**Unit 1 Introduction to Numerical Computing**

**Introduction**

Numerical computations play an indispensible role in solving real life mathematical, physical and engineering problems. They have been in use for centuries even before digital computers appeared on the scene. The advent of digital computers has, however, enhanced the speed and accuracy of numerical computations.

Numerical computing is an approach for solving complex mathematical problems using only simple arithmetic operations. The approach involves formulation of mathematical models of physical situations that can be solved with arithmetic operations. It requires development, analysis and use of algorithms.

Numerical computations invariably involve a large number of arithmetic calculations and therefore, require fast and efficient computing devices. The development of high power and low cost computers has had a profound impact on the application of numerical computing methods to solve scientific problems.

**Numerical computing usually deal with the following topics.**

1. Finding roots of equations
2. Solving systems of linear algebraic equations
3. Interpolation and regression analysis
4. Numerical integration
5. Numerical differentiation
6. Solution of differential equations
7. Boundary value problems
8. Solution to matrix problems

**Numeric Data**

Numerical computing may involve two types of data, namely discrete data and continuous data. Data that are obtained by counting are called discrete data. Examples of discrete data are the total number of items in a box or the total number of people participating in a race

Data that are obtained through measurement are called continuous data. Examples of continuous data are the speed of a vehicle as given by a speedometer, or temperature of a patient as measured by a thermometer.

**Process of Numerical Computing**

The process of numerical computing can be roughly divided into the following four phases.

1. Formulation of Mathematical Model
2. Construction of an appropriate numerical method
3. Implementation of the method to obtain a solution
4. Validation of the solution

The formulation of a suitable mathematical model is a critical to the solution of the problem. A mathematical model can be broadly defined as a formulation of a certain mathematical equation that expresses the essential features of a physical systems or processes. Model may range from a simple algebraic equation to a complex set of differential equations.

Modeling is the process of translating a physical problem into a mathematical problem. The process involves

1. Making a number of simplifying assumptions
2. Identification of important variables
3. Postulation of relationship between the variables

Once a mathematical model is available, our first step would be to try to obtain an explicit analytical solution. In most of the cases, the mathematical models may not be amenable to analytical solutions or they may not be solved efficiently using analytical techniques. In such cases, we have to construct appropriate numerical methods to solve mathematical models. Numerical methods is a computational technique which involves only a finite number of basic arithmetic operations.

The third phase of the numerical computing is computing process is implementation of the method selected. This phase is concerned with the following three tasks.

1. Design of an algorithm
2. Writing of the problem
3. Executing it on a computing to obtain the result

Once we are able to obtain the results, the next step is the validation of the process. Validation means the verification of the results to see that it is within the desired limits of accuracy. It is not then we must go back and check each of the following

1. Mathematical model itself
2. Numerical method selected
3. Computational algorithm used to implement the model

This may mean modification of the model, selection of an alternative numerical method or improving the algorithm (or a combination of them). Once a modification is introduced, the cycle begins again.

**Characteristics of Numerical Computing**

Numerical methods exhibit certain computational characteristics during their implementation. It is important to consider these characteristics while implementing. It is important to consider the characteristics that are critical to the success of implementation. These characteristics are:

**Accuracy:** Every method of numerical computing introduces errors. They must be either due to using an approximation in place of an exact mathematical procedure and manipulation of numbers in the computer. These errors affect the accuracy of the results. The results we obtain must be sufficiently accurate to serve the purpose for which the mathematical model was built. Choose of a method is, therefore, very much dependent on the particular problem.

**Rate of Convergence:** Many numerical methods are based on the idea of iterative process. This process involves generation of a sequence of approximations with the hope that the process will converse the required solution. Certain methods converse faster than others. Some methods may not converse at all. It is therefore, important to test for convergence before a method is used. Rapid convergence takes less execution time on computer

**Numerical Stability:** Another problem introduced by some numerical computing methods is that of a numerical instability. Errors introduced into a computation, from whatever source, propagate in different ways. In some cases, these errors tend to grow exponentially with disastrous computational results. A computing process that exhibits such exponential error growth is said to be numerically unstable. We must choose methods that are not only fast but also stable

Numerical instability may also arise due to ill-conditioned problems. There are many problems which are inherently sensitive to round off errors and other uncertainties. When the problem is ill-conditioned, there is nothing we can we can do to make a method to become numerically stable.

**Efficiency:** Once more consideration in choosing a numerical method for solution of a mathematical model is efficiency. It means the amount of effort required by both human and computer to implement the method. A method that requires less computing time and less programming effort and yet achieves the desired accuracy is always preferred.

**Types of Errors in Numerical Computing**

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

**True Error**

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

True Error = True Error – Approximate value

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

f’(x) =

For f(x) = 7e0.5x and h=0.3, find the approximate value of f’(2), the true value of f’(2), and the true error.

Solution:

Approximate value of derivative is calculated by using formula

f’(x) =

Now, for x=2, h = 0.3

f’(x) = = 10.265

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

f(x) = 7e0.5x

f’(x) = 7\*0.5\*e0.5x = 3.5e0.5x

So, the true value of f’(2) = 3.5 e0.5\*2 = 9.5140

Now, true error is calculated as

Et = True value-Approximate value

9.5140-10.265 = 0.75061

**Relative Error**

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

Relative Error =

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

f’(x) =

For f(x) = 7e0.5x and h=0.3, find the relative true error at x=2

Solution: From above example, Et = True value – Approximate value

9.5140-10.265

Now, relative error is calculated as

Er =

Er = = -0.078895

Relative errors are also presented as percentages. For example,

Er = -0.078895\*100 = -7.58895%

Absolute relative error may also need to be calculated. In such as cases,

|Er| = |-7.58895| = 7.58895%

**Approximate Error**

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

Approximate Error = Present Approximation – Previous Approximation

**Sources of Error**

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

1. **Round off Error** : A computer can only represent a number approximately. For example, a number like 1/3 may be represented as 0.33333 on a PC. Then the round off error in this case is 1/3-0.333333 = 0.00000033. There are also other numbers that cannot be represented exactly. For example, √2, π are numbers that need to be approximated in computer calculations.
2. **Truncation Error:** Truncation error is defined as the error caused by truncating a mathematical procedure. For example, the Maclaurin series for ex is given as

ex = 1+x++

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

ex = 1+x+

The truncation error for such an approximation is

Truncation error = ex – (1+x+)

**Propagation of Errors**

Numerical computing involves a series of computations consisting of basic arithmetic operations. Therefore, it is not the individual round off errors that are important but the final error on the result. Our major concern is how an error at one point in the process propagates and how it effects the final total error. Here we will see the arithmetic of error propagation and its effects.

**Addition and Subtraction**

Consider addition of two numbers, say x and y,

xt+yt = xa+ex+ya+ey

= (xa+ya)+(ex+ey)

Therefore,

Total Error = ex+y = ex+ey

Similarly, for subtraction

Total Error = ex-y = ex-ey

The addition of errors does not mean that error will increase in all cases. It depends on the sign of individual errors. Similar is the case with subtraction

Since, we do not normally know the sign of errors, we can only estimate errors bounds. That is , we can say that

| ex+y| ≤ |ex|+|ey|

Therefore, the rule for addition and subtraction is: the magnitude of the absolute error of a sum (or difference) is equal to or less than the sum of the magnitudes of the absolute errors of the operands.

This inequality is called triangle inequality. The equality applies when the operands have the same signs and the inequality applies if the signs are different.

**Multiplication:**

Here, we have

xt\*yt = (xa+ex)\*(ya+ey) = xaya+yaex+xaey+exey

Errors are normally small and their products will be much smaller. Therefore, if we neglect the product of the errors, we get

xt\*yt = xaya+xaey+yaex

= xaya+xaya(ex/xa+ey/ya)

Then total error = exy = xaya(ex/xa+ey/ya)

**Division**

We have,

=

Multiplying both numerator and denominator by ya-ey and rearranging the terms we get

=

Dropping the terms that involve only product of errors, we have

**Review of Taylor Theorem**

Taylor’s theorem is given by

f(x+h) = f(x)+f’(x)h+h2+h3………………………….. (1)

Provided that all derivatives of f(x) exist and are continuous between x and x+h.

Equation (1) can also be written as

f(h+x) = f(h)+f’(h)x+x2+x3…………………………..

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

f(x) = f(h)+f’(0)x+x2+x3………………………….. (3)

Equation (3) is called Maclaurin series

Example: Derive the Maclaurin series of sin x = x – +-+…………………..

Solution: Maclaurin series of Taylor series for the point h = 0

f(x) = sin x🡪 f(0) = 0

f’(x) = cos x 🡪f’(0)=1

f’’(x) = -sinx -🡪f’’(0) = 0

f’’’(x) = -cos x = 🡪f’’’(0) = -1

f’’’’(x) = sin x = 🡪f’’’’(x) = 0

f’’’’’(x) = = cos x🡪f’’’’’(x) = = 1

Using the Maclaurin series now,

f(x) = f(0) + f’(0)x+f’’(0)x2/2+f’’’’(0)x3/3!+f’’’’(0)x4/4!...

f(x) = 0+1.x+0.x2/2-1.x3/3!+0.x4/4!...

f(x) = x-x3/3!+x5/5!

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

Example: Find the value of e0.25 using the first five terms of Maclaurin series

Solution:

**Error in Taylor Series**

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

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

f(x+h) = f(x)+f’(x)h+h2+h3+………fn(x) hn/n! + Rn(x)

where the remainder is given by

Rn(x) = f(n+1)(c )

Where

X<c<x+h

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

Example:

**The Mean-Value Theorem**

The **Mean Value Theorem** is one of the most important theoretical tools in Calculus. It states that if *f*(*x*) is defined and continuous on the interval [*a*,*b*] and differentiable on (*a*,*b*), then there is at least one number *c* in the interval (*a*,*b*) (that is *a* < *c* < *b*) such that

\begin{displaymath}f'(c) = \frac{f(b) - f(a)}{b-a} \cdot\end{displaymath}

The special case, when *f*(*a*) = *f*(*b*) is known as **Rolle's Theorem**. In this case, we have *f* '(*c*) =0. In other words, there exists a point in the interval (*a*,*b*) which has a horizontal tangent. In fact, the Mean Value Theorem can be stated also in terms of slopes. Indeed, the number

\begin{displaymath}\frac{f(b) - f(a)}{b-a} \end{displaymath}

is the slope of the line passing through (*a*,*f*(*a*)) and (*b*,*f*(*b*)). So the conclusion of the Mean Value Theorem states that there exists a point $c\in(a,b)$ such that the tangent line is parallel to the line passing through (*a*,*f*(*a*)) and (*b*,*f*(*b*)).

**Example.** Let $f(x) = \displaystyle \frac{1}{x}$, *a* = -1and *b*=1. We have

\begin{displaymath}\frac{f(b) - f(a)}{b-a} = \frac{2}{2} = 1.\end{displaymath}

**Roots of Nonlinear Equations**

An equation is a statement that expresses the equality of two mathematical expressions. An equation has an equal sign, a right side expression and a left side expression.

For example x+y -10 = 0 is an equation

Equation may belong to one of the following types of equations:

1. Algebraic Equation ( 3x+5y-21=0)
2. Polynomial Equation (5x5-x3+3x2=0)
3. Transcendental Equations (2sinx-x = 0)

**Root:** A real number x will be called a solution or a **root** ofequation if it satisfies the **equation.**

For example, **2** is the root of the equation x2-4x+4 =0

**Method of Solution**

There are a number of ways to find the roots of non-linear equations. They include

1. Direct analytical methods
2. Graphical Methods
3. Trial and Error Methods
4. Iterative Methods

**Direct Analytical Method:** In certain cases, roots can be found by using direct analytical methods. For example, consider a quadratic equation such as

ax2+bx+c = 0…………………..**(1)**

We know that the solution of this equation is

..…………………….**(2)**

This equation **(2)** gives the two roots of equation **(1)**.

However, there are equations that cannot be solved by analytical methods. For example, the simple transcendental equation

2sinx –x = 0

Cannot be solved analytically

Direct methods for solving non-linear equations do not exist except for certain simple cases.

**Graphical Methods:**

Graphical methods are useful when we are satisfied with approximate solution for a problem. This method involves plotting the given function and determining the points where it crosses the x-axis. These points represent approximate values of the roots if the function.

**Trial and Error method**

Trial and error method involves a series of guesses for x, each time evaluating the function to see whether it is close to zero. The value of x that causes the function to value closer to zero is one of the approximate roots of the equation. This method is time consuming method.

**Iterative Method**

With the advent of computers, algorithmic approaches known as iterative methods have become popular. An iterative technique usually begins with an approximate value of the root, known as the initial guess, which is then successively corrected iteration by iteration. The process of iteration stops when the desired level of accuracy is obtained. Since iterative methods involve a large number of iterations and arithmetic operations to reach a solution, the use of computers has become inevitable to make the task simple and efficient.

Iterative methods, based on the number of guesses they use, can be grouped into two categories:

1. Bracketing Methods
2. Open end Methods

**Bracketing Method:** Bracketing method starts with two initial guesses that “bracket” the root and then systematically reduce the width of the bracket until the solution is reached. Two popular methods under this category are:

1. Bisection Method
2. False Position Method

**Open End Method :** Open end Methods use a single starting value or two values that do not necessarily bracket the root. The following methods fall under this category

1. Newton Raphson Method
2. Secant Method
3. Muller’s Method
4. Fixed point Method
5. Bairstow’s Method

**Largest Possible Root:** For a polynomial represented by f(x) = anxn+an-1xn-1+…..+a1x+a0

The largest possible root is given by

x\* =

This value is taken as the initial approximation when no other value is suggested by the knowledge of the problem at hand.

**Search Bracket:** The important relationship that might be useful for determining the search intervals that contain the real roots of a polynomial is

|x| ≤√()2 – 2(()

**Example:** Consider the polynomial equation 2x3-8x2+2x+12=0. Estimate the possible initial guess values.

**Solution:** The largest possible root is:

x = -8/2 = 4

That is, no root can be larger than the value 4

All roots must satisfy the relation

|x| ≤√()2 – 2(() = √14

Therefore all real roots lie in the interval (-√14, √14).

We can use these two points as the initial guesses for the bracketing methods and one of them for the open end methods.

**Stopping Criterion**

An iterative process must be terminated at some stage. When? We must have an objective criterion for deciding when to stop the process. We may use one or combination) of the following tests depending on the behavior of the function to terminate the process.

1. |xi+1-xi| ≤ Ea (Absolute Error in x)
2. ≤ Er (Relative error in x)
3. |(f(xi+1)| ≤ E (Value of the function at root)
4. |(f(xi+1)-f(xi)| ≤ E (Difference in function values)

**Bisection Method (Half Interval Method)**

The bisection method is one of the simplest and most reliable iterative methods for the solution of non-linear equations. This method, also known as binary chopping or half interval method , relies on the fact that if f(x) is real and continuous in the interval a<x<b and f(a) and f(b) are of opposite signs that is,

f(a)\*f(b)<0

then there us at least one real root in the interval between a and b

(There may be more than one root in the interval)

Let x1 = a and x2 = b. Let us also define another point x0 to be the mid point between a and b. That is,

x0 = (a+b)/2

Now, there are exists the following three conditions:

1. If f(x0) = 0, we have a root at x0.
2. If f(x0)\*f(x1)<0 then there is a root between x0 and x1
3. If f(x0)\*f(x2)<0 then there is a root between x0 and x2

It follows that by testing the sign of the function at midpoint, we can deduce which part of the interval contains the root. We can further divide this subinterval into two halves to locate a new subinterval containing the root. This process can be repeated until the interval containing the root is as small as we desire

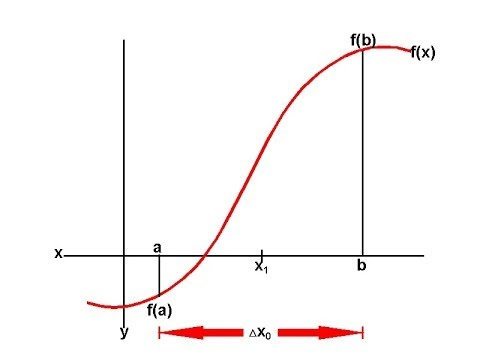


Figure: Illustration of bisection method

Example 1: Find a root the equation x2-4x-10=0

Solution: First we calculate the values that bracket the root

Thus,

|

|x| ≤√()2 – 2(() = 6

Therefore, we have both the roots in the interval (-6,6). The table below gives the values of f(x) between -6 and 6.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | -6 | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| f(x) | 50 | 35 | 22 | 11 | 2 | -4 | -10 | -13 | -14 | -13 | -10 | -5 | 2 |

From above table, we see that is a root in the interval (-2,-1) and (5,6)

**Iteration 1**

Let us take x1 = -2 and x2 = -1

Then x0 = (-2-1)/2 = -1.5

Now,

F(-2) = 2 and f(1.5) = -1,75

Since f(-2)\*f(-1.5)<0, the root must be in the interval (-2,-1.5).

**Iteration 2**

x1 = -2, x2 = -1.5

x0 = (-2-1.5)/2 = -1.75

Now f(-1.75) = 0.0625

Since f(1.75) and f(-1.5) are opposite sign, the root lies in the interval (-1.75, -1.5)

**Iteration 3**

x1 = -1.75 and x2 = -1.5

x0 = (-1.75-1.5)/2 = -1.625

Now, f(-1.625) = -0.859

Since f(-1.75) and f(-1..625) are of opposite sign, root lies in the interval (-1.75,-1.625)

**Iteration 4**

x1 = -1.75, x2 = -1.625 and x0 = (-1.75-1.625)/2 = -1.72

Now f(-1.72) = -0.1615

Since f(-1.75) and f(-1..72) are of opposite sign, root lies in the interval (-1.75,-1.72)

**Iteration 5**

x1 = -1.75, x2 = -1.72 and x0 = (-1.75-1.72)/2 = -1.735

Now f (-1.735) = -0.05

f(-1.75) and f(-1.735) are of opposite sign, root lies in the interval (-1.75,-1.735)

**Iteration 6**

x1 = -1.75, x2 = -1.735 and x0 = (-1.75-1.735)/2 = -1.7425

Now f(-1.7425) = 0.0063

f(-1.7425) and f(-1.735) are of opposite sign, root lies in the interval (-1.7425,-1.735)

**Iteration 7**

x1 = -1.7425 , x2 = -1.735 and x0 = (-1.7425-1.735)/2 = -1.7416

Approximate root is **-1.7416**

**Algorithm Bisection Method**

Step 1: Start

Step 2: Decide initial values fox x1 and x2 and stopping criterion E

Step 3: Compute f1 = f(x1) and f2 = f(x2)

Step 4: if f1\*f2>0, x1 and x2 do not bracket any root and go to step 9 otherwise continue

Step 5: Compute x0 = (x1+x2)/2 and compute f0 = f(x0)

Step 6: If f1\*f2<0 then

Set x2 = x0 and f2=f0

Else

Set x1 = x0 and f1 = f0

Step 7: If absolute value of (x2-x1)/x2 is less than E then

Root = (x1+x2)/2

Write the value of root

Go to step 9

Else

Go to step 5

Step 9: Stop

**C program for Bisection Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define E 0.00001

#define F(x) x\*x-4\*x-10

int main()

{

float x1,x2,x0,f1,f2,f0,r;

printf("Enter initial guesses\n");

scanf("%f%f",&x1,&x2);

f1 = F(x1);

f2 = F(x2);

if(f1\*f2>0)

{

printf(" Initial guesses do not bracket roots\n");

}

else

{

while(1)

{

x0 = (x1+x2)/2;

f0 = F(x0);

if(f1\*f0<0)

{

x2 = x0;

f2 = f0;

}

else

{

x1 = x0;

f1 = f0;

}

if(fabs(x2-x1)<E)

{

r = x2;

break;

}

}

}

printf("The root = %f\n",r);

printf("f(%f) = %f\n",r,F(r));

getch();

return 0;

}

**Convergence of Bisection Method**

In the bisection method, we choose a midpoint x0 in the interval between x1 and x2. Depending on the sign of function f(x0), f(x0) and f(x2), x1 or x2 is to set to x0 such that new interval contains the root. In either case, the interval containing the root is reduced by a factor of 2. The same procedure is repeated for the new interval. The procedure is repeated for the new interval. If the procedure is repeated n times then the interval containing the root is reduced to the size

= ∆x/2n

After n iterations, the root must lie within ±∆x/2n of our estimates. This means that the error bound at nth iteration is = En = |∆x/2n|

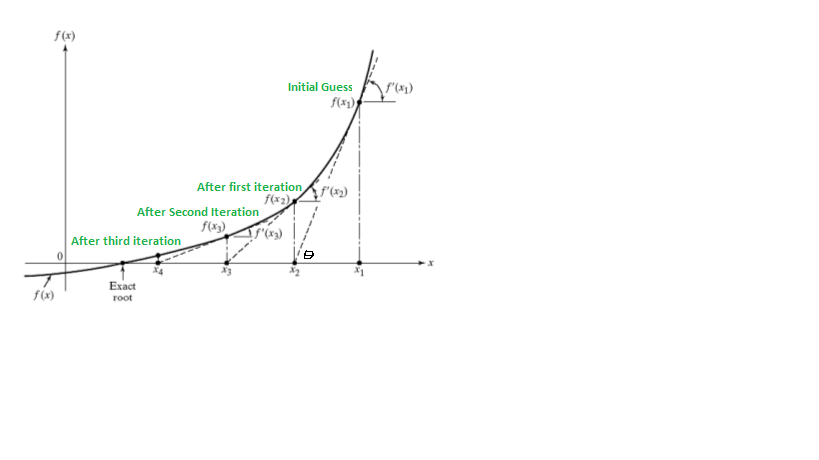
Similarly,

En+1 = |∆x/2n+1| = En/2

That is, the error decrease linearly with each step by a factor of 0.5. The bisection method is therefore, linearly convergent

Since the convergence is slow to achieve a high degree of accuracy, a large number of iterations may be needed. However, the bisection algorithm is guaranteed to converge

**Newton Raphson Method**

****

Let us consider the graph given as shown above. Let is assume that x1 is an appropriate root of f(x) = 0. Draw a tangent at the curve f(x) at x = x1 as shown in the figure. The point of intersection of this tangent with x-axis gives the second approximation to the root. Let the point of intersection be x2. The slope of the tangent is given by

tanθ = = f’(x1)

where f(x1) is the slope of f(x) at x = x1. Solving for x2, we obtain

x2 = x1-

This is called Newton Raphson formula

The next approximation would be

x3 = x2-

In general,

xn+1 = xn-

This method of successive approximation is called the Newton-Raphson method. The process will be terminated when the difference between two successive values is within a prescribed limit.

Perhaps the most widely of all the methods for finding roots is the Newton-Raphson method.

**Newton Raphson Algorithm**

Step 1: Start

Step 2: Input initial guess x0

Step 3: Find the improved estimate of x0 as

x1 = x0 - f(x0)/f’(x0)

Step 4: Check the accuracy of the latest estimate. Compare relative error to a predefined value E, If |(x1-x0|<E then goto step 6

Step 5: Replace x0 by x1 and repeat steps 3 and 4

Step 6: Stop

**C program for Newton Raphson Method**

#include<conio.h>

#include<stdio.h>

#include<math.h>

#define F(x) x\*x+4\*x-9

#define D(x) 2\*x+4

#define E 0.001

int main()

{

float x0,x1,f1,f2,r;

int i=1;

printf("Enter intial guess\n");

scanf("%f",&x0);

while(1)

{

f1 = F(x0);

f2 = D(x0);

x1 = x0-f1/f2;

if((fabs(x1-x0)/x1)<E||i>=20)

{

r=x1;

break;

}

x0 = x1;

}

printf("The root = %f",r);

getch();

return 0;

}

Enter initial guess

4

The root = 1.605551

**Example:** Find the root of the equation f(x) = x2-3x+2 in the vicinity of x=0 using Newton-Raphson method

Solution: here f(x) = x2-3x+2

f’(x) = 2x-3

Let x1 = 0 (first approximation)

x2= x1 -

x2 = 0- 2/-3 = 0.6667

Similarly,

x3= x2 - = 0.6667-0.4444/-1.6667 = 0.9333

x4= x3 - = 0.9333- 0.07/-1.334 = 0.9959

x5= x4 - = 0.9959-0.0041/-1.00082 = 0.99999

x6= x5 - = 0.99999-0.00001/-1.0002 = 1.0000

Since F(1.00) = 0, the root closer to the point x=0, is 1.00

**Convergence of Newton –Raphson Method**

Let xn be an estimate of a root of the function f(x). If xn and xn+1 are close to each other then using Taylor’s series expansion, we can state

f(xn+1) = f(xn)+f’(xn)(xn+1-xn)+f’’(R )(xn+1-xn)2/2…………………….(1)

Where R lies somewhere in the interval xn and xn+1 and third and higher order have been dropped.

Let us assume that the exact root of f(x) is xr. Then xn+1 = xr.

Therefore f(xn+1)=0 and substitute these values in equation (1) we get

0 = f(xn)+f’(xn)(xr-xn)+f’’(R )(xr-xn)2/2……………………….(2)

We know that the Newton’s formula is given by

xn+1 = xn-

Rearranging the terms w get

f(fn) = f’(xn)(xn-xn+1)

Substituting this value in equation (2) we get

O = f’(xn)(xr-xn+1)+ f’’(R )(xn+1-xn)2/2………………….(3)

We know that the error in the estimate xn+1 is given by

en+1 = xr-xn+1

Similarly, en = xr-en

Now equation 3 can be expressed in terms of these errors as

O = f’(xn)en+1 +f’’ (R )/2en2

Rearranging the terms we get,

en+1 = -f’’(R ) en2/2f’(xn)

This shows that the error is roughly proportional to the square of the error in the previous iteration. Therefore Newton Raphson method is said to have quadratic convergence

**Secant Method**

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

xn+1 = xn- ……………………………………(1)

From the above equation, it is clear that one of the drawbacks of Newton Raphson method is that we have evaluated the derivative of the function. When this value very small (non zero), the next iteration will be far worse approximation. To overcome this drawback, the derivative f’(x) of the function f(x) is approximately as:

f’(x) = ……………………………..(2)

Substitution equation (2) in (1), gives,

xi+1 = xi -

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

**Derivation of Secant Method from Geometry**

The secant method can also be derived from geometry as shown below. Let us consider two initial guesses xi and xi-1 and a straight line between f(xi) and f(xi-1) passing through the x-axis and xi+1. Now, slope of secant line passing through x1 and x2 is given by

=

Or f(x1)(x2-x1) = f(x2)(x1-x2)

Or x3[f(x2)-f(x1)] = f(x2)x1 - f(x1)x2

x3 =

By adding and subtracting f(x2)x2 to the numerator and rearranging the terms we get

x3 = x2-

This equation is called secant formula. If the secant line represents the linear interpolation polynomial of the function f(x) (with interpolating points x1 and x2 ) then x3, which intercepts the x-axis, represents the appropriate root of f(x)

The appropriate value of the root can be refined by repeating this procedure by replacing x1 and x2 by x2 and x3 respectively. That is, the next appropriate value is given by

x4 = x3-

This procedure is continued till the desired level of accuracy is obtained. We can express the secant formula in general form as follows:

xi+1 = x3-

**Algorithm secant method**

1. Start
2. Decide two initial points x1 and x2, accuracy level required E
3. Compute f1 = f(x1) and f2 = f(x2)
4. Compute x3 =x2-
5. Test for accuracy of x3

If |(x3-x1)/x3|>E then

Set x1 = x2 and f1 = f2

Set x2 = x3 and f2 = f(x3)

Goto step 3

Otherwise

Set root =x3 and print “root”

1. Stop

**C program for Secant Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define E 0.001

#define F(x) x\*x-4\*x-10

int main()

{

float x1,x2,x3,f1,f2,f3,r;

int i=0;

printf("Enter intial guesses\n");

scanf("%f%f",&x1,&x2);

while(1)

{

f1 = F(x1);

f2 = F(x2);

x3 = x2- (f2\*(x2-x1))/(f2-f1);

if(fabs((x3-x2)/x3)<E)

{

r = x3;

break;

}

else

{

x1 = x2;

x2 = x3;

}

i=i+1;

}

printf("The root = %f\n",r);

printf("f(%f) = %f\n",r,F(r));

printf("Number of iterations = %d\n",i);

getch();

return 0;

}

**Example 1:** Use the secant method to estimate the root of the equation

x2-4x-10=0

Solution: Given x1 = 4 and x2 = 2

f(x1) = f(4) = -10

f(x2) = f(2) = -14

Iteration 1:

x3 = x2-

or x3 = 2-

Iteration 2:

x1 = x2 = 2

x2 = x3 = 9

f(x1) = f(2) = -14

f(x2) = f(9) = 35

x3 = x2-

or x3 = 9-35(9-2)/(35+14) = 4

Iteration 3:

x1=9

x2=4

f(x1) = f(9) = 35

f(x2) = f(4) = -10

x3 = 4-(-10)(4-9)/(-10-35) = 5.1111

Iteration 4:

x1 = 4

x2 = 5.1111

f(x1) = f(4) = -10

f(x2) = f(5.1111) = -4.3207

x3 = 5.1111 + 4.3207(5.111-4)/(-4.3207-10) = 5.9563

Iteration 5:

x1 = 5.1111

x2 = 5.9563

f(5.1111) = -4.3207

f(x2) = f(5.9563) = 5.0331

x3 = 5.9563-5.0331(5.9563-5.1111)/(5.0331+4.3207) = 5.5014

Iteration 6:

x1 = 5.9563

x2 = 5.5014

(x1) = f(5.9539) = 5.0331

f(x2) = f(5.5014) = -1.7392

x3 = 5.5014 + 1.7392(5.5014-5.9563)/(-1.7392+5.0331) = 5.6182

The value can be further refined by continuing the process, if necessary

**Example 2:** Solve the equation 2x2+4x-10=0 using secant method with error precision 0.05

Solution:

**Example 3:** Solve the equation cos x+2sinx +x2 = 0 with error precision is 0.01

**Solution:**

**Convergence of Secant Method**

The secant formula of iteration is

xi+1 = xi - ……………………………..(1)

Let xr be actual root of f(x) and ei be the error in the estimate of xi

Then

xi+1 = ei+1+xr

xi = ei+xr

ei-1 = ei-1+xr

Substituting these values in equation (1) and simplifying we get the error equation as:

ei+1 =

According to mean value theorem, there exists at least one point, say x = Ri in the interval xi and xr such that

f’(Ri ) =

We know that

f(xr) = 0

xi-xr = ei

and therefore

f’(Ri) =

f(xi) = f’(Ri)\*ei

or Similarly,

f(xi-1) = ei-1f’(Ri-1)

Substituting these in the numerator of equation (1) we get

ei+1 =eiei-1

That is, we can say

ei+1 αeiei-1………………………………………….(2)

We know that the order convergence of an iteration process is p if

ei α epi-1

or ei+1 α epi

Substituting these values in equation (2), we get

epi α epi-1 ei-1

or

ei αei-1 (p+1)/p

Comparing the relations we get

p = (p+1)/p

That is,

p2-p-1 = 0

which has the solutions

p =

Since p is always positive, we have,

p =1.618

It follows that the order of convergence of the secant method is 1.618 and the convergence is referred to as super linear convergence

**Fixed Point method**

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

f(x) = 0………………….(1)

is written as

x = g(x) ……………………..(2)

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

Example 1: x2+3x-1=0 can be written as x =

Example 2: sin x =0 can be written as x = sin x +x

Since equation (1) and equation (2) are same, root of equation (2) is also root of equation (1). Root of equation (2) is given by point of intersection of straight line y = x and the curve x = g(x). This point of intersection is called fixed point of g(x). Point p is a fixed point of the function f(x) if and only if f(p) = p. For example, if function f is defined on the real numbers f(x) = x2-3x+4 then 2 is fixed point of f, because f(2) = 2

Equation (2) provides a convenient way of predicting the next value of x on the basis of current value of x. If x0 is the initial guess to the root, next approximated value of a root can be calculated as

x1 = g(x0)

Similarly, further approximation can be given as

x2 = f(x1)00

After generalization this, we get

xn+1 = g(xn)

This equation is called fixed point formula and gives rise to the sequence x0 , x1, x2,… which is hoped to converge to a point x (fixed point) as shown in figure below:

**Algorithm**

1. Start
2. Input initial guess (say x0) and precision say E
3. Convert f(x) = 0 to the form x = g(x)
4. Estimate new value of the root x2 as x1 = g(x0)
5. Find the absolute relative approximate error as |(x1-x2)/x1|<E then x0 = x1 and goto step 4 else goto step 6
6. Stop

**C program for Fixed point Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define E 0.001

#define F(x) x\*x-6\*x+8

#define G(x) x\*x-5\*x+8

int main()

{

float x0,x1,r;

int i=0;

printf("Enter intial guess\n");

scanf("%f",&x0);

while(1)

{

x1 = G(x0);

if((fabs(x1-x0)<E)||i>=30)

{

r=x1;

break;

}

else

{

x0 = x1;

}

i=i+1;

}

printf("The root = %f\n",r);

printf("The number of iterations =%d",i);

getch();

return 0;

}

**Example 1:** Solve the equation x2-6x+8 = 0 by assuming error precision 0.01

**Solution:** Above equation can be written as x = x\*x-5\*x+8

Assume that initial value of guess is 1

Iteration 1:

x0 = 1.00

x1 = g(1) = 4.00

Error = |(4-1)/4 = 0.7500

Iteration 2:

x0 = 4.00

x1 = G(x) = G(4) = 4.00

Error = |(4-4)/4| = 0

**Example 2**: Solve the equation 1+1/2 sin x – x = 0 by assuming error precision 0.001

**Solution:**

Above equation can be written as x = 1+1/2 sin x

Assume that initial guess as x0 = 0.

**Iteration 1:**

x0 = 1+1/2 sin 0 = 1

**Iteration 2:**

x0 = x1 = 1

x1= 1+1/sin 1 = 1.42

**Iteration 3**

x0 = x1 = 1.42

x1= 1+1/2sin 1.494 = 1.494

**Iteration 4:**

x0 = x1 = 1.494

x1 = 1+1/sin 1.494 = 1.4985

**Iteration 5:**

x0 = x1 = 1.4985

x1 = 1+1/2sin x = 1+1/2 sin 1.4985 = 1.4986

Error = |1.4986-1.4985|<0.001

Since, absolute value of error is less than error limit

Root = 1.4986

**Convergence of Fixed Point Method**

Fixed point formula is given by as

xn+1 = g(xn)

Let xf be the root of the given equation

xf = g(xf)

Subtracting above two equations we get

xf – xn+1 = g(xf) – g(xn) ………………………………..(1)

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

g’(t) =

or g(xf)-g(xn) = g’(t)(xf-xn)

Substituting this value in equation (1), we get

xf-xn+1 = g’(t)(xf-xn) ………………………………(2)

Since

en+1 = xf-xn+1

en = xf – xn

Equation (2) becomes

en+1 = g’(t)en …………………………………………….(3)

Now equation (2) becomes

en+1 = g’(t)en ………………………………………….(4)

Equation **(4)** says that error decreases with each iteration only of g’(t)<1. Hence, we say that fixed point method converges only under the condition g’(t)<1

**Synthetic Division**

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

**Step 1: Set up the synthetic division**

When you write out the dividend make sure that you write it in descending powers and you insert 0’s for any missing terms. For example, if you had the problem (x4-3x+5/(x-4), the polynomial starts out with degree 4 then the next highest degree is 1. It is missing degrees 3 and 2. So if we were to put it inside a division box we would write it like this:

x-4)x4+0x3+0x2-3x+5

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

4|1 0 0 -3 5

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

4|1 0 0 -3 5

………………………………………………

|1

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

4|1 0 0 -3 5

……………………………………………….

|1 4

Step 4: Add the column created in step 3

4|1 0 0 -3 5

4

………………………………………………..

1 4

Step 5: Repeat until done

4|1 0 0 -3 5

|1 4 16 64 244

………………………………………………….

1 4 16 61 249

**Step 6:** **Write out the answer**

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

Quotient = x3+4x2+16x+61

Remainder = 249

**Algorithm**

1. Input coefficients of dividend polynomial say, an, an-1,….a0
2. Input constant term of divisor polynomial of the form x-a
3. Set bn = 0
4. Repeat till n>0
5. Remainder R = a0+b0\*c

bn-1 = an+bn\*c

1. Print quotient polynomial and Remainder
2. Stop

**C program for Synthetic Division**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int a[30],b[30];

int i, m, n, c;

printf ("Enter degree of polynomial\n");

scanf ("%d",&n);

printf ("Enter coefficients of dividend polynomial\n");

for(i=n; i>=0;i--)

{

scanf("%d", &a[i]);

}

printf ("Enter the constant term of divisor polynomial\n");

scanf ("%d", &c);

b[n]=0;

m=n;

while(m>0)

{

b[m-1]=a[m]+b[m]\*c;

m = m-1;

}

printf ("Quotient:");

m = n-1;

while(m>=0)

{

if(b[m]!=0)

printf("%dx^%d+",b[m],m);

m = m-1;

}

getch();

return 0;

}

**Output:**

Enter degree of polynomial

4

Enter coefficients of dividend polynomial

2 1 0 -6 4

Enter the constant term of divisor polynomial

2

**Quotient:** 2x^3+5x^2+10x^1+14x^0+

**Remainder Theorem**

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

**Example:** Given the function f(x) = 3x3+x2+x-5, use remainder theorem to find f(-2)

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

-2|3 1 1 -5

-6 10 -22

…………………………………………

3 -5 11 -27

Remainder = -27

Thus f(-2) = -27

**Finding Multiple Roots by using Newton Raphson Method**

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

xn+1 = xn-

Suppose xr is one of the roots of polynomial of degree n. Now, deflate the polynomial by dividing it by x-xr to get another polynomial of degree n-1 and use xr as initial guess for finding next root. Continue this process until polynomial of degree one is achieved as quotient. This polynomial will be of the form a1x+a0 = 0. Now final root can be calculated as follows:

xr = -a0/a1

**Algorithm**

1. Start
2. Input degree and coefficients of polynomial
3. Input initial guess x0 and error criterion E
4. While n>1

Find root using Newton Raphson algorithm say nth root is xr

Divide the polynomial by x-xr to get the polynomial of degree n-1

Set x0 = xr

n = n-1;

1. End while
2. Root1 = -
3. Stop

**C Program for Calculating Multiple Roots using Newton Raphson Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define F(x) (a[4]\*x\*x\*x\*x+a[3]\*x\*x\*x+a[2]\*x\*x+a[1]\*x+a[0])

#define FD(x) (4\*a[4]\*x\*x\*x+3\*a[3]\*x\*x+2\*a[2]\*x+a[1])

#define E 0.0001

float a[20],q[20];

int main()

{

float x0,xr,fx0,fdx0,Er,c;

int i,n,m;

printf("Enter degree of polynomial\n");

scanf("%d",&n);

printf("Enter coefficients of dividend polynomial\n");

for(i=n;i>=0;i--)

scanf("%f",&a[i]);

printf("Enter initial guess\n");

scanf("%f",&x0);

while(n>1)

{

while(1)

{

fx0 = F(x0);

fdx0 =FD(x0);

xr = x0-fx0/fdx0;

Er = (xr-x0)/xr;

if(fabs(Er)<E)

{

printf("root%d=%f\n",n,xr);

break;

}

x0 = xr;

}

c = xr;

q[n]=0;

m = n-1;

while(m>=0)

{

q[m]=a[m+1]+q[m+1]\*c;

m--;

}

for(i=n;i>=0;i--)

{

a[i]=q[i];

}

n = n-1;

x0 = xr;

}

xr = a[0]/a[1];

printf("root%d=%f\n",n,xr);

getch();

return 0;

}

**Output:**

Enter degree of polynomial

4

Enter coefficients of dividend polynomial

1 -2 -13 38 -24

Enter initial guess

0.5

root4=1.000000

root3=2.000000

root2=3.000000

root1=4.000000

**Example:** Find the roots of the equation (x4-2x3-13x2+38x-24) = 0 using Newton Raphson Method

**Solution:**

Assume initial guess x0 = 0.5.

Error E = 0.01

**Step 1:**

Use Newton Raphson method to get root.

Thus, fourth root of the equation r4 = 1.00

**Step 2:**

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

x3-x2-14x+24=0

Again use Newton Raphson method with initial guess x0 = 1

Third root of the equation r3 = 2.00

**Step 3:**

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

x2+x-12=0

From Newton Raphson method with initial guess x0 = 2 and with the same error limit

Second root of equation r2 = 3.00

Again use synthetic division to deflate the polynomial by (x-3) which gives the polynomial

x+4 = 0

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

Root = -

Thus the first root r1 = -4

**Horner’s Method for Polynomial Evaluation**

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

x4+8x3+6x2+4x+3 = K\*x+3 where K = (8x3+8x2+6x+4)

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

x4+8x3+6x2+4x+3 = ((((1)\*x+8)\*x+6)\*x+4)\*x+3)

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

Given the polynomial

p(x) = = a0+a1x+a2x2+a3x3+……+anxn

where a0, a1…an are real numbers, we wish to evaluate the polynomial at a specific value of, say at x0. To accomplish this, we need to define given polynomial in terms of nested multiplication as below

p(x) = (a0+x(a1+a2x+a3x2+……+anxn-1))

p(x) = (a0+x(a1+x(a2+a3x+….+anxn-2)))

………………………………………………………….

…………………………………………………………

p(x) = (a0+x(a1+x(a2+x(a3+x(a4+…..+(an-1+x(an)……)

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

bn = an

bn-1 = an-1+bnx0

Then b0 is the value of p(x0)

**Algorithm**

1. Start
2. Enter the degree of polynomial say n
3. Enter the value at which polynomial is to be evaluated say x
4. Initialize bn = an
5. While n>0

bn-1 = an-1+bn\*x

1. Display the value of b0 which is the value of polynomial at x
2. Stop

**Example: Evaluate the polynomial p(x) = 3x3-4x2+5x-6 at x=2**

**Solution:**

We know that

a3 = 3, a2 = -1 a1 = 5 a0 = 6

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

b3 = a3 = 3

b2 = a2+b3\*x = -4+3\*2 = 2

b1 = a1+b2\*x = 5+2\*2 = 9

b0 = a0+b1\*x = -6+9\*2 = 12

Thus (2) = 12

**C program for evaluating the polynomial using Horner’s Method**

#include<stdio.h>

#include<conio.h>

#define P(x) (a[4]\*x\*x\*x\*x+a[3]\*x\*x\*x+a[2]\*x\*x+a[1]\*x+a[0])

float a[30],b[30];

int main()

{

float x;

int i,n;

printf("Enter degree of polynomial\n");

scanf("%d",&n);

printf("Enter %d coefficients of dividend polynomial\n",n+1);

for(i=n;i>=0;i--)

{

scanf("%f",&a[i]);

}

printf("Enter the polynomial at which polynomial to be evaluated\n");

scanf("%f",&x);

b[n]=a[n];

while(n>0)

{

b[n-1] = a[n-1]+b[n]\*x;

n = n-1;

}

printf("Value of polynomial p(%f)= %f",x,b[0]);

getch();

return 0;

}

Output:

Enter degree of polynomial

3

Enter 4 coefficients of dividend polynomial

3 -4 5 -6

Enter the polynomial at which polynomial to be evaluated

2

Value of polynomial p(2.000000)= 12.000000