$\begin{aligned} y &= a_0 + a_1 x + a_2 x^2 \\ &= 6.0217 \times 10^{-6} + 6.2782 \times 10^{-9} x - 1.2218 \times 10^{-11} x^2 \end{aligned}$$

//C Program to implement the polynomial regression

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int i,j,k,m,n;
    float a[20][20],b[20],z[20], x[20],fx[20];
    float sum,pivot,term;
    printf("Enter no of data points\n");
    scanf("%d",&n);
    printf("Enter degree of polynomial to be fitted\n");
    scanf("%d",&m);
```

```

printf("Enter data points (xi,fx(i))\n");
for(i=0;i<n;i++)
{
    scanf("%f%f",&x[i],&fx[i]);
}

//Construction of coefficient matrix
for(i=0;i<=m;i++)
for(j=0;j<=m;j++)
{
    sum=0;
    for(k=0;k<n;k++)
        sum=sum+pow(x[k],i+j);
    a[i][j]=sum;
}

//Construction of RHS vectors
for(i=0;i<=m;i++)
{
    sum=0;
    for(k=0;k<n;k++)
        sum=sum+fx[k]*pow(x[k],i);
    b[i]=sum;
}

for (k=0;k<m;k++) //Forward Elimination
{
    pivot=a[k][k];
    if (pivot<0.000001)
        printf("Method failed\n");
    else
        for(i=k+1;i<=m;i++)
        {
            term=a[i][k]/pivot;
            for(j=0;j<=m;j++)
            {
                a[i][j]=a[i][j]-a[k][j]*term;
            }
            b[i]=b[i]-b[k]*term;
        }
    z[m]=b[m]/a[m][m];
}

```

20 Numerical Methods with Practical Approach

```
for(i=m-1;i>=0;i--) //Back substitution
{
    sum=0;
    for(j=i+1;j<=m;j++)
    {
        sum=sum+a[i][j]*z[j];
    }
    z[i]=(b[i]-sum)/a[i][i];
}

printf("The polynomial of regression is :\n");
printf("y=%f + %f x",z[0],z[1]);
for(i=2;i<=m;i++)
printf("%f x^%d",z[i],i);
getch();
return 0;
}
```

Output

Enter no of data points

4

Enter degree of polynomial to be fitted

2

Enter data points (xi,fx(i))

1	6
2	11
3	18
4	27

The polynomial of regression is: $y=3.0 + 2.0 x + 1.0 x^2$

EXERCISE

1. How interpolation and regression are similar and in what sense they are different? When it is better to use regression rather than interpolation.
2. Describe the principle of least square regression. Why it is more superior than other regression principles?
3. Verify that least square regression principle gives more accurate prediction than other regression principles using example of your interest.
4. Develop flowcharts for fitting a line and polynomial through given data points. Describe applications of regression.
5. Determine the normal equations if the cubic polynomial $y = a + bx^2 + cx^3$ is fitted to n data points.

6. Write a mnu driven program to fit a straight line, exponential curve and polynomial through n data points

7. Find the least square approximation to the function $f(x) = e^x$, defined in $[0,1]$ as $f(0) = 1$, $f(0.5) = 1.6487$, $f(1) = 2.7183$

8. Using the method of least square find the straight line $y = ax + b$, that fits the following data.

X	0.5	1.0	1.5	2.0	2.5	3.0
Y	15	17	19	14	10	7

9. Using the method of least square find an equation of the form $y = ax + bx^2$, that fits the following data.

X	1.0	1.2	1.4	1.6	1.8	2.0
Y	0.98	1.40	1.86	2.55	2.28	3.20

10. Using the method of least square fit a parabola to the following data.

X	1	2	3	4	5	6
Y	2.6	5.4	8.7	12.1	16.0	20.2

11. Using the method of least square find the relation of the form $y = ax^b$ that fits the following data.

X	0.3010	0.4771	0.6021	0.6990
Y	1.4440	1.7931	2.0414	2.2068

12. Using the method of least square find the relation of the form $y = ae^{bx}$ to the following data

X	1	2	3	4
Y	1.65	2.70	4.50	7.35

13. Find the power fit $y = ax$ (straight line through the origin) for the data given below

X	1	2	3	4	5
Y	1.6	2.8	4.7	6.4	8.0

14. The result of measurement of electric resistance R of a copper wire at various temperatures is listed below.

X	19	25	30	36	40	45	50
Y	76	77	79	80	82	83	85

Using the method of least square, find the straight line $R = a + bt$ that fits best in the data.

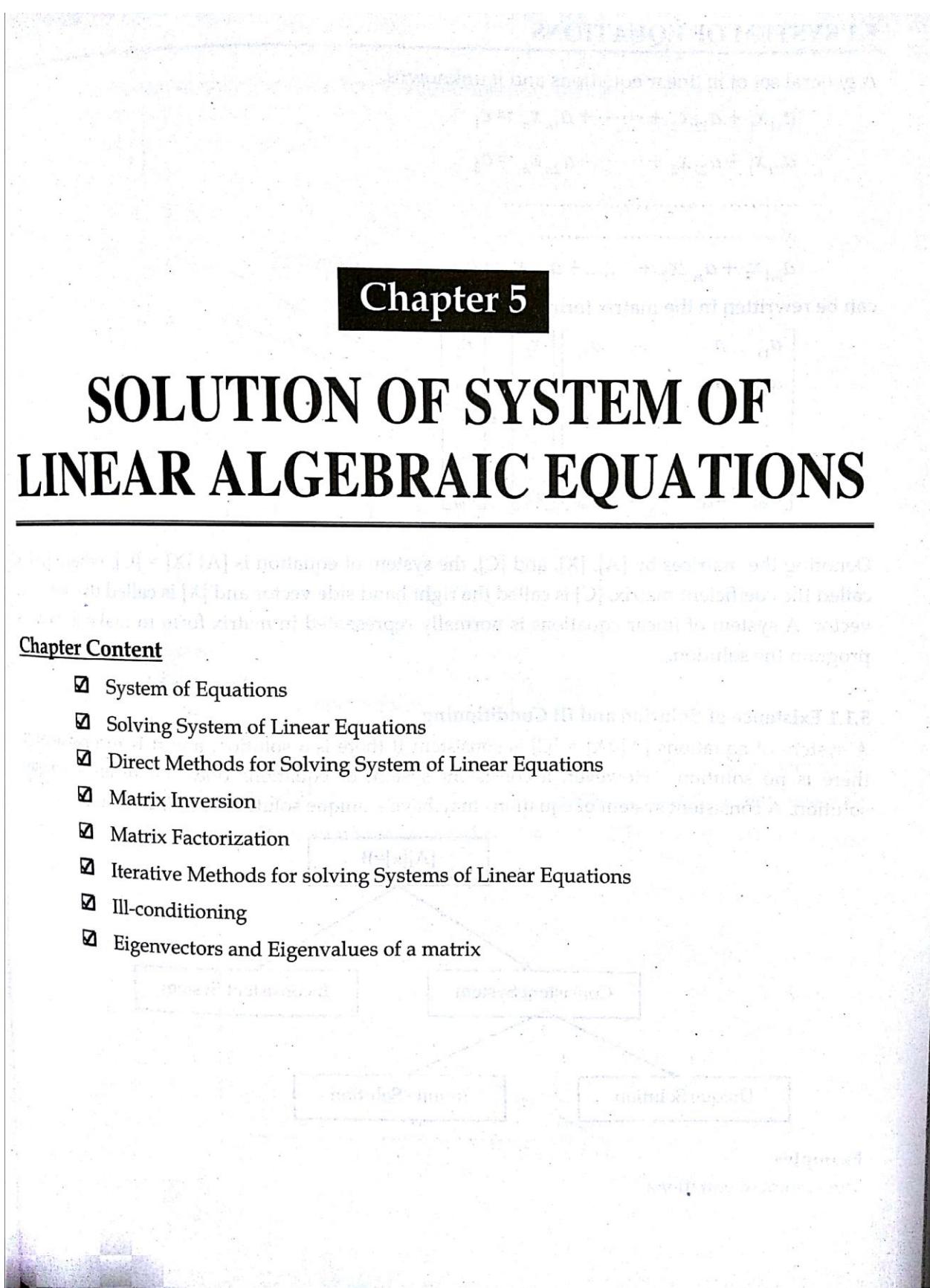


Chapter 5

SOLUTION OF SYSTEM OF LINEAR ALGEBRAIC EQUATIONS

Chapter Content

- System of Equations
- Solving System of Linear Equations
- Direct Methods for Solving System of Linear Equations
- Matrix Inversion
- Matrix Factorization
- Iterative Methods for solving Systems of Linear Equations
- Ill-conditioning
- Eigenvectors and Eigenvalues of a matrix



5.1 SYSTEM OF EQUATIONS

A general set of m linear equations and n unknowns,

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

.....

.....

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = c_m$$

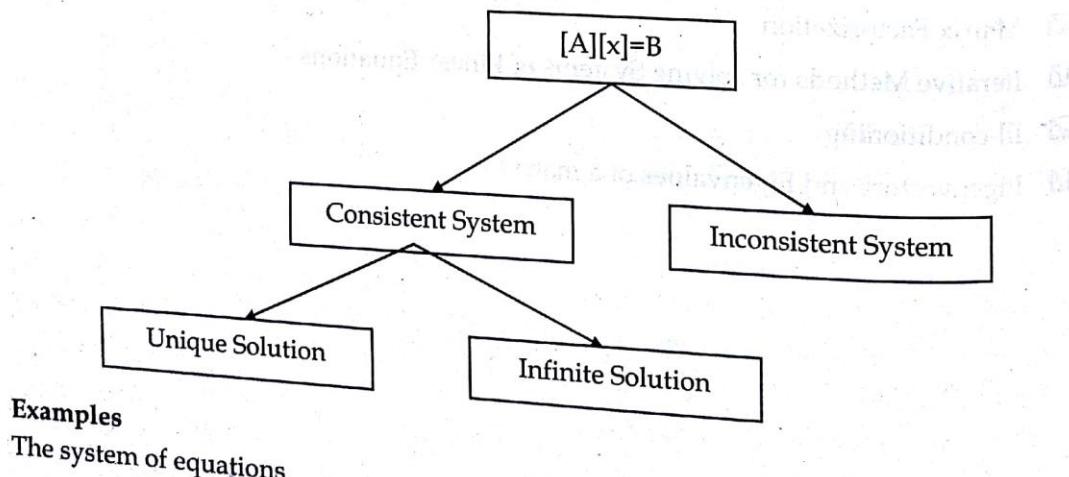
can be rewritten in the matrix form as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_m \end{bmatrix}$$

Denoting the matrices by $[A]$, $[X]$, and $[C]$, the system of equation is $[A][X] = [C]$, where $[A]$ is called the coefficient matrix, $[C]$ is called the right hand side vector and $[X]$ is called the solution vector. A system of linear equations is normally represented in matrix form to make it easy to program the solution.

5.1.1 Existence of Solution and Ill Conditioning

A system of equations $[A][X] = [C]$ is consistent if there is a solution, and it is inconsistent if there is no solution. However, a consistent system of equations does not mean a unique solution. A consistent system of equations may have a unique solution or infinite solutions.



Examples

The system of equations

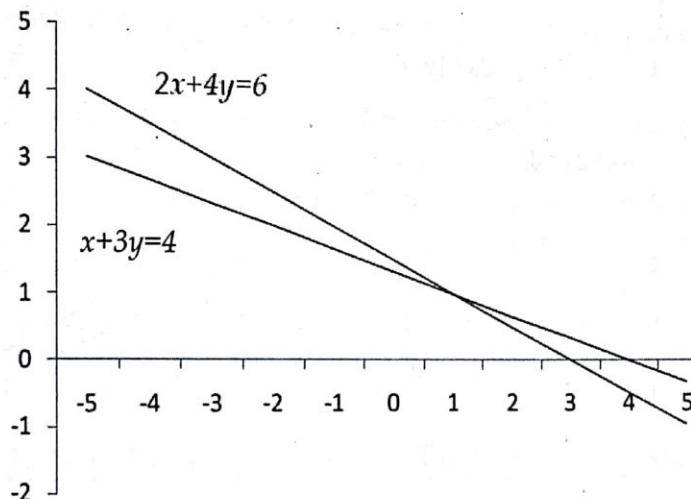
$$\begin{aligned} 2x + 4y &= 6 \\ x + 3y &= 4 \end{aligned}$$

in matrix form

$$\begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

is a consistent system of equations as it has a unique solution, that is,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



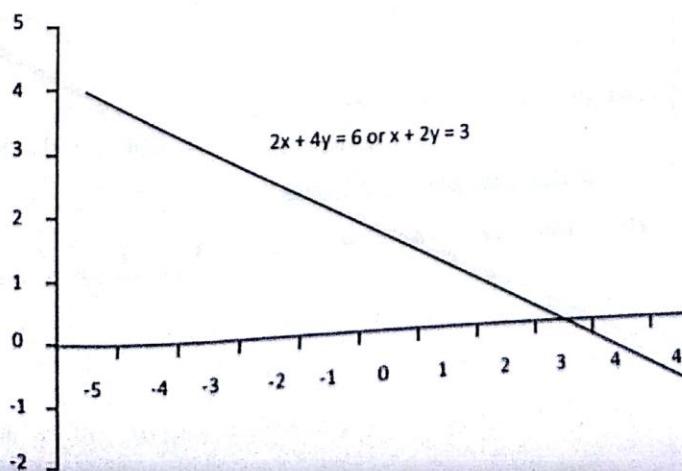
The system of equations

$$\begin{aligned} 2x + 4y &= 6 \\ x + 2y &= 3 \end{aligned}$$

in matrix form

$$\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$$

is also a consistent system of equations but it has infinite solutions. Since the lines are same, any combination of (x, y) that satisfies $2x + 4y = 6$ also satisfies $x + 2y = 3$. For example $(x, y) = (1, 1)$ is a solution. Other solutions include $(x, y) = (0.5, 1.25)$, $(x, y) = (0, 1.5)$, and so on.



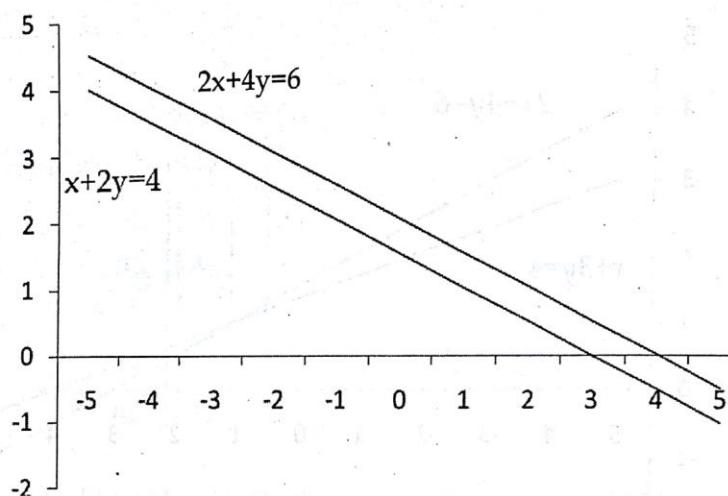
The system of equations

$$2x + 4y = 6$$

$$x + 2y = 4$$

in matrix form $\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$

is inconsistent as no solution exists.



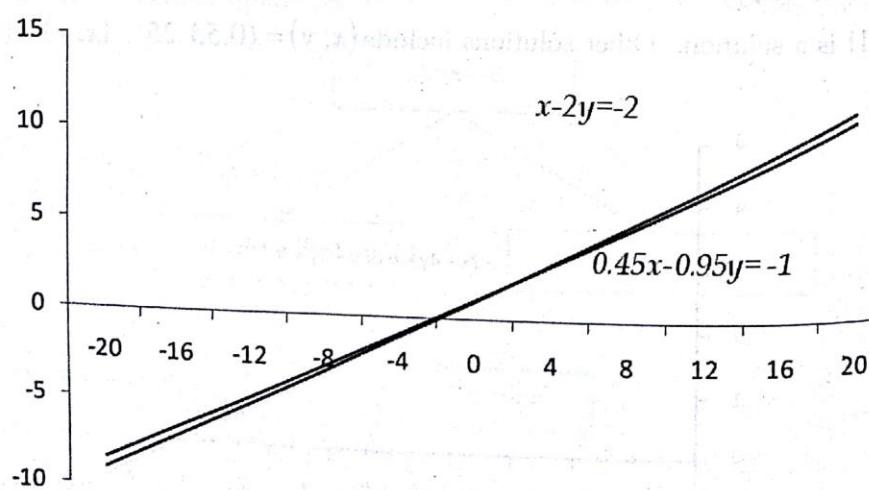
The system of equations

$$x - 2y = -2$$

$$0.45x - 0.95y = -1$$

in matrix form $\begin{bmatrix} 1 & 0.45 \\ -2 & -0.95 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}$

is a consistent system of equations but it has many solutions and it is difficult to find the exact point at which lines intersect. Such systems are called ill-conditioned systems.



5.2 SOLVING SYSTEM OF LINEAR EQUATIONS

There are two basic classes of methods for solving system of linear equations. The first class is represented by direct methods. A direct method is one that gives the exact solution to the system, if it is assumed that all calculations can be performed without round off error, which is almost impossible in computational practice. Moreover, solving an equation system by means of matrix decompositions can be classified as a direct method as well. The second class is called iterative methods, which construct a series of solution approximations that converges to the solution of the system.

Direct methods are not necessarily optimal for an arbitrary system. Let us deal with the main exceptions. First, even if a unique solution exists, direct numerical methods can fail to find the solution if the number of unknown variables n is large. Rounding errors can accumulate and result in a wrong solution. One alternative is to use iterative methods, which are less sensitive to these problems. Second, very large problems including hundreds or thousands of equations and unknown variables may be very time demanding to solve by standard direct methods. On the other hand, their coefficient matrices are often sparse, that is, most of their elements are zeros. Special strategies to store and solve such problems are required.

5.3 DIRECT METHODS FOR SOLVING SYSTEM OF LINEAR EQUATIONS

5.3.1 Naïve Gauss Elimination Method

One of the most popular techniques for solving simultaneous linear equations is the Gauss elimination method. The approach is designed to solve a general set of n equations and n unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n &= b_2 \end{aligned}$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n = b_n$$

Gaussian elimination consists of two steps

1. **Forward Elimination of Unknowns:** In this step, the unknowns are eliminated from each equation starting with the first equation.

2. **Back Substitution:** In this step, starting from the last equation, value of each of the unknowns is found.

Forward Elimination of Unknowns

In the first step of forward elimination, the first unknown, x_1 , is eliminated from all rows below the first row. The first equation is selected as the pivot equation to eliminate x_1 . So, to eliminate x_1 in the second equation, one divides the first equation by a_{11} (hence called the pivot element)

and then multiplies it by a_{21} . This is the same as multiplying the first equation by a_{21}/a_{11} give

$$a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \dots + \frac{a_{21}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}b_1$$

Now, this equation can be subtracted from the second equation to give

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12} \right)x_2 + \dots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n} \right)x_n = b_2 - \frac{a_{21}}{a_{11}}b_1$$

or

$$a'_{22}x_2 + \dots + a'_{2n}x_n = b'_2$$

Where,

$$a'_{22} = a_{22} - \frac{a_{21}}{a_{11}}a_{12}$$

$$a'_{2n} = a_{2n} - \frac{a_{21}}{a_{11}}a_{1n}$$

This procedure of eliminating x_1 , is now repeated for the third equation to the n^{th} equation to reduce the set of equations as

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2$$

$$a'_{32}x_2 + a'_{33}x_3 + \dots + a'_{3n}x_n = b'_3$$

$$a'_{n2}x_2 + a'_{n3}x_3 + \dots + a'_{nn}x_n = b'_n$$

This is the end of the first step of forward elimination. Now for the second step of forward elimination, we start with the second equation as the pivot equation and a'_{22} as the pivot element. So, to eliminate x_2 in the third equation, one divides the second equation by a'_{22} (the pivot element) and then multiply it by a'_{32} . This is the same as multiplying the second equation by a'_{32}/a'_{22} and subtracting it from the third equation. This makes the coefficient of x_2 zero in the third equation. The same procedure is now repeated for the fourth equation till the n^{th} equation to give

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2$$

$$a''_{32}x_2 + a''_{33}x_3 + \dots + a''_{3n}x_n = b''_3$$

$$a''_{n3}x_3 + \dots + a''_{nn}x_n = b''_n$$

The next steps of forward elimination are conducted by using the third equation as a pivot equation and so on. That is, there will be a total of $n-1$ steps of forward elimination. At the end of $(n-1)^{\text{th}}$ steps of forward elimination, we get a set of equations that looks like

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2$$

$$a''_{33}x_3 + \dots + a''_{3n}x_n = b''_3$$

$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)}$$

Back Substitution

Now the equations are solved starting from the last equation as it has only one unknown.

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

Then the second last equation, that is the $(n-1)^{\text{th}}$ equation, has two unknowns: x_n and x_{n-1} , but x_n is already known. This reduces the $(n-1)^{\text{th}}$ equation also to one unknown. Back substitution hence can be represented for all equations by the formula

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^n a_{ij}^{(i-1)}x_j}{a_{ii}^{(i-1)}} \quad \text{for } i = n-1, n-2, \dots, 1$$

and

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

Algorithm for Naïve Gauss Elimination Method

1. Start
2. Read Dimension of System of equations, Say n
3. Read coefficients of matrix row-wise
4. Read RHS vector
5. Perform forward elimination as below

For $k=1$ to $n-1$

 pivot=a[k][k]

 if (pivot<0.000001)

 Display the message "Method failed"

 else

 For $i=k+1$ to n

term=a[i][k]/pivot
 Multiply row k of coefficient matrix by "term" and subtract it from row i
 Multiply row k of B matrix by "term" and subtract it from row i

End for
 End for

6. Perform back substitution as below

```
x[n]=b[n]/a[n][n]
for i=n-1 to 1
  sum=0
  for j=i+1 to n
    sum=sum+a[i][j]*x[j]
  End for
  x[i]=(b[i]-sum)/a[i][i]
End for
```

7. Display solution vector

8. Terminate

Example

Use Naïve Gauss elimination to solve

$$x_1 - 3x_2 + x_3 = 4$$

$$2x_1 - 8x_2 + 8x_3 = -2$$

$$-6x_1 + 3x_2 - 15x_3 = 9$$

Solution

Given equations are

$$x_1 - 3x_2 + x_3 = 4$$

$$2x_1 - 8x_2 + 8x_3 = -2$$

$$-6x_1 + 3x_2 - 15x_3 = 9$$

Representing above equations in matrix form, we get

$$\begin{bmatrix} 1 & -3 & 1 \\ 2 & -8 & 8 \\ -6 & 3 & -15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ 9 \end{bmatrix}$$

Forward Elimination of Unknowns

Step 1

Multiply Row 1 by 2 and subtract it from Row 2. That is, perform: {R₂=R₂-2R₁}

$$\begin{bmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ -6 & 3 & -15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -10 \\ 9 \end{bmatrix}$$

Multiply Row 1 by 6 and add it to Row 3. That is, perform: {R₃=R₃+6R₁}

The resulting matrix is

$$\begin{bmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & -15 & -9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -10 \\ 33 \end{bmatrix}$$

Step 2

Multiply Row 2 by 15/2 and subtract it from Row 3. That is, perform: $\{R_3=R_3 - 15/2 R_2\}$

The resulting matrix is

$$\begin{bmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & 0 & 54 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -10 \\ -108 \end{bmatrix}$$

This is the end of the forward elimination steps.

Back substitution

We can now solve the above equations by back substitution. From the third row,

$$54x_3 = -108 \Rightarrow x_3 = -2$$

Substituting the value of x_3 in the second row

$$-2x_2 + 6x_3 = -10$$

$$-2x_2 = 2 \Rightarrow x_2 = -1$$

Substituting the value of x_3 and x_2 in the first row,

$$x_1 - 3x_2 + x_3 = 4$$

$$\Rightarrow x_1 = 3$$

Hence the solution is

$$[X] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \\ -2 \end{bmatrix}$$

Second Example

Use Naïve Gauss elimination to solve

$$20x_1 + 15x_2 + 10x_3 = 45$$

$$-3x_1 - 2.249x_2 + 7x_3 = 1.751$$

$$5x_1 + x_2 + 3x_3 = 9$$

Solution

Given equations are

$$20x_1 + 15x_2 + 10x_3 = 45$$

$$-3x_1 - 2.249x_2 + 7x_3 = 1.751$$

$$5x_1 + x_2 + 3x_3 = 9$$

Representing above equations in matrix form, we get

$$\begin{bmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 1.751 \\ 9 \end{bmatrix}$$

Forward Elimination of Unknowns**Step 1**

Multiply Row 1 by $3/20$ and add it to Row 2. That is, perform $R_2=R_2+3/20 R_1$

The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ 9 \end{bmatrix}$$

Again,

Multiply Row 1 by $5/20=0.25$ and subtract it from Row 3. That is, perform $R_3=R_3-5/20 R_1$

The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & -2.75 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ -2.25 \end{bmatrix}$$

Step 2

Multiply Row 2 by 0 by $2.75/0.001$ and add it to Row 3. That is, perform $R_3=R_3 + 2.75/0.001 R_2$

The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & 0 & 23375.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ 23375.4 \end{bmatrix}$$

This is the end of the forward elimination steps.

Back substitution

We can now solve the above equations by back substitution. From the third row,

$$23375.5x_3 = 23375.4 \Rightarrow x_3 = \frac{23375.4}{23375.5} = 0.999995$$

Substituting the value of x_3 in the second row

$$0.001x_2 + 8.5x_3 = 8.501$$

$$x_2 = \frac{8.501 - 8.5x_3}{0.001} = \frac{8.501 - 8.5 \times 0.999995}{0.001} = \frac{8.501 - 8.49995}{0.001} = 1.05$$

Substituting the value of x_3 and x_2 in the first row,

$$20x_1 + 15x_2 + 10x_3 = 45$$

$$x_1 = \frac{45 - 15x_2 - 10x_3}{20} = \frac{45 - 15 \times 1.05 - 10 \times 0.999995}{20} = \frac{45 - 15.75 - 9.99995}{20} = 0.9625$$

Hence the solution is

$$[X] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.9625 \\ 1.05 \\ 0.999995 \end{bmatrix}$$

//C Program for Naïve Gauss Elimination method

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,k,p,q;
    float pivot, term,sum=0, a[10][10], b[10], x[10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients row-wise\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    printf("Enter RHS vector\n");
    for(i=0;i<n;i++)
    {
        scanf("%f",&b[i]);
    }
    for (k=0;k<=n-2;k++) //Forward Elimination
    { //Multiply row k of coefficient matrix by a[i][k]/a[k][k] and subtract it from row i
        pivot=a[k][k];
        if (fabs(pivot)<0.000001)
            printf("Method failed\n");
        else
            for (i=k+1;i<n;i++)
        {
            term=a[i][k]/pivot;
            for(j=0;j<n;j++)
            {
                a[i][j]=a[i][j]-a[k][j]*term;
            }
            //Multiply row k of B matrix by a[i][k]/a[k][k] and subtract it from row i
            b[i]=b[i]-b[k]*term;
        }
    }
}

```

```

x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--) //Back substitution
{
    sum=0;
    for(j=i+1;j<=n-1;j++)
    {
        sum=sum+a[i][j]*x[j];
    }
    x[i]=(b[i]-sum)/a[i][i];
}

printf("Solution:\n");
for(i=0;i<n;i++) //Display Solution Vector
{
    printf("x%d=%f\n",i+1,x[i]);
}
getch();
return 0;
}

```

Output:

Enter Dimension of System of equations

3

Enter coefficients row-wise

2 1 1

3 2 3

1 4 9

Enter RHS vector

10 18 16

Solution

x1=7.00 x2=-9.00

x3=5.00

Drawbacks of Naïve Gauss Elimination Method

- ✓ **Division by Zero:** Naïve Gauss elimination method may suffer from division by zero problems at the beginning of the each step of forward elimination. This occurs when pivot element is zero.
- ✓ **Round-off Error:** The Naïve Gauss elimination method is prone to round-off errors. This is can be serious problem when there are large numbers of equations as errors propagate. Also, if there is subtraction of floating point numbers from each other, it may create large errors.

5.3.2 Gauss Elimination with Partial and Complete Pivoting

Round off errors were large in case of Naïve Gauss Elimination method. One method of decreasing the round-off error would be to use more significant digits, that is, use double or quad precision for representing the numbers. However, this would not avoid problem of division by zero present in the Naïve Gauss elimination method. To avoid division by zero as well as reduce (not eliminate) round-off error, Gaussian elimination with partial pivoting is the method of choice. Thus partial pivoting is required due to following two reasons

- To avoid division by zero problem
- To reduce round-off errors

Naïve Gauss elimination method and Gauss elimination with partial are the same, except in the beginning of each step of forward elimination; a row switching is done based on the following criterion. If there are n equations, then there are $n-1$ forward elimination steps. At the beginning of the k^{th} step of forward elimination, one finds the maximum of

$$|a_{kk}|, |a_{k+1,k}|, \dots, |a_{nk}|$$

Then if the maximum of these values is $|a_{pk}|$ in the p^{th} row, $k \leq p \leq n$, then switch rows p and k . The other steps of forward elimination are the same as the Naïve Gauss elimination method. The back substitution steps stay exactly the same as the Naïve Gauss elimination method.

Algorithm for Gauss Elimination with Partial Pivoting

1. Start
2. Read Dimension of System of equations, Say n
3. Read coefficient matrix row-wise
4. Read RHS vector
5. Perform forward elimination as below

For $k=1$ to $n-1$

Find largest of $a[p][k]$ for $p=k, k+1, \dots, n$

Swap row k and row p in coefficient matrix

Swap row k and row p in RHS vector

$\text{pivot} = a[k][k]$

For $i=k+1$ to n

$\text{term} = (a[i][k] / \text{pivot})$

Multiply row k of coefficient matrix by "term" and subtract it from row i

Multiply row k of B matrix by "term" and subtract it from row i

End for

End for

6. Perform back substitution as below

$$x[n] = b[n] / a[n][n]$$

for $i=n-1$ to 1

```

sum=0
for j=i+1 to n
    sum=sum+a[i][j]*x[j]
End for
x[i]=(b[i]-sum)/a[i][i]
End for
7. Display solution vector
8. Terminate

```

Example

Solve following system of linear equations by using gauss elimination with partial pivoting

$$2x_1 + x_2 + x_3 = 5$$

$$4x_1 - 6x_2 = -2$$

$$-2x_1 + 7x_2 + 2x_3 = 9$$

Solution

Given equations are

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

$$4x_1 - 6x_2 = -2$$

$$-2x_1 + 7x_2 + 2x_3 = 9$$

Representing above equations in matrix form, we get

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

Forward Elimination of Unknowns**Step 1**

Since, largest absolute value among a_{11} , a_{12} , and a_{13} is 4, switch Row 1 and Row 2. The resulting matrix is $R1 \leftrightarrow R2$

$$\begin{bmatrix} 4 & -6 & 0 \\ 2 & 1 & 1 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ 9 \end{bmatrix}$$

Multiply Row 1 by $1/2$ and subtract it from Row 2. That is, perform: $\{R_2=R_2 - 1/2 R_1\}$
The resulting matrix is

$$\begin{bmatrix} 4 & -6 & 0 \\ 0 & 4 & 1 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \\ 9 \end{bmatrix}$$

Multiply Row 1 by $1/2$ and add it to Row 2. That is, perform: $\{R_3=R_3 + 1/2 R_1\}$
The resulting matrix is

$$\begin{bmatrix} 4 & -6 & 0 \\ 0 & 4 & 1 \\ 0 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \\ 8 \end{bmatrix}$$

Step 2
Since, largest absolute value among a_{22}, a_{23} is 4, we do not need to switch rows.

$$\begin{bmatrix} 4 & -6 & 0 \\ 0 & 4 & 1 \\ 0 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \\ 8 \end{bmatrix}$$

Now, subtract Row 2 from Row 3. That is, perform: $\{R_3=R_3-R_2\}$
The resulting matrix is

$$\begin{bmatrix} 4 & -6 & 0 \\ 0 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \\ 2 \end{bmatrix}$$

This is the end of the forward elimination steps.

Back substitution

We can now solve the above equations by back substitution. From the third row,

$$x_3 = 2$$

Substituting the value of x_3 in the second row

$$4x_2 + x_3 = 6$$

$$4x_2 = 4 \Rightarrow x_2 = 1$$

Substituting the value of x_3 and x_2 in the first equation,

$$4x_1 - 6x_2 = -2$$

$$4x_1 = 4 \Rightarrow x_1 = 1$$

Hence the solution is

$$[X] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Second Example

Use Gauss elimination with partial pivoting to solve following equations

$$20x_1 + 15x_2 + 10x_3 = 45$$

$$-3x_1 - 2.249x_2 + 7x_3 = 1.751$$

$$5x_1 + x_2 + 3x_3 = 9$$

Solution

$$\begin{bmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 1.751 \\ 9 \end{bmatrix}$$

Forward Elimination of Unknowns

Step 1

Since, largest absolute value among a_{11} , a_{12} , and a_{13} is 20, Row switching is not needed. The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 1.751 \\ 9 \end{bmatrix}$$

Multiply Row 1 by 3/20 and add it to Row 2. That is, perform $\{R_2=R_2+3/20 R_1\}$.
The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ 9 \end{bmatrix}$$

Multiply Row 1 by 5/20 and subtract it from Row 3 $\{R_3=R_3-5/20 R_1\}$.
The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & -2.75 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ -2.25 \end{bmatrix}$$

Step 2

Since, largest absolute value among a_{22} and a_{23} is 2.75. Switch Row 2 and Row 3. The resulting matrix is:

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & -2.75 & 0.5 \\ 0 & 0.001 & 8.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -2.25 \\ 8.501 \end{bmatrix}$$

Multiply Row 2 by 0.001/2.75 and add it to Row 3. That is, perform $R_3=R_3-0.001/2.75 R_1$. The resulting matrix is

$$\begin{bmatrix} 20 & 15 & 10 \\ 0 & -2.75 & 0.5 \\ 0 & 0 & 8.5001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ -2.25 \\ 8.5001 \end{bmatrix}$$

$$8.5001x_3 = 8.5001 \Rightarrow x_3 = \frac{8.5001}{8.5001} = 1$$

Substituting the value of x_3 in Row 2

$$-2.75x_2 + 0.5x_3 = -2.25$$

$$\Rightarrow x_2 = \frac{-2.25 - 0.5x_3}{-2.75} = \frac{-2.25 - 0.5}{-2.75} = 1$$

Substituting the value of x_3 and x_2 in Row 1

$$20x_1 + 15x_2 + 10x_3 = 45$$

$$\Rightarrow x_1 = \frac{45 - 15x_2 - 10x_3}{20} = \frac{45 - 15 - 10}{20} = 1$$

So the solution is

$$[X] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This, in fact, is the exact solution. By coincidence only, in this case, the round-off error is fully removed.

//C program for Gauss Elimination with pivoting

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,k,j,p,q,row;
    float pivot, temp, largest, term,sum=0, a[10][10], b[10], x[10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients row-wise\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    printf("Enter RHS vector\n");
    for(i=0;i<n;i++)
    {
        scanf("%f",&b[i]);
    }
    for (k=0;k<=n-2;k++) //Forward Elimination
    {
        largest=fabs(a[k][k]);
        p=k;
        for (j=k+1;j<n;j++)
        {
            if (fabs(a[j][k])>largest)
            {
                largest=fabs(a[j][k]);
                p=j;
            }
        }
        if (p!=k)
        {
            for (i=k;i<n;i++)
            {
                temp=a[p][i];
                a[p][i]=a[k][i];
                a[k][i]=temp;
            }
            temp=b[p];
            b[p]=b[k];
            b[k]=temp;
        }
        for (j=k+1;j<n;j++)
        {
            term=a[j][k]/a[k][k];
            for (i=k;i<n;i++)
            {
                a[j][i]-=term*a[k][i];
            }
            b[j]-=term*b[k];
        }
    }
    //Swapping original pivotb row with the one containing largest element
    largest=fabs(a[n-1][n-1]);
    p=n-1;
    for (j=n-2;j>=0;j--)
    {
        if (fabs(a[j][n-1])>largest)
        {
            largest=fabs(a[j][n-1]);
            p=j;
        }
    }
    if (p!=n-1)
    {
        for (i=n-1;i>=0;i--)
        {
            temp=a[p][i];
            a[p][i]=a[n-1][i];
            a[n-1][i]=temp;
        }
        temp=b[p];
        b[p]=b[n-1];
        b[n-1]=temp;
    }
    for (i=n-1;i>=0;i--)
    {
        x[i]=b[i]/a[i][i];
    }
}
```

```

for(p=k+1;p<=n-1;p++)
{
    if(fabs(a[p][k])>largest)
    {
        largest=fabs(a[p][k]);
        row=p;
    }
}

for(p=0;p<n;p++)
{
    temp=a[k][p];
    a[k][p]=a[row][p];
    a[row][p]=temp;
}
temp=b[k];
b[k]=b[row];
b[row]=b[p];

//Multiply row k of coefficient matrix by a[i][k]/a[k][k] and subtract it from row i
pivot=a[k][k];
for (i=k+1;i<n;i++)
{
    term=a[i][k]/pivot;
    for(j=0;j<n;j++)
    {
        a[i][j]=a[i][j]-a[k][j]*term;
    }
    //Multiply row k of B matrix by a[i][k]/a[k][k] and subtract it from row i
    b[i]=b[i]-b[k]*term;
}
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--) //Back substitution
{
    sum=0;
    for(j=i+1;j<=n-1;j++)
    {
        sum=sum+a[i][j]*x[j];
    }
    x[i]=(b[i]-sum)/a[i][i];
}
printf("Solution:\n");
for(i=0;i<n;i++) //Display Solution Vector
{
    printf("x%d=%f\n",i+1,x[i]);
}
getch();
return 0;
}

```

Output

Enter Dimension of System of equations

3

Enter coefficients row-wise

2 2 1

4 2 3

1 1 1

Enter RHS vector

6 4 0

Solution

x1=3.00 x2=-1.00 x3=-2.00

5.3.3 Gauss-Jordan Method

It is a variation of Gauss elimination. The differences between Gauss Elimination method and Gauss Jordan method are described below:

- When an unknown is eliminated from an equation, it is also eliminated from all other equations. All rows are normalized by dividing them by their pivot element. Hence, the elimination step results in an identity matrix rather than a triangular matrix.
- Back substitution is not required. We can obtain solution directly from identity matrix obtained from elimination step.

All other techniques developed for Gauss elimination are still valid for Gauss-Jordan elimination. However, Gauss-Jordan requires more computational work than Gauss elimination (approximately 50% more operations). The goal in Gauss-Jordan Elimination is to use row operations (interchange two rows, multiply a row by a nonzero constant, add two rows, or add a multiple of one row to another row) to change the augmented matrix to row reduced echelon form which looks like the following matrix:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{array} \right]$$

Notice the left hand side is the identity matrix while the right hand side can be any numbers.
To get a matrix in row reduced echelon form, follow the method suggested below.

First: $\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array} \right]$ Use row operations to make $a_{11}=1$. Then use row operations to make the other elements in column 1 zero.

Second: $\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & a_{22} & a_{23} & b_2 \\ 0 & a_{32} & a_{33} & b_3 \end{array} \right]$ Use row operations to make $a_{22}=1$. Then use row operations to make the other elements in column 2 zero.

Third:
$$\left[\begin{array}{ccc|c} 1 & 0 & a_{13} & b_1 \\ 0 & 1 & a_{23} & b_2 \\ 0 & 0 & a_{33} & b_3 \end{array} \right]$$

Use row operations to make $a_{33} = 1$. Then use row operations to make the other elements in column 3 zero.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{array} \right]$$

Now we can obtain values of unknowns directly from right hand side of augmented matrix without employing back substitution. The solution vector is:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Algorithm for Gauss-Jordan Method

1. Start
2. Read Dimension of System of equations, say n
3. Read coefficients of augmented matrix row-wise
4. Perform row operations to convert LHS of augmented matrix into identity matrix

For k=1 to n

pivot=a[k][k]

for p=1 to n+1 //Normalize row k

a[k][p]=a[k][p]/pivot

for i=1 to n

term=a[i][k]

if(i≠k)

Multiply row k by "term" and subtract it from row i

End for

End for

5. Display solution vector which is last column of augmented matrix
6. Terminate

Example

Solve the following system of linear equations by Gauss - Jordan method

$$2x_1 - x_2 + 4x_3 = 15$$

$$2x_1 + 3x_2 - 2x_3 = 1$$

$$3x_1 + 2x_2 - 4x_3 = -4$$

Solution

Augmented matrix is

$$\left[\begin{array}{ccc|c} 2 & -1 & 4 & 15 \\ 2 & 3 & -2 & 1 \\ 3 & 2 & -4 & -4 \end{array} \right]$$

Step 1

 Perform the operation $R_1 = R_1 / 2$

$$\left[\begin{array}{ccc|c} 1 & -1/2 & 2 & 15/2 \\ 2 & 3 & -2 & 1 \\ 3 & 2 & -4 & -4 \end{array} \right]$$

Step 2

 Perform the operations $R_2 = R_2 - 2R_1$ and $R_3 = R_3 - 3R_1$

$$\left[\begin{array}{ccc|c} 1 & -1/2 & 2 & 15/2 \\ 0 & 4 & -6 & -14 \\ 0 & 7/2 & -10 & -53/2 \end{array} \right]$$

Step 3

 Perform the operation $R_2 = R_2 / 4$

$$\left[\begin{array}{ccc|c} 1 & -1/2 & 2 & 15/2 \\ 0 & 1 & -3/2 & -7/2 \\ 0 & 7/2 & -10 & -53/2 \end{array} \right]$$

Step 4

 Perform the operations $R_1 = R_1 + \frac{1}{2}R_2$ and $R_3 = R_3 - \frac{7}{2}R_2$

$$\left[\begin{array}{ccc|c} 1 & 0 & 5/4 & 23/4 \\ 0 & 1 & -3/2 & -7/2 \\ 0 & 0 & -19/4 & -57/4 \end{array} \right]$$

Step 5

 Perform the operation $R_3 = -\frac{4}{19}R_3$

$$\left[\begin{array}{ccc|c} 1 & 0 & 5/4 & 23/4 \\ 0 & 1 & -3/2 & -7/2 \\ 0 & 0 & 1 & 3 \end{array} \right]$$

Step 6

 Perform the operations $R_1 = R_1 - \frac{5}{4}R_3$ and $R_2 = R_2 + \frac{3}{2}R_3$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{array} \right]$$

Thus solution of above system of linear equation is: $x_1 = 2$, $x_2 = 1$, $x_3 = 3$

Second Example

Solve the following system of linear equations by Gauss - Jordan method.

$$a+b+2c=1$$

$$2a-b+d=-2$$

$$a-b-c-2d=4$$

$$2a-b+2c-d=0$$

Solution

Augmented matrix is

$$\left[\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 2 & -1 & 0 & 1 & -2 \\ 1 & -1 & -1 & -2 & 4 \\ 2 & -1 & 2 & -1 & 0 \end{array} \right]$$

Step 1

Perform the operations $R_2 = R_2 - 2R_1$, $R_3 = R_3 - R_1$ and $R_4 = R_4 - 2R_1$

$$\left[\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 0 & -3 & -4 & 1 & -4 \\ 0 & -2 & -3 & -2 & 3 \\ 0 & -3 & -2 & -1 & -2 \end{array} \right]$$

Step 2

Perform the operation $R_2 = -R_2 / 3$

$$\left[\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 0 & 1 & 4/3 & -1/3 & 4/3 \\ 0 & -2 & -3 & -2 & 3 \\ 0 & -3 & -2 & -1 & -2 \end{array} \right]$$

Step 3

Perform the operations $R_1 = R_1 - R_2$, $R_3 = R_3 + 2R_2$ and $R_4 = R_4 + 3R_2$

$$\left[\begin{array}{cccc|c} 1 & 0 & 2/3 & 1/3 & -1/3 \\ 0 & 1 & 4/3 & -1/3 & 4/3 \\ 0 & 0 & -1/3 & -8/3 & 17/3 \\ 0 & 0 & 2 & -2 & 2 \end{array} \right]$$

Step 4

Perform the operation $R_3 = -3R_3$

$$\left[\begin{array}{cccc|c} 1 & 0 & 2/3 & 1/3 & -1/3 \\ 0 & 1 & 4/3 & -1/3 & 4/3 \\ 0 & 0 & 1 & 8 & -17 \\ 0 & 0 & 2 & -2 & 2 \end{array} \right]$$

Step 5

Perform the operations $R_1 = R_1 - \frac{2}{3}R_3$, $R_2 = R_2 - \frac{4}{3}R_3$ and $R_4 = R_4 - 2R_3$

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & -5 & 11 \\ 0 & 1 & 0 & -11 & 24 \\ 0 & 0 & 1 & 8 & -17 \\ 0 & 0 & 0 & -18 & 36 \end{array} \right]$$

Step 6

Perform the operation $R_4 = -\frac{1}{18}R_4$

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & -5 & 11 \\ 0 & 1 & 0 & -11 & 24 \\ 0 & 0 & 1 & 8 & -17 \\ 0 & 0 & 0 & 1 & -2 \end{array} \right]$$

Step 7

Perform the operations $R_1 = R_1 + 5R_4$, $R_2 = R_2 + 11R_4$ and $R_3 = R_3 - 8R_4$

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -2 \end{array} \right]$$

Thus solution of above system of linear equation is: $a = 1$, $b = 2$, $c = -1$, and $d = -2$

//C program for Gauss Jordan Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,k,j,p,q;
    float pivot,term, a[10][10];
```

```

printf("Enter Dimension of System of equations\n");
scanf("%d",&n);
printf("Enter coefficients Augmented Matrix\n");
for(i=0;i<n;i++)
for(j=0;j<n+1;j++)
{
    scanf("%f",&a[i][j]);
}
for (k=0;k<n;k++) //Elimination
{
    pivot=a[k][k];
    for(p=0;p<n+1;p++)
        a[k][p]=a[k][p]/pivot; //Normalization
    for (i=0;i<n;i++)
    {
        term=a[i][k];
        if(k!=i)
            for(j=0;j<n+1;j++)
            {
                a[i][j]=a[i][j]-a[k][j]*term;
            }
    }
}
printf("Solution:\n");
for(i=0;i<n;i++) //Display Solution Vector
{
    printf("x%d=%f\n",i+1,a[i][3]);
}
getch();
return 0;
}

```

Output

Enter Dimension of System of equations
3
Enter coefficients Augmented Matrix
2 4 -6 -8
1 3 1 10
2 -4 -2 -12
Solution:
x1=1.00 x2=2.00 x3=3.00

5.4 MATRIX INVERSION

Although Gauss-Jordan method is more time consuming than Gauss-Elimination method, it provides simple approach for computing inverse of a matrix. A matrix X is said to be inverse of M if $MX=I$, where I is identity matrix of same order as that of matrix M. Matrix inverse can be computed by using Gauss-Jordan method in following two steps:

Step 1: Augment the coefficient matrix with identity matrix as below:

$$\left[\begin{array}{ccc|ccc} a_{11} & a_{12} & a_{31} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 1 & 0 & 1 \end{array} \right]$$

Step 2: Apply Gauss-Jordan method to the augmented matrix to reduce coefficient matrix to identity matrix as below:

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & a'_{11} & a'_{12} & a'_{31} \\ 0 & 1 & 0 & a'_{21} & a'_{22} & a'_{23} \\ 0 & 0 & 1 & a'_{31} & a'_{32} & a'_{33} \end{array} \right]$$

Now the right hand side of above augmented matrix is the inverse of original coefficient matrix.

Algorithm for Matrix Inversion

1. Start
2. Read dimension of Matrix, say n
3. Read coefficients of matrix row-wise
4. Augment the coefficient matrix by identity matrix
5. Perform elimination operation as below

For k=1 to n

pivot=a[k][k]

for p=1 to 2n //Normalize row k

a[k][p]=a[k][p]/pivot

End for

for i=1 to n

term=a[i][k]

if(i≠k)

Multiply row k by "term" and subtract it from row i

End for

End for

6. Display inverse matrix which is second half of augmented matrix

7. Terminate

Second Example

Find the inverse of the matrix

$$\begin{bmatrix} 7 & 3 \\ 5 & 2 \end{bmatrix}$$

Solution

Step 1

Augmented matrix is:

$$\left[\begin{array}{cc|cc} 7 & 3 & 1 & 0 \\ 5 & 2 & 0 & 1 \end{array} \right]$$

Step 2

Perform the operation $R_1 = \frac{1}{7}R_1$

$$\left[\begin{array}{cc|cc} 1 & 3/7 & 1/7 & 0 \\ 5 & 2 & 0 & 1 \end{array} \right]$$

Step 2

Perform the operation $R_2 = R_2 - 5R_1$

$$\left[\begin{array}{cc|cc} 1 & 3/7 & 1/7 & 0 \\ 0 & -1/7 & -5/7 & 1 \end{array} \right]$$

Step 3

Perform the operation $R_2 = -7R_2$

$$\left[\begin{array}{cc|cc} 1 & 3/7 & 1/7 & 0 \\ 0 & 1 & 5 & -7 \end{array} \right]$$

Step 4

Perform the operation $R_1 = R_1 - \frac{3}{7}R_2$

$$\left[\begin{array}{cc|cc} 1 & 0 & -2 & 3 \\ 0 & 1 & 5 & -7 \end{array} \right]$$

Therefore, the inverse of matrix A is

$$\begin{bmatrix} -2 & 3 \\ 5 & -7 \end{bmatrix}$$

Second Example

Find the inverse of the matrix

$$\begin{bmatrix} 1 & -1 & 1 \\ 2 & 3 & 0 \\ 0 & -2 & 1 \end{bmatrix}$$

Solution

Step 1

Augmented matrix is:

$$\left[\begin{array}{ccc|ccc} 1 & -1 & 1 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 & 1 & 0 \\ 0 & -2 & 1 & 0 & 0 & 1 \end{array} \right]$$

Step 2

Perform the operation $R_2 = R_2 - 2R_1$

$$\left[\begin{array}{ccc|ccc} 1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 5 & -2 & -2 & 1 & 0 \\ 0 & -2 & 1 & 0 & 0 & 1 \end{array} \right]$$

Step 2

Perform the operation $R_2 = \frac{1}{5}R_2$

$$\left[\begin{array}{ccc|ccc} 1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -2/5 & -2/5 & 1/5 & 0 \\ 0 & -2 & 1 & 0 & 0 & 1 \end{array} \right]$$

Step 3

Perform the operations $R_1 = R_1 + R_2$ and $R_3 = R_3 + 2R_2$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 3/5 & 3/5 & 1/5 & 0 \\ 0 & 1 & -2/5 & -2/5 & 1/5 & 0 \\ 0 & 0 & 1/5 & -4/5 & 2/5 & 1 \end{array} \right]$$

Step 4

Perform the operation $R_3 = 5R_3$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 3/5 & 3/5 & 1/5 & 0 \\ 0 & 1 & -2/5 & -2/5 & 1/5 & 0 \\ 0 & 0 & 1 & -4 & 2 & 5 \end{array} \right]$$

Step 5

Perform the operations $R_1 = R_1 - \frac{3}{5}R_3$ and $R_2 = R_2 + \frac{2}{5}R_3$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 3 & -1 & -3 \\ 0 & 1 & 0 & -2 & 1 & 2 \\ 0 & 0 & 1 & -4 & 2 & 5 \end{array} \right]$$

Therefore, the inverse of matrix A is

$$\left[\begin{array}{ccc} 3 & -1 & -3 \\ -2 & 1 & 2 \\ -4 & 2 & 5 \end{array} \right]$$

//C Program for Matrix Inversion by using Gauss-Jordan Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,k,j,p,q;
    float pivot,term, a[10][10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients Matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    for (i=0;i<n;i++) //Augment the matrix
    {
        for (j=n;j<2*n;j++)
        {
            if(i==j-n)
                a[i][j]= 1;
            else
                a[i][j]=0;
        }
    }
    for (k=0;k<n;k++) //Inverse Computation
    {
```

```

pivot=a[k][k];
for(p=0;p<2*n;p++)
a[k][p]=a[k][p]/pivot;
for (i=0;i<n;i++)
{
    term=a[i][k];
    if(k!=i)
    for(j=0;j<2*n;j++)
    {
        a[i][j]=a[i][j]-a[k][j]*term;
    }
}
printf("Matrix Inverse is:\n");
for(i=0;i<n;i++) //Display Inverse Matrix
{
    for(j=n;j<2*n;j++)
    printf("%f\t",a[i][j]);
    printf("\n");
}
getch();
return 0;
}
    
```

Output

Enter Dimension of System of equations

3

Enter coefficients Matrix

2 1 1

3 2 3

1 4 9

Matrix Inverse is:

-3.0 2.5 -0.50

12.0 -8.5 1.50

-5.0 3.5 -0.50

5.5 MATRIX FACTORIZATION

LU decomposition (also called LU factorization) factors a matrix as the product of a lower triangular matrix and an upper triangular matrix. Thus the coefficient matrix A of a system of linear equations can be decomposed into two triangular matrices L and U such that $A = LU$

Where,

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} l_{11}$$

L is known as lower triangular matrix and U is known as upper triangular matrix. Now System of linear equations $AX=C$ can be written as:

$$LUX=C.$$

Multiplying both sides by L^{-1} , we get

$$L^{-1} L UX = L^{-1} C$$

$$I UX = L^{-1} C$$

$$UX = L^{-1} C$$

$$\text{Since, } L^{-1} L = I$$

$$\text{Since } I UX = UX$$

Let,

$$L^{-1} C = Z$$

\Rightarrow

$$LZ = C$$

$$UX = Z$$

(1)

(2)

We can also use matrix factorization methods to solve system of linear equation. We can solve system of equations in two steps

1. Solve equation (1) first for $[Z]$ by using forward substitution
2. Use equation (2) to calculate the solution vector $[X]$ by back substitution.

5.5.1 Doolittle LU Decomposition

Coefficient matrix A of a system of linear equations can be decomposed into two triangular matrices L and U such that

$$[A] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

If L has 1's on its diagonal, then it is called a Doolittle factorization, thus Doolittle algorithms assumes that $\ell_{11} = 1$, $\ell_{22} = 1, \dots$ and $\ell_{nn} = 1$

From above matrices,

$$a_{11} = \ell_{11}u_{11} \Rightarrow u_{11} = a_{11} \quad \because \ell_{11} = 1$$

$$a_{12} = \ell_{11}u_{12} \Rightarrow u_{12} = a_{12} \quad \because \ell_{11} = 1$$

$$\vdots \quad \vdots$$

$$a_{1n} = \ell_{11}u_{1n} \Rightarrow u_{1n} = a_{1n} \quad \because \ell_{11} = 1$$

$$a_{21} = \ell_{21} u_{11} \Rightarrow \ell_{21} = \frac{a_{21}}{u_{11}} = \frac{a_{21}}{a_{11}}$$

$$a_{22} = \ell_{21} u_{12} + \ell_{22} u_{22} \Rightarrow u_{22} = \frac{a_{22} - \ell_{21} u_{12}}{\ell_{22}} = a_{22} - \ell_{21} u_{12} \quad \because \ell_{22} = 1$$

$$\vdots \\ a_{2n} = \ell_{21} u_{1n} + \ell_{22} u_{2n} \Rightarrow u_{2n} = \frac{a_{2n} - \ell_{21} u_{1n}}{\ell_{22}} = a_{2n} - \ell_{21} u_{1n} \quad \because \ell_{22} = 1$$

And

$$a_{n1} = \ell_{n1} u_{11} \Rightarrow \ell_{n1} = \frac{a_{n1}}{u_{11}} = \frac{a_{n1}}{a_{11}}$$

$$a_{n2} = \ell_{n1} u_{12} + \ell_{n2} u_{22} \Rightarrow \ell_{n2} = \frac{a_{n2} - \ell_{n1} u_{12}}{u_{22}} = \frac{1}{u_{22}} (a_{n2} - \ell_{n1} u_{12})$$

$$a_{n3} = \ell_{n1} u_{13} + \ell_{n2} u_{23} + \ell_{n3} u_{33} \Rightarrow \ell_{n3} = \frac{a_{n3} - \ell_{n1} u_{13} - \ell_{n2} u_{23}}{u_{33}} = \frac{1}{u_{33}} (a_{n3} - \ell_{n1} u_{13} - \ell_{n2} u_{23})$$

$$\vdots \\ a_{nn} = \ell_{n1} u_{1n} + \ell_{n2} u_{2n} + \dots + \ell_{nn} u_{nn} \Rightarrow u_{nn} = \frac{a_{nn} - \ell_{n1} u_{1n} - \ell_{n2} u_{2n} - \dots - \ell_{n,n-1} u_{n-1,n}}{\ell_{nn}} \\ \Rightarrow u_{nn} = (a_{nn} - \ell_{n1} u_{1n} - \ell_{n2} u_{2n} - \dots - \ell_{n,n-1} u_{n-1,n})$$

Generalizing this we get,

If $i \leq j$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj} \quad j = 1, 2, 3, \dots, n$$

Where, $u_{11} = a_{11}$, $u_{12} = a_{12}$, \dots , $u_{1n} = a_{1n}$

and if $i > j$

$$\ell_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right) \quad j = 1, 2, \dots, i-1$$

$$\text{Where, } \ell_{11} = 1, \ell_{22} = 1, \dots, \ell_{nn} = 1 \quad \text{and} \quad \ell_{ii} = \frac{a_{ii}}{u_{ii}}$$

Algorithm for Matrix Factorization using Doolittle LU Decomposition

1. Start
2. Read Dimension of Matrix, say n
3. Read elements of matrix row-wise
4. Assign values to first row of U matrix as below
For $j=1$ to n

- $u[1][j] = a[1][j]$
- End for
5. Assign values to first row of L matrix as below
- For $i=1$ to n
- $l[i][i] = 1$
- End for
6. Compute and assign values to first column of L matrix as below
- For $i=2$ to n
- $l[i][1] = a[i][1] / u[1][1]$
- End for
7. Compute and assign values to 2^{nd} to n^{th} rows of L and U matrix as below
8. For $j=2$ to n
- For $i=2$ to j
- $u[i][j] = a[i][j] - \sum_{k=1}^{j-1} (l[i][k] * u[k][j])$
- End for
- For $i=j+1$ to n
- $l[i][j] = \frac{1}{u[j][j]} \left[a[i][j] - \sum_{k=1}^{j-1} (l[i][k] * u[k][j]) \right]$
- End for
- End for
9. Display L and M matrix
10. Terminate

Example

Factorize the following matrix by using Dolittle LU decomposition

$$[A] = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}$$

Solution

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Where,

$$u_{11} = a_{11} = 25, \quad u_{12} = a_{12} = 5 \quad u_{13} = a_{13} = 1$$

$$\ell_{21} = \frac{a_{21}}{u_{11}} = \frac{64}{25} = 2.56$$

$$u_{22} = a_{22} - \ell_{21}u_{12} = 8 - 2.56 \times 5 = -4.8 \quad u_{23} = a_{23} - \ell_{21}u_{13} = 1 - 2.56 \times 1 = -1.56$$

$$\ell_{31} = \frac{a_{31}}{u_{11}} = \frac{144}{25} = 5.76 \quad \ell_{32} = \frac{1}{u_{22}}(a_{32} - \ell_{31}u_{12}) = \frac{12 - 5.76 \times 5}{4.8} = 3.5$$

$$u_{33} = (a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23}) = 1 - 5.76 \times 1 - 3.5 \times (-1.56) = 0.7$$

Now,

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

C program for Matrix Factorization using Do-Little LU Decomposition

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float sum=0, a[10][10],u[10][10],l[10][10];
    printf("Enter Dimension Matrix\n");
    scanf("%d",&n);
    printf("Enter Elements of Matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    // Compute Elements of L and U matrix
    for(j=0;j<n;j++)
        u[0][j]=a[0][j];
    for(i=0;i<n;i++)
        l[i][i]=1;
    for(i=1;i<n;i++)
        l[i][0]=a[i][0]/u[0][0];
    for(j=1;j<n;j++)
    {
        for(i=1;i<=j;i++)
    }
```

```

for(k=0;k<=i-1;k++)
{
    sum=sum+(l[i][k]*u[k][j]);
}
u[i][j]=a[i][j]-sum;
sum=0;
}

for(i=j+1;i<n;i++)
{
    for(k=0;k<=j-1;k++)
    {
        sum=sum+(l[i][k]*u[k][j]);
    }
    l[i][j]=(a[i][j]-sum)/u[j][j];
    sum=0;
}
printf("*****L Matrix*****\n");
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        printf("\t%d",l[i][j])
    }
    printf("\n");
}
printf("*****U Matrix*****\n");
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        printf("\t%d",u[i][j])
    }
    printf("\n");
}
getch();
return 0;
}

```

Output
Enter Dimension of System of equations

3
Enter Elements of Matrix

2 -1 -1

4 6 1

4 -2 6

*****L Matrix*****

1 0 0

-21 0

-2 -1 1

*****U Matrix*****

2 -1 -1

0 4 -1

0 0 3

Algorithm for Solving System of Equations using Dolittle LU Decomposition

1. Start
2. Read Dimension of Matrix, say n
3. Read coefficients of matrix row-wise
4. Read elements of B matrix
5. Assign values to first row of U matrix as below
For j=1 to n
 $u[1][j] = a[1][j]$
End for
6. Assign values to first row of L matrix as below
For i=1 to n
 $l[i][i] = 1$
End for
7. Compute and assign values to first column of L matrix as below
For i=2 to n
 $l[i][1] = a[i][1] / u[1][1]$
End for
8. Compute and assign values to 2nd to nth rows of L and U matrix as below
For j=2 to n
 For i=2 to j
 $u[i][j] = a[i][j] - \sum_{k=1}^{j-1} (l[i][k] * u[k][j])$
 End for
 For i=j+1 to n

$$l[i][j] = \frac{1}{u[j][j]} \left[a[i][j] - \sum_{k=1}^{j-1} (l[i][k] * u[k][j]) \right]$$

End for

End for

10. Set $z[1]=b[1]$

11. Use forward substitution to calculate values of Z vector as below

For i=2 to n

$$z[i] = b[i] - \sum_{j=1}^{i-1} (l[i][j] * z[j])$$

End for

12. Set $x[n]=z[n]/u[n][n]$

13. Use backward substitution to calculate values of Z vector as below

For i=n-1 to 1

$$x[i] = (z[i] - \sum_{j=i+1}^n (u[i][j] * x[j])) / u[i][i]$$

End for

14. Display solution vector

15. Terminate

Example

Solve the following system of equations by using Doolittle LU decomposition method

$$3x_1 + 2x_2 + x_3 = 10$$

$$2x_1 + 3x_2 + 2x_3 = 14$$

$$x_1 + 2x_2 + 3x_3 = 14$$

Solution

Coefficient Matrix

$$[A] = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

Decompose the coefficient matrix using Doolittle decomposition as below:

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Where,

$$u_{11} = a_{11} = 3, \quad u_{12} = a_{12} = 2, \quad u_{13} = a_{13} = 1$$

$$\ell_{21} = \frac{a_{21}}{u_{11}} = \frac{2}{3}$$

$$u_{22} = a_{22} - \ell_{21}u_{12} = 3 - \frac{2}{3} \times 2 = \frac{5}{3} \quad u_{23} = 2 - \frac{2}{3} = \frac{4}{3}$$

$$\ell_{31} = \frac{a_{31}}{u_{11}} = \frac{1}{3} \quad \ell_{32} = \frac{1}{u_{22}}(a_{32} - \ell_{31}u_{12}) = \frac{2 - 2/3}{5/3} = \frac{4}{5}$$

$$u_{33} = (a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23}) = 3 - \frac{1}{3} - \frac{16}{15} = \frac{24}{15} = \frac{8}{5}$$

Thus,

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 4/5 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 & 1 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 8/5 \end{bmatrix}$$

Now, Solve $[L][Z] = [C]$ by using forward substitution. That is, solve

$$\Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 4/5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \\ 14 \end{bmatrix}$$

We get,

$$z_1 = 10$$

$$z_2 = 14 - 2/3 \times z_1 = 22/3$$

$$z_3 = 14 - 1/3 \times z_1 - 4/5 \times z_2 = 14 - 10/3 - 88/15 = 72/15 = 24/5$$

Again solve $[U][X] = [Z]$ by using backward substitution.

$$\begin{bmatrix} 3 & 2 & 1 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 8/5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 22/3 \\ 24/5 \end{bmatrix}$$

We get,

$$x_3 = 3$$

$$x_2 = \frac{22/3 - 4/3 \times 3}{5/3} = 2$$

$$x_1 = \frac{10 - 2 \times 2 - 1 \times 3}{3} = 1$$

//C program for Solving System of Linear Equations using Doolittle LU Decomposition

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float sum=0, a[10][10],b[10],x[10],z[10],u[10][10],l[10][10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients Matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    printf("Enter RHS vector\n");
    for(i=0;i<n;i++)
    {
        scanf("%f",&b[i]);
    }

    // Compute Elements of L and U matrix
    for(j=0;j<n;j++)
        u[0][j]=a[0][j];
    for(i=1;i<n;i++)
    {
        l[i][i]=1;
        for(j=1;j<n;j++)
            l[i][j]=a[i][j]/u[0][0];
        for(j=1;j<n;j++)
        {
            for(i=1;i<=j;i++)
            {
                for(k=0;k<=i-1;k++)
                {
                    sum=sum+(l[i][k]*u[k][j]);
                }
                u[i][j]=a[i][j]-sum;
                sum=0;
            }
        }
        for(i=j+1;i<n;i++)
    }
```

```

    {
        for(k=0;k<=j-1;k++)
        {
            sum=sum+(l[i][k]*u[k][j]);
        }
        l[i][j]=(a[i][j]-sum)/u[j][j];
        sum=0;
    }
}

// Solve for Z using Forward Substitution
z[0]=b[0];
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    sum=sum+(l[i][j]*z[j]);
    z[i]=b[i]-sum;
    sum=0;
}

//Solve for X using Backward Substitution
x[n-1]=z[n-1]/u[n-1][n-1];
for(i=n-2;i>=0;i--)
{
    for(j=i+1;j<n;j++)
    sum=sum+(u[i][j]*x[j]);
    x[i]=(z[i]-sum)/u[i][i];
    sum=0;
}
printf("Solution:\n");
for(i=0;i<n;i++) //Display Solution Vector
{
    printf("%d=%f\n",i+1,x[i]);
}
getch();
return 0;
}

```

Output

Enter Dimension of System of equations

3

Enter coefficients Matrix

2	3	1
---	---	---

1 2 3
3 1 2

Enter RHS vector

9 6 8

Solution:

x1=1.944445

x2=1.611111

x3=0.277778

5.5.2 Cholesky Method

For many practical systems of linear equations, the coefficient matrix $[A]$ is symmetric. In such case, Cholesky method can be used to factorize the coefficient matrix and to solve systems of linear equations in more efficient way. If coefficient matrix is symmetric the upper factor is transpose of lower factor or vice versa. Thus, we can factorize matrix A as:

$$[A] = [L][L^T] \quad \text{or} \quad [A] = [U^T][U]$$

Thus,

$$[A] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} u_{11} & 0 & \dots & 0 \\ u_{12} & u_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u_{1n} & u_{2n} & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

Like Doolittle LU decomposition, multiplying the matrices of right hand side and comparing it with coefficient matrix of left hand side, we can get

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2} \quad i = 1, 2, \dots, n$$

$$u_{ij} = \frac{1}{u_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj} \right) \quad j > i$$

Algorithm for Cholesky Matrix Factorization

1. Start
2. Read dimension of matrix, say n
3. Read elements of matrix row-wise
4. Compute and assign elements of U matrix as below:
For i=1 to n

$$u[i][i] = \sqrt{a[i][i] - \sum_{k=1}^{i-1} (u[k][i] * u[k][i])}$$

End for

For i=1 to n

For j=i+1 to n

$$u[i][j] = \frac{1}{u[i][i]} a[i][j] - \sum_{k=1}^{i-1} u[k][i] * u[k][j]$$

- End for
 End for
 5. Set [L] = Transpose of [U]
 6. Display L and U matrix
 7. Terminate

Example
 Factorize the given matrix Cholesky decomposition.

$$[A] = \begin{bmatrix} 1 & 4 & 7 \\ 4 & 80 & 44 \\ 7 & 44 & 89 \end{bmatrix}$$

Solution

Let,

$$[A] = [L][U] = \begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Where,

$$u_{11} = \sqrt{a_{11}} = \sqrt{1} = 1, \quad u_{12} = \frac{a_{12}}{u_{11}} = \frac{4}{1} = 4, \quad u_{13} = \frac{a_{13}}{u_{11}} = \frac{7}{1} = 7$$

$$u_{22} = \sqrt{a_{22} - u_{12}^2} = \sqrt{80 - 16} = 8 \quad u_{23} = \frac{a_{23} - u_{12}u_{13}}{u_{22}} = \frac{44 - 4 \times 7}{8} = 2$$

$$u_{33} = \sqrt{a_{33} - u_{13}^2 - u_{23}^2} = \sqrt{89 - 49 - 4} = \sqrt{36} = 6$$

Thus, two factors of coefficient matrix are:

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 8 & 0 \\ 7 & 2 & 6 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & 8 & 2 \\ 0 & 0 & 6 \end{bmatrix}$$

C program to factorize the matrix by using Cholesky decomposition

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    float a[10][10], l[10][10], u[10][10];
    int i=0, j=0, k=0, n;
    float temp=0;
    printf("Enter Dimension of Matrix\n");
    scanf("%d", &n);
}
```

```

printf("Enter Elements of Matrix\n");
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
}

//Calculate Elements of U matrix
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        temp=0;
        if(i==j)
        {
            for(k=0;k<j;k++)
            {
                temp+=((u[k][i])*(u[k][i]));
            }
            u[i][i]=sqrt((a[i][i]-temp));
        }
        else if(j>i)
        {
            for(k=0;k<i;k++)
            {
                temp+=((u[k][i]*u[k][j]));
            }
            u[i][j]=(1/(u[i][i]))*((a[i][j])-temp);
        }
        else
        {
            u[i][j]=0;
        }
    }
}

// Find Transpose
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
}

```

```

    l[i][j]=u[j][i];
}

printf("*****L Matrix*****\n");
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        printf("\t%f",l[i][j]);
    }
    printf("\n");
}

printf("*****U Matrix*****\n");
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        printf("\t%f",u[i][j]);
    }
    printf("\n");
}

 getch();
 return 0;
}

```

Output

Enter Dimension of Matrix

3

Enter Elements of Matrix

1	2	3
2	8	22
3	22	82

*****L Matrix*****

1.00	0.00	0.00
2.00	2.00	0.00
3.00	8.00	3.00

*****U Matrix*****

1.00	2.00	3.00
0.00	2.00	8.00
0.00	0.00	3.00

Algorithm for Solving System of Linear Equations by Using Cholesky Factorization

1. Start
2. Read dimension of matrix, say n
3. Read coefficients of matrix row-wise
4. Read elements of RHS vector
5. Compute elements of U matrix as below

For i=1 to n

$$u[i][i] = \sqrt{a[i][i] - \sum_{k=1}^{i-1} (u[k][i] * u[k][i])}$$

End for

For i=1 to n

For j=i+1 to n

$$u[i][j] = \frac{1}{u[i][i]} a[i][j] - \sum_{k=1}^{i-1} u[k][i] * u[k][j]$$

End for

End for

6. Set [L]= Transpose of [U]

7. Set z[1]=b[1]

8. Find Z vector as below

For i=2 to n

$$z[i] = b[i] - \sum_{j=1}^{i-1} (l[i][j] * z[j])$$

End for

9. Set x[n]=z[n]/u[n][n]

10. Find Solution vector as below

For i=n-1 to 1

$$x[i] = (z[i] - \sum_{j=i+1}^n (u[i][j] * x[j])) / u[i][i]$$

End for

11. Display solution vector

12. Terminate

Example

Solve following system of equations by using Cholesky decomposition technique.

$$4x_1 + 10x_2 + 8x_3 = 44$$

$$10x_1 + 26x_2 + 26x_3 = 128$$

$$8x_1 + 26x_2 + 61x_3 = 214$$

Solution
Let,

$$[A] = [L][U] = \begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Where,

$$u_{11} = \sqrt{a_{11}} = \sqrt{4} = 2$$

$$u_{12} = \frac{a_{12}}{u_{11}} = \frac{10}{2} = 5$$

$$u_{13} = \frac{a_{13}}{u_{11}} = \frac{8}{2} = 4$$

$$u_{22} = \sqrt{a_{22} - u_{12}^2} = \sqrt{26 - 5^2} = 1$$

$$u_{23} = \frac{a_{23} - u_{12}u_{13}}{u_{22}} = \frac{26 - 5 \times 4}{1} = 6$$

$$u_{33} = \sqrt{a_{33} - u_{13}^2 - u_{23}^2} = \sqrt{61 - 4^2 - 6^2} = 3$$

Thus, two factors of coefficient matrix are:

$$[A] = [L][U] = \begin{bmatrix} 2 & 0 & 0 \\ 5 & 1 & 0 \\ 4 & 6 & 3 \end{bmatrix} \begin{bmatrix} 2 & 5 & 1 \\ 0 & 1 & 6 \\ 0 & 0 & 3 \end{bmatrix}$$

Now Solve the Equation

$$[L][Z] = [C]$$

∴

$$\begin{bmatrix} 2 & 0 & 0 \\ 5 & 1 & 0 \\ 4 & 6 & 3 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 44 \\ 128 \\ 214 \end{bmatrix}$$

Thus, we can get

$$z_1 = 22 \quad z_2 = 18 \quad z_3 = 6$$

Now Solve the Equation

∴ $[U][X] = [Z]$

$$\begin{bmatrix} 2 & 5 & 4 \\ 0 & 1 & 6 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 22 \\ 18 \\ 6 \end{bmatrix}$$

Thus, we can get

$$x_1 = -8 \quad x_2 = 6 \quad x_3 = 2$$

//C program for solving system of equations by using Cholesky Decomposition

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    float a[10][10], l[10][10], u[10][10], x[10], z[10], b[10];
    int i=0, j=0, k=0, n;
    float temp=0;
    printf("Enter Dimension of Matrix\n");
    scanf("%d", &n);
    printf("Enter coefficients Matrix\n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
    {
        scanf("%f", &a[i][j]);
    }

    printf("Enter RHS vector\n");
    for(i=0; i<n; i++)
    {
        scanf("%f", &b[i]);
    }

    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            temp=0;
            if(i==j)
            {
                for(k=0; k<j; k++)
                {
                    temp+=((u[k][i])*(u[k][i]));
                }
                u[i][i]=sqrt((a[i][i]-temp));
            }
            else if(j>i)
            {

```

```

for(k=0;k<i;k++)
{
    temp+=((u[k][i]*u[k][j]));
}
u[i][j]=(1/(u[i][i]))*((a[i][j])-temp);
}

else
{
    u[i][j]=0;
}
}

//Find Transpose

for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        l[i][j]=u[j][i];
    }
}

//Solve for Z using Forward Substitution
z[0]=b[0];
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    sum=sum+(l[i][j]*z[j]);
    z[i]=b[i]-sum;
    sum=0;
}

//Solve for X using Backward Substitution
x[n-1]=z[n-1]/u[n-1][n-1];
for(i=n-2;i>=0;i--)
{
    for(j=i+1;j<n;j++)
    sum=sum+(u[i][j]*x[j]);
    x[i]=(z[i]-sum)/u[i][i];
    sum=0;
}

```

```

    }
    printf("Solution:\n");
    for(i=0;i<n;i++) //Display Solution Vector
    {
        printf("%d=%f\t",i+1,x[i]);
    }

    getch();
    return 0;
}

```

Output

Enter Dimension of Matrix

3

Enter coefficients Matrix

4	10	8
10	26	26
8	26	61

Enter RHS vector

44	128	214
----	-----	-----

Solution:

x0=-8	x1=6	x2=2
-------	------	------

5.6 ITERATIVE METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

Iterative methods for solving general, large sparse linear systems have been gaining popularity in many areas of scientific computing. Iterative methods are gaining ground because they are easier to implement efficiently on high-performance computers than direct methods. Because of round-off errors, direct methods become less efficient than iterative methods when they applied to large systems. In addition, the amount of storage space required for iterative solutions on computer is far less than the one required for direct methods when the coefficient matrix of the system is sparse. Thus, especially for sparse matrices, iterative methods are more attractive than direct methods.

For the iterative solution of a system of equations, one starts with an arbitrary starting vector x_0 and computes a sequence of iterates x_m for $m=1,2,\dots$

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_m \rightarrow x_{m+1} \dots$$

In the following x_{m+1} is dependent only on x_m , so that the mapping $x_m \rightarrow x_{m+1}$ determines the iteration method.

5.6.1 Jacobi Iteration Method

Jacobi method or Jacobi iteration method is an algorithm for determining the solutions of a diagonally dominant system of linear equations. That is, it used to solve system of equations in which diagonal elements of coefficient matrix are non-zero. Each diagonal element is solved for, and an approximate value is determined. The process is then iterated until it converges. Let A be an $n \times n$ nonsingular matrix and $Ax=b$ is the system to be solved. That is, we have to solve the system of equations

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

\vdots

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

This can be rewritten as:

$$x_1 = \frac{b_1 - (a_{12}x_2 + \dots + a_{1n}x_n)}{a_{11}}$$

$$x_2 = \frac{b_2 - (a_{21}x_1 + a_{23}x_3 + \dots + a_{2n}x_n)}{a_{22}}$$

\vdots

$$x_n = \frac{b_n - (a_{n1}x_1 + a_{n2}x_2 + \dots + a_{n-1}x_{n-1})}{a_{nn}}$$

Now we can compute values of $x_1, x_2, x_3, \dots, x_n$ by using initial guess.

Then Jacobi's iteration method is used to find new values of unknown by using previously computes values as below:

$$x_i^{k+1} = \frac{b_i - (a_{i1}x_1^k + \dots + a_{i(i-1)}x_{i-1}^k + a_{i(i+1)}x_{i+1}^k + \dots + a_{in}x_n^k)}{a_{ii}} = \frac{(b_i - \sum_{j \neq i} a_{ij}x_j^k)}{a_{ii}} \quad k = 0, 1, \dots$$

At the end of each iteration, the absolute relative approximate error for each x_i is calculated as below:

$$|e_a|_i = \left| \frac{x_i^{\text{new}} - x_i^{\text{old}}}{x_i^{\text{new}}} \right| \times 100$$

Where, x_i^{new} is the recently obtained value of x_i and x_i^{old} is the previous value of x_i . When the absolute relative approximate error for each x_i is less than the pre-specified tolerance, the iterations are stopped.

Algorithm for Jacobi Iteration Method

1. Start
2. Read dimension of System of equations, say n
3. Read coefficients of matrix row-wise

4. Read elements of RHS vector
5. Read accuracy limit, say ϵ
6. Read initial guesses
- For $i=1$ to n
 $old_x[i]=0$ //sets initial guess
- End for
7. Compute new set of approximate roots as below
- For $i=1$ to n
 $sum=b[i]$
For $j=1$ to n
if($i \neq j$)
 $sum=sum-a[i][j]*old_x[j]$
- End for
 $new_x[i]=sum/a[i][i]$
 $E[i]=\left|\frac{new_x[i]-old_x[i]}{new_x[i]}\right|$
- End for
8. Compare error with specified precision
For $i=1$ to n
if($E[i] > \epsilon$)
for $j=1$ to n
 $old_x[j]=new_x[j]$
- End for
go to step 7
- End for
9. Display Results in new_x vector
10. Terminate

Example

Use the Jacobi iteration method to obtain the solution of the following equations

$$6x_1 - 2x_2 + x_3 = 11$$

$$x_1 + 2x_2 - 5x_3 = -1$$

$$-2x_1 + 7x_2 + 2x_3 = 5$$

Solution**Step 1**

Re-write the equations such that each equation has the unknown with largest coefficient on the left hand side:

$$x_1 = \frac{2x_2 - x_3 + 11}{6}$$

$$x_2 = \frac{2x_1 - 2x_3 + 5}{7}$$

$$x_3 = \frac{x_1 + 2x_2 + 1}{5}$$

Step 2

Assume the initial guesses $(x_1)^0 = (x_2)^0 = (x_3)^0 = 0$, then calculate $(x_1)^1$, $(x_2)^1$ and $(x_3)^1$:

$$(x_1)^1 = \frac{2(x_2)^0 - (x_3)^0 + 11}{6} = \frac{2 \times 0 - 0 + 11}{6} = 1.833$$

$$(x_2)^1 = \frac{2(x_1)^0 - 2(x_3)^0 + 5}{7} = \frac{2 \times 0 - 2 \times 0 + 5}{7} = 0.714$$

$$(x_3)^1 = \frac{(x_1)^0 + 2(x_2)^0 + 1}{5} = \frac{0 + 2 \times 0 + 1}{5} = 0.200$$

Step 3

Use the values obtained in the first iteration, to calculate the values for the 2nd iteration:

$$(x_1)^2 = \frac{2(x_2)^1 - (x_3)^1 + 11}{6} = \frac{2 \times 0.714 - 0.2 + 11}{6} = 2.038$$

$$(x_2)^2 = \frac{2(x_1)^1 - 2(x_3)^1 + 5}{7} = \frac{2 \times 1.833 - 2 \times 0.2 + 5}{7} = 1.181$$

$$(x_3)^2 = \frac{(x_1)^1 + 2(x_2)^1 + 1}{5} = \frac{1.833 + 2 \times 0.714 + 1}{5} = 0.852$$

And so on for the next iterations such that the next values are calculated using the current values:

$$(x_1)^{i+1} = \frac{2(x_2)^i - (x_3)^i + 11}{6} \quad (x_2)^{i+1} = \frac{2(x_1)^i - 2(x_3)^i + 5}{7}$$

$$(x_3)^{i+1} = \frac{(x_1)^i + 2(x_2)^i + 1}{5} \text{ and continue the above iterative procedure until}$$

$\{(x_k)^{i+1} - (x_k)^i\}/(x_k)^{i+1} < \epsilon$ for $k=1,2$ and 3 . The results for 9 iterations are:

Iteration \ Values	x_1	x_2	x_3	E_1	E_2	E_3
1	1.833	0.714	0.200	1	1	1
2	2.038	1.085	0.852	0.1004	0.3951	0.7653
3	2.004	1.001	1.038	0.0224	0.1214	0.2107
4	1.994	0.990	1.001	0.0401	0.0515	0.0402
5	1.997	0.998	0.999	0.0051	0.0111	0.0367
6	1.999	0.999	0.999	0.0012	0.00759	0.0065
7	1.999	1.000	0.9994	0.0018	0.0025	0.0035
8	1.9998	1.000	0.9999	0.0001	0.0000	0.0017
9	2.000	1.000	1.000	0.0000	0.0000	0.0005
Thus, solution is: $x_1 = 1$ $x_2 = 1$ and $x_3 = 1$.						

Second Example

Use the Jacobi iteration method to obtain the solution of the following system of equations

$$5x_1 - 2x_2 + 3x_3 = -1$$

$$-3x_1 + 9x_2 + x_3 = 2$$

$$2x_1 - x_2 - 7x_3 = 3$$

Solution

From given equations, we can obtain

$$x_1 = -\frac{1}{5} + \frac{2}{5}x_2 - \frac{3}{5}x_3$$

$$x_2 = \frac{2}{9} + \frac{3}{9}x_1 - \frac{1}{9}x_3$$

$$x_3 = \frac{3}{7} - \frac{2}{7}x_1 + \frac{1}{7}x_2$$

Let, initial guesses are $x_1 = 0$, $x_2 = 0$, and $x_3 = 0$

Iteration \ Values	x_1	x_2	x_3	E_1	E_2	E_3
1	-0.200	0.222	-0.429	1	1	1
2	0.146	0.203	-0.517	2.3736	0.0940	0.1713
3	0.192	0.328	-0.416	0.2380	0.3810	0.2436
4	0.180	0.332	-0.420	0.0596	0.0011	0.0113
5	0.185	0.329	-0.428	0.0239	0.0092	0.0085
6	0.186	0.330	-0.422	0.0051	0.0056	0.0040
7	0.185	0.330	-0.422	0.0014	0.0003	0.0000
8	0.185	0.330	-0.422	0.0002	0.0002	0.0000

Thus, solution is: $x_1 = 0.185$, $x_2 = 0.330$, and $x_3 = -0.422$

//C program for Jacobi Iteration method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float sum,error,E[10],a[10][10],b[10],new_x[10],old_x[10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients row-wise\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%f",&a[i][j]);
    printf("Enter RHS vector\n");
}
```

```

for(i=0;i<n;i++)
{
    scanf("%f",&b[i]);
}
printf("Enter Accuracy Limit\n");
scanf("%f",&error);
for(i=0;i<n;i++)
    old_x[i]=0;
while (1)
{
    for(i=0;i<n;i++)
    {
        sum=b[i];
        for(j=0;j<n;j++)
        {
            if(i!=j)
                sum=sum-a[i][j]*old_x[j];
        }
        new_x[i]=sum/a[i][i];
        E[i]=fabs(new_x[i]-old_x[i])/fabs(new_x[i]);
    }
    for(i=0;i<n;i++)
    {
        if(E[i]>error)
        {
            for(j=0;j<n;j++)
                old_x[j]=new_x[j];
            break;
        }
    }
    if(i==n) // Specified level of accuracy is achieved
        break;
    else
        continue;
}
printf("Solution:\n");
for(i=0;i<n;i++)
    printf("%6.2f\t",new_x[i]);
getch();
return 0;

```

6 Numerical Methods with Practical Approach

utput

Enter Dimension of System of equations

4

Enter coefficients row-wise

10 -2 -1 -1
-2 10 -1 -1
-1 -1 10 -2
-1 -1 -2 10

Enter RHS vector

3 15 27 -9

Enter Accuracy Limit

0.01

Solution:

x1=1.00 x2=2.00 x3=3.00 x4=0.00

5.6.2 Gauss Seidel Method

Gauss Seidel method is also an iterative approach of solving system of linear equations. It is similar to Jacobi Iteration method only the difference is that with the Jacobi method, the values of $x_i^{(k)}$ obtained in the k^{th} iteration remains unchanged until the entire $(k+1)^{\text{th}}$ iteration has been calculated. But with the Gauss-Seidel method, we use the new values as soon as they are known.

Given a general set of n equations and n unknowns, we have

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2$$

$$b_{n1}x_1 + b_{n2}x_2 + b_{n3}x_3 + \dots + b_{nn}x_n = b_n$$

If the diagonal elements are non-zero, each equation is rewritten for the corresponding unknowns. That is, the first equation is rewritten with x_1 on the left hand side, the second equation is rewritten with x_2 on the left hand side and so on as follows

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n}{a_{11}}$$

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n}{a_{22}}$$

$$\vdots$$

$$\vdots$$

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,1}x_1 - a_{n-1,2}x_2 - \dots - a_{n-1,n-2}x_{n-2} - a_{n-1,n}x_n}{a_{n-1,n-1}}$$

$$x_n = \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \dots - a_{n,n-1}x_{n-1}}{a_{nn}}$$

Now we start with initial guesses and use above equations to find new estimates for x_i 's. Remember, unlike Jacobi iteration method Gauss-Seidel method always uses the most recent estimates to calculate the next estimates for x_i . This means Gauss-Seidel method is used to find new values of unknown as below:

$$x_i^{k+1} = \frac{b_i - (a_{i1}x_1^{k+1} + \dots + a_{i,i-1}x_{i-1}^{k+1} + a_{i,i+1}x_{i+1}^k + \dots + a_{in}x_n^k)}{a_{ii}} \quad k = 0, 1, \dots$$

At the end of each iteration, the absolute relative approximate error for each x_i is calculated as below:

$$|e_a|_i = \left| \frac{x_i^{\text{new}} - x_i^{\text{old}}}{x_i^{\text{new}}} \right| \times 100$$

Where x_i^{new} is the recently obtained value of x_i , and x_i^{old} is the previous value of x_i . When the absolute relative approximate error for each x_i is less than the pre-specified tolerance, the iterations are stopped.

Algorithm for Gauss-Seidel Method

1. Start
2. Read dimension of System of equations, say n
3. Read coefficients of matrix row-wise
4. Read elements of RHS vector
5. Read accuracy limit, say ϵ
6. Set initial guesses as below
For i=1 to n
 new_x[i]=0

7. End for
Compute new estimates as below
For i=1 to n

 sum=b[i]

 For j=1 to n

```

if(i≠j)
    sum=sum-a[i][j]* new_x[j]

```

End for

old_x[i]=new_x[i]

new_x[i]=sum/a[i][i]

$$E[i] = \left| \frac{\text{new_x}[i] - \text{old_x}[i]}{\text{new_x}[i]} \right|$$

End for

8. Check for tolerance level as below

For i=1 to n

if($|E[i]| > \epsilon$)

go to step 7

End for

9. Display Results in "new_x" vector

10. Terminate

Example

Use the Gauss Seidel method to obtain the solution of the following equations:

$$6x_1 - 2x_2 + x_3 = 11$$

$$x_1 + 2x_2 - 5x_3 = -1$$

$$-2x_1 + 7x_2 + 2x_3 = 5$$

Solution

Step 1

Re-write the equations such that each equation has the unknown with largest coefficient on the left hand side (not necessary always):

$$x_1 = \frac{2x_2 - x_3 + 11}{6} \quad x_2 = \frac{2x_1 - 2x_3 + 5}{7} \quad x_3 = \frac{x_1 + 2x_2 + 1}{5}$$

Step 2:

Assume the initial guesses $(x_2)^0 = (x_3)^0 = 0$, then calculate $(x_1)^1$:

$$(x_1)^1 = \frac{2(x_2)^0 - (x_3)^0 + 11}{6} = \frac{2(0) - (0) + 11}{6} = 1.833$$

Use $(x_1)^1 = 1.833$ and $(x_3)^0 = 0$ to calculate $(x_2)^1$

$$(x_2)^1 = \frac{2(x_1)^1 - 2(x_3)^0 + 5}{7} = \frac{2(1.833) - 2(0) + 5}{7} = 1.238$$

Similarly, use $(x_1)^1 = 1.833$ and $(x_2)^1 = 1.238$ to calculate $(x_3)^1$

$$(x_3)^1 = \frac{(x_1)^1 + 2(x_2)^1 + 1}{5} = \frac{(1.833) + 2(1.238) + 1}{5} = 1.062$$

Step 3:
Repeat the same procedure for the 2nd iteration

$$(x_1)^2 = \frac{2(x_2)^1 - (x_3)^1 + 11}{6} = \frac{2(1.238) - (1.062) + 11}{6} = 2.069$$

$$(x_2)^2 = \frac{2(x_1)^2 - 2(x_3)^1 + 5}{7} = \frac{2(2.069) - 2(1.062) + 5}{7} = 1.002$$

$$(x_3)^2 = \frac{(x_1)^2 + 2(x_2)^2 + 1}{5} = \frac{(2.069) + 2(1.002) + 1}{5} = 1.015$$

And continue the above iterative procedure until $\{(x_k)^{i+1} - (x_k)^i\}/(x_k)^{i+1} < \epsilon$ for k=1,2 and 3.
The results for 5 iterations are:

Iteration \ Values	x ₁	x ₂	x ₃	E ₁	E ₂	E ₃
1	1.833	1.238	1.062	1	1	1
2	2.069	1.002	1.015	0.1141	0.2355	0.0463
3	1.998	0.995	0.998	0.0355	0.0251	0.0170
4	1.999	1.000	1.000	0.0005	0.005	0.002
5	2.000	1.000	1.000	0.0010	0	0

Thus, solution is: x₁ = 2 x₂ = 1 and x₃ = 1

Second Example

Use the Gauss Seidel method to obtain the solution of the following system of equations

$$5x_1 - 2x_2 + 3x_3 = -1$$

$$-3x_1 + 9x_2 + x_3 = 2$$

$$2x_1 - x_2 - 7x_3 = 3$$

Solution

From given equations, we can obtain

$$x_1 = -\frac{1}{5} + \frac{2}{5}x_2 - \frac{3}{5}x_3$$

$$x_2 = \frac{2}{9} + \frac{3}{9}x_1 - \frac{1}{9}x_3$$

$$x_3 = \frac{3}{7} - \frac{2}{7}x_1 + \frac{1}{7}x_2$$

Initial guesses are x₁ = 0 x₂ = 0 and x₃ = 0

Values	x_1	x_2	x_3	E_1	E_2	E_3
Iteration						
0	0	0	0	1	1	1
1	-0.200	0.1554	-0.507	2.2019	0.5342	0.1848
2	0.1664	0.334	-0.428	0.1253	0.00251	0.0146
3	0.190	0.333	-0.421	0.0241	0.0068	0.0022
4	0.185	0.331	-0.422	0.0017	0.0001	0.0002
5	0.185	0.331	-0.422			

Thus, solution is: $x_1 = 0.185$ $x_2 = 0.330$ and $x_3 = -0.422$

//Program 5.8: C program for Gauss-Seidal Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float sum,error, E[10],a[10][10],b[10],new_x[10],old_x[10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients row-wise\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
    {
        scanf("%f",&a[i][j]);
    }
    printf("Enter B vector\n");
    for(i=0;i<n;i++)
    {
        scanf("%f",&b[i]);
    }
    printf("Enter Accuracy Limit\n");
    scanf("%f",&error);

    for(i=0;i<n;i++)
    {
        new_x[i]=0;
    }

    while (1)
    {
        for(i=0;i<n;i++)
        {
            sum=0;
            for(j=0;j<n;j++)
                if(j!=i)
                    sum+=a[i][j]*new_x[j];
            new_x[i]=(b[i]-sum)/a[i][i];
        }
        sum=0;
        for(i=0;i<n;i++)
            sum+=abs(new_x[i]-old_x[i]);
        if(sum<=error)
            break;
        old_x=new_x;
    }
}
```

```

    sum=b[i];
    for(j=0;j<n;j++)
    {
        if(i!=j)
            sum=sum-a[i][j]*new_x[j];
    }
    old_x[i]=new_x[i];
    new_x[i]=sum/a[i][i];
    E[i]=fabs(new_x[i]-old_x[i])/fabs(new_x[i]);
}
for(i=0;i<n;i++)
{
    if(E[i]>error)
        break;
}
if(i==n)
    break;
else
    continue;
}
printf("Solution:\n");
for(i=0;i<n;i++)
    printf("x%d=%6.2f\n",i+1,new_x[i]);
getch();
return 0;
}

```

Output

Enter Dimension of System of equations

3

Enter coefficients row-wise

2 1 1

3 5 2

2 1 4

Enter B vector

5 15 8

Enter Accuracy Limit

0.01

Solution:

x1= 1.00 x2= 2.00 x3= 1.00

5.7 ILL-CONDITIONING

Certain systems of linear equations are such that their solutions are very sensitive to small changes (and therefore to errors) in their coefficients and constants. We give an example below in which 1 % changes in two coefficients change the solution by a factor of 10 or more. Such systems are said to be **ill-conditioned**. If a system is ill-conditioned, a solution obtained by a numerical method may differ greatly from the exact solution, even though great care is taken to keep round-off and other errors very small. As an example, consider the system of equations is:

$$\begin{aligned} 2x + y &= 4 \\ 2x + 1.01y &= 4.02 \end{aligned}$$

Which has the exact solution $x = 1, y = 2$. Changing coefficients of the second equation by 1% and the constant of the first equation by 5% yields the system:

$$\begin{aligned} 2x + y &= 3.8 \\ 2.02x + y &= 4.02 \end{aligned}$$

It is easily verified that the exact solution of this system is $x = 11, y = -18.2$. This solution differs greatly from the solution of the first system. Both these systems are said to be ill-conditioned.

If a system is ill-conditioned, then the usual procedure of checking a numerical solution by calculation of the residuals may not be valid. In order to see why this is so, suppose we have an approximation s to the true solution x . The vector of residuals r is then given by $r = b - As = A(x - s)$. Thus $e = x - s$ satisfies the linear system $Ae = r$. In general, r will be a vector with small components. However, in an ill-conditioned system, even if the components of r are small so that it is close to the solution of the linear system $Ae = r$ could differ greatly from the solution of the system $Ae = 0$, namely 0. It then follows that s may be a poor approximation to x despite the residuals in r being small.

5.8 EIGENVECTORS AND EIGENVALUES OF A MATRIX

The eigenvectors of a square matrix are the non-zero vectors that always remains proportional to original vector when multiplied by the matrix. This means any vector \mathbf{x} that satisfies the equation: $\mathbf{Ax} = \lambda\mathbf{x}$, is called eigenvector. Here \mathbf{A} is the matrix in question, \mathbf{x} is the eigenvector and λ is the associated eigenvalue. Eigenvectors of a matrix \mathbf{A} are exceptional vectors \mathbf{x} that are in the same direction as that of \mathbf{Ax} . Eigenvectors are not unique in the sense that any eigenvector can be multiplied by a constant to form another eigenvector. For each eigenvector there is only one associated eigenvalue, however.

5.8.1 Power Method

Eigenvalues can be ordered in terms of their absolute values to find the dominant or largest eigenvalue of a matrix. Thus, if two distinct hypothetical matrices have the following set of eigenvalues

5.8.7; then $|8| > |-7| > |5|$ and 8 is the dominant eigenvalue.

0.2, -1, 1; then $|1| = |-1| > |0.2|$ and since $|1| = |-1|$ there is no dominant eigenvalue.

One of the simplest methods for finding the largest eigenvalue and eigenvector of a matrix is the Power Method, also called the Vector Iteration Method. The method fails if there is no dominant eigenvalue. It uses iterative approach and starts with initial guess for vector x . Power method can be implemented as follows:

$$Y = AX \dots \dots \dots (1)$$

$$X = \frac{1}{k} Y \dots \dots \dots \quad (2)$$

New value of X can be obtained by using equation (2) and then value of Y is calculated by using equation (1). This process is repeated until the desired level of accuracy is obtained. The parameter k is known as sealing factor and it is the element of Y with the largest magnitude.

Inverse power method, which is similar to power method, is used to calculate the smallest eigenvalue and its corresponding eigenvector. Here, we simply use the property: If $\mathbf{Ax} = \lambda\mathbf{x}$,

$$\text{then } A^{-1}x = \frac{1}{\lambda}x$$

Example

Find the dominant eigenvalue and corresponding eigenvectors of the matrix given below by using power method.

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Solution

Assume that initial guess for eigenvector as

$$X = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Iteration 1

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

=> k=2, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2} \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \\ 0 \end{bmatrix}$$

$$E_1 = \left| \begin{array}{c} 1-0 \\ 1 \end{array} \right| = 1 \quad E_2 = \left| \begin{array}{c} 0.5-1 \\ 0.5 \end{array} \right| = 1 \quad E_3 = \left| \begin{array}{c} 0-0 \\ 0 \end{array} \right| = 0$$

Iteration 2

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.5 \\ 0 \end{bmatrix}$$

=> k=2.5, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2.5} \begin{bmatrix} 2 \\ 2.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1 \\ 0 \end{bmatrix}$$

$$E_1 = \left| \begin{array}{c} 0.8-1 \\ 0.8 \end{array} \right| = 0.25 \quad E_2 = \left| \begin{array}{c} 1-0.5 \\ 1 \end{array} \right| = 0.5 \quad E_3 = \left| \begin{array}{c} 0-0 \\ 0 \end{array} \right| = 0$$

Iteration 3

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0.8 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.8 \\ 2.6 \\ 0 \end{bmatrix}$$

=> k=2.8, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2.8} \begin{bmatrix} 2.8 \\ 2.6 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.93 \\ 0 \end{bmatrix}$$

$$E_1 = \left| \begin{array}{c} 1-0.8 \\ 1 \end{array} \right| = 0.2 \quad E_2 = \left| \begin{array}{c} 0.93-1 \\ 0.93 \end{array} \right| = 0.0752 \quad E_3 = \left| \begin{array}{c} 0-0 \\ 0 \end{array} \right| = 0$$

Iteration 4

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.93 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.86 \\ 2.93 \\ 0 \end{bmatrix}$$

$\Rightarrow k=2.93$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2.93} \begin{bmatrix} 2.86 \\ 2.93 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$$

$$E_1 = \left| \begin{array}{c} 1-1 \\ 1 \end{array} \right| = 0 \quad E_2 = \left| \begin{array}{c} 0.99-0.93 \\ 0.99 \end{array} \right| = 0.0606 \quad E_3 = \left| \begin{array}{c} 0-0 \\ 0 \end{array} \right| = 0$$

Iteration 5

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.98 \\ 2.99 \\ 0 \end{bmatrix}$$

$\Rightarrow k=2.99$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2.99} \begin{bmatrix} 2.98 \\ 2.99 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$E_1 = \left| \begin{array}{c} 1-1 \\ 1 \end{array} \right| = 0 \quad E_2 = \left| \begin{array}{c} 1-0.99 \\ 1 \end{array} \right| = 0.01 \quad E_3 = \left| \begin{array}{c} 0-0 \\ 0 \end{array} \right| = 0$$

Iteration 6

$$Y = AX = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix}$$

$\Rightarrow k=3$, New value of X can be calculated as
Thus largest eigenvalue is 3, and the corresponding eigenvector is

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

$$E_1 = \begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} = 0 \quad E_2 = \begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} = 0 \quad E_3 = \begin{vmatrix} 0 & 0 \\ 0 & 0 \end{vmatrix} = 0$$

Second Example

Find the smallest eigenvalue and corresponding eigenvectors of the matrix given below by using power method. Note that finding smallest eigenvalue of A is equivalent to finding dominant eigenvalue of A^{-1} .

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 6 \end{bmatrix}$$

Solution

Find inverse of given matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 6 \end{bmatrix}$$

We can easily obtain inverse of A by using Gauss-Jordan method

$$B = A^{-1} = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix}$$

Now, use power method to find dominant eigenvalue of A^{-1}

Assume initial is:

$$X = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Iteration 1

$$Y = BX = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

$\Rightarrow k=1$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{2} \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 0 \end{bmatrix}$$

Iteration 2

$$Y = BX = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 21 \\ -17.5 \\ 4.5 \end{bmatrix}$$

$\Rightarrow k=21$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{21} \begin{bmatrix} 21 \\ -17.5 \\ 4.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.833 \\ 0.214 \end{bmatrix}$$

Iteration 3

$$Y = BX = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -0.833 \\ 0.214 \end{bmatrix} = \begin{bmatrix} -37.924 \\ 31.639 \\ -8.118 \end{bmatrix}$$

$\Rightarrow k= -37.924$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{-37.924} \begin{bmatrix} -37.924 \\ 31.639 \\ -8.118 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.834 \\ 0.214 \end{bmatrix}$$

Iteration 4

$$Y = BX = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -0.834 \\ 0.214 \end{bmatrix} = \begin{bmatrix} -37.942 \\ 31.654 \\ -8.122 \end{bmatrix}$$

$\Rightarrow k= -37.942$, New value of X can be calculated as

$$X = \frac{1}{k} Y = \frac{1}{-37.942} \begin{bmatrix} -37.942 \\ 31.654 \\ -8.122 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.834 \\ 0.214 \end{bmatrix}$$

Thus dominant eigenvalue of A^{-1} is -37.942 and corresponding eigenvector is:

$$\begin{bmatrix} 1 \\ -0.834 \\ 0.214 \end{bmatrix}$$

And

Smallest eigenvalue of A is $1/-37.942 = -0.0264$ and corresponding eigenvector is:

$$\begin{bmatrix} 1 \\ -0.834 \\ 0.214 \end{bmatrix}$$

Algorithm for Power Method

1. Start
2. Read dimension of matrix, say n
3. Read elements of matrix row-wise
4. Read guess vector, say x
5. Compute y vector as below
 For $i=1$ to n
 $y[i] = a[i][1]*x[1]+a[i][2]*x[2]+a[i][3]*x[3]$
 End for
6. Set $k=fabs(y[1])$
7. Find largest absolute element of y vector and put that element in k
 For $i=2$ to n
 $\text{if}(k < fabs(y[i]))$
 $k=y[i]$
 End for
8. Find new values of x vector as below
 For $i=1$ to n
 $nx[i]=1/k*y[i]$
 End for
9. Compute and check error precision
 For $i=1$ to n
 $E[i]=(nx[i]-x[i])/nx[i]$
 $\text{If}(E[i]>\epsilon)$

```

        break
    End for
    If i=n+1
        Display eigenvalue & eigenvector, which is the vector nx
    Else
        For i=1 to n
            x[i]=nx[i]
        go to step 4
    10. Terminate

```

C program for calculating eigenvalue and eigenvector of a matrix by using power method

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j;
    float el, k, E[10],a[10][10],nx[10],x[10], y[10];
    printf("Enter Dimension of System of equations\n");
    scanf("%d",&n);
    printf("Enter coefficients row-wise\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%f",&a[i][j]);
    printf("Enter guess vector\n");
    for(i=0;i<n;i++)
        scanf("%f",&x[i]);
    printf("Enter Accuracy Limit\n");
    scanf("%f",&el);
    while(1)
    {
        for (i=0; i<n; i++)
            y[i]= a[i][0]*x[0]+a[i][1]*x[1]+a[i][2]*x[2];
        k=y[0];
        for(i=1;i<n;i++)
            if(k<y[i])
                k=y[i];
        for(i=0;i<n;i++)
            nx[i]=1/k*y[i];
        for(i=0; i<n;i++)
        {
            E[i]=(nx[i]-x[i])/nx[i];
        }
        if(E[0]<el)
            break;
    }
}

```

```

        if(E[i]>el)
            break;
    }
    if(i==n)
    {
        printf("Largest Eigenvalue is:%f\n",k);
        printf("Eigenvector is:\n");
        for(i=0;i<n;i++)
            printf("%f\t",nx[i]);
        break;
    }
    else
        for(i=0;i<n;i++)
            x[i]=nx[i];
}
getch();
return 0;
}

```

Output

Enter Dimension of System of equations
3

Enter coefficients row-wise

5 4 2

4 5 2

2 2 2

Enter guess vector

0 1 0

Enter Accuracy Limit

0.0001

Largest Eigenvalue is: 9.999887

Eigenvector is:

0.99998 1.000000 0.499999

EXERCISE

1. Define the key terms.
 - a. System of linear equations
 - b. Solution to a system of linear equations
 - c. Consistent system d. Inconsistent system
2. Identify whether the following system is consistent or inconsistent.
 - a. $x - y = -3$, $3x + y = -1$
 - b. $5x - 3y = -1$, $2x - 3y = -3$
 - c. $2x - y = 4$, $-4x + 2y = 2$
3. How direct methods of solving system of linear equations are different from iterative methods? Explain
4. Why pivoting is necessary? Use a counter example to show that pivoting reduces round-off errors.
5. Compute the number of operations needed by Gauss-Elimination method & Gauss-Jordan method.
6. Which iterative method is better and why?
7. Define inverse of a matrix. Discuss the Gauss-Jordan technique for calculating inverse of a matrix.
8. What is eigenvector & eigenvalue of matrices. Discuss applications of eigenvectors.
9. Solve the following system of equations by Gauss Elimination Method or Gauss-Jordan method.
 - a. $2x + 4y + z = 3$; $3x + 2y - 2z = -2$; $x - y + z = 6$
 - b. $6x - y + z = 13$; $x + y + z = 9$; $10x + y - z = 9$
 - c. $x + y + 2z = 4$; $3x + y - 3z = -4$; $2x - 3y - 5z = -5$
 - d. $4x + y + 3z = 11$; $3x + 4y + 2z = 11$; $2x + 3y + z = 7$
 - e. $2x - 3y - 20z = 25$; $20x + y - 2z = 17$; $3x + 20y - z = -18$
 - f. $2x + y + z - 2w = -10$; $4x + 2z + w = 8$; $3x + 2y + 2z = 7$; $x + 3y + 2z - w = 5$
10. Solve the following system of equations by Jacobi Iteration Method or Gauss Seidal Method.
 - a. $2x + 4y + z = 3$; $3x + 2y - 2z = -2$; $x - y + z = 6$
 - b. $6x - y + z = 13$; $x + y + z = 9$; $10x + y - z = 9$
 - c. $x + y + 2z = 4$; $3x + y - 3z = -4$; $2x - 3y - 5z = -5$
 - d. $4x + y + 3z = 11$; $3x + 4y + 2z = 11$; $2x + 3y + z = 7$
 - e. $2x - 3y - 20z = 25$; $20x + y - 2z = 17$; $3x + 20y - z = -18$
 - f. $2x + y + z - 2w = -10$; $4x + 2z + w = 8$; $3x + 2y + 2z = 7$; $x + 3y + 2z - w = 5$
11. Solve the following system of equations by Doolittle LU Decomposition Method.
 - a. $4x + y + 3z = 11$; $3x + 4y + 2z = 11$; $2x + 3y + z = 7$
 - b. $x + y + 5z = 16$; $2x + 3y + z = 4$; $4x + y - z = 4$
 - c. $4x - y + z = 23$; $x + y + z = 4$; $10x + y - z = 19$
 - d. $2x - 3y - 20z = 25$; $20x + y - 2z = 17$; $3x + 20y - z = -18$

e. $x - y + z = 6; 2x + 4y + z = 3; 3x + 2y - 2z = -2$

f. $3x + y + 2z = 3; 2x - 3y - z = -3; x + 2y + z = 4$

12. Using Cholesky Method Find the upper & lower triangular Matrix of the following matrices

a) $\begin{matrix} 2 & 1 & 1 \\ 3 & 2 & 3 \\ 1 & 4 & 9 \end{matrix}$

b) $\begin{matrix} 4 & 1 & 2 \\ 2 & 3 & -1 \\ 1 & -2 & 2 \end{matrix}$

c) $\begin{matrix} 1 & 2 & -1 \\ 3 & 8 & 2 \\ 4 & 9 & -1 \end{matrix}$

d) $\begin{matrix} 3 & -3 & 4 \\ 2 & -3 & 4 \\ 0 & -1 & 1 \end{matrix}$

e) $\begin{matrix} 2 & 1 & -1 \\ 0 & 2 & 1 \\ 5 & 2 & -3 \end{matrix}$

f) $\begin{matrix} 1 & 1 & 1 \\ 1 & 2 & -3 \\ 2 & -1 & 3 \end{matrix}$

13. Find the inverse of matrix by Gauss-Jordan Method.

a) $\begin{matrix} 2 & 1 & 1 \\ 3 & 2 & 3 \\ 1 & 4 & 9 \end{matrix}$

b) $\begin{matrix} 4 & 1 & 2 \\ 2 & 3 & -1 \\ 1 & -2 & 2 \end{matrix}$

c) $\begin{matrix} 1 & 2 & -1 \\ 3 & 8 & 2 \\ 4 & 9 & -1 \end{matrix}$

d) $\begin{matrix} 3 & -3 & 4 \\ 2 & -3 & 4 \\ 0 & -1 & 1 \end{matrix}$

e) $\begin{matrix} 2 & 1 & -1 \\ 0 & 2 & 1 \\ 5 & 2 & -3 \end{matrix}$

f) $\begin{matrix} 1 & 1 & 1 \\ 1 & 2 & -3 \\ 2 & -1 & 3 \end{matrix}$

14. Find the largest eigenvalue and the corresponding eigenvector for the following matrices.

a) $\begin{matrix} 4 & 1 & 0 \\ 1 & 20 & 1 \\ 0 & 1 & 4 \end{matrix}$

b) $\begin{matrix} 3 & 2 & 4 \\ 2 & 0 & 2 \\ 4 & 2 & 3 \end{matrix}$

c) $\begin{matrix} 1 & 2 & -1 \\ 3 & 8 & 2 \\ 4 & 9 & -1 \end{matrix}$

d) $\begin{matrix} 1 & 6 & 1 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{matrix}$

e) $\begin{matrix} 2 & 1 & -1 \\ 0 & 2 & 1 \\ 5 & 2 & -3 \end{matrix}$

f) $\begin{matrix} 1 & 1 & 1 \\ 1 & 2 & -3 \\ 2 & -1 & 3 \end{matrix}$



Chapter 6

NUMERICAL DIFFERENTIATION

Chapter Content

- ☒ Introduction
- ☒ Differentiating Continuous Functions
- ☒ Differentiating Discrete (Tabulated) Functions
- ☒ Maxima and Minima of tabulated functions

6.1 INTRODUCTION

The derivative or differentiation of a function represents the rate of change of a variable with respect to another variable. For example, the velocity of a body is defined as the rate of change of the location of the body with respect to time. The location is the dependent variable while time is the independent variable. Now if we measure the rate of change of velocity with respect to time, we get the acceleration of the body. In this case, the velocity is the dependent variable while time is the independent variable. Numerical differentiation is the process of obtaining the value of the derivative of a function from a set of numerical values of that function. There are basically two situations where numerical differentiation is required.

1. Differentiation of continuous function is required when the function to be differentiated is complicated and it is difficult to differentiate.
2. Differentiation of discrete (tabulated) functions is required when function values at some discrete points are known but function is unknown

6.2 DIFFERENTIATING CONTINUOUS FUNCTIONS

Here we discuss the process of approximating the derivatives $f'(x)$ of a function $f(x)$, when the function itself is available. If the function becomes too complex, it is sometimes easier to differentiate numerically rather than analytically.

6.2.1 Two Point Forward Difference Formula

Taylor's theorem says that if you know the value of a function $f(x)$ at a point x_i and all its derivatives at that point, provided the derivatives are continuous between x_i and x_{i+1} , then

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

Substituting for convenience $h = x_{i+1} - x_i$

$$f(x_i + h) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots$$

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} - \frac{f''(x_i)}{2!}h + \dots$$

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} + E \approx \frac{f(x_i + h) - f(x_i)}{h}$$

Here E is the error term in the approximation which is of the order of $O(h)$. This equation (1) is called Two Point Forward Difference Formula.

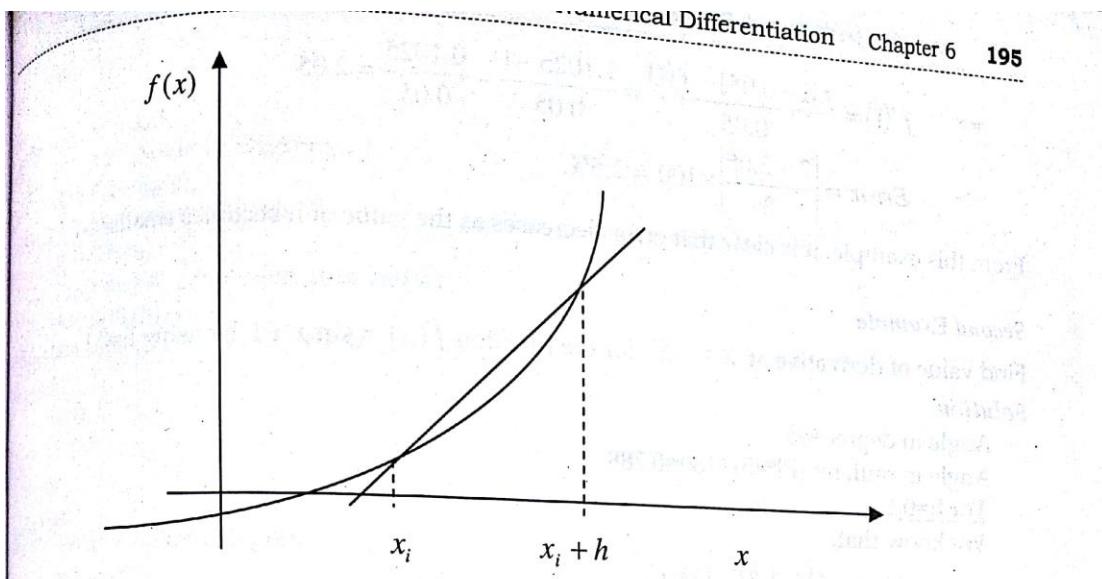


Figure 6.1: Graphical representation of forward difference approximation of first derivative.

Algorithm for Two-Point Forward difference formula

1. Start
2. Read the value at which derivative is needed, say x
3. Read interval gap, say h
4. Calculate $f(x_i)$ & $f(x_i+h)$
5. Calculate $d = f'(x_i) = (f(x_i+h) - f(x_i))/h$
6. Display the value derivative
7. Terminate

Example

Find value of derivative at $x = 1$ for the function $f(x) = x^2$ by using $h=0.2$ and 0.05 .

Solution

For $h=0.2$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$$\Rightarrow f'(1) = \frac{f(1+0.2) - f(1)}{0.2} = \frac{1.44 - 1}{0.2} = \frac{0.44}{0.2} = 2.2$$

True value of derivative at $x=1$ is

$$f'(x) = 2x = 2$$

$$\Rightarrow \text{Error} = \left| \frac{2 - 2.2}{2} \right| \times 100 = 10\%$$

For $h=0.05$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$$\Rightarrow f'(1) = \frac{f(1+0.05) - f(1)}{0.05} = \frac{1.1025 - 1}{0.05} = \frac{0.1025}{0.05} = 2.05$$

$$\Rightarrow Error = \left| \frac{2 - 2.05}{2} \right| \times 100 = 2.5\%$$

From this example, it is clear that error decreases as the value of h becomes smaller.

Second Example

Find value of derivative at $x = 45^\circ$ for the function $f(x) = \sin x + 1$ by using $h=0.1$ and 0.01

Solution

Angle in degree=45

Angle in radian= $(\text{PI}*45)/180 = 0.788$

For $h=0.1$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$$\Rightarrow f'(45^\circ) = f'(0.788) = \frac{f(0.788 + 0.1) - f(0.788)}{0.1} = \frac{f(0.888) - f(0.788)}{0.1}$$

$$= \frac{0.776 - 0.709}{0.1} = 0.67$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow Error = \left| \frac{0.705 - 0.67}{0.705} \right| = 0.049 = 4.9\%$$

For $h=0.01$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$$\Rightarrow f'(45^\circ) = f'(0.788) = \frac{f(0.788 + 0.01) - f(0.788)}{0.01} = \frac{f(0.798) - f(0.788)}{0.01}$$

$$= \frac{0.716 - 0.709}{0.01} = 0.70$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow Error = \left| \frac{0.705 - 0.70}{0.705} \right| = 0.007 = 0.7\%$$

//C Program for calculating derivative using forward difference formula

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define PI 3.1415
#define f[x] sin[x]+1
```

```

int main()
{
    float angle,h,x,d;
    printf("Enter Angle in Degree\n");
    scanf("%f",&angle);
    printf("Enter increment h\n");
    scanf("%f",&h);
    x=PI/180*angle; //Convert into radian
    d=((f[x+h])-f[x])/h;
    printf("Value of Derivative=%f\n",d);
    getch();
    return 0;
}

```

Output**First Run**

Enter Angle in Degree
45
Enter increment h
0.1
Value of Derivative=0.670620

Second Run

Enter Angle in Degree
45
Enter increment h
0.01
Value of Derivative=0.703576

6.2.2 Two Point Backward Difference Formula

Taylor's theorem says that if you know the value of a function $f(x)$ at a point x_i and all its derivatives at that point, provided the derivatives are continuous between x_i and x_{i+1} , then

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

Note that here x_{i+1} is the point behind x_i . Substituting for convenience $h = x_i - x_{i+1}$

$$f(x_i - h) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots$$

$$f'(x_i) = \frac{f(x_i) - f(x_i - h)}{h} + \frac{f''(x_i)}{2!}h + \dots$$

$$f'(x_i) = \frac{f(x_i) - f(x_i - h)}{h} + E \approx \frac{f(x_i) - f(x_i - h)}{h}$$

(1)

Here E is the error term in the approximation which is of the order of $O(h)$. This equation (1) is called **Two Point Backward Difference Formula**.

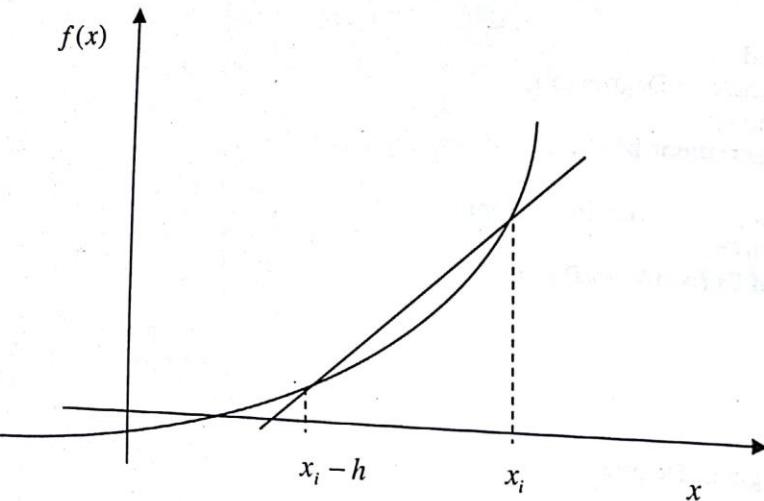


Figure 6.2: Graphical representation of backward difference approximation of first derivative.

Algorithm for Two-Point Backward difference formula

1. Start
2. Read the value at which derivative is needed, say x
3. Read interval gap, say h
4. Calculate $f(x_i)$ & $f(x_i-h)$
5. Calculate $d = f'(x_i) = (f(x_i) - f(x_i-h))/h$
6. Display the value of derivative
7. Terminate

Example

Find value of derivative at $x=1$ for the function $f(x) = x^2$ by using $h=0.2$ and 0.05 .

Solution

For $h=0.2$

We know that,

$$f'(x_i) = \frac{f(x_i) - f(x_i-h)}{h}$$

$$\Rightarrow f'(1) = \frac{f(1) - f(0.8)}{0.2} = \frac{1 - 0.64}{0.2} = \frac{0.36}{0.2} = 1.8$$

True value of derivative at $x=1$ is

$$f'(x) = 2x = 2$$

$$\Rightarrow \text{Error} = \left| \frac{2 - 1.8}{2} \right| \times 100 = 10\%$$

For $h=0.05$

We know that,

$$f'(x_i) = \frac{f(x_i) - f(x_i-h)}{h}$$

$$\Rightarrow f'(1) = \frac{f(1) - f(0.95)}{0.05} = \frac{1 - 0.9025}{0.05} = \frac{0.0975}{0.05} = 1.95$$

$$\Rightarrow \text{Error} = \left| \frac{2 - 1.95}{2} \right| \times 100 = 2.5\%$$

Second Example

Find value of derivative at $x = 45^0$ for the function $f(x) = \sin x + 1$ by using $h=0.1$ and 0.001 .

Solution

Angle in degree=45

Angle in radian= $(\pi * 45) / 180 = 0.788$ For h=0.1

We know that,

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

$$\begin{aligned} \Rightarrow f'(45^0) &= f'(0.788) = \frac{f(0.788) - f(0.788 - 0.1)}{0.1} = \frac{f(0.788) - f(0.688)}{0.1} \\ &= \frac{0.709 - 0.635}{0.1} = 0.74 \end{aligned}$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow \text{Error} = \left| \frac{0.705 - 0.74}{0.705} \right| = 0.049 = 4.9\%$$

For h=0.01

We know that,

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

$$\begin{aligned} \Rightarrow f'(45^0) &= f'(0.788) = \frac{f(0.788) - f(0.788 + 0.01)}{0.01} = \frac{f(0.788) - f(0.778)}{0.01} \\ &= \frac{0.709 - 0.702}{0.01} = 0.70 \end{aligned}$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow \text{Error} = \left| \frac{0.705 - 0.70}{0.705} \right| = 0.007 = 0.7\%$$

*/*C program for calculating derivatives using backward difference formula*/*

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float h,x,d;
    printf("Enter the value at which derivative is required\n");
    scanf("%f",&x);
```

```

printf("Enter increment h\n");
scanf("%f",&h);
d=((f(x))-(f(x-h))/h;
printf("Value of Derivative=%f\n",d);
getch();
return 0;
}

```

Output**First Run**

Enter the value at which derivative is required

3

Enter increment h

0.1

Value of Derivative=42.248287

Second Run

Enter the value at which derivative is required

3

Enter increment h

0.01

Value of Derivative=40.372601

6.2.3 Three Point Formula

From Taylor Series

$$f(x_i + h) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots \quad (1)$$

$$f(x_i - h) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots \quad (2)$$

Subtracting equation (2) from equation (1), we get

$$f(x_i + h) - f(x_i - h) = f'(x_i)2h + \frac{f''(x_i)}{2!}h^2 + \dots$$

$$\Rightarrow f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} + E \approx \frac{f(x_i + h) - f(x_i - h)}{2h} \quad (3)$$

Hence we have obtained a more accurate formula as the error is of the order of $O(h^2)$. This equations is called Three Point Formula.

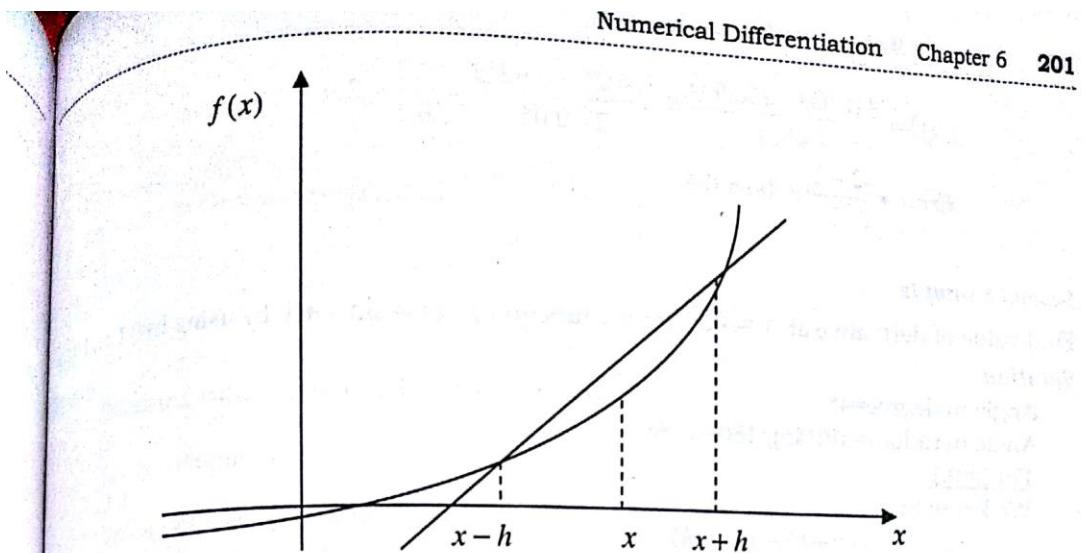


Figure 6.3: Graphical representation of Central difference approximation of first derivative.

Algorithm for Two-Point Backward difference formula

1. Start
2. Read the value at which derivative is needed, say x
3. Read interval gap, say h
4. Calculate $f(x_i)$ & $f(x_i+h)$
5. Calculate $d = f'(x_i) = (f(x_i+h) - f(x_i-h)) / 2h$
6. Display the value of derivative
7. Terminate

Example

Find value of derivative at $x=1$ for the function $f(x) = x^2$ by using $h=0.2$ and 0.05 .

Solution

For $h=0.2$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h}$$

$$\Rightarrow f'(1) = \frac{f(1.2) - f(0.8)}{2 \times 0.2} = \frac{1.44 - 0.64}{2 \times 0.2} = \frac{0.8}{0.4} = 2.0$$

True value of derivative at $x=1$ is

$$f'(x) = 2x = 2$$

$$\Rightarrow \text{Error} = \left| \frac{2 - 2}{2} \right| \times 100 = 0\%$$

For $h=0.05$

We know that,

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h}$$

$$\Rightarrow f'(1) = \frac{f(1.05) - f(0.95)}{2 \times 0.05} = \frac{1.1025 - 0.9025}{2 \times 0.05} = \frac{0.2}{0.1} = 2.0$$

$$\Rightarrow \text{Error} = \left| \frac{2 - 2}{2} \right| \times 100 = 0\%$$

Second Example

Find value of derivative at $x = 45^\circ$ for the function $f(x) = \sin x + 1$ by using $h=0.1$ and 0.001 .

Solution

Angle in degree=45

Angle in radian= $(\pi \cdot 45)/180 = 0.788$

For $h=0.1$

We know that,

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

$$\begin{aligned} f'(45^\circ) &= f'(0.788) = \frac{f(0.788+0.1) - f(0.788-0.1)}{2 \times 0.1} = \frac{f(0.888) - f(0.688)}{0.2} \\ &= \frac{0.776 - 0.635}{0.2} = 0.705 \end{aligned}$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow \text{Error} = \left| \frac{0.705 - 0.705}{0.705} \right| = 0 = 0\%$$

For $h=0.01$

We know that,

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

$$\begin{aligned} f'(45^\circ) &= f'(0.788) = \frac{f(0.788+0.01) - f(0.788-0.01)}{2 \times 0.01} = \frac{f(0.798) - f(0.778)}{0.02} \\ &= \frac{0.716 - 0.702}{0.02} = 0.7 \end{aligned}$$

True value of derivative at $x=45$ is

$$f'(x) = \cos x = 0.705$$

$$\Rightarrow \text{Error} = \left| \frac{0.705 - 0.70}{0.705} \right| = 0.007 = 0.7\%$$

'/C program for calculating derivative using central difference formula

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) 2*(x)*(x)+1
int main()
```

```
float h,x,d;
```

```
printf("Enter the value at which derivative is required\n");
```

```

scnaf("%f", &x);
printf("Enter increment h\n");
scnaf("%f", &h);
d=((f(x+h))-f(x-h))/(2*h);
printf("Value of Derivative=%f\n", d);
getch();
return 0;
}

```

Output

Enter the value at which derivative is required

2

Enter increment h

0.1

Value of Derivative=8.000000

6.3 DIFFERENTIATING DISCRETE (TABULATED) FUNCTIONS

If the functional value is known at some points but function is not known, we can still find derivatives of such tabulated functions. In such cases first we have to interpolate the tabulated function and then differentiate it. Differentiation of tabulated functions suffers from conflict between roundoff errors (due to limited machine precision) and errors inherent in interpolation. For this reason, a derivative of a function can never be computed with the same precision as the function itself. Two situations exists here:

- ☞ If arguments are equally spaced, we will use Newton-Gregory forward formula, if we desire to find the derivative of the function at a point near to beginning. If we desire to find the derivative of the function at a point near to end then we will use Newton-Gregory backward formula. And if the derivative at a point is near the middle of the table we apply Central difference formula.
- ☞ In case the arguments are unequally spaced then we should use Newton's divided difference formula.

6.3.1 Derivatives Using Newton's Divided Difference Formula

We know that, the general form of the Newton's divided difference polynomial for $n+1$ data points, $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_n, y_n)$, is given by

$$P_n(x) = f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0)(x - x_1)\dots(x - x_{n-1})$$

$$\text{or } P_n(x) = f(x) = f[x_0] + f[x_1, x_0](x - x_0) + \dots + f[x_n, x_{n-1}, \dots, x_0](x - x_0)(x - x_1)\dots(x - x_{n-1}) \quad (1)$$

$$\text{or } P_n(x) = f[x_0] + \sum_{i=1}^n f[x_0, x_1, \dots, x_n] \prod_{j=0}^{i-1} (x - x_j)$$

Where the definition of the m^{th} divided difference is

$$f[x_m, \dots, x_0] = \frac{f[x_m, \dots, x_1] - f[x_{m-1}, \dots, x_0]}{(x_m - x_0)}$$

Differentiating equation (1) w.r.t. x, we get

$$f'(x) = f[x_1, x_0] + f[x_2, x_1, x_0]\{(x - x_1) + (x - x_0)\} + f[x_3, x_2, x_1, x_0]\{(x - x_1)(x - x_2) + (x - x_0)(x - x_2) + (x - x_0)(x - x_1)\} + \dots \quad (2)$$

By putting $x=a$ in equation (2) we can get value of first derivative at $x=a$. Again, differentiating equation (2) w.r.t. x , we get

$$f''(x) = 2f[x_2, x_1, x_0] + 2f[x_3, x_2, x_1, x_0]\{(x - x_0) + (x - x_1) + (x - x_2)\} + \dots \quad (3)$$

By putting $x=a$ in equation (3) we can get value of second derivative at $x=a$

Algorithm

1. Start
2. Read number of points, say n
3. Read n data points
4. Read the value at which derivative is needed, say x
5. Compute divided differences as below


```

        For i=0 to n-1
          dd[i]=fx[i]
        End for
        For i=0 to n-1
          for j = n-1 to i+1
            dd[j]=dd[j]-dd[j-1]/(x[j]-x[j-1-i])
          End For
        End for
      
```
6. Compute differentiated values as below


```

        Set v=dd[1]
        For i=2 to n-1
          term=0
          For j=0 to i
            factor=1
            For k=0 to i
              If(j!=k) factor=factor*(x-x[k])
            End for
            term=term+factor
          End For
          vod=vod+dd[i]*term
        End for
      
```
7. Print the value of first derivative, which is value of variable v
8. Terminate

Example

Find $f'(10)$ from the following data points:

x	3	5	11	27	34
$f(x)$	-13	23	899	17315	35606

Solution

In this case the values of the arguments are not equally spaced. So we shall use Newton's divided difference formula. The divided difference table is given below:

$f(x)$	First Divided Differences	Second Divided Differences	Third Divided Differences	Fourth Divided Differences	
23		$f[x_1, x_0]$			
13	$\frac{23 - (-13)}{5 - 3} = 18$		$f[x_2, x_1, x_0]$		
5		$\frac{146 - 18}{11 - 3} = 16$		$f[x_3, x_2, x_1, x_0]$	
3	$\frac{899 - 23}{11 - 5} = 146$		$\frac{40 - 16}{27 - 3} = 1$		$f[x_4, x_3, x_2, x_1, x_0]$
27		$\frac{1026 - 146}{27 - 5} = 40$		$\frac{1 - 1}{34 - 3} = 0$	
11	$\frac{17315 - 899}{27 - 11} = 1026$		$\frac{69 - 40}{34 - 5} = 1$		
17		$\frac{2613 - 1026}{34 - 11} = 69$			
35	$\frac{35606 - 17315}{34 - 27} = 261$				
6					

We know that,

$$f'(x) = f[x_1, x_0] + f[x_2, x_1, x_0]\{(x - x_1) + (x - x_0)\} + f[x_3, x_2, x_1, x_0]\{(x - x_1)(x - x_2) + (x - x_0)(x - x_2) + (x - x_0)(x - x_1)\} + \dots$$

At $x=10$

$$\begin{aligned} f'(10) &= 18 + 16\{(10 - 5) + (10 - 3)\} + 1\{(10 - 5)(10 - 11) + (10 - 3)(10 - 11) + (10 - 3)(10 - 5)\} \\ &= 18 + 16 \times 12 + 1 \times \{-5 - 7 + 35\} \\ &= 18 + 192 + 23 = 233 \end{aligned}$$

Second Example

A slider in a machine moves along a fixed straight rod. Its distance covered by the machine is given below for various values of the time. Find the velocity of the slider when at $x=0.3$ seconds.

Time (x)	0.1	0.2	0.3	0.4	0.5
Distance($f(x)$)	31.62	32.87	33.64	33.95	33.81

Since we have to find velocity at $t=0.3$, which lies around center of data points, we shall use Newton's divided difference formula. The divided difference table is given below:

$f(x)$	1 st Divided Difference	2 nd Divided Difference	3 rd Divided Difference	4 th Divided Difference
31.62				
	<u>12.5</u>			
32.87		<u>-24</u>		
	7.7		<u>3.333</u>	
33.64		-23		<u>0.00</u>
	3.1		3.333	
33.95		-22		
	-1.3			
33.81				

We know that,

$$f'(x) = f[x_1, x_0] + f[x_2, x_1, x_0]\{(x - x_1) + (x - x_0)\} + f[x_3, x_2, x_1, x_0]\{(x - x_1)(x - x_2) + (x - x_0)(x - x_2) + (x - x_0)(x - x_1)\} + \dots$$

At $x=0.3$

$$f'(0.3) = 12.5 - 24 \times \{(0.3 - 0.2) + (0.3 - 0.1)\} + 3.333 \times \{(0.3 - 0.2) \times (0.3 - 0.3) + (0.3 - 0.1) \times (0.3 - 0.3) + (0.3 - 0.1) \times (0.3 - 0.2)\}$$

\therefore

$$f'(0.3) = 12.5 - 7.2 + 0.666 = 5.97 \text{ cm/sec}$$

```
// C Program for computing differentiation using divided difference polynomial
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,j,k;
    float factor, term, vod, xv,x[10],fx[10],a[10];
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter values of data points\n");
    for(i=0;i<n;i++)
        scanf("%f",&x[i]);
    for(i=0;i<n;i++)
        scanf("%f",&fx[i]);
    for(i=0;i<n;i++)
        scanf("%f",&a[i]);
    for(i=1;i<n;i++)
    {
        for(j=0;j<i;j++)
            a[j+1]=(fx[j+1]-fx[j])/(x[j+1]-x[j]);
    }
    term=a[1];
    for(i=1;i<n;i++)
    {
        factor=term*(x[i]-0.3);
        term+=factor;
    }
    printf("The value of derivative at x=0.3 is %f",term);
}
```

```

for(i=0;i<n;i++)
{
    scanf("%f%f", &x[i], &fx[i]);
}
printf("Enter the value at which derivative is required\n");
scanf("%f", &xv);

//Calculating divided differences
for(i=0;i<n;i++)
{
    a[i]=fx[i];
}
for(i=0;i<n;i++)
{
    for(j=n-1;j>i;j--)
    {
        a[j]=(a[j]-a[j-1])/(x[j]-x[j-1-i]);
    }
}

//Calculating value of derivative
vod=a[1];
for(i=2;i<n;i++)
{
    term=0;
    for(j=0;j<i;j++)
    {
        factor=1;
        for(k=0;k<i;k++)
        {
            if(k!=j)
                factor=factor*(xv-x[k]);
        }
        term=term+factor;
    }
    vod=vod+(a[i]*term);
}
printf("Value of First Derivative=%f",vod);
getch();
return 0;

```

Output

Enter the number of points

5

Enter values of data points

3 -13

5 23

11 899

27 17315

34 35606

Enter the value at which derivative is required

10
Value of First Derivative=233.000000

6.3.2 Derivatives Using Newton's Forward Difference Formula

Newton's forward difference formula for $n+1$ data points, $(x_0, f(x_0)), \dots, (x_n, f(x_n))$, can

be written as

$$f(x) = f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \dots \quad (1)$$

$$\text{Where, } s = \frac{(x-x_0)}{h}$$

Differentiating equation (1) w.r.t. x , we get

$$f'(x) = \left[f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \right] \frac{ds}{dx}$$

$$f'(x) = \left[\frac{1}{4!} s(s-1)(s-2)(s-3) \Delta^4 f(x_0) + \dots \right]$$

$$f'(x) = \left[\Delta f(x_0) + \frac{1}{2!} \{s + (s-1)\} \Delta^2 f(x_0) + \frac{1}{3!} \left\{ s(s-1) + (s-2) \frac{d}{ds} \{s(s-1)\} \right\} \Delta^3 f(x_0) + \right]$$

$$f'(x) = \left[\frac{1}{4!} \left\{ s(s-1)(s-2) + (s-3) \frac{d}{ds} \{s(s-1)(s-2)\} \right\} \Delta^4 f(x_0) + \dots \right]$$

$$f'(x) = \left[\Delta f(x_0) + \frac{1}{2!} \{2s-1\} \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \right]$$

$$f'(x) = \left[\frac{1}{4!} (4s^3 - 18s^2 + 22s - 6) \Delta^4 f(x_0) + \dots \right] \quad (2)$$

Since,

$$s = \frac{(x-x_0)}{h} \Rightarrow \frac{ds}{dx} = \frac{1}{h}$$

Thus, equation (2) becomes

$$f'(x) = \frac{1}{h} \left\{ \Delta f(x_0) + \frac{1}{2!} (2s-1) \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \dots \right\} \quad (3)$$

By putting $x=a$, equation (3) can be used to find the value of first derivative at the point $x=a$. Again differentiating equation (3) w.r.t. x , we get

$$f''(x) = \frac{d}{ds} \left[\frac{1}{h} \left\{ \Delta f(x_0) + \frac{1}{2!} (2s-1) \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \dots \right\} \right] \frac{ds}{dx}$$

$$f''(x) = \frac{1}{h} \left\{ \Delta^2 f(x_0) + \frac{1}{3!} (6s-6) \Delta^3 f(x_0) + \frac{1}{4!} (12s^2 - 36s + 22) \Delta^4 f(x_0) + \dots \right\} \frac{ds}{dx}$$

$$f''(x) = \frac{1}{h^2} \left\{ \Delta^2 f(x_0) + \frac{1}{3!} (6s-6) \Delta^3 f(x_0) + \frac{1}{4!} (12s^2 - 36s + 22) \Delta^4 f(x_0) + \dots \right\} \quad (4)$$

By putting $x=a$, equation (4) can be used to find the value of second derivative at the point $x=a$.

Algorithm

1. Start
2. Read number of data points, say n
3. Read the value at which derivative is needed, say x_p
4. Read n data points, say $x[i]$ and $f[x][i]$
5. $h=x[1]-x[0]$ and $s=(x_p-x[0])/h$
6. Compute forward differences as below:
 For $i=0$ to $n-1$
 $fd[i]=f[x][i]$
 End for
 For $i=0$ to $n-1$
 For $j=n-1$ to $i+1$
 $fd[j]=fd[j]-fd[j-1]$
 End for
 End For
7. Compute value of derivative as below
 $val=fd[1]$
 $prevterm=1$
 For $i=2$ to $n-1$
 $term1=1$
 For $k=2$ to i
 $term1=term1*(s-k+2)$
 End For
 $term2=term2+(s-i+1)*prevterm$
 $prevterm=term1+term2$
 $val=val+(fd[i]*(term1+term2))/i!$
8. Print the value of first derivative (val)
9. Terminate

Example

Find the first and second derivatives of the functions tabulated below at 1.1

x	1.0	1.2	1.4	1.6	1.8	2.0
$f(x)$	0.0	0.128	0.544	1.296	2.432	4.000

Solution

Since $x=1.1$ lies near the beginning of the table therefore in this case we shall use Newton's Gregory forward formula.

The difference table is as below:

x	$f(x)$	First Divided Differences	Second Divided Differences	Third Divided Differences	Fourth Divided Differences	Fifth Divided Differences
1.0	0.0					
		<u>0.128</u>				
1.2	0.128		<u>0.288</u>			
		0.416		<u>0.048</u>		
1.4	0.544		0.336		<u>0</u>	
		0.752		0.048		<u>0</u>
1.6	1.296		0.384		0	
		1.136		0.048		
1.8	2.432		0.432			
		1.568				
2.0	4.0					

We know that,

$$f'(x) = \frac{1}{h} \left\{ \Delta f(x_0) + \frac{1}{2!} (2s-1) \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \dots \right\}$$

Here,

$$h = 0.2 \quad \text{and} \quad s = \frac{(x - x_0)}{h} = \frac{(1.1 - 1.0)}{0.2} = 0.5$$

Thus,

$$f'(x) = \frac{1}{0.2} \left\{ 0.128 + \frac{1}{2} (2 \times 0.5 - 1) 0.288 + \frac{1}{6} (3 \times 0.25 - 6 \times 0.5 + 2) 0.048 \right\}$$

$$f'(x) = \frac{1}{0.2} \left\{ 0.128 + \frac{1}{2} (1 - 1) 0.288 + \frac{1}{6} (0.75 - 3 + 2) 0.048 \right\}$$

$$f'(x) = \frac{1}{0.2} \{ 0.128 + 0 - 0.002 \} = 0.63$$

Again, since

$$f''(x) = \frac{1}{h^2} \left\{ \Delta^2 f(x_0) + \frac{1}{3!} (6s - 6) \Delta^3 f(x_0) + \frac{1}{4!} (12s^2 - 36s + 22) \Delta^4 f(x_0) \right\}$$

$$f''(x) = \frac{1}{0.2^2} \left\{ 0.288 + \frac{1}{6} (6 \times 0.5 - 6) 0.048 + \frac{1}{24} (12 \times 0.25 - 36 \times 0.5 + 22) \times 0 \right\}$$

$$f''(x) = \frac{1}{0.04} \{ 0.288 + -0.024 + 0 \} = 6.6$$

Second Example

Find the first derivative of the function tabulated below at 1.1

<i>x</i>	1.0	1.1	1.2	1.3	1.4
<i>f(x)</i>	7.989	8.403	8.781	9.129	9.451

Solution
Since $x=1.1$ lies near the beginning of the table therefore in this case we shall use Newton's Gregory forward formula. The difference table is as below:

<i>x</i>	<i>f(x)</i>	<i>First Divided Differences</i>	<i>Second Divided Differences</i>	<i>Third Divided Differences</i>	<i>Fourth Divided Differences</i>
1.0	7.989				
		<u>0.414</u>			
1.1	8.403		<u>-0.036</u>		
		0.378		<u>0.006</u>	
1.2	8.781		-0.03		<u>-0.002</u>
		0.348		0.004	
1.3	9.129		-0.026		
		0.322			
1.4	9.451				

We know that,

$$f'(x) = \left[\Delta f(x_0) + \frac{1}{2!} \{2s-1\} \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \frac{1}{4!} (4s^3 - 18s^2 + 22s - 6) \Delta^4 f(x_0) + \dots \right]$$

Here,

$$h = 0.1 \quad \text{and} \quad s = \frac{x - x_0}{h} = \frac{1.1 - 1.0}{0.1} = 1$$

Thus,

$$\begin{aligned} f'(x) &= \frac{1}{0.1} \left\{ 0.414 + \frac{1}{2} (2-1) \times (-0.036) + \frac{1}{6} (3-6+2) \times 0.006 + \frac{1}{24} (4-18+22-6) \times (-0.002) \right\} \\ &= \frac{1}{0.1} \{ 0.414 - 0.018 - 0.001 - 0.00016 \} \\ &\approx 3.95 \end{aligned}$$

//C program for calculating value derivative using forward differences

```

#include<stdio.h>
#include<conio.h>
int fact(int n)
{
    if (n==1) return 1;
    else return(n*fact(n-1));
}
int main()
{
    int n,i,j,k;
    float val=0,p,xp,x[10],fx[10],fd[10],h,s,term1,term2,prev;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter values of x & f(x)\n");
    for(i=0;i<n;i++)
    {
        scanf("%f%f",&x[i],&fx[i]);
    }
    printf("Enter the value at which Derivative is needed\n");
    scanf("%f",&xp);
    h=x[1]-x[0];
    s=(xp-x[0])/h;
    //Calculation of forward differences
    for(i=0;i<n;i++)
    {
        fd[i]=fx[i];
    }
    for(i=0;i<n;i++)
    {
        for(j=n-1;j>i;j--)
        {
            fd[j]=(fd[j]-fd[j-1]);
        }
    }
    //Calculating value of first derivative
    val=fd[1];
    prev=1;
    for(i=2;i<n;i++)
    {
        term1=1;
        for(k=2;k<=i;k++)
        {
            term1=term1*(s-k+2);
        }
        term2=(s-i+1)*prev;
        prev=(term1+term2);
    }
}

```

```

val=val+(fd[i]*(term1+term2))/fact(i);
}
val=val/h;
printf("Value of First Derivative=%f",val);
getch();
return 0;

```

Output

Enter the number of points

7

Enter values of x & f(x)

1.0	2.7183
1.2	3.3201
1.4	4.0552
1.6	4.9530
1.8	6.0496
2.0	7.3891
2.2	9.0250

Enter the value at which Derivative is needed

1.2

Value of First Derivative=3.320233

6.3.3 Derivatives Using Newton's Backward Difference Formula

Newton's backward difference formula for $n+1$ data points, $(x_0, f(x_0)), \dots, (x_n, f(x_n))$, can be written as

$$P_n(x) = f(x) = f(x_n) + \nabla f(x_n)s + \frac{1}{2} \nabla^2 f(x_n)s(s+1) + \dots + \frac{1}{n!} \nabla^n f(x_n)s(s+1)\dots(s+n-1) \quad (1)$$

Where, $s = \frac{(x - x_n)}{h}$

Differentiating equation (1) w.r.t. x, we get

$$f'(x) = \frac{d}{ds} \left[f(x_n) + \nabla f(x_n)s + \frac{1}{2!} \nabla^2 f(x_n)s(s+1) + \frac{1}{3!} \nabla^3 f(x_n)s(s+1)(s+2) + \dots \right] \frac{ds}{dx} \quad (2)$$

$$f'(x) = \left\{ \nabla f(x_n) + \frac{1}{2!} (2s+1) \nabla^2 f(x_n) + \frac{1}{3!} (3s^2 + 6s + 2) \nabla^3 f(x_n) + \dots \right\} \frac{ds}{dx} \quad (2)$$

Since,

$$s = \frac{(x - x_n)}{h} \Rightarrow \frac{ds}{dx} = \frac{1}{h}$$

Thus, equation (2) becomes

$$f'(x) = \frac{1}{h} \left\{ \nabla f(x_n) + \frac{1}{2!} (2s+1) \nabla^2 f(x_n) + \frac{1}{3!} (3s^2 + 6s + 2) \nabla^3 f(x_n) + \dots \right\} \quad (3)$$

By putting $x = a$, equation (3) can be used to find the value of first derivative at the point $x = a$

Again differentiating equation (3) w.r.t. x, we get

$$\begin{aligned}
 f''(x) &= \frac{d}{ds} \left[\frac{1}{h} \left\{ \nabla f(x_n) + \frac{1}{2!}(2s+1)\nabla^2 f(x_n) + \frac{1}{3!}(3s^2+6s+2)\nabla^3 f(x_n) + \dots \right\} \right] \frac{ds}{dx} \\
 f''(x) &= \frac{1}{h} \left\{ \nabla^2 f(x_n) + \frac{1}{3!}(6s+6)\nabla^3 f(x_n) + \frac{1}{4!}(12s^2+36s+22)\nabla^4 f(x_n) + \dots \right\} \frac{ds}{dx} \\
 f''(x) &= \frac{1}{h^2} \left\{ \nabla^2 f(x_n) + \frac{1}{3!}(6s+6)\nabla^3 f(x_n) + \frac{1}{4!}(12s^2+36s+22)\nabla^4 f(x_n) + \dots \right\} \quad (4)
 \end{aligned}$$

By putting $x = a$, equation (4) can be used to find the value of second derivative at the point $x = a$

Algorithm

1. Start
2. Read number of data points, say n
3. Read the value at derivative is needed, say x_p
4. Read n data points, say $x[i]$ and $f_x[i]$
5. Set $h=x[1]-x[0]$ and $s=(x_p-x[n-1])/h$
6. Compute backward differences as below:
 - For $i=0$ to $n-1$
 - $bd[i]=f_x[i]$
 - End for
 - For $i=n-1$ to 1
 - For $j=0$ to $i-1$
 - $bd[j]=bd[j+1]-bd[j]$
 - End for
 - End For
7. Compute value of derivative as below
 - Set $val=bd[n-2]$
 - Set $prevterm=1$
 - For $i=2$ to $n-1$
 - $term1=1$
 - For $k=2$ to i
 - $term1=term1*(s+k-2)$
 - End For
 - $term2=term2+(s+i-1)*prevterm$
 - $prevterm=term1+term2$
 - $val=val+(bd[n-i-1]*(term1+term2))/i!$
 - End for
 - $val=val/h$
8. Print the value of first derivative (val)
9. Terminate

Example

Find the first and second derivatives of the functions tabulated below at $x=9$

x	5	6	7	8	9
$f(x)$	10.0	14.5	19.5	25.5	32.0

Solution
Since $x=9$ lies at the end of the table therefore in this case we shall use Newton's Gregory backward formula. The difference table is as below:

x	$f(x)$	First Divided Differences	Second Divided Differences	Third Divided Differences	Fourth Divided Differences
5	10				
6	14.5	4.5	0.5		
7	19.5	5.0	1.0	0.5	-1.0
8	25.5	6.0	0.5	-0.5	
9	32.0	6.5			

We know that,

$$f'(x) = \frac{1}{h} \left\{ \nabla f(x_n) + \frac{1}{2!}(2s+1)\nabla^2 f(x_n) + \frac{1}{3!}(3s^2 + 6s + 2)\nabla^3 f(x_n) + \dots \right\}$$

Here,

$$h=1.0 \quad \text{and} \quad s = \frac{(x - x_n)}{h} = \frac{(9 - 9)}{1.0} = 0$$

Thus,

$$f'(9) \approx \frac{1}{1.0} \left\{ 6.5 + \frac{1}{2}(2 \times 0 + 1)0.5 + \frac{1}{6}(3 \times 0 - 6 \times 0 + 2)(-0.5) \right\}$$

$$f'(9) = \frac{1}{1.0} \{ 6.5 + 0.25 - 0.166 \} = 6.584$$

Again, since

$$f''(x) = \frac{1}{h^2} \left\{ \nabla^2 f(x_n) + \frac{1}{3!}(6s+6)\nabla^3 f(x_n) + \frac{1}{4!}(12s^2 + 36s + 22)\nabla^4 f(x_n) + \dots \right\}$$

$$f''(9) = \frac{1}{1.0} \left\{ 0.5 + \frac{1}{6}(0 + 6)(-0.5) + \frac{1}{24}(0 + 0 + 22)(-1) \right\}$$

$$f''(9) = \frac{1}{1.0} \{ 0.5 - 0.5 - 0.9166 \} = -0.9166$$

Second Example

Find the first derivative of the function tabulated below at 1.4

x	1.0	1.1	1.2	1.3	1.4
$f(x)$	7.989	8.403	8.781	9.129	9.451

Solution

Since $x=1.5$ lies at the end of the table therefore in this case we shall use Newton's Gregory backward formula. The difference table is as below:

x	$f(x)$	First Divided Differences	Second Divided Differences	Third Divided Differences	Fourth Divided Differences
1.0	7.989				
		0.414			
1.1	8.403		-0.036		
		0.378		0.006	
1.2	8.781		-0.03		<u>-0.002</u>
		0.348		<u>0.004</u>	
1.3	9.129		<u>-0.026</u>		
		<u>0.322</u>			
1.4	9.451				

We know that,

$$f'(x) = \frac{1}{h} \left\{ \nabla f(x_n) + \frac{1}{2!} (2s+1) \nabla^2 f(x_n) + \frac{1}{3!} (3s^2 + 6s + 2) \nabla^3 f(x_n) + \dots \right\}$$

Here,

$$h = 0.1 \quad \text{and} \quad s = \frac{(x - x_n)}{h} = \frac{(1.5 - 1.4)}{0.1} = 1$$

Thus,

$$\begin{aligned} f'(1.5) &= \frac{1}{0.1} \left\{ 0.322 + \left(\frac{1}{2}(2+1) \times (-0.026) + \frac{1}{6}(3+6+2) \times 0.006 + \frac{1}{24}(4+18+22+6) \times (-0.002) \right) \right\} \\ f'(1.5) &= \frac{1}{0.1} \{ 0.322 - 0.039 + 0.011 - 0.0042 \} = 2.898 \end{aligned}$$

```
//C program for calculating derivative using backward divided differences
#include<stdio.h>
#include<conio.h>
int fact(int n)
{
    if (n==1) return 1;
    else return(n*fact(n-1));
```

```

int main()
{
    int n,i,j,k;
    float val=0,p,xp,x[10],fx[10],bd[10],h,s,term1,term2,prev;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter values of x & f(x)\n");
    for(i=0;i<n;i++)
    {
        scanf("%f%f",&x[i],&fx[i]);
    }
    printf("Enter the value at which Derivative is needed\n");
    scanf("%f",&xp);
    h=x[1]-x[0];
    s=(xp-x[n-1])/h;
    //Calculation of backward differences
    for(i=0;i<n;i++)
    {
        bd[i]=fx[i];
    }
    for(i=n-1;i>0;i--)
    {
        for(j=0;j<i;j++)
        {
            bd[j]=(bd[j+1]-bd[j]);
        }
    }
    //Calculating value of first derivative
    val=bd[n-2];
    prev=1;
    for(i=2;i<n;i++)
    {
        term1=1;
        for(k=2;k<=i;k++)
        {
            term1=term1*(s+k-2);
        }
        term2=(s+i-1)*prev;
        prev=(term1+term2);
        val=val+(bd[n-i-1]*(term1+term2))/fact(i);
    }
    val=val/h;
    printf("Value of First Derivative=%f",val);
    getch();
    return 0;
}

```

Output

Enter the number of points

7

Enter values of x & f(x)

1.0 2.7183

1.2 3.3201

1.4 4.0552

1.6 4.9530

1.8 6.0496

2.0 7.3891

2.2 9.0250

Enter the value at which Derivative is needed

2.2

Value of First Derivative=9.022895

6.4 MAXIMA AND MINIMA OF TABULATED FUNCTIONS

We know that maximum and minimum values of a function can be computed by equating first derivative of the function and solving for unknown variables. The same concept can be applied to determine the maxima and minima of tabulated functions.

We know that, Newton's forward difference formula is given by

$$f(x) = f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \dots \quad (1)$$

$$\text{Where, } s = \frac{(x - x_0)}{h}$$

Differentiating equation (1) w.r.t. x, we get

$$f'(x) = \frac{1}{h} \left\{ \Delta f(x_0) + \frac{1}{2!} (2s-1) \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) + \dots \right\} \quad (2)$$

For maxima or minima $f'(x)$ must be zero. Terminating the terms after third order difference in RHS and equating it with zero, we get

$$\begin{aligned} \Delta f(x_0) + \frac{1}{2!} (2s-1) \Delta^2 f(x_0) + \frac{1}{3!} (3s^2 - 6s + 2) \Delta^3 f(x_0) &= 0 \\ \{\Delta f(x_0) - 1/2 \Delta^2 f(x_0) + 1/3 \Delta^3 f(x_0)\} + \{\Delta^2 f(x_0) - \Delta^3 f(x_0)\}s + 1/2 \Delta^3 f(x_0)s^2 &= 0 \\ as^2 + bs + c &= 0 \end{aligned} \quad (3)$$

Where,

$$a = \frac{1}{2} \Delta^3 f(x_0)$$

$$b = \Delta^2 f(x_0) - \Delta^3 f(x_0)$$

$$c = \Delta f(x_0) - 1/2 \Delta^2 f(x_0) + 1/3 \Delta^3 f(x_0)$$

Equation (3) is quadratic in 's' and can be solved. Then, values of x can be computed from the relation $x = x_0 + sh$

Algorithm

1. Start
2. Read number of data points, say n
3. Read n data points, say $x[i]$ and $f[x][i]$
4. Set $h=x[1]-x[0]$
5. Compute forward differences as below
- For $i=0$ to $n-1$
 $fd[i]=f[x][i]$
- End for
- For $i=0$ to $n-1$
 For $j=n-1$ to $i+1$
 $fd[j] = fd[j] - fd[j-1]$
- End for
- End For
- Calculate $a=(1/2.0)*fd[3]$ $b=fd[2]-fd[3]$ $c=fd[1]-(1/2.0)*fd[2]+((1/3.0)*fd[3])$
- Calculate $s1=(-b+\sqrt{b^2-4*a*c})/(2*a)$ $s2=(-b-\sqrt{b^2-4*a*c})/(2*a)$
- Calculate $x1=x[0]+s1*h$ $x2=x[0]+s2*h$
- Find point of maxima and minima as below
 Calculate $val=(fd[2]+(((6*s1-6)*fd[3])/6))/(h*h)$ // 2nd derivative of eq(1) wrt x
 if($val<0$)
 print "Maxima is at x1"
 else
 print "Minima is at x1"
 val= $(fd[2]+(((6*s2-6)*fd[3])/6))/(h*h)$
 if($val<0$)
 print "Maxima is at x2"
 else
 print "Minima is at x1"
10. Terminate

Example

Find maximum and minimum values of the function tabulated below:

x	0	1	2	3
$f(x)$	-5	-7	-3	13

Solution

The forward difference table is as below:

<i>x</i>	<i>f(x)</i>	First Divided Differences	Second Divided Differences	Third Divided Differences
0	-5			
		-2		
1	-7		6	
		4		6
2	-3		12	
		16		
3	13			

We know that,

$$as^2 + bs + c = 0$$

Where,

$$a = \frac{1}{2} \Delta^3 f(x_0)$$

$$b = \Delta^2 f(x_0) - \Delta^3 f(x_0)$$

$$c = \Delta f(x_0) - 1/2 \Delta^2 f(x_0) + 1/3 \Delta^3 f(x_0)$$

$$a = \frac{1}{2} \Delta^3 f(x_0) = \frac{1}{2} \times 6 = 3$$

$$b = \Delta^2 f(x_0) - \Delta^3 f(x_0) = 6 - 6 = 0$$

$$c = \Delta f(x_0) - 1/2 \Delta^2 f(x_0) + 1/3 \Delta^3 f(x_0) = -2 - \frac{1}{2} \times 6 + \frac{1}{3} \times 6 = -2 - 3 + 2 = -3$$

Thus, Equation (1) becomes

$$as^2 + bs + c = 0$$

$$3s^2 - 3 = 0$$

 \Rightarrow

$$s = \pm 1$$

Again

$$x = x_0 + sh$$

$$\text{Here, } x_0 = 0 \text{ and } h = 1 \\ \Rightarrow x = \pm 1$$

$$x = s$$

Again putting $s=x$ in Newton's forward difference formula

$$f(x) = (-5) + x(-2) + \frac{1}{2}x(x-1)6 + \frac{1}{6}x(x-1)(x-2)6$$

$$f(x) = x^3 - 3x - 5$$

$$\Rightarrow f'(x) = 3x^2 - 3$$

$$f''(x) = 6x$$

Therefore, we have maxima at $x=-1$ and minima at $x=1$

//C program for calculating maxima & minima of tabulated functions

```

//include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j;
    float val,x[10],fx[10],fd[10],h,s1,s2,x1,x2,a,b,c;
    printf("Enter the number of points\n");
    scanf("%d",&n);
    printf("Enter values of x & f(x)\n");
    for(i=0;i<n;i++)
    {
        scanf("%f%f",&x[i],&fx[i]);
    }

    h=x[1]-x[0];
    //Calculation of forward differences
    for(i=0;i<n;i++)
    {
        fd[i]=fx[i];
    }
    for(i=0;i<n;i++)
    {
        for(j=n-1;j>i;j--)
        {
            fd[j]=(fd[j]-fd[j-1]);
        }
    }

    a=(1/2.0)*fd[3];
    c=fd[1]-((1/2.0)*fd[2])+((1/3.0)*fd[3]);
    b=fd[2]-fd[3];

    //solving as^2+bs+c=0
    s1=(-b+sqrt(b*b-4*a*c))/(2*a);

```

```

s2=(-b-sqrt(b*b-4*a*c))/(2*a);
x1=x[0]+s1*h;
x2=x[0]+s2*h;

//Determining the point of maxima
val=(fd[2]+((6*s1-6)*fd[3])/6)/(h*h);
if(val<0)
printf("Maxima exists at x=%f\n",x1);
else
printf("Minima exists at x=%f\n",x1);

val=(fd[2]+((6*s2-6)*fd[3])/6)/(h*h);
if(val<0)
printf("Maxima exists at x=%f\n",x2);
else
printf("Minima exists at x=%f\n",x2)
getch();
return 0;
}

```

Output

Enter the number of points

5

Enter values of x & f(x)

1.2	0.9320
1.3	0.9636
1.4	0.9855
1.5	0.9975
1.6	0.9996

Minima exists at x=-8.669949

Maxima exists at x=1.568676

EXERCISE

1. Define the following formulas for first- and second-order derivatives: forward finite difference; backward finite-difference; centred finite-difference.
2. Explain how central difference formula can be derived, giving the error associated with it.
3. Illustrate the difference between two-point formula and three point formula using geometrical interpretation.
4. Derive a formula to estimate the third derivatives of tabulated functions using newton's divided difference interpolation.
5. Derive formula for computing second derivative of a function using taylor's series expansion.

6. Approximate the derivative of $f(x) = \cos x$ at $x = 0.1$ by central difference formula of order $O(h^2)$.
7. Compute value of first derivative for following functions at $x=1$ for $h=0.0$, using two-point formula and three point formula
- $\cos(x)$
 - $\cosh(x)$
 - $e^x \sin(x)$
 - $\log(2+x^2)$
 - $1+2x+x^2$
 - $1/x^2$
8. Use counter example to prove that accuracy of approximation $f'(x) = (f(x+h) - f(x))/h$ can be enhanced choosing small value of h .
9. The data given below give the distance covered by a body at a specified period. Calculate the velocity of the body at 0.3 second using Newton's divided differences.

Time(t)	0	0.1	0.2	0.3	0.4	0.5	0.6
Distance (d)	30.13	31.62	32.87	33.64	33.95	33.81	33.24

10. Find the value of dy/dx at $x = 2.03$ using the following table

x	1.96	1.98	2.0	20.2	2.04
$f(x)$	30.13	31.62	32.87	33.64	33.95

11. Find $f'(x)$ at $x = 1.5$ using the following table

x	1.5	2.0	2.5	3	3.5
$f(x)$	3.375	7.000	32.87	33.64	33.95

12. Find the value of $\cos 1.74$ using the table given below:

X	1.70	1.74	1.78	1.82	1.86
$f(x)$	0.9916	0.9857	0.9781	0.9691	0.9584

13. Using the table given below, determine the value of x for which y is maximum. Also find the maximum value of y .

X	1.2	1.3	1.4	1.5	1.6
$f(x)$	0.9320	0.9636	0.9855	0.9975	0.9996

□□□

Chapter 7

NUMERICAL INTEGRATION

Chapter Content

- ❑ Introduction
- ❑ Newton-Cotes Integration Formulae
- ❑ A General Quadrature Formula for Equally Spaced Arguments
- ❑ Gaussian Integration
- ❑ Romberg Integration

7.1 INTRODUCTION

Numerical integration is the process of computing the approximate value of a definite integral using a set of numerical values of the integrand. Integration is the process of measuring the area under a function plotted on a graph. It is represented by

$$\int_a^b f(x) dx$$

Where the symbol \int is an integral sign, and a and b are the lower and upper limits of integration, respectively. The function f is the integrand of the integral, and x is the variable of integration.

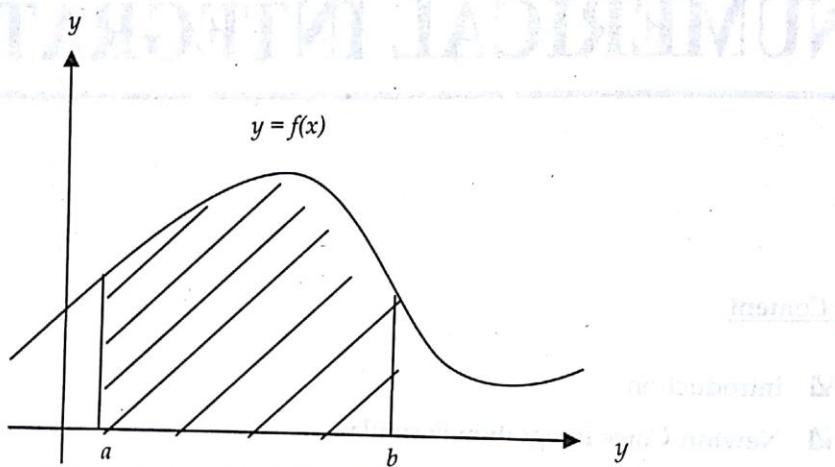


Figure 7.1: The definite integral as the area of a region under the curve, $\text{Area} = \int_a^b f(x) dx$

Why would we want to integrate a function? Among the most common examples are finding the velocity of a body from an acceleration function, and displacement of a body from a velocity function. Throughout many science & engineering fields, there are countless applications for integral calculus. Sometimes, the evaluation of expressions involving these integrals can become daunting, if not indeterminate. For this reason, a wide variety of numerical methods has been developed to simplify the integral. Methods of numerical integration can be divided into two groups: Newton-Cotes formulas and Gaussian quadrature. Newton-Cotes formulas are characterized by equally spaced abscissas, and include well-known methods such as the trapezoidal rule and Simpson's rule. They are most useful if $f(x)$ has already been computed at equal intervals, or can be computed at low cost. Since Newton-Cotes formulas are based on local interpolation, they require only a piecewise fit to a polynomial. In Gaussian quadrature the locations of the abscissas are chosen to yield the best possible accuracy. Since Gaussian quadrature requires fewer evaluations of the integrand for a given level of precision, it is popular in cases where $f(x)$ is expensive to evaluate.

7.2 NEWTON-COTES INTEGRATION FORMULAE

This is the most common numerical integration schemes. The strategy is to replace a complicated function or tabulated data with simpler polynomial function for easy integration. Newton-Cotes integration comes in following two forms:

- ✓ **Closed form** - If the limits of integration a and b are in the set of interpolating points then the formula is referred to as closed form
 - ✓ **Open form** - If the limits of integration a and b lie beyond the set of interpolating points then the formula is referred to as open form
- Since open form is not used for definite integration, in this chapter we only discuss about closed form methods. Some of the closed form methods are:
- Trapezoidal rule**
 - Simpson's 1/3 rule**
 - Simpson's 3/8 rule**

All of the above rules can be formulated by using interpolating polynomial for approximating the function $f(x)$. Here we will use Newton-Gregory forward difference formula for further discussion.

7.3 A GENERAL QUADRATURE FORMULA FOR EQUALLY SPACED ARGUMENTS

Let us suppose we have to evaluate $\int_{x_0}^{x_n} f(x) dx$ (1)

Let us divide the interval (x_0, x_n) into n sub-intervals of equal width. i.e $h = \frac{x_n - x_0}{n}$

Thus, $x_1 = x_0 + h, x_2 = x_0 + 2h, x_3 = x_0 + 3h, \dots, x_n = x_0 + nh$

From Newton-Gregory forward difference formula, we know that

$$f(x) = f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \dots \quad (2)$$

$$\text{Where, } s = \frac{(x - x_0)}{h}$$

Now equation (1) can be written as

$$\int_{x_0}^{x_n} f(x) dx = \int_{x_0}^{x_n} \left\{ f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \dots \right\} dx$$

$$\text{Since, } x = x_0 + sh \Rightarrow dx = h ds$$

$$\int_{x_0}^{x_n} f(x) dx = h \int_0^n \left\{ f(x_0) + s \Delta f(x_0) + \frac{1}{2!} s(s-1) \Delta^2 f(x_0) + \frac{1}{3!} s(s-1)(s-2) \Delta^3 f(x_0) + \dots \right\} ds$$

$$\Rightarrow \int_{x_0}^{x_n} f(x) dx = h \left[s f(x_0) + \frac{s^2}{2} \Delta f(x_0) + \frac{1}{2!} \left(\frac{s^3}{3} - \frac{s^2}{2} \right) \Delta^2 f(x_0) + \frac{1}{3!} \left(\frac{s^4}{4} - s^3 + s^2 \right) \Delta^3 f(x_0) + \dots \right]_0^n$$

$$\Rightarrow \int_{x_0}^{x_n} f(x) dx = h \left[n f(x_0) + \frac{n^2}{2} \Delta f(x_0) + \frac{1}{2!} \left(\frac{n^3}{3} - \frac{n^2}{2} \right) \Delta^2 f(x_0) + \frac{1}{3!} \left(\frac{n^4}{4} - n^3 + n^2 \right) \Delta^3 f(x_0) + \dots \right]$$

$$\Rightarrow \int_{x_0}^{x_n} f(x) dx = nh \left[f(x_0) + \frac{n}{2} \Delta f(x_0) + \frac{1}{12} (2n^2 - 3n) \Delta^2 f(x_0) + \frac{1}{24} (n^3 - 4n^2 + 4n) \Delta^3 f(x_0) + \dots \right] \quad (3)$$

This equation is called general quadrature formula. From this formula we can obtain different integration by putting $n = 1, 2, 3, \dots$ etc.

7.3.1 Trapezoidal Rule

The trapezoidal rule is based on the Newton-Cotes formula. The trapezoidal rule works by approximating the region under the graph of the function $f(x)$ as a trapezoid and calculates its area. The trapezoidal rule assumes $n = 1$. That is, it approximates the integral by a linear polynomial (straight line). General quadrature formula for integration is given by

$$\int_a^b f(x) dx = \int_{x_0}^{x_0+nh} f(x) dx = nh \left[f(x_0) + \frac{n}{2} \Delta f(x_0) + \frac{1}{12} (2n^2 - 3n) \Delta^2 f(x_0) + \frac{1}{24} (n^3 - 4n^2 + 4n) \Delta^3 f(x_0) + \dots \right] \quad (4)$$

By putting $n=1$ in above relation and neglecting higher order forward differences, it can be written as

$$\begin{aligned} \int_{x_0}^{x_0+h} f(x) dx &= \int_{x_0}^{x_1} f(x) dx = h \left[f(x_0) + \frac{1}{2} \Delta f(x_0) \right] \\ &= h \left[f(x_0) + \frac{1}{2} (f(x_1) - f(x_0)) \right] = \frac{(x_1 - x_0)}{2} (f(x_0) + f(x_1)) \\ \Rightarrow \int_{x_0}^{x_1} f(x) dx &= (x_1 - x_0) \frac{(f(x_1) - f(x_0))}{2} \end{aligned} \quad (2)$$

Equation (2) is called trapezoidal rule and it is the area of the trapezoid whose width is $(x_1 - x_0)$ and height is the average of $f(x_0)$ and $f(x_1)$.

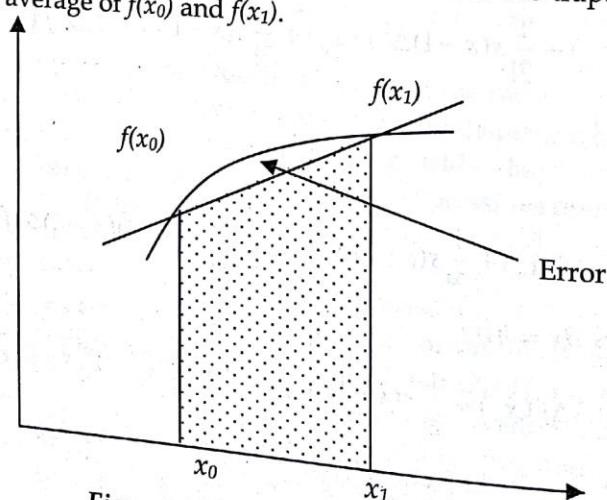


Figure 7.2: Geometric representation of trapezoidal rule.

Algorithm

1. Start
2. Read value of lower & upper limit, say x_0 & x_1
3. Calculate values $f(x_0)$ & $f(x_1)$
4. Calculate $h = (x_1 - x_0)$
5. Calculate the value of integration by using formula

$$v = \int_{x_0}^{x_1} f(x) dx = h \left[\frac{f(x_0) + f(x_1)}{2} \right]$$

6. Display the value of integration "v"
- Terminate

Example

Find $\int_2^8 \{x^3 + 2\} dx$ by using trapezoidal rule

Solution

Here, $x_0 = 2$ $x_1 = 8$

From Two point trapezoidal rule, we know that

$$\int_{x_0}^{x_1} f(x) dx = (x_1 - x_0) \left[\frac{f(x_0) + f(x_1)}{2} \right]$$

$$\Rightarrow \int_{x_0}^{x_1} f(x) dx = \int_2^8 \{x^3 + 2\} dx = (8 - 2) \left[\frac{10 + 514}{2} \right] = 1572$$

Second Example

Find $\int_0^1 e^{-x^2} dx$ by using trapezoidal rule

Solution

Here, $x_0 = 0$ $x_1 = 1$

From Two point trapezoidal rule, we know that

$$\int_{x_0}^{x_1} f(x) dx = (x_1 - x_0) \left[\frac{f(x_0) + f(x_1)}{2} \right]$$

=>

$$\int_{x_0}^{x_1} f(x) dx = (1 - 0) \left[\frac{1 + 0.368}{2} \right] = 0.684$$

/* C Program for Trapezoidal Rule

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) x+2
int main()
```

```

float h,x0,x1,fx0,fx1,v;
printf("Enter Lower & Upper Limit\n");
scanf("%f%f",&x0,&x1);
h=x1-x0;
fx0=f(x0);
fx1=f(x1);
v=h/2*(fx0+fx1);
printf("Value of Integration=%f\n",v);
getch();
return 0;
}

```

Output

```

Enter Lower & Upper Limit
0
2
Value of Integration=6.000000

```

7.3.2 Composite Trapezoidal Rule

In order to improve the accuracy of the trapezoidal rule, the integration interval can be divided into k segments of equal width. The equations are called the multiple-segment or composite integration formulae. Divide $(x_n - x_0)$ into k equal segments as shown in Figure below. Then the width of each segment is

$$h = \frac{x_n - x_0}{k}$$

Now, the integral can be broken into k integrals as

$$\int_{x_0}^{x_n} f(x) dx = \int_{x_0}^{x_0+h} f(x) dx + \int_{x_0+h}^{x_0+2h} f(x) dx + \cdots + \int_{x_0+(k-2)h}^{x_0+(k-1)h} f(x) dx + \int_{x_0+(k-1)h}^{x_n} f(x) dx \quad (1)$$

Applying trapezoidal rule on each segment, equation (1) gives

$$\begin{aligned}
& \int_{x_0}^{x_n} f(x) dx = \{(x_0 + h) - x_0\} \left\{ \frac{f(x_0) + f(x_0 + h)}{2} \right\} + \{(x_0 + 2h) - (x_0 + h)\} \left\{ \frac{f(x_0 + h) + f(x_0 + 2h)}{2} \right\} \\
& \quad \cdots + \{x_n - (x_0 + (k-1)h)\} \left\{ \frac{f(x_0 + (k-1)h) + f(x_n)}{2} \right\} \\
& \Rightarrow \int_{x_0}^{x_n} f(x) dx = h \left\{ \frac{f(x_0) + f(x_0 + h)}{2} \right\} + h \left\{ \frac{f(x_0 + h) + f(x_0 + 2h)}{2} \right\} + \cdots + h \left\{ \frac{f(x_0 + (k-1)h) + f(x_n)}{2} \right\} \\
& \Rightarrow \int_{x_0}^{x_n} f(x) dx = \frac{h}{2} \{f(x_0) + f(x_0 + h) + f(x_0 + 2h) + \cdots + f(x_0 + (k-1)h) + f(x_n)\} \\
& \Rightarrow \int_{x_0}^{x_n} f(x) dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_n) \right]
\end{aligned} \quad (2)$$

This equation (2) is called composite trapezoidal formula

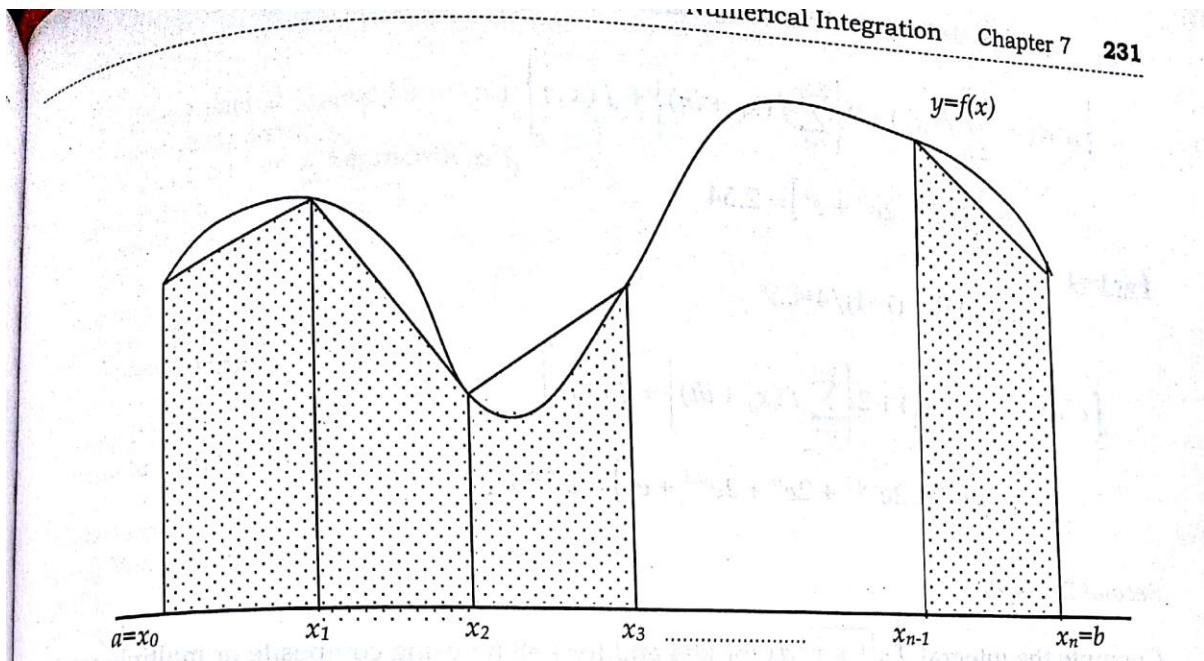


Figure 7.3: Multiple segment trapezoidal rule

Algorithm

1. Start
2. Read value of lower & upper limit, say x_0 & x_n
3. Read number of segments, say k
4. Calculate $h = (x_n - x_0)/k$
5. Set term = $f(x_0) + f(x_n)$
6. For $i=1$ to $k-1$
 - term = term + $2*f(x_0+i*h)$
7. End for
8. Calculate the value of integration by using formula $v = \frac{h}{2} * term$
9. Display the value of integration "v"
10. Terminate

Example

Compute the integral $\int_{-1}^1 e^x dx$ for $k=2$ and for $k=4$ by using composite or multiple-segment trapezoidal rule.

Solution

Here, $x_0=-1$ and $x_n=1$
For $k=2$

$$h = (x_n - x_0)/k = (1 - (-1))/2 = 1$$

$$\int_{-1}^1 e^x dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_2) \right]$$

$$= \frac{1}{2} [e^{-1} + 2e^0 + e^1] = 2.54$$

For k=4

$$h = (x_n - x_0)/k = (1+1)/4 = 0.5$$

$$\int_{-1}^1 e^x dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_2) \right]$$

$$= \frac{1}{2} [e^{-1} + 2e^{-0.5} + 2e^0 + 2e^{0.5} + e^1] = 2.399$$

Second Example

Compute the integral $\int_1^5 \sqrt{1+x^2} dx$ for k=4 and for k=8 by using composite or multiple-segment trapezoidal rule.

Solution

Here, $x_0=1$ and $x_n=5$

For k=4

$$h = (x_n - x_0)/k = (5-1)/4 = 1$$

$$\int_1^5 \sqrt{1+x^2} dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_2) \right]$$

$$= \frac{1}{2} [1.414 + 2 \times \{2.236 + 3.162 + 4.123\} + 5.099] = 12.78$$

For k=8

$$h = (x_n - x_0)/k = (5-1)/8 = 0.5$$

$$\int_1^5 \sqrt{1+x^2} dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_2) \right]$$

$$= \frac{0.5}{2} [1.414 + 2 \times \{1.803 + 2.236 + 2.696 + 3.162 + 3.64 + 4.123 + 4.609\} + 5.099] = 12.76$$

//C program for computing integration by using composite trapezoidal rule

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) 3*(x)*(x)+ 2*(x)-5
int main()
{
    float a,h,x0,xn,fx0,fxn,term,v;
```

```

int i,k;
printf("Enter Lower & Upper Limit\n");
scanf("%f%f",&x0,&xn);
printf("Enter Number of Segments\n");
scanf("%d",&k);
h=(xn-x0)/k;
fx0=f(x0);
fxn=f(xn);
term=f(x0)+f(xn);
for(i=1;i<=k-1;i++)
{
    a=x0+i*h;
    term=term+2*(f(a));
}
v=h/2*term;
printf("Value of Integration=%f\n",v);
getch();
return 0;
}

```

Output**First Run**

Enter Lower & Upper Limit

0 2

Enter Number of Segments

2

Value of Integration=3.00

Second Run

Enter Lower & Upper Limit

0 2

Enter Number of Segments

4

Value of Integration=2.250000

Third Run

Enter Lower & Upper Limit

0 2

Enter Number of Segments

8

Value of Integration=2.062500

7.3.3 Simpson's 1/3 Rule

The trapezoidal rule was based on approximating the integrand by a first order polynomial, and then integrating the polynomial over interval of integration. Simpson's 1/3 rule is an extension of Trapezoidal rule where the integrand is approximated by a second order polynomial. The Simpson's 1/3 rule assumes $n = 2$. General quadrature formula for integration is given by

$$\int_a^b f(x) dx = \int_{x_0}^{x_0+nh} f(x) dx = nh \left[f(x_0) + \frac{n}{2} \Delta f(x_0) + \frac{1}{12} (2n^2 - 3n) \Delta^2 f(x_0) + \frac{1}{24} (n^3 - 4n^2 + 4n) \Delta^3 f(x_0) + \dots \right] \quad (1)$$

$$\text{Here, } h = \frac{b-a}{n}$$

By putting $n=2$ in above relation and neglecting higher order forward differences, it can be written as

$$\begin{aligned} \int_{x_0}^{x_0+2h} f(x) dx &= \int_{x_0}^{x_2} f(x) dx = 2h \left[f(x_0) + \Delta f(x_0) + \frac{1}{6} \Delta^2 f(x_0) \right] \\ &= 2h \left[f(x_0) + (f(x_1) - f(x_0)) + \frac{1}{6} (f(x_0) - 2f(x_1) + f(x_2)) \right] \\ &= h \left[2f(x_0) + 2\{f(x_1) - f(x_0)\} + \frac{1}{3} (f(x_2) - 2f(x_1) + f(x_0)) \right] \\ \Rightarrow I &= \int_{x_0}^{x_2} f(x) dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] \quad (2) \end{aligned}$$

This equation (2) is called Simpson's 1/3 rule.

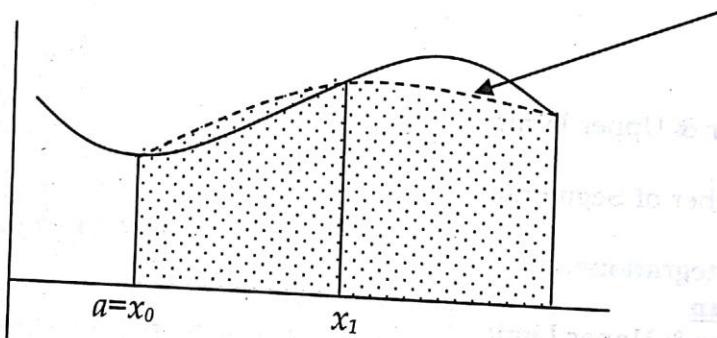


Figure: Geometrical Interpretation of Simpsons 1/3 rule

Algorithm

1. Start
 2. Read value of lower & upper limit, say x_0 & x_2
 3. Set $n=2$
 4. $h=(x_2-x_0)/n$
 5. $x_1=x_0+h$
 6. Calculate values $f(x_0)$, $f(x_1)$ and $f(x_2)$
 7. Calculate the value of integration by using formula
- $$v = \int_{x_0}^{x_2} f(x) dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)]$$
8. Display the value of integration "v"
 9. Terminate

Example

Apply Simpson's 1/3 rule to calculate $\int_0^1 \sqrt{1-x^2} dx$

Solution

$$\text{Here, } h = \frac{b-a}{n} = \frac{1-0}{2} = 0.5$$

Simpson's 1/3 rule is given by

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

$$\text{Since, } f(x) = \sqrt{1-x^2}$$

$$\Rightarrow f(x_0) = f(0) = 1$$

$$f(x_1) = f(x_0 + h) = f(0.5) = 0.866$$

$$f(x_2) = f(x_0 + 2h) = f(1) = 0$$

Thus,

$$\begin{aligned} I &= \int_{x_0}^{x_2} f(x) dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] \\ &= \frac{0.5}{3} [1 + 4 * 0.866 + 0] = 0.744 \end{aligned}$$

Second Example

Apply Simpson's 1/3 rule to calculate $\int_0^\pi \sin x dx$

Solution

$$\text{Here, } h = \frac{\pi - 0}{n} = \frac{\pi}{2}$$

Simpson's 1/3 rule is given by

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

Since,

$$f(x) = \int_0^\pi \sin x dx$$

\Rightarrow

$$\begin{aligned}f(x_0) &= f(0) = 0 \\f(x_1) &= f(x_0 + h) = f(\pi/2) = 1 \\f(x_2) &= f(x_0 + 2h) = f(\pi) = 0\end{aligned}$$

Thus,

$$\begin{aligned}I &= \int_{x_0}^{x_2} f(x) dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] \\&= \frac{\pi}{6} [0 + 4 \times 1 + 0] = 2.101\end{aligned}$$

//C program for computing integral value by using Simpson's 1/3 rule

```
#include<stdio.h>
#include<conio.h>
#define f(x) 3*(x)*(x)+ 2*(x)-5
int main()
{
    float h,x0,x1,x2,fx0,fx1,fx2,v;
    int n=2;
    printf("Enter Lower & Upper Limit\n");
    scanf("%f%f",&x0,&x2);
    h=(x2-x0)/n;
    x1=x0+h;
    fx0=f(x0);
    fx1=f(x1);
    fx2=f(x2);
    v=h/3*(fx0+4*fx1+fx2);
    printf("Value of Integration=%f\n",v);
    getch();
    return 0;
}
```

Output

```
Enter Lower & Upper Limit
0 2
Value of Integration=2.000000
```

7.3.4 Composite Simpson's 1/3 Rule

It is also called multi-segment Simpson's 1/3 rule. Just like in multiple-segment trapezoidal rule, one can subdivide the interval $[a, b]$ into k segments and apply Simpson's 1/3 rule repeatedly over every two segments. Note that k needs to be even. Divide interval $[a, b]$ into n equal segments, so that the segment width is given by

$$h = \frac{b-a}{k}$$

Apply Simpson's 1/3rd Rule over each two interval

$$\int_a^b f(x)dx = \int_{x_0}^{x_n} f(x)dx = \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \dots + \int_{x_{n-4}}^{x_{n-2}} f(x)dx + \int_{x_{n-2}}^{x_n} f(x)dx$$

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] + \frac{h}{3} [f(x_2) + 4f(x_3) + f(x_4)] + \dots$$

$$+ \frac{h}{3} [f(x_{n-4}) + 4f(x_{n-3}) + f(x_{n-2})] + \frac{h}{3} [f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

$$= \frac{h}{3} [f(x_0) + 4\{f(x_1) + f(x_3) + \dots + f(x_{n-1})\} + 2\{f(x_2) + f(x_4) + \dots + f(x_{n-2})\} + f(x_n)]$$

$$\Rightarrow \int_{x_0}^{x_n} f(x)dx = \frac{h}{3} \left[f(x_0) + 4 \sum_{\substack{i=1 \\ i=odd}}^{k-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=even}}^{k-2} f(x_i) + f(x_n) \right] \quad (3)$$

This equation (3) is called composite Simpson's 1/3 rule

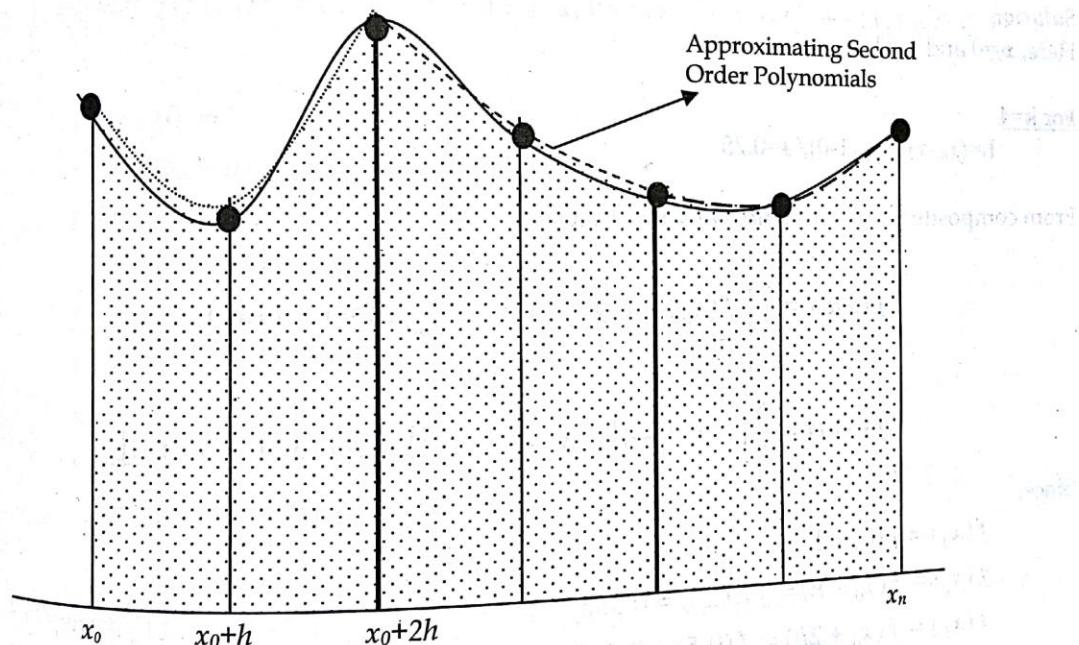


Figure: geometrical Interpretation of composite Simpson's 1/3 rule

Algorithm

1. Start
2. Read value of lower & upper limit, say x_0 & x_n
3. Read number of segments, say k
4. Calculate $h = (x_n - x_0)/k$
5. Set term = $f(x_0) + f(x_n)$
6. For $i=1$ to $k-1$
 - term = term + $4 * f(x_0 + i * h)$
 - $i = i + 2$

term = term + $4 * f(x_0 + i * h)$
 $i = i + 2$

7. End for
8. For i=2 to k-2
term=term + 2*f(x₀+i*h)
i=i+2
9. End for
10. Calculate the value of integration by using formula $v = \frac{h}{3} * term$
11. Display the value of integration "v"
12. Terminate

Example

Apply Simpson's 1/3 rule to calculate $\int_0^1 \sqrt{1-x^2} dx$ by using 4 segments (i.e. k=4) and 8 segments (i.e. k=8)

Solution

Here, $x_0=0$ and $x_n=1$

For k=4

$$h=(x_n-x_0)/k=(1-0)/4=0.25$$

From composite Simpson's rule, we know that

$$\int_{x_0}^{x_n} f(x) dx = \frac{h}{3} \left[f(x_0) + 4 \sum_{\substack{i=1 \\ i=odd}}^{k-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=even}}^{k-2} f(x_i) + f(x_n) \right]$$

$$\int_{x_0}^{x_n} f(x) dx = \frac{h}{3} [f(x_0) + 4\{f(x_1) + f(x_3)\} + 2\{f(x_2)\} + f(x_n)]$$

Since,

$$f(x_0) = f(0) = 1$$

$$f(x_1) = f(x_0 + h) = f(0.25) = 0.968$$

$$f(x_2) = f(x_0 + 2h) = f(0.5) = 0.866$$

$$f(x_3) = f(x_0 + 3h) = f(0.75) = 0.661$$

$$f(x_4) = f(x_0 + 4h) = f(1) = 0$$

Thus,

$$\begin{aligned}\int_{x_0}^{x_n} f(x) dx &= \frac{h}{3} [f(x_0) + 4\{f(x_1) + f(x_3)\} + 2\{f(x_2)\} + f(x_n)] \\ &= \frac{h}{3} [1 + 4\{0.968 + 0.661\} + 2\{0.866\} + 0] \\ &= \frac{0.25}{3} \{9.248\} = 0.771\end{aligned}$$

For $k=8$
 $h=(x_n - x_0)/k = (1+0)/8 = 0.125$

From composite Simpson's rule, we know that

$$\begin{aligned}\int_{x_0}^{x_n} f(x) dx &= \frac{h}{3} \left[f(x_0) + 4 \sum_{\substack{i=1 \\ i=odd}}^{k-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=even}}^{k-2} f(x_i) + f(x_n) \right] \\ &= \frac{h}{3} [f(x_0) + 4\{f(x_1) + f(x_3) + f(x_5) + f(x_7)\} + 2\{f(x_2) + f(x_4) + f(x_6)\} + f(x_n)]\end{aligned}$$

Since,

$$\begin{aligned}f(x_0) &= f(0) = 1 \\ f(x_1) &= f(x_0 + h) = f(0.125) = 0.992 \\ f(x_2) &= f(x_0 + 2h) = f(0.25) = 0.968 \\ f(x_3) &= f(x_0 + 3h) = f(0.375) = 0.927 \\ f(x_4) &= f(x_0 + 4h) = f(0.5) = 0.866 \\ f(x_5) &= f(x_0 + 5h) = f(0.625) = 0.787 \\ f(x_6) &= f(x_0 + 6h) = f(0.75) = 0.661 \\ f(x_7) &= f(x_0 + 7h) = f(0.875) = 0.484 \\ f(x_8) &= f(x_0 + 8h) = f(1) = 0\end{aligned}$$

Thus,

$$\begin{aligned}\int_{x_0}^{x_n} f(x) dx &= \frac{h}{3} [f(x_0) + 4\{f(x_1) + f(x_3) + f(x_5) + f(x_7)\} + 2\{f(x_2) + f(x_4) + f(x_6)\} + f(x_n)] \\ &\quad + 0 \\ \int_{x_0}^{x_n} f(x) dx &= \frac{h}{3} \{1 + 4(0.992 + 0.927 + 0.787 + 0.484) + 2(0.968 + 0.866 + 0.661)\} \\ \int_{x_0}^{x_n} f(x) dx &= \frac{h}{3} \{18.75\} = 0.78125\end{aligned}$$

Second Example

Apply Simpson's 1/3 rule to calculate $\int_0^\pi \sin x dx$ by using Simpson's 1/3 rule with 6 segments.

SolutionHere, $x_0 = 0$ and $x_n = \pi$ **For k=6**

$$h = (x_n - x_0)/k = \pi/6$$

From composite Simpson's rule, we know that

$$\int_{x_0}^{x_n} f(x) dx = \frac{h}{3} \left[f(x_0) + 4 \sum_{\substack{i=1 \\ i=odd}}^{k-1} f(x_i) + 2 \sum_{i=even}^{k-2} f(x_i) + f(x_n) \right]$$

$$\int_0^{\pi} f(x) dx = \frac{\pi}{3} [f(x_0) + 4\{f(x_1) + f(x_3) + f(x_5)\} + 2\{f(x_2) + f(x_4)\} + f(x_6)]$$

Since,

$$f(x_0) = f(0) = 0$$

$$f(x_1) = f(x_0 + h) = f(\pi/6) = 0.5$$

$$f(x_2) = f(x_0 + 2h) = f(\pi/3) = 0.866$$

$$f(x_3) = f(x_0 + 3h) = f(\pi/2) = 1$$

$$f(x_4) = f(x_0 + 4h) = f(2\pi/3) = 0.866$$

$$f(x_5) = f(x_0 + 5h) = f(5\pi/6) = 0.5$$

$$f(x_6) = f(x_0 + 6h) = f(\pi) = 0$$

Thus,

$$\begin{aligned} \int_0^{\pi} f(x) dx &= \frac{\pi}{3} [f(x_0) + 4\{f(x_1) + f(x_3) + f(x_5)\} + 2\{f(x_2) + f(x_4)\} + f(x_6)] \\ &= \frac{\pi}{18} [0 + 4\{0.5 + 1 + 0.5\} + 2\{0.866 + 0.866\} + 0] = 2.007 \end{aligned}$$

//C program for computing integral value using composite Simpsons 1/3 rule

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) 1/(1+x)
int main()
{
    float a,h,x0,xn,fx0,fxn,term,v;
    int i,k;
    printf("Enter Lower & Upper Limit\n");
    scanf("%f%f",&x0,&xn);
    printf("Enter Number of Segments\n");
    scanf("%d",&k);
    h=(xn-x0)/k;
    fx0=f(x0);
    fxn=f(xn);
    term=f(x0)+f(xn);
```

```

for(i=1;i<=k-1;i=i+2)
{
    a=x0+i*h;
    term=term+4*(f(a));
}
for(i=2;i<=k-2;i=i+2)
{
    a=x0+i*h;
    term=term+2*(f(a));
}
v=h/3*term;
printf("\nValue of Integration=%f\n",v);
getch();
return 0;
}

```

Output**First Run**

Enter Lower & Upper Limit

0 1

Enter Number of Segments

2

Value of Integration=0.694444

Second Run

Enter Lower & Upper Limit

0 1

Enter Number of Segments

4

Value of Integration=0.693254

6.3.5 Simpson's 3/8 Rule

As already discussed, Simpsons 1/3 rule for integration was derived by approximating the integrand $f(x)$ with a second order polynomial or quadratic polynomial function. In a similar fashion, Simpson 3/8 rule for integration can be derived by approximating the given function $f(x)$ with the third order polynomial or cubic polynomial.

General quadrature formula for integration is given by

$$\int_a^b f(x)dx = \int_a^{x_0+nh} f(x)dx = nh \left[f(x_0) + \frac{n}{2} \Delta f(x_0) + \frac{1}{12} (2n^2 - 3n) \Delta^2 f(x_0) + \frac{1}{24} (n^3 - 4n^2 + 4n) \Delta^3 f(x_0) + \dots \right] \quad (1)$$

Here, $h = \frac{b-a}{n}$

By putting $n=3$ in above relation and neglecting higher order forward differences, it can be written as

$$\begin{aligned}
 \int_{x_0}^{x_0+nh} f(x) dx &= \int_{x_0}^{x_0+3h} f(x) dx = 3h \left[f(x_0) + \frac{3}{2} \Delta f(x_0) + \frac{3}{4} \Delta^2 f(x_0) + \frac{1}{8} \Delta^3 f(x_0) \right] \\
 &= \frac{3}{8} h [8f(x_0) + 12(f(x_1) - f(x_0)) + 6(f(x_0) - 2f(x_1) + f(x_2)) + (-f(x_0) + 3f(x_1) - 3f(x_2) + f(x_3))] \\
 &= \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \\
 \Rightarrow I &= \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \quad (2)
 \end{aligned}$$

This equation (2) is called Simpson's 3/8 rule.

Algorithm

1. Start
2. Read value of lower & upper limit, say x_0 & x_3
3. Set $n=3$
4. Set $h=(x_3-x_0)/n$
5. Set $x_1=x_0+h$ $x_2=x_0+2h$
6. Calculate values $f(x_0)$, $f(x_1)$, $f(x_2)$ and $f(x_3)$
7. Calculate the value of integration by using formula

$$v = \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

8. Display the value of integration "v"
9. Terminate

Example

Apply Simpson's 3/8 rule to calculate $\int_0^1 \sqrt{1-x^2} dx$

Solution

$$\text{Here, } h = \frac{b-a}{3} = \frac{1-0}{3} = 0.33$$

Simpson's 3/8 rule is given by

$$I = \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

Since,

$$f(x) = \sqrt{1-x^2}$$

$$\Rightarrow$$

$$f(x_0) = f(0) = 1$$

$$f(x_1) = f(x_0 + h) = f(0.33) = 0.944$$

$$f(x_2) = f(x_0 + 2h) = f(0.66) = 0.751$$

$$f(x_3) = f(x_0 + 3h) = f(1) = 0$$

Thus,

$$\begin{aligned} I &= \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \\ &= \frac{3}{8} * 0.33 [1 + 3 * 0.944 + 3 * 0.751 + 0] = 0.753 \end{aligned}$$

Second Example

Apply Simpson's 3/8 rule to calculate $\int_0^2 2 + \cos(2\sqrt{x}) dx$

Solution

$$\text{Here, } h = \frac{2-0}{3} = 0.666$$

Simpson's 3/8 rule is given by

$$I = \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

Since,

$$f(x) = 2 + \cos(2\sqrt{x})$$

∴

$$f(x_0) = f(0) = 3$$

$$f(x_1) = f(x_0 + h) = f(0.666) = 1.939$$

$$f(x_2) = f(x_0 + 2h) = f(1.332) = 1.328$$

$$f(x_3) = f(x_0 + 3h) = f(2) = 1.049$$

Thus,

$$\begin{aligned} I &= \int_{x_0}^{x_3} f(x) dx = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \\ &= \frac{3}{8} \times 0.666 [3 + 3 \times 1.939 + 3 \times 1.328 + 1.049] = 3.46 \end{aligned}$$

*/*C program for computing integral value using Simpsons 3/8 rule*

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) (x)*(x)*(x)+1
int main()
{
    float h,x0,x1,x2,x3,fx0,fx1,fx2,fx3,v;
    int n=3;
    printf("Enter Lower & Upper Limit\n");
    scanf("%f%f",&x0,&x3);
    h=(x3-x0)/n;
    x1=x0+h;
    x2=x0+2*h;
    fx0=f(x0);
    fx1=f(x1);
```

```

fx2=f(x2);
fx3=f(x3);
v=3/8.0*h*(fx0+3*fx1+3*fx2+fx3);
printf("Value of Integration=%f\n",v);
getch();
return 0;
}

```

Output

Enter Lower & Upper Limit

1 2

Value of Integration=4.750000

6.3.6 Composite Simpson's 3/8 Rule

It is also called multi-segment Simpson's 3/8 rule or multiple segment 3/8 Simpson's rule. It divides the interval $[x_0, x_n]$ into n segments and apply Simpson's 3/8 rule repeatedly over every three segments. Therefore n needs to be multiple of 3. Now, the segment width is given by

$$h = \frac{x_n - x_0}{n}$$

Apply Simpson's 3/8 Rule over each three interval

$$\begin{aligned}
\int_{x_0}^{x_n} f(x) dx &= \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] + \frac{3}{8} h [f(x_3) + 3f(x_4) + 3f(x_5) + f(x_6)] + \dots \\
&\quad + \frac{3}{8} h [f(x_{n-6}) + 3f(x_{n-5}) + 3f(x_{n-4}) + f(x_{n-3})] + \frac{3}{8} h [f(x_{n-3}) + 3f(x_{n-2}) + 3f(x_{n-1}) + f(x_n)] \\
\Rightarrow \quad \int_{x_0}^{x_n} f(x) dx &= \frac{3}{8} h \left[f(x_0) + 3 \sum_{\substack{i=1 \\ i \bmod 3 \neq 0}}^{n-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i \bmod 3=0}}^{n-1} f(x_i) + f(x_n) \right]
\end{aligned} \tag{3}$$

This equation (3) is called composite Simpson's 3/8 rule

Algorithm

1. Start
2. Read value of lower & upper limit, say x_0 & x_n
3. Read number of segments, say k
4. Calculate $h = (x_n - x_0)/k$
5. Set term = $f(x_0) + f(x_n)$
6. For $i=1$ to $k-1$
 - if ($i \bmod 2 \neq 0$)
 $\text{term} = \text{term} + 3 * f(x_0 + i * h)$
 - else
 $\text{term} = \text{term} + 2 * f(x_0 + i * h)$
7. End for
8. Calculate the value of integration by using formula $v = \frac{3}{8} * h * \text{term}$

9. Display the value of integration "v"
10. Terminate

Example

Calculate the integral value of following tabulated function from $x=0$ to $x=1.6$ using Simpson's 3/8 rule.

x	0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6
$f(x)$	0	0.24	0.55	0.92	1.63	1.84	2.37	2.95	3.56

Solution

$$\text{Here, } h = 0.2$$

Composite Simpson's 3/8 rule is given by

$$I = \int_{x_0}^{x_n} f(x) dx = \frac{3}{8} h \left[f(x_0) + 3 \sum_{\substack{i=1 \\ i \bmod n \neq 0}}^{n-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i \bmod n = 0}}^{n-1} f(x_i) + f(x_n) \right]$$

$$\Rightarrow I = \frac{3}{8} h [f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + 3f(x_5) + 2f(x_6) + 3f(x_7) + 3f(x_8) + f(x_9)]$$

Thus,

$$I = \frac{3}{8} h [0 + 3 * 0.24 + 3 * 0.55 + 2 * 0.92 + 3 * 1.63 + 3 * 1.84 + 2 * 2.37 + 3 * 2.95 + 3.56]$$

$$= \frac{3}{8} * 0.2 [0 + 3 * 0.24 + 3 * 0.55 + 2 * 0.92 + 3 * 1.63 + 3 * 1.84 + 2 * 2.37 + 3 * 2.95 + 3.56] = 2.38$$

Second Example

Calculate the integral value of $\int_0^3 \frac{1}{x+4}$ by using composite Simpson's 3/8 rule with 9 segments

Solution

$$\text{Here, } h = \frac{3-0}{9} = 0.333$$

Composite Simpson's 3/8 rule is given by

$$I = \int_{x_0}^{x_n} f(x) dx = \frac{3}{8} h \left[f(x_0) + 3 \sum_{\substack{i=1 \\ i \bmod n \neq 0}}^{n-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i \bmod n = 0}}^{n-1} f(x_i) + f(x_n) \right]$$

For k=9

$$I = \frac{3}{8} h [f(x_0) + 3\{f(x_1) + f(x_2) + f(x_4) + f(x_5) + f(x_7) + f(x_8)\} + 2\{f(x_3) + f(x_6)\} + f(x_9)]$$

Since,

$$f(x) = \frac{1}{x+4}$$

\Rightarrow

$$\begin{aligned}
 f(x_0) &= f(0) = 0.25 \\
 f(x_1) &= f(x_0 + h) = f(0.333) = 0.23 \\
 f(x_2) &= f(x_0 + 2h) = f(0.666) = 0.214 \\
 f(x_3) &= f(x_0 + 3h) = f(1) = 0.2 \\
 f(x_4) &= f(x_0 + 4h) = f(1.333) = 0.188 \\
 f(x_5) &= f(x_0 + 5h) = f(1.666) = 0.176 \\
 f(x_6) &= f(x_0 + 6h) = f(2) = 0.167 \\
 f(x_7) &= f(x_0 + 7h) = f(2.333) = 0.158 \\
 f(x_8) &= f(x_0 + 8h) = f(2.666) = 0.15 \\
 f(x_9) &= f(x_0 + 9h) = f(3) = 0.143
 \end{aligned}$$

Thus,

$$\begin{aligned}
 I &= \frac{3}{8}h[0.25 + 3\{0.23 + 0.214 + 0.188 + 0.176 + 0.158 + 0.15\} + 2\{0.2 + 0.167\} + 0.143] \\
 &= \frac{3}{8} \times 0.333 \times 4.475 = 1.118
 \end{aligned}$$

//C program for computing integral value using composite Simpsons 3/8 rule

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) (x)*(x)*(x)+3*(x)*(x)
int main()
{
    float a,h,x0,xn,fx0,fxn,term,v;
    int i,k;
    printf("Enter Lower & Upper Limit\n");
    scanf("%f%f",&x0,&xn);
    printf("Enter Number of Segments\n");
    scanf("%d",&k);
    h=(xn-x0)/k;
    fx0=f(x0);
    f xn=f(xn);
    term=f(x0)+f(xn);
    for(i=1;i<=k-1;i++)
    {
        if(i%3!=0)
        {
            a=x0+i*h;
            term=term+3*(f(a));
        }
        else
        {
            a=x0+i*h;
            term=term+2*(f(a));
        }
    }
    v=3/8.0*h*term;
}

```

```

printf("\nValue of Integration=%f\n",v);
getch();
return 0;
}

```

Output**First Run**

Enter Lower & Upper Limit

2 4

Enter Number of Segments

6

Value of Integration=116.000008.

Second Run

Enter Lower & Upper Limit

2 4

Enter Number of Segments

9

Value of Integration=116.000000

6.4 GAUSSIAN INTEGRATION

Main idea behind Gaussian integration or quadrature is that the accuracy of numerical integration can be improved by choosing the sampling points wisely, rather than on the basis of equal spacing. Consider the figure given below: if we choose points x_1 and x_2 as in figure below rather than points a and b, integration error can be minimized dramatically. We should choose the points x_1 and x_2 such that $\text{Area } A \approx \text{Area } B + \text{Area } C$.

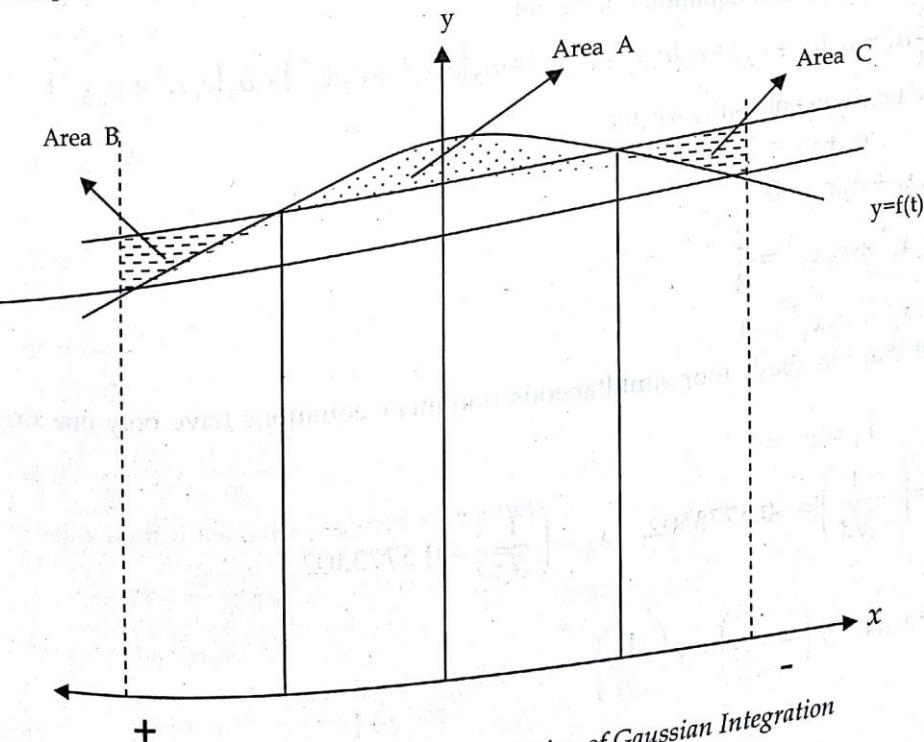


Figure : Geometrical Interpretation of Gaussian Integration

Thus, the two-point Gauss-Quadrature rule is an extension of the trapezoidal rule approximation where the arguments of the function are not predetermined as a and b , but as unknowns x_1 and x_2 . So in the two-point Gauss-Quadrature rule, the integral is approximated as

$$I = \int_{-1}^1 f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

There are four unknowns x_1 , x_2 , c_1 and c_2 . These are found by assuming that the formula gives exact results for integrating a general third order polynomial, $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$. Hence

$$\begin{aligned} \int_{-1}^1 f(x)dx &= \int_{-1}^1 (a_0 + a_1x + a_2x^2 + a_3x^3)dx \\ &= \left[a_0x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + a_3 \frac{x^4}{4} \right]_{-1}^1 \\ &= 2a_0 + \frac{2}{3}a_2 \end{aligned} \quad (1)$$

The formula gives

$$\begin{aligned} \int_{-1}^1 f(x)dx &\approx c_1 f(x_1) + c_2 f(x_2) = c_1(a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3) + c_2(a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3) \\ &= a_0(c_1 + c_2) + a_1(c_1x_1 + c_2x_2) + a_2(c_1x_1^2 + c_2x_2^2) + a_3(c_1x_1^3 + c_2x_2^3) \end{aligned} \quad (2)$$

Thus from equation (1) and equation (2), we get

$$2a_0 + \frac{2}{3}a_2 = a_0(c_1 + c_2) + a_1(c_1x_1 + c_2x_2) + a_2(c_1x_1^2 + c_2x_2^2) + a_3(c_1x_1^3 + c_2x_2^3)$$

Equating the terms on both side, we get

$$c_1 + c_2 = 2$$

$$c_1x_1 + c_2x_2 = 0$$

$$c_1x_1^2 + c_2x_2^2 = \frac{2}{3}$$

$$c_1x_1^3 + c_2x_2^3 = 0$$

We can find that the above four simultaneous nonlinear equations have only one acceptable solution

$$c_1 = c_2 = 1$$

$$\text{Hence } x_1 = \left(-\frac{1}{\sqrt{3}} \right) = -0.5773502, \quad x_2 = \left(\frac{1}{\sqrt{3}} \right) = 0.5773502$$

$$\int_{-1}^1 f(x)dx \approx f\left(-\frac{1}{\sqrt{3}} \right) + f\left(\frac{1}{\sqrt{3}} \right)$$

Since two points are chosen, it is called the two-point Gauss quadrature rule. Higher point versions can also be developed. Generalized n-point Gaussian quadrature rule is given as:

$$\int_{-1}^1 f(x)dx = \sum_{i=1}^n w_i f(x_i)$$

By using the procedure used in deriving two-point Gaussian quadrature rule, we can calculate the parameters w_i, z_i for higher order versions of Gaussian quadrature. Some parameters for Gaussian integration is listed below:

For $n=3$

$$\begin{array}{lll} w_1=0.55556 & w_2=0.88889 & w_3=0.55556 \\ z_1=-0.7746 & z_2=0 & z_3=-0.7746 \end{array}$$

For $n=4$

$$\begin{array}{llll} w_1=0.34785 & w_2=0.65215 & w_3=0.65215 & w_4=0.34785 \\ z_1=-0.86114 & z_2=-0.33998 & z_3=0.33998 & z_4=0.86114 \end{array}$$

Changing Limits

In the above discussion, Gaussian integration imposes the restriction on limits of integration from -1 to 1. By using the concept of "interval transformation", this restriction on limits can be relaxed.

Suppose,

$$\int_a^b f(x)dx = c \int_{-1}^1 g(z)dz$$

Let's assume the following transformation between x and z

$$x = Az + B$$

=>

$$a = B - A$$

$$b = A + B$$

=>

$$A = \frac{b-a}{2}$$

$$B = \frac{a+b}{2}$$

Thus,

$$x = \frac{b-a}{2}z + \frac{a+b}{2}$$

Differentiating both sides with respect to x we get

$$dx = \frac{b-a}{2} dz \quad \Rightarrow C = \frac{b-a}{2}$$

Thus the integral becomes

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^1 g(z)dz$$

Algorithm

1. Start
2. Read values of Lower & Upper Limit, say a & b
3. Set $c_1=c_2=1$, $z_1=-0.57735$, $z_2=0.57735$
4. Compute $x_1 = \frac{(b-a)}{2} z_1 + \frac{(b-a)}{2}$
5. Compute $x_2 = \frac{(b-a)}{2} z_2 + \frac{(b-a)}{2}$
6. Compute $v = \frac{b-a}{2} \{f(x_1) + f(x_2)\}$
7. Display value of integral, v
8. Terminate

Example

Compute the integral $\int_{-2}^2 e^{-x/2} dx$ using Gaussian two-point formula.

Solution

Here, $a=-2$ and $b=2$

After changing limits, we get

$$\int_a^b e^{-x/2} dx = \frac{b-a}{2} \int_{-1}^1 g(z) dz = 2 \int_{-1}^1 g(z) dz$$

$$x = \frac{b-a}{2} z + \frac{a+b}{2} = 2z$$

Thus,

$$\int_a^b e^{-x/2} dx = 2 \int_{-1}^1 e^{-z} dz$$

From, Gaussian two-point formula, we know that

$$\int_{-1}^1 f(x) dx = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

Thus,

$$\int_a^b e^{-x/2} dx = 2 \int_{-1}^1 e^{-z} dz = 2 \left\{ e^{-1/\sqrt{3}} + e^{1/\sqrt{3}} \right\} = 4.6854$$

Alternatively we can solve it as below:

We have,

$$a=-2 \text{ and } b=2$$

We know that

$$z_1=-0.57735$$

Thus,

$$z_2=0.57735$$

$$x_1 = \frac{(b-a)}{2} z_1 + \frac{(b+a)}{2} = 2 \times -0.57735 = -1.1547$$

$$x_2 = \frac{(b-a)}{2} z_1 + \frac{(b+a)}{2} = 2 \times 0.57735 = 1.1547$$

Now,

$$v = \frac{(b-a)}{2} (f(x_1) + f(x_2)) = 2(e^{1.1547/2} + e^{-1.1547/2}) = 4.6854$$

//C program for calculating integral by using two-point Gaussian quadrature

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) (x)*(x)*(x)+1
int main()
{
    float a,b,z1,z2,c1,c2,x1,x2,v;
```

```
printf("Enter Lower & Upper Limit\n");
scanf("%f %f",&a,&b);
```

```
//Set value of parameters
c1=c2=1;
z1=-0.57735, z2=0.57735;
```

```
//calculating xi
x1=(b-a)/2*z1+(b+a)/2;
x2=(b-a)/2*z2+(b+a)/2;
```

```
//Calculating Integral value
v=(b-a)/2*((f(x1))+f(x2)));
```

```
printf("\nValue of Integration=%f\n",v);
getch();
return 0;
}
```

Output

Enter Lower & Upper Limit

2 4

Value of Integration=61.99996

6.5 ROMBERG INTEGRATION

One way of reducing numerical error, which we will define as the difference between the exact result and the numerical result, is to decrease the step size. However, a second means of reducing the numerical error is to use a better integration rule. For example, for a given step size, Simpson's rule will generally yield a more accurate result than the Trapezoid rule. This more general method is called Romberg integration. The Euler-Maclaurin rule for integration includes higher powers of the step size than the Trapezoid rule and is given by

$$\int_{x_0}^{x_n} f(x) dx = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_n) \right] + \frac{h^2}{12} (f'(x_0) - f'(x_n)) - \frac{h^4}{720} (f'''(x_0) - f'''(x_n)) + \dots$$

We can now derive a new integration scheme - Romberg integration - which is correct to $O(h^4)$, etc.

$$\int_{x_0}^{x_n} f(x) dx = T_{m,0} + \alpha h^2 + \beta h^4 + \dots \quad (5)$$

Where,

$$T_{m,0} = \frac{h}{2} \left[f(x_0) + 2 \left\{ \sum_{i=1}^{k-1} f(x_0 + ih) \right\} + f(x_n) \right]$$

$n = 2^m$ (m integer) is the number of intervals the range of integration has been divided into. The meaning of the second subscript on T will become clear shortly. Now rewrite equation 5 for a step size, $h/2$, half of its original size. Obviously the number of intervals in the integration range has doubled so m increases by 1 and equation 5 becomes

$$\int_{x_0}^{x_n} f(x) dx = T_{m+1,0} + \alpha \left(\frac{h}{2}\right)^2 + \beta \left(\frac{h}{2}\right)^4 + \dots \quad (6)$$

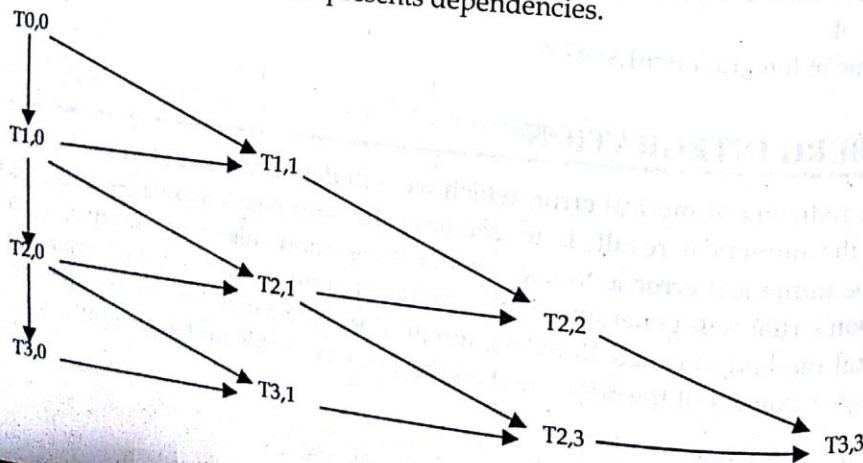
Now multiply equation 6 by 4, subtract equation 5 from it, and divide the result by 3 to obtain

$$\int_{x_0}^{x_n} f(x) dx = \frac{4T_{m+1,0} - T_{m,0}}{3} - \frac{\beta}{4} h^4 + \dots = T_{m+1,1} + O(h^4). \quad (7)$$

We now have an expression which is correct to $O(h^4)$. One set of differences has been used and so the second subscript is incremented from 0 to 1 and we will refer to this as the first order Romberg integration rule. We can now subtract two versions of equation 7 to eliminate errors up to $O(h^6)$. This procedure can be carried out to higher and higher order. The general relationship between T coefficients is

$$\int_a^b f(x) dx = T_{m+k,k} + O(h^{2(k+1)}) = \frac{4^k T_{m+k,k-1} - T_{m+k-1,k-1}}{4^k - 1} + O(h^{2(k+1)}). \quad (8)$$

This equation is called recursive Romberg Integration formula. Expanded form of this equation is shown in figure below, where arrows represent dependencies.



Elements in first column represents trapezoidal rule for intervals, $h, h/2, h/4, h/8$ etc. These can be evaluated as below.

$$T(0,0) = \frac{h}{2} (f(x_0) + f(x_1))$$

$$T(i,0) = \frac{T(i-1,0)}{2} + \frac{h}{2^i} \sum_{k=1}^{2^{i-1}} f(x_0 + (2k-1)h/2^i),$$

where $h = (x_1 - x_0)$

This equation is called recursive trapezoidal rule

Algorithm

1. Start
2. Read Lower & Upper Limit of integration, say x_0 & x_n
3. Compute $h = x_n - x_0$
4. Read p & q of required $T(p,q)$
5. Compute $T(0,0)$ using formula, $T(0,0) = h/2(f(x_0) + f(x_1))$
6. For $i=1$ to p

$$T[i][0] = \frac{T[i-1][0]}{2} + \frac{h}{2^i} \sum_{k=1}^{2^{i-1}} f(x_0 + (2k-1)h/2^i)$$

7. End for
8. For $c=1$ to p
 - For $k=1$ to c
 - $m=c-k$
 - if ($k \leq q$)

$$T[m+k][k] = (4^k * T[m+k][k-1] - T[m+k-1][k-1]) / 4^{k-1};$$
 - else

$$\text{break}$$
 - End for
9. End for
10. Display Romberg Estimate of integration, $T[p][q]$
11. Terminate

Example

Compute Romberg estimate $T(2,2)$ for $\int_0^1 1/(1+x) dx$

Solution

Use trapezoidal rule to compute $T(0,0)$

$$T(0,0) = \frac{h}{2} (f(x_0) + f(x_1)) = \frac{1}{2} (f(0) + f(1)) = 0.5 * (1 + 0.5) = 0.75$$

Now calculate $T(1,0)$ & $T(1,1)$ using recursive trapezoidal rule

$$T(1,0) = \frac{T(0,0)}{2} + \frac{h}{2^1} \sum_{k=1}^{2^{1-1}} f(x_0 + (2k-1)h/2^1) = \frac{T(0,0)}{2} + \frac{h}{2} f(x_0 + h/2) = \frac{0.75}{2} + 0.5 * f(0.5) = 0.7083$$

$$T(2,0) = \frac{T(1,0)}{2} + \frac{h}{2^2} \sum_{k=1}^{2^{2-1}} f(x_0 + (2k-1)h/2^2) = \frac{T(1,0)}{2} + \frac{h}{4} f(x_0 + h/4) + \frac{h}{4} f(x_0 + 3h/4)$$

$$= \frac{0.7083}{2} + \frac{1}{4} f(0.25) + \frac{1}{4} f(0.75) = 0.3541 + 0.2 + 0.1428 = 0.6969$$

Now use Romberg Integration formula to compute T(1,1), T(2,1) & T(2,2) as below:

$$T(1,1) = \frac{4 * T(1,0) - T(0,0)}{4-1} = \frac{4 * 0.7083 - 0.75}{3} = 0.6944$$

$$T(2,1) = \frac{4 * T(2,0) - T(1,0)}{4-1} = \frac{4 * 0.6969 - 0.7083}{3} = 0.6931$$

$$T(2,2) = \frac{16 * T(2,1) - T(1,1)}{16-1} = \frac{16 * 0.6931 - 0.6944}{15} = 0.6930$$

//C Program to estimate Romberg estimate T(p,q) of integration

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x) 1/(x)
int main()
{
    float x0,xn,T[10][10],h,sm,sl,a;
    int i,k,c,r,m,p,q;
    printf("Enter Lower & Upper Limit\n");
    scanf("%f%f",&x0,&xn);
    printf("Enter p & q of required T(p,q)\n");
    scanf("%d%d",&p,&q);
    h=xn-x0;
    T[0][0]=h/2*((f(x0))+(f(xn)));
}
```

```
for(i=1;i<=p;i++)
{
    sl=pow(2,i-1);
    sm=0;
    for(k=1;k<=sl;k++)
    {
        a=x0+(2*k-1)*h/pow(2,i);
        sm=sm+(f(a));
    }
    T[i][0]=T[i-1][0]/2+sm*h/pow(2,i);
}
```

```
for(c=1;c<=p;c++)
{
```

```

for(k=1;k<=c&&k<=q;k++)
{
    m=c-k;
    T[m+k][k]=(pow(4,k)*T[m+k][k-1]-T[m+k-1][k-1])/(pow(4,k)-1);
}
printf("\nRomberg Estimate of integration=%f\n",T[p][q]);
getch();
return 0;
}

Output
Enter Lower & Upper Limit
1   2
Enter p & q of required T(p,q)
2   2
Romberg Estimate of integration=0.693175

```

Appendix

for an integral $\int_{-1}^1 f(x)dx$, show that the two-point Gauss quadrature rule approximates to

$$\int_{-1}^1 f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

Where,

$$c_1 = 1 \quad c_2 = 1$$

$$x_1 = -\frac{1}{\sqrt{3}} \quad x_2 = \frac{1}{\sqrt{3}}$$

Solution

Assuming the formula

$$\int_{-1}^1 f(x)dx = c_1 f(x_1) + c_2 f(x_2) \quad (1)$$

Gives exact values for integrals $\int_{-1}^1 1dx$, $\int_{-1}^1 xdx$, $\int_{-1}^1 x^2dx$, and $\int_{-1}^1 x^3dx$. Then

$$\int_{-1}^1 1dx = 2 = c_1 + c_2 \quad (2)$$

$$\int_{-1}^1 xdx = 0 = c_1 x_1 + c_2 x_2 \quad (3)$$

$$\int_{-1}^1 x^2dx = \frac{2}{3} = c_1 x_1^2 + c_2 x_2^2 \quad (4)$$

$$\int_{-1}^1 x^3 dx = 0 = c_1 x_1^3 + c_2 x_2^3$$

Multiplying equation (3) by x_1^2 and subtracting from equation (5) gives

$$c_2 x_2 (x_1^2 - x_2^2) = 0$$

The solution to the above equation is

$$c_2 = 0, \text{ or/and}$$

$$x_2 = 0, \text{ or/and}$$

$$x_1 = x_2, \text{ or/and}$$

$$x_1 = -x_2.$$

I. $c_2 = 0$ is not acceptable as equations (2-5) reduce to $c_1 = 2$, $c_1 x_1 = 0$, $c_1 x_1^2 = \frac{2}{3}$, and $c_1 x_1^3 = 0$. But since $c_1 = 2$, then $x_1 = 0$ from $c_1 x_1 = 0$, but $x_1 = 0$ conflicts with $c_1 x_1^2 = \frac{2}{3}$.

II. $x_2 = 0$ is not acceptable as equations (2-5) reduce to $c_1 + c_2 = 2$, $c_1 x_1 = 0$, $c_1 x_1^2 = \frac{2}{3}$, and $c_1 x_1^3 = 0$. Since $c_1 x_1 = 0$, then c_1 or x_1 has to be zero but this violates $c_1 x_1^2 = \frac{2}{3} \neq 0$.

III. $x_1 = x_2$ is not acceptable as equations (2-5) reduce to $c_1 + c_2 = 2$, $c_1 x_1 + c_2 x_1 = 0$, $c_1 x_1^2 + c_2 x_1^2 = \frac{2}{3}$, and $c_1 x_1^3 + c_2 x_1^3 = 0$. If $x_1 \neq 0$, then $c_1 x_1 + c_2 x_1 = 0$ gives $c_1 + c_2 = 0$ and that violates $c_1 + c_2 = 2$. If $x_1 = 0$, then that violates $c_1 x_1^2 + c_2 x_1^2 = \frac{2}{3} \neq 0$.

This leaves the solution of $x_1 = -x_2$ as the only possible acceptable solution and in fact, it does not have violations (see it for yourself)

$$x_1 = -x_2$$

Substituting (7) in equation (3) gives

$$c_1 = c_2$$

From Equations (2) and (8),

$$c_1 = c_2 = 1$$

Equations (4) and (9) gives

$$x_1^2 + x_2^2 = \frac{2}{3}$$

Since equation (7) requires that the two results be of opposite sign, we get

$$x_1 = \frac{1}{\sqrt{3}}$$

$$x_2 = -\frac{1}{\sqrt{3}}$$

Hence

$$\int_{-1}^1 f(x) dx = c_1 f(x_1) + c_2 f(x_2) \\ = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \quad (11)$$

EXERCISE

- What is the physical significance of integration. Discuss Application areas of numerical integration.
- Why numerical integration is needed rather than analytical method of integration. Explain basic principle used in Newton-Cotes methods.
- Create flowchart of trapezoidal rule and composite trapezoidal rule. Verify that composite trapezoidal rule yields better performance than trapezoidal rule with suitable example.
- Show that Simpson's 1/3 rule is exact for integrating a polynomials degrees three or less by using a counter example.
- What is Romberg integration? How does it improve the accuracy of integration?
- Name at least two kinds of functions for which the Simpson sum always gives the exact answer, but not the trapezoid sum. Explain.
- What is the concept behind Gaussian Quadrature? Derive Gaussian three point formula for calculating integrals.
- Prove that Romberg integration with $k=1$ is actually Simpson's rule.
- Using regular partitions of size $n = 1, 2 & 4$, find the trapezoidal estimates for the definite integrals.

$$\int_0^1 \sqrt{1+x^2} dx$$

$$\int_{-1}^1 \arctan(x) dx$$

$$\int_0^1 e^{-x^2} dx$$

- Using regular partitions of size $n = 2 & 4$, calculate each integral using the trapezoid rule and Simpson's 1/3 rule.

$$\int_0^1 \sqrt{1+x^2} dx$$

$$\int_{-1}^1 \tan(x) dx$$

$$\int_0^1 e^{-x^2} dx$$

- Approximate the following integrals using composite simpsons 3/8 rule using 3 & 6 segments.

$$\int_0^1 \frac{4}{1+x^2} dx$$

$$\int_1^5 \log x dx$$

$$\int_0^3 (x^2 + x^3) dx$$

12. Evaluate $\int_0^1 \frac{1}{x} dx$ by Simpson's rule and compare the approximate value obtained with the exact solution.

13. The velocity v of a particle at distance s from a point on its path is given by the table: Using Simpson's one-third rule, determine the time taken by the particle to travel 60 ft.

s (ft)	0	10	20	30	40	50	60
V (velocity)	47	58	64	65	61	52	38

14. Use Romberg estimate to evaluate

s (ft)	0	10	20	30	40	50	60
V (velocity)	47	58	64	65	61	52	38

- 14 Use Romberg estimate to evaluate

$$\int_0^{\pi} \frac{\cos x}{\sqrt{1 + \sin x}} dx$$

$$\int_0^{\pi/2} e^x \sin x dx$$

$$\int_0^2 (3x^2 + 2)dx$$

Estimate the integral $\int_0^{\pi} (1 - \sin x) dx$ given below buy using two-point Gauss Quadrature formula

Chapter 8

SOLVING ORDINARY DIFFERENTIAL EQUATIONS

Chapter Content

- Introduction
- Solving Initial Value problems
- System of Ordinary Differential Equations
- Higher Order Differential Equations
- Solving Boundary Value Problems

8.1 INTRODUCTION

An equation which uses differential calculus to express relationship between variables is known as differential equations. Quantity being differentiated is called dependent variables and the quantity with respect to which the dependent variable is differentiated is called independent variable. The Differential equations have applications in all areas of science and engineering. Differential equations are used to model problems in science and engineering that involve the change of some variable with respect to another. Mathematical formulation of most of the physical and engineering problems leads to differential equations. So, it is important for engineers and scientists to know how to set up differential equations and solve them.

Differential equations are of two types:

- ✓ Ordinary differential equations (ODE)
- ✓ Partial differential equations (PDE)

8.1.1 Ordinary vs. Partial differential equations

An ordinary differential equation with single independent variable is called ordinary differential equation. Examples of ordinary differential equations include

$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + y = 0$$

$$\frac{d^3y}{dx^3} + 3\frac{d^2y}{dx^2} + 5\frac{dy}{dx} + y = \sin x$$

A differential equation with more than one independent variable is called partial differential equation. An example of such an equation would be

$$3\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = x^2 + y^2$$

Where u is the dependent variable, and x and y are the independent variables.

8.1.2 Order and Degree of Differential Equations

Ordinary differential equations are classified in terms of order and degree. Order of an ordinary differential equation is the same as the highest derivative and the degree of an ordinary differential equation is the power of highest derivative. Thus the differential equation

$$x^3 \frac{d^3 y}{dx^3} + x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + xy = e^x$$

is of order 3 and degree 1, whereas the differential equation

$$\left(\frac{dy}{dx} + 1\right)^2 + x^2 \frac{dy}{dx} = \sin x$$

is of order 1 and degree 2.

8.3 General vs. Particular Solutions of Differential Equations

Relationship between dependent and independent variables that satisfies differential equation is called solution of the differential equations. For example

$y = 3x^2 + x$ is the solution of $y' = 6x + 1$

A solution of the differential equation that contains arbitrary constants such that it can be modified to represent any condition is called general solution. For example

$y = 3x^2 + x + c$ is general solution of $y' = 6x + 1$

A particular solution is defined as a solution that satisfies the differential equation and some initial or boundary conditions. In another word, if the value of constant in general solution is known than the solution is called particular (unique) solution.

8.4 Initial vs. Boundary Values Problems

The solution of an ordinary differential equation requires auxiliary conditions. For an n^{th} order equation, n conditions are required. If all the conditions are specified at the same value of the independent variable, then the problem is called initial-value problem. For Example

Solve the equation $y' = x^2 + y^2$, given $y(0) = 1$

If the conditions are known at different values of the independent variable, usually at the extreme points or boundaries of a system, then the problem is known as boundary-value problem. For Example

Solve the equation $y' = y$, given $y(0) + y(1) = 2$

8.2 SOLVING INITIAL VALUE PROBLEMS**8.2.1 Taylors Series Method**

Taylor series expansion of function $y(x)$ about a point $x = x_0$ is given by the relation given below:

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + (x - x_0)^2 \frac{y''(x_0)}{2!} + \dots + (x - x_0)^n \frac{y^n(x_0)}{n!} \quad (1)$$

Value of the function $y(x)$ can be calculated if we know the values of its derivatives at some point. Thus, if we are given the equation $y' = f(x, y)$, we can differentiate it repeatedly and evaluate them at $x = x_0$. Finally these values can be substituted in equation (1) to obtain $y(x)$.

i.e. If $y' = f(x, y)$ then

$$\begin{aligned} y'' &= \frac{d}{dx} f(x, y) \\ &= \frac{\partial}{\partial x}[f(x, y)] + \frac{\partial}{\partial y}[f(x, y)] \frac{dy}{dx} \\ &= f_x + f_y f \end{aligned}$$

Where, f denotes the function $f(x, y)$, f_x denotes partial derivative of $f(x, y)$ with respect to x , and f_y denotes partial derivative of $f(x, y)$ with respect to y .

Similarly, we can get

$$y''' = f_{xx} + 2f f_{xy} + f^2 f_{yy} + f_x f_y + f f_y^2$$

Algorithm

1. Start
2. Read initial values, say x_0 & y_0
3. Read the value at which function to be evaluated, say x
4. Compute value of y' , y'' and y'''
5. Compute $v = y(x) = y_0 + (x-x_0)y' + (x-x_0)^2y''/2! + (x-x_0)^3y'''/3!$
6. Display functional value at x , i.e v
7. Terminate

Example

Given $y' = x - y^2$ with the initial condition $y = 1$ when $x = 0$. Find y for $x = 0.1$ by using first four terms of the series.

Solution

From Taylor's Series Method, we know that

$$y' = x - y^2$$

$$y'' = f_x + f_y f = 1 - 2y(x - y^2) = 1 - 2yy'$$

$$y''' = f_{xx} + 2f f_{xy} + f^2 f_{yy} + f_x f_y + f f_y^2$$

$$= 0 + 2y \times 0 + y'^2 \times -2 + 1 \times -2y + y \times 4y^2$$

$$= -2y^2 - 2y + 4y'y^2 = -2y^2 - 2y(1 - 2yy') = -2y^2 - 2yy''$$

Now, Values of derivatives can be calculated at $x=0$ as below:

$$y' = -1$$

$$y'' = 3$$

$$y''' = -8$$

Substituting above values in the Taylors series

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + (x - x_0)^2 \frac{y''(x_0)}{2!} + \dots + (x - x_0)^n \frac{y^n(x_0)}{n!}$$

We get,

$$y(x) = 1 - x + \frac{3}{2}x^2 - \frac{4}{3}x^3 + \dots$$

This is the solution of given differential equation

Now put $x=0.1$ in above equation, we get

$$y(0.1) = 0.913$$

Second Example

Solve the differential equation $y' = 3x^2$ such that $y = 1$ at $x = 1$. Find y for $x = 2$ by using first four terms.

Solution

From Taylor's Series Method, we have

$$y' = 3x^2$$

$$y'' = f_x + f_y f = 6x$$

$$\begin{aligned} y''' &= f_{xx} + 2f f_{xy} + f^2 f_{yy} + f_x f_y + f f_y \\ &= 6 \end{aligned}$$

Now, Values of derivatives can be calculated at $x=1$ as below:

$$y' = 3$$

$$y'' = 6$$

$$y''' = 6$$

Substituting above values in the Taylors series

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + (x - x_0)^2 \frac{y''(x_0)}{2!} + \dots + (x - x_0)^n \frac{y^n(x_0)}{n!}$$

We get,

$$\begin{aligned} y(x) &= 1 + (x - 1) \times 3 + (x - 1)^2 \times \frac{6}{2} + (x - 1)^3 \times \frac{6}{6} \\ &= 1 + 3(x - 1) + 3(x - 1)^2 + (x - 1)^3 \end{aligned}$$

This is the solution of given differential equation

Now put $x=2$ in above equation, we get

$$y(2) = 1 + 3 + 3 + 1 = 8$$

//C program to solve ODE by using Taylor's series method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int fact(int n)
{
    if(n==1)
        return 1;
    else
        return (n*fact(n-1));
}
int main()
{
    float x,x0,yx0,yx,f dy,s dy,t dy;
```

```

printf("Enter initial values of x & y \n");
scanf("%f %f",&x0,&yx0);
printf("Enter x at which function to be evaluated\n");
scanf("%f",&x);
fdy=(x0)*(x0)+(yx0)*(yx0); //First Derivative
sdy= 2*(x0) + 2*(yx0)*fdy; // Second Derivative
tdy=2+2*yx0*sdy+2*fdy*fdy;// Third Derivative
yx=yx0+(x-x0)*fdy+(x-x0)*(x-x0)*sdy/fact(2)+(x-x0)*(x-x0)*(x-x0)*tdy/fact(3);
printf("Function value at x=%f is %f\n",x,yx);
getch();
return 0;
}

```

Output

Enter initial values of x & y
 0 1
 Enter x at which function to be evaluated
 0.5
 Function value at x=0.500000 is 1.916667

8.2.2 Picard's Method

Consider the differential equation

$$y' = \frac{dy}{dx} = f(x, y) \quad (1)$$

with initial condition $y = y_0$ for $x = x_0$

Integrating the differential equation (1), we get

$$\begin{aligned}
 \int_{x_0}^{x_1} y' dx &= \int_{x_0}^{x_1} f(x, y) dx \\
 y(x_1) - y(x_0) &= \int_{x_0}^{x_1} f(x, y) dx \\
 y(x_1) &= y(x_0) + \int_{x_0}^{x_1} f(x, y) dx
 \end{aligned} \quad (2)$$

In equation (2) the y appears under the integral sign, so integration cannot be done. Either we need to replace y by constant or function of x . We know that $y = y_0$ for $x = x_0$. Thus, the first approximation y is obtained by replacing y by y_0 in $f(x, y)$ of equation (2) and integrating w.r.t. x , we get

$$y^{(1)} = y_0 + \int_{x_0}^{x_1} f(x, y_0) dx$$

The second approximation $y^{(2)}$ is obtained by replacing y by $y^{(1)}$ in $f(x, y)$ in R.H.S. of equation (2) and integrating w.r.t. x , we get

$$y^{(2)} = y_0 + \int_{x_0}^{x_1} f(x, y^{(1)}) dx$$

Proceeding in the same way we obtain $y^{(3)}, y^{(4)}, \dots, y^{n-1}$ and y^n

$$\text{Where, } y^{(n)} = y_0 + \int_{x_0}^{x_1} f(x, y^{(n-1)}) dx \quad (3)$$

$$\text{with } y^{(0)} = y_0$$

We repeat the steps till the two value of y becomes same or reaches to the desired degree of accuracy.

Algorithm of Picard's Method

1. Start
2. Read initial values of x & y , say x_0 & y_0
3. Read the value at which functional value is required, say x
4. Set $y = y_0$
5. Compute estimated value of y as below

$$\text{Compute } ny = y_0 + \int_{x_0}^{x_1} f(x, y) dx$$

$$\text{Compute Error} = \left| \frac{ny - y}{ny} \right|$$

If error is less than desired level of accuracy

Go to step 7

Else

Set $y = ny$

Go to step 5

6. Display the functional value, y
7. Terminate

Example

Solve the equation $y' = x^2 + y^2$ by using Picard's method with the initial condition $y(0) = 0$.

Find the value of $y(0.2)$ correct up to second approximation.

Solution

$$y' = x^2 + y^2 \quad y_0 = 0, \quad x_0 = 0$$

Since,

$$\Rightarrow y^{(1)} = y_0 + \int_{x_0}^x f(x, y_0) dx$$

$$\begin{aligned}y^{(1)} &= 0 + \int_0^x f(x, y_0) dx \\&= \int_0^x f(x, y_0) dx = \int_0^x x^2 + (y^0)^2 dx = \frac{x^3}{3}\end{aligned}$$

Since, $y^{(2)} = y_0 + \int_{x_0}^x f(x, y^{(1)}) dx$

=>

$$\begin{aligned}y^{(2)} &= 0 + \int_0^x f(x, y^{(1)}) dx \\&= \int_0^x x^2 + (y^1)^2 dx = \int_0^x \left(x^2 + \frac{x^3}{3}\right) dx = \frac{x^3}{3} + \frac{x^7}{63}\end{aligned}$$

If we stop at $y^{(2)}$, then

$$y(x) = \frac{x^3}{3} + \frac{x^7}{63} = 0.00026667$$

Second Example

Solve the equation $y' = 1 + xy$ by using Picard's method with the initial condition $y(0) = 1$. Find the value of $y(0.2)$ correct up to 3 decimal places.

Solution

Given,

$$y' = 1 + ty \quad y_0 = 1 \quad x_0 = 0$$

First Approximation

$$\begin{aligned}y^{(1)} &= y_0 + \int_{x_0}^x f(x, y_0) dx \\&=>\end{aligned}$$

$$\begin{aligned}y^{(1)} &= 1 + \int_0^x f(x, y^{(0)}) dx \\&= 1 + \int_0^x 1 + xy^{(0)} dx \\&= 1 + x + \frac{x^2}{2}\end{aligned}$$

At $x=0.2$

$$\begin{aligned}y(x) &= 1 + x + \frac{x^2}{2} \\&= 1 + 0.2 + 0.02 = 1.22\end{aligned}$$

Second Approximation

$$\begin{aligned}y^{(2)} &= y_0 + \int_{x_0}^x f(x, y^{(1)}) dx \\&=>\end{aligned}$$

$$\begin{aligned}
 y &= 1 + \int_0^x 1 + xy^{(1)} dx \\
 &= 1 + \int_0^x 1 + x \left\{ 1 + x + \frac{x^2}{2} \right\} dx \\
 &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8}
 \end{aligned}$$

At $x=0.2$

$$\begin{aligned}
 y(x) &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8} \\
 &= 1 + 0.2 + 0.02 + 0.00266 + 0.0002 = 1.22286
 \end{aligned}$$

Third Approximation

$$y^{(3)} = 1 + \int_{x_0}^{x_1} f(x, y^{(2)}) dx$$

$$\begin{aligned}
 y^{(3)} &= 1 + \int_0^x f(x, y^{(2)}) dx \\
 &= 1 + \int_0^x 1 + xy^{(2)} dx \\
 &= 1 + \int_0^x 1 + x \left\{ 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8} \right\} dx \\
 &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8} + \frac{x^5}{15} + \frac{x^6}{48}
 \end{aligned}$$

If we stop at $y^{(3)}$, we get

$$y(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8} + \frac{x^5}{15} + \frac{x^6}{48}$$

At $x=0.2$

$$\begin{aligned}
 y(x) &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{8} + \frac{x^5}{15} + \frac{x^6}{48} \\
 &= 1 + 0.2 + 0.02 + 0.00266 + 0.0002 + 0.0000213 + 0.00000133 = 1.222883
 \end{aligned}$$

//C program to solve ODE by using Picard's Method

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
#define y1(x) 2+(x)-2/3*pow(x,3)
#define y2(x) 2+(x)+pow(x,2)-2/3*pow(x,3)+pow(x,4)/4
#define y3(x) 2+(x)+pow(x,2)-pow(x,4)/3-pow(x,5)/15
int main()
{

```

```

float x,x0,y0,y,ny,er;
printf("Enter initial values of x & y \n");
scanf("%f%f",&x0,&y0);
printf("Enter x at which function to be evaluated\n");
scanf("%f",&x);
y=y0;
y=y0+y1(x); //First approximation
y=y0+y2(x); //Second approximation
y=y0+y3(x); //Third approximation
printf("Function value at x=%f is %f\n",x,y);
getch();
return 0;
}

```

Output

Enter initial values of x & y
0 2
Enter x at which function to be evaluated
0.4
Function value at x=0.400000 is 4.550784

8.2.3 Euler's Method

This is simplest and oldest method that was devised by Euler. It illustrates, the basic idea of those numerical methods which seek to determine the change Δy in y corresponding to a small increase in the arguments x . Consider the ordinary differential equations of the form

$$\frac{dy}{dx} = f(x, y), \quad y(0) = y_0 \quad (1)$$

We wish to solve equation (1), for the values of y at $x = x_i$, where $x_i = x_0 + ih \quad i = 1, 2, 3, \dots$. Now integrate equation (1), we get

$$\int_{x_0}^{x_1} y' = \int_{x_0}^{x_1} f(x, y) dx \Rightarrow y(x_1) = y(x_0) + \int_{x_0}^{x_1} f(x, y) dx \quad (2)$$

Assuming that $f(x, y) \approx f(x_0, y_0)$ in the range $x_0 \leq x \leq x_1$, equation (2) can be written as

$$\begin{aligned} y(x_1) &\approx y(x_0) + f(x_0, y_0) \int_{x_0}^{x_1} dx \\ &= y(x_0) + f(x_0, y_0)(x_1 - x_0) \\ &= y(x_0) + hf(x_0, y_0) \end{aligned}$$

Similarly, for the range $x_1 \leq x \leq x_2$, equation (2) can be written as
 $y(x_2) \approx y(x_1) + hf(x_1, y_1)$
Generalizing this, we can get

$$y(x_{i+1}) \approx y(x_i) + hf(x_i, y_i) \quad n = 0, 1, 2, 3, \dots \quad (3)$$

This equation (3) is called Euler equation. For better accuracy, we should choose small value of h . Therefore this method has slow convergence rate.

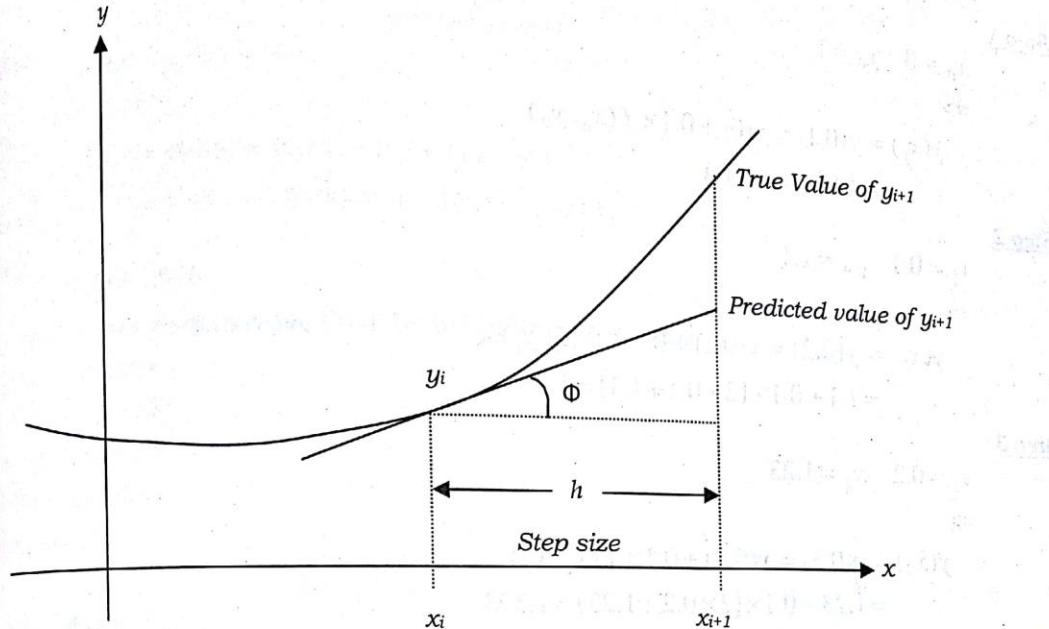


Figure: General graphical interpretation of Euler's method.

Algorithm

1. Start
2. Read initial values of x & y , say x_0 & y_0
3. Read the value at which functional value is required, say x_p
4. Read step size, say h
5. Set $y=x_0=y_0$
6. Compute value of y as below
For $x=x_0$ to x_p
 $y=y+f(x,y)$
 $x=x+h$
End for
7. Display functional value, y
8. Terminate

Example

Approximate the solution of the initial-value problem $y'=2x+y$, $y(0)=1$ by using Euler method with step size of 0.1. Approximate the value of $y(0.4)$.

Solution

Here,

$$f(x, y) = 2x + y$$

We know that,

$$y(x_{i+1}) = y(x_i) + hf(x_i, y_i)$$

Step 1

$$x_0 = 0 \quad y_0 = 1$$

=>

$$\begin{aligned} y(x_1) &= y(0.1) = y(0) + 0.1 \times f(x_0, y_0) \\ &= 1 + 0.1 \times 1 = 1.1 \end{aligned}$$

Step 2

$$x_1 = 0.1 \quad y_1 = 1.1$$

=>

$$\begin{aligned} y(x_2) &= y(0.2) = y(0.1) + 0.1 \times f(x_1, y_1) \\ &= 1.1 + 0.1 \times (2 \times 0.1 + 1.1) = 1.23 \end{aligned}$$

Step 3

$$x_2 = 0.2 \quad y_2 = 1.23$$

=>

$$\begin{aligned} y(x_3) &= y(0.3) = y(0.2) + 0.1 \times f(x_2, y_2) \\ &= 1.23 + 0.1 \times (2 \times 0.2 + 1.23) = 1.393 \end{aligned}$$

Step 4

$$x_3 = 0.3 \quad y_3 = 1.393$$

=>

$$\begin{aligned} y(x_4) &= y(0.4) = y(0.3) + 0.1 \times f(x_3, y_3) \\ &= 1.393 + 0.1 \times (2 \times 0.3 + 1.393) = 1.5923 \end{aligned}$$

Thus,

$$y(0.4) = 1.5923$$

Second Example

Approximate the solution of the initial-value problem $y' = x^2 + y^2$, $y(0) = 1$ by using Euler method with step size of 0.2. Approximate the value of $y(0.6)$.

Solution

Here,

$$f(x, y) = x^2 + y^2$$

We know that,

$$y(x_{i+1}) = y(x_i) + hf(x_i, y_i)$$

Step 1

$$x_0 = 0 \quad y_0 = 1$$

=>

$$\begin{aligned} y(x_1) &= y(0.2) = y(0) + 0.2 \times f(x_0, y_0) \\ &= 1 + 0.2 \times 1 = 1.2 \end{aligned}$$

Step 2 $x_1 = 0.2 \quad y_0 = 1.2$

$$\Rightarrow y(x_2) = y(0.4) = y(0.2) + 0.2 \times f(x_1, y_1) \\ = 1.2 + 0.2 \times (0.2^2 + 1.2^2) = 1.496$$

Step 3 $x_2 = 0.4 \quad y_2 = 1.496$

$$\Rightarrow y(x_3) = y(0.6) = y(0.4) + 0.2 \times f(x_2, y_2) \\ = 1.496 + 0.2 \times (0.4^2 + 1.496^2) = 1.976$$

Thus, $y(0.6) = 1.976$

//Program: C program to solve ODE by using Euler's method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x,y) 2*y/x
int main()
{
    float x,xp,x0,y0,y,h;
    printf("Enter initial values of x & y \n");
    scanf("%f %f",&x0,&y0);
    printf("Enter x at which function to be Evaluated\n");
    scanf("%f",&xp);
    printf("Enter the step size\n");
    scanf("%f",&h);
    y=y0;
    x=x0;
    for(x=x0;x<xp;x=x+h)
    {
        y=y+f(x,y)*h;
    }
    printf("Function value at x=%f is %f\n",xp,y);
    getch();
    return 0;
}
```

Output

Enter initial values of x & y

1 2

Enter x at which function to be Evaluated

2

Enter the step size

0.25

Function value at $x=2.0$ is 7.2**8.2.4 Heun's Method**

It is clear that in Euler's method the slope at (x_i, y_i) is used to estimate the value of $y(x_{i+1})$ below:

$$y(x_{i+1}) = y(x_i) + m_1 h, \quad m_1 = f(x_i, y_i) \quad (1)$$

Alternatively we can use the line that is parallel to the tangent at (x_{i+1}, y_{i+1}) to estimate the value of $y(x_{i+1})$ as below:

$$y(x_{i+1}) = y(x_i) + m_2 h, \quad m_2 = f(x_{i+1}, y_{i+1}) \quad (2)$$

If we look at the figure below, estimate given by equation (1) appears to be underestimated whereas the estimate given by equation (2) seems to be overestimated. Better estimate of $y(x_{i+1})$ is given by Heun's method. The main idea behind the Heun's method is to use the average of the slopes computed at the beginning and at the end of the interval (i.e. average of slopes m_1 and m_2).

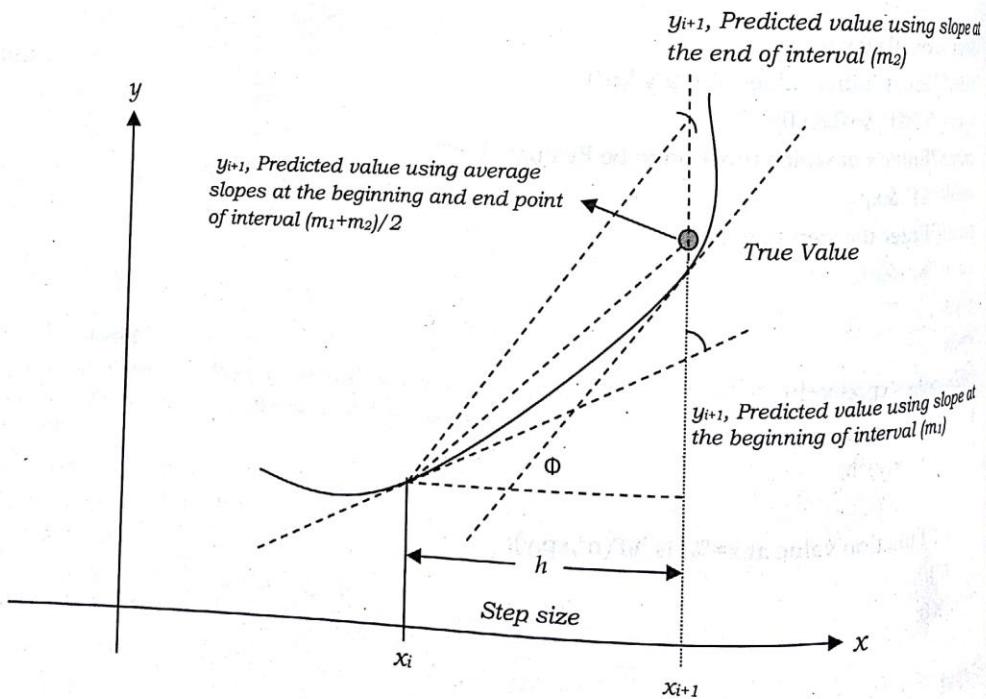


Figure: Geometrical Interpretation of Heun's Method

Thus, by using Heun's method we can estimate the value of $y(x_{i+1})$ as below:

$$\begin{aligned}y(x_{i+1}) &= y(x_i) + \frac{h}{2}(m_1 + m_1) \\&\Rightarrow y(x_{i+1}) = y(x_i) + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1}))\end{aligned}\quad (3)$$

Since $y(x_{i+1})$ appears on both side of equation (3), it cannot be evaluated until the value $y(x_{i+1})$ inside the function $f(x_{i+1}, y_{i+1})$ is available. This value $y(x_{i+1})$ inside the function $f(x_{i+1}, y_{i+1})$ can be predicted by using Euler formula as below:

$$y(x_{i+1}) = y(x_i) + hf(x_i, y_i)$$

Thus the Heun's formula given in (3) becomes

$$y(x_{i+1}) = y(x_i) + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_i) + hf(x_i, y_i)) \quad (4)$$

Algorithm

1. Start
2. Read initial values of x & y , say x_0 & y_0
3. Read the value at which functional value is required, say x_p
4. Read step size, say h
5. Set $y=x_0, y=y_0$
6. Compute values of $y(x_p)$ as below
For $x=x_0$ to x_p
 $m_1=f(x, y);$
 $m_2=f(x+h, y+h*m1);$
 $y=y+h/2*(m1+m2);$
- End for
7. Display functional value, y
8. Terminate

Example

Approximate the solution of the initial-value problem $y'=2x + y, y(0) = 1$ by using Heun's method with step size of 0.1. Approximate the value of $y(0.4)$

Solution

Iteration 1

$$\begin{aligned}i &= 0, \quad x_0 = 0, \quad y_0 = 1 \\m_1 &= f(x_0, y_0) = 1 \\m_2 &= f(x_1, y(x_1) + hf(x_0, y_0)) = f(0.1, 1 + 0.1 * 1) = f(0.1, 1.1) = 1.3 \\y(x_1) &= y(x_0) + \frac{h}{2}(m_1 + m_2) = 1 + \frac{0.1}{2}(1 + 1.3) = 1.115 \\&\Rightarrow y(0.1) = 1.115\end{aligned}$$

Iteration 2

$$i = 1, \quad x_1 = 0.1, \quad y_1 = 1.115$$

$$m_1 = f(x_1, y_1) = 1.315$$

$$m_2 = f(x_2, y(x_1) + hf(x_1, y_1)) = f(0.2, 1.115 + 0.1 * 1.315) = f(0.2, 1.2465) = 1.6465$$

$$y(x_2) = y(x_1) + \frac{h}{2}(m_1 + m_2) = 1.115 + \frac{0.1}{2}(1.315 + 1.6465) = 1.263$$

$$\Rightarrow y(0.2) = 1.263$$

Iteration 3

$$i = 2, \quad x_2 = 0.2, \quad y_2 = 1.263075$$

$$m_1 = f(x_2, y_2) = 1.663075$$

$$m_2 = f(x_3, y(x_2) + hf(x_2, y_2)) = f(0.3, 1.263 + 0.1 * 1.663) = f(0.3, 1.429) = 2.029$$

$$y(x_3) = y(x_2) + \frac{h}{2}(m_1 + m_2) = 1.263 + \frac{0.1}{2}(1.663 + 2.029) = 1.447$$

$$\Rightarrow y(0.3) = 1.447$$

Iteration 4

$$i = 3, \quad x_3 = 0.3, \quad y_3 = 1.4475$$

$$m_1 = f(x_3, y_3) = 2.0475$$

$$m_2 = f(x_4, y(x_3) + hf(x_3, y_3)) = f(0.4, 1.4475 + 0.1 * 2.0475) = f(0.4, 1.65225) = 2.45225$$

$$y(x_4) = y(0.4) = y(x_3) + \frac{h}{2}(m_1 + m_2) = 1.4475 + \frac{0.1}{2}(2.0475 + 2.45225) = 1.6724$$

$$\Rightarrow y(0.4) = 1.6724$$

Second Example

Approximate the solution of the initial-value problem $y' = x^2 + y$, $y(0) = 1$ by using Heun's method with step size of 0.05. Approximate the value of $y(0.2)$.

Solution Here $f(x, y) = x^2 + y$

Iteration 1

$$x_0 = 0 \quad y_0 = 1$$

$$m_1 = f(x_0, y_0) = 1$$

$$m_2 = f(x_1, y(x_0) + hf(x_0, y_0)) = f(0.05, 1 + 0.05 * 1) = f(0.05, 1.05) = 1.0525$$

$$y(x_1) = y(0.05) = y(x_0) + \frac{h}{2}(m_1 + m_2) = 1 + \frac{0.05}{2}(1 + 1.0525) = 1.0513$$

Iteration 2

$$x_1 = 0.05 \quad y_1 = 1.0513$$

$$m_1 = f(x_1, y_1) = 1.054$$

$$m_2 = f(x_2, y(x_1) + hf(x_1, y_1)) = f(0.1, 1.0513 + 0.05 \times 1.054) = f(0.1, 1.104) = 1.114$$

$$y(x_2) = y(0.1) = y(x_1) + \frac{h}{2}(m_1 + m_2) = 1.0513 + \frac{0.05}{2}(1.054 + 1.114) = 1.105$$

Iteration 3

$$x_2 = 0.1 \quad y_2 = 1.105$$

$$m_1 = f(x_2, y_2) = 1.115$$

$$m_2 = f(x_3, y(x_2) + hf(x_2, y_2)) = f(0.15, 1.105 + 0.05 \times 1.115) = f(0.15, 1.104) = 1.161$$

$$y(x_3) = y(0.15) = y(x_2) + \frac{h}{2}(m_1 + m_2) = 1.105 + \frac{0.05}{2}(1.115 + 1.161) = 1.162$$

Iteration 4

$$x_3 = 0.15 \quad y_3 = 1.162$$

$$m_1 = f(x_3, y_3) = 1.184$$

$$m_2 = f(x_4, y(x_3) + hf(x_3, y_3)) = f(0.2, 1.162 + 0.05 \times 1.184) = f(0.2, 1.22) = 1.221$$

$$y(x_4) = y(0.2) = y(x_3) + \frac{h}{2}(m_1 + m_2) = 1.162 + \frac{0.05}{2}(1.184 + 1.221) = 1.122$$

Thus, $y(0.2) = 1.122$

//C program for solving ODE by using Heun's method

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#define f(x,y) 2*(y)/(x)
```

```
int main()
```

```
{
```

```
float x,xp,x0,y0,y,h,m1,m2;
```

```
printf("Enter initial values of x & y \n");
```

```
scanf("%f%f",&x0,&y0);
```

```
printf("Enter x at which function to be Evaluated\n");
```

```
scanf("%f",&xp);
```

```
printf("Enter the step size\n");
```

```
scanf("%f",&h);
```

```
y=y0; x=x0;
```

```
for(x=x0;x<xp;x=x+h)
```

```
{
```

```
    m1=f(x,y);
```

```
    m2=f(x+h,y+h*m1);
```

```
    y=y+h/2*(m1+m2);
```

```
}
```

```
printf("Function value at x=%f is %f\n",xp,y);
```

```

    getch();
    return 0;
}

```

Output

Enter initial values of x & y
 1 2
 Enter x at which function to be Evaluated
 2
 Enter the step size
 0.25
 Function value at x=2.000000 is 7.860846

8.2.5 Fourth Order Runge-Kutta Method

Heun's method can be further refined by replacing the average slope of two points with a slope that is the weighted average of $f(x, y)$ at four points within the interval. This refinement in the Heun's method improves the order of approximation from h^2 to h^4 . This refinement was carried out by two German mathematicians C.D.T. Runge and M.W. Kutta using the Taylor series expansion with remainder of function $y(x)$.

Given the initial-value problem

$$\frac{dx}{dy} = f(x, y), \quad y(x_0) = y_0$$

for a fixed constant value of h; $y(x_n + h)$ can be approximated by

$$y(x_n + h) = y_{n+1} = y_n + \frac{1}{6}h(m_1 + 2m_2 + 2m_3 + m_4)$$

Where

$$m_1 = f(x_i, y_i) \quad \{ \text{This is the slope at } (x_i, y_i) \}$$

$$m_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h m_1\right) \quad \{ \text{The slope at the midpoint of the interval along the line connecting } (x_i, y_i) \text{ and } (x_i + h, y_i + h m_1) \}$$

$$m_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h m_2\right) \quad \{ \text{The slope at the midpoint of the interval along the line connecting } (x_i, y_i) \text{ and } (x_i + h, y_i + h m_2) \}$$

$$m_4 = f(x_i + h, y_i + h m_3) \quad \{ \text{Slope at } (x_i + h, y_i + h m_3) \}$$

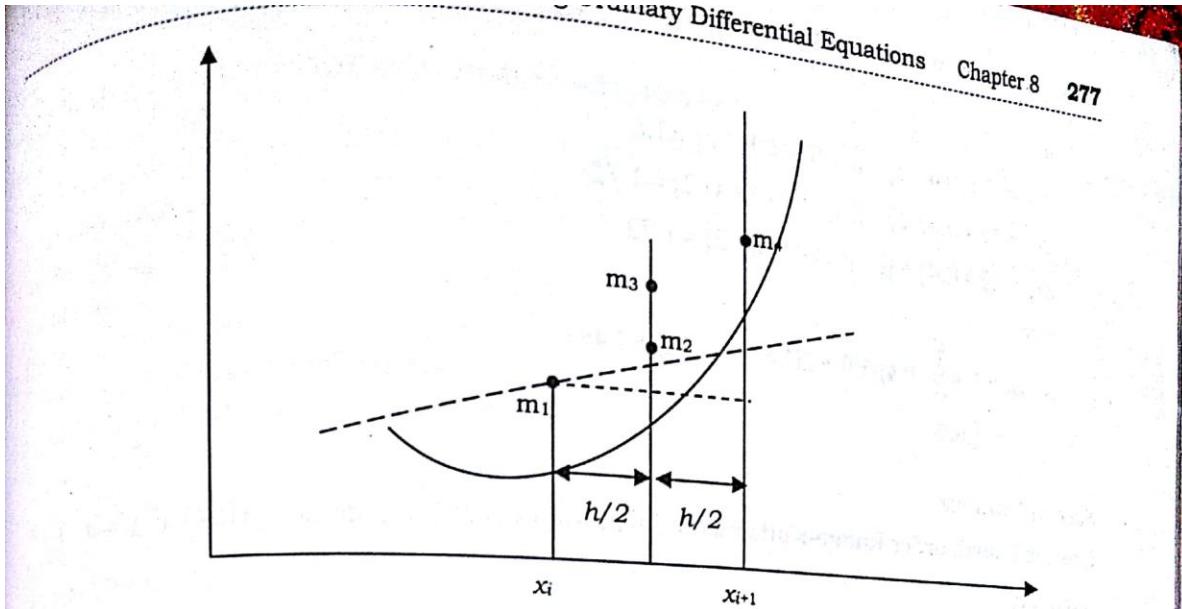


Figure: Geometrical Interpretation of 4th Order RK Method

Algorithm

1. Start
2. Read initial values of x & y , say x_0 & y_0
3. Read the value at which functional value is required, say x_p
4. Read step size, say h
5. Set $y=x_0, y=y_0$
6. Approximate value of y as below
 For $x=x_0$ to x_p

```

      m1=f(x,y);
      m2=f(x+h/2,y+h/2*m1);
      m3=f(x+h/2,y+h/2*m2);
      m4=f(x+h,y+h*m3);
      y=y+h/6*(m1+2*m2+2*m3+m4);
    
```
7. End for
8. Display functional value, y
9. Terminate

Example

Use the Runge-Kutta method to estimate $y(0.4)$ if $y'=2x+y$, $y(0)=1$

Solution

Here,

$$f(x, y) = 2x + y, \quad x_0 = 0, \text{ and } y_0 = 1, h = 0.4$$

Now from Runge-Kutta method, we have

$$\begin{aligned}m_1 &= \{2 \times 0 + 1\} = 1 \\m_2 &= \{2 \times 0.4 / 2\} + \{1 + (0.4 \times 1) / 2\} = 1.6 \\m_3 &= \{2 \times 0.4 / 2\} + \{1 + (0.4 \times 1.6) / 2\} = 1.72 \\m_4 &= \{2 \times 0.4\} + \{1 + (0.4 \times 1.6) / 2\} = 1.72\end{aligned}$$

Hence

$$\begin{aligned}y(0.4) &= 1 + \frac{1}{6} (0.4)[1.0 + 2(1.6) + 2(1.72) + 2.488] \\&= 1.675\end{aligned}$$

Second Example

Use the Fourth order Runge-Kutta method with step size 0.2 to estimate $y(0.4)$ if $y' = x^2 + y$
 $y(0) = 0$

Solution

Here,

$$f(x, y) = x^2 + y^2 \quad h = 0.2$$

Now,

Iteration 1

$$\begin{aligned}x_0 &= 0 \quad y_0 = 0 \\m_1 &= f(x_0, y_0) = f(0, 0) = 0 \\m_2 &= f\left(x_0 + \frac{0.2}{2}, y_0 + \frac{m_1 \times 0.2}{2}\right) = f(0.1, 0) = 0.01 \\m_3 &= f\left(x_0 + \frac{0.2}{2}, y_0 + \frac{m_2 \times 0.2}{2}\right) = f(0.1, 0.001) = 0.01 \\m_4 &= f(x_0 + 0.2, y_0 + m_3 h) = f(0.2, 0.002) = 0.04 \\y(x_1) &= y(0.2) = y(x_0) + \frac{0 + 2 \times 0.01 + 2 \times 0.01 + 0.04}{6} \times h = 0.00267\end{aligned}$$

Iteration 2

$$\begin{aligned}x_1 &= 0.2 \quad y_1 = 0.00267 \\m_1 &= f(x_1, y_1) = f(0.2, 0.00267) = 0.04 \\m_2 &= f\left(x_1 + \frac{0.2}{2}, y_1 + \frac{m_1 \times 0.2}{2}\right) = f(0.3, 0.00667) = 0.09004 \\m_3 &= f\left(x_1 + \frac{0.2}{2}, y_1 + \frac{m_2 \times 0.2}{2}\right) = f(0.3, 0.0117) = 0.090136 \\m_4 &= f(x_1 + 0.2, y_1 + m_3 h) = f(0.4, 0.0207) = 0.1604 \\y(x_2) &= y(0.4) = y(x_1) + \frac{0.04 + 2 \times 0.09004 + 2 \times 0.090136 + 0.1604}{6} \times h = 0.02135\end{aligned}$$

Thus,

6

$$y(0.4) = 0.02135$$

//C program for solving ODE using Runge-Kutta method

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f(x,y) 2*(x)+(y)
int main()
{
    float x,xp,x0,y0,y,h,m1,m2,m3,m4;
    printf("Enter initial values of x & y \n");
    scanf("%f %f",&x0,&y0);
    printf("Enter x at which function to be Evaluated\n");
    scanf("%f",&xp);
    printf("Enter the step size\n");
    scanf("%f",&h);
    y=y0;
    x=x0;
    for(x=x0;x<xp;x=x+h)
    {
        m1=f(x,y);
        m2=f(x+1/2.0*h,y+1/2.0*h*m1);
        m3=f(x+1/2.0*h,y+1/2.0*h*m2);
        m4=f(x+h,y+h*m3);
        y=y+h/6*(m1+2*m2+2*m3+m4);
    }
    printf("Function value at x=%f is %f\n",xp,y);
    getch();
    return 0;
}

```

Output

Enter initial values of x & y

0

Enter x at which function to be Evaluated

0.4

Enter the step size

0.1

Function value at x=0.4 is 1.675473

8.3 SOLVING SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS

Many mathematical problems require solving the system of several first order differential equations represented as below:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \dots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \dots, y_n)$$

⋮

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \dots, y_n)$$

The solution of such a system requires that n initial conditions be known at the starting value of x . Single-equation methods can be used to solve systems of ODE's as well. We can use Euler's method or Heun's method or Fourth order RK method for solving system of ordinary differential equations.

Algorithm for Solving system of Two ODE's

1. Start
2. Read initial values of x, y & z , say x_0, y_0, z_0
3. Read the value at which functional value is required, say x_p
4. Read step size, say h
5. Set $y=x_0, z=z_0$
6. Compute value of y as below
For $x=x_0$ to x_p //Using Euler's method
 $ny=y+f(x,y,z)*h$
 $nz=z+f(x,y,z)*h$
 $y=ny$ $z=nz$
 $x=x+h$
7. Display functional value, y

Example

Solve following two simultaneous first order differential equations with step size 0.25.

$$\frac{dy}{dx} = z = f_1(x, y, z), \quad y(0) = 1$$

$$\frac{dz}{dx} = e^{-x} - 2z - y = f_2(x, y, z), \quad z(0) = 2$$

Use Euler method to find $y(0.75)$

Solution

From Euler's method, we have

$$y_{i+1} = y_i + f_1(x_i, y_i, z_i)h$$

$$z_{i+1} = z_i + f_2(x_i, y_i, z_i)h$$

Here

$$f_1(x, y, z) = z \quad \text{and} \quad f_2(x, y, z) = e^{-x} - 2z - y$$

Iteration 1

$$x_0 = 0 \quad y_0 = 1 \quad z_0 = 2$$

$$y_1 = y_0 + f_1(x_0, y_0, z_0)h$$

$$= 1 + f_1(0, 1, 2)(0.25)$$

$$= 1 + 2(0.25)$$

$$= 1.5$$

$$\Rightarrow y(0.25) = 1.5$$

$$z_1 = z_0 + f_2(x_0, y_0, z_0)h$$

$$= 2 + f_2(0, 1, 2)(0.25)$$

$$= 2 + (e^{-0} - 2(2) - 1)(0.25)$$

$$= 1$$

$$\Rightarrow z(0.25) = 1$$

Iteration 2

$$x_1 = 0.25 \quad y_1 = 1 \quad z_1 = 1,$$

$$y_2 = y_1 + f_1(x_1, y_1, z_1)h$$

$$= 1.5 + f_1(0.25, 1.5, 1)(0.25)$$

$$= 1.5 + (1)(0.25)$$

$$= 1.75$$

$$\Rightarrow y(0.5) = 1.75$$

$$z_2 = z_1 + f_2(x_1, y_1, z_1)h$$

$$= 1 + f_2(0.25, 1.5, 1)(0.25)$$

$$= 1 + (e^{-0.25} - 2(1) - 1.5)(0.25)$$

$$= 1 + (-2.7211)(0.25)$$

$$= 0.31970$$

$$\Rightarrow z(0.5) = 0.31970$$

Iteration 3

$$x_2 = 0.5 \quad y_2 = 1.75 \quad z_2 = 0.31970$$

$$y_3 = y_2 + f_1(x_2, y_2, z_2)h$$

$$= 1.75 + f_1(0.50, 1.75, 0.31970)(0.25)$$

$$= 1.75 + (0.31970)(0.25)$$

$$\begin{aligned}
 &= 1.8299 \\
 \Rightarrow y(0.75) &= 1.8299 \\
 z_3 &= z_2 + f_2(x_2, y_2, z_2)h \\
 &= 0.31972 + f_2(0.50, 1.75, 0.31970)(0.25) \\
 &= 0.31972 + (e^{-0.50} - 2(0.31970) - 1.75)(0.25) \\
 &= 0.31972 + (-1.7829)(0.25) \\
 &= -0.1260 \\
 \Rightarrow z(0.75) &= -0.1260
 \end{aligned}$$

Thus, $y(0.75) = 1.8299$

Second Example

Solve following two simultaneous first order differential equations with step size 0.1.

$$\begin{aligned}
 \frac{dy}{dx} &= x + y + z \quad y(0) = 1 \\
 \frac{dz}{dx} &= 1 + y + z \quad z(0) = -1
 \end{aligned}$$

Use Heun's method to find $y(0.2)$

Solution

From Heun's method, we have

$$\begin{aligned}
 y_{i+1} &= y_i + \frac{h}{2}(m_1 + m_2) \\
 z_{i+1} &= z_i + \frac{h}{2}(m_1 + m_2)
 \end{aligned}$$

Here $f_1(x, y, z) = x + y + z$ $f_2(x, y, z) = 1 + y + z$

Iteration 1

$$\begin{aligned}
 x_0 &= 0 \quad y_0 = 1 \quad z_0 = -1 \\
 m_{1y} &= f_1(x_0, y_0, z_0) = 0 \\
 m_{1z} &= f_2(x_0, y_0, z_0) = 1 \\
 m_{2y} &= f_1(x_1, y(x_0) + hm_{1y}, z(x_0) + hm_{1z}) = f_1(0.1, 1, -0.9) = 0.2 \\
 m_{2z} &= f_2(x_1, y(x_0) + hm_{1y}, z(x_0) + hm_{1z}) = f_2(0.1, 1, -0.9) = 1.1 \\
 y(x_1) &= y(0.1) = y(x_0) + \frac{h}{2}(m_{1y} + m_{2y}) = 1 + \frac{0.1}{2}(0 + 0.2) = 1.01 \\
 z(x_1) &= z(0.1) = z(x_0) + \frac{h}{2}(m_{1z} + m_{2z}) = -1 + \frac{0.1}{2}(1 + 1.1) = -0.895
 \end{aligned}$$

Iteration 2

$$x_1 = 0.1 \quad y_1 = 1.01 \quad z_1 = -0.895$$

$$m_{1y} = f_1(x_1, y_1, z_1) = 0.215$$

$$m_{1z} = f_2(x_1, y_1, z_1) = 1.115$$

$$m_{2y} = f_1(x_2, y(x_1) + hm_{1y}, z(x_1) + hm_{1z}) = f_1(0.2, 1.0315, -0.7835) = 0.448$$

$$m_{2z} = f_2(x_2, y(x_1) + hm_{1y}, z(x_1) + hm_{1z}) = f_2(0.2, 1.0315, -0.7835) = 1.248$$

$$y(x_2) = y(0.2) = y(x_1) + \frac{h}{2}(m_{1y} + m_{2y}) = 1.01 + \frac{0.1}{2}(0.215 + 1.115) = 1.0765$$

$$z(x_2) = z(0.2) = z(x_1) + \frac{h}{2}(m_{1z} + m_{2z}) = -0.895 + \frac{0.1}{2}(0.215 + 1.115) = -0.828$$

Thus, $y(0.2) = 1.0765$

//C program for solving system of ODE's by using Euler's method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f1(x,y,z) z
#define f2(x,y,z) exp(-x)-2*(z)-(y)
int main()
{
    float x,xp,x0,y0,z0,y,z,ny,nz,h;
    printf("Enter initial values of x,y & z \n");
    scanf("%f%f%f",&x0,&y0,&z0);
    printf("Enter x at which function to be Evaluated\n");
    scanf("%f",&xp);
    printf("Enter the step size\n");
    scanf("%f",&h);
    y=y0;
    x=x0;
    z=z0;
    for(x=x0;x<xp;x=x+h)
    {
        ny=y+(f1(x,y,z))*h;
        nz=z+(f2(x,y,z))*h;
        y=ny;
        z=nz;
    }
    printf("Function value at x=%f is %f\n",xp,y);
    getch();
    return 0;
}
```

Output

Enter initial values of x,y & z

0 1 2

Enter x at which function to be Evaluated

0.75

Enter the step size

0.25

Function value at x=0.750000 is 1.829925

8.4 HIGHER ORDER DIFFERENTIAL EQUATIONS

Many Scientific and engineering problems are modeled as differential equations that are higher than first order. An n^{th} order differential equation has the form

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

with $n-1$ initial conditions it can be solved by assuming

$$y(x_0) = a_1, \quad y'(x_0) = a_2, \quad y''(x_0) = a_3, \quad \dots \quad y^{n-1}(x_0) = a_n$$

Let us denote

$$y = z_1, \quad \frac{dy}{dx} = z_2, \quad \frac{d^2 y}{dx^2} = z_3, \quad \dots \quad \frac{d^{n-1} y}{dx^{n-1}} = z_n$$

Then above Equation represents n first order differential equations as follows

$$\frac{dz_1}{dx} = z_2, \quad z_1(x_0) = z_{10} = a_1$$

$$\frac{dz_2}{dx} = z_3, \quad z_2(x_0) = z_{20} = a_2$$

⋮

$$\frac{dz_{n-1}}{dx} = z_n, \quad z_{n-1}(x_0) = z_{n-1,0} = a_{n-1}$$

$$\frac{dz_n}{dx} = f(x, z_1, z_2, \dots, z_m), \quad z_n(x_0) = z_{n0} = a_n$$

Each of the n first order ordinary differential equations are accompanied by one initial condition. These first order ordinary differential equations are simultaneous in nature and hence can be solved by the methods used for solving system of first order ordinary differential equations that we have already learned.

Algorithm for solving Second Order Differential Equation

1. Start
2. Read initial values of x & y, say x_0 & y_0
3. Read the value at which functional value is required, say x_p
4. Read step size, say h

5. Reduce the equation into system of differential equations
6. Set $x=x_0, y=y_0$
7. Compute value of y as below
For $x=x_0$ to x_p // Using Euler's method

$$\begin{aligned} ny &= y + f(x, y, z) * h \\ nz &= z + f(x, y, z) * h \\ y &= ny \\ z &= nz \\ x &= x + h \end{aligned}$$

End for
8. Display functional value, y
9. Terminate

Example

Rewrite the following differential equation as a set of first order differential equations

$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} - 3y = 6x \quad y(0) = 0 \quad y'(0) = 1$$

Find $y(0.2)$ by Euler's method with step size 0.1

Solution

First, the second order differential equation is rewritten as two simultaneous first-order differential equations as follows.

Assume

$$\frac{dy}{dx} = z$$

then

$$\frac{dz}{dx} + 2z - 3y = 6x$$

$$\frac{dz}{dx} = 6x + 3y - 2z$$

So the two simultaneous first order differential equations are

$$\frac{dy}{dx} = z, \quad y(0) = 0$$

$$\frac{dz}{dx} = 6x + 3y - 2z, \quad z(0) = 1$$

From Euler's method, we have

$$y_{i+1} = y_i + \frac{1}{2} f(x_i, y_i, z_i)h = y_i + \frac{1}{2} mh$$

$$z_{i+1} = z_i + \frac{1}{2} f(x_i, y_i, z_i)h = z_i + \frac{1}{2} mh$$

$$f_1(x, y, z) = z \quad \text{and} \quad f_2(x, y, z) = 6x + 3y - 2z$$

Iteration 1

$$\begin{aligned}x_0 &= 0 \quad y_0 = 0 \quad z_0 = 1 \\y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\&= 0 + f_1(0, 0, 1)h = 0.1 \\=> y(0.1) &= 0.1 \\z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\&= 1 + f_2(0, 0, 1)h = 0.8 \\=> z(0.1) &= 0.8\end{aligned}$$

Iteration 2

$$\begin{aligned}x_1 &= 0.1 \quad y_1 = 0.1 \quad z_1 = 0.8, \\y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\&= 0.1 + f_1(0.1, 0.1, 0.8)h = 0.18 \\=> y(0.2) &= 0.18 \\z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\&= 0.8 + f_2(0.1, 0.1, 0.8)h = 0.73 \\=> z(0.2) &= 0.73\end{aligned}$$

Thus, $y(0.2)=0.18$

Second Example

Rewrite the following differential equation as a set of first order differential equations.

$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + y = e^{-x}, \quad y(0)=1, \quad \frac{dy}{dx}(0)=2$$

Find $y(0.5)$ by Heun's method with step size 0.25

Solution

First, the second order differential equation is rewritten as two simultaneous first order differential equations as follows.

Assume

$$\frac{dy}{dx} = z$$

then

$$\begin{aligned}\frac{dz}{dx} + 2z + y &= e^{-x} \\ \frac{dz}{dx} &= e^{-x} - 2z - y\end{aligned}$$

So the two simultaneous first order differential equations are

$$\frac{dy}{dx} = z, \quad y(0) = 1$$

$$\frac{dz}{dx} = e^{-x} - 2z - y, \quad z(0) = 2$$

Using Heun's method on Equations (1) and (2), we get

$$y_{i+1} = y_i + \frac{1}{2}(m_1(1) + m_2(1))h$$

$$z_{i+1} = z_i + \frac{1}{2}(m_1(2) + m_2(2))h$$

Here

$$f_1(x, y, z) = z \quad \text{and} \quad f_2(x, y, z) = e^{-x} - 2z - y$$

Iteration 1

$$i = 0, x_0 = 0, y_0 = 1, z_0 = 2$$

$$m_1(1) = f_1(x_0, y_0, z_0) = f_1(0, 1, 2) = 2$$

$$m_1(2) = f_2(x_0, y_0, z_0) = f_2(0, 1, 2) = -4$$

$$\begin{aligned} m_2(1) &= f_1(x_0 + h, y_0 + hm_1(1), z_0 + hm_1(2)) \\ &= f_1(0 + 0.25, 1 + (0.25)(2), 2 + (0.25)(-4)) \\ &= f_1(0.25, 1.5, 1) \\ &= 1 \end{aligned}$$

$$\begin{aligned} m_2(2) &= f_2(x_0 + h, y_0 + hm_1(1), z_0 + hm_1(2)) \\ &= f_2(0 + 0.25, 1 + (0.25)(2), 2 + (0.25)(-4)) \\ &= f_2(0.25, 1.5, 1) \\ &= e^{-0.25} - 2(1) - 1.5 \\ &= -2.7212 \end{aligned}$$

Thus,

$$y_1 = y_0 + \frac{1}{2}(m_1(1) + m_2(1))h$$

$$= 1 + \frac{1}{2}(2 + 1)(0.25)$$

$$= 1.375$$

$$\Rightarrow y(0.25) = 1.375$$

$$z_1 = z_0 + \frac{1}{2}(m_1(2) + m_2(2))h$$

$$= 2 + \frac{1}{2}(-4 + (-2.7212))(0.25)$$

$$= 1.1598$$

$$\Rightarrow z(0.25) = 1.1598$$

Iteration 2

$$i = 1, x_1 = 0.25, y_1 = 1.375, z_1 = 1.1598$$

$$m_1(1) = f_1(x_1, y_1, z_1) = f_1(0.25, 1.375, 1.1598) = 1.1598$$

$$m_1(2) = f_2(x_1, y_1, z_1) = f_2(0.25, 1.375, 1.1598) = -2.9158$$

$$m_2(1) = f_1(x_1 + h, y_1 + hm_1(1), z_1 + hm_1(2))$$

$$= f_1(0.25 + 0.25, 1.375 + (0.25)(1.1598), 1.1598 + (0.25)(-2.9158))$$

$$= f_1(0.50, 1.6649, 0.43087)$$

$$= 0.43087$$

$$m_2(2) = f_2(x_1 + h, y_1 + hm_1(1), z_1 + hm_1(2))$$

$$= f_2(0.25 + 0.25, 1.375 + (0.25)(1.1598), 1.1598 + (0.25)(-2.9158))$$

$$= f_2(0.50, 1.6649, 0.43087)$$

$$= e^{-0.50} - 2(0.43087) - 1.6649$$

$$= -1.9201$$

Thus,

$$y_2 = y_1 + \frac{1}{2}(m_1(1) + m_2(1))h$$

$$= 1.375 + \frac{1}{2}(1.1598 + 0.43087)(0.25)$$

$$= 1.5738$$

$$\Rightarrow y(0.50) = 1.5738$$

$$z_2 = z_1 + \frac{1}{2}(m_1(2) + m_2(2))h$$

$$= 1.1598 + \frac{1}{2}(-2.9158 + (-1.9201))(0.25)$$

$$= 0.55533$$

$$\Rightarrow z(0.50) = 0.55533$$

Thus, $y(0.5) = 1.5738$

```
//Program: C program to solve Second order ODE using Heun's method
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f1(x,y,z) z
#define f2(x,y,z) -6*(x)+3*(y)-2*(z)
```

```

int main()
{
    float x,xp,x0,y0,z0,y,z,m1y,m2y,m1z,m2z,h;
    printf("Enter initial values of x,y & z \n");
    scanf("%f %f %f",&x0,&y0,&z0);
    printf("Enter x at which function to be Evaluated\n");
    scanf("%f",&xp);
    printf("Enter the step size\n");
    scanf("%f",&h);
    y=y0;
    x=x0;
    z=z0;
    for(x=x0;x<xp;x=x+h)
    {
        m1y=f1(x,y,z);
        m1z=f2(x,y,z);
        m2y=f1(x+h,y+h*m1y,z+h*m1z);
        m2z=f2(x+h,y+h*m1y,z+h*m1z);

        y=y+h/2*(m1y+m2y);
        z=z+h/2*(m1z+m2z);
    }
    printf("Function value at x=%f is %f\n",xp,y);
    getch();
    return 0;
}

```

Output

Enter initial values of x,y & z
 0 0 1
 Enter x at which function to be Evaluated
 0.2
 Enter the step size
 0.1
 Function value at x=0.200000 is 0.160800

8.5 SOLVING BOUNDARY VALUE PROBLEMS

In this section we show how to approximate the solution to boundary-value problems, differential equations with conditions imposed at different points. For first-order differential equations, only one condition is specified, so there is no distinction between initial-value and boundary-value problems. We will consider second-order equations with two boundary values.

Ordinary differential equations are given either with initial conditions or with boundary conditions. Look at the problem below.

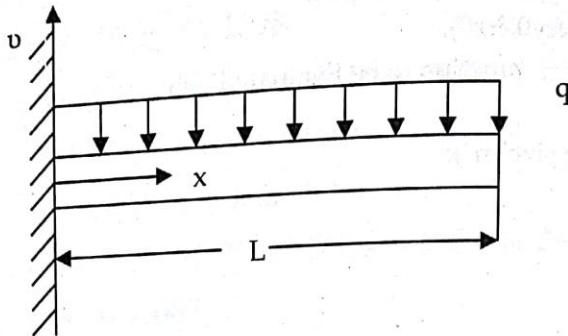


Figure: A cantilevered uniformly loaded beam.

To find the deflection v as a function of location x , due to a uniform load q , the ordinary differential equation that needs to be solved is

$$\frac{d^2v}{dx^2} = \frac{q}{2EI}(L-x)^2$$

Where

L is the length of the beam,

E is the Young's modulus of the beam, and

I is the second moment of area of the cross-section of the beam.

Two conditions are needed to solve the problem, and those are

$$v(0) = 0$$

$$\frac{dv}{dx}(0) = 0$$

As it is a cantilevered beam at $x = 0$. These conditions are initial conditions as they are given at an initial point, $x = 0$, so that we can find the deflection along the length of the beam. Now consider a similar beam problem, where the beam is simply supported on the two ends

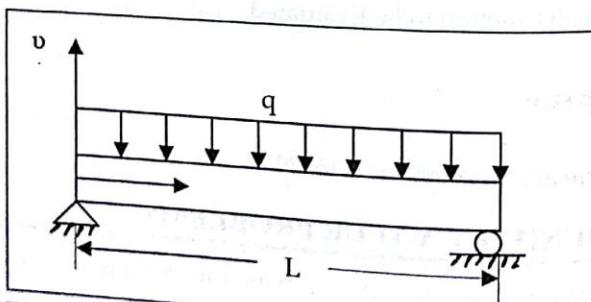


Figure 8.5: A simply supported uniformly loaded beam.

To find the deflection v as a function of x , due to the uniform load q , the ordinary differential equation that needs to be solved is

$$\frac{d^2v}{dx^2} = \frac{qx}{2EI}(x-L)$$

Two conditions are needed to solve the problem, and those are

$$v(0) = 0$$

$$v(L) = 0$$

As it is a simply supported beam at $x=0$ and $x=L$. These conditions are boundary conditions as they are given at the two boundaries, $x=0$ and $x=L$.

8.5.1 Shooting Method

In this method given boundary value problem is first transformed into equivalent initial value problem and then it is solved by using any of the method used for solving initial value problem. Thus main steps involved in shooting method are:

- ✓ Transform boundary value problem into equivalent initial value problem
- ✓ Get solution of initial value problem by using any existing method
- ✓ Get solution of boundary value problem

Consider the boundary value problem

$$y'' = f(x, y, y') \quad y(a) = u \quad y(b) = v$$

Let $y' = z$, now we can obtain following set of two equations

$$y' = z$$

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

To solve above initial value problem, we need to have two conditions at $x=a$. We have given one condition $y(a) = u$. Let's guess another condition is $z(a) = g_1$. Here g_1 represents slope of $y(x)$ at $x=a$. Thus the problem can be written as system of two first order equations as below:

$$\begin{aligned} y' &= z & y(a) &= u \\ z' &= f(x, y, z) & z(a) &= g_1 \end{aligned} \tag{1}$$

Now, equation (1) can be solved by using any method for solving initial value problem until the solution at $x=b$ reaches to specified accuracy level. Suppose, first estimated value of $y(x)$ at $x=b$ is given by $y(b) = v_1$. If $v_1 = v$ then we are done and it is the required solution. Otherwise, we should repeat the same process by taking second guess say g_2 . Suppose v_2 is the estimated value at $y(b)$ for second guess. If solution is not achieved from second guess, we can obtain better approximation by using linear interpolation as below:

$$\begin{aligned} \frac{g_3 - g_2}{v - v_2} &= \frac{g_2 - g_1}{v_2 - v_1} \\ \Rightarrow g_3 &= g_2 - \frac{v_2 - v}{v_2 - v_1} (g_2 - g_1) \end{aligned}$$

$$\begin{aligned} z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\ &= 10 + f_2(1.5, 5.5, 10)h = 14.5 \end{aligned}$$

Thus, $y(2)=10.5$
 The given value of this boundary condition is: $y(2)=9$.
 Since, predicted value of $y(2)$ is higher than actual value

Let us assume

$$z(1) = \frac{1}{2} \times \frac{y(2) - y(1)}{2 - 1} = 3.5$$

Calculation of Second Approximation

Iteration 1

$$x_0 = 1 \quad y_0 = 2 \quad z_0 = 3.5$$

$$\begin{aligned} y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\ &= 2 + f_1(1, 2, 3.5)h = 3.75 \end{aligned}$$

$$\begin{aligned} z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\ &= 3.5 + f_2(1, 2, 3.5)h = 6.5 \end{aligned}$$

Iteration 2

$$x_1 = 1.5 \quad y_1 = 3.75 \quad z_1 = 6.5$$

$$\begin{aligned} y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\ &= 3.75 + f_1(1.5, 3.75, 6.5)h = 7 \end{aligned}$$

$$\begin{aligned} z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\ &= 6.5 + f_2(1.5, 3.75, 6.5)h = 11 \end{aligned}$$

Thus, $y(2)=7$

Since, Predicted values $y(1.5)$ is lower than actual value

Use linear interpolation on the previous guesses to obtain new guess as below:

$$\begin{aligned} g_3 &= g_2 - \frac{v_2 - v}{v_2 - v_1} (g_2 - g_1) \\ &= 3.5 - \frac{7 - 9}{7 - 10.5} (3.5 - 7) = 3.5 + 0.57 \times 3.5 \approx 5.5 \end{aligned}$$

Thus, new guess (g_3)=5.5.

Calculation of Third Approximation

Iteration 1

$$x_0 = 1 \quad y_0 = 2 \quad z_0 = 5.5$$

$$\begin{aligned} y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\ &= 2 + f_1(1, 2, 5.5)h = 4.75 \end{aligned}$$

$$\begin{aligned} z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\ &= 5.5 + f_2(1, 2, 5.5)h = 8.5 \end{aligned}$$

Iteration 2

$$\begin{aligned} x_1 &= 1.5 \quad y_1 = 4.75 \quad z_1 = 8.5 \\ y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\ &= 0 + f_1(1.5, 4.75, 8.5)h = 9 \\ z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\ &= 8.5 + f_2(1.5, 4.75, 8.5)h = 13 \end{aligned}$$

Thus, $y(2)=9$ And the given value of this boundary condition is: $y(2)=9$.Thus, we can use third approximation to obtain value $y(1.5)$

$$\Rightarrow y(1.5)=4.75$$

Second Example

Solve the ordinary differential equation given below by using shooting method with Euler's method. And calculate the value of $y(3)$ and $y(6)$ by using $h=3$.

$$\frac{d^2y}{dx^2} - 2y = 72x - 8x^2 \quad y(0) = 0 \quad y(9) = 0$$

Solution

$$\text{Let } \frac{dy}{dx} = z$$

Then

$$\frac{dz}{dx} - 2y = 72x - 8x^2$$

This gives us two first order differential equations

$$\frac{dy}{dx} = z \quad y(0) = 0$$

$$\frac{dz}{dx} = 2y + 72x - 8x^2 \quad z(0) = \text{unknown}$$

Let us assume

$$z(0) = \frac{y(9) - y(0)}{9 - 0} = 0$$

Now, set up the initial value problem as

$$\frac{dy}{dx} = z \quad y(0) = 0$$

$$\frac{dz}{dx} = 2y + 72x - 8x^2 \quad z(0) = 0$$

Where,

$$\begin{aligned}f_1(x, y, z) &= z \\f_2(x, y, z) &= 2y + 72x - 8x^2\end{aligned}$$

From Euler's method, we know that

$$\begin{aligned}y_{i+1} &= y_i + f_1(x_i, y_i, z_i)h \\z_{i+1} &= z_i + f_2(x_i, y_i, z_i)h\end{aligned}$$

Calculate First Approximation

Iteration 1

$$\begin{aligned}x_0 &= 0 \quad y_0 = 0 \quad z_0 = 0 \\y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\&= 0 + f_1(0, 0, 0)h = 0 \\z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\&= 0 + f_2(0, 0, 0)h = 0\end{aligned}$$

Iteration 2

$$\begin{aligned}x_1 &= 3 \quad y_1 = 0 \quad z_1 = 0 \\y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\&= 0 + f_1(3, 0, 0)h = 0 \\z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\&= 0 + f_2(3, 0, 0)h = 432\end{aligned}$$

Iteration 3

$$\begin{aligned}x_2 &= 6 \quad y_2 = 0 \quad z_2 = 432 \\y_3 &= y_2 + f_1(x_2, y_2, z_2)h \\&= 0 + f_1(6, 0, 432)h = 1296 \\z_3 &= z_2 + f_2(x_2, y_2, z_2)h \\&= 0 + f_2(6, 0, 432)h = 432\end{aligned}$$

Thus, $y(9)=1296$

The given value of this boundary condition is: $y(9)=0$.

Since, predicted value of $y(9)$ is much higher than actual value

Assume that $z(0)=-10$

Calculation of Second Approximation

Iteration 1

$$\begin{aligned}x_0 &= 0 \quad y_0 = 0 \quad z_0 = -10 \\y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\&= 0 + f_1(0, 0, -10)h = -30\end{aligned}$$

$$\begin{aligned} z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\ &= 0 + f_2(0, 0, -10)h = 0 \end{aligned}$$

Iteration 2

$$\begin{aligned} x_1 &= 3 \quad y_1 = -30 \quad z_1 = 0 \\ y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\ &= 0 + f_1(3, -30, 0)h = 0 \\ z_2 &= z_1 + f_2(x_1, y_1, z_1)h \\ &= 0 + f_2(3, -30, 0)h = 252 \end{aligned}$$

Iteration 3

$$\begin{aligned} x_2 &= 6 \quad y_2 = 0 \quad z_2 = 252 \\ y_3 &= y_2 + f_1(x_2, y_2, z_2)h \\ &= 0 + f_1(6, 0, 252)h = 756 \\ z_3 &= z_2 + f_2(x_2, y_2, z_2)h \\ &= 0 + f_2(6, 0, 252)h = 432 \end{aligned}$$

Thus, $y(9)=756$

Since, Predicted values $y(9)$ is much higher than actual value

Use linear interpolation on the previous guesses to obtain new guess as below:

$$\begin{aligned} g_3 &= g_2 - \frac{v_2 - v}{v_2 - v_1}(g_2 - g_1) \\ &= -10 - \frac{756 - 0}{756 - 1296}(-10 - 0) = -10 - 1.4 \times 10 = -24 \end{aligned}$$

Thus, new guess (g_3)=-24

Calculation of Third ApproximationIteration 1

$$\begin{aligned} x_0 &= 0 \quad y_0 = 0 \quad z_0 = -24 \\ y_1 &= y_0 + f_1(x_0, y_0, z_0)h \\ &= 0 + f_1(0, 0, -24)h = -72 \\ z_1 &= z_0 + f_2(x_0, y_0, z_0)h \\ &= 0 + f_2(0, 0, -24)h = 0 \end{aligned}$$

Iteration 2

$$\begin{aligned} x_1 &= 3 \quad y_1 = -72 \quad z_1 = 0 \\ y_2 &= y_1 + f_1(x_1, y_1, z_1)h \\ &= 0 + f_1(3, -72, 0)h = 0 \\ z_2 &= z_1 + f_2(x_1, y_1, z_1)h \end{aligned}$$

$$= 0 + f_2(3, -72, 0)h = 0$$

Iteration 3

$$\begin{aligned}x_2 &= 6 \quad y_2 = 0 \quad z_2 = 0 \\y_3 &= y_2 + f_1(x_2, y_2, z_2)h \\&= 0 + f_1(6, 0, 0)h = 0 \\z_3 &= z_2 + f_2(x_2, y_2, z_2)h \\&= 0 + f_2(6, 0, 0)h = 432\end{aligned}$$

Thus, $y(9)=0$

And the given value of this boundary condition is: $y(9)=0$.

Thus, we can use third approximation to obtain value $y(3)$ and $y(6)$

$$\Rightarrow \quad y(3) = -72 \quad y(6) = 0$$

//C program for solving boundary value problem using shooting method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define f1(x,y,z) (z)
#define f2(x,y,z) 6*(x)
int main()
{
    float xa, xb, ya, yb, x0, y0, z0, x, y, z, xp, h, sol, ny, nz, error, E, g[3], v[3], gs;
    int i;
    printf("Enter Boundary Conditions\n");
    scanf("%f %f %f %f", &xa, &ya, &xb, &yb);
    printf("Enter x at which value is required\n");
    scanf("%f", &xp);
    printf("Enter the step size\n");
    scanf("%f", &h);
    printf("Enter accuracy limit\n");
    scanf("%f", &E);
    x=xa; y=ya;
    g[1]=z=(yb-ya)/(xb-xa);
    printf("g=%f\n", g[1]);
    while(x<xb) //Using Euler's method
    {
        ny=y+(f1(x,y,z))*h;
        nz=z+(f2(x,y,z))*h;
        x=x+h;
        y=ny;
        z=nz;
```

```

if(x==xp)
sol=y;
}
v[1]=y;
if(y>=b)
g[2]=z=2*g[1]
else
g[2]=z=1/2*g[1]
printf("g=%f\n",g[2]);
while(x<xb) //Using Euler's method
{
ny=y+(f1(x,y,z))*h;
nz=z+(f2(x,y,z))*h;
x=x+h;
y=ny;
z=nz;
if(x==xp)
sol=y;
}
while(1)
{
xa=x; ya=y;
gs=z=g[2]-(v[2]-yb)/(v[2]-v[1])*(g[2]-g[1]);
while(x<xb)
{
ny=y+(f1(x,y,z))*h;
nz=z+(f2(x,y,z))*h;
x=x+h;
y=ny;
z=nz;
if(x==xp)
sol=y;
}
error=fabs(y-yb)/y;
v[1]=v[2]; v[2]=y;
g[1]=g[2]; g[2]=gs;
if(error<E)
{
printf("y(%f)=%f",xp,sol);
break;
}
}

```

```

    getch();
    return 0;
}

```

Output

Enter Boundary Conditions

1 2
2 9

Enter x at which value is required

1.5

Enter the step size

0.5

Enter accuracy limit

0.01

y(1.50000)=4.750000

EXERCISE

1. What is differential equation? How partial differential equations are different from ordinary differential equations?
2. What is meant by solution of differential equation? Distinguish boundary value problems from initial value problems.
3. How can you use Tailor's series method and Picard's method for solving ODE's? Justify with suitable example.
4. Develop flowcharts for solving ODE's using Euler's, Heun's and 4th order Runge-Kutta methods.
5. Why Heun's method is more correct than Euler's method? Justify your answer with suitable geometric interpretation.
6. Write down the algorithm for solving system of differential equations using Heun's method.
7. Write down the algorithm for solving boundary value problem using shooting method in combination with fourth order Runge-Kutta method.
8. Solve the following by Taylor Series Method.
 - a) $Y' = x^2y - 1; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - b) $Y' = 3x + y^2; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - c) $Y' = -xy; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - d) $Y' = x + y^2; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - e) $Y' = 2y - 3e^x; y(0) = 0$; find $y(0.1), y(0.2), y(0.3)$
 - f) $Y' = y - xy; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - g) $Y' = x - y^2; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
 - h) $Y' = x + y^2; y(0) = 0$; find $y(0.1), y(0.2), y(0.3)$

Solve the following by Picard's Method by giving third degree of approximation.

9. a) $y' = 1 + xy; y(0) = 0$; find $y(0.1), y(0.2), y(0.3)$
- b) $y' = y - x^2; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
- c) $y' = x + y; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
- d) $y' = x + y^2; y(0) = 1$; find $y(0.1), y(0.2), y(0.3)$
- e) $y' = 1 + 2xy; y(0) = 0$; find $y(0.1), y(0.2), y(0.3)$

Solve the following by Euler's Method.

10. a) $y' = 1 + xy; y(0) = 2$; find $y(0.1), y(0.2), y(0.3)$
- b) $y' = x + y; y(0) = 1$; find $y(0.2), y(0.4), \dots, y(1)$
- c) $y' = -y; y(0) = 1$; find $y(0.01)$ to $y(0.04); h = 0.01$
- d) $y' = 3x^2 + 1; y(0) = 2$; find $y(2)$ when $h = 0.5$ & 0.25
- e) $y' = \log(x + y); y(1) = 2$; find $y(1.2), y(1.4)$
- f) $y' = x + y; y(0) = 1$; find upto $y(0.1); h = 0.05$

Solve the following by Heun's Method.

11. a) $y' = 1 + xy; y(0) = 2$; find $y(0.1), y(0.2), y(0.3)$
- b) $y' = x + y; y(0) = 1$; find $y(0.2), y(0.4), \dots, y(1)$
- c) $y' = -y; y(0) = 1$; find $y(0.01)$ to $y(0.04); h = 0.01$
- d) $y' = 3x^2 + 1; y(0) = 2$; find $y(2)$ when $h = 0.5$ & 0.25
- e) $y' = \log(x + y); y(1) = 2$; find $y(1.2), y(1.4)$
- f) $y' = x + y; y(0) = 1$; find upto $y(0.1); h = 0.05$

Solve the following by Runge Kutta Method of fourth order.

- a) $y' = 1 + xy; y(0) = 2$; find $y(0.1), y(0.2), y(0.3)$
- b) $y' = x + y; y(0) = 1$; find $y(0.2), y(0.4), \dots, y(1)$
- c) $y' = -y; y(0) = 1$; find $y(0.01)$ to $y(0.04); h = 0.01$
- d) $y' = 3x^2 + 1; y(0) = 2$; find $y(2)$ when $h = 0.5$ & 0.25
- e) $y' = \log(x + y); y(1) = 2$; find $y(1.2), y(1.4)$
- f) $y' = x + y; y(0) = 1$; find upto $y(0.1); h = 0.05$

Evaluate the following by using the boundary condition

- a) $y'' = e^{x^2}$ with $y(0) = 0, y(1) = 0$. Evaluate y at $x = 0.25, 0.5, 0.75$.
- b) $y'' = x + y$ with $y(0) = 0, y(1) = 0$. Evaluate y at $x = 0.25, 0.5, 0.75$.
- c) $y'' = e^{x^2}$ with $y(0) = 0, y(1) = 0$. Evaluate y at $x = 0.25, 0.5, 0.75$.
- d) $y'' + y = 0$ with $y(0) = 1, y(1) = 1$. Evaluate y_1, y_2, y_3 when $h = 0.25$.
- e) $y'' = e^{x^2}$ with $y(0) = 0, y(1) = 0$. Evaluate y at $x = 0.25, 0.5, 0.75$.
- f) $y'' = x + y$ with $y(0) = 0, y(1) = 1$. Evaluate y at $x = 0.25, 0.5, 0.75$.
- g) $y'' = e^{x^2}$ with $y(0) = 1, y(1) = 1$. Evaluate y at $x = 0.25, 0.5, 0.75$.

Solve the following 2nd order differential equation by fourth order using Heun's method.

$$L \frac{d^2q}{dt^2} + R \frac{dq}{dt} + \frac{q}{C} = 5 \quad \text{Where } L=1, C=0.25, R=0.5 \text{ for } t=0 \text{ to } 0.2 \text{ with a step of } 0.1, \text{ given } q(0)=0 \text{ and } q'(0)=0$$

15. Using Runge-Kutta (RK-4) method solve $y'' = x y' + y^2$, $y(0) = 1$, $y'(0) = 2$, use step size 0.1 to find $y(0.2)$.
16. Solve the following system of simultaneous equation with associated initial conditions using fourth order Runge-Kutta (RK-4) method.
- $$\begin{aligned}y' &= y - z & y(0) &= 0 \\z' &= -y + z & z(0) &= 0\end{aligned}$$
- in the range $0 \leq x \leq 1$ with step size of 0.5.
17. Solve $y'' - x^2 y' - 2xy = 0$; $y(0) = 1$, $y'(0) = 0$ for $y(0.1)$ using Runge Kutta (RK4) method.



Chapter 9

SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Chapter Content

- What is a Partial Differential Equation (PDE)
- Deriving Difference Equation
- Solving Elliptic PDEs
- Solving Poisson's Equation

9.1 PARTIAL DIFFERENTIAL EQUATION (PDE)

A differential equation with one independent variable is called an ordinary differential equation. An example of such an equation would be

$$3 \frac{dy}{dx} + 5y^2 = 3e^{-x}, y(0) = 5$$

Where, y is the dependent variable, and x is the independent variable.

If there is more than one independent variable, then the differential equation is called a partial differential equation. An example of such an equation would be

$$3 \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = x^2 + y^2$$

where u is the dependent variable, and x and y are the independent variables.

9.2 CLASSIFICATION OF PDE'S

As an introduction to solve PDEs, most textbooks concentrate on linear second order PDEs with two independent variables and one dependent variable. The general form of such an equation is

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D = 0$$

Where A , B , and C are functions of x and y and D is a function of x , y , u and $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$.

Depending on the value of $B^2 - 4AC$, a Second order linear PDE can be classified into three categories.

1. if $B^2 - 4AC < 0$, it is called elliptic
2. if $B^2 - 4AC = 0$, it is called parabolic
3. if $B^2 - 4AC > 0$, it is called hyperbolic

9.1.1 Elliptic Equation

The Laplace equation for steady state temperature in a plate is an example of an elliptic second order linear partial differential equation. The Laplace equation for steady state temperature in a plate is given by

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \quad (1)$$

Using the general form of second order linear PDEs with one dependent variable and two independent variables,

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D = 0$$

$$A = 1, B = 0, C = 1, D = 0,$$

$$\text{gives } B^2 - 4AC = 0 - 4(1)(1) = -4 \\ = -4 < 0$$

This classifies Equation (1) as elliptic.

9.1.2 Parabolic Equation

The heat conduction equation is an example of a parabolic second order linear partial differential equation. The heat conduction equation is given by

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

Using the general form of second order linear PDEs with one dependent variable and two independent variables,

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D = 0$$

$$A = k, B = 0, C = 0, D = -1,$$

$$\text{gives } B^2 - 4AC = 0 - 4(0)(k) \\ = 0$$

This classifies Equation (2) as parabolic.

9.1.3 Hyperbolic Equation

The wave equation is an example of a hyperbolic second order linear partial differential equation. The wave equation is given by

$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 y}{\partial t^2} \quad (3)$$

Using the general form of second order linear PDEs with one dependent variable and two independent variables,

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D = 0$$

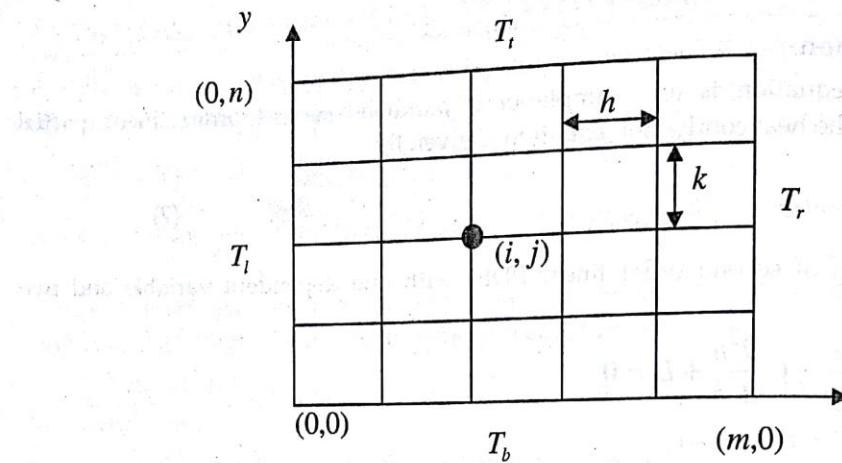
$$A = 1, B = 0, C = -\frac{1}{c^2}, D = 0$$

$$\text{gives } B^2 - 4AC = 0 - 4(1)\left(\frac{-1}{c^2}\right) = \frac{4}{c^2} > 0$$

This classifies Equation (3) as hyperbolic.

9.2 DERIVING DIFFERENCE EQUATION

Consider a two dimensional solution domain as shown in the figure below. The domain is divided into rectangular grids of width h and height k . The value at the intersection of grid points are functions of two variables x and y and that represented by $f(x, y)$.

Figure 9.1: Rectangular Grid with $m \times n$ grids

We know that if the function $f(x)$ has continuous first derivative then its first and second derivative is given by

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

and

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

When f is a function of two variables x and y , we can use the function in x -direction and y -direction to determine partial derivatives with respect to x and y as below:

$$\frac{\partial f(x_i, y_j)}{\partial x} = f_x(x_i, y_j) = \frac{f(x_{i+1}, y_j) - f(x_{i-1}, y_j)}{2h}$$

$$\frac{\partial f(x_i, y_j)}{\partial y} = f_y(x_i, y_j) = \frac{f(x_i, y_{j+1}) - f(x_i, y_{j-1})}{2h}$$

$$\frac{\partial^2 f(x_i, y_j)}{\partial x^2} = f_{xx}(x_i, y_j) = \frac{f(x_{i+1}, y_j) - 2f(x_i, y_j) + f(x_{i-1}, y_j)}{h^2}$$

$$\frac{\partial^2 f(x_i, y_j)}{\partial y^2} = f_{yy}(x_i, y_j) = \frac{f(x_i, y_{j+1}) - 2f(x_i, y_j) + f(x_i, y_{j-1})}{k^2}$$

$$\frac{\partial^2 f(x_i, y_j)}{\partial x \partial y} = \frac{f(x_{i+1}, y_{j+1}) - f(x_{i+1}, y_{j-1}) - f(x_{i-1}, y_{j+1}) + f(x_{i-1}, y_{j-1})}{4hk}$$

These finite difference equivalents of the partial derivatives can be used to construct various types of differential equations

9.3 SOLVING LAPLACE'S EQUATION

We know that Laplace equation is given by

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \text{ or } \nabla^2 u = 0 \quad (1)$$

To solve the Laplace equation take a rectangular plate as shown in figure below where each side of the plate is maintained at a specific temperature. We are interested in finding the temperature within the plate at steady state. No heat sinks or sources exist in the problem. To find the temperature within the plate, we divide the plate area by a grid as shown in Figure below. The length L along the x -axis is divided into m equal segments, while the width W along the y -axis is divided into n equal segments, hence giving

$$h = \frac{L}{m} \quad k = \frac{W}{n}$$

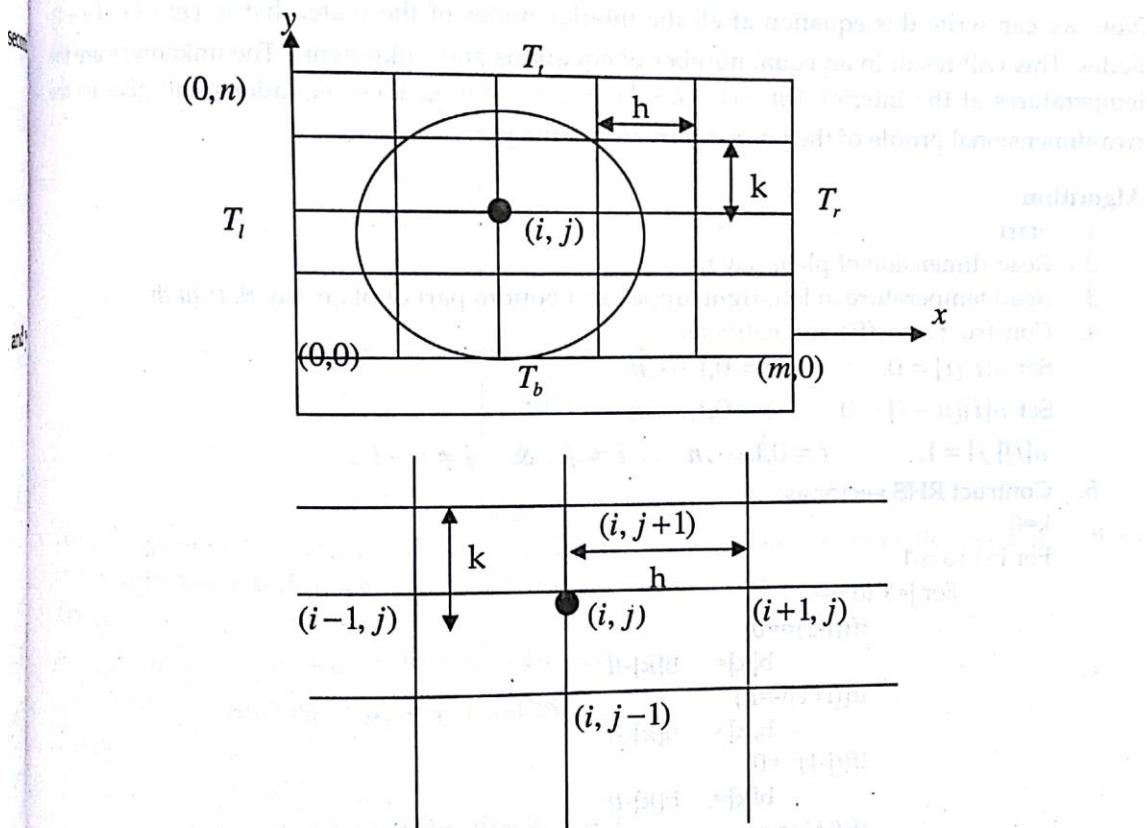


Figure 9.2: Plate area divided into a grid

Now we will apply the finite difference approximation of the partial derivatives at a general interior node (i, j) .

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} \quad (2)$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{k^2} \quad (3)$$

Equations (2) and (3) are central divided difference approximations of the second derivatives. Substituting Equations (2) and (3) in Equation (1), we get

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{k^2} = 0 \quad (4)$$

For a grid with $h = k$, Equation (4) can be simplified as

$$\begin{aligned} & \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{h^2} = 0 \\ & T_{i+1,j} - 2T_{i,j} + T_{i-1,j} + T_{i,j+1} - 2T_{i,j} + T_{i,j-1} = 0 \\ & T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0 \end{aligned}$$

Now we can write this equation at all the interior nodes of the plate, that is $(m-1) \times (n-1)$ nodes. This will result in an equal number of equations and unknowns. The unknowns are the temperatures at the interior $(m-1) \times (n-1)$ nodes. Solving these equations will give us the two-dimensional profile of the temperature inside the plate.

Algorithm

1. Start
2. Read dimension of plate, say n
3. Read temperature in left, right, upper and bottom part of plate, say tl, tr, tu, tb
4. Construct a coefficient matrix as:
Set $a[i][i] = 0, \quad i = 0, 1, \dots, n$
Set $a[i][n-i] = 0, \quad i = 0, 1, \dots, n$
 $a[i][j] = 1, \quad i = 0, 1, \dots, n \quad i \neq j \quad \& \quad j \neq n-i$
5. Construct RHS vector as:
 $k=0$
For $i=1$ to $n-1$
 For $j=1$ to $n-1$
 if $((i-1)==0)$
 $b[k] = b[k] - tl$
 if $((i+1)==n)$
 $b[k] = b[k] - tr$
 if $((j-1)==0)$
 $b[k] = b[k] - tb$
 if $((j+1)==n)$
 $b[k] = b[k] - tu$
 $k++$
 End for
End for
6. Use Gauss-Seidal / Jacobi iteration method to solve the equation
7. Display the solution vector

8. Terminate

Example

A plate of dimension $2.4m \times 2.4m$ is subjected to temperatures as follows: left side at $75^\circ C$, right side at $100^\circ C$, Upper part at $300^\circ C$, and lower part at $50^\circ C$. If square grid length of $0.8m \times 0.8m$ is assumed, what will be temperature at the interior nodes.

Solution

Since height and width of grid is same

$$\Rightarrow h=k=0.8 \text{ m}$$

Thus we can divide rectangular plate into $m \times n$ grids where,

$$m = \frac{L}{h} = 3 \quad n = \frac{W}{k} = 3$$

The nodes are shown in Figure below.

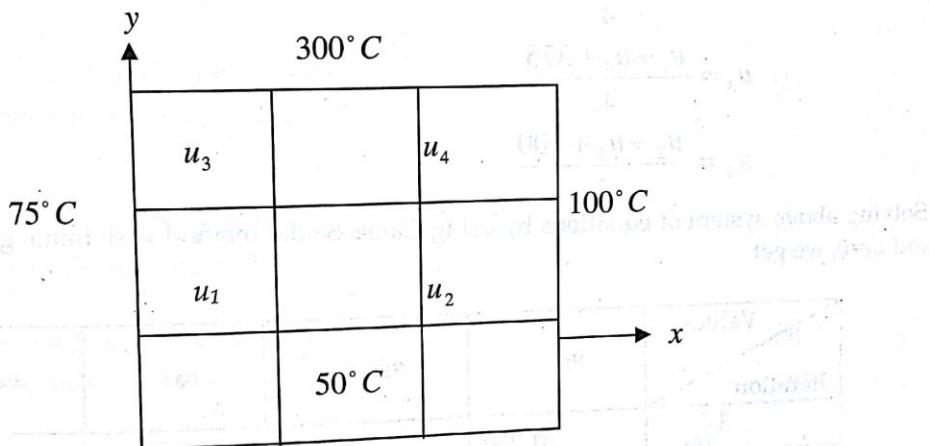


Figure 9.3: Plate with nodes

Now to get the temperature at the interior nodes we have to write Equation for all the combinations of i and j , $i = 1, \dots, m-1; j = 1, \dots, n-1$

For u_1

$$u_3 + u_2 + 75 + 50 - 4u_1 = 0 \quad (1)$$

$$\Rightarrow -4u_1 + u_2 + u_3 = -125$$

For u_2

$$u_4 + u_1 + 100 + 50 - 4u_2 = 0 \quad (2)$$

$$\Rightarrow u_1 - 4u_2 + u_4 = -150$$

For u_3

$$u_4 + u_1 + 300 + 75 - 4u_3 = 0 \quad (3)$$

$$\Rightarrow u_1 + u_4 - 4u_3 = -375$$

For u_4

$$\begin{aligned} u_2 + u_3 + 300 + 100 - 4u_4 &= 0 \\ \Rightarrow u_2 + u_3 - 4u_4 &= -400 \end{aligned} \quad (4)$$

Equations (1) to (4) represent a set of four simultaneous linear equations, which is given below:

$$-4u_1 + u_2 + u_3 = -125$$

$$u_1 - 4u_2 + u_4 = -150$$

$$u_1 + u_4 - 4u_3 = -375$$

$$u_2 + u_3 - 4u_4 = -400$$

\Rightarrow

$$u_1 = \frac{u_2 + u_3 + 125}{4}$$

$$u_2 = \frac{u_1 + u_4 + 150}{4}$$

$$u_3 = \frac{u_1 + u_4 + 375}{4}$$

$$u_4 = \frac{u_2 + u_3 + 400}{4}$$

Solving above system of equations by using Gauss-Seidal method with initial guess $u_2=0, u_3=0$, and $u_4=0$, we get

Iteration \ Values	u_1	u_2	u_3	u_4
1	31.250	45.313	101.563	136.719
2	67.969	88.672	144.922	158.398
3	89.648	99.512	155.762	163.818
4	95.068	102.222	158.472	165.173
5	96.423	102.899	159.149	165.512
6	96.762	103.069	159.319	165.597
7	96.847	103.111	159.361	165.618
8	96.868	103.121	159.371	165.623
9	96.873	103.124	159.374	165.625

Thus,

$$u_1 = 96.873$$

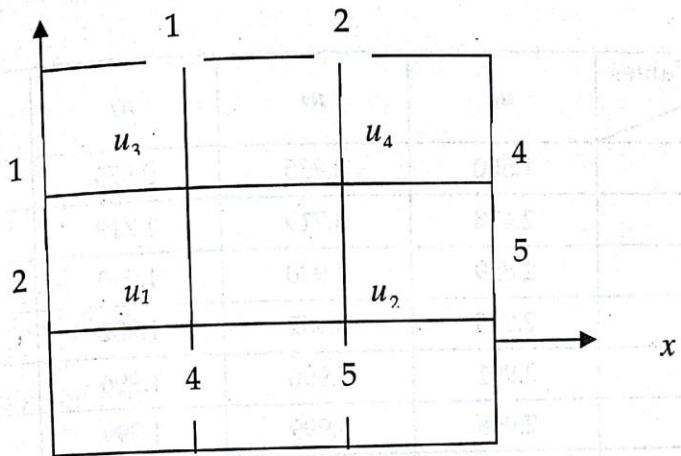
$$u_2 = 103.124$$

$$u_3 = 159.374$$

$$u_4 = 165.625$$

Second Example

Solve the Laplace's Equation for square region shown below. Boundary values are as given in the figure.



Solution

For u_1

$$\begin{aligned} u_3 + u_2 + 2 + 4 - 4u_1 &= 0 \\ \Rightarrow -4u_1 + u_2 + u_3 &= -6 \end{aligned} \quad (1)$$

For u_2

$$\begin{aligned} u_4 + u_1 + 5 + 5 - 4u_2 &= 0 \\ \Rightarrow u_1 - 4u_2 + u_4 &= -10 \end{aligned} \quad (2)$$

For u_3

$$\begin{aligned} u_4 + u_1 + 1 + 1 - 4u_3 &= 0 \\ \Rightarrow u_1 + u_4 - 4u_3 &= -2 \end{aligned} \quad (3)$$

For u_4

$$\begin{aligned} u_2 + u_3 + 2 + 4 - 4u_4 &= 0 \\ \Rightarrow u_2 + u_3 - 4u_4 &= -6 \end{aligned} \quad (4)$$

Equations (1) to (4) represent a set of four simultaneous linear equations, which is given below:

$$\begin{aligned} -4u_1 + u_2 + u_3 &= -6 \\ u_1 - 4u_2 + u_4 &= -10 \\ u_1 + u_4 - 4u_3 &= -2 \\ u_2 + u_3 - 4u_4 &= -6 \end{aligned}$$

\Rightarrow

$$\begin{aligned} u_1 &= \frac{u_2 + u_3 + 6}{4} \\ u_2 &= \frac{u_1 + u_4 + 10}{4} \\ u_3 &= \frac{u_1 + u_4 + 2}{4} \\ u_4 &= \frac{u_2 + u_3 + 6}{4} \end{aligned}$$

Solving above system of equations by using Gauss-Seidal method with initial guess $u_1=0, u_2=0, u_3=0$, and $u_4=0$, we get

Iteration \ Values	u_1	u_2	u_3	u_4
1	1.500	2.875	0.875	2.438
2	2.438	3.719	1.719	2.859
3	2.859	3.930	1.930	2.965
4	2.965	3.982	1.982	2.991
5	2.991	3.996	1.996	2.998
6	2.998	3.999	1.999	2.999
7	2.999	4.000	2.000	3.000
8	3.000	4.000	2.000	3.000

Thus,

$$u_1 = 3 \quad u_2 = 4 \quad u_3 = 2 \quad u_4 = 3$$

//C program for solving elliptic PDE's by using finite difference method

```

//Assumption: T11=x1      T12=x2          T21=x3          T22=x4
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int n,i,j,k;
    float sum,error, E[10],a[10][10],b[10],new_x[10],old_x[10],tl,tr,tu,tb;
    printf("Enter Dimension of plate\n");
    scanf("%d",&n);
    printf("Enter teperatures at left, right, bottom & upper part of plate\n");
    scanf("%f%f%f%f",&tl,&tr,&tb,&tu);

    // Construction of Cofficient matrix
    for(i=0;i<=n;i++)
        a[i][i]=-4;
    for(i=0;i<=n;i++)
        a[i][n-i]=0;
    for(i=0;i<=n;i++)
        for(j=0;j<=n;j++)
    {
        if(i!=j && j!=(n-i))
            a[i][j]=1;
    }
}

```

```

//Construction of RHS vector
for(i=0;i<=n;i++)
    b[i]=0;
k=0;
for(i=1;i<n;i++)
{
    for(j=1;j<n;j++)
    {
        if((i-1)==0)
            b[k]=b[k]-tl;
        if((i+1)==n)
            b[k]=b[k]-tr;
        if((j-1)==0)
            b[k]=b[k]-tb;
        if((j+1)==n)
            b[k]=b[k]-tu;
        k++;
    }
}
printf("Enter Accuracy Limit\n");
scanf("%f",&error);

//Solving system of linear equations by using Gauss-Seidal method
for(i=0;i<=n;i++)
{
    new_x[i]=0;
}
while (1)
{
    for(i=0;i<=n;i++)
    {
        sum=b[i];
        for(j=0;j<=n;j++)
        {
            if(i!=j)
                sum=sum-a[i][j]*new_x[j];
        }
        old_x[i]=new_x[i];
        new_x[i]=sum/a[i][i];
        E[i]=fabs(new_x[i]-old_x[i])/fabs(new_x[i]);
    }
    for(i=0;i<=n;i++)
    {
        if(E[i]>error)
            break;
    }
}

```

```

    }
    if(i==(n+1))
        break;
    else
        continue;
}

printf("Solution:\n");
for(i=0;i<=n;i++)
printf("x%d=%6.2f\n",i+1,new_x[i]);
getch();
return 0;
}

```

Output

Enter Dimension of plate
3
Enter temperatures at left, right, bottom & upper part of plate
75 100 50 300
Enter Accuracy Limit: 0.0001
Solution:
x1=96.87
x2=159.37
x3=103.12
x4=165.62

9.4 SOLVING POISON'S EQUATION

We know that general partial differential equation is given by

$$A \frac{\partial^2 f}{\partial x^2} + B \frac{\partial^2 f}{\partial x \partial y} + C \frac{\partial^2 f}{\partial y^2} - D = 0$$

When, $A = 1, B = 0, C = 1$, and $D = g(x, y)$, above equation becomes

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = g(x, y) \quad \text{or} \quad \nabla^2 f = g(x, y) \quad (1)$$

This equation is called Poisson's equation. To solve the equation take a rectangular plate as shown in figure below where each side of the plate is maintained at a specific temperature. We are interested in finding the temperature within the plate at steady state. No heat sinks or sources exist in the problem. To find the temperature within the plate, we divide the plate area by a grid as shown in Figure below. The length L along the x -axis is divided into m equal segments, while the width W along the y -axis is divided into n equal segments, hence giving

$$h = \frac{L}{m} \quad k = \frac{W}{n}$$

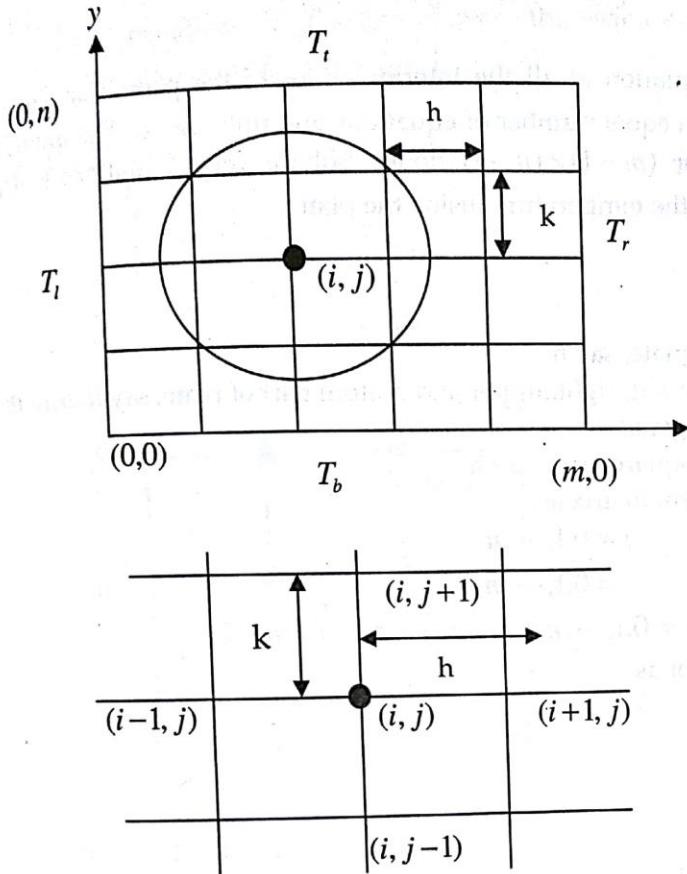


Figure 9.4: Plate area divided into a grid

Now we will apply the finite difference approximation of the partial derivatives at a general interior node (i, j) .

$$\frac{\partial^2 f}{\partial x^2} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2} \quad (2)$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{k^2} \quad (3)$$

Equations (2) and (3) are central divided difference approximations of the second derivatives. Substituting Equations (2) and (3) in Equation (1), we get

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{k^2} = g_{i,j} \quad (4)$$

For a grid with $h = k$, Equation (4) can be simplified as

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{h^2} = g_{i,j}$$

$$T_{i+1,j} - 2T_{i,j} + T_{i-1,j} + T_{i,j+1} - 2T_{i,j} + T_{i,j-1} = h^2 g_{i,j}$$

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = h^2 g_{i,j} \quad (5)$$

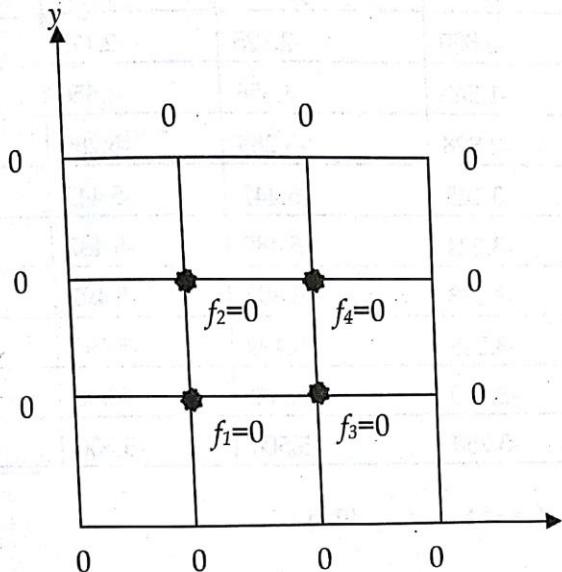
Now we can write this equation at all the interior nodes of the plate, that $(m-1) \times (n-1)$ nodes. This will result in an equal number of equations and unknowns. The unknowns are the temperatures at the interior $(m-1) \times (n-1)$ nodes. Solving these equations will give us the two-dimensional profile of the temperature inside the plate.

Algorithm

1. Start
2. Read dimension of plate, say n
3. Read temperature in left, right, upper and bottom part of plate, say tl, tr, tu, tb
4. Define the function $g(x,y)$
5. Read dimension of square grid, say h
6. Construct a coefficient matrix as:
 Set $a[i][i] = 0, \quad i = 0, 1, \dots, n$
 Set $a[i][n-i] = 0, \quad i = 0, 1, \dots, n$
 $a[i][j] = 1, \quad i = 0, 1, \dots, n \quad i \neq j \quad \& \quad j \neq n-i$
7. Construct RHS vector as:
8. For $i=0$ to $n-1$
 $b[k] = h^2 g(i,j)$
9. End for
10. Set $k=0$
11. For $i=1$ to $n-1$
 For $j=1$ to $n-1$
 $\text{if } ((i-1) == 0)$
 $\quad b[k] = b[k] - tl$
 $\text{if } ((i+1) == n)$
 $\quad b[k] = b[k] - tr$
 $\text{if } ((j-1) == 0)$
 $\quad b[k] = b[k] - tb$
 $\text{if } ((j+1) == n)$
 $\quad b[k] = b[k] - tu$
 $\quad k++$
12. End for
13. Use Gauss Seidal/Jacobi iteration method to solve the equation
14. Display the solution vector
15. Terminate

Example
Solve the Poisson's equation $\nabla^2 f = 2x^2 y^2$ over the square domain $0 \leq x \leq 3$ and $0 \leq y \leq 3$ with $f = 0$ on the boundary and $h = 1$

Solution
Let's divide the domain into grids of 3×3 as below:



For f_1

$$f_2 + f_3 - 4f_1 = 2 \times 1^2 \times 1^2 = 2 \quad (1)$$

For f_2

$$f_1 + f_4 - 4f_2 = 2 \times 2^2 \times 1^2 = 8 \quad (2)$$

For f_3

$$f_1 + f_4 - 4f_3 = 2 \times 1^2 \times 2^2 = 8 \quad (3)$$

For f_4

$$f_2 + f_3 - 4f_4 = 2 \times 2^2 \times 2^2 = 32 \quad (4)$$

Now, we have following system of equations

$$f_2 + f_3 - 4f_1 = 2$$

$$f_1 + f_4 - 4f_2 = 8$$

$$f_1 + f_4 - 4f_3 = 8$$

$$f_2 + f_3 - 4f_4 = 32$$

\Rightarrow

$$f_1 = \frac{f_2 + f_3 - 2}{4}$$

$$f_2 = \frac{f_1 + f_4 - 8}{4}$$

$$f_3 = \frac{f_1 + f_4 - 8}{4}$$

$$f_4 = \frac{f_2 + f_3 - 32}{4}$$

Solving above system of equations by using Gauss-Seidal method, we get

Iteration \ Values	f_1	f_2	f_3	f_4
1	-0.500	-2.125	-2.125	-9.063
2	-1.563	-4.656	-4.656	-10.328
3	-2.828	-5.289	-5.289	-10.645
4	-3.145	-5.447	-5.447	-10.724
5	-3.224	-5.487	-5.487	-10.743
6	-3.243	-5.497	-5.497	-10.748
7	-3.248	-5.499	-5.499	-10.750
8	-3.250	-5.500	-5.500	-10.750
9	-3.250	-5.500	-5.500	-10.750

Thus,

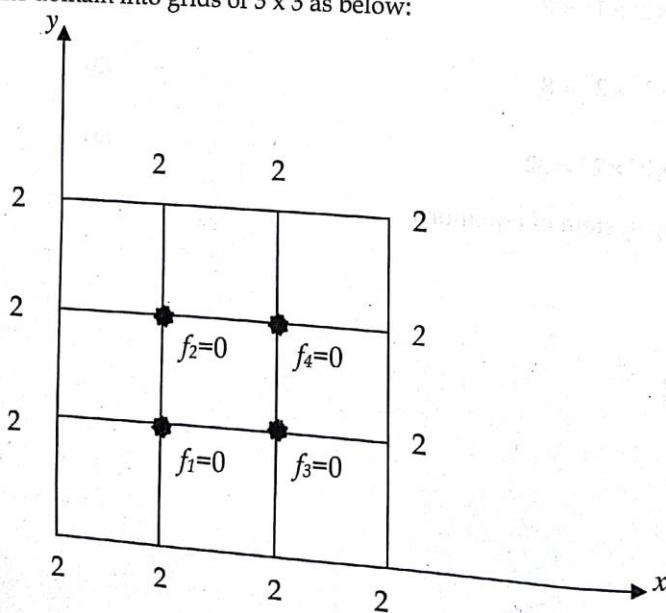
$$f_1 = -3.25 \quad f_2 = -5.5 \quad f_3 = -5.5 \quad f_4 = -10.75$$

Second Example

Solve the Poisson's equation $\nabla^2 f = f(x, y)$ with $f(x, y) = xy$ and $f = 2$ on boundary by assuming square domain $0 \leq x \leq 3$ and $0 \leq y \leq 3$ and $h = 1$

Solution

Lets divide the domain into grids of 3×3 as below:



For f_1 $f_2 + f_3 - 4f_1 + 4 = 1$ (1)

For f_2 $f_1 + f_4 - 4f_2 + 4 = 2$ (2)

For f_3 $f_1 + f_4 - 4f_3 + 4 = 2$ (3)

For f_4 $f_2 + f_3 - 4f_4 + 4 = 4$ (4)

Now, we have following system of equations

$$f_2 + f_3 - 4f_1 + 4 = 1$$

$$f_1 + f_4 - 4f_2 + 4 = 2$$

$$f_1 + f_4 - 4f_3 + 4 = 2$$

$$f_2 + f_3 - 4f_4 + 4 = 4$$

=>

$$f_1 = \frac{f_2 + f_3 + 3}{4}$$

$$f_2 = \frac{f_1 + f_4 + 2}{4}$$

$$f_3 = \frac{f_1 + f_4 + 2}{4}$$

$$f_4 = \frac{f_2 + f_3}{4}$$

Solving above system of equations by using Gauss-Seidal method, we get

Iteration \ Values	f_1	f_2	f_3	f_4
1	0.750	0.688	0.688	0.344
2	1.094	0.859	0.859	0.430
3	1.180	0.902	0.902	0.451
4	1.201	0.913	0.913	0.457
5	1.207	0.916	0.916	0.458
6	1.208	0.916	0.916	0.458
7	1.208	0.917	0.917	0.458

Thus,

$$f_1 = 1.208 \quad f_2 = 0.917 \quad f_3 = 0.917 \quad f_4 = 0.458$$

//C program for solving Poisson's equation by using finite difference method

//Assumption: $f_{11}=x_1 \quad f_{12}=x_2 \quad f_{21}=x_3 \quad f_{22}=x_4$

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define g(x,y) 2*(x)*(x)*(y)*(y)

```

int main()
{
    int n,i,j,k;
    float sum,error, E[10],a[10][10],b[10],new_x[10],old_x[10],tl,tr,tu,tb,h;
    printf("Enter Dimension of plate\n");
    scanf("%d",&n);
    printf("Enter Dimension of grid\n");
    scanf("%f",&h);
    printf("Enter teperatures at left, right, bottom & upper part of plate\n");
    scanf("%f%f%f%f",&tl,&tr,&tb,&tu);

    //Constructing Cofficient matrix
    for(i=0;i<=n;i++)
        a[i][i]=-4;
    for(i=0;i<=n;i++)
        a[i][n-i]=0;
    for(i=0;i<=n;i++)
        for(j=0;j<=n;j++)
    {
        if(i!=j && j!=(n-i))
            a[i][j]=1;
    }

    //Constructing RHS vector
    k=0;
    for(i=1;i<n;i++)
        for(j=1;j<n;j++)
            b[k++]=h*h*g(i,j);
    k=0;
    for(i=1;i<n;i++)
    {
        for(j=1;j<n;j++)
        {
            if((i-1)==0)
                b[k]=b[k]-tl;
            if((i+1)==n)
                b[k]=b[k]-tr;
            if((j-1)==0)
                b[k]=b[k]-tb;
            if((j+1)==n)
                b[k]=b[k]-tu;
            k++;
        }
    }
    printf("Enter Accuracy Limit\n");
    scanf("%f",&error);
}

```

//Solving System of linear equations by using Gauss-Seidal Method

```

for(i=0;i<=n;i++)
{
    new_x[i]=0;
}
while (1)
{
    for(i=0;i<=n;i++)
    {
        sum=b[i];
        for(j=0;j<=n;j++)
        {
            if(i!=j)
                sum=sum-a[i][j]*new_x[j];
        }
        old_x[i]=new_x[i];
        new_x[i]=sum/a[i][i];
        E[i]=fabs(new_x[i]-old_x[i])/fabs(new_x[i]);
    }
    for(i=0;i<=n;i++)
    {
        if(E[i]>error)
            break;
    }
    if(i==(n+1))
        break;
    else
        continue;
}
printf("Solution:\n");
for(i=0;i<=n;i++)
printf("%d=%6.2f\n",i+1,new_x[i]);
getch();
return 0;
}

```

Output

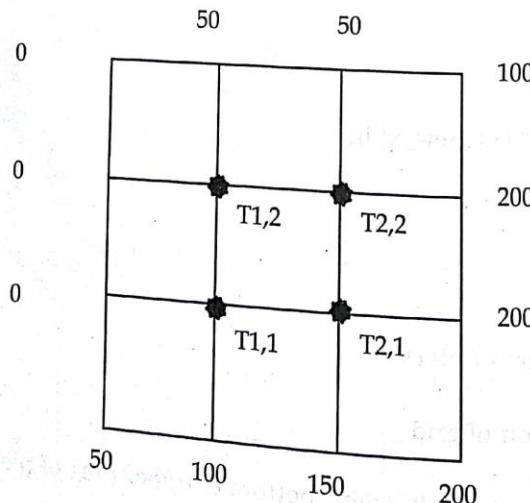
Enter Dimension of plate
3
Enter Dimension of grid
1
Enter teperatures at left, right, bottom & upper part of plate
0 0 0 0
Enter Accuracy Limit
0.001
Solution:
x1= -3.25
x2= -5.50

$$\begin{aligned}x_3 &= -5.50 \\x_4 &= -10.75\end{aligned}$$

EXERCISE

- Write down the general form of linear second order PDE and categorize them along with suitable example.
- Draw flowchart for solving elliptic partial differential equations by using divided difference method.
- State two real life problems, where partial differential equations where partial differential equations are required to construct mathematical models.
- Differentiate Laplace's equation from Poisson's equation. Provide any two examples where Laplace's or Poisson's equation is applicable.
- Determine which of the following equations are elliptic, parabolic, and hyperbolic.
 - $3f_{xx} + 4f_{yy} = 0$
 - $f_{xx} + 2f_{xy} + 2f_{yy} = 2x + 3y$
 - $2f_{xx} + 5f_{xy} + f_{yy} = 0$
 - $f_{xx} + 2f_{yy} = 0$
- Suppose that steady-state two dimensional heat flows in a metal is given by:

$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$. If temperature s of plate is as given in figure below, find temperatures at interior nodes of the plate.



- Consider a rectangular plate of $15 \text{ cm} \times 18 \text{ cm}$ in which each side is held at 50°C . What will be temperature at interior nodes, if steady-state two dimensional heat flows in a metal is given by: $\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = xy$. Assume square grids of size $3 \text{ cm} \times 3 \text{ cm}$.

Approximate the solution to the following elliptic partial differential equation and compare the results to the actual solution $u(x, y) = (x - y)^2$. Assume $h=1/2$.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad 0 < x < 1, 0 < y < 2$$

$$u(x, 0) = x^2, u(x, 2) = (x - 2)^2, 0 \leq x \leq 1, u(0, y) = y^2, u(1, y) = (y - 1)^2, 0 \leq y \leq 2.$$

9. Determine the steady-state heat distribution in a thin square metal plate with dimensions $0.5 \text{ m by } 0.5 \text{ m}$ using $n = m = 4$. Two adjacent boundaries are held at 0°C , and the heat on the other boundaries increases linearly from 0°C at one corner to 100°C where the sides meet.

10. Use the Poisson finite-difference method with $n = 6, m = 5$, and a tolerance of 0^{-10} to

approximate the solution to $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = xe^y, 0 < x < 2, 0 < y < 1$, with the boundary conditions $u(0, y) = 0, u(2, y) = 2ey, 0 \leq y \leq 1, u(x, 0) = x, u(x, 1) = ex, 0 \leq x \leq 2$, and compare the results with the exact solution $u(x, y) = xe^y$.

Bibliography

- C.F. Gerald and P.O. Wheatley, "Applied Numerical Analysis", 9th Edition, Addison Wesley Publishing Company, New York, 2011.
- E. Balagurusamy, "Numerical Methods", Tata McGraw-Hill Publishing Company Ltd, New Delhi, 1999.
- F.B. Hilderbrand, **Introduction to Numerical Analysis, Second Revised Edition, 1987**
- J. M. Mathews and K. Fink, "Numerical Methods using MATLAB ", 4th Edition, Prentice Hall Publication, 2004.
- K. A. Stroud, D. J. Booth, Engineering Mathematics, Industrial Press, Inc., Fifth Edition, 2001.
- R. W. Hamming, **Numerical Methods for Scientists and Engineers, Dover Publications, Second Revised Edition, 1987**
- S. S. Shatri, Introductory Methods of Numerical Analysis, Prentice Hall India Learning Private Limited, Fifth Edition, 2012
- S. Yakwitz and F. Szidarovszky, "An Introduction to Numerical Computations", 2nd Edition, Macmillan Publishing Co., New York.
- W. Cheney and D. Kincaid, "Numerical Mathematics and Computing", 7th Edition, Brooks/Cole Publishing Co, 2012
- W.H. Press, B.P. Flannery et al., "Numerical Recipes: Art of Scientific Computing", 3rd Edition, Cambridge Press, 2007.

KEC's B.Sc. CSIT Text Book Series

First Semester

- Introduction to Information Technology
- C Programming
- Digital Logic
- Mathematics I
- Physics

Second Semester

- Discrete Structure
- Object Oriented Programming
- Microprocessor
- Mathematics II
- Statistics I

Third Semester

- Data Structure and Algorithms
- Numerical Method
- Computer Architecture
- Computer Graphics
- Statistics II

Fourth Semester

- Theory of Computation
- Computer Networks
- Operating System
- Database Management System
- Artificial Intelligence

Fifth Semester

- Design and Analysis of Algorithms
- System Analysis and Design
- Cryptography
- Simulation and Modeling
- Web Technology
- Elective I

Sixth Semester

- Software Engineering
- Compiler Design and Construction
- E-Governance
- NET Centric Computing
- Technical Writing
- Elective II

Seventh Semester

- Advanced Java Programming
- Data Warehousing and Data Mining
- Principles of Management
- Project Work
- Elective III

Eighth Semester

- Advanced Database
- Internship
- Elective IV
- Elective V



KEC

Publication and Distribution Pvt. Ltd.

Kathmandu, Nepal, Tel.: 01-4168301, 01-4241777
E-mail: kecpublication14@gmail.com

