

OBJECT-ORIENTED PROGRAMMING OF NUMERICAL METHODS WITH GUI

A Thesis

**SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE AWARD OF THE DEGREE OF**

**MASTER OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING**

TO

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA - 147 001**



SUPERVISOR

Shri. Himanshu Aggarwal

SUBMITTED BY

A. SIVAPRASAD

REG. NO. ME-120/98(R)/1



**DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL TEACHERS' TRAINING INSTITUTE
SECTOR 26, CHANDIGARH-160019**

February, 2000

OBJECT-ORIENTED PROGRAMMING OF NUMERICAL METHODS WITH GUI

A Thesis

**SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE AWARD OF THE DEGREE OF**

**MASTER OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING**

TO

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA - 147 001**



SUPERVISOR

Shri. Himanshu Aggarwal

SUBMITTED BY

A. SIVAPRASAD

REG. NO. ME-120/98(R)/1



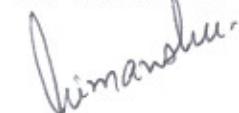
**DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL TEACHERS' TRAINING INSTITUTE
SECTOR 26, CHANDIGARH-160019
February, 2000**

CERTIFICATE

Certified that this Thesis report entitled "**OBJECT-ORIENTED PROGRAMMING OF NUMERICAL METHODHS WITH GUI**" submitted by **A. Sivaprasad** in the partial fulfillment of the requirement, for the award of Master of Engineering (Computer Science & Engg.) Degree of Thapar Institute of Engineering and Technology, Patiala is a record of student's own study carried under my supervision and guidance.

This thesis has not been submitted to any other university or institution for the award of any degree.

SUPERVISOR



(Himanshu Aggarwal)
Lecturer in Computer Science
T.T.T.I.
Chandigarh 160 019



Head
Dept. of Computer Science
T.T.T.I.
Chandigarh 160 019

ACKNOWLEDGEMENTS

I express my sincere gratitude to **Sri. Himanshu Aggarwal**, Lecturer, Department of Computer Science, T.T.T.I., Chandigarh, for his valuable guidance and timely help.

I express my sincere thanks to **Dr. Renu Vig**, Asst. Professor, Department of Computer Science and Engineering, T.T.T.I., Chandigarh, for her valuable suggestions and help during the this thesis work.

I am grateful to Prof. **Vijay P. Puri**, Head of Department of Computer Science, T.T.T.I., Chandigarh, for providing me facilities in the department.

I am thankful to all the faculty and staff members of the Computer Science Department for their help.



(A. SIVAPRASAD)

ABSTRACT

Computer oriented numerical methods are widely used to solve many real life engineering and scientific problems. The prime objective of this thesis work is to prepare a software package to solve numerical problems by using object-oriented approach. The design (modeling) of this package is done using Unified Modeling Language (UML). The implementation of this UML design is done using object-oriented languages C++ and Java. The software is developed using VC++6 and Java2 (independently in both languages). The package consists of three layers. The first is a GUI menu from which the user can interact with the software. The menu program activates GUI front-end program (second layer) as per user's choice. One GUI front-end program is developed for each type of problem. The user can get the solution of numerical problems by entering data in the GUI screen. To get the solution of the problem the user need not know the procedure to solve the problem. The GUI screen is user friendly and user need not have any programming knowledge to use this software. The third layer in this software numerical library or package that consists of classes for different types of problems. The GUI program calls operations of these classes after entry of required data (by the user) on the screen, gets solution of the problem, and displays result. The programmers can make use of classes in the library or package in their own programs. In VC++6 the library is implemented as static linked library, where as in Java2 the classes are put together in a package named numerical. VC++ programmers can make use of these classes using include statement for the corresponding header file, where as Java2 programmers can do it using import statement.

CONTENTS

1. Introduction	1
1.1 Numerical methods	1
1.2 Problem definition	2
1.3 Object oriented programming	3
1.3.1 Benefits of object-oriented programming	4
2. Numerical Methods	6
2.1 Method of problem solving and Errors	6
2.1.1 Input errors and data modeling	7
2.1.2 Algorithmic errors	7
2.1.3 Computational errors	8
2.2 Operation on Matrices	9
2.3 Linear Algebraic Equations	10
2.3.1 Gauss-Jordan method	11
2.3.2 Gauss-Elimination method	11
2.3.3 Jacobi iteration method	12
2.3.4 Gauss-Seidal iteration method	13
2.3.5 Choleski's method	13
2.3.6 Comparison of various methods	13
2.4 Solution of Algebraic & transcendental equations	14
2.4.1 Bisection method	15
2.4.2 Newton Raphson method	15
2.4.3 Secant method	16
2.4.4 False position method	16
2.5 Interpolation	17
2.5.1 Interpolation with equal intervals	17
2.5.2 Interpolation with unequal intervals	19
2.6 Numerical differentiation	20
2.6.1 Newton's forward difference formula	21
2.6.2 Newton's backward difference formula	22
2.6.3 Stirling's formula	22
2.6.4 Bessel's formula	22
2.7 Numerical integration	22
2.7.1 Trapezoidal rule	23
2.7.2 Simpson's one-third rule	23
2.7.3 Simpson's three-eighth rule	23
2.7.4 Weddle's rule	23
2.8 Linear programming	23
3. Algorithms	25
3.1 Solution of simultaneous equations	25
3.2 Solution of Algebraic and transcendental equations	26
3.3 Integration	28
3.4 Linear Programming	29

4. Design (Modeling)	30
4.1 Unified Modeling language (UML)	30
4.2 GRAPPLE	31
4.3 Organization of the components	31
4.3.1 Opening menu	32
4.3.2 Graphical user interface	32
4.3.3 Numerical library or package	32
4.4 Classes	32
4.5 Modeling of GUI programs	39
4.5.1 General principles of GUI design	39
4.5.2 design of matrix GUI window	41
5. Development using VC++6	44
5.1 Introduction to VC++6	44
5.1.1 C++	44
5.1.2 Features of C++ as object-oriented language	44
5.1.3 Microsoft VC++6	46
5.2 Numerical library	47
5.2.1 Library in VC++6	47
5.2.2 Implementation of numerical library	48
5.3 Development of GUI	56
5.3.1 VC++6 for GUI development	56
5.3.2 Controls in MFC	58
5.4 Opening menu	60
6. Developemnt using Java2	67
6.1 Introduction to Java2	67
6.1.1 The Java programming language	67
6.1.1.1 Features of Java2	
6.1.1.2 Byte codes	70
6.1.2 The Java platform	70
6.2 Numerical package	71
6.2.1 Packages in Java2	71
6.2.2 Development of numerical package	72
6.2.3 Differences between C++ and Java 2	77
6.3 Development of GUI programs	77
7. Conclusions & Future scope	81
Bibliography	83

Chapter 1**INTRODUCTION**

1.1 Numerical Methods

Numerical methods play very important role in the solution of many real world problems in fields of Science and Engineering. Keeping in view of its importance, the study of computer oriented numerical methods is introduced as core subject in the curriculum at polytechnic, graduate and post graduate courses in Engineering.

Over the past few years, object-oriented technology has evolved in diverse segments of the computer sciences as a means of managing the complexity inherent in many different kinds of systems. The object-oriented paradigm has been widely used throughout the world for such diverse domains as the administration of banking transactions, the data base systems, the telephony systems, E-commerce tools, and multimedia applications. An object-oriented system supports features like abstraction or encapsulation, inheritance, self-recursion. Hence the implementation of these numerical methods using object-oriented programming technique (OOP) will be quite useful for students as well as teachers.

Computer oriented numerical methods can be effectively explained and demonstrated to the students with Graphical User Interface (GUI). The GUI is user friendly and interesting to use and learn when compared to the DOS based programs. At present there are very few commercial windows based packages, which demonstrates the implementation and working of numerical methods, are available and they are very costly.

By considering the above said factors a software package that has object-oriented design (OOD) and implementation of numerical

methods may be quite useful for diploma and graduate level students. It may also be useful for the post graduate students of engineering, science and applied Mathematics.

1.2 Problem definition

This project work is initiated to provide solutions for the following problems

1. Matrix operations
 - 1.1 Addition, subtraction and multiplication.
 - 1.2 Inverse, transpose and determinant
2. Solution of Linear Equations
 - 2.1 Gauss elimination method
 - 2.2 Gauss-Jordan method
 - 2.3 Jacobi iteration method
 - 2.4 Gauss-Seidal iteration method
 - 2.5 Choleski's method (Triangular factorization method)
3. Solution of Algebraic & Transcendental equations
 - 3.1 Bisection method
 - 3.2 Newton-Raphson method
 - 3.3 Secant method
 - 3.4 False position method or Regula falsi method
4. Interpolation
 - 4.1 Interpolation with equal intervals
 - 4.1.1 Newton Gregory forward interpolation formula
 - 4.1.2 Newton's backward interpolation formula
 - 4.1.3 Stirling's formula
 - 4.1.4 Bessel's formula
 - 4.2 Interpolation with unequal intervals
 - 4.2.1 Lagrange's interpolation formula
 - 4.2.2 Newton's divided difference formula
 - 4.3 Extrapolation
5. Numerical differentiation
 - 5.1 Newton's forward difference formula

- 5.2 Newton's backward difference formula
- 5.3 Stirling's formula
- 5.4 Bessel's formula
- 6. Numerical integration
 - 6.1 Trapezoidal rule
 - 6.2 Simpson's one-third rule
 - 6.3 Simpson's three-eighth rule
 - 6.4 Weddle's rule
- 7. Linear programming
 - 7.1 Simplex method
- 8. Statistical methods

This thesis work consists of preparation classes for all the above said methods and putting them together as a static library in C++, and as a package in Java. The numerical library or package is such that programmers can use the static library or package to develop their own software. The programmers can use classes of this library or package in their programs as header files in case of C++ and import files in case of Java. The solutions are separately developed in both Java and C++ so that the programmers of either language can make use of it.

This thesis work also contains Graphical User Interface for every problem in the package such that a novice user can use the package to solve scientific engineering problems without knowing programming and numerical methods. The GUI is menu driven and user can make use of any method by activating that from the menu.

1.3 Object-Oriented Programming

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

Object types or classes are similar to data types and to entity types with encapsulated methods. Data and methods are encapsulated and hidden by objects. Classes may have concrete instances. The term *object* means either a class or an instance.

Abstraction is the process of identifying relevant objects in the application and ignoring the irrelevant background. Abstraction delivers reusability through encapsulation or information hiding. *Encapsulation* consists in hiding the implementation of objects and declaring publicly the specification of their behavior through a set of attributes and operations.

Objects may communicate only by message passing. Polymorphism or overloading increases readability and programmer productivity but may lead to slower implementations.

Object-based	= encapsulation + object identity
Class-based	= object-based + set abstraction
Object-oriented	= class-based + inheritance + self-recursion

Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic modules of the system under design.

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the class and objects found in the vocabulary of the problem domain.

1.3.1 Benefits of object-oriented programming:

1. Well-designed objects in object-oriented systems are the basis for systems to be assembled largely from reusable modules, leading to higher productivity.

2. Reusing existing classes that have been tested in the field on earlier projects leads to systems which are of higher quality, meet business requirements better and contain fewer bugs.
3. Object-oriented programming, and inheritance in particular, makes it possible to define clearly and use modules that are functionally incomplete and then allow their extension without upsetting the operation of other modules or their clients. This makes systems more flexible, more easily extensible and less costly to maintain.
4. The message-passing convention for communication between objects means that interface descriptions between modules and external systems become much easier. It also facilitates the description and construction of graphical user interfaces and distributed systems.
5. Partitioning systems on the basis of encapsulated object types helps with the problem of scalability. There is no reason why effort should increase exponentially with project size and complexity, as is the case with conventional systems.
6. In the same way the partitioning of work in a project has natural basis and should be easier. Analysis and design by partitioning a domain into objects corresponding to a solution-oriented view of their real-world counterparts is often more natural than a top-down functional decomposition. Object-oriented systems should scale up better from small to large.
7. Information hiding helps to build secure systems.
8. The data-centered design approach of object-oriented analysis, because it subsumes process as well, is able to capture more of the semantics of a model in form that can be implemented.
9. Formal specification methods may be made to blend with object-oriented design.
10. Systems evolution and maintenance problems are mitigated by the strong partitioning, resulting from encapsulation and uniform object interfaces.

2.1 Method of problem solving and Errors

The process of solving physical problems can be roughly divided into three phases. The first consists of constructing a mathematical model for the corresponding physical problem. This model could be in the form of differential equations or algebraic equations. In the most cases, this mathematical model cannot be solved analytically, and hence a numerical solution is required. In which case, the second phase in the solution process usually consists of constructing an appropriate numerical model or approximation to the mathematical model. For example, an integral or differential equation in the mathematical formulation will have to be approximated appropriately for numerical solution. A numerical model is one where everything in principle can be calculated by using a finite number of basic arithmetic operations. The third phase of the solution process is the actual implementation and solution of the numerical model. Even though numerical solution of the mathematical problems cannot substitute analytic techniques. Instead, they compliment analysis and quite often, experimenting with numerical solution, provides a better understanding of the physical problem.

Solving a computational problem using by using a digital computer involves the following important concepts:

- 1) Problem formulation and construction of a mathematical model.
- 2) Construction of an algorithm.
- 3) Programming.
- 4) Computation.

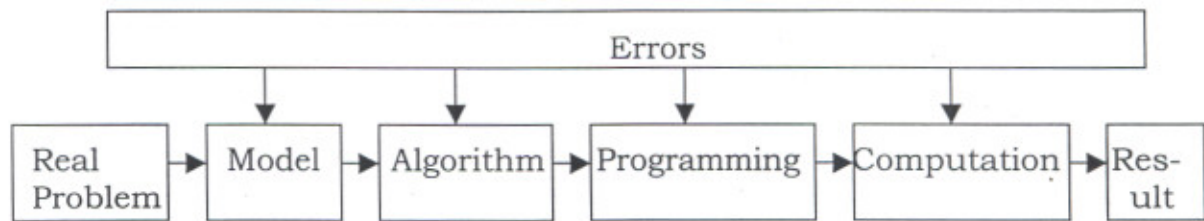


Fig.2.1 Schematic diagram for solving a problem using computer.

Computers do not use the real number system, but instead a simulation of it called a floating-point number system. According to this system the number is expressed as a fraction or an integer and an exponent. This introduces problem of round off error. In addition to this round off error the result may contain following errors.

2.1.1 Input Errors in Data and Modeling

The initial data are often obtained from the experiments, and any experiment can produce results of only limited accuracy in modeling. The quantities used can be represented in the computer for only a limited number of digits.

2.1.2 Algorithmic errors

If direct algorithms, which take a finite sequence of operations, are used, errors due to a limited number of steps do not amplify the existing errors and/or introduce fresh errors; however, if infinite algorithms are used, ideally exact results are obtained only after an infinite number of steps. As this cannot be done practice, the algorithm has to be stopped after a finite number of steps and consequently the results are not exact.

In fact, the situation is often worse than expected. Normally, an infinite algorithm is terminated when it has produced a solution of a desired accuracy. For many actual problems, there are no infallible means of determining just how good a particular estimate is. This implies that there is no surety as to when exactly the infinite algorithm can be terminated. Although methods for testing the confidence limits of the estimates are available, there is always the possibility that the estimate is much worse than expected.

In very complex algorithms it is also possible that there are bugs. It may seem unlikely that such algorithms exist, but they do occur in very complex simulation cases. It is assumed that such bugs are removed.

The following tests can safeguard algorithmic errors:

- 1) Check the results against those obtained independently of the algorithms
- 2) Examine the algorithm in very simple cases.
- 3) Examine the algorithm in extreme cases.
- 4) When there are two or more algorithms compare them.

2.1.3 Computational Errors

Even with simple operations such as multiplication and division, the number of digits can so greatly increase that the results cannot be held fully in the registers available in the computer; in such a case a certain number of digits must be discarded.

The errors accumulate one after another from operation to operation, changing during the process and producing new errors.

There are three useful methods for testing the errors in computations:

- 1) Do the computations with different amounts of precision in the arithmetic.
- 2) Solve several different problems whose initial data are only slightly different.
- 3) Solve the same problem with the arithmetic performed in different possible sequences.

Tests 1 and 2 reveal the stability of the algorithms. Test 3 helps to discover the effect of round-off errors.

2.2 Operations on Matrices

A vector A_n , n -dimensional vector is an n -tuple $(x_1, x_2, \dots, x_n)$ of real or complex numbers. In matrix representation a vector is written either as column vector

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

or as row vector

$$[x_1, x_2, \dots, x_n].$$

A **matrix** is a rectangular array of real or complex numbers written as

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

A matrix can be represented as $A = (a_{ij})$; $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$.

Adding corresponding elements of two matrices can do **addition of matrices** of same size. Similarly subtracting each element of the one

matrix from the corresponding element of other matrix does **subtraction of matrices**.

Multiplication of matrices: Let A be an $m \times n$ and B be an $n \times p$ matrix. The product of two matrices A and B is an $m \times p$ matrix.

$$A = (a_{ij}) ; B = (b_{ij})$$

then the product matrix $C = A \times B = (c_{ij})$

where
$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Transpose of matrix A = (a_{ij}) is obtained by interchanging rows and columns.

$$\text{If } A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \text{ then } A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nm} \end{bmatrix}$$

A matrix B is said to be **inverse** of A if $AB = I = BA$ ($B = A^{-1}$), where I is identity matrix.

A square matrix with 1's in the principal diagonal and 0's in all off-diagonal positions is called an **identity matrix**.

2.3 Linear Algebraic Equations

A set of linear algebraic equations looks like this:

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

Here the n unknowns x_j , $j = 1, 2, \dots, n$ are related by m equations. The coefficients a_{ij} with $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$ are known numbers.

The above system of linear equations can be written matrix form as $A \cdot x$

$$= b, \text{ where } A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & & . \\ . & . & & . \\ a_{m1} & a_{m2} & & a_{mn} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}$$

2.3.1 Gauss-Jordan Method

This is the best general method available for solving a system of linear equations. Let A is arbitrary $m \times n$ matrix and B , an arbitrary $m \times 1$ matrix. The method consists of performing elementary transformations (adding a multiple of a row to some other row, interchanging two rows and multiplying a row by a constant) on the $n \times (n+1)$ matrix $[A, B]$ until A has been reduced to the identity matrix. Once A had been reduced to identity matrix last column of matrix $[A, B]$ is the solution.

Both Gauss-Jordan and Gauss Elimination are direct methods for solution of simultaneous equations.

2.3.2 Gauss Elimination method

The method is similar to the Gauss-Jordan method, it reduces a matrix not all the way to the identity matrix, but only halfway. In this method, only elements of lower triangle are made zero by performing elementary matrix operations. The solution is obtained by the *backsubstitution*.

The advantage of Gauss elimination and *backsubstitution* over Gauss-Jordan elimination is simply that the former is faster in raw operations

count. The innermost loops of Gauss-Jordan method, each containing one subtraction and one multiplication, are executed N^3 and N^3M times, respectively (where there are N equations and M unknowns). The corresponding loops in Gauss elimination are executed only $\frac{1}{3}N^3$ times (only half of the matrix is reduced, and the increasing numbers of predictable zeros reduce the count to one-third), and $\frac{1}{2}N^3M$ times, respectively. Gauss elimination thus has about a factor three advantage over Gauss-Jordan. For computing inverse matrix Gauss-Jordan and Gauss elimination methods have identical efficiencies.

2.3.3 Jacobi iteration Method

This is an iterative method, which calculate the solution in indirect way. If a system has property that each of the equations possesses one large coefficient (numerically) and the larger coefficients in equations correspond to the different unknowns in different equations, and then the iteration method can be applied. Such a system is called a diagonal system as by re-ordering the equations; largest coefficients can be put on the diagonal.

If the coefficients of $x_1, x_2, \dots, x_n$ are the largest coefficients in the first, second, n th equation respectively of the system of linear equations, each equation of the system is solved for the unknown with largest coefficient, in terms of the remaining unknowns.

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{q=1 \\ q \neq i}}^n a_{iq} x_q^{(k)} \right), \quad i = 1, \dots, n.$$

$$k = 0, 1, 2, \dots;$$

$x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})$ is an initial approximation,

where (k) indicates the k -th iteration step.

2.3.4 Gauss-Seidel iteration method

Gauss-Seidel method, is similar to the Jacobi method (also called one step cyclic iteration or the method of successive displacements), is given by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{q=1}^{i-1} a_{iq} x_q^{(k+1)} - \sum_{q=i+1}^n a_{iq} x_q^{(k)} \right), \quad i = 1, \dots, n.$$

$$k = 0, 1, 2, \dots;$$

$x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})$ is an initial approximation.

The Gauss-Seidel method differs from the Jacobi method in the fact that it (Gauss-Seidel) always uses the most recently obtained values for x_i .

The disadvantage of Jacobi and Gauss-Seidel methods is the solution converges only for diagonally dominant coefficient matrix.

2.3.5 Choleski's method (Triangular factorization method)

This method is designed for symmetric square matrices. In this we split the matrix A as $A = T.T'$, where T is a lower triangular matrix and T' is its transpose which is obtained by interchanging the rows and columns T . The decomposition of the symmetric square matrix can be done by gauss elimination. This method involves just half of arithmetic work than the previous methods.

2.3.6 Comparison of various methods

Direct and indirect methods have their advantages and disadvantages and a choice of method depends on a particular system of equations. The direct methods yield a solution in a finite number of steps for any non-

singular set of equations, while in an iterative method the amount of computation depends on the accuracy desired. In general, it is preferable to use a direct method for the solution of a linear system. However for large systems, an iterative method yields the solution faster and should therefore be preferred.

The rounding off errors also get propagated in the elimination method whereas in the iteration techniques only the rounding off errors committed in the final iteration have any effect. In general, the iteration methods have smaller round-off errors for iteration is a self-correcting technique. Thus the use of an iterative method for ill-conditioned system is preferable.

On the other hand, an iterative method may not always converge. When it converges, the iterative method is definitely better than direct methods.

2.4 Solution of Algebraic & Transcendental Equations

The equations of the form $f(x) = 0$ are called Algebraic or Transcendental according as $f(x)$ is purely a polynomial in x or contains some other functions such as trigonometric, logarithmic, and exponential functions etc. Thus the equation $x^3 - 4x^2 + 9x = 0$ and $ax^2 + \log(x-3) + e^x \sin x = 0$ are algebraic and transcendental respectively. By finding the solution of an equation $f(x) = 0$, we mean to find those values of x for which $f(x) = 0$ is satisfied. Such values of x are called roots of the given equation. Thus a is a root of the equation $f(x) = 0$ if and only if $f(a) = 0$.

2.4.1 Bisection method

Suppose $f(x) = 0$ has real roots in the interval $[a_0, b_0]$. Bisect the interval at $c_0 = (a_0 + b_0)/2$ i.e., the point c_0 divides interval $[a_0, b_0]$ into two equal parts. Choose one of these intervals $[a_0, c_0]$ and $[c_0, b_0]$ such that $f(x)$ has opposite signs at the end point and call this interval as $[a_1, b_1]$. Bisect $[a_1, b_1]$ at c_1 and choose either $[a_1, c_1]$ and $[c_1, b_1]$ depending upon whether $f(a_1)f(c_1) < 0$ or $f(b_1)f(c_1) < 0$ and call this interval as $[a_2, b_2]$. Repeating this process, we either hit at a root at some stage or arrive at a sequence of nested intervals $[a_n, b_n]$ after n iterations, which contains the root.

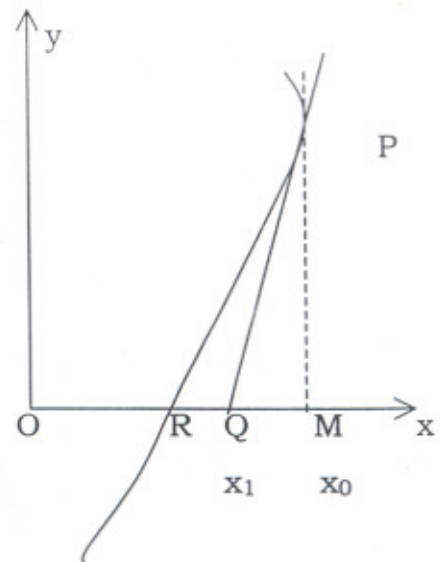
The advantage of this method is that it is simple to use. In this method to ensure that the chosen interval (a, b) contains only one root of the equation $f(x) = 0$; the derivative of $f(x)$, $f'(x)$ must contain the same sign in the interval (a, b) i.e., $f(x)$ must be monotonic. The disadvantage is convergence is very slow and requires much number of iterations.

2.4.2 Newton-Raphson method

The graph of $y = f(x)$ intersects x -axis at R corresponding to the equation $f(x) = 0$, suppose current approximation to the root is x_r in the r th iteration. Let P be the corresponding point on the curve, which cuts the x -axis at Q , where $x = x_1$.

It gives approximation of the root $PM = f(x_r)$ and $QM = x_r - x_{r+1}$, so that the slope of the

tangent at P is $PQ = f'(x_r) = \frac{f(x_r)}{(x_r - x_{r+1})}$



$$\text{Or } x_{r+1} - x_r = -f(x_r) / f'(x_r)$$

$$\text{Or } x_{r+1} = x_r - f(x_r) / f'(x_r)$$

The process is continued till the difference between the successive approximations, $x_n - x_{n+1}$, is less than prescribed degree of accuracy of Q , i.e., $|x_{n+1} - x_n| < Q$.

The advantage of this method is convergence is quite fast. But the convergence depends on the selection of initial value, and in some situations the solution may not converge at all.

2.4.3 Secant method

In this method two initial values are chosen such that x_1 and x_2 are such that $f(x_1).f(x_2) < 0$. Find the next approximation x_3 by using the formula

$$x_3 = x_2 - \left[\frac{x_2 - x_1}{f(x_2) - f(x_1)} \right] f(x_2).$$

Use the iterative procedure to calculate the

$$\text{next approximation by using } x_r = x_r - \left[\frac{x_r - x_{r-1}}{f(x_r) - f(x_{r-1})} \right] f(x_r).$$

Repeat the process until the difference between the two successive values is less than the prescribed error.

2.4.4 False position method or Regula Falsi method

This is same as Secant method, except that the initial points x_1 and x_2 are taken in such a way that they lie on opposite sides of R , which corresponds to actual value of $y = f(x)$. This means that $f(x_1)$ and $f(x_2)$ are of opposite signs, in other words $f(x_1).f(x_2) < 0$. From two such points x_1 and x_2 , the next iterate x_3 is determined using the iterate formula. Next to determine x_4 , choose with x_3 the value of x_1 or x_2 such that $f(x_1).f(x_3) < 0$.

0 or $f(x_3).f(x_2) < 0$. The procedure is continued every time choosing end points of the interval (x_{r-1}, x_r) such that $f(x_{r-1}).f(x_r) < 0$. The Regula- Falsi method combines the features of Secant and Bisection methods. However the convergence rate of solution is 1.618 and for Newton-Raphson method it is 2. But while the solution in Newton-Raphson method is not guaranteed, in these two methods we are sure to arrive at a solution after several iterations. However this method is useful if the range of the root is not clearly known.

2.5 Interpolation

The method of estimating unknown value with the help of given set of observations is known as **interpolation**. Mathematically we can say that given a set of values of the function $f(x)$ for certain values of the independent variable x , is known as interpolation. The value of $f(x)$ thus determined is known as interpolated value e.g. let the observations for $x = a, a+h, a+2h, \dots, a+nh$ for the function $y = f(x)$ be $f(a), f(a+h), \dots, f(a+nh)$. The method of finding $f(m)$ for $x = m$, where m lies in the range a and $a+nh$ is known as **Interpolation** and if the value of $x = m$ lies outside this range then it is known as **Extrapolation**.

2.5.1 Interpolation with equal intervals

Newton's Gregory Forward Interpolation Formula

Let the function $y = f(x)$ take the values $y_0, y_1, \dots, y_n$ corresponding to the values $x_0, x_1, \dots, x_n$ of x . Let these values of x be equi-spaced such that $x_i = x_0 + ih$ ($i = 0, 1, \dots$). Assuming $y(x)$ to be a polynomial of the n th degree in x such that $y(x_0) = y_0, y(x_1) = y_1, \dots, y(x_n) = y_n$. We can evaluate y for $x = x_0 + ph$ by using

$$y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!} \Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!} \Delta^3 y_0 + \dots + \frac{p(p-1)\dots(p-(n-1))}{n!} \Delta^n y_0$$

The above formula is known as Newton's forward interpolation formula. Here x_0 is initial value of x , h is interval and p is $(x - x_0)/h$.

Newton's backward interpolation formula.

Let the function $y = f(x)$ take the values $y_0, y_1, \dots, y_n$ corresponding to the values $x_0, x_0+h, x_0+2h, \dots$ of x . Suppose it is required to evaluate $f(x)$ for $x = x_n + ph$, where p is any real number. Then we can evaluate y using

$$y_p = y_n + p\nabla y_n + \frac{p(p+1)}{2!} \nabla^2 y_n + \frac{p(p+1)(p+2)}{3!} \nabla^3 y_n + \dots + \frac{p(p+1)\dots(p+(n-1))}{n!} \nabla^n y_n$$

The above formula is called Newton's backward interpolation formula as it contains y_n and backward difference of y_n .

Stirling's formula

This formula is derived by taking mean of Gauss forward interpolation formula (central difference) and Gauss backward interpolation formula (central difference). The Stirling's formula is

$$y_p = y_0 + p \left(\frac{\Delta y_0 + \Delta y_{-1}}{2} \right) + \frac{p^2}{2!} \Delta^2 y_{-1} + \frac{p(p^2-1)}{3!} \left(\frac{\Delta^3 y_{-1} + \Delta^3 y_{-2}}{2} \right) + \dots + \frac{p^2(p^2-1)}{4!} \Delta^4 y_{-2}$$

This formula involves means of the odd differences just above and below the central line and even differences on this line as shown below:

$$\dots y_0 \dots \left(\frac{\Delta y_{-1}}{\Delta y_0} \right) \dots \Delta^2 y_{-1} \dots \left(\frac{\Delta^3 y_{-2}}{\Delta^3 y_{-1}} \right) \dots \Delta^4 y_{-2} \dots \left(\frac{\Delta^5 y_{-3}}{\Delta^5 y_{-2}} \right) \dots \Delta^6 y_{-3} \dots \text{Central line.}$$

Bessel's formula

The Bessel's formula is given by

$$y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!} \left(\frac{\Delta^2 y_{-1} + \Delta^2 y_0}{2} \right) + \frac{(p-1/2)p(p-1)}{3!} \Delta^3 y_{-1} \\ + \frac{(p+1)(p-1)(p-2)}{4!} \left(\frac{\Delta^4 y_{-2} + \Delta^4 y_{-1}}{2} \right) + \dots$$

This is a very useful formula for practical purposes. It involves odd differences below the central line and means of even differences on and below this line.

Choice of an interpolation formula

The coefficients in the central difference formulae are smaller and converge faster than those in Newton's formulae. After a few terms, the coefficients in the Stirling's formula decrease more rapidly than decrease more rapidly than those of Newton's formula. As such, whenever possible, central difference formulae should be used in preference to Newton's formulae.

The right choice of an interpolation formula however, depends on the position of the interpolated value in the given data. The following rules can be used for choosing the interpolation formula.

1. To find a tabulated value near the beginning of the table, use Newton's forward formula.
2. To find a value near the end of the table, use Newton's backward formula.
3. To find an interpolated value near the center of the table, use either Stirling's or Bessel's formula.
4. If interpolation is required for p lying between $-\frac{1}{4}$ and $\frac{1}{4}$, prefer Stirling's formula.
5. If interpolation is desired for p lying between $\frac{1}{4}$ and $\frac{3}{4}$, use Bessel's formula.

2.5.2 Interpolation with unequal intervals

Lagrange's interpolation formula

If $y = f(x)$ takes the value $y_0, y_1, \dots, y_n$ corresponding to $x = x_0, x_1, \dots, x_n$, then

$$\begin{aligned}
 f(x) = & \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} y_0 \\
 & + \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} y_1 \\
 & + \dots + \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})} y_n
 \end{aligned}$$

This is known as Lagrange's interpolation formula for unequal intervals. Lagrange's formula can be applied whether the values x_i are equally spaced or not. This formula has the drawback that on insertion of another interpolation value, it needs the recalculation of interpolation coefficients.

Newton's divided difference formula

Let $y_0, y_1, \dots, y_n$ be the values of $y = f(x)$ corresponding to $x = x_0, x_1, \dots, x_n$.

Then $f(x)$ at any value of x can be calculated by using

$$\begin{aligned}
 y = f(x) = & y_0 + (x-x_0)[x_0, x_1] + (x-x_0)(x-x_1)[x_0, x_1, x_2] \\
 & + (x-x_0)(x-x_1)(x-x_2)[x_0, x_1, x_2, x_3] + \dots \\
 & + (x-x_0)(x-x_1)\dots(x-x_n)[x_0, x_1, \dots, x_n]
 \end{aligned}$$

Where $[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}$ (first divided difference)

$$[x_0, x_1, x_2] = \frac{[x_1, x_2] - [x_0, x_1]}{x_2 - x_0} = \frac{1}{2h} \left[\frac{\Delta y_1}{h} - \frac{\Delta y_0}{h} \right] = \frac{1}{2!h^2} \Delta^2 y_0$$

(Second divided difference)

$$[x_0, x_1, x_2, \dots, x_n] = \frac{1}{n!h^n} \Delta^n y_0 \text{ (nth divided difference)}$$

2.6 Numerical differentiation

It is the process of calculating the value of the derivative of a function at some assigned value of x from the given set of values (x_i, y_i) . To compute dy/dx , we first replace the exact relation $y = f(x)$ by the best interpolating

2.6.2 Newton's backward difference formula

The first derivative at x_n , i.e., $(x_n + ph)$ where h is interval, is given by

$$f'(x) = \frac{1}{h} \left[\nabla y_n + \frac{2p+1}{2!} \nabla^2 y_n + \frac{3p^2+6p+2}{3!} \nabla^3 y_n + \frac{4p^3+18p^2+22p+6}{4!} \nabla^4 y_n + \dots \right]$$

and second derivative is given by

$$f''(x) = \frac{1}{h^2} \left[\nabla^2 y_n + (p+1) \nabla^3 y_n + \frac{6p^2+18p+11p}{12} \nabla^4 y_n + \dots \right]$$

At $x = x_0$, $p = 0$; substituting this in the above equations

$$f'(x_n) = \frac{1}{h} \left[\nabla y_n + \frac{1}{2} \nabla^2 y_n + \frac{1}{3} \nabla^3 y_n + \frac{1}{4} \nabla^4 y_n + \frac{1}{5} \nabla^5 y_n + \frac{1}{6} \nabla^6 y_n + \dots \right]$$

$$f''(x_n) = \frac{1}{h^2} \left[\nabla^2 y_n + \nabla^3 y_n + \frac{11}{12} \nabla^4 y_n + \frac{5}{6} \nabla^5 y_n + \frac{137}{180} \nabla^6 y_n + \dots \right]$$

2.6.3 Stirling's formula

$$f'(x_0) = \frac{1}{h} \left[\frac{\Delta y_0 - \Delta y_{-1}}{2} - \frac{1}{6} \left(\frac{\Delta^3 y_{-1} + \Delta^3 y_{-2}}{2} \right) + \frac{1}{30} \left(\frac{\Delta^5 y_{-2} + \Delta^5 y_{-3}}{2} \right) + \dots \right]$$

$$f''(x_0) = \frac{1}{h^2} \left[\Delta^2 y_{-1} - \frac{1}{12} \Delta^4 y_{-2} + \frac{1}{90} \Delta^6 y_{-3} - \dots \right]$$

2.6.4 Bessel's formula

$$f'(x_0) = \frac{1}{h} \left[\Delta y_0 - \frac{1}{4} (\Delta^2 y_{-1} + \Delta^2 y_0) + \frac{1}{12} \Delta^3 y_{-1} + \frac{1}{24} (\Delta^4 y_{-2} + \Delta^4 y_{-1}) \right. \\ \left. - \frac{1}{120} \Delta^5 y_{-2} - \frac{1}{240} (\Delta^6 y_{-3} + \Delta^6 y_{-2}) \right]$$

2.7 Numerical integration

The process of evaluating a definite integral from a set of tabulated values of the integrand $f(x)$ is called **Numerical integration**. This process when applied to a single variable is known as quadrature. The problem of

numerical integration, like that of numerical differentiation, is solved by representing $f(x)$ by an interpolation formula and then integrating it between the given limits. In this ways, we can derive quadrature formulae for approximate integration of a function defined by a set numerical values only.

2.7.1 Trapezoidal rule

$$\int_{x_0}^{x_0+nh} f(x)dx = \frac{h}{2}[(y_0 + y_n) + 2(y_1 + y_2 + \dots + y_{n-1})]$$

2.7.2 Simpson's one-third rule

$$\int_{x_0}^{x_0+nh} f(x)dx = \frac{h}{3}[(y_0 + y_n) + 4(y_1 + y_3 + \dots + y_{n-1}) + 2(y_2 + y_4 + \dots + y_{n-2})]$$

2.7.3 Simpson's three eighth rule

$$\int_{x_0}^{x_0+nh} f(x)dx = \frac{3h}{8}[(y_0 + y_n) + 3(y_1 + y_2 + y_4 + y_5 + \dots + y_{n-1}) + 2(y_3 + y_6 + \dots + y_{n-3})]$$

2.7.4 Weddle's rule

$$\int_{x_0}^{x_0+nh} f(x)dx = \frac{3h}{10}[y_0 + 5y_1 + y_2 + 6y_3 + y_4 + 5y_5 + 2y_6 + 5y_7 + y_8 + \dots]$$

2.8 Linear programming

Simplex method

Linear programming is widely used to tackle a number of industrial, economic, marketing and distribution problems. This technique has found its applications to important areas of product mix, blending problems and diet problems. A linear programming problem (L.P.P.) may

be applied to any situation where a linear function of variables has to be optimized subject to a set of linear equations or inequalities.

To solve a practical problem it is to be presented in a linear programming form, it requires defining the variables involved, establishing relationships between them and formulating the objective function and the constraints. Then the L.P.P. is to be expressed in the standard form.

A linear programming problem is said to be in **standard form** if it possesses the following characteristics:

1. Objective function is of maximization type.
2. All constraints are expressed as equations.
3. Right hand side of each constraint is non-negative.
4. All variables are non-negative.

Working procedure of the simplex method.

Assuming the existence of an initial basic feasible solution, an optimal solution to any L.P.P. by simplex method is obtained in the steps shown below:

1. (i) Check whether the objective function is to be maximized or minimized.
(ii) Check whether all constants of the constraint equations are positive.
2. Express the problem in standard form.
3. Find initial basic feasible solution.
4. Apply optimality test.
5. (i) Identify the incoming and outgoing variables.
(ii) Iterate towards an optimal solution.
6. Go to step 4 and repeat the computational procedure until either an optimal solution is obtained.

Chapter 3

ALGORITHMS

3.1 Solution of simultaneous equations

Gauss Elimination method (with pivoting)

1. Form a coefficient matrix $A(a_{ij})$ of $n \times (n+1)$ size (n is number of variables) with the coefficients of linear equations to be solved
2. for $i = 1$ to $n-1$ (elimination with pivoting)

Identify the largest element in the column with $columnno = i$;
 Interchange the row with $rowno = i$ and the row which the largest element in that column.;

for $k = i+1$ to n

$$u = a_{ik} / a_{kk};$$

for $j = 1$ to $n+1$

$$a_{ij} = a_{ij} - u a_{kj}$$
3. (backsubstitution)

for $i = n$ to 1 in steps of -1

$$s = 0;$$

for $j = i+1$ to n

$$s = s + a_{ij} * x_j ;$$

$$x_i = (a_{i(n+1)} - s) / a_{ii};$$

Gauss-Jordan method (with pivoting)

1. Form a coefficient matrix $A(a_{ij})$ of $n \times (n+1)$ size (n is number of variables) with the coefficients of linear equations to be solved
2. for $i = 1$ to n (elimination with pivoting)

Identify the largest element in the column with $columnno = i$;
 Interchange the row with $rowno = i$ and the row which the largest element in that column.;

for $k = 1$ to n

$$\text{if}(i \neq k)$$

$$u = a_{ik} / a_{kk};$$

for $j = 1$ to $n+1$

$$a_{ij} = a_{ij} - u a_{kj};$$

Gauss-Seidel iterative method

1. Form augmented matrix A of size $n \times (n+1)$ with coefficients of linear equation
2. For $i = 1$ to n

$$s = 0$$

for $j = 1$ to n

if $(j \neq i)$ $s = s + a_{ij} * x_i$;

$$\text{temp} = (a_{i(n+1)} - s) / a_{ii};$$

$$x_i = \text{temp};$$
3. Repeat the process till the difference between two successive values of x_i is less than the prescribed accuracy (E).

Jacobi method

This method is similar to the Gauss-Seidel method except that the next approximation is calculated by using the values calculated in the previous iteration.

Choleski's method

1. Form augmented matrix A of size $n \times (n+1)$ with coefficients of linear equation
2. Calculate elements of Lower triangle by using the formula

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} u_{kj} l_{ik}) / u_{jj} \quad \text{where } U \text{ is upper triangular matrix.}$$

3. Calculate the values of x_i using backsubstitution.

3.2 Solution of Algebraic and transcendental equations

Bisection method

1. Choose two approximate values a and b such that $f(a) \cdot f(b) < 0$.
2. Calculate mid point by $c = (a+b)/2$
3. If $f(c) \cdot f(b) < 0$ rename c as a otherwise rename c as b .

4. Repeat from step 2
5. Stop evaluation when the difference of two successive values of c obtained in step 2 is less than E , the prescribed accuracy.

Newton-Raphson method

1. Choose one approximate initial value x_i .
2. Find next approximation by using the formula

$$x_{i+1} = x_i - f(x_i) / f'(x_i)$$
3. Repeat step 2 until difference between the two successive values is less than the prescribed accuracy (E). i.e., $|x_{i+1} - x_i| < E$.

Secant method

1. Choose two values x_1 and x_2 such that $f(x_1).f(x_2) < 0$.
2. Find next approximation x_3 by $x_3 = x_2 - \left[\frac{x_2 - x_1}{f(x_2) - f(x_1)} \right] f(x_2)$
3. evaluate the successive approximation using the recursive formula $x_{r+1} = x_r - \left[\frac{x_r - x_{r-1}}{f(x_r) - f(x_{r-1})} \right] f(x_r)$
4. Stop processing when difference between two successive values is less than the prescribed accuracy (E). i.e., $|x_{r+1} - x_r| < E$.

False position or Regula-falsi method

1. Choose two values x_1 and x_2 such that $f(x_1).f(x_2) < 0$.
2. Find next approximation x_3 by $x_3 = x_2 - \left[\frac{x_2 - x_1}{f(x_2) - f(x_1)} \right] f(x_2)$
3. If $f(x_3).f(x_2) < 0$ then go to next step. If not rename x_1 as x_2 and go to next step
4. Evaluate successive approximation using

$$x_{r+1} = x_r - \left[\frac{x_r - x_{r-1}}{f(x_r) - f(x_{r-1})} \right] f(x_r) \quad \text{But for applying the formula for}$$

x_{r+1} , ensure that for $f(x_r) \cdot f(x_{r-1}) < 0$, if not then rename x_{r-2} as x_{r-1} .

5. Stop evaluation when difference between two successive values is less than the prescribed accuracy (E). i.e., $|x_{r+1} - x_r| < E$.

3.3 Integration

Trapezoidal Rule

1. Get the values of $y_i = f(x)$ and corresponding values of x_i . Get number of values n .
2. Calculate increment $h = x_1 - x_0$.
3. Initialize integral to zero. ($s=0$)
4. for $i=0$ to $n-1$
 $s = s + (y_i + y_{i+1}) * h / 2;$

Simpson's one-third rule

1. Get the values of $y_i = f(x)$ and corresponding values of x_i . Get number of values n .
2. Calculate increment $h = x_1 - x_0$.
3. Initialize integral to zero. ($s=0$)
4. for $i=0$ to $n-1$ in steps of 2
 $s = s + (y_i + 4 * y_{i+1} + y_{i+2}) * h / 3;$

Simpson's three-eighth rule

1. Get the values of $y_i = f(x)$ and corresponding values of x_i . Get number of values n .
2. Calculate increment $h = x_1 - x_0$.
3. Initialize integral to zero. ($s=0$)
4. for $i=0$ to $n-1$ in steps of 3
 $s = s + (y_i + 3 * y_{i+1} + 3 * y_{i+2} + y_{i+3}) * 3 * h / 8;$

Simpson's three-eighth rule

1. Get the values of $y_i = f(x)$ and corresponding values of x_i . Get number of values n .
2. Calculate increment $h = x_1 - x_0$.
3. Initialize integral to zero. ($s=0$)
4. for $i=0$ to $n-1$ in steps of 6

$$s = s + (y_i + 5y_{i+1} + y_{i+2} + 6y_{i+3} + y_{i+4} + 5y_{i+5} + y_{i+6}) * 3h / 10;$$

3.4 Linear programming problem (Simplex method)

1. Initialize all arrays to zeros.
2. Set coefficients of slack variables equal to 1
3. Put slack variables in the basis
4. Get the constraints and the objective function
5. Calculate the constraints and the objective function
6. Calculate C_j and identify the incoming variable
7. Perform optimality test (check whether all $C_j \leq 0$)
If the above condition is not satisfied go to step 9
8. Calculate the optimal solution and stop
9. Calculate values of θ
10. If all $\theta < 0$, give message unbounded solution and stop
11. Search for outgoing variable
12. Divide key row by key element
13. Make all other elements of key column zero and go to step 6

DESIGN (MODELING)

4.1 Unified Modeling Language (UML)

The UML is one of the most exciting tools in the world of system development today. The UML enables the system builders to create blueprints that capture their visions in a standard, easy-to-understand way and communicate them to others.

The UML is the brainchild of Grady Booch, James Rumbaugh, and Ivan Jacobson. Recently dubbed “the Three Amigos,” these gentlemen worked in separate organizations through the ‘80s and early ‘90s, each devising his own methodology for object-oriented analysis and design. Their methodologies achieved preeminence over those of numerous competitors. By the mid’90s, they began to borrow ideas from each other, so they decided to evolve their work together. In 1994 Rumbaugh joined Rational Software Corporation, where Booch was already working. Jacobson enlisted at Rational a year later.

Draft versions of the UML began to circulate throughout the software industry and the resulting feedback brought substantial changes. As many corporations felt the UML would serve their strategic purposes, a UML consortium sprang up. Members included DEC, Hewlett-Packard, Intellicorp, Microsoft, Oracle, Texas Instruments, Rational and others. In 1997, the consortium produced version 1.0 of the UML and submitted it to the Object Management Group (OMG) in response to OMG’s request for a proposal for a standard modeling language.

The consortium expanded, generated version 1.1, and submitted it to the OMG, who adopted it in late 1997. The OMG took over the maintenance of the UML and produced two more versions in 1998.

The UML has become a de facto standard in the software industry, and it continues to evolve.

4.2 GRAPPLE

GRAPPLE is an acronym of Guidelines for Rapid APPLication Engineering. The ideas within GRAPPLE aren't original. They are a distillation of the ideas of a number of others. The Three Amigos have created the Rational Unified Process, and prior to that, each Amigo had his own process; the ideas in those processes similar to GRAPPLE.

The first in GRAPPLE's name, Guidelines, is important: This isn't a methodology written in stone. Instead, it's a set of adaptable, flexible ideas. Think of it as a simplified skeleton of a development process.

GRAPPLE consists of five segments. Each segment, in turn, consists of a number of actions. Each action produces a work-product, and each action is a responsibility of a particular player. GRAPPLE is intended for object-oriented systems. Thus, the actions within each segment are geared toward producing work-products of an object-oriented nature.

The segments are

1. Requirements gathering
2. Analysis
3. Design
4. Development
5. deployment

4.3 Organization of the components

The entire software is divided into three groups. They are:

1. Opening menu
2. Graphical User Interface
3. Numerical library or package

4.3.1 Opening Menu

This is a GUI program with which a novice user can interact with the software. It contains a menu bar. Each menu item in the menu bar contains pull down menu of submenu items. The user can select any method from the menu. On selection of a method by the user the opening menu program will activate the GUI program (Front-end tool for that type of problem).

4.3.2 Graphical User Interface

A GUI front-end is provided for every method. The user can enter data in the edit boxes or text fields. On entry of data the user can press appropriate button. On pressing the button Front-end GUI program creates an object of corresponding class. The class information is available in the numerical static library in case of C++ and in package in case of Java2. After creation of the object, required method is called by passing data as message to that method. The class method returns the result to the GUI program. The GUI program displays the results.

4.3.3 Numerical library or package

The numerical library consists of classes for each problem. Each class contains different attributes and methods. For example matrix has one class which has methods or operations like addition, subtraction, multiplication, inverse, transpose and determinant.

4.4 Classes

A *class* is a category or group of things that have similar attributes and common behaviors. In UML a rectangle is the icon that represents the class. It is divided into three areas. The uppermost area contains the name, the middle area holds the attributes, and the lowest area shows the operations. A *class diagram* consists of a number of these rectangles connected by lines that show how the classes relate to one

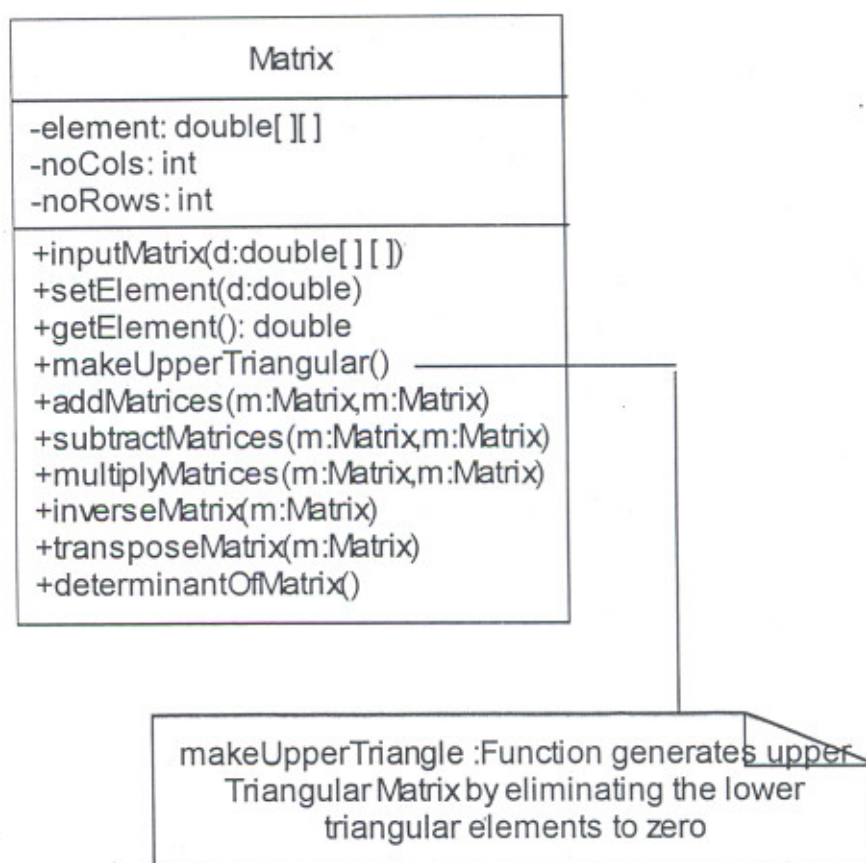


Fig 4.1 Matrix Class

The above figure shows the Matrix class used in the numerical package. In the above class the attributes are either private. The access modifier is represented with prefix -, +, # for attributes and operations. In UML '-' represent private access, '+' represent public access, and # represent protected access. The data type of attributes is represented after the attribute separated by colon. The data type of return value of the operation is also expressed on the right side separated by colon. The arguments of the operation are expressed in the parenthesis after the operation name.

In the above class a rectangle folded at the right top indicates the attached note. This attached note is used to write comments regarding

In this class attributes are given private access and these attributes are accessed by the outside programs by using public methods `inputMatrix()`, `setElement()`, `getElement()`. The other methods like `addMatrices()`, `inverseMatrix()` perform matrix operations.

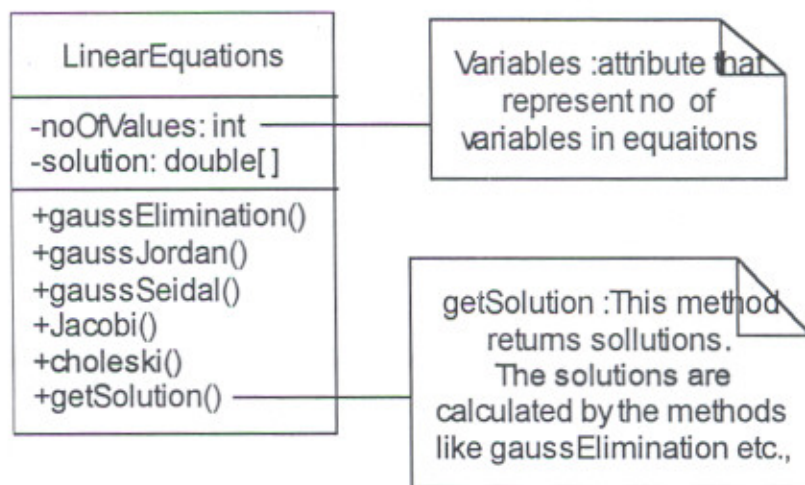


Fig 4.2 Linear Equations Class

The above diagram shows Linear Equations class, which performs solution of Linear equations or simultaneous equations.

The Fig 4.3 shows class icon for solution of non-linear equations or algebraic and transcendental equations. In this class the equation for which root is to be calculated is accepted in the form of a string. The value of this expression is calculated by using the `parse expression()` operation of the Expression class. In this class the attributes are assigned with default values. The UML notation for these default values is shown figure.

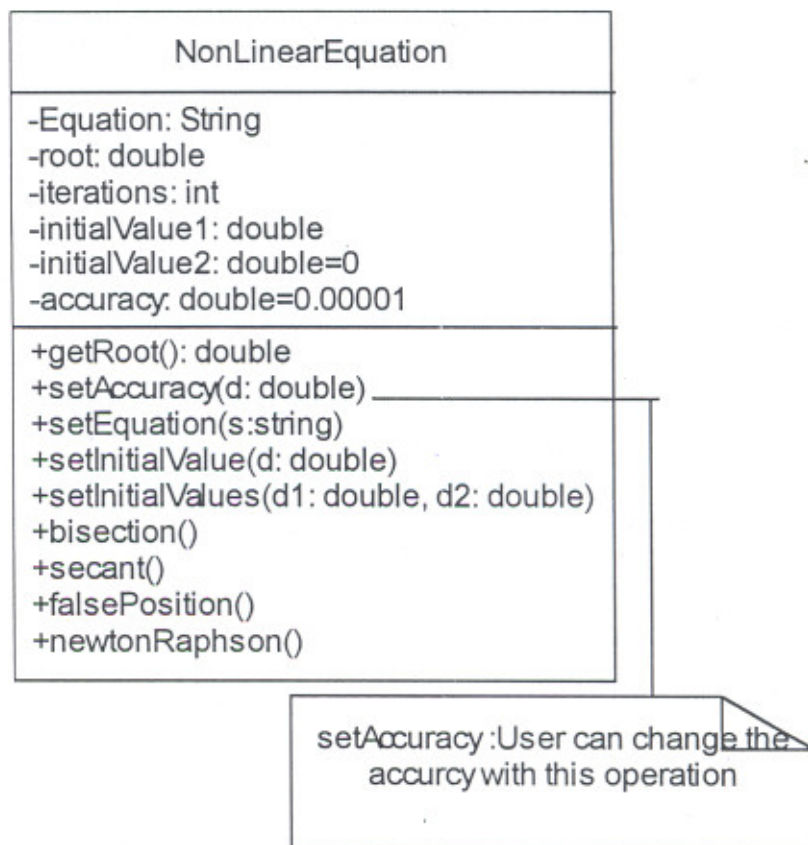


Fig 4.3 Non Linear Equations Class

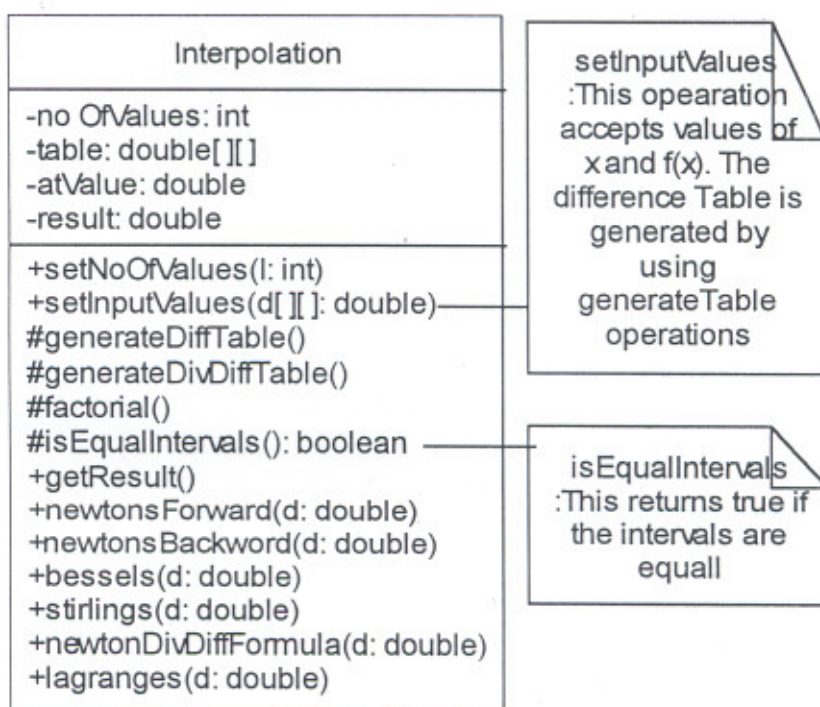


Fig 4.4 Interpolation Class

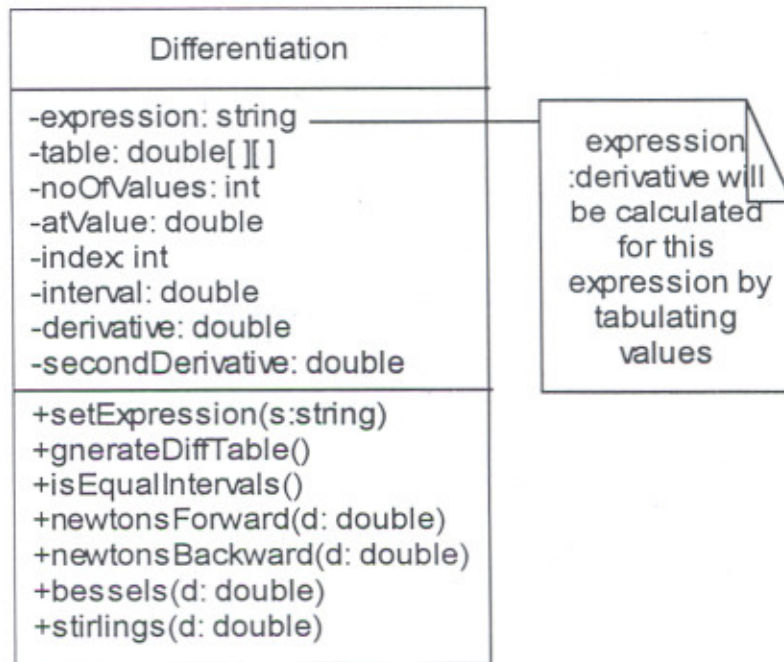


Fig 4.5 Differentiation Class

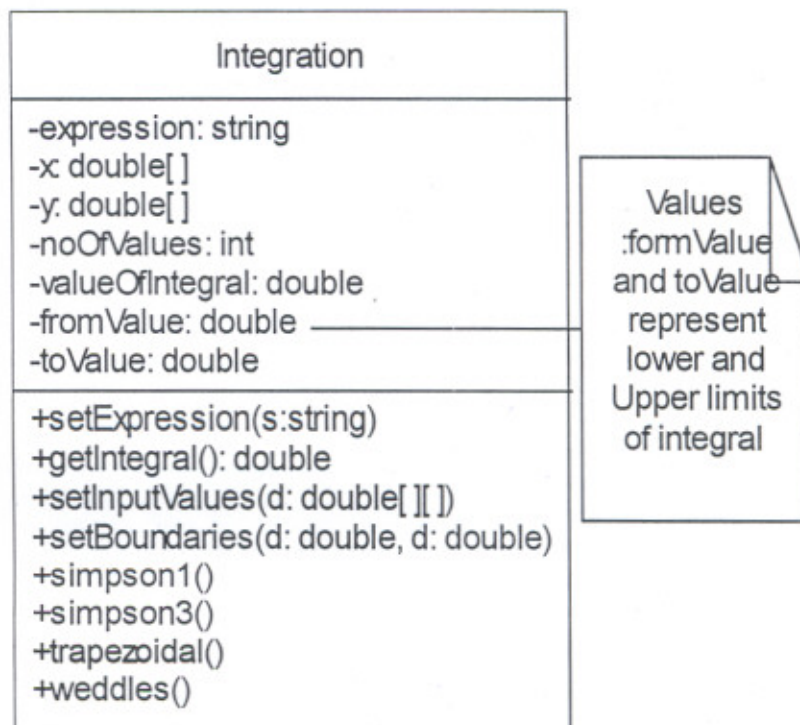


Fig 4.6 Integration Class

Expression
-expression: String -value: double -tokens: String[] -tempToken: String
+parseExpression() -getToken(): String #tokenize() #hasMoreTokens(): boolean +eval_exp1(): double +eval_exp2(): double +eval_exp3(): double +eval_exp4(): double +eval_exp5(): double +function(): double +atom(): double

Fig 4.7 Expression Class
Used for expression Parsing

LinearProgramming
-decisionVariables: int -slackVariables: int -table: double[][] -coefficientsObjectiveFunction: double[] -solution: double[]
+set Input values(d: double[][]) +setNoOfVariables(l: int, l: int) +setcoefficientsObjectiveFunction(d: double[]): double[] +simplex()

Fig 4.8 Linear Programming Class

All the modeling diagrams for the design of this software are prepared using paradigm plus 3.6 Enterprise Edition. Paradigm plus is a software tool of object orient modeling using UML. Paradigm Plus stores all information about a project in a set of one or more integrated, active databases called object repositories ("objectories").

These repositories can be distributed, and they serve as the foundation for all object-oriented application development, storing all properties of objects and their relationships. Through the use of the various generic editors, developers can create, view, and modify all stored objects.

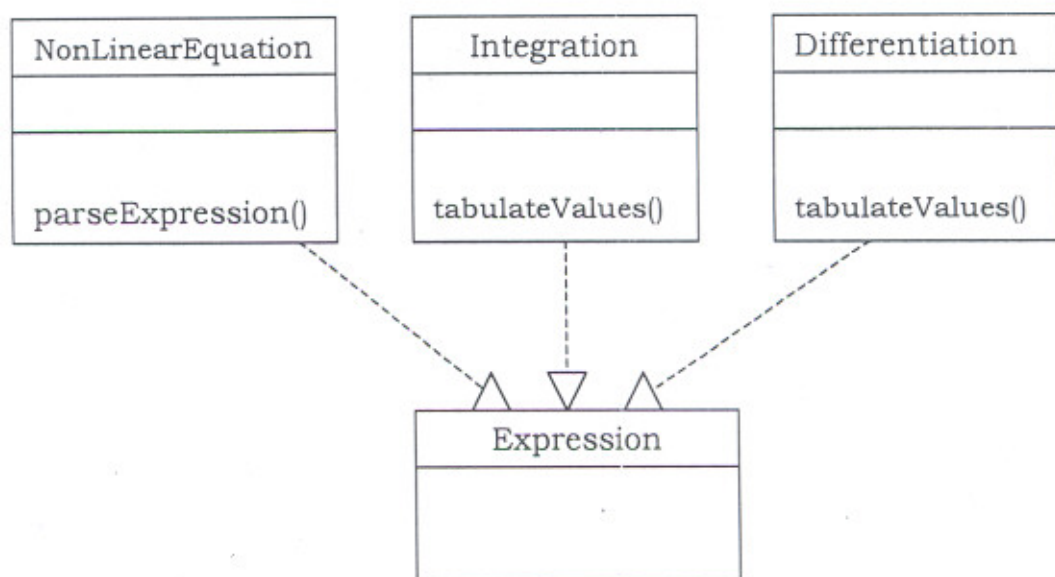


Fig. 4.9 Dependency diagram

Figure 4.9 shows the dependency of different classes. A dotted line with arrowhead represents dependency. The above diagram represents that the classes **NonLinearEquation**, **Differentiation**, and **Integration** depend on **Expression**.

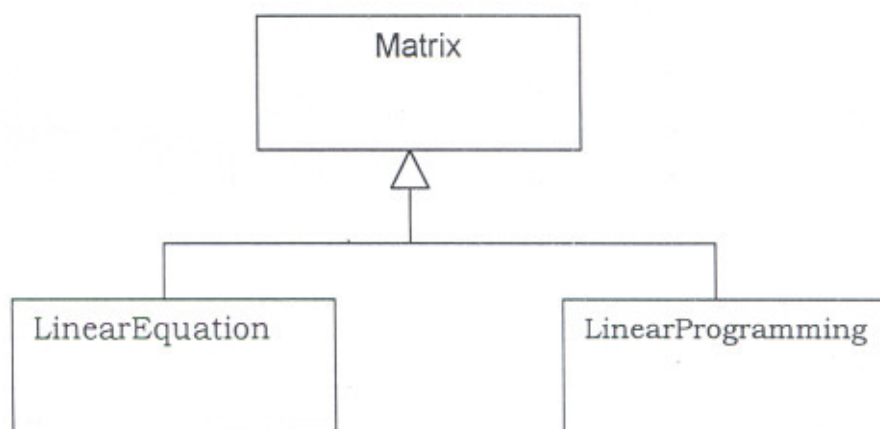


Fig. 4.10 Inheritance diagram

Figure 4.10 shows an inheritance diagram. In UML inheritance diagram indicates the base classes and derived classes. The figure means that LinearEquation and LinearProgramming classes are inherited from the matrix class.

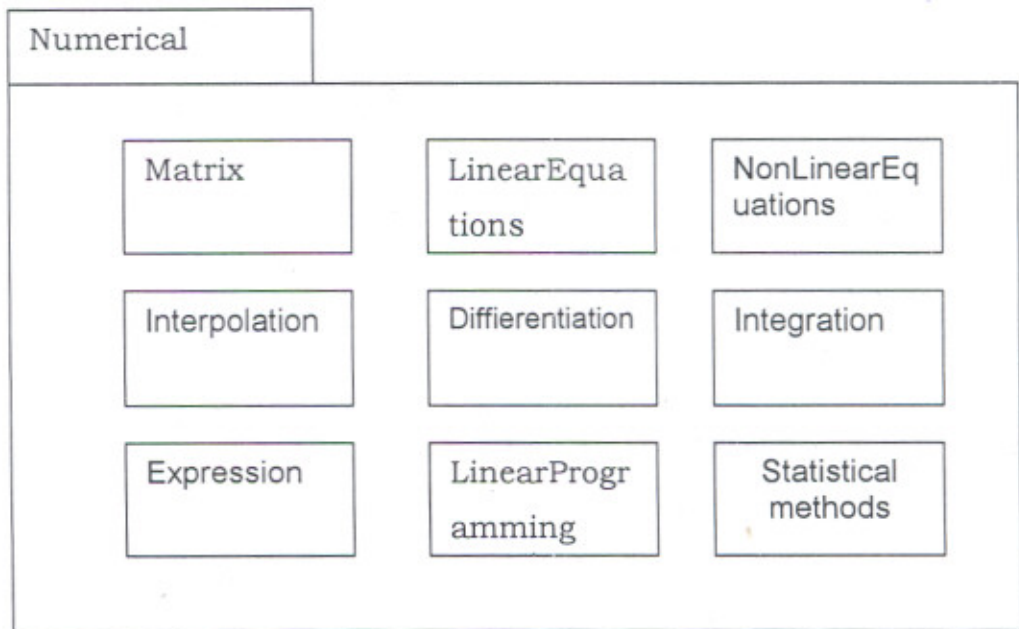


Fig. 4.11 Numerical Package

Figure 4.11 is a package diagram that represents classes of the package.

4.5 Modeling the GUI programs

4.5.1 General principles of GUI design

User interface design, equal parts art and science, draws upon the vision of the graphic artist, the findings of the human factors researcher, and the intuitions of the potential user. After much experience with WIMP (Windows, Icons, Menus, and Pointing Devices) interfaces, some general principles have emerged. Here are some of the major ones:

1. Understand what the user has to do. User interface designers typically perform a task analysis to understand the nature of the user's work. Our use case analysis roughly corresponds to this.
2. Make the user feel in control of the interaction. Always include the capability for the user to cancel an interaction after it's started.
3. Give the user multiple ways to accomplish each interface-related action (like closing a window or a file) and forgive user errors gracefully.
4. Because of cultural influences, our eyes are drawn to the upper-left corner of a screen. Put the highest priority information there.
5. Take advantage of spatial relationships. Screen components that are related should appear near one another, perhaps with a box around them.
6. Emphasize readability and understanding. (Words for all of us to live by!) Use active voice to communicate ideas and concepts.
7. Even though it is possible to include upwards of umpteen gazillion colors on a screen, limit the number of colors you use. Limit the number severely. Too many colors will distract the user from the task at hand. By the way it's a good idea to give the user the option of modifying the colors.
8. Colors should not be used to represent meaning. It is not always easy for the user to see an association between a color and a meaning. A point to be remembered in this context is that some users (10% of adult males) have color confusion, and may find it difficult to distinguish one color from another.
9. As is the case with color, use limited fonts. Avoid Italics and ornate fonts.
10. Try to keep components (like buttons and list boxes) the same size as much as possible. Multiplicity of colors, and a variety of fonts, creates a patchwork that GUI specialists call a "clown-pants" design.

11. Left-align components and data fields. Line them up according to their left-side edges. This minimizes eye movements when the user has to scan the screen.
12. When the user has to read and process information and then click a button put the buttons in a column to the right of the information or in a row below and to the right of the information. This is consistent with the natural tendency (in our culture) to read left to right. If one of the buttons is a default button, highlight it and make it the first button in the set.

These dozen principles aren't only ones, but they give an idea of what's involved in designing a GUI. The challenge is to convey the proper information in an uncomplicated, straightforward, intuitive visual context.

4.5.2 Design of Matrix GUI window

Figure 4.12 shows GUI window for performing matrix operations. The user can enter values of in this window and can obtain result. The GUI window is designed using UML specifications. The components are shown inside the window frame represented with a rectangle. The elliptical components outside the window represent the use cases. The use cases in UML represent how the user interacts with the system. Each use case represents one type of interaction with the user. This is generally implemented in the program using event handlers. In the diagram the use cases are connected to the components of the window with a solid line. This connection represents that the use case affects that component. Use cases are very important in the GUI design because these things represent how users interact with the software. In the diagram the use case connected to the edit boxes of matrix sizes is retrieving the matrix size information from the user.

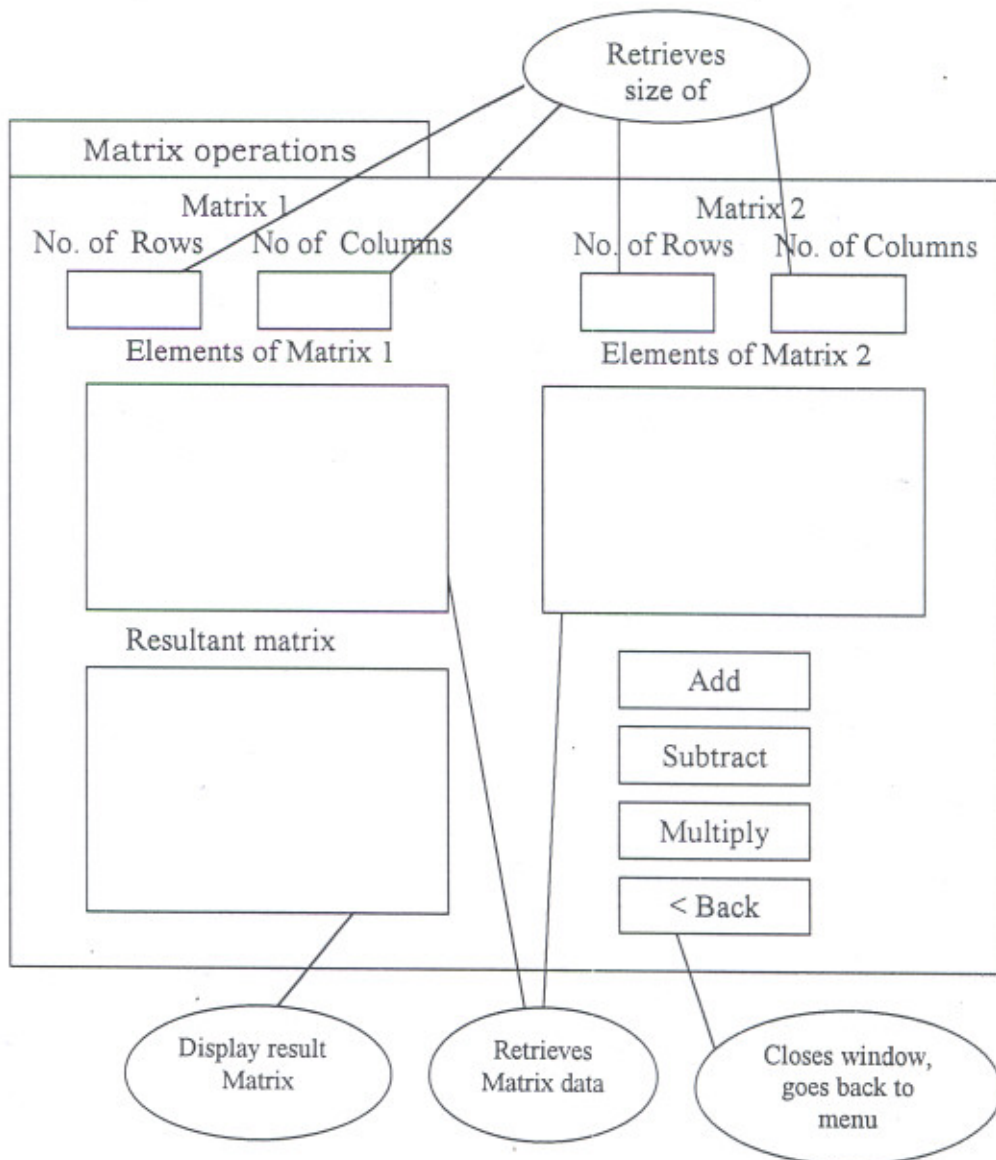


Fig 4.12 Matrix operations GUI window

Fig 4.13 shows a class diagram for matrix window. The class diagram shows the sequence of events during the running of the program. The class diagram shows how users interact with the GUI window. This diagram also shows the aggregations. The diagram tells that edit boxes, tables, and buttons are components of the matrix window. The user first enters the data in the edit box of the window. The users

in the edit box by the user the tables are shown on the window to accept the matrix elements. After entering the elements in the tables the user press button. On clicking the button the program calculates the result and displays the same in the result table. The component that affects with the event handler is represented with dotted lines.

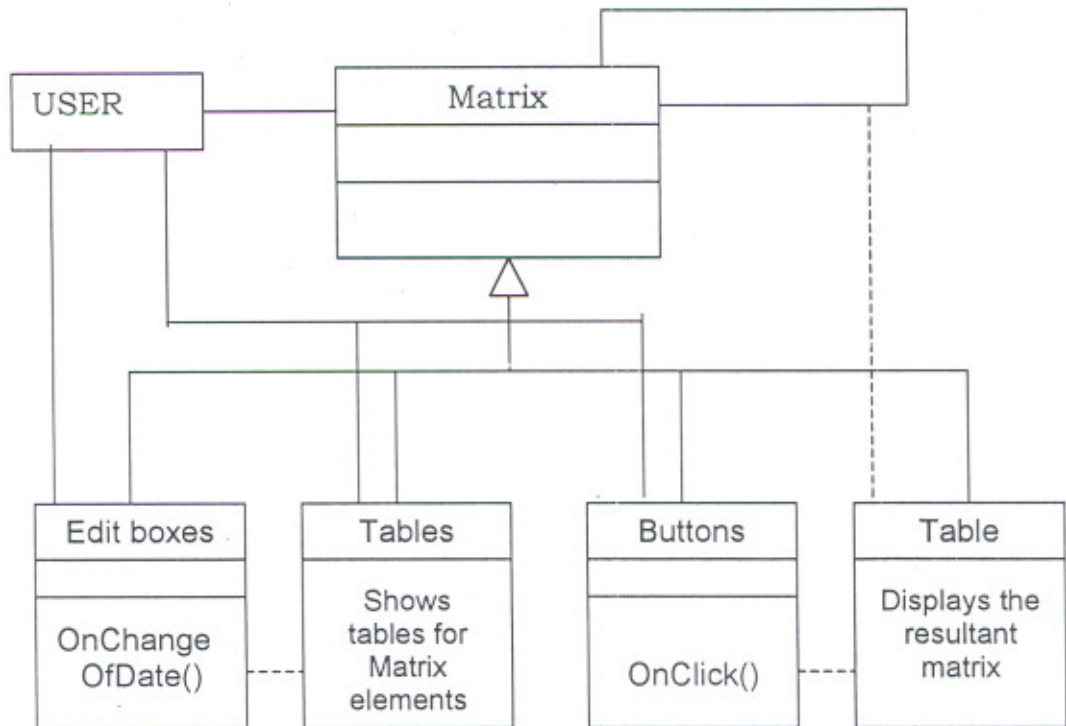


Fig.4.13 Class diagram for Matrix GUI window

The design of the remaining nine GUI programs are designed using the UML in the same way that of the matrix GUI window. Design steps of all these windows are not shown in the report as the same procedure is adopted for the design. However the screens are shown in chapters 5 and 6.

Chapter 5**DEVELOPMENT USING VC++6**

5.1 Introduction to VC++6**5.1.1 C++**

C++ is an object-oriented programming language (OOPL). C++ is a superset of 'C'. C++ is most popular OOPL, which can easily manage complexity of large software systems. Number of C++ users are doubling every 7.5 to 9 months in the software industry. By knowing its importance the C++ is introduced as basic programming language in all diploma and degree courses as a core subject. It is also introduced as a separate subject to the XIIth class students in the CBSE syllabus.

5.1.2 Features of C++ as object-oriented language

C++ is genuine modification of C language itself rather than an extension. The principal change is the introduction of new genuine primitive data type 'class'. It has no high-level data types or primitives and relies, as with C, on libraries to provide such extensions. Thus, new types are defined in the language itself. The main inspirations for the language came from C and from Simula. C++ is designed with both portability and efficiency in mind. It emphasizes the goal of being able to incorporate the large quantity of existing C code and strives to be a superset of C, although some incompatibilities exist. C++ is a compromise between object-oriented ideal and pragmatism.

C++ supports abstraction, inheritance, self-reference and dynamic binding. Both static and dynamic typing are supported. Class declarations are divided into a public and a private part. Ordinary C structures are regarded as classes where every method is public. In a class the private data and methods are only available to the object's

or function. Methods in C++ are variables ('private member variables' in the terminology) store data. Data can be stored at the class level, in a limited way, by allocating store when the program is loaded. Thus such data are fixed for all instances of the class.

One novel feature of C++ is the ability to give a function access privileges to the private parts of several classes of which it is not a member. Declaring the function to be a 'friend' of these classes does this. This can be used to simplify the description of an operation on two types. For example, if we wish to define multiplication of vectors by matrices, defining multiplication as a friend of these classes allows their implementation to be accessed. Friend functions need an extra argument to point at the object to which the operation is applied. Friends support the facility of being able to call C++ functions from normal C programs. This feature needs to be handled with great care if encapsulation is not to be compromised. C++ also provide facility of operator overloading a form of *ad hoc* polymorphism.

Inheritance is implemented though the definition of subclasses. In C++ subclasses are referred to as 'derived classes'. Derived classes inherit all the properties of the base class in terms of which they are defined as public. Complete hierarchies are possible since a base class may itself be a derived class. Inherited method may be overridden.

By default, binding is established at compile time, as in conventional language, either as a global variable or one scoped within a program block. Alternatively the object can be created in the free store and created and destroyed under program control. With these techniques the types of arguments of a function must be known at compile time. Dynamic binding is implemented by virtual functions defined in the base class that can be redefined in the derived classes.

C++ gives low-level control over the hardware when required and the benefits of object-oriented programming when required. The worst things that can be said about C++ are, first, that it is a very difficult language to use well and, second, that many programmers continue to use it as simply a 'better C'. It is one of the most successful and widely used object-oriented programming languages in commercial applications on workstations and PCs.

5.1.3 Microsoft VC++6

Microsoft Visual C++ has always been one of the most comprehensive and sophisticated software development environments available. It provides a high level of programming power and convenience, while offering a diverse set of tools designed to suit almost every programming style. At present it is one of the best language for development of object-oriented applications.

The Microsoft Developer Studio is an integrated application that serves as the primary interface for accessing the development tools provided by Microsoft Visual C++. It facilitates the user to design program resources (such as menus, dialog boxes, and icons), and even to generate some of the basic program source code (by means of wizards). The user can build and debug user programs or applications directly within the Developer Studio. User can also examine and manage program symbols and C++ classes, and user can access the VC++ online help, using commands on the Help menu.

Microsoft Foundation Classes (MFC) is an extensive C++ class library designed for creating Windows GUI programs. The MFC simplifies writing these programs, and it provides many high-level features that can save considerable coding effort. The use MFC is one of the major advantages in application development in VC++, which saves lot of time for the developer.

5.2 Numerical Library

The classes designed to solve all numerical problems are put together in numerical library. This is a static linked library.

5.2.1 Library in VC++6

A library is a collection of source code or compiled code that can be reused in programs. In C++ libraries are two types.

1. static linked library
2. dynamically linked library

A static linked library is a software module, which can be exclusively used by Visual C++ programmers. A programmer can use any part of source code in their programs as plug in components. In statically linked library the part of the library program is bound directly in programs executable file. Where as in dynamically linked library the part of the code in DLL file will called by the users program during run time.

5.2.2 Implementation of Numerical library

In VC++6 the static linked library consists of two types of files for each class. They are

1. Declaration file or header file (with 'h' extension)
2. Definition file or implementation file or source file (with 'cpp' extension)

The file name for both header and source file should be same for each class. The header file declares attributes and operations used in that class. Variable type and default values can be shown in the header file itself. All functions or operations used by the class should be declared in the header files. A variable declared in a class is known as member variable and function defined in a class is known as member function.

The implementation file consists of definition or code for all operations used in that class.

The numerical library consists of following classes

1. CMatrix
2. CLinearEquation
3. CnonLinearEquation
4. CInterpolation
5. CIntegration
6. CDifferentiation
7. CExpression
8. CLinearProgramming
9. CExtrapolation
10. CstatisticalMethods

Each class has two files with viz., source file with 'cpp' extension and header file with 'h' extension. For example CMatrix class has two files with file names Matrix.h and Matrix.cpp.

5.2.3 Creation of a class

For creation of each class, variables or attributes are using UML class diagrams shown in chapter four with class icons. The variable declaration needs prefixing of access modifiers and variable type. The attributes are generally classified as private, public and protected. Even though the C++ language does not enforce that the variables of same access level should be grouped together in a class declaration, it is a general practice to declare attributes of same access level as a group. In C++ access level is private by default. The member variables are prefixed with 'm_'. In the CMatrix class the variables m_rows, m_cols are private member variables. The following program example shows header file for CMatrix.

```
// header file of CMatrix
// Matrix.h
```

```

class CMatrix
{
public:
    // constructor
    CMatrix(int row, int col);
    // destructor
    ~CMatrix();
    // function to get index
    // operator overloading
    double & operator()(int,int);
    // operation that performs addition
    // references are used in this function for arguments
    void AddMatrices(matrix &, matrix &);
    void SubtractMatrices(matrix &, matrix &);
    void MultiplyMatrices(matrix &, matrix &);
    void TransposeMatrix(matrix &);
    void InverseMatrix(matrix &);
    double Determinant(matrix &);
    void setMatrix(Matrix m);
    // inline functions
    void setElement(double e, int r, int c)
    {
        m_element[r*m_cols+c] = element;
    }
    double getElement(int x, int y)
    {
        return m_element[x*m_cols+y];
    }
private:
    int m_rows,m_cols;
    double *m_element;
};

```

Operations in a class are prefixed with return type, class name scope operator in declaration. The arguments of the operation are suffixed to the name of operation with parenthesis. For example in CMatrix class the operation getElement returns a double number and accept the row and column integer numbers as parameters. This is expressed as

```
Public double CMatrix::getElement(int x, int y);
```

The access modifier is prefixed at the beginning as shown in the above example.

The operations may also include constructors and destructors. The constructor in C++ should have the same name of the class. The destructor also has the same name of the class with prefix 'tilde' ('~'). For example the constructor and destructors of CMatrix class are CMatrix() and ~Cmatrix(). The declaration of constructors and destructors is done in the header file. The definition of the constructors and destructors is done in the implementation file. However, inline constructors and destructors can be defined in the header file itself.

C++ permits the use of inline functions. The definition of inline function is done in the header file itself. Inline functions are allowed if the function consists of few statements in its body. The Matrix.h file consists of two inline functions getElement() and setElement(). It can be observed that the function body is written in the header file itself.

The source code of CMatrix class is shown below.

```
// matrix.cpp
// Implementation file for matrix functions
#include "StdAfx.h"
#include "matrix.h"
#include <math.h>

CMatrix::matrix(int row = 5,int col = 5)           // constructor
```

```

{
    // allocation of memory space
    if(!(m_element = new double [row*col]))
    {
        AfxMessageBox("Memory allocation error");
        return;
    }
    m_rows = row;
    m_cols = col;
}

CMatrix::~matrix()           // destructor
{
    delete m_element;        // releasing memory space
}

// function to get the index of matrix
double & CMatrix::operator()(int x, int y)
{
    return m_element[x * m_rows + y];
}

void CMatrix::AddMatrices(matrix & m1, matrix & m2)
{
    if((m1.m_rows != m2.m_rows)|| (m1.m_cols != m2.m_cols))
    {
        AfxMessageBox("Size of matrices should be same for
        addition");
        return;
    }
    for(int i = 0; i < m1.m_rows; i++)
        for( int j = 0; j < m1.m_cols; j++)
            m_element[i*m_rows+j]=m1(i,j)+m2(i,j);
}

void CMatrix::SubtractMatrices(matrix & m1, matrix & m2)
{
    if((m1.m_rows != m2.m_rows)|| (m1.m_cols != m2.m_cols))
    {

```

```

        AfxMessageBox("Size of matrices should be same for
        subtraction");
        return;
    }
    for(int i = 0;i<m1.m_rows;i++)
        for( int j = 0;j<m1.m_cols;j++)
            m_element[i*m_rows+j]=m1(i,j)-m2(i,j);
}

void CMatrix::MultiplyMatrices(matrix & m1, matrix & m2)
{
    if((m1.m_cols != m2.m_rows))
    {
        AfxMessageBox("No.of columns of 1st matrix must be equal to
        No of rows of 2nd matrix for multiplication");
        return;
    }
    for(int i = 0;i<m1.m_rows;i++)
        for( int j = 0;j<m2.m_cols;j++)
        {
            m_element[i*m_rows+j]=0;
            for(int k=0;k<m1.m_cols;k++)
                m_element[i*m_rows+j]+=m1(i,k)*m2(k,j);
        }
}

void CMatrix::TransposeMatrix(matrix & m1)
{
    for(int i = 0;i<m1.m_rows;i++)
        for( int j = 0;j<m1.m_cols;j++)
            m_element[i*m_rows+j]=m1(j,i);
}

double CMatrix::Determinant(matrix & m1)
{
    int i,j,k;
    double temp,large;
    for(k=0;k<m_rows;k++)

```

```

        // process of partial pivoting
        // finding largest element
        large = fabs(m1(k,k));
        int index = k;
        for(j=k+1;j<m_rows;j++)
            if(fabs(m1(j,k))>large)
            {
                large = fabs(m1(j,k));
                index = j;
            }

        // interchanging rows
        if(index != k)
            for(j=k;j<=m_rows;j++)
            {
                temp = m1(k,j);
                m1(k,j)=m1(index,j);
                m1(index,j) = temp;
            }

        // elimination process
        for(j=k+1; j<m_rows;j++)
        {
            temp = m1(j,k)/m1(k,k);
            for(i=k;i<m_rows;i++)
                m1(j,i) -=temp*m1(k,i);
        }
    }
    temp = 1;
    for(i = 0;i<m_rows;i++)
        temp = temp*m1(i,i);
    return temp;
}

void CMatrix::InverseMatrix(matrix & m1)
{
    if((m1.m_rows != m1.m_cols))
    {

```

```

        AfxMessageBox("No. of rows and columns should be same for
        Inversion");
        return;
    }
    matrix m(m1.m_rows,2*m1.m_cols);

    for(int i = 0;i<m1.m_rows;i++)
    {
        for( int j = 0;j<m1.m_cols;j++)
            m(i,j)=m1(i,j);
        for(j=m1.m_cols;j<m.m_cols;j++)
            if(i==j-m1.m_cols)
                m(i,j)=1.0f;
            else
                m(i,j)=0.0f;
    }
    for(i=0;i<m1.m_cols;i++)
    {
        // performing partial pivoting
        double large = fabs(m(i,i));
        int index = 0;
        for(int j=i+1;j<m1.m_cols;j++)
            if(fabs(m(j,i))>large)
            {
                large = fabs(m(j,i));
                index = j;
            }
        if((index != i) &&(index != 0))
            for(j=0;j<m.m_cols;j++)
            {
                double t = m(index,j);
                m(index,j)=m(i,j);
                m(i,j)=t;
            }
        double p = m(i,i);
        // making diagonal elements of current row
    }

```

```

// Exception handling for singular Matrix
try
{
    for(j=0;j<m.m_cols;j++)
        if(m(i,i)=0)
            throw "Matrix is Singular";
        else
            m(i,j)/=p;
}
catch(char * ErrorMessage)
{
    AfxMessageBox(ErrorMessage);
}
// making the other elements zero
for(j=0;j<m1.m_cols;j++)
    for(int k =i+1;k<m.m_cols;k++)
        if(j!=i) m(j,k)-=m(i,k) * m(j,i);
}
for(i = 0;i<m1.m_rows;i++)
    for(int j=0;j<m1.m_cols;j++)
        m_element[i*m_cols+j] = m(i,j+m1.m_cols);
}

```

In CMatrix class encapsulation or data hiding is provided by declaring attributes matrix element, no of rows and no of columns as private. The outside programs access this data by using public operations getElement() and setElement().

The CMatrix class also uses operator overloading. The operator '()' is overloaded to get element of a matrix from outside programs. The overloaded operator permits access of a matrix element by using a(4,3) format. Where a is matrix, 4 is row index and 3 is column index. Hence a(4,3) refers an element of matrix a in fifth row and third column. Operator overloading is a very good feature in c++.

C++ allows use of references. When references are used in a function definition, the parameters passed to the called function are used as it is. Whereas a function without using references makes a copy of parameters passed to the called function. This is a very good feature in C++. The advantage of this is it saves memory space and it directly modifies the variable of the calling function. In the CMatrix class references are used in AddMatrices, SubtractMatrices and MultiplyMatrices operations.

Error checking and exception handling is also provided in Matrix.cpp program. Addmatrices and SubtractMatrices operations in the above program check whether the size of two matrices to be added or subtracted is same or not. If the size of two matrices is not same the program displays error message the size of two matrices must be same for addition or subtraction. Exception handling is provided by using try and catch block. Using the try catch block does the Singular matrix exception in InverseMatirx operation.

5.3 Development of GUI

5.3.1 VC++6 for GUI Development

Microsoft VC++6 provides several pathways for writing windows GUI programs. They are:

1. Win 32 Application Programming Interface (API), which is a part of Windows '95 and Windows NT operating systems.
2. Microsoft Foundation Classes (MFC), which provides a large collection of prewritten classes, as well as supporting code, which handle many window programming tasks such as creating windows and processing messages.
3. MFC Application Wizard, which generate basic source files for a variety of different types of GUI programs. The class wizard tool is useful to generate much of routine code

required to derive classes, to define member functions for processing messages or customizing the behavior of the MFC, to manage dialog boxes and to accomplish other tasks. The code generated by the wizards makes full use of the MFC. Wizards can produce programs containing extensive collections of advanced features, including tool bars, status windows, context sensitive online help, OLE support, database access, and complete menus with partially or fully functioned commands for opening and saving files, printing, print previewing, and performing other tasks.

The 3rd approach is quite beneficial than other pathways. In this approach the source code is generated by using MFC and handles much routine programming tasks. The MFC and wizards helps a programmer by reducing much effort required in creating visual interface. It also helps to ensure that this interface conforms to Microsoft guidelines.

5.3.2 Controls in MFC

MFC provides so many controls for GUI programming, in addition to windows and dialog boxes. The controls are displayed; as icons in controls tool bar. The user can drag and drop these controls into dialog box. These controls can be placed anywhere in the dialog box and their sizes can also be changed with the help of mouse. Each control has properties, which can be set or modified by the programmer. Changing the properties the programmer can change shapes labels of the controls.

In addition to the controls MFC also supports creation of Active X controls. The VC++6 itself has many Active X controls which can be used directly in the users programs as plug in components.

In this package GUI front-end is provided for each type of problem. For example matrix operations are having two front-end programs. One program performs addition, subtraction, and multiplication. Second program performs inverse, transpose and determinant. The matrix GUI program is developed using dialog box. Number of controls is added to the dialog box for creating user interface.

Matrix arithmetic

Input Matrix 1
 No. of Rows: 4 No. of Columns: 4

	1	2	3	4	5
1	3	5	2	7.5	
2	1.2	2	4	5	
3	2	1	5	3	
4	1	2	3	4	
5					

Input Matrix 2
 No. of Rows: 4 No. of Columns: 5

	1	2	3	4	5
1	1	2	3	1	2
2	3	1	2	3	1
3	2	3	1	2	3
4	1	2	3	5	4
5					

PRODUCT MATRIX

	1	2	3	4	5
1	29.50	32.00	43.50	50.50	20.20
2	20.20	26.40	26.60	40.20	13.00
3	16.00	26.00	22.00	30.00	17.00
4	17.00	27.00	22.00	33.00	29.00
5					

Press ONE Button to see Result after entering Data

Add
 Subtract
 Multiply

<< Back

Fig 5.1 Matrix dialog box

The matrix dialog box consists of following controls.

1. *Edit box*: An edit box is used to accept data from the user. It accepts all primitive data types of c++ in addition to String data type. In matrix dialog box it is used to accept information regarding number of rows, columns and value of elements.

2. *Static text*: This control displays text on the dialog box. In the matrix dialog box No. of rows, No. of columns etc. are shown by using this static text.
3. *Push button*: This control is activated when user presses this button. This button control is generally used in event handling. In matrix dialog box buttons are used to perform addition, subtraction and multiplication operations. When user presses a button it calls the concerned operation from the library. After completing the calculation the results are displayed in the resultant matrix.
4. *Active X control*: VC++6 enables programmers to incorporate the Active X control into their program and use it just as they use standard VC++6 controls. Active X control can be placed in programs dialog boxes. The programmer can set the properties of Active X controls and attach code to the events of Active X controls. Active X control is file with the extension 'OCX'. In the matrix dialog box MS Flex Grid is used to display the resultant matrix.

All the components in the dialog boxes are given prefixed tab order. The user can select any control in the dialog box with the help of mouse or with the help of tab switch in the keyboard. Initially the cursor appears on the edit box with label No. of Rows. The user can move the cursor from that position by pressing tab switch. The user can navigate up to the last control (push button with label <Back) of the dialog box with the help of tab switch.

The program will take care of many errors in the data entered by the user. For example if the user enters number of rows more than 5 or less than 1 the program prompts the user that this variable must be in between 1 to 5. If the user keeps an edit box blank, the program will not allow the user to navigate to the next control of the dialog box.

The following steps explain the usage of Matrix GUI program.

1. On clicking the matrix-addition menu Item of the opening menu, the matrix GUI program will be activated and get into running state.
2. When user enters No. of rows and No. of columns of Matrix 1, the number of cells of the Matrix 1 will be automatically adjusted according to the no. of rows and columns.
3. When user enters No. of rows and No. of columns of Matrix 2, the number of cells of the Matrix 2 will be automatically adjusted according to the no. of rows and columns.
4. The user has to enter data in the each cell of the two matrices.
5. After entering data for two matrices the user has to press a button according to the matrix operation that the user intended to perform. When user presses Add button the resultant SUM matrix will be displayed on the MS Flex Grid Active X control. The program also displays a label 'SUM MATRIX' on the top of the Active X control.

5.4 Opening Menu

This is a dialog based windows program created using the MFC application wizard (exe) option of the VC++6. At the start of this package this program is activated first. The user can interact with this package using menu items of this program. It consists of pull down menus and context sensitive (pop-up menus). The user can activate any GUI application by clicking menu item of that program. The user can get context sensitive menu onto the screen by clicking right mouse button on the screen. User can also activate any GUI program using pop-up menu also. The screen of the opening menu is figure 5.2

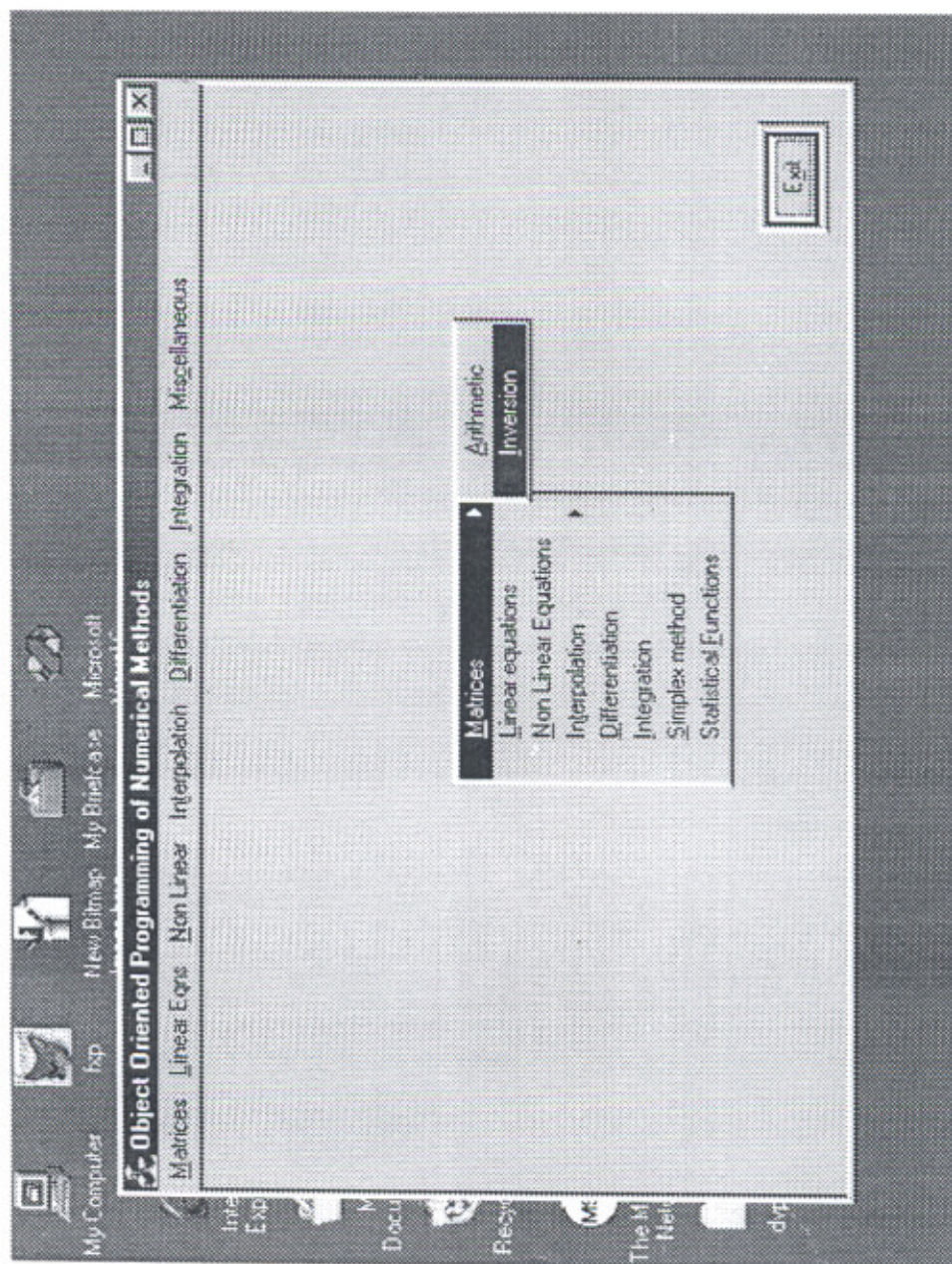


Fig. 5.2 Opening Menu

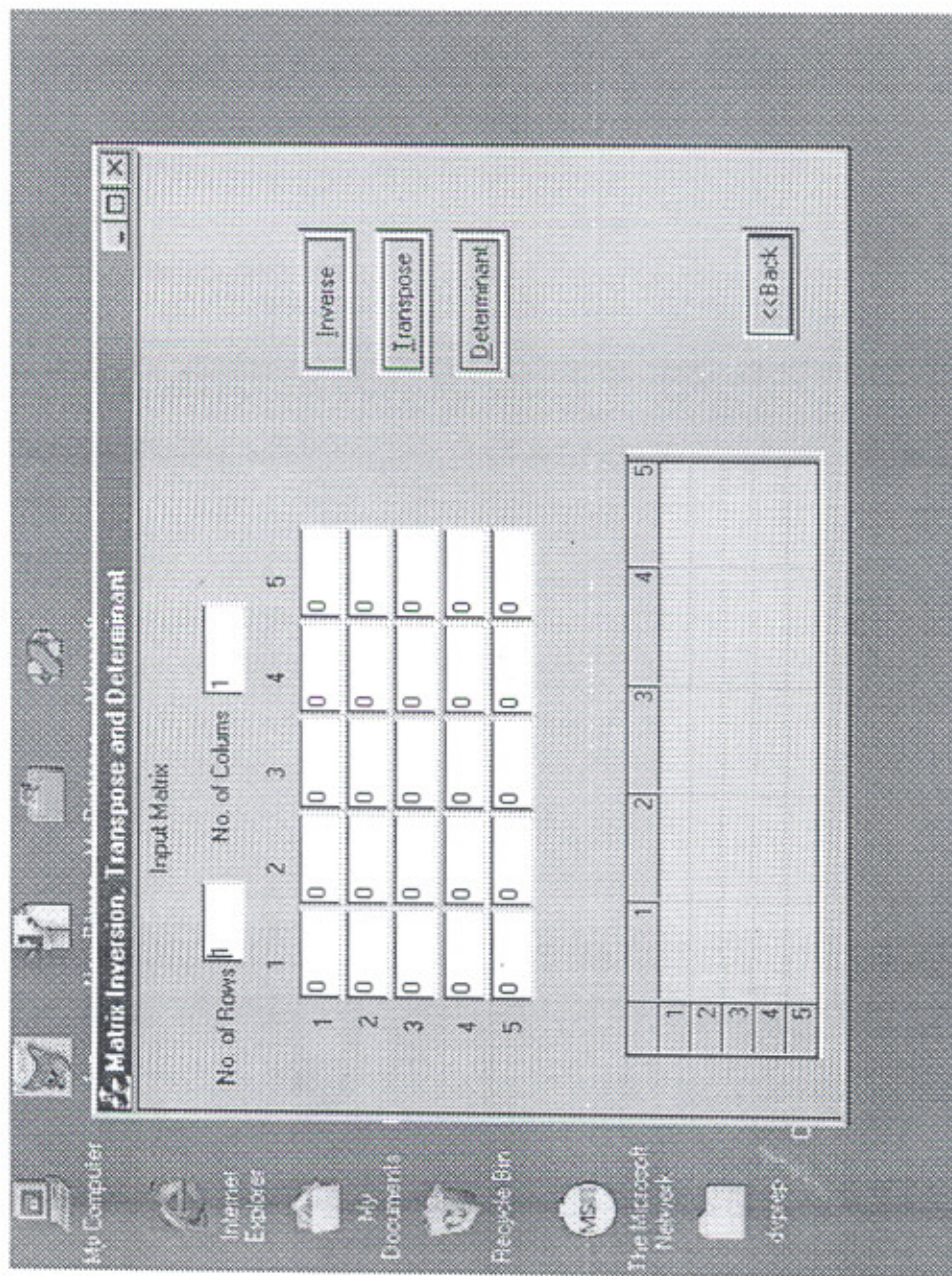


Fig 5.3 Matrix Inversion Dialog box

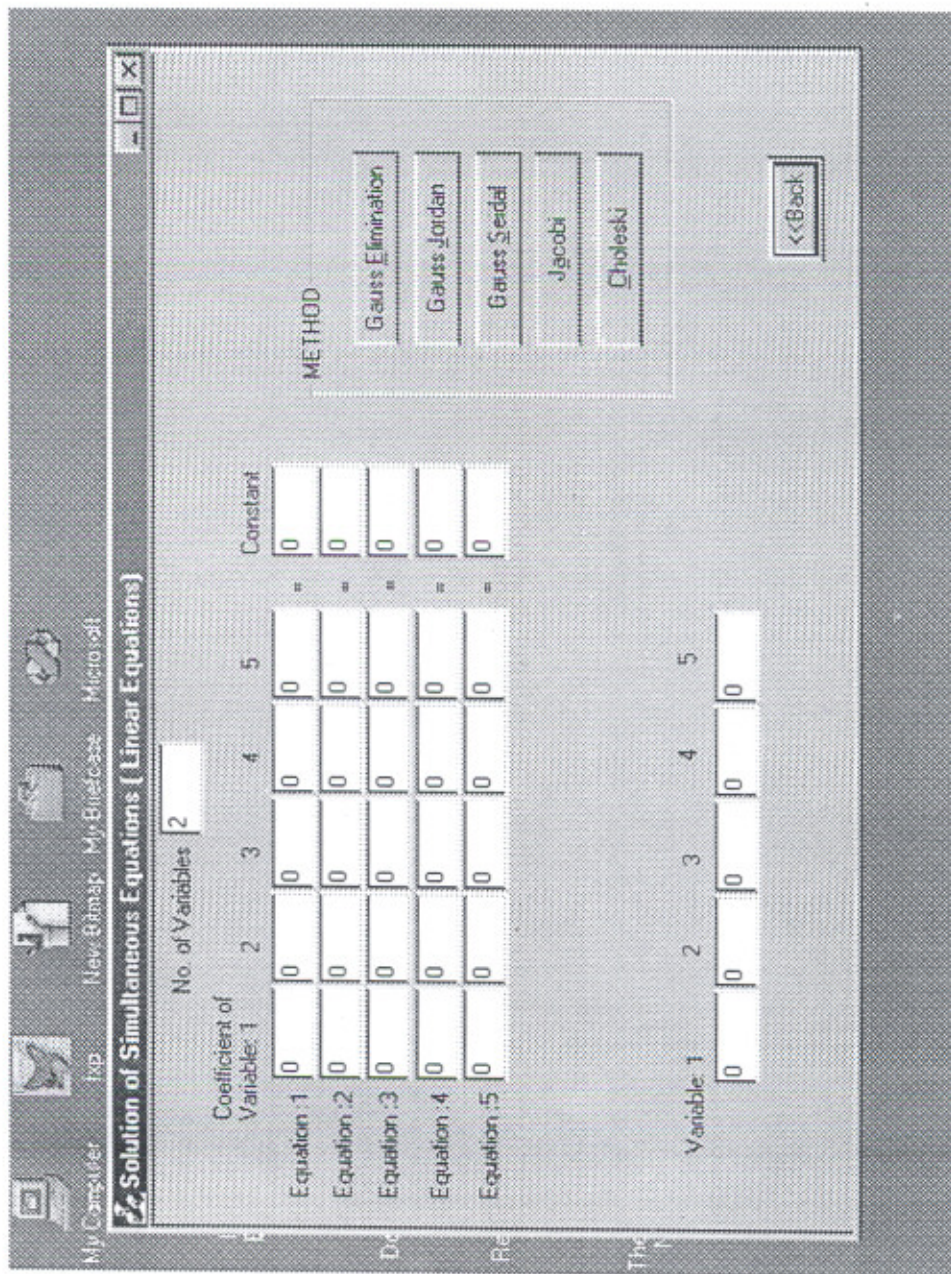


Fig 5.4 Simultaneous Equation Dialog Box

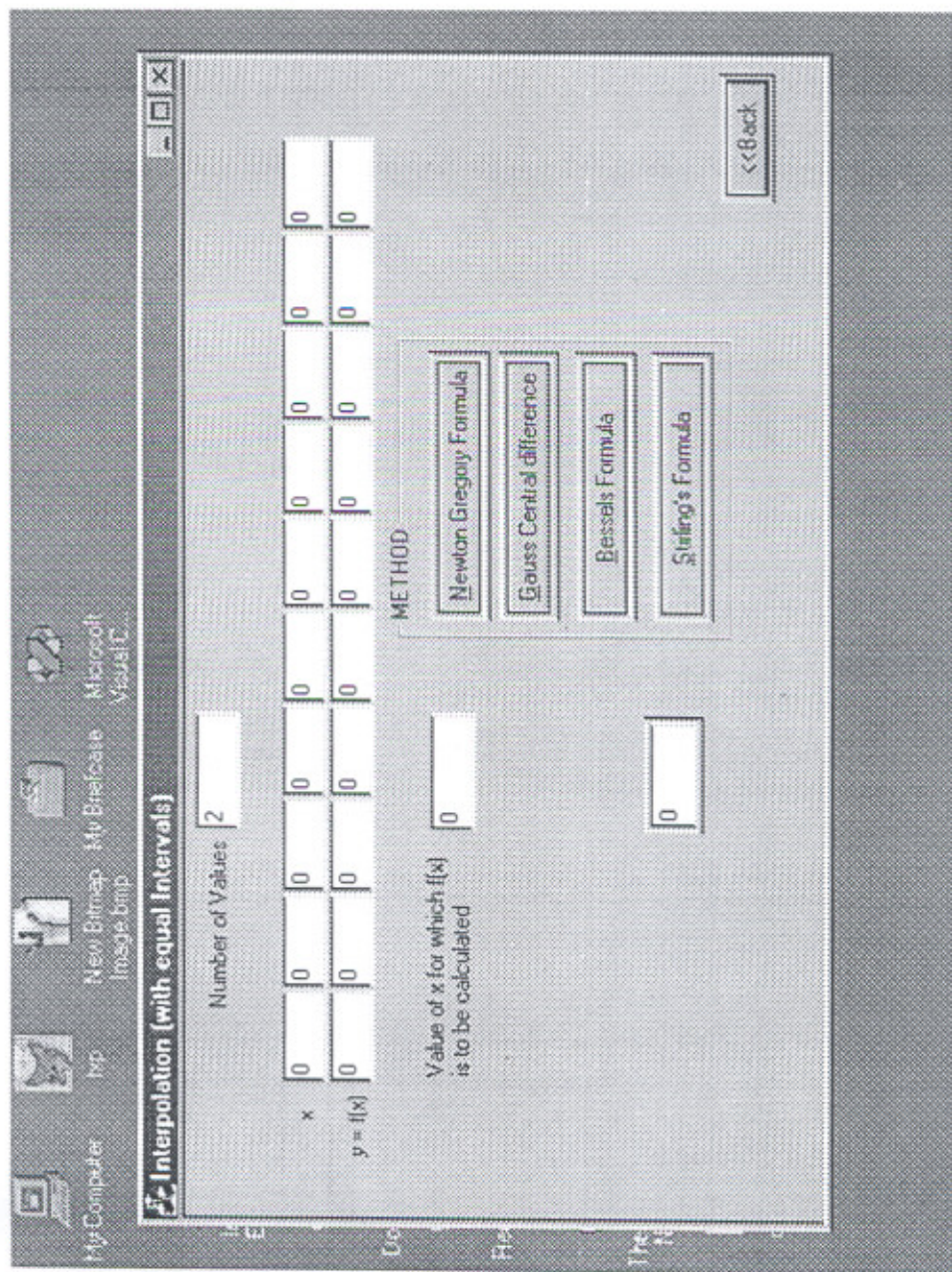


Fig 5.5 Interpolation Dialog Box

Solution of a Linear Programming Problem in Simplex Method

No. of Decision Variables: No. of Slack Variables:

Coefficient of Decision Variable: 1	2	3	4	5	Constant
Constraint Equation 1	0	0	0	0	0
Equation 2	0	0	0	0	0
Equation 3	0	0	0	0	0
Equation 4	0	0	0	0	0
Equation 5	0	0	0	0	0

Coefficient of Objective Function:

SOLUTION:

Variable 1: Optimal Value:

Fig 5.6 Solution of Linear programming problem Dialog box

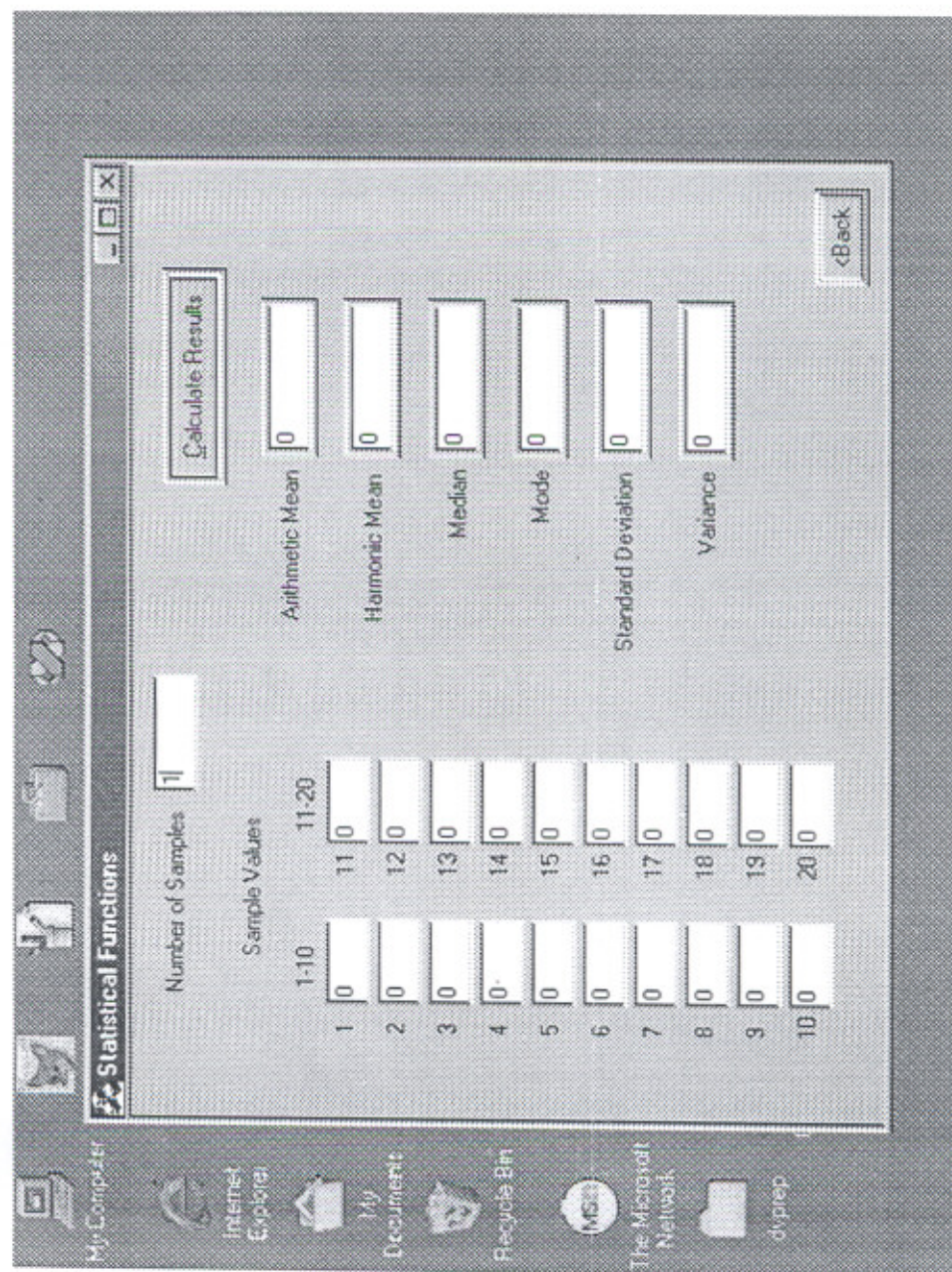


Fig. 5.7 Statistical methods Dialog Box

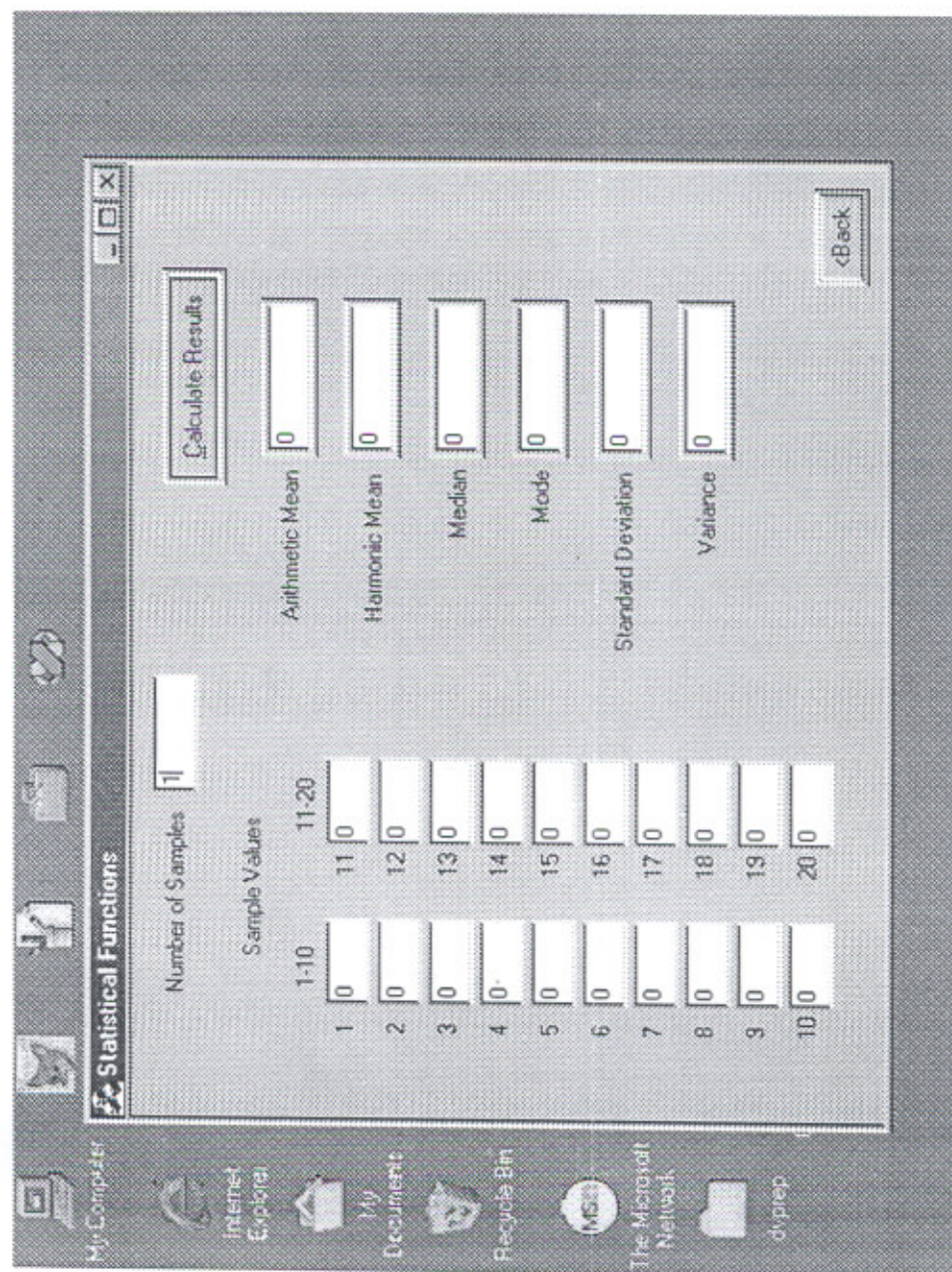


Fig. 5.7 Statistical methods Dialog Box

Chapter 6

DEVELOPMENT USING JAVA2

6.1 Introduction to Java 2

Java is a proprietary product of Sun Micro Systems. It consists of two things: a programming language and a platform.

6.1.1 The Java Programming Language

The Java programming language is designed to meet the challenges of application development in the context of heterogeneous network-wide distributed environments. Paramount among these challenges is secure delivery of applications that consume the minimum of system resources, can run on any hardware and software platform and can be extended dynamically.

6.1.1.1 Features of Java 2

It is a high-level programming language and has following features.

1. *Simple*: Java omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit. These omitted features primarily consist of operator overloading (although the Java language does have method overloading), multiple inheritance. Another aspect of being simple is being small. One of the goals of Java is to enable the construction of software that can run stand-alone in small machines. The Java interpreter and standard libraries have a small footprint. A small size is important for use in embedded systems and so Java can be easily downloaded over the net.
2. *Architecture-neutral*: Java was designed to support applications on networks. In general, networks are composed of a variety of systems with a variety of CPU and operating systems architectures. To enable a Java

application to execute anywhere on the network, the compiler generates an architecture-neutral object file format, the compiled code is executable on many processors, given the presence of Java runtime system.

3. *Object-Oriented*: This is, unfortunately, one of the most overused buzzwords in the industry. But object-oriented design is very powerful because it facilitates the clean definition of interfaces and makes it possible to provide reusable “software ICs”. The object oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.
4. *Portable*: The libraries that are a part of the system define portable interfaces. For example, there is an abstract window class and implementations of it for Unix, Windows NT/95, and the Macintosh. The Java system itself is quite portable. The compiler is written in Java and the runtime is written in ANSI C with a clean portability boundary. The portability boundary is essentially a POSIX subset.
5. *High-performance*: the bytecode format was designed with generating machine codes in mind, so the actual process of generating machine code is generally simple. Efficient code is produced because the compiler does automatic register allocation and some optimization when it produces the bytecodes.
6. *Interpreted*: Java bytecodes are translated on the fly to native machine instructions (interpreted) and not stored anywhere. And since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory. As a part of the bytecodes stream, more compile-time information is carried over and available at runtime. This is what the linker’s type checks are based on it also makes programs more amenable to debugging.

7. *Multithreaded*: Java has a sophisticated set of synchronization primitives that are based on the widely used monitor and condition variable paradigm. By integrating these concepts into the language (rather than only in classes) they become much easier to use and are more robust. Other benefits of multithreading are better interactive responsiveness and real-time behavior. This is limited, however, by the underlying platform: stand-alone Java runtime environments have good real-time behavior. Running on top of other systems like Unix, Windows, the Macintosh, or Windows NT limits the real-time responsiveness to that of underlying system.
8. *Robust*: Java is intended for writing programs that must be reliable in a variety of ways. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error prone. The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data. Instead of pointer arithmetic, Java has true arrays. This allows subscript checking to be performed in addition, it is not possible to turn an arbitrary integer into a pointer by casting.
9. *Dynamic*: In a number of ways, Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment.
10. *Secure*: Java is intended for use in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems. The authentication techniques are based on public-key encryption.
11. *Network-Savvy*: Java has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and FTP. This

makes creating network connections much easier than in C or C++. Java applications can open and access objects across the net via URLs with this same ease that programmers are used to when accessing a local file system.

6.1.1.2 Bytecodes

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler one can translate a Java program into an intermediate language called Java bytecodes—the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once. Interpretation occurs each time the program is executed. This figure illustrates how this works. We can think of Java bytecodes as the machine code instructions for the Java virtual machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware. Java bytecodes help make “write once, run anywhere” possible. You can compile your Java program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

6.1.2 The Java Platform

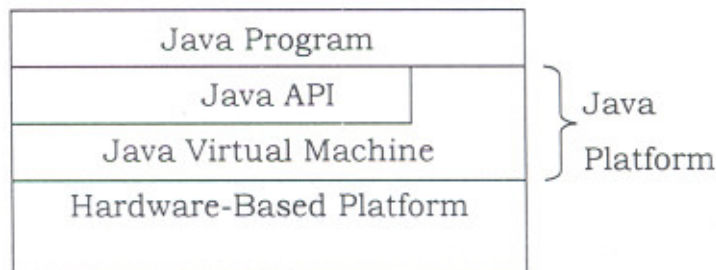
The Java™ platform is a fundamentally new way of computing, based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices. A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

- The Java Virtual Machine
- The Java Application Programming interface (Java API)

Java Virtual Machine is the base for the Java platform and is ported onto various hardware-based platforms. The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual machine insulates the Java program from hardware dependencies.



As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java's performance close to that of native code without threatening portability.

6.2 Numerical package

6.2.1 Packages in Java 2

A package is collection of classes that are related in purpose, in scope, or by inheritance. Packages are useful for several broad reasons:

1. They allow the organization of classes into units.

2. They reduce problems with conflicts in names. Packages allow the hiding of classes so that conflicts can be avoided.
3. They protect classes, variables and methods in larger ways
4. They can be used to identify classes according to their function.

6.2.2 Development of numerical package

The numerical package consists of Ten classes viz., Matrix, LinearEquations, NonLinearEquations, Interpolation, Differentiation, Integration, Expression, LinearProgramming, Extrapolation, and Statistical Methods.

Each class is stored with a file name same as class name with 'java' extension. This file will be converted to class file on compilation. This class file can be used by the programmers to create objects and makes use of methods of that class. A programmer can make use of numerical package by writing `import numerical.Matrix` to make use of Matrix class. The class files of all classes are to be stored in the directory with the same name of the package. As an example program for solution of linear equations is shown below.

```

/* LinearEquations.java
 * SOLUTION OF SIMULTANEOUS LINEAR EQUATIONS
 *          Gauss elimination method
 *          Gauss Jordan method
 *          Gauss Seidal method
 *          Jacobi method
 *          Choleski method (Triangular Factorisation method)
 */
package numerical;
import numerical.Matrix;
public class LinearEquations extends Matrix{
    int noOfVariables;
    double solution[];
    // constructor

```

```

public LinearEquations(int n){
    super(n,n+1);
    noOfVariables = n;
    solution = new double[n];
}

public double getSolution(int r){
    return solution[r];
}

public void gaussElimination(){
    int i,j,k;
    double p,large,loop_large,t;
    // Elimination process
    for(k=0;k<noOfVariables;k++){
        // process of partial pivoting
        large = (element[k][k]<0) ? -element[k][k] : element[k][k];
        int index = k;
        for(j=k+1;j<noOfVariables;j++){
            loop_large = (element[j][k]<0) ? -element[j][k] : element[j][k];
            if(loop_large>large){
                large = loop_large;
                index = j;
            }
        }
        if(index != k)
            for(j=k;j<noOfVariables;j++){
                t = element[index][j];
                element[index][j]=element[k][j];
                element[k][j]=t;
            }
    }

    for(j=k+1;j<noOfVariables;j++){
        p = element[j][k]/element[k][k];
        for(i=k;i<noOfVariables;i++){
            element[j][i]-=p*element[k][i];
        }
    }

    // finding solution
    for(i=noOfVariables-1;i>=0;i--) {
        p = element[i][noOfVariables];
        if(i!=noOfVariables-1)
            for(j=i+1;j<noOfVariables;j++)

```

```

        p=element[i][j]*solution[j];
        solution[i]=p/element[i][i];
    }
}

public void gaussJordan(){
    int i,j,k;
    double p,large,loop_large,t;
    // Elimination process
    for(k=0;k<noOfVariables;k++){
        // process of partial pivoting
        large = (element[k][k]<0) ? -element[k][k] : element[k][k];
        int index = k;
        for(j=k+1;j<noOfVariables;j++){
            // finding pivotal element
            loop_large = (element[j][k]<0) ? -element[j][k] : element[j][k];
            if(loop_large>large){
                large = loop_large;
                index = j;
            }
            // interchanging rows
            if(index != k)
                for(j=k;j<=noOfVariables;j++){
                    t = element[index][j];
                    element[index][j]=element[k][j];
                    element[k][j]=t;
                }
        }
        // making diagonal element 1
        t= element[k][k];
        for(j=0;j<=noOfVariables;j++){
            element[k][j] = element[k][j]/t;
        }
        // making elements of lower and upper triangle 0
        for(i=0;i<noOfVariables;i++){
            if(i!=k){
                p = element[i][k];
                for(j=k;j<=noOfVariables;j++){
                    element[i][j]-=p*element[k][j];
                }
            }
        }
        // finding solution
    }
}

```

```

        for(i=noOfVariables-1;i>=0;i--)
            solution[i]=element[i][noOfVariables];
    }
    public void gaussSeidal(){
        int i=noOfVariables,j,k,m=100;
        double x1,sum;
        // iterative process
        for(k=0;k<m;k++){
            // storing the value of the previous iteration
            x1=solution[i-1];
            for(i=0;i<noOfVariables;i++){
                sum = element[i][noOfVariables];
                for(j=0;j<noOfVariables;j++){
                    if(i != j) sum-=element[i][j]*solution[j];
                }
                solution[i]= sum/element[i][i];
            }
            //terminating iterative process
            double error = solution[i-1]-x1;
            double abs_error = (error<0) ? -error : error;
            if((abs_error)<0.00001)
                break;
        }
        if(k>=100){
            for(i=0;i<noOfVariables;i++) solution[i]=0;
        }
    }
    public void jacobi(){
        int i=noOfVariables,j,k,m=100;
        double sum;
        double[] x = new double[noOfVariables];
        // iterative process
        for(k=0;k<m;k++){
            // storing the value of the previous iteration
            for(i=0;i<noOfVariables;i++)
                x[i]=solution[i];
            for(i=0;i<noOfVariables;i++){
                sum = element[i][noOfVariables];
                for(j=0;j<noOfVariables;j++){
                    if(i != j) sum-=element[i][j]*solution[j];
                }
                solution[i]= sum/element[i][i];
            }
        }
    }

```

```

    }
    //terminating iterative process
    double error = solution[i-1]-x[i-1];
    double abs_error = (error<0) ? -error : error;
    if((abs_error)<0.00001) break;
}
if(k>=100){
    for(i=0;i<noOfVariables;i++)
        solution[i]=0;
}
}
public void choleski(){
    double p;
    int i,j,k;
    for(i=1;i<noOfVariables;i++){
        for(j=0;j<=i-1;j++){
            p = element[i][j];
            if(j!=0)
                for(k=0;k<=j-1;k++)
                    p = p-element[i][k]*element[k][j];
            element[i][j] = p/element[j][j];
        }
        for(j=i;j<=noOfVariables;j++){
            p= element[i][j];
            for(k=0;k<=i-1;k++)
                p-=element[i][k]*element[k][j];
            element[i][j] = p;
        }
    }
    // finding solution
    for(i=noOfVariables-1;i>=0;i--) {
        p = element[i][noOfVariables];
        if(i!=noOfVariables)
            for(j=i+1;j<noOfVariables;j++)
                p-=element[i][j]*solution[j];
        solution[i] = p/element[i][i];
    }
}
}
}

```

In the above program the package statement means that this package belongs to the numerical package. The statement `public class LinearEquations extends Matrix` indicates that the `LinearEquations` class is derived from `Matrix` class (inheritance). Here `import numerical. Matrix` permits use of `Matrix` class in this program.

6.2.3 Differences between C++ and Java

Java is simpler and easy to use than C++. The concepts of object-oriented programming can be efficiently learned in Java than C++. The length of Java program is approximately 30% of its counter part in C++. The following table shows features of Java and C++ languages.

Parameter	C++	Java
Type checking/binding	Late/early	early
Polymorphism	yes	yes
Information hiding	yes	yes
Concurrency	poor	good
Inheritance	yes	yes
Multiple inheritance	yes	no
Garbage collection	no	yes
Persistence	no	yes
Genericity	yes (templates)	no
Object libraries	yes	yes

6.3 Development of GUI programs

Java 2 has many tools for GUI and applet programming. Abstract Windowing Tool kit (AWT) and swing objects in JFC (Java Foundation Classes) facilitates different objects to the user for GUI development. The GUI programs are developed in this software using awt and swing objects. One front-end GUI program is developed for every problem. This software contains 10 such GUI programs. The figure shows GUI

In the above program the package statement means that this package belongs to the numerical package. The statement `public class LinearEquations extends Matrix` indicates that the `LinearEquations` class is derived from `Matrix` class (inheritance). Here `import numerical. Matrix` permits use of `Matrix` class in this program.

6.2.3 Differences between C++ and Java

Java is simpler and easy to use than C++. The concepts of object-oriented programming can be efficiently learned in Java than C++. The length of Java program is approximately 30% of its counter part in C++. The following table shows features of Java and C++ languages.

Parameter	C++	Java
Type checking/binding	Late/early	early
Polymorphism	yes	yes
Information hiding	yes	yes
Concurrency	poor	good
Inheritance	yes	yes
Multiple inheritance	yes	no
Garbage collection	no	yes
Persistence	no	yes
Genericity	yes (templates)	no
Object libraries	yes	yes

6.3 Development of GUI programs

Java 2 has many tools for GUI and applet programming. Abstract Windowing Tool kit (AWT) and swing objects in JFC (Java Foundation Classes) facilitates different objects to the user for GUI development. The GUI programs are developed in this software using awt and swing objects. One front-end GUI program is developed for every problem. This software contains 10 such GUI programs. The figure shows GUI

Solution of Linear Equations

No of Variables
4

Accept no of variables

Coefficient Matrix

1	3	2.6	4.8
9	7	3.4	1.2
2.6	3.7	1.908	3.2
2	4.3	5.8	1.8

Constants

24
45
67
34

Solutions:

-44.9...	84.73...	-35.4...	-19.4...
----------	----------	----------	----------

Gauss Elimination Method

Jacobi Method

Gauss Jordan Method

Gauss Seidal Method

EXIT

Fig 6.1 Solution of Linear Equations

Solution of Non Linear Equations

Enter Expression
 $\sin(x) - \text{pow}(x, 3) + \text{pow}(x, 2) + 5 * x + 1.98$

Initial Value 1: 10
Initial Value 2: 0
Accuracy: 0.00001

OK

The root of the Given Expression
2.946939468383789

No of Iterations
19

Secant Method
Bisection Method
False Position Method
Newton Raphson Method

EXIT

Fig. 6.2 Solution of Non Linear Equations

Interpolation

No of Values

Accept no of Values

x	f(x)
4	19
5	25
6	30
7	36
8	40
9	45
10	56

Value of x for which
f(x) is to be calculated

$f(6.5)$

Equal Intervals

Newton Gregory Formula

Gauss Central Difference

Bessel's Formula

Stirling's Formula

Unequal Intervals

Newton Divided Difference

Lagrange's Formula

EXIT

Fig. 6.3 Interpolation Frame

Chapter 7**CONCLUSIONS & FUTURE SCOPE**

7.1 Conclusions

This software package achieved the objective of implementing numerical methods using an object-oriented approach. The software consists of class library developed in both VC++6 and Java2. The class library consists of classes for several numerical methods, statistical methods and simplex method in operation research. The class libraries are quite useful to the students at various levels (diploma, degree students at undergraduate and postgraduate courses). The programmer can also make use of these classes in their programs for solving scientific and engineering problems requiring numerical methods. VC++6 programs can use the static library in VC++6 by including concerned header files of numerical library in their programs. Java2 programmers can use the numerical package by importing the concerned class in to their programs. Java2 programmers can make use of the classes either in applications or applets. As C++ and Java are two object-oriented languages the class libraries in these two languages will be quite useful to most of the software developers (more than 80% of the software in the world is developed using these two languages). This software package consists of following components.

1. Class library for numerical methods, statistical methods, simplex method of operation research, expression parsing. (static library in VC++6 and package in Java2)
2. GUI programs for all classes in the library to be used as front-end tools.
3. Opening menu GUI program for accessing each program in this software package.

The main advantage of this software is getting the solution of real life

knowledge of numerical methods and programming. The GUI programs are users friendly and any novice user can make use of this package. This package will be quite useful for students who are learning the numerical methods. The students can very easily compare results in different methods very easily with out any effort. If the data is entered once on the GUI screen a simple click of button will give the results in any method.

The software in VC++6 is a collection of 'exe' files. These exe files run on any system with Windows '95 or Windows NT or Windows '98 operating systems. No other software is required to run this software.

The size of software in VC++6 is approximately 1 MB. The software in Java2 is a collection of 'class' files. Java2 platform is required to run this software. The size of software in Java2 is approximately 100 KB.

7.2 Future scope

Developments in the software technology are taking place at a explosive speed. Each day new techniques, new packages, new versions are reported. Even though this software can serve the purpose of solving scientific and engineering problems requiring numerical methods, some of the following enhancements make the usage of this software more effective. The additional features to be added are:

1. Graphical display of results, which makes analysis easy.
2. Addition of more numerical methods, statistical methods, and methods of operation research.
3. Providing more exception handling such that software is more immune to errors.
4. Thorough testing of the software at different situations.
5. Adding the features required for commercial software, such as documentation, online help, and deployment or installation facilities.

BIBLIOGRAPHY

1. Bjorn Stroustrup, The C++ programming language, Addison Wesley Publishing Company.
2. Dr. B.S.Grewal, Numerical methods for Engineering and Science, Khanna Publishers.
3. Chris Corry, COM / DCOM Primer Plus, Techmedia.
4. David Bennet, Visual C++ Developer's Guide, BPB publishing company
5. Erwin Kreyszig, Advanced Engineering Mathematics, Wiley Eastern Publishing Company.
6. Grady Booch, Object-Oriented Analysis and Design with Applications, Benjamin Cummings.
7. Herbert Schildt C++ The Complete Reference, Tata Mc.Graw Hill.
8. H.M. Anita Numerical Methods for Scientists and Engineers, Tata Mc. Graw Hill.
9. Ian Graham, Object-Oriented methods, Addison Wesley Publishing Company.
10. Micheal J. Young, Mastering VC++ 6., BPB publishing company.
11. M.K. Jain, Numerical Methods for Scientists and Engineers, SBW Publishers.
12. Robert Sedgewick, Algorithms in C++, Addison Wesley publishing company.
13. V. RajaRaman, Computer Oriented Numerical Methods, Prentice-Hall India.
14. William Press, Numerical Recipes in C, Cambridge University Press.

BIBLIOGRAPHY

1. Bjorn Stroustrup, The C++ programming language, Addison Wesley Publishing Company.
2. Dr. B.S.Grewal, Numerical methods for Engineering and Science, Khanna Publishers.
3. Chris Corry, COM / DCOM Primer Plus, Techmedia.
4. David Bennet, Visual C++ Developer's Guide, BPB publishing company
5. Erwin Kreyszig, Advanced Engineering Mathematics, Wiley Eastern Publishing Company.
6. Grady Booch, Object-Oriented Analysis and Design with Applications, Benjamin Cummings.
7. Herbert Schildt C++ The Complete Reference, Tata Mc.Graw Hill.
8. H.M. Anita Numerical Methods for Scientists and Engineers, Tata Mc. Graw Hill.
9. Ian Graham, Object-Oriented methods, Addison Wesley Publishing Company.
10. Micheal J. Young, Mastering VC++ 6., BPB publishing company.
11. M.K. Jain, Numerical Methods for Scientists and Engineers, SBW Publishers.
12. Robert Sedgewick, Algorithms in C++, Addison Wesley publishing company.
13. V. RajaRaman, Computer Oriented Numerical Methods, Prentice-Hall India.
14. William Press, Numerical Recipes in C, Cambridge University Press.

15. Simon, Richard J., Michael Gonker, and Brian Barnes, Windows '95 WIN32 programming API Bible", the Waite group
16. Joseph schmuller, Teach yourself UML in 24 hours, TechMedia
17. Herbert Shildt, and patrick Naughton, Java2 Complete reference, Tata Mc. Graw Hill.
18. Laura Lemay and Charles L. Perkins, Teach yourself Java 1.1 in 21 days, sams publishing, USA.
19. Mitchell Waite, object-oriented programming in Java, Techmedia.
20. Simon Roberst, Philip Heller, Michael Earnest, Java 2 Certification Guide, BPB publishers.
21. Michael Morrison, Java Unleashed, Sams publishing, USA.
22. Mike Cohn, Java Developers Reference, Sams publishing, USA.
23. Nathan Gurewich and Ori Gurewich, Teach yourself VC++5 in 21 days, Techmedia.
24. Mickey williams, Teach yourself VC++6 in 24 Hours VC++6 in 24 hours, Techmedia.

