Mid Term Syllabus

Computational Physics

Prof. Dr. Saeed Ahmad Buzdar

Chairman, Department of Physics

Mid Term Syllabus

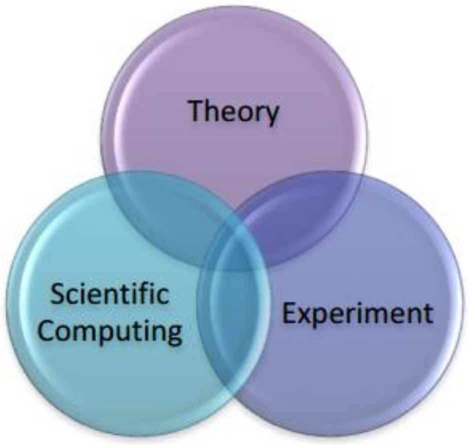
Introduction to Scientific Computing

Most problem solving in science and engineering uses scientific computing. A scientist might devise a system of differential equations to model a physical system, then use a computer to calculate their solutions. An engineer might develop a formula to predict cost as a function of several variables, then use a computer to find the combination of variables that minimizes that cost. A scientist or engineer needs to know science or engineering to make the models. He or she needs the principles of scientific computing to find out what the models predict. Scientific computing is challenging partly because it draws on many parts of mathematics and computer science. Beyond this knowledge, it also takes discipline and practice. A problem-solving code is built and tested procedure by procedure. Algorithms and program design are chosen based on considerations of accuracy, stability, robustness, and performance. Modern software development tools include programming environments and debuggers, visualization, profiling, and performance tools, and high-quality libraries. The training, as opposed to just teaching, is in integrating all the knowledge and the tools and the habits to create high quality computing software \solutions."

One common theme is the need to understand what is happening \under the hood" in order to understand the accuracy and performance of our computations. We should understand how computer arithmetic works so we know which operations are likely to be accurate and which are not. To write fast code, we should know that adding is much faster if the numbers are in cache, that there is overhead in getting memory (using new in C++ or malloc in C), and that printing to the screen has even more overhead. It isn't that we should not use dynamic memory or print statements, but using them in the wrong way can make a code much slower. State of-the-art eigenvalue software will not produce accurate eigenvalues if the problem is ill-conditioned. If it uses dense matrix methods, the running time will scale as $n3$ for an $n \times n$ matrix. Doing the exercises also should give the student a feel for numbers. The exercises are calibrated so that the student will get a feel for run time by waiting for a run to finish (a moving target given hardware advances). Many exercises ask the student to comment on the sizes of numbers. We should have a feeling for whether $4{:}5 \times 10\text{-}6$ is a plausible roundoff error if the operands are of the order of magnitude of 50. Is it plausible to compute the inverse of an $n \times n$ matrix if $n = 500$ or $n = 5000$? How accurate is the answer likely to be? Is there enough memory? Will it take more than ten seconds? Is it likely that a Monte Carlo computation with $N = 1000$ samples gives $:1\%$ accuracy?

What is Scientific Computing?

Scientific Computing (SC) is a broad, multidisciplinary area that encompasses applications in science, engineering, mathematics, and computer science. SC makes use of the techniques of applied mathematics and computer science for the solution of scientific and engineering problems. Therefore, SC is nowadays regarded as a "**third pillar**" of science, along with **theory** and **experiment** in the advancement of scientific knowledge and engineering practice.



Modern scientists increasingly rely on computational modelling and data analysis to explore and understand the natural world. Given the ubiquitous use in science and its critical importance to the future of science and engineering, computational modelling plays a central role in progress and scientific developments in the 21st Century.

AIM OF SCIENTIFIC COMPUTING

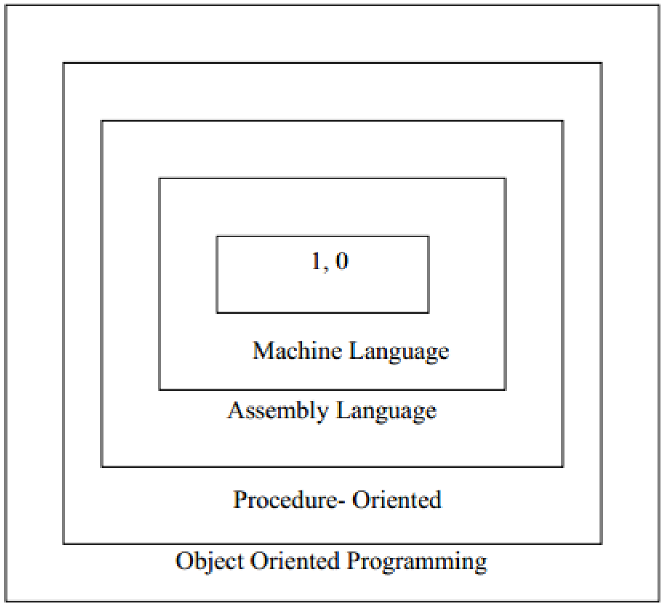
The aim of Scientific Computing Program is:

- **To design** state-of-the-art mathematical and computational models and algorithms;
- **To train** graduates from different disciplines with the aim to develop and apply their skills to the solution of real-life problems from science, engineering, industry;
- **To develop** collaboration with scientist elsewhere by building a comprehensive and international research platform, to support academic and technological exchange and advancement;
- **To conduct** fundamental and frontier research with advanced computational approaches, thereby talents worldwide and train highly qualified research personal, to support scientific development in Pakistan.

Software Evaluation

Ernest Tello, A well known writer in the field of artificial intelligence, compared the evolution of software technology to the growth of the tree. Like a tree, the software evolution has had distinct phases “layers” of growth. These layers were building up one by one over the last five decades as shown in fig., with each layer representing and improvement over the previous one. However, the analogy fails if we consider the life of these layers. In software system each of the layers continues to be functional, whereas in the case of trees, only the uppermost layer is functional

Alan Kay, one of the promoters of the object-oriented paradigm and the principal designer of Smalltalk, has said: “*As complexity increases, architecture dominates the basic materials*”. To build today’s complex software it is just not enough to put together a sequence of programming statements and sets of procedures and modules; we need to incorporate sound construction techniques and program structures that are easy to comprehend implement and modify. With the advent of languages such as c, structured programming became very popular and was the main technique of the 1980’s. Structured programming was a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired result in terms of bug-free, easy-to- maintain, and reusable programs.



Object Oriented Programming (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. However, not all languages are suitable to implement the OOP concepts easily.

Procedure-Oriented Programming

In the procedure-oriented approach, the problem is viewed as the sequence of things to be done such as reading, calculating and printing such as cobol, fortran and c. The primary focus is on functions. A typical structure for procedural programming is shown in fig.1.2. The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.

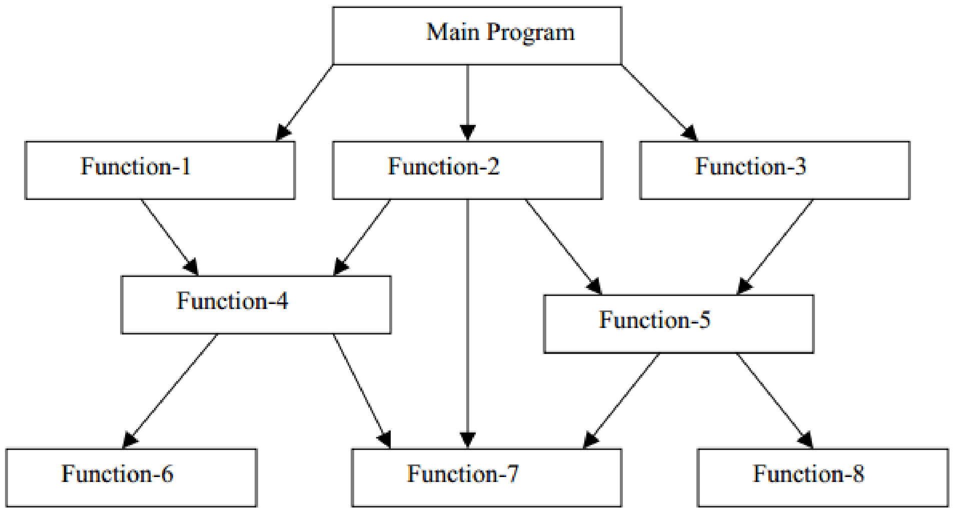


Fig. Typical structure of procedural oriented programs

Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as *functions*. We normally use flowcharts to organize these actions and represent the flow of control from one action to another.

In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

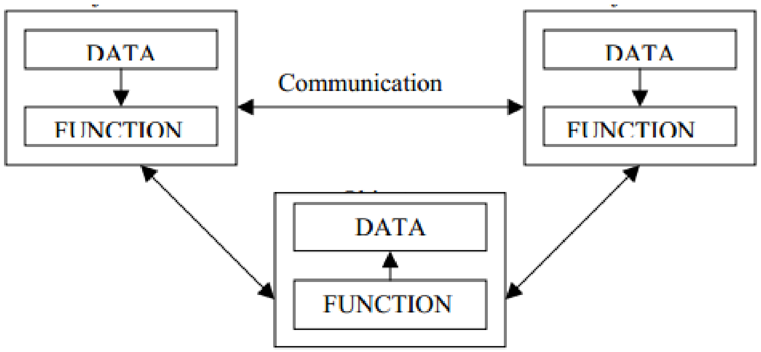
Another serious drawback with the procedural approach is that we do not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the element of the problem.

Some Characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design

Object Oriented Paradigm

The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function. OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects. The organization of data and function in object-oriented programs is shown in fig. The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.



Some of the features of object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are ties together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people.

Basic Concepts of Object Oriented Programming

It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism

- Dynamic binding
- Message passing

Benefits of OOP

OOP offers several benefits to both the program designer and the user. Object Orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

While it is possible to incorporate all these features in an object-oriented system, their importance depends on the type of the project and the preference of the programmer. There are a number of issues that need to be tackled to reap some of the benefits stated above. For instance, object libraries must be available for reuse. The technology is still developing and current product may be superseded quickly. Strict controls and protocols need to be developed if reuse is not to be compromised.

Object Oriented Language

Object-oriented programming is not the right of any particular languages. Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. However, programming becomes clumsy and may generate confusion when the programs grow large. A language that is specially id designed to support the OOP concepts makes it easier to implement them.

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

a) Object-based programming languages, and
b) Object-oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major feature that are required for object based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

Application of OOP

OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas. The most popular application of object-oriented programming, up to now, has been in the area of user interface design such as window. Hundreds of windowing systems have been developed, using the OOP techniques.

Real-business system are often much more complex and contain many more objects with complicated attributes and method. OOP is useful in these types of application because it can simplify a complex problem. The promising areas of application of OOP include:

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

The object-oriented paradigm sprang from the language, has matured into design, and has recently moved into analysis. It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software system but also its productivity. Object-oriented technology is certainly going to change the way the software engineers think, analyze, design and implement future system.

INTRODUCTION TO C++

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's. Stroustrup, an admirer of Simula67 and a strong supporter of C, wanted to combine the best of both the languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. Therefore, C++ is an extension of C with a major addition of the class construct feature of Simula67. Since the class was a major addition to the original C language, Stroustrup initially called the new language 'C with classes'. However, later in 1983, the name was changed to C++. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented version of C.

C+ + is a superset of C. Almost all c programs are also C++ programs. However, there are a few minor differences that will prevent a c program to run under C++ complier. We shall see these differences later as and when they are encountered.

The most important facilities that C++ adds on to C care classes, inheritance, function overloading and operator overloading. These features enable creating of abstract data types, inherit properties from existing data types and support polymorphism, thereby making C++ a truly object-oriented language.

Application of C++

C++ is a versatile language for handling very large programs; it is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life applications systems.

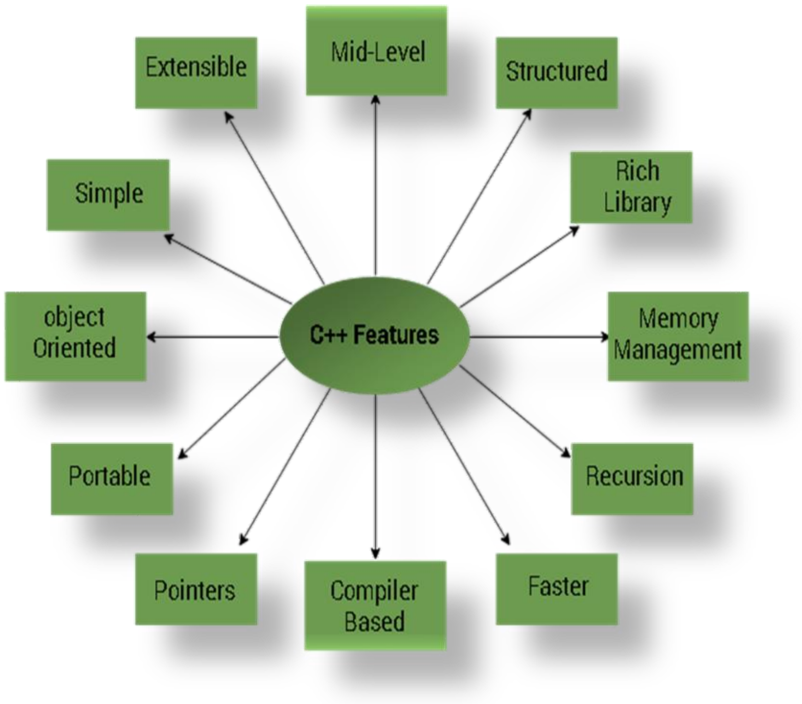
- Since C++ allow us to create hierarchy related objects, we can build special object-oriented libraries which can be used later by many programmers.
- While C++ is able to map the real-world problem properly, the C part of C++ gives the language the ability to get closed to the machine-level details.
- C++ programs are easily maintainable and expandable. When a new feature needs to be implemented, it is very easy to add to the existing structure of an object.
- It is expected that C++ will replace C as a general-purpose language in the near future.

Historical Perspective of C++

The C++ programming language was created by Bjarne Stroustrup and his team at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way. The earliest versions, which were originally referred to as "C with classes," date back to 1980. As the name C++ implies, C++ was derived from the C programming language: ++ is the increment operator in C. As early as 1989 an ANSI Committee (American National Standards Institute) was founded to standardize the C++ programming language. The aim was to have as many compiler vendors and software developers as possible agree on a unified description of the language in order to avoid the confusion caused by a variety of dialects. In 1998 the ISO (International Organization for Standardization) approved a standard for C++ (ISO/IEC 14882)

Characteristics of C++

- ❑ Convenient Language
- ❑ Well-Structure Language
- ❑ Case Sensitive
- ❑ Machine Independence
- ❑ Object Oriented
- ❑ C compatibility
- ❑ Modular programming



Basic Structure of C++

The format of writing program in C++ is called structure of C++. The structure of C++ program is very flexible. It increases power of language.

Preprocessor Directives.

Main function ()

Program Body.

Preprocessor Directives

Preprocessor directives are lines included in the code of our programs that are not program statements but directives for the preprocessor. These lines are always preceded by a pound sign (#). The preprocessor is executed before the actual compilation of code begins, therefore the preprocessor digests all these directives before any code is generated by the statements. These preprocessor directives extend only across a single line of code. As soon as a newline character is found, the preprocessor directive is considered to end. No semicolon (;) is expected at the end of a preprocessor directive.

Include

The first preprocessor directive which we will cover is one which you should already know quite a lot about! We've been using the #include directive ever since our first C++ program, but we've sort of glossed over what it does. When the preprocessor finds an 'include' directive, it fetches the file specified and dumps it in the place of the directive - so in the case of <iostream>, the preprocessor looks into the directories where it knows it might find these kind of files, and then dumps the file (e.g. iostream.h) where the directive is present.

Files specified using triangular brackets (e.g. <iostream>) will be looked for in the directories that the compiler has "noted down", and files specified using double quotes will be looked for in the same directory as your project. As such, you can create your own custom '.cpp' and '.h' files, and include these using double quotes with the include directive.

C++ Header Files

#include<conio.h> (Console input-output header)

Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

#include<math.h> (Math header)

Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.

#inlcude<iostream> (Input Output Stream

Used as a stream of Input and Output.

#include<iomanip.h> (Input-Output Manipulation)

Used to access set() and setprecision()

Syntax of Header

#include<Header file.h>

Main Function

int/void

int/void is a return value, which will be explained in a while.

main()

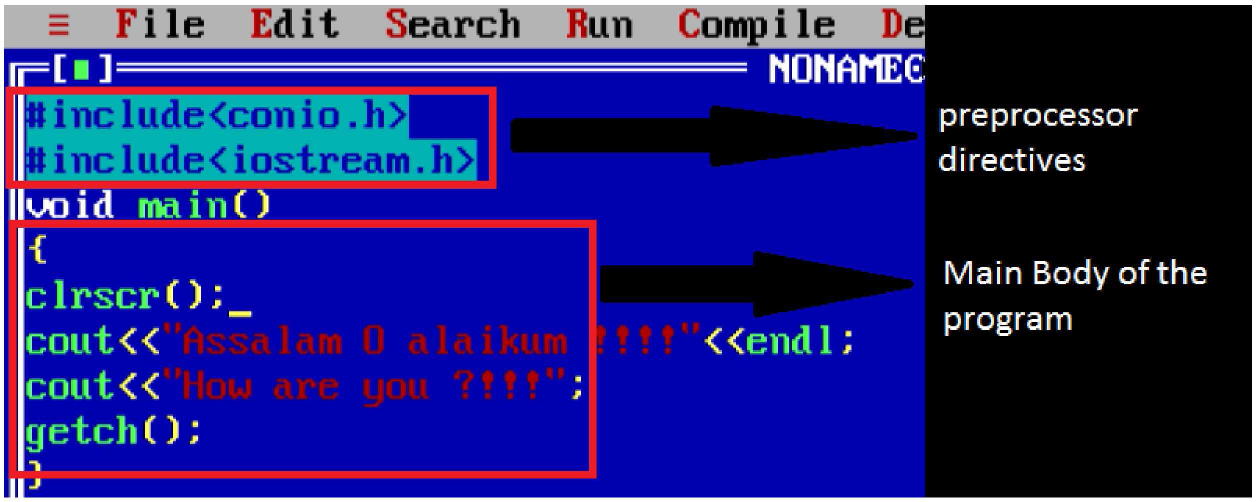
The main() is the main function where program execution begins. Every C++ program must contain only one main function.

or

This is a main function, which is the default entry point for every C++ program and the void in front of it indicates that it does not return a value.

Main Body of Program

Main body of C++ consist of statements. Start and end with the curly braces. As shown in fig.

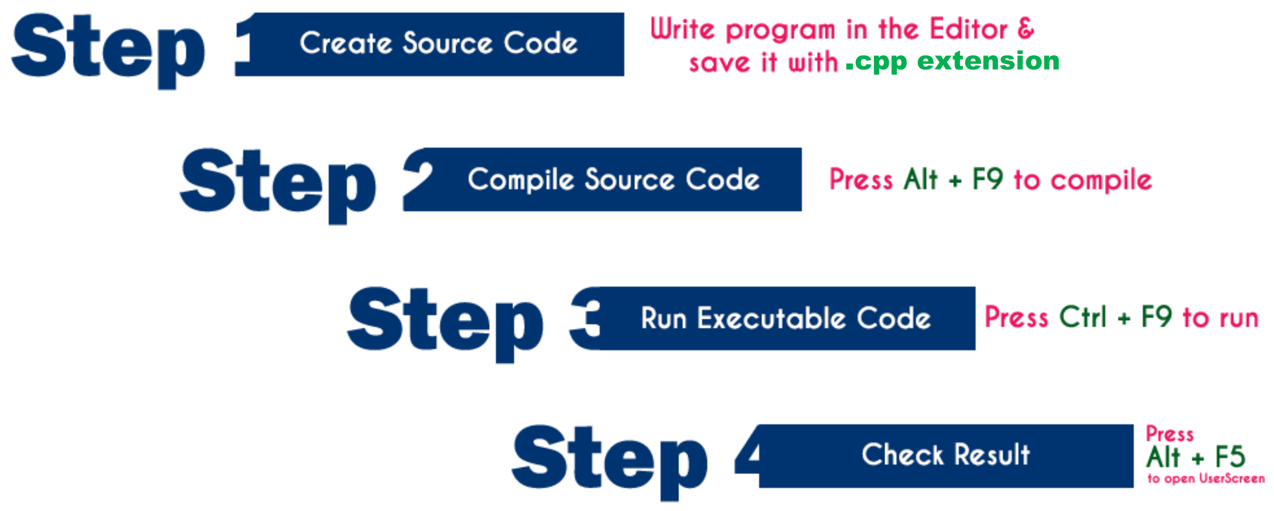


```
#include<conio.h>
#include<iostream.h>
void main()
{
clrscr();
cout<<"Assalam O alaikum !!!!"<<endl;
cout<<"How are you ?!!!";
getch();
}
```

Creating and Running C Program

Generally, the programs created using programming languages like C, C++, Java, etc., are written using a high-level language like English. But, the computer cannot understand the high-level language. It can understand only low-level language. So, the program written in the high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.

Popular programming languages like C, C++, Java, etc., use the compiler to convert high-level language instructions into low-level language instructions. A compiler is a program that converts high-level language instructions into low-level language instructions. Generally, the compiler performs two things, first it verifies the program errors, if errors are found, it returns a list of errors otherwise it converts the complete code into the low-level language. To create and execute C++ programs in the Windows Operating System, we need to install Turbo C++ software. We use the following steps to create and execute C++ programs in Windows.


Step 1 Create Source Code
Write program in the Editor & save it with .cpp extension
Step 2 Compile Source Code
Press Alt + F9 to compile
Step 3 Run Executable Code
Press Ctrl + F9 to run
Step 4 Check Result
Press Alt + F5 to open UserScreen

C++ Tokens

Every C++ program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c++ program is called token. Every instruction in a C++ program is a collection of tokens. Tokens are used to construct c programs and they are said to the basic building blocks of a C++ program. In a c program tokens may contain the following.

1. Keywords
2. Identifiers
3. Operators

4. Special Symbols
5. Constants
6. Strings
7. Data values

In a C++ program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.

Debugging in Turbo C++

Error is an illegal operation performed by the user which results in abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing. The most common errors can be broadly classified as follows.

Types of errors

There are basically three types of errors that you must contend with when writing computer programs:

- Syntax errors
- Runtime errors
- Logic errors

Generally speaking, the errors become more difficult to find and fix as you move down the above list.

Comments

The program that you write should be clear not only to you, but also to the reader of your program. Part of good programming is the inclusion of comments in the program. Typically, comments can be used to identify the authors of the program, give the date when the program is written or modified, give a brief explanation of the program, and explain the meaning of key
statements in a program. In the programming examples, for the programs that we write, we will not include the date when the program is written, consistent with the standard convention for writing such books.
Comments are for the reader, not for the compiler. So when a compiler compiles a program to check for the syntax errors, it completely ignores comments. Comments are shown in green.

The program in Example 2-1 contains the following comments:

// This is a C++ program. It prints the sentence:

// Welcome to C++ Programming.

THE BRIEF INTRODUCTION TO:

- Data types
- Operator precedence
- scientific manipulators
- Transferring input from data files to programs and from programs to output files
- Basic control structures
- Multiple selection using else if
- Programming with choice statements and introduction to looping
- Repetition using while loops, increment and decrement, the use of “for” and “do-while” loops
- User-defined functions
- One-dimensional arrays—motivation

INTRODUCTION TO MATLAB

Matlab is an interactive system designed specifically for scientific computation that is used widely in academia and industry. At its core, Matlab contains an efficient, high level programming language and powerful graphical visualization tools which can be easily accessed through a development environment (that is, a graphical user interface containing various workspace and menu items). Matlab has many advantages over computer languages such as C, C++, Fortran and Java. For example, when using the Matlab programming language, essentially no declaration statements are needed for variables. In addition, Matlab has a built-in extensive mathematical function library that contains such items as simple trigonometric functions, as well as much more sophisticated tools that can be used, for example, to compute difficult integrals. Matlab also provides additional *toolboxes* that are designed to solve specific classes of problems, such as for image processing. It should be noted that there is no free lunch; because Matlab is an interpreted language, codes written in Fortran and C are usually more efficient for very large problems. (Matlab may also be compiled.) Therefore, large production codes are usually written in one of these languages, in which case supplementary packages, such as the NAG or IMSL library, or free software from *netlib*, are recommended for scientific computing. However, because it is an excellent package for developing algorithms and problem solving environments, and it can be quite efficient when used properly.

We provide a very brief introduction to Matlab. Though our discussion assumes the use of Matlab 7.0 or higher, in most cases version 6.0 or 6.5 is sufficient. There are many good sources for more complete treatments on using Matlab, both on-line, and as books. One excellent source is the *MATLAB Guide, 2nd ed.* by D.J. Higham and N.J. Higham published by SIAM Press, 2005.

Starting, Quitting, and Getting Help

The process by which you start Matlab depends on your computer system; you may need to request specific commands from your instructor or system administrator. Generally, the process is as follows:

- On a PC with a Windows operating system, double-click the \Matlab" shortcut icon on your Windows desktop. If there is no \Matlab" icon on the desktop, you may bring up DOS (on Windows XP by going to the Command Prompt in Accessories) and entering matlab at the operating system prompt. Alternatively, you may search for the \Matlab" icon in a subdirectory and click on it. Where it is to be found depends on how Matlab was installed; in the simplest case with a default installation it is found in the C:*n*$MATLAB directory, where $MATLAB is the name of the folder containing the MATLAB installation.
- On a Macintosh running OS X 10.1 or higher, there may be a \Matlab" icon on the dock. If so, then clicking this icon should start Matlab. If the \Matlab" icon is not on the dock, then you need to find where it is located. Usually it is found in /Applications/$MATLAB/, or /Applications/$MATLAB/bin/, where $MATLAB is the name of the folder containing the Matlab installation. Once you find the \Matlab" icon, double clicking on it should start Matlab.
- On Unix or Linux platforms, typically you enter Matlab at the shell prompt.

When you have been successful in getting Matlab to start, then the development tool Graphical User Interface (GUI) should appear on the screen. Although there are slight differences (such as key stroke short cuts) between platforms, in general the GUI should have the same look and feel independently of the platform.

The *command window*, where you will do much of your work, contains a prompt:

```
>>
```

We can enter data and execute commands at this prompt. One very useful command is doc, which displays the \help browser". For example, entering the command

```
>> doc matlab
```

opens the help browser to a good location for first time Matlab users to begin reading. Alternatively, you can pull down the *Help* menu, and let go on *MATLAB Help*. We recommend that you read some of the information on these help pages now, but we also recommend returning periodically to read more as you gain experience using Matlab.

Throughout this book we provide many examples using Matlab. In all cases, we encourage readers to \play along" with the examples provided. While doing so, it may be helpful at times to use the doc command to find detailed information about various Matlab commands and functions.

For example,

```
>> doc plot
```

opens the help browser, and turns to a page containing detailed information on using the built-in plot function.

To exit Matlab, you can pull down the *File* menu, and let go on or *Exit MATLAB*. Alternatively, in the command window, you can use the exit command:

```
>> exit
```

NUMERICAL METHOD ERROR AND ALGORITHM

NATURF OF SOLUTION OF A PROBLEM

Whenever we talk about the solutions of problem, the first question, which arises, is: which type of solution can we obtain? Is it an exact solution? Is it an approximate solution? Is its analytical solution? Is it an empirical solution? Is it a numerical solution? All these questions are differentiated as follows:

Analytical solutions are those that are obtained using direct methods. In such cases results are obtained in the form of expression of independent variable(s). Such a solution may be solution of exact modeling equation for a given problem or solution of modified modeling equation. Modified modeling equations are obtained approximating the exact equation in order to simplify the nature of the equation. The solution obtained in the later case is not the exact one but closer to it.

Empirical solutions are those solutions that are obtained on the basis of experience. In this case the solution function is assumed from the knowledge about the behavior of variations. Such solutions are also in the form of expressions of independent variable(s).

By the use of above methods closed form solutions of a given problem are obtained. From the closed form solution, value of solution in terms of number is obtained after substituting particular value of independent variable.

There are many problems whose solutions cannot be obtained analytically even after the approximation; we need to go for another type of method to get a solution. For such cases numerical method is only approach to get a solution.

NUMERICAL METHOD

Numerical methods are techniques that provide solution to mathematical problems in the form of numbers at different values of independent variable and not as an expression of independent variable. The solution obtained in terms of numbers are known as numerical solutions. Numerical solutions are the substitutes to analytical, empirical and other types of approximate solutions obtained in the form of expression. Further it is important to know that computed solutions, obtained by numerical methods, arc not exact mathematical solutions. Instead they yield approximate solutions differing from the exact ones by less than a specified tolerance. The precision of a numerical solution can he diminished in several subtle ways. Understanding these difficulties can often guide the practitioners of numerical methods in the proper implementation and/or development of numerical algorithms.

The inaccuracies in the given input data, upon which calculations are based, or the inaccuracies introduced in the subsequent analysis of these data cause errors in the numerical calculations. Due care has to be taken to minimize these errors while finding numerical solutions. In the process of problem solving using numerical method first of all the given mathematical model is rewritten according to our need on the basis of proper input data and adequate checks for particular type of output.

CHARACTERISTICS OF NUMERICAL COMPUTING

- Accuracy
- Efficiency
- Rate of convergence
- Numerical stability

DECIMAL NUMBER SYSTEM

Numerical methods have numbers as man material for processing. Number representation on a computer plays as important role in the accuracy of results of numerical methods. Lets discuss first the decimal representation of number.

Basics of Decimal number system:

We are familiar with decimal number system, i.e., a number system of base 10. It has ten digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. To get numbers of large or small magnitude instead of one digit, combination of more than basic digits are placed in successive positions and weights are attached to individual digits, depending on their position.

Representation of integers

The representation of the number 5386 means the value:

$5\times1000 + 3\times100 + 8\times10 + 6\times1 = 5000+300+80+6$

Representation of fraction:

The representation of .3578 means the value

$3\times10^{-1} + 5\times10^{-2} + 7\times10^{-3} + 8\times10^{-4}$

BINARY NUMBER SYSTEM

It is natural that human beings having 10 fingers and 10 toes have developed and are using a number system of base 10, i.e., decimal number system. However number systems with any other base can also be developed and used. Primary logic units of computer are either off or on states, i.e., they have two states. Hence for a computer a number system of base, 2 is a proper number system. The number system with base two is known as binary number system. it has two digits 0,1 to act as basic **BI**nary digi**TS** (BITS). To get numbers of large or small magnitude a combination of more than one bit are placed in successive positions and weights are attached to individual bits depending on the position of the bit.

Representation of integers: In binary system representation of integers, like decimal system, the rightmost bit has unit (=2°) weight and its position is identified as position zero. The second bit from the rightmost has a weight 2 ($=2^1$) and identified as position 1; the third digit from the rightmost has a weight 4 ($=2^2$) and position as 2 and so on. The binary number 1011 has 1, 1, 0, 1 in its zero, first, second and third position, respectively, and have weights 1,2,4,8. The binary representation (1011) means the value

$1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 1\times8 + 0\times4 + 1\times2 + 1\times1 = 8 + 0 + 2 + 1 = 11$ (in decimal)

Thus the decimal value of binary number 1011 is 11. Retaining bases as subscripts in the numbers representation above conversion can be written as $(1011)_2 = (11)_{10}$.

Representation of fractions: In binary system "." is used to represent existence of fractional part. It is known as binary point. Weight to the positions on the right of decimal point are 2^{-1}, 2^{-2}, 2^{-3}, etc. The number .1011 in binary has 1, 0, 1, 1 in its first, second, third, and fourth position, respectively. In the same order their weights are 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}. The binary representation of .1011 has its decimal equivalent as

$1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4} = 1/2+0/4 + 1/8 +1/16 = .5 + 0 +.125 +.0625 = .6875$ (in decimal)
Hence the decimal value of the binary number .1011 is .6875. Retaining bases as subscripts in the numbers representation, above conversion can be written as $(.1011)_2 = (.6875)_{10}$.

Thus the binary real numbers also contain both integer and fractional parts. In the computer they are represented in normalized binary floating-point notation form.

LIMITATIONS OF RH RESENTING NUMBERS IN COMPUTER When we imagine or write a number in any number system there is no restriction on the length of digits, i.e., the considered number can be of any length. We can imagine two numbers, which are close yet different from each other. There is neither an upper limit nor a lower limit on the value of number; If you have some number you can always imagine another number, present in the number system with value greater or smaller than the first number. It is according to usual considerations about numbers.

Numbers on computer: When we come to the world of computers only finite positions or finite memory locations are available for writing a number; it may vary from machine to machine yet it remains limited. This limitation of memory size available to represent a number leads to unusual and surprising properties attached to numbers in computer. These surprises are:

- Values of numbers have upper and lower limits, i.e., the number system is bounded.
- Numbers are not only discrete but also quantized for a group of permitted numbers.
- Quantum jumps in successive permitted group of numbers are different.
- Most of the input numbers are approximated during their representation on computer.

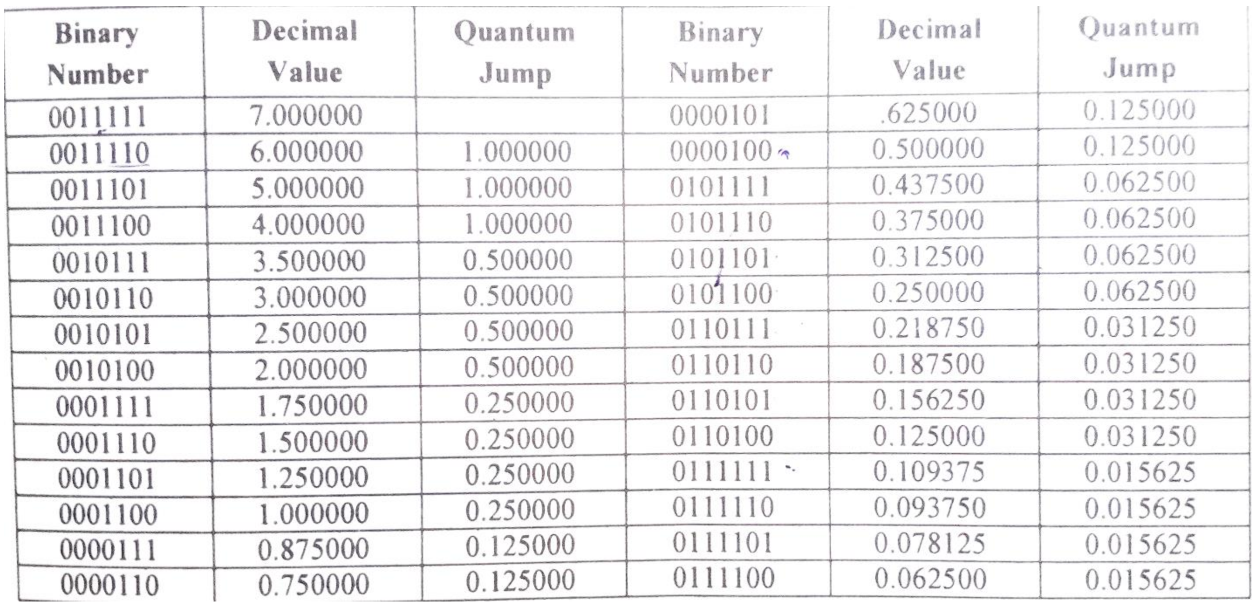
Justification of existence of limitations: To justify these limitations here we restrict to normalized floating-point representation in binary number system only as it is being widely used in computers. However, it should be noted that these surprises also exist in the representation of numbers in any other number system on computer.

Let us discuss how numbers are represented in the computer memory. The manner, in which a floating-point number is stored in a computer word length, is as follows. The word length of a computer is the number of bits, which are read and write together. There are four components in a floating-point number.

(a) Sign of the number, (b) Sign of the exponent, (c) Exponent magnitude and (d) Mantissa

In the word length first most significant bit is reserved for the sign of number, next series of a few bits for signed exponent and set of last bits for mantissa.

Consider a hypothetical machine that stores a number using 7 bits. It employs first bit for sign of a number, next three bits for sign and magnitude of exponent and last three for magnitude of mantissa (the decimal point is assumed automatically). The positive or negative sign of number depends on its first bit. Normally 0 represents the positive and 1 represents the negative number. The possible highest positive exponent is 011 (+1 1); highest negative exponent is 111 (- 11); and maximum value mantissa is 111(.111) lowest value mantissa is 100 (.100). Numbers with mantissa 010, 011, 001, 000 cannot exist in floating-point representation. The different permitted numbers with their decimal values are shown in the following table.



Binary Number	Decimal Value	Quantum Jump	Binary Number	Decimal Value	Quantum Jump
0011111	7.000000		0000101	.625000	0.125000
0011110	6.000000	1.000000	0000100	0.500000	0.125000
0011101	5.000000	1.000000	0101111	0.437500	0.062500
0011100	4.000000	1.000000	0101110	0.375000	0.062500
0010111	3.500000	0.500000	0101101	0.312500	0.062500
0010110	3.000000	0.500000	0101100	0.250000	0.062500
0010101	2.500000	0.500000	0110111	0.218750	0.031250
0010100	2.000000	0.500000	0110110	0.187500	0.031250
0001111	1.750000	0.250000	0110101	0.156250	0.031250
0001110	1.500000	0.250000	0110100	0.125000	0.031250
0001101	1.250000	0.250000	0111111	0.109375	0.015625
0001100	1.000000	0.250000	0111110	0.093750	0.015625
0000111	0.875000	0.125000	0111101	0.078125	0.015625
0000110	0.750000	0.125000	0111100	0.062500	0.015625

The highest +ve number is 0011111$(= (.111)_2 \times (10)_2^{(11)_2})$. Its equivalent decimal value is 7.

The decimal values of the permitted numbers indicate that the next number larger than 2.5 is 3 and intermittent numbers are not present. This is true not only at 2.5 but at every permitted number on the 7-bit machine. This justifies that the numbers are not only discrete but also quantized. The table also shows that the quantum jump at decimal number 4.0 is 1.0, at 2.0 is 0.5 but at 0.5 it is 0.125 and at 0.125 it is 0.03125. This justifies the statement of changing quantum jumps with different group of numbers.

Now if we supply a decimal number 2.15 to this 7-bit machine, it cannot be represented exactly ts number is not present in the allowed numbers on the machine. Under such a situation the nearest closer number 2.0 is considered as its approximate representative and 2.15 is written as 0010100 in binary number system.

ABSOLUTE, RELATIVE AND PERCENTAGE ERRORS

In numerical computation errors are bound to occur and it is essential to see whether the obtained result is within a limit of tolerable errors or not. Hence knowledge about different measures of error is necessary. The three basic measures of errors are absoluter error, relative error and percentage error.

These errors can be defined as follows:

Absolute Error: If pa is an approximation to p then absolute error is defined as absolute value of difference between p and pa, i.e.,

$$Ep = |p - pa|$$

Relative Error: If pa is an approximation to p and p is not equal to zero then the relative error is defined as absolute value of the ratio of difference between p and pa with p, i.e.,

$$RP = |\frac{(p - pa)}{p}|$$

Percentage error is hundred times the relative error. PP = 100 × RP. If pa is an approximation to p and p is not equal to zero then percentage error can be expressed as:

$$PP = |\frac{(p - pa)}{p}| \times 100$$

Errors while representing number on computer system:

- Chopping off error
- Round off error
- Truncation Error

Question: How error propagates in different arithmetic operations?

ALGORITHM

An algorithm is a precise specification of a sequence of instructions to be carried out in order to solve a given problem.

BASIC PROPERTIES OF AN ALGORITHM

In order to qualify as an algorithm, a sequence of instructions must possess the following five characteristics:

i. An algorithm should contain instructions to accept inputs. Then inputs are processed by the subsequent instructions in the algorithm.
ii. Processing rules specified in the algorithm must be precise and unambiguous. It should be possible to carry out the given instructions
iii. Each instruction must be such that a person can carry it out in finite time with pen and paper
iv. The total time required to carry out all the steps in the algorithm must be finite
v. An algorithm must produce one or more outputs.

In fact, there are no syntax or semantic rules for writing a computer algorithm. However, one can design one's own way to express an algorithm. A structured approach is always helpful in developing an algorithm. Some people use Pascal-like approach. This can be directly implemented into any programming language.

GOLDEN RULES FOR ALGORITHM DESIGN

In order to carry out a task using a computer the following rules are very useful. First take a look at the problem as a whole. Make sure that you understand the problem completely.

- Analyze the requirements of the problem; you may use your own language to write important points.
- Stop and think and think again.
- Use available literature for a background reading of the problem.
- Find out if there exists any known method for solving your problem.
- Now develop an algorithm to describe how your task may be performed.
- Don't try to be clever! Even a simple method may prove superior to a complicated one
- As far as possible avoid using jumps' in the set of algorithms. Try to use concepts of structures.
- Use meaningful variable names.
- Use remarks wherever relevant. They may be useful even for the program designer at a later time.

PROGRAMMING

When the required algorithm is developed, it is converted into a computer program. In fact, programming is a process of designing and describing an algorithm to solve a class of problems with the help of computer.

In order to develop a complete program and get it to work correctly, series of steps, based on above information, should be followed. The steps are:

- Defining the problem
- Planning
- Coding
- Desk checking
- Program testing
- Debugging
- Documentation

FLOW CHART

A flowchart is an intermediate step between algorithm design and programming. It is pictorial representation of an algorithm. It uses boxes of different shapes to denote different types of instructions as shown in fig below. The actual instructions are written inside the boxes. Solid lines having arrow marks on them contact the boxes. The arrow marks shows the exact sequence in which the instructions are to be executed.

Basically, an algorithm is first represented in the form of a flowchart and the flowchart is then translated in some programming language. These two steps approach in programming has some advantages. During flowcharting one is not Concerned with the, details of the elements of programming languages. Hence, he can fully concentrate on the logic of the procedure. Also, any error in the logic of the procedure can be detected more easily than in the case of a program.

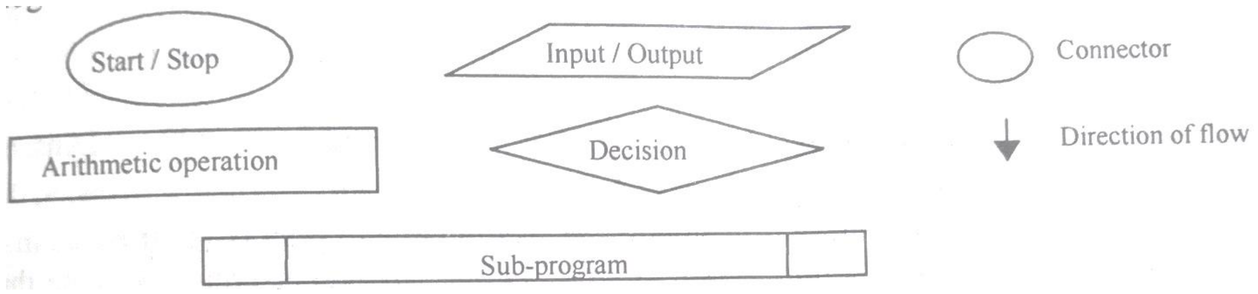


Fig. Flow chart symbols

Once the flowchart is ready, the programmer can concentrate only on coding the operation in each box of the flowchart in terms of the statements of the programming language. Normally, this will ensure an error free program.

Experienced programmers write programs without flowchart. However, for a beginner it is recommended that a flowchart be drawn first in order to reduce the number of errors and omissions in the program.

QUESTIONS

1. What is numerical method? Explain the characteristics of numerical computing.
2. Develop an algorithm to find the value of e^x up to three decimal places of accuracy. Read the value of x (< 1) from keyboard and print the result on screen.
3. If coefficients of a quadratic equation are known write an algorithm to find roots of quadratic equation in the closed form.
4. Explain the decimal system of numbers. Discuss the representation of integers and fractions in this system.
5. Explain the binary system of numbers. Discuss the representation of integers and fractions in this system.
6. What are the surprises of computer representation of numbers? Justify them.
7. Define absolute error, relative error and percentage error in number representation. Explain them with one example each.
8. Distinguish between chopping error and rounding error.
9. Discuss the validity of commutative rules of arithmetic on computer calculations.
10. What is an algorithm? Give its basic properties.

Program: If coefficients of a quadratic equation are known write an program to find roots of quadratic equation in the closed form

NUMERICAL SOLUTIONS OF TRANSCENDENTAL EQUATIONS

INTRODUCTION

Root or zero of a function f (X) is the value of X such that f (X) = 0. The need to find some or all the roots of a function is a common situation in scientific work. In some cases, like quadratic function aX^2+bX+c, there may be an explicit non-iterative formula for the roots; but such cases are rare. There is no general formula for the roots of a polynomial greater than fourth order and for the roots of an expression containing trigonometric, hyperbolic and logarithmic functions or in general non-linear functions or expressions. When these expressions are set equal to zero, they are identified as transcendental equation. In such cases one has to use iterative methods to obtain approximate roots. There are many numerical iterative methods to obtain roots of such functions.

Of course, one particular method may be suitable only for specific type of function like a polynomial or a function of general nature. Further it may find only real roots or complex roots also. The goal of the chapter is to introduce different methods for finding numerical approximations for the roots of a function. The bisection method, False position method, Newton-Raphson method, secant method and successive iteration method are discussed in length. The Birge-Vieta method especially useful for obtaining roots of polynomials is also discussed.

DEFINITION OF ROOT OR ZERO OF A FUNCTION

By mathematical definition, zero of a function f(X) or root of an equation f(X) = 0 is the value 'a' for is f(a) = 0 Geometrically root of a function f(X) is the point on f(X) and X curve at which f(X) has zero value. In other words, root is the point of intersection of function curve with X-axis. The X-value of this point is also identified as solution of the equation.

In numerical calculations (i.e., practical computing) it must be understood that the equation f(X) = 0 cannot be satisfied exactly because of different computational errors associated with the method. Hence the mathematical definition of root has to be modified suitably. The numerical root of a function f(X) can be considered as the value of X for which |f(X)| < € where € is a given tolerance or permitted error in root value. The inequality equation allows, as numerical root, a range of X_1 and X_2 about the exact root. The interval $[X_1, X_2]$ is very small and the function values $f(X_1)$ and $f(X_2)$ have opposite signs; in rare case one of the function values may be zero.

CONCEPT OF ITERATIVE METHODS

Iterative method is a trial and error process for finding an answer to a question. In this method One guesses the answer and then tests whether the guess is really the answer. If the guess is not the correct answer then another guess is made and the process is repeated till satisfactory guess is reached. Thus, iterative method is a repetitive procedure, with every iterative operation consisting of guess and check operations. An algorithm for the method is as follows:

Step 1: Start

Step 2: Guess the answer

Step 3: Evaluate the value of function.

Step 4: If absolute value of the function > error limit then go to step 2

Step 5: Print the result

Step 6: Stop

Random guess of the answer may lead either to large number of trials before reaching the correct guess or to not getting the answer at all even after very large number of trials. This situation can be avoided by removing randomness for initial and successive guesses by deciding some criteria for them.

Iterative methods are applicable to varieties of problems; finding root of a function is one of them. Bisection method, False position method, Newton Raphson method and secant method are the four important methods of finding root of transcendental equation.

Error limit and accuracy of result: On computer, even correct guess of result may not be accepted as correct answer because it is difficult to calculate functions without approximation and because there is a limit on precession in representing number on a computer. Hence some error is always allowed to check correctness of the answer. The limit on the error decides the accuracy or precession of the result obtained. If absolute error allowed in X is 0.001 then its obtained value may differ from the correct one at most by 0.001 or it, is correct up to three digits after the decimal point. If the value of X is small and its order is comparable to the absolute error limit then absolute error limit is not good for testing the correctness of the answer. Relative error limit provides another criterion for testing the correctness of the answer. The relative error limit gives us the idea of precession in terms of significant digits. If relative error limit of X is 10^{-n} then the answer has n significant figures, i.e., the answer is certain up to the nth significant position.

SEARCH METHOD FOR INITIAL GUESS

Fundamentals: In order to find roots of a function by iterative method one needs some starting values. The theorem that is useful for this purpose is the Intermediate value theorem. It states that if the values of a continuous function at two points are of opposite sign then the function must have its zero between the two points. This suggests that we can develop a method to find two points containing a root by calculating the values of the function at different points and then selecting the neighboring pair of points at which function values are of opposite sign. If it is known that search of intervals containing zero is to be made in the range [a, b] of X then this range is divided into 'n' equal subintervals so that the size of the subinterval is h = (b-a)/n. The function values are calculated at a+h, a+2h, .. a+nh and suitable neighboring X values with opposite function values are selected. If X_1, are two such X values then they can be taken as lower and upper limits of a range for finding a root by bisection method, false position method and secant method. Any one of X_1 or X_2 can be used as starting value for Newton-Raphson method or successive iteration method.

Limitations: The search method has the drawback that it may not provide the range of root for all the zeros even in the specified limits of X (i.e., a and b). If there is two or any even number of closely spaced roots in a subinterval then they would be missed entirely because of even number of sign changes in the function value within a subinterval lead to no sign change in the function value at the end points of the subinterval. If there are an odd number of closely spaced roots in a subinterval then only one can be isolated. If function value is zero at one of the end points of a subinterval then it cannot provide the range containing root.

Algorithm: In writing an algorithm for the search method it is assumed that search of intervals containing zero is to be made in the range [a, b] of X. This range is divided into n equal subintervals so that the size of the subinterval is h = (b—a)/n and that the function values are calculated for x at a, a+h , a+2h . a+nh. Existence of root in a specific subinterval is decided from the opposite signs of Y values at the end x values of that subinterval and this is tested by negative value of product of the two Y values. The values of a, b and n are the inputs and pairs of x values containing root are the outputs.

```
Reading input data
READ a b, N
H ← (b - a)/N
Initialing interval left side
I ← 0
X1 ← a
Y1 ← f (a)
Y2 ← Y1
Deciding and checking suitability of interval
WHILE I < = N DO
     I ← I +1
     X2 ← X1 + h
     Y2 ← f (X2 )
     IF Y1*Y2 <= 0 THEN
          PRINT Xl, X2, Y1, Y2
     END OF IF
     X1 ← X2
     Y1 ← Y2
END OF WHILE
END
```

Example 1: Divide the interval [1,5] in 8 equal subintervals and find those subintervals which contain root of the function X^2 - 6.310X + 9.061.

Answer: In this example a = 1, b = 5, N= 8 and f(X)= X^2 - 6.310X + 9.061. Hence h=(5-1)/8= 0.5. The values of X and corresponding f(X) are

X	1.000	1.555	2.000	2.500	3.000	3.500	4.000	4.500	5.000
f(X)	3.751	1.846	**0.441**	**-0.464**	-0.869	-0.774	**-0.179**	**0.916**	2.511

The subintervals with opposite function values and hence containing root are [2.000, 2.500] and 4.000, 4.500].

Example 2: Divide the interval [0,1.4] in equal subintervals of size 0.2 and find those subintervals, which Contain root of the function X sin(X) - 0.2

Answer: In this example a = 0, b =1.4, h = 0.2, and f(X) = X sin(X) - 0.2. The values of X and corresponding f(X) are

X	0.000	0.200	0.400	0.600	0.800	1.000	1.200	1.400
f(X)	-0.200	-0.160	**-0.044**	**0.139**	0.374	0.641	**0.918**	**1.180**

The subintervals with opposite function values and hence containing root are [0.400, 0.600].

Exercise:

1. Give the fundamentals of search method to get initial guess of root or an interval containing it. Write an algorithm for it.
2. What are the different possible criteria for terminations of iterative methods to find root of an equation? Explain the situations under which each of them is used.

Program: Write a Program to find the subinterval containing the root from the initial guess from the user.

Answer:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main ()
{
clrscr();
int i,n;
float a,b,h,d,x1,x2,y1,y2;
cout<<"Pleae enter the lower limit of the
interval =";
cin>>a;
cout<<"Please enter the upper limit of the
interval =";
cin>>b;
cout<<"Please enter the number of intervals =
";
cin>>n;
h=(b-a)/n;
x1=a;
cout<<"The values of x and corresponding
f(x) are \n x \t f(x)"<<endl;
for (i=1;i<=n;i++)
{
y1=x1*sin(x1)-0.2;
cout<<x1<<"\t"<<y1<<"\t\n";
x2=x1+h;
y2=x2*sin(x2)-0.2;
if (y1*y2<0)
cout<<"The root exist b/w the interval =
"<<x1<<"        "<<x2<<endl;
x1=x2;
y1=y2;
}
getch();
}
```

BISECTION METHOD

Fundamentals: Bisection method is an iterative method used to find solution of transcendental equation. The basic idea of the method is:

(i) guessing an interval $[X_1, X_2]$ of X containing the root X_r of a function f(X)

(ii) splitting the interval into two subintervals and

(iii) using part of it, containing root, as new interval making it successively smaller.

Ultimately such splitting will reduce the range of intervals to the root value X_r.

The reduction in the size of the X interval is made by splitting it into two equal intervals $[X_1, X_a]$, $[X_a, X_2]$ with the help of average value $X_a = (X_1+X_2)/2$ and using only that interval which will contain the root. For every guess of the interval the average value is considered as the approximate root X_a the function f(X). When the interval size will become very small then X_1, X_2 and X_a will nearly coincide and hence they will represent the exact root X_r.

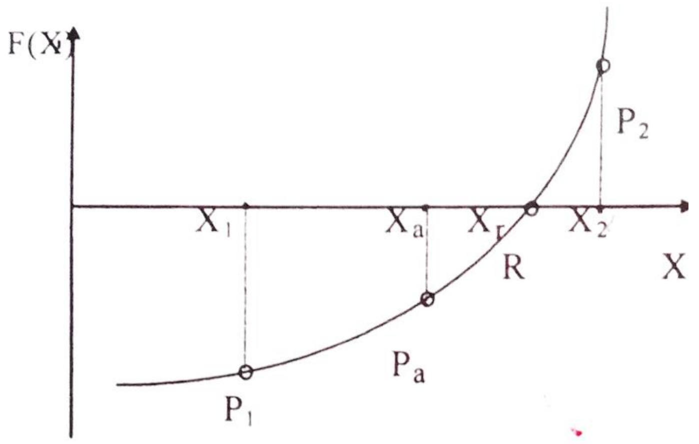


Fig. Bisection method

Geometrical interpretation: The process of bisection method can be presented graphically using a plot of function f(X) against X. [refer fig.]. Consider R as point of intersection of the function curve with X-axis. This point represents the root of the function. Let P_1, P_2 be two points (with X values X_1, X_2) on the function curve and on either side of R. The average of X_1, X_2, i.e., $X_a=(X_1+X_2)/2$ corresponds to the point P_a on the function curve and it is taken as an approximation of R. As seen from the figure X_a divides the interval $[X_1, X_2]$ into two equal parts $[X_1, X_a]$ and $[X_a, X_2]$. From the figure the new interval containing R is $[X_a, X_2]$. Thus, bisection method is, geometrically, the process of making the interval half, by the mid-point of the earlier interval.

Guess criteria: The initial guess of the interval $[X_1, X_2]$ is made such that the values of f(X) at X_1 and X_2 are of_opposite signs. This criterion assures that the interval contains a root of f(X). The average of X_I, X_2, denoted by X_a is taken as a guess of root and from the two intervals $[X_1, X_a]$ and $[X_a, X_2]$, a new suitable interval is obtained on the basis of opposite signs of f(X) at end points of the interval. If X_1 and X_a are such that the function values $f(X_1)$, $f(X_a)$ have opposite signs then

(X_1, X_a) is taken as new interval with $X_1 \rightarrow X_1$ and $X_a \rightarrow X_2$ else the interval (X_a, X_2) is taken as new interval with $X_a \rightarrow X_1$ and $X_2 \rightarrow X_2$. In other words the new guess interval is formed by replacing X_1 or X_2 by X_a, depending on which X the value of f(X) has the same sign as that of $f(X_a)$.

Termination criteria: One can continue the process of finding new intervals and new central points till the central point of the interval satisfies $f(X)\approx 0$. Many times, to arrive to this situation hundreds of steps of finding new interval and finding its central point will be required or the situation may not be arrived at the situation at all. Hence the process of finding new interval is stopped when some specified number of significant figures is obtained in the value of X_a, the approximate root. This can be tested by checking smallness of relative error in root as compared to specified limit €. ,i.e.

$$|(X_{anew} - X_{aold}) / X_{anew}| < €$$

The value of X_{anew} is X_a and X_{aold} is either X_1 or X_2; hence $|X_{anew} - X_{aold}|$ is either $|X_a-X_1|$ or $|X_2-X_a|$ and both being equal, imply

$$|(X_{anew} - X_{aold})| = |X_2-X_1| / 2$$

The condition on relative error testing can, then, be considered as

$$|0.5(X_2 - X_1) / Xa| < €$$

The interval having process is stopped when either $f(X)\approx 0$ or $|0.5(X_2 - X_1) / Xa| < €$ criterion is satisfied.

Steps in the method: The program of finding root of a function f(X) by bisection method will consist of the following steps .

i. Input the initial interval points and limit of error.
ii. Check the correctness of the interval by testing opposite signs of $f(X_1)$ and $f(X_2)$.
iii. Initialize iteration counter and set the values of function and calculated error.
iv. Check if error reached its limiting value or iteration number crossed its limit; if so then go to step (vi).
v. Decide the new interval, new X_a, new error and go to step(iv) after incrementing the counter.
vi. Output, the result.

Computational effort: It is normally measured in terms of number of evaluations of function values as it is an extremely time-consuming operation. In this method there is only one function value evaluation and hence the method is with less computational effort.

Rate of convergence: It gives us the idea about the fastness of the method to arrive at the root or fastness to reduce the error in the approximate result. The rate of convergence k is the largest integer such that

$$\lim_{i\to\infty} \frac{E_i + 1}{E_i^k}$$

where M is a finite number. E_i is the error in ith step. For a process with rate of convergence k, error in i+1 step is proportional to kth power of error in the previous, i.e., ith step. In bisection method the error at any step is half of the interval size, i.e., $(X_2 - X_1)/2$ and the next step has an error half of the error in the previous step; this is so because every step reduces interval containing a root to half the size. Thus

$$E_{i+1} = 0.5\ E_i$$

Comparing this with the relation defining k we find that M=0.5 or k=1. Thus, the convergence by the bisection method is slow with rate of convergence unity.

Stability: Stability of an iterative method is decided by certainty of providing convergence. The method of bisection being the method of bracketing (i.e., the interval will always contain root) and every step of method decreases the interval size, it will always bring approximate root closer to the correct root. This is also obvious from the fig. ultimately it will reduce to a single point. This provides guarantee of convergence and makes the method reliable.

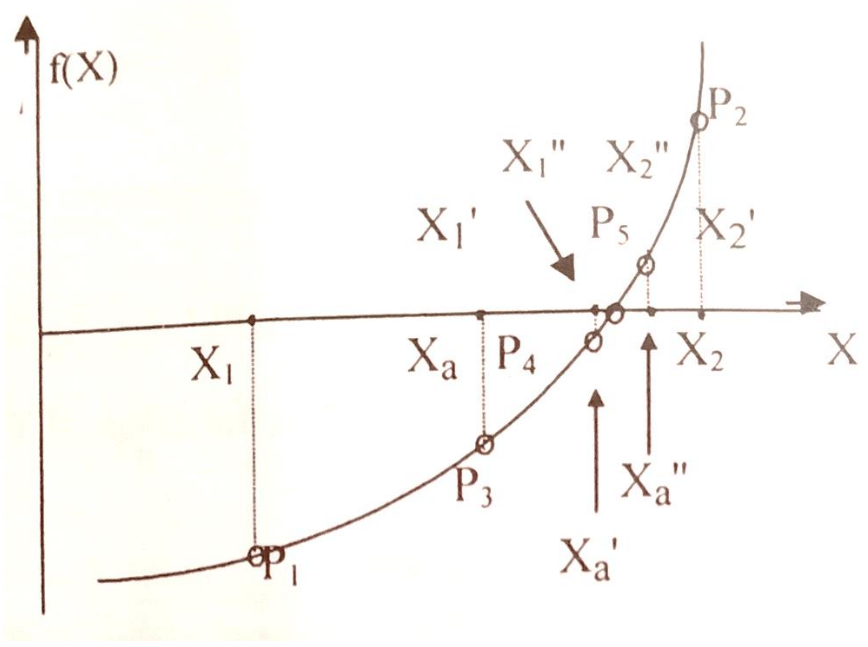


Fig. Stability of bisection method

Efficiency: The bisection method divides an interval into two exactly equal parts without taking into account the function values at the interval end points and without considering at which end point f (X) is closer to zero. This indicates that the method follows a typical routine procedure without the use of any intelligent idea to speed up the action of reaching the correct root. Because of this the bisection method is known to be of "brute-force" type and is less efficient.

Bisection Method

Algorithm: In writing the algorithm of the bisection method X_1 and X, are considered as end point X values of an interval with X_a as mid point X value. Every iteration consists of guessing the interval, guessing the root value from that interval and checking for the required convergence. Since initially the guess interval is already known from input, the part of guessing the interval can be avoided in the first iteration. However, for the sake of forming a block of iterative steps, X_a is initialized as X_2 and guessing of interval is allowed in first iteration also. The interval guessed in the first iteration will be the supplied interval only. To stop the iterative procedure checking is done for smallness of relative error in the value of X_a. The value of eps is taken as the limit of this error. The algorithm is such that at least one iteration process will take place. The algorithm, also, makes the provision for:

a) Checking the suitability of initial interval before starting the iterative steps.
b) Stopping the iterative procedure if number of iterations exceeds some number (say N), and
c) Printing root value with its possible error and warning of "no convergence" if number of iterations become equal to N. The values X_1 X_2, eps and N are the inputs.

```
Reading input data
READ Xl, X2 , eps , N
Y1 ← f (X1) and Y2 ←f (X2)
Checking suitability of given interval
IF SIGN(Y1) = SIGN(Y2) THEN
    WRITE " Starting interval unsuitable"
    END
END OF IF
Initialization of iteration counter and set the
value of function and calculated error
I ← 1
ER ← 1.5 * eps
Checking for convergence and deciding new
interval & root
WHILE (ER>eps AND 1≤ N) DO
    XA ← (Xl+X2 ) / 2
    YA ← f (XA )
    ER ← |0.5* (X2 -X1) / XA|
IF SIGN(Y1)=SIGN (YA)
THEN
          X1← XA
          Y1←YA
ELSE
          X2← XA
          Y2 ←YA
    END OF IF
    I ←I+ 1
END OF WHILE
Output of results
WRITE "root = "; XA, " with error = "; eps
IF I > N THEN DO
          WRITE " No convergence "
ELSE
          WRITE "Convergence "
END OF IF
END
```

Example 1: Using bisection method find the root of X^2 - 21=0 between (2.0,6.0) with relative error 0.01. Terminate the program on exceeding 10 iterations.

Answer: The example is for finding root by bisection method. In this example $f(X)= X^2-21$; X1=2.00 and X2= 6.00. The convergence check is by relative error = 0.01.

The formula for error is: error = $0.5*(X_2-X_1)/Xa$.

Iter	Change	X1	X2	Y1	Y2	XA	YA	Error	Check
		2.000	6.000	-17.000	15.000	---	---	---	Proper Range
1		2.000	6.000	-17.000	15.000	4.000	-5.000	0.500	No Convergence
2	$X_a \rightarrow X_1$	4.000	6.000	-5.000	15.000	5.000	4.000	0.200	No Convergence
3	$X_a \rightarrow X_2$	4.000	5.000	-5.000	4.000	4.500	-0.750	0.111	No Convergence
4	$X_a \rightarrow X_1$	4.500	5.000	-0.750	4.000	4.750	1.563	0.053	No Convergence
5	$X_a \rightarrow X_2$	4.500	4.750	-0.750	1.563	4.625	0.391	0.027	No Convergence
6	$X_a \rightarrow X_2$	4.500	4.625	-0.750	0.391	4.563	-0.184	0.014	No Convergence
7	$X_a \rightarrow X_1$	4.563	4.625	-0.184	0.391	4.594	0.103	0.007	Convergence

Root value is = 4.594

Question 1: Using bisection method find the cube root of 10 between (2.0,3.0) within 1.0% error.

Answer: 2.14

Question 2: Using bisection method, find the root of X^3-4X+1=0 between (1.0,2.0) with absolute error 0.02. Terminate the program if iteration number exceeds 10.

Answer: 1.859

Question 3: Using bisection method find the root of X sin(X) -3 cos(X) = 0 between (0.0,1.8) with accuracy of 2 digits after decimal point. Terminate the program if number of iterations exceeds 10.

Answer: 1.195

Question 4: Using bisection method find the root of 0.4*X^2-39=0 between (9.0, 11.0) with percentage error 1. Terminate the program if number of iterations exceeds 10.

Answer: 9.813

Question 5: Using bisection method find the root of $e^x - 2 = 0$ between (0.0, 1.4) correct up to 2 significant figures. Terminate the program if number of iterations exceeds 10.

Answer: 0.695

Question 6: Give the fundamentals of Bisection method. Discuss the stability of the method.

Question 7: Write an algorithm of Bisection method considering an interval, error limit as input. The algorithm is expected to check the suitability of the initial interval.

Question 8: What is rate of convergence of an iterative method? Derive its expression for bisection method.

Question 9: Give the flow chart of bisection method. Mention the facilities provided in it.

Question 10: Explain, geometrically, how the choice of guess for new interval and new root are made in any iteration or bisection method. Discuss the criteria for testing convergence in the method

Week 6 Bisection Method

Program: Write a Program to find the root of equation by using bisection method, With relative error of 0.01. Terminate the program if number of iterations exceeds than the given limit by user.

Answer:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<iomanip.h>
void main ()
{
clrscr();
float x1,x2,y1,y2,xa,ya,err=1,tol;
int i=1,n;
cout<<"Enter the starting interval = ";
cin>>x1;
cout<<"Enter the limiting interval = ";
cin>>x2;
cout<<"Enter the tolerance = ";
cin>>tol;
cout<<"Please enter the number of itterations = ";
cin>>n;
cout<<"x1\tx2\ty1\ty2\txa\tya\terr\tCheck\n";
cout<<"________________________________\n";
while (err>tol&&i<=n)
{
y1=x1*x1-21;
y2=x2*x2-21;
xa=(x1+x2)/2;
ya=xa*xa-21;
if (ya*y1<0)
{
y2=ya;
x2=xa;}
else
{
y1=ya;
x1=xa;}
err=(0.5*(x2-x1)/xa);
cout<<setprecision(4)<<x1<<"\t"<<x2<<"\t"
<<y1<<"\t"<<y2<<"\t"<<xa<<"\t"<<ya<<"\t"
<<err<<"\t";
if (err>tol)
cout<<"No Convergence\n\n";
else
cout<<"Convergence\n\n";
i++;}
cout<<"_________________\n";
cout<<"The root value of equation = 
"<<xa<<endl;
getch();
}
```

FALSE POSITION METHOD (linear interpolation or regula falsi method)

Fundamentals: False position method is also an iterative method to find solution of transcendental equation. The basic idea of the method is to guess an interval [X_1, X_2] of X containing the root X_r of a function f(X) and modify the interval successively such that one of the end of the interval ultimately converge to X_r. In this method the interval is made successively smaller to converge to some finite size and not to zero size. Between X_1, X_2 the function is assumed to vary linearly and the point of intersection of straight line and X-axis is considered as new guess. It is calculated from the relation

$$X_a = \frac{X_1 f(X_2) - X_2 f(X_1)}{f(X_2) - f(X_1)}$$

Here f(X_1) and f(X_2) are the function values at X_1 and X_2. Now out of intervals [X_I, X_a], [X_a, X_2] the one containing root is considered as new interval. The above procedure is repeated. The interval size becomes constant when one end point coincides with the exact root X_r.

Derivation of formula for root: The formula for X_a can be obtained by considering the nature of the function between X_1 and X_2 as linear function like

$$f(X) = \frac{f(X_2) - f(X_1)}{X_2 - X_1}(X - X_1) + f(X_1)$$

If X_a is the root of the function ,i.e., the value of X at which f (X)=0 then

$$0 = \frac{f(X_2) - f(X_1)}{X_2 - X_1}(X_a - X_1) + f(X_1)$$

$$X_a = \frac{X_1 f(X_2) - X_2 f(X_1)}{f(X_2) - f(X_1)}$$

Geometrical interpretation: The process of false position method can be presented graphically using a plot of function f(X) against X [fig.]. Consider R as the point of intersection of function curve with X axis. This point represents the root of the function and has X value as X_r,

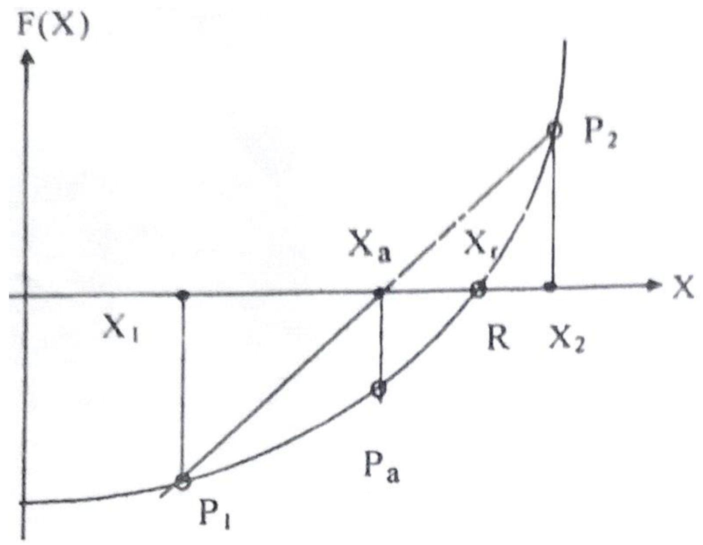


Fig. False position method

Let P_1 and P_2 be two points (with X values X_1, X_2) on the function curve and on either side of R. The linear approximation of the function between P_1 and P_2 is shown by a straight line or chord joining the points P_1 and P_2. The value of X_a is calculated using relation

$$X_a = \frac{X_1 f(X_2) - X_2 f(X_1)}{f(X_2) - f(X_1)}$$

is the same as X value of the point of intersection of chord of the function curve between X_1, X_2 and X-axis. The corresponding point P_a on the function curve is taken as the approximation of R.

Guess criteria: The initial guess of the interval $[X_1, X_2]$ is made such that the values of f(X) at X_1 and X_2 are of opposite signs. This criterion assures that the interval contains a root of f(X); while varying X from X_1 to X_2. The guess of new root X_a is obtained using relation

$$X_a = \frac{X_1 f(X_2) - X_2 f(X_1)}{f(X_2) - f(X_1)}$$

The guess of new suitable interval from the two intervals obtained after the division of earlier interval at X_a is made on the basis of existence of opposite signs of f(X) at end points of the selected interval. If X_1 and X_a are such that the function values $f(X_1)$,$f(X_a)$ have opposite signs then $[X_1, X_a]$ is taken as new interval with $X1 \rightarrow X1$ and $Xa \rightarrow X2$ else the interval $[X_a , X_2]$ taken as new interval with $X_a \rightarrow X_l$ and $X_2 \rightarrow X_2$. In other words the new guess interval is formed by replacing X_a in place of X_1 or X_2, at which function value has the same sign as that of f (X_a).

Termination criteria: One can continue the process of finding new guess of root and modifying the interval till the new guess of root satisfies f(X) = 0 exactly. Many times to arrive to this situation hundreds of steps will be required or the situation may not be arrived at all due to limitation of precision in number representation on the computer. Hence the iterative process of finding new interval is stopped when $|f(X)| \leq$ some specified limit, say $€_i$, or difference between X_{anew} and X_{aold}

is less than some specified limit. This can be tested by checking if relative error in root is less than the specified limit €.

i.e. $|(X_{anew} - X_{aold}) / (X_{anew}| \leq €$

The interval modifying process is stopped by either

$| f(X) | \leq €_1$ or

$|(X_{anew} - X_{aold}) / X_{anew}| \leq €$

whichever will be satisfied earlier.

Steps in the method: The program of finding root of a function f(X) by false position method will consists of the following steps.

i. Input the initial interval points and limit of error.
ii. Check the correctness of the interval by testing opposite signs of $f(X_1)$ and $f(X_2)$.
iii. Initialization of iteration counter, approximate root and calculated error.
iv. Check if error reached its limiting value or iteration number crossed its limit; if so then go to step (vi).
v. Decide the new interval, new Xa, new error and go to step (iv) after incrementing the counter by one.
vi. Output the result.

Computational effort: It is normally measured in terms of number of evaluations of function values (it being most time-consuming operation). In this method there is only one function value evaluation in one iterative step and hence the method is with less computational effort.

Rate of convergence: It gives us the idea about the fastness of the method to arrive at the root or fastness of reduction in the error in the approximate result obtained. The rate of convergence k is the largest integer such that

$$\lim_{i\to\infty} \frac{E_i + 1}{E_i^k} \leq M$$

where M is a finite number. E, is the error in i^th iteration or step. For a process with rate of convergence k, error in any step is proportional to k^th power of error in the previous step.

In false position method the root value is enclosed in the interval $[X_1,X_2]$ and hence possible error at i^th iteration is $e_i = X_2—X_1$. In the next i.e 1+1^th iteration new interval $[X_2, X_a]$ is found and error becomes $e_{i+1} =(X_2-X_a)$.

Substituting the expression X_a we get

$$e_{i+1} = X_2 - \frac{X_1 f(X_2) - X_2 f(X_1)}{f(X_2) - f(X_1)} = \frac{f(X_2)(X_2 - X_1)}{f(X_2) - f(X_1)} = \frac{f(X_2)}{f(X_2) - f(X_1)}\, e_i$$

$$= \frac{f(X_r) + (X_2 - X_r)\acute{f}(X_r) + \cdots}{f(X_r) + (X_2 - X_r)\acute{f}(X_r) + \cdots - \ f(X_r) - (X_1 - X_r)\acute{f}(X_r) + \cdots} e_i$$

$$\approx \frac{(X_2 - X_r)\acute{f}(X_r) + \cdots}{(X_2 - X_r)\acute{f}(X_r) - (X_1 - X_r)\acute{f}(X_r)} e_i$$

Since X_a is going closer to X_1 and also to X_r, we can replace X_r by X_1, this simplification gives

$$e_{i+1} \approx \frac{(X_2 - X_r)\acute{f}(X_r)}{(X_2 - X_r)\acute{f}(X_r)} e_i$$

$$= e_i$$

and the next step has an error nearly equal to the error in the previous step. Comparison of this with the relation defining k we find k = 1. Thus, the convergence in false position method is slow with rate of convergence unity.

Stability: Stability of an iterative method is decided by certainty of providing convergence. Method of false position being the method of bracketing, i.e., the interval always contains root and, every step of the method always brings one end point of interval closer to the correct root. This provides guarantee of convergence and makes the method reliable.

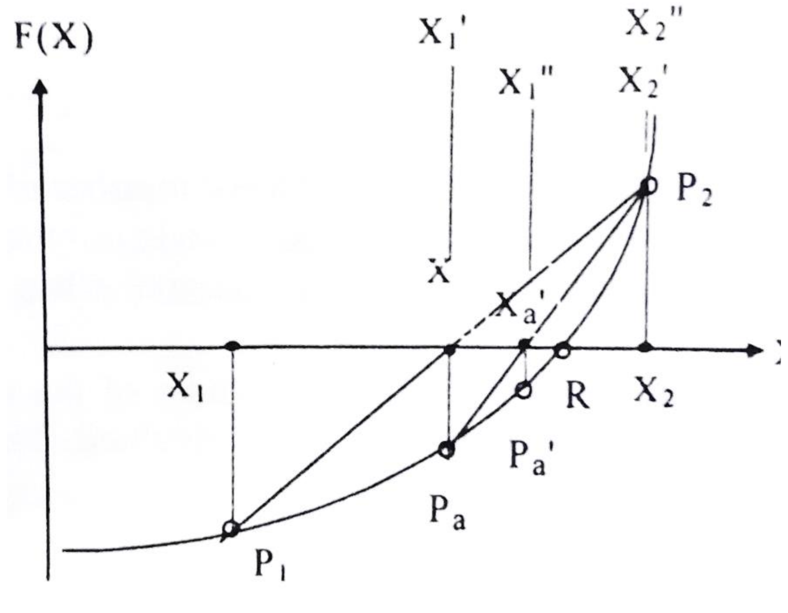


Fig. Stability of False position method

In the graphical representation (fig.) successive approximation points in false position method are indicated by points P_a, P_a', P_a'', etc. whereas the exact root is shown by point R. In the first iteration the initial guess interval is [X_1, X_2] and root is X_a. During second iteration guess interval is [X_1', X_2'] which is same as the interval [X_{aold}, X2] and root is X_a'. It is clear that the successive approximation points P_a, P_a' approach R and X_a, X_a' approach the exact root. Every step of the method brings the approximate root closer to the exact root and provides stability of the method.

Efficiency: This method decides new value of approximate root not by merely finding central point but by finding the point of intersection with X axis with due consideration of the nature of function variation. In general this speeds up the action of reaching correct root and provides more efficient action.

Algorithm: In writing the algorithm of the false position method X1 and X2 are considered as end point X values of an interval with XA as new guess for X value. A block of iterative procedure is considered and iteration consists of guessing the interval, guessing the root value from that interval and checking for the required convergence. Since initially the guess interval is already known from input, the algorithm makes a provision that during the first iteration the guessing of the interval gives the input interval. The iterative procedure is stopped by checking the relative error in the value of XA with eps taken as the limit of this error. The algorithm, also, makes the provision of

a) Checking suitability of the initial interval
b) Stopping the iterative procedure if number of iterations exceed some number (say N), and
c) Printing root value with its possible error and warning of “no convergence" if number of iterations become equal to N. The values XI, X2, eps, and N are the inputs.

```
Reading input data
READ X1, X2, eps, N
Y1 ← f(X1)
Y2 ← f(X2)
Checking suitability of given interval
IF SIGN(Y1) = SIGN(Y2) THEN
    WRITE "Starting interval unsuitable"
    END
END OF IF
Initialization of iteration counter, approximate root and error
I ← 1
XO ← X2
ER ← 1.5 * eps
Checking for convergence and deciding new interval & root
WHILE ER>eps AND I≤N DO
    XA ← (X1*Y2 - X2* Y1) / (Y2-Y1)
    YA ← f(XA)
    ER ← | (XA - XO) / XA|
    IF SIGN(Y1) = SIGN (YA) THEN
        X1 ← XA
        Y1 ← YA
    ELSE
        X2 ← XA
        Y2 ← YA
    END OF IF
    XO = XA
    I = I+1
END OF WHILE
Output of results
WRITE "root = "; XA, with error = "; eps
IF I > N THEN DO
    WRITE " No convergence "
ELSE
    WRITE "Convergence"
END OF IF
END
```

Example: Using false position method finds the root of X^2-21= 0 between (2.0,6.0) With relative error 0.01. Terminate the program if iteration number exceeds 10.

Answer: The example is on finding root by false position method. In this example f(X) =X^2-21; X1=2.00 and X2= 6.00. The convergence check is by relative error = 0.01. The formula for error is: error = (X_a—X_{aold}) / Xa; X_{aold} =X_1 for I=1.

Iter	Change	X1	X2	Y1	Y2	XA	YA	Error	Check
0	---	2.0000	6.0000	-17.000	15.0000	---	---	---	Proper Range
1	---	2.0000	6.0000	-17.000	15.0000	4.1250	-3.9844	0.5152	No Convergence
2	$X_a \rightarrow X_1$	4.1250	6.0000	-3.9844	15.0000	4.5185	-0.5830	0.0871	No Convergence
3	$X_a \rightarrow X_1$	4.5185	6.0000	-0.5830	15.0000	4.5739	-0.0790	0.0121	No Convergence
4	$X_a \rightarrow X_1$	4.5739	6.0000	-0.0790	15.0000	4.5814	-0.0106	0.0016	Convergence

Root Value is = 4.58

Question 1: Using false position method finds the cube root of 10 between (2.0,3.0) With relative error of 1 percent.

Answer: 2.15

Question 2: Using false position method finds the root of X^3- 4X + 1= 0 between (1.0,2.0) With absolute error 0.02. Terminate the program if iteration number exceeds 10.

Answer: 1.861

Question 3: Using false position method find the root of X sin(X) -3 cos(X) = 0 between (0.0,1.8) with accuracy of 2 digits after decimal point. Terminate the program if number of iterations exceeds 10.

Answer: 1.192

Question 4: Using false position method find the root of 0.4*X^2-39=0 between (9.0, 11.0) with percentage error 1. Terminate the program if number of iterations exceeds 10.

Answer: 9.87

Question 5: Using false position method find the root of $e^x - 2 = 0$ between (0.0, 1.4) correct up to 2 significant figures. Terminate the program if number of iterations exceeds 10.

Answer: 0.691

Question 6: Derive the formula for guess of root of a function in false position method. Give the geometrical interpretation of the formula.

Question 7: Show that the rate of convergence of false position method is unity.

Question 8: Draw the flow chart of false position method. Mention the facilities provided in it.

Question 9: With the help of graphical representation of false position method justify that the method is stable.

Question 10: Give algorithm of false position method to find root of a transcendental equation. Explain the variables used and mention the facilities provided in it.

Question 11: Explain, geometrically, the how the choice of guess for new interval and new root are made in any iteration of False position method. Discuss the criteria for testing convergence in the method.

Program: Write a Program to find the root of equation by using false position method, With relative error of 0.01. Terminate the program if number of iterations exceeds than the given limit by user.

Answer:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<iomanip.h>
void main ()
{
clrscr();
float x1,x2,y1,y2,xa,ya,err=1,tol;
int i=1,n;
cout<<"Enter the starting interval = ";
cin>>x1;
cout<<"Enter the limiting interval = ";
cin>>x2;
cout<<"Enter the tolerance = ";
cin>>tol;
cout<<"Please enter the number of itterations
= ";
cin>>n;
cout<<"x1\tx2\ty1\ty2\txa\tya\terr\tCheck\n";
cout<<"___________________________\n";
while (err>tol&&i<=n)
{
y1=x1*x1-21;
y2=x2*x2-21;
xa=(x1*y2-x2*y1)/(y2-y1);
err=(xa-x1)/xa;
ya=xa*xa-21;
if (ya*y1<0)
{
y2=ya;
x2=xa;
}
else
{
y1=ya;
x1=xa;
}
cout<<setprecision(4)<<x1<<"\t"<<x2<<"\t"
<<y1<<"\t"<<y2<<"\t"<<xa<<"\t"<<ya<<"\t"
<<err<<"\t";
if (err>tol)
cout<<"No Convergence\n\n";
else
cout<<"Convergence\n\n";
i++;
}
cout<<"_______________\n";
cout<<"The root value of equation =
"<<xa<<endl;
getch();
}
```

NEWTON-RAPHSON METHOD

Fundamentals: Newton-Raphson method is also an iterative method to find root of a non-linear function. The basic idea of this method is to consider only one initial guess value X_0 (instead of two end values of an interval as used in bisection and false position methods) and then to improve it to obtain the new guess X_a by using the relation $X_a = X_o - f''(X_0) / f''(X_0)$. In every step of improvement, X_a of earlier step is considered as X_0 for the present step. When the new guess value X_a does not differ from the old guess value X_0 then it coincides with the exact root X_r.

Derivation of formula for root: The above formula for X_a can be derived from the Taylor's series of f(X) about X_o. If X_a is the correct root and $X_a=X_o+h$ then $f(X_a) = f(X_o+h)=0$. Expending using Taylor series we can write

$$f(X_o) = h\acute{f}(X_o) + \frac{h^2}{2} f''^{(X_o)} + \cdots = 0$$

Neglecting second and higher order terms in step size h we have

$$f(X_o) + h\acute{f}(X_o) = 0$$

$$h = -\frac{f(X_o)}{f'(X_o)}$$

Hence new approximation of root is obtained by substituting for h in $X_a=X_0+ h$ as

$$X_a = X_o - \frac{f(X_o)}{f'(X_o)}$$

This is the required Newton-Raphson method formula for new guess to obtain the root of a given function f(X).

Geometrical interpretation: The process of Newton-Raphson method can be presented graphically as shown in fig. In this plot R is the point of intersection of the function curve with X-axis. This point represents the root of the function with root value X_r.

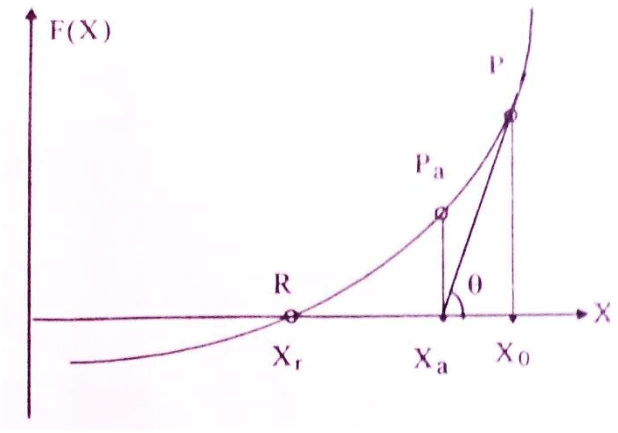


Fig. Newton-Raphson method

Let P be a point close to R representing approximately the root point and having X value as X_o. if we draw a tangent to the function curve at P that meets the X-axis at a point with $X=X_a$ then the slope of this tangent line is given by the relation

$$tan\theta = \frac{f(X_o) - 0}{X_o - X_a}. its\ inversion\ gives$$

$$X_a = X_o - \frac{f(X_o)}{tan\theta}. But\ tan\theta = f'(X), hence$$

$$X_a = X_o - \frac{f(X_o)}{f'(X_o)}$$

Thus, geometrically, finding new guess X_a from X_o in Newton-Raphson method is same as finding X value of the point of intersection of X-axis and the tangent at X_o.

Guess criteria: The method requires an initial guess not of the interval $[X_1,X_2]$ but of single value of X which must be near the real root. Such value of X can be found by graphical sketch of given function against X. The X value of the point where function curve intersects or crosses X-axis is the root. The initial guess value X_o can be on either side of this intersection X value. The guess of next X_a is obtained from X_0 with the help of formula

$$X_a = X_o - \frac{f(X_o)}{f'(X_o)}$$

Termination criteria: One can continue the process of finding new guess of root till the new guess of root satisfies f (X) = 0 exactly. Many times to arrive at this situation hundreds of steps will be required or the situation may not be arrived at all due to limitation of precision in number representation on the computer. Hence the iterative process of finding new guess is stopped when $|f(X)| \leq$ some specified limit € or error between X_a and X_0 is less than some specified limit which can be tested by checking if relative error in root is less than specified limit e.

i.e. $| (X_a - X_o) / X_a | \leq e$

The iterative process will be stopped by either $|f(X)| \leq €$ or $| (X_a - X_o) / X_a | \leq e$ whichever is satisfied earlier.

An additional situation that has to be taken into consideration is the one when $\acute{f}(X)$ is very small, almost nearly zero. In such cases next new guess becomes infinite. Thus iterative process must be stopped if magnitude of $\acute{f}(X)$ is less than or equal to some small value.

Steps in the method: The program of finding root of a function f(X) by Newton-Raphson method will consist of the following steps.

i. Input the initial guess value and limit of error.
ii. Initialization of iteration counter, approximate root and error.
iii. Check if error reached its limiting value or iteration number crossed its limit; if so then go to step (v).
iv. Decide new guess of root with its possible error and go to step (iii).
v. Output the result.

Computational effort: It is normally measured in terms of number of evaluations of function values (it being most time-consuming operation). In this method there are two function value evaluations in one iterative step one of the function and one for its derivative and hence the method is with more computational effort.

The need for calculation of derivative is displeasure of the method. In cases where closed form expression of derivative is available, the expressions may be long and much computational effort may be required to find its value. When closed form expression of derivative is not available one has to find it by numerical methods, which will lead to additional numerical error.

Rate of convergence: The idea about the fastness of the method to arrive to the root or fastness of reduction in the error in the approximate result is provided by the rate of convergence. It is the largest integer k such that

$$\lim_{i\to\infty}\frac{E_i+1}{E_i^k}\leq M$$

where M is a finite number. For a process with rate of convergence k, error E in any step is proportional to kth power of error in the previous step.

Let the successive guess values of roots be $X_i = X_r + e_i$ and $X_{i+1} = X_r + e_{i+1}$ with e_i and e_{i+1} as errors. In Newton-Raphson method these are related by $X_r + e_{i+1} = X_r + e_i - \frac{f(X_r+e_i)}{f'(X_r+e_i)}$.

$$e_{i+1} = e_i - \frac{f(X_r)+e_i f'(X_r)+\left(\frac{e_i{}^2}{2}\right)f''(X_r)+\cdots}{f'(X_r)+e_i f''(X_r)+\left(\frac{e_i{}^2}{2}\right)f'''(X_r)+\cdots}$$

Since $f(X_r) = 0$, therefore

$$e_{i+1} = e_i - \frac{e_i f'(X_r)+\left(\frac{e_i{}^2}{2}\right)f''(X_r)+\cdots}{f'(X_r)+e_i f''(X_r)+\left(\frac{e_i{}^2}{2}\right)f'''(X_r)+\cdots}$$

$$e_{i+1} \approx e_i \left\{ 1 - \frac{1 + \frac{\left(\frac{e_i}{2}\right) f''(X_r)}{f'(X_r)} + \cdots}{1 + e_i f''(X_r) + \frac{\left(\frac{e_i}{2}\right) f'''(X_r)}{f'(X_r)} + \cdots} \right\}$$

$$= e_i \left\{ 1 - \left[1 + \frac{\left(\frac{e_i}{2}\right) f''(X_r)}{f'(X_r)}\right]\left[1 + \frac{e_i f''(X_r)}{f'(X_r)}\right]^{-1} \right\}$$

$$= \frac{{e_i}^2}{2} \frac{f''(X_r)}{f'(X_r)}$$

If $\frac{f''(X_r)}{f'(X_r)}$ is finite and non-zero then comparison of limit of above equation with the equation defining order of convergence we get k=2. Physically this leads to doubling of significant figures in approximation during each iteration.

Stability: To test stability of Newton-Raphson one must check the certainty of convergence in the method. In the graphical representation successive approximations in Newton-Raphson method are indicated by points P_0, P_1, P_2, etc. On the function curve with X values X_0, X_1, X_2 etc., respectively, as shown in (fig.). The exact root is shown by point R. It is clear that the successive approximation.

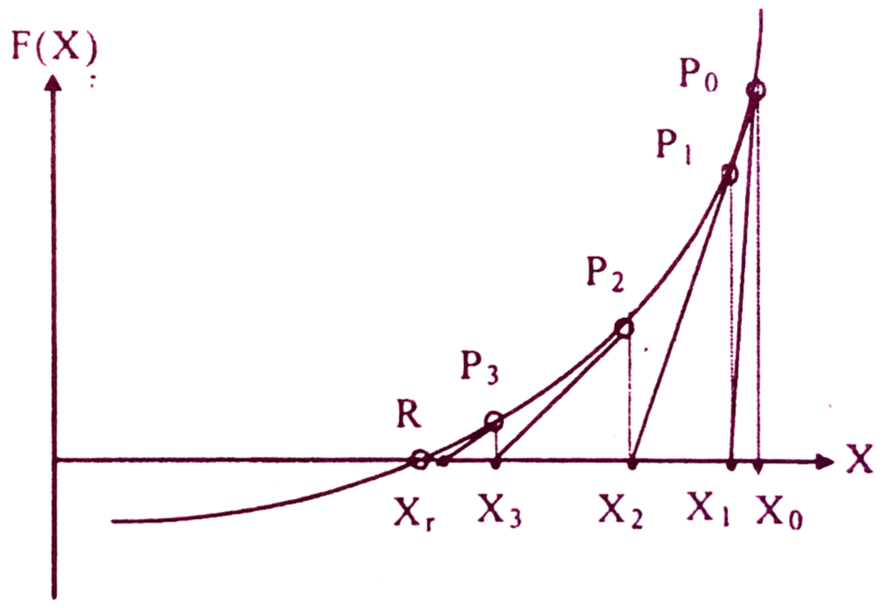


Fig. Stability of Newton-Raphson method

points to approach R and initial root guess X_0 approaches the exact root value X_r. Every step of the method brings the approximate root closer to the exact root and indicates convergence of the method.

Limitations: There are certain limitations of Newton-Raphson method. Some of them are given below:

a) If the value of gradient at guess point is very small then even a very small change in the gradient value of function has very large effect on the next value of guess and the next guess may be taken away from the exact root.

b) Above figure shows graphical representation of Newton-Raphson method for another function and it indicates non-convergence. If P_o is selected as initial guess point with X_o as initial guess root, the next guess root value is X' which is more away from X_r than X_o and hence the **iteration is divergent**.

c) Another situation of non-convergence is also shown in fig. below. If P_1 is used as initial guess point then the next guess point is P_2 whose next guess point is again P_1 and the process becomes oscillatory in nature. it leads to endless cycle of fluctuations between P_1 and P_2 without convergence. Hence if there is a change in sign of gradient at guess point and gradient at root point then Newton-Raphson method fails.

The above discussion indicates that Newton Raphson method, in general, provides no guarantee of convergence if initial guess is not appropriate. Stability of the method depends on the initial guess.

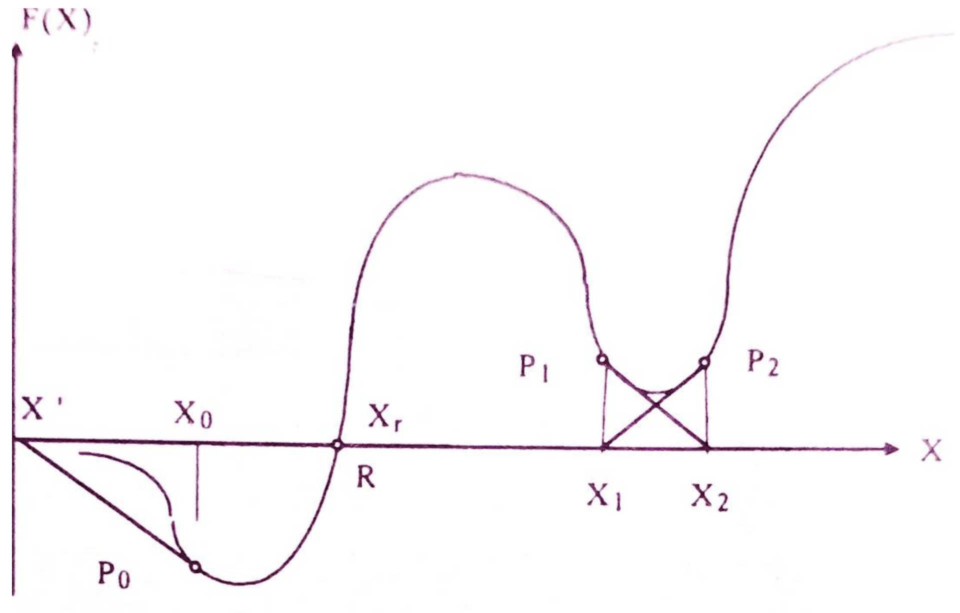


Fig. Instability with Newton-Raphson method

Efficiency: This method decides the new value of approximate root not by merely finding central point but by guessing the root point as point of intersection of tangent with X axis. This takes into consideration the nature of function variation and speeds up the action of reaching correct root and provides more efficient action.

Algorithm: In writing the algorithm of the Newton-Raphson method X_o is considered as starting guess root value and X_a as new guess value of the root with X as present root value. Every iteration consists of guessing the new guess value and checking for the required convergence. Since initially the guess value is already known from input, the part of guessing the root value is avoided in the first iteration. The iterative procedure is stopped by checking the relative error in the value of Xa.

```
Reading input data
  READ Xo, eps, N
Initialization of iteration counter, approximate root and error
  Xa ← Xo
  ER ← 1.5* eps
  I ← 1
Checking for convergence and deciding new root
  WHILE ER > eps AND I <= N DO
    X ← Xa
    Y ← f(X)
    Y' ← f '(X)
    Xa ← X -Y/ Y'
    ER ← |(Xa-X) / Xa|
    I ← I + 1
  END OF WHILE
Output of results
  WRITE "root = "; Xa, " with error = "; ER
  IF I > N THEN
      WRITE " No convergence "
  ELSE
      WRITE "Convergence "
  END
```

The limit of this error is taken as `eps'. The algorithm also makes the provision of stopping the iterative procedure if number of iterations exceeds some number (say N). In this case last obtained root value is printed along with its possible error and warning of "no convergence". The values X_o, eps and N are the inputs. In the following algorithm additional Print statement may be used to print intermediate steps values.

Example: Using Newton-Raphson method find the root of X^2-21=0 for given initial guess as 2.0 with relative error 0.01. Terminate the program if number of iterations exceeds 10.

Answer: The example is for finding root by Newton-Raphson method. In this example f(X)= X^2-21; f ' (X) = 2 X and X= 2.00 . The convergence check is by relative error = 0.01. The formula for error is: error = (Xa-X)/Xa .

Iter	Change	X	Y	YD	XA	YA	Error	Check
1	---	2.000	-17.000	4.000	6.250	18.063	0.680	No Convergence
2	$X_a \rightarrow X$	6.250	18.063	12.500	4.805	2.088	0.301	No Convergence
3	$X_a \rightarrow X$	4.805	2.088	9.610	4.588	0.047	0.047	No Convergence
4	$X_a \rightarrow X$	4.588	0.047	9.175	4.583	0.000	0.001	Convergence

Root value is = 4.58

Question 1: Using Newton-Raphson method find the root of X^3-10 = 0 for given initial guess as 2.0 with percentage error 1.

Answer: 2.15

Question 2: Using Newton-Raphson method finds the root of X^3- 4X + 1= 0 for initial guess as 1.0 with absolute error 0.02. Terminate the program if iteration number exceeds 10.

Answer: 1.861

Question 3: Using Newton-Raphson method find the root of X sin(X) -3 cos(X) = 0 for initial guess as 1.0 with accuracy of 2 digits after decimal point. Terminate the program if number of iterations exceeds 10.

Answer: 1.192

Question 4: Using Newton-Raphson method find the root of 0.4*X^2-39=0 for initial guess as 9.0 with percentage error 1. Terminate the program if number of iterations exceeds 10.

Answer: 9.87

Question 5: Using Newton-Raphson method find the root of e^x – 2 = 0 for initial guess as 0.0 correct up to 2 significant figures. Terminate the program if number of iterations exceeds 10.

Answer: 0.693

Question 6: Derive the formula for guess of root of a function in Newton-Raphson method. Give the geometrical interpretation of the formula.

Question 7: Show that the rate of convergence of Newton-Raphson method is two. How is it related with significant figures of the result?

Question 8: Draw the flow chart of Newton-Raphson method. Mention the facilities provided in it.

Question 9: With the help of graphical representation of Newton-Raphson method discuss stability of the method.

Question 10: Give algorithm of Newton—Raphson method to find root of a transcendental equation. Explain the variables used and mention the facilities provided in it.