

---

# SECTION 1: COMPUTER ARITHMETIC AND SOLVING LINEAR AND NON LINEAR EQUATIONS

---

Structure	Page Nos.
1.0 Introduction	
1.1 Objectives	
1.2 Round-off and Truncation Errors	
1.3 Solving Linear Algebraic Equations	
1.4 Solving Non-Linear Equations	
1.5 Summary	
1.6 Further Readings	

---

## 1.0 INTRODUCTION

---

Numerical Methods are used for solving mathematical problems using computations. One of the key concepts used in developing computational solutions is the Taylor Series. You must go through the details of BCS054: Numerical Methods course prior to attempting lab sessions on numerical methods.

This section includes three practical sessions. This section is based on Block 1 of BCS054. Therefore, you must go through the Block 1 before performing activities suggested in this section. You need to use a programming language to implement a numerical method. You may use C for the purpose of this course. In this section, we will be using Dev C++ compiler for demonstration of some of the programs relating to Numerical methods. This section demonstrates three main concepts viz. Use of Floating point numbers and errors, solving of linear and non-linear equations. We will demonstrate one program as example for these topics.

---

## 1.1 OBJECTIVES

---

After performing the activities of this section, you should be able to:

- Write and run C programs for solving non-linear equations
- Write and run C programs for solving linear equations
- Appreciate the round off and truncation error and their removal in a C program.

---

## 1.2 ROUND OFF AND TRUNCATION ERRORS

---

The round off and truncation errors sometimes can play major problems while solving a problem. We demonstrate this concept with the help of following problem.

**Problem:** Write a program to find if a given number is an Armstrong number. Test your program using three digits number 153 and 135.

But first let us define what an Armstrong Number is? Let us consider the number 153. You separate the three digits of this number; they are 1, 5 and 3. Now, you take a cube

## Computer Oriented Numerical Techniques Lab

Lines 1-3: Macro statements to include header files. Math.h is included as Power function (pow) is used in the program.

Line 4: Function prototype for the function created for finding cubes of digits of a number

Line 12-15: Code to compares the calculated digit cube to the number and prints if the number is Armstrong Number or not.

Line 19-30: Code to extract digits one by one and calculate cube of the extracted digits. It returns the sum of cubes of digits.

of each of these three digits, viz.  $1^3+5^3+3^3=1+125+27=153$  which is same as the number itself.

When the sum of cubes of the digits of a number is same as the number itself then the number is called an Armstrong Number. Two Armstrong numbers of three digits are 153 and 371.

Now, you can write a program for finding if the given number is Armstrong Number or not. The basic logic is simple – extract the digits of a number, perform the cube of each of these digits and sum them up. Following program may perform this task:

```
1. #include<stdio.h>
2. #include<conio.h>
3. #include<math.h>

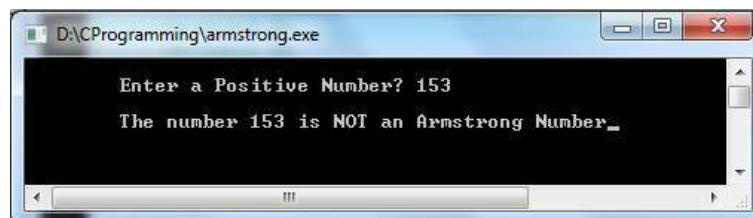
4. int digitcube(int); // Function to separate digits, Cube them and add the cubes.

5. int main()           // main function
6. {
7.     int number;
8.     printf("\nEnter a Positive Number? ");
9.     scanf("%d",&number);
10.    if (number <= 1)
11.        printf("The Number is One or Less than 1. Try again");
12.    else if(number==digitcube(number))
13.        printf("\nThe number %d is an Armstrong Number",number);
14.    else
15.        printf("\nThe number %d is NOT an Armstrong Number",number);
16.    getch();
17.    return 0;
18. }                    //main ends

19. int digitcube(int num)
20. {
21.     int sum=0, digit;
22.     while(num!=0)
23.     {
24.         digit=num%10;           //separate a digit
25.         sum=sum+(int)pow(digit,3); //use power function to perform cube of digit
26.                                     //Note pow function uses float parameters and returns float
27.         num=num/10;             //reduce the number
28.     }
29.     return sum;
30. }
```

**Figure 1: A Sample Program for finding if a number is Armstrong Number**

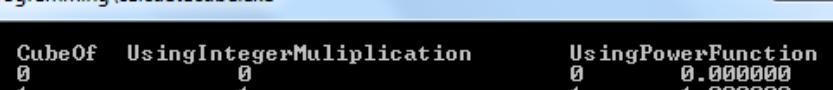
Let us show a simple example of how this program would work. As you compile and execute the program, first it will display a message to input a number, enter a number, 153. The following screen shows the output of the program.



(a) Output for 153

**Figure 2: Input and Output of the Program of Figure 2**

```
#include<stdio.h>
#include<math.h>
int main()
{
    int digit=0;
    printf("\n\tCubeOf\tUsingIntegerMuliplication\tUsingPowerFunction");
    while(digit < 10)
    {
        printf("\n\t%d\t\t%d\t\t%d\t\t\t\t\t",digit,digit*digit*digit, (int)pow(digit,3), pow(digit,3));
        digit++;
    }
    getch();
    return 0;
}
```



```
D:\CProgramming\calcuatcube.exe
```

CubeOf	UsingIntegerMultiplication	UsingPowerFunction
0	0	0.000000
1	1	1.000000
2	8	8.000000
3	27	27.000000
4	64	64.000000
5	125	125.000000
6	216	216.000000
7	343	343.000000
8	512	512.000000
9	729	729.000000_

Thus, you must be very careful about the round off and truncation errors while dealing with conversions of higher data type to lower data type such as float to integer. When you use approximations for solving problems, round off and truncation errors get propagated from iteration to iteration and sometimes may lead to incorrect results.

## ☞ Lab Session 1

- 1) Write a program to obtain the value of  $e$  correct up to second decimal place using the series

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

- 2) Write program to calculate the approximate value of Sine of a value given in radians. Compare the value obtained by you with actual values. Explain, if you find the difference between the two. The Sine can be defined using the formula:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

- 3) Write a program that accepts a decimal number and displays its floating point equivalent number. You may make assumptions to simplify the program, however, your representation of floating point number should be closer to IEEE 754 standard 32 bit representation.

## 1.3 SOLVING LINEAR ALGEBRAIC EQUATIONS

Linear algebraic equations can be solved using several methods. Some of these methods include Gauss elimination, Gauss Elimination with pivot control, Gauss Seidel Methods etc. In this section, we demonstrate the program of simple Gauss Elimination method. Please note that you may enhance this program to include input-output from a file, error checks etc.

*Problem:* Write a Program to find the solution of linear equation using Gauss Elimination for solving the equations:

$$\begin{aligned} x+y+z &= 2 \\ x-2y+3z &= 14 \\ x+3y-6z &= -23 \end{aligned}$$

The first step here is to identify the data structure. Since, you need to represent the coefficient matrix as well right side vector, you need to create a  $n*(n+1)$  array, where  $n$  is the number of equations which we are assuming are equal to the number of unknown values. We are also assuming that all the equations are independent, thus, a unique solution exist. We are also assuming that no pivot element is zero or will become zero while calculation (A very strong assumption, you may have to change program if you want to test this.) The following program performs the Gauss Elimination.

Lines 1,2: Macro statements to include header files.

Line 3: States the number of independent equations = number of variables = 3 in this case.

Line 6: Creates the Coefficient matrix appended with right side vector

```

1. #include<stdio.h>
2. #include<conio.h>

3. #define neq 3 //defines the number of equations

4. int main()

5. {
6.     float copvec[neq][neq+1]={
           {1,1,1,2},
           {1,-2,3,14},
           {1,3,-6,-23}
       };

7.     float x[neq],sumofmultcoeffandx =0;
8.     int i,j,k;
```

```

9. // The process of forward elimination
10. for (k=0;k<neq-1;k++)
11.     for (i=k+1;i<neq;i++)
12.         { m=copvec[i][k]/copvec[k][k];
13.           for (j=k;j<neq+1;j++)
14.               copvec[i][j]=copvec[i][j]-m*copvec[k][j];
15.         }

16. // Printing of reduced matrix
17. printf("\nThe Upper Triangular matrix including coefficient matrix and right side
vector\n");
18. for (i=0;i<neq;i++)
19. {
20.     for (j=0;j<neq+1;j++)
21.         printf("\t%f",copvec[i][j]);
22.     printf("\n");
23. }

24. // Back Substitution
25. for (i=neq-1; i>=0; i--)
26. {
27.     for (j=i+1;j<neq;j++)
28.         sumofmultcoeffandx = sumofmultcoeffandx + copvec[i][j] * x[j];
29.     x[i]= (copvec[i][neq] - sumofmultcoeffandx)/copvec[i][i];
30.     sumofmultcoeffandx = 0;
31. }
32. printf("\nThe evaluated values of variables");
33. for (i=0;i<neq;i++)
34.     printf("\n\tx[%d] = %.2f", i, x[i]);
35. getch();
36. return 0;
37. }

```

Line 9-15: The process of forward elimination is explained after the Figure 5.

Line 24-31: The process of back substitution is explained after the Figure 5.

**Figure 5: A Simple implementation of Gauss Elimination**

You may recollect from BCS052, Block 1, that for solving linear equations, Gauss elimination method follows two steps viz. Forward Elimination and Backward substitution. The purpose of forward elimination is to reduce the matrix to upper triangular matrix. Consider the following equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = d_1 \quad (1)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = d_2 \quad (2)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = d_3 \quad (3)$$

Then, the first step of forward elimination requires you to multiply equation 1 by  $m = a_{21}/a_{11}$  (Please note the calculation of m at line number 12 of the program) and

perform the operation (equation2 – m \* equation1) similarly, you may multiply equation 3 with  $m = a_{31}/a_{11}$  and perform (equation3 – m \* equation1). This will

result in elimination of variable  $x_1$  from equations 2 and 3. The reduced equations can be represented as:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = d_1 \quad (1) \text{ (Remains unchanged)}$$

$$b_{22}x_2 + b_{23}x_3 = e_2 \quad (4)$$

$$b_{32}x_2 + b_{33}x_3 = e_3 \quad (5)$$

In the second step, you need to eliminate  $x_2$  in equation 5. Once again, you follow the similar procedure that is multiply equation 4 by  $m = b_{32}/b_{22}$  and perform the operation (equation5 – m \* equation4). This will eliminate  $x_2$  from equation 5. Thus, the result of forward elimination process may be represented as:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = d_1 \quad (1) \text{ (Remains unchanged)}$$

$$b_{22}x_2 + b_{23}x_3 = e_2 \quad (4) \text{ (Remains unchanged)}$$

$$b_{33}x_3 = f_3 \quad (6)$$

Please note the following point about the forward elimination:

- Since, you had three independent equations with the independent variables, therefore you required 2 steps. Thus, for neq equations, you need neq-1 iterations.
    - In every iteration, you perform operations for equations. In the given example in the first iteration you perform processing of equations 2, 3. For second iteration you perform operation on 3<sup>rd</sup> equation. Thus, for n equations, iteration 1 will process equations 2 to neq; Iteration 2 will process equations 3 to neq.
      - In every processing of equation, you process all the elements in an equation starting from the  $a_{k+1,k}$  element,
- where k is the iteration.

Thus, you require three loops to perform the operation:

- The outermost loop for iterations, k values from 1 to neq-1. Since, in Programming Language C arrays the first index is 0, therefore, k will vary from 0 to neq-2 (Line number 10).
  - The middle loop will be for every row of equation, i values will vary from k+1 to neq-1(line number 11).
    - The inner loop, j varies from k to neq-1, but since, we are using coefficient matrix along with right side vector, the value j values will vary from k to neq (line number 13).

*Backward Substitution:* Once you have obtained equations like (1),(4) and (6) using forward elimination, you can perform the task of backward substitution. Equation 6 can directly be used to find the values of variables as:

$$x_3 = f_3/b_{33} \quad (\text{from equation 6})$$

$$x_2 = (e_2 - b_{23}x_3)/b_{22} \quad (\text{from equation 4})$$

$$x_1 = (d_1 - (a_{12}x_2 + a_{13}x_3))/a_{11} \quad (\text{from equation 1})$$

Please note first  $x_3$  is evaluated followed by  $x_2$  and  $x_1$ . Lines 24-31 of the program perform this task.

You can compile and run the program given in Figure 5 to get the following output:

```

The Upper Triangular matrix including coefficient matrix and right side vector
1.000000  1.000000  1.000000  2.000000
0.000000  -3.000000  2.000000  12.000000
0.000000  -0.000000  -5.666667  -17.000000

The evaluated values of variables
x1 = 1.00
x2 = -2.00
x3 = 7.00
  
```

**Figure 6: Output of the Program of Figure 5**

You can verify the matrix and results. Please note that one of the values in the matrix is -0.000000. Why? Please discuss with your counselor.

Now you should do the following lab session.

### Lab Session 2

- 1) Write a program to implement Gauss Elimination method for solving linear equations. Your method should check if a given pivot is zero or not. It should also minimise the floating point errors.
- 2) Write a program to implement Gauss Seidel method for finding the roots of linear equations.
- 3) Distance covered by an object can be represented by the equation:

$$s(t) = x_1 + x_2 t + x_3 t^2$$

The object distance was observed in the time interval. The following table shows these observations:

Time (t) in seconds	Distance (s) in meters
0	0
2	50
5	150
10	500

Create the system of linear equations using the data given in the table. Solve it using any program that you have created (in problem 1 or 2) for this purpose.

## 1.4 SOLVING NON-LINEAR EQUATIONS

Non-linear equations may include 2<sup>nd</sup> degree or higher degree polynomials. There are several numerical methods such as Bisection method, Newton-Raphson method, Regula-Falsi method, Secant method etc. that can be used to solve non-linear equations. MCS052 Block 1 Unit 3 explains these methods in details. In this section, we demonstrate a program on Bisection Method for finding a root. You can enhance this program and perform the lab session activities.

**Problem:** Find one of the equation  $x^2 - 4x = 0$  using Bisection Method which is

non-zero and positive. Make suitable assumptions of the lower and upper bound of the roots.

Figure 7 shows the program that implements the Bisection method for finding the roots of the equations.

```

1. //Program to find the real root of a non-linear equation using Bisection Method
2. //You must change f(x) as per the need
3. #include<stdio.h>
4. #include<conio.h>
5. #include<math.h>

6. #define epsilon 0.001 //Margin of error
7. #define maxiteration 20 //defines the maximum number of iterations

8. double f(double);

9. int main()
10. {
11.     double l, u, m;    //lowerbound - l, upperbound - u and mid point - m
12.     int iteration=0;
13.     printf("\n\tEnter the lower and upper bound values? ");
14.     scanf("%lf %lf",&l, &u);
15.     // check if the bounds specified by you, contains the root. If not then exit.
16.     if(fabs(f(l)) <= epsilon){
17.         printf("\n\tThe Root is at the Lower Limit specified by You, that is %.2f", l);
18.         getch();
19.         exit(0);
20.     }
21.     if(fabs(f(u)) <= epsilon) {
22.         printf("\n\tThe Root is at the Upper Limit specified by You, that is %.2f", u);
23.         getch();
24.         exit(0);
25.     }
26.     // check if the bounds specified by you, contains the root, if not exit.
27.     if(f(l)*f(u) > 0){
28.         printf("\n\tThe Root MAY NOT be in the lower bound %.2f (function value = %.2f)\n\tand upper bound %.2f (function value = %.2f)",l,f(l),u,f(u));
29.         getch();
30.         exit(0);
31.     }
32.     // bisect the interval and check if this is the root, if yes then output the value, repeat it for only a number of fixed iterations
33.     m=(l+u)/2.0;
34.     while ((fabs(f(m))>epsilon) && (iteration < maxiteration))
35.     {
36.         if(f(l)*f(m)<0)
37.             u=m;
38.         else
39.             l=m;
40.         printf("\n\tValues at iteration %d are: l= %.2f(%.2f), and u = %.2f(%.2f)", iteration+1, l,f(l), u, f(u));
41.         iteration++;
42.         m=(l+u)/2.0;
43.     }
44.     if(fabs(f(m))<epsilon)
45.         printf("\n\tThe Root of the Equation is %.2f",m);
46.     else
47.         printf("\n\tThe Root could not be reached after %d iterations",maxiteration);
48.     getch();
49.     return 0;
50. }

51. double f(double x)
52. {
53.     return x*x-4*x;
54. }

```

Lines 3  
header

Line 6: You  
floating poi  
instead of z  
close to zero  
defined as E

Lines 15-24  
of the lower  
is the root. I  
display the  
program.

Lines 26-31: Checks if the lower and upper bounds are NOT on either side of the root. How by using formula:  $f(a) \times f(b) > 0$ , where a and b are the two bounds.

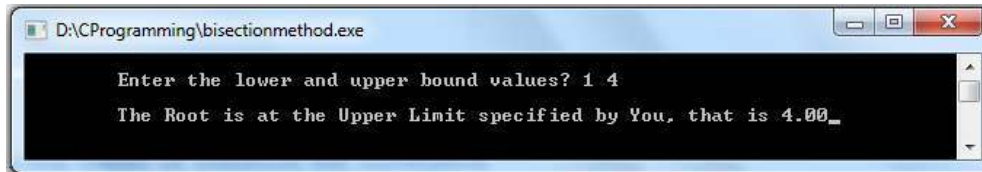
Lines 34-40: Performs iterations of Bisection method. Please note line 36 which performs the check  $f(a) \times f(b) < 0$ .

Lines 48-51: Equation for which root is to be found. Here, it is polynomial -  $x^2 - 4x$



Figure 7: Program to find root of a non-linear equation using Bisection method.

The program given in Figure 7, typically implements the Bisection method. The comments given at various parts of the program explain it. The logic of the program is simple and if you have read Bisection method, you will be able to write the code. You can test this program for various inputs. If you want to change the equation, then you need to change line 50 of the code suitably. Please also note the use of fabs() function in the program that finds out the absolute value of a floating point number. Figure 8 shows some of the output of this program.



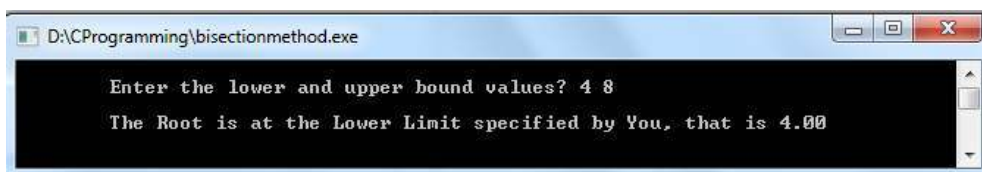
```

D:\CProgramming\bisectionmethod.exe

Enter the lower and upper bound values? 1 4
The Root is at the Upper Limit specified by You, that is 4.00_

```

(a) Output when root is at the upper bound



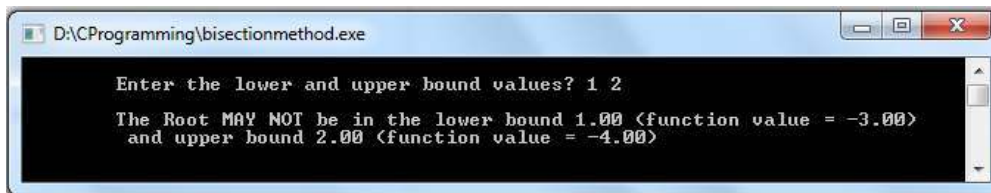
```

D:\CProgramming\bisectionmethod.exe

Enter the lower and upper bound values? 4 8
The Root is at the Lower Limit specified by You, that is 4.00

```

(b) Output when root is at the lower bound



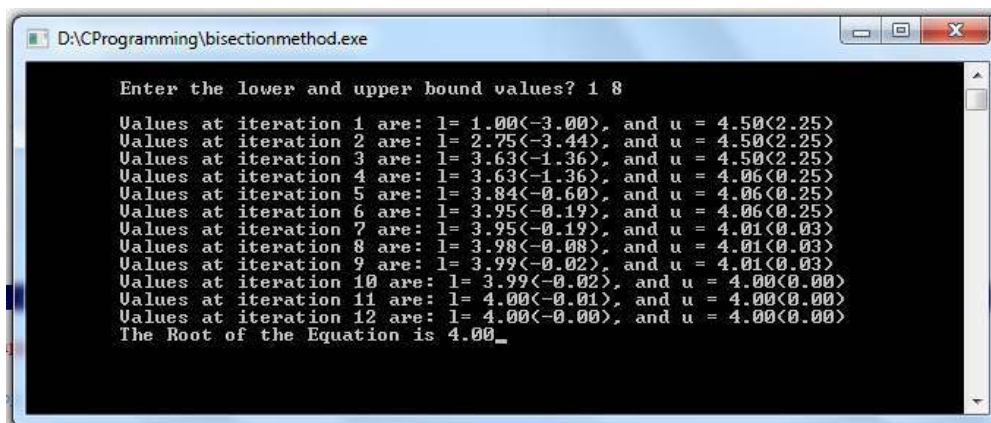
```

D:\CProgramming\bisectionmethod.exe

Enter the lower and upper bound values? 1 2
The Root MAY NOT be in the lower bound 1.00 <function value = -3.00>
and upper bound 2.00 <function value = -4.00>

```

(c) Output when bounds may not contain the root



```

D:\CProgramming\bisectionmethod.exe

Enter the lower and upper bound values? 1 8
Values at iteration 1 are: l= 1.00<-3.00>, and u = 4.50<2.25>
Values at iteration 2 are: l= 2.75<-3.44>, and u = 4.50<2.25>
Values at iteration 3 are: l= 3.63<-1.36>, and u = 4.50<2.25>
Values at iteration 4 are: l= 3.63<-1.36>, and u = 4.06<0.25>
Values at iteration 5 are: l= 3.84<-0.60>, and u = 4.06<0.25>
Values at iteration 6 are: l= 3.95<-0.19>, and u = 4.06<0.25>
Values at iteration 7 are: l= 3.95<-0.19>, and u = 4.01<0.03>
Values at iteration 8 are: l= 3.98<-0.08>, and u = 4.01<0.03>
Values at iteration 9 are: l= 3.99<-0.02>, and u = 4.01<0.03>
Values at iteration 10 are: l= 3.99<-0.02>, and u = 4.00<0.00>
Values at iteration 11 are: l= 4.00<-0.01>, and u = 4.00<0.00>
Values at iteration 12 are: l= 4.00<-0.00>, and u = 4.00<0.00>
The Root of the Equation is 4.00_

```

(d) Output showing iterations while finding the root

Figure 8: Output of Bisection Method Program of Figure 7

Please note that in Figure 8(c), for both the upper and lower bounds the value of function is negative, thus, you may state that no root may lie between the interval 1 and 2.

Now, you may perform the following lab session.

### ☞ Lab Sessions 3

1. Write a program to implement Bisection method for finding positive root of the equation  $x^2 - 9x + 21 = 0$ . You have to make suitable choice for the bounds.
2. Write a program to find one root of a non-linear equation using Newton-Raphson method. Compare the number of iterations taken by this method to that of Bisection Method.
3. Write a program to implement Regula-Falsi method.
4. Write a program to implement secant method.
5. Find the root of the following functions using any of the method given above.

a.  $f(x) = x^2$

b.  $f(x) = 1/x$

Explain your answer.

---

## 1.5 SUMMARY

---

This section provides you implementation specific details on floating point numbers and related errors, solving linear equation and non-linear equations. A program is presented to show problems relating to floating point numbers. The problems can be attributed to round off and truncation error. Since, you are dealing with a discrete digital binary system you will face these errors. Sometimes, such errors may result in incorrect computations as shown in the problem of calculation of Armstrong number. For solving linear equations, we have demonstrated basic Gauss Elimination method. This method is a two step process – forward elimination and backward substitution. The simplest way to implement this method will be to recognize the general formula and in terms of row and column number and then write the program. Finally, we have discussed about the Bisection method to find a root of a non-linear equation. Please note that both these methods if you change the function, you need to rewrite that portion of the code.

You must perform all the suggested lab sessions. They will help you in learning the numerical methods in a practical way. In addition, it will also enhance your programming skills.

---

## 1.6 FURTHER READINGS

---

- Numerical Methods With Application, Authors: Autar K Kaw, Co-Author: Egwu E Kalu, Duc Nguyen and Contributors: Glen Besterfield, Sudeep Sarkar, Henry Welch, Ali Yalcin, Venkat Bhethanabotla, Dedicated Website for book is: [http://mathforcollege.com/nm/topics/textbook\\_index.html](http://mathforcollege.com/nm/topics/textbook_index.html)
- [www.wikipedia.org](http://www.wikipedia.org)

---

## SECTION 1: INTEROPATION

---

### Structure

### Page Nos.

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Operators
- 1.3 Equidistant interpolation
- 1.4 Non-Equidistant interpolation
- 1.5 Summary
- 1.6 Further Readings

---

## 1.0 INTRODUCTION

---

“Knowing the unknown by using the known” is the art of interpolation, a numerical technique which are widely used in every domain of science, including computer science.

In this section you are going to learn how to implement various interpolation techniques, by using C or spreadsheets. This section includes three practical sessions, which are based on Block 2 of BCS054 i.e. *Interpolation*. Therefore, you must go through the Block 2 before performing activities suggested in this section.

To have the flavor of interpolation, refer to the following example.

Example:

Say you are having a thermometer, graduated at interval of 5 degrees i.e. least count of the instrument is 5 degrees. You want this device to be calibrated, for this reason you submitted to some calibration laboratory and the report they handed over to you make you to think about its outcome at other points, apart from the points considered in calibration report.

Say the report provided by calibration laboratory is

DEVICE READING	ACTUAL READING
0	1
10	12
20	19
30	31
40	43
50	49

Now, you are equipped with the calibration report of the thermometer and the thermometer itself. Someday, while using this thermometer you found that the reading shown by it is say 25 degree. Now, you referred your calibration report for the actual temperature, but it didn't exist there.

What should you do, should you get it re-calibrated? Or use some mechanism to find the result at 25 degrees. Getting it recalibrated for the result at unknown

point will be foolish, it is advised to use the numerical technique of interpolation to get the job done.

You might be surprised how to use this technique, studied by you in BCS054, to get the result at 25 degrees. It appears to be challenging but its quite simple, you will learn the solution to this problem in this unit itself. Just wait! For some time and you will be able to do it yourself.

---

## 1.1 OBJECTIVES

---

Execution of the activities given in this section will enable you to implement:

- the understanding of operators used in interpolation, through C or Excel
- the understanding of Equidistant interpolation, , through C or Excel
- the understanding of Non-Equidistant interpolation, , through C or Excel

---

## 1.2 OPERATORS

---

You learned from BCS054, that “if we are given  $(n + 1)$  pair of values  $(x_0, y_0)$ ,  $(x_1, y_1)$ , . . .  $(x_n, y_n)$  and the problem is to find the value of  $y$  at some intermediate point  $x$  in the interval  $[x_0, x_n]$ . Then the process of computing or determining this value is known as interpolation”.

There are several methods of interpolation when the abscissas  $x_i$ ,  $i = 0$  to  $n$ , known as tabular points, are equidistant i.e.  $x_i - x_{i-1} = h$ ,  $i = 1$  to  $n$  is same throughout. Before discussing these methods we need to introduce some operators.

You had already learned in BCS054 about following operators:

**a) Forward Difference Operator**

$$\Delta f(x) = f(x+h) - f(x)$$

**b) Backward Difference Operator**

$$\nabla f(x) = f(x) - f(x - h)$$

**c) Central Difference Operator**

$$\delta f(x_i) = f(x_i + \frac{h}{2}) - f(x_i - \frac{h}{2})$$

**d) Averaging Operator**

$$\mu f(x_i) = \frac{[f(x_i + h/2) + f(x_i - h/2)]}{2}$$

Now, you can apply your understanding of C, to write the program for demonstrating the operations performed by respective operators.

Just to start with, you are given below a C program, which demonstrates the execution of said operators.

**/\* PROGRAM - 1 : TO DEMONSTRATE THE OPERATION OF FORWARD/BACKWARD/CENTRAL/AVERAGING OPERATORS FOR A FUNCTION F(X)\*/**

**Computer  
Arithmetic and  
Solving Linear and  
Non-Linear  
Equations**

```
#include<conio.h>
#include<stdio.h>
#define F(x) (x)*(x) + (x) + 7 //FUNCTION F(X)
void main()
{
    int n;
    float a,b,h,x;
    double f1,f2,f0,FWDOP,BWDOP,CDOP,AVGOP;

/*FWDOP      -      FORWARD      DIFFERENCE      OPERATOR      ,
BWDOP      -      BACKWARD      DIFFERENCE      OPERATOR,
CDOP      -      CENTRAL      DIFFERENCE      OPERATOR,
AVGOP-AVERAGING OPERATOR */

    clrscr();
    printf("\n Enter the interval (a,b)");
    printf("\nEnter the value of a: ");
    scanf("%f",&a);
    printf("\nEnter the value of b: ");
    scanf("%f",&b);
    printf("\nEnter the number of nodes(n) in the interval (a,b): ");
    scanf("%d",&n);
    h=(b-a)/n; //STEP SIZE CALCULATION
    printf("\nSTEP SIZE (h) =: %f",h);
    printf("\n VALUE OF POINT OF EVALUATION (X) :");
    scanf("%f",&x);
    FWDOP = F(x+h) - F(x);
    BWDOP = F(x) - F(x+h);
    CDOP = F(x+(h/2)) - F(x-(h/2));
    AVGOP = 0.5*(F(x+(h/2)) + F(x-(h/2)));
    printf("\n RESULT OF FORWARD DIFFERENCE OPEATOR = %f",FWDOP);
    printf("\n RESULT OF BACKWARD DIFFERENCE OPEATOR = %f",BWDOP);
    printf("\n RESULT OF CENTRAL DIFFERENCE OPEATOR = %f",CDOP);
    printf("\n RESULT OF AVERAGING OPEATOR = %f",AVGOP);
    getch();
}
```

```
Enter the interval (a,b)
Enter the value of a: 2

Enter the value of b: 7

Enter the number of nodes(n) in the interval (a,b): 5

STEP SIZE (h) =: 1.000000
VALUE OF POINT OF EVALUATION (X) :3.4

RESULT OF FORWARD DIFFERENCE OPEATOR = 29.600000
RESULT OF BACKWARD DIFFERENCE OPEATOR = 14.000000
RESULT OF CENTRAL DIFFERENCE OPEATOR = 27.600000
RESULT OF AVERAGING OPEATOR = 22.210001_
```

### 👉 Lab Session 1

- 1) Modify the program given above to calculate the result for  $\Delta^2 f(x)$
- 2) Modify the program given above to calculate the result for  $\Delta^2 f(x)$

### HINT :

Second Order Forward difference operator

$$\Delta^2 f_k(x) = \Delta f_{k+1}(x) - \Delta f_k(x) = f_{k+2}(x) - 2f_{k+1}(x) + f_k(x);$$

Refer to BCS054 for the expression of backward difference operator.

### NOTE :

Developing the programs for Lab Session-1 will help you to develop the program for Forward Difference Table and Backward Difference table, which are explicitly required to perform interpolation with equal intervals i.e. our next section.

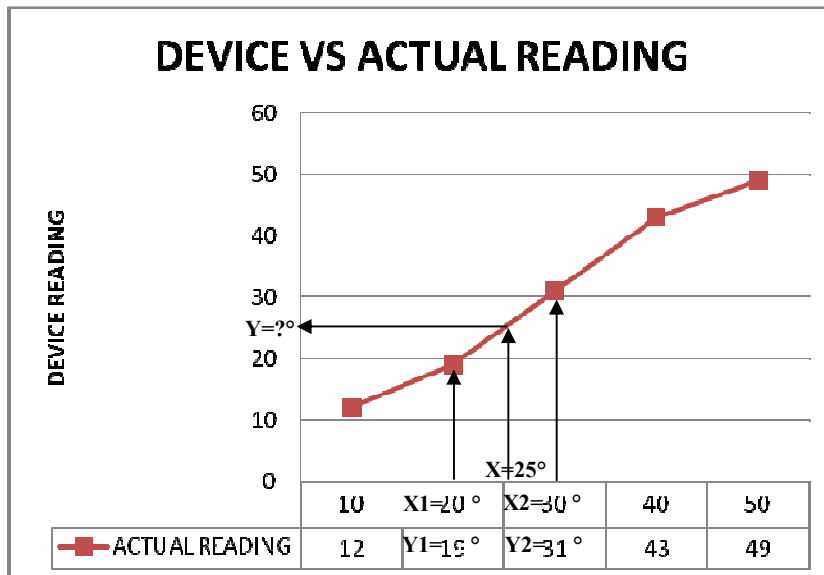
---

## 1.3 INTERPOLATION WITH EQUAL INTERVALS

---

If you are done with Lab session-1 then it's great! And it's time to answer your query for the example given in introduction for demonstrating the art of interpolation i.e. "Knowing the unknown through some known", we will start with the linear interpolation and then extend our discussion to other techniques of interpolation, which you had learned in BCS054.

To answer your query regarding the calculation of temperature, when device shows 25°C of temperature, let's plot the graph for the calibration report i.e.



Now you know that there is a straight line between two points, as in this case it is shown by a straight line between point(x1,y1) and (x2,y2). Further, it is to be noticed that device temperature(x) is known i.e. 25°C but the actual temperature Y°C is not known. The interval in which this point i.e.(x,y) lies is (x1,y1) and (x2,y2) i.e.(20,19) and (30,31). We can apply two point equation of line to find the value of Y. You learned in BCS054 that The expression of linear interpolation resolves to the formula of two point equation of line i.e.

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

Solving the equation you can determine Y which turns out to be

$$y = y_1 + \frac{(y_2 - y_1) * (x - x_1)}{(x_2 - x_1)}$$

Calculation:

$$y = 19 + \frac{(31 - 19) * (25 - 20)}{(30 - 20)} = 25^\circ$$

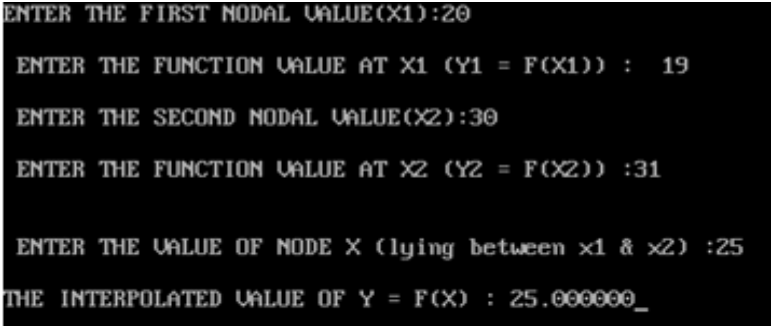
i.e. the device works perfect at 25°C , its device reading and actual reading matches well.

Lets demonstrate this concept of Linear Interpolation with program in C

**/\* PROGRAM-2 : LINEAR INTERPOLATION \*/**

```
#include<stdio.h>
main()
{
float x1,y1,x2,y2,x,y,k1,k;
clrscr();
printf("ENTER THE FIRST NODAL VALUE(X1):");
scanf("%f",&x1);
printf("\n ENTER THE FUNCTION VALUE AT X1(Y1=F(X1)) : ");
scanf("%f",&y1);
printf("\n ENTER THE SECOND NODAL VALUE(X2):");
scanf("%f",&x2);
printf("\n ENTER THE FUNCTION VALUE AT X2(Y2= F(X2)) :");
scanf("%f",&y2);
printf("\n");
printf("\n ENTER THE VALUE OF NODE X (BETWEEN X1 & X2) :");
scanf("%f",&x);
k=(x-x1)/(x2-x1);
k1=k*(y2-y1);
y=(y1+k1);
printf("\nTHE INTERPOLATED VALUE OF Y = F(X) : %f",y);
}
```

**OUTPUT : PROGRAM-2 LINEAR INTERPOLATION**



```
ENTER THE FIRST NODAL VALUE(X1):20
ENTER THE FUNCTION VALUE AT X1 (Y1 = F(X1)) : 19
ENTER THE SECOND NODAL VALUE(X2):30
ENTER THE FUNCTION VALUE AT X2 (Y2 = F(X2)) :31
ENTER THE VALUE OF NODE X (lying between x1 & x2) :25
THE INTERPOLATED VALUE OF Y = F(X) : 25.000000_
```

Now, you are required to use the understanding of the tasks assigned to you in lab session-1 for the development of the programs, related to Newton's Forward Difference Formula, Newton's Backward Difference Formula, and Central Difference Formula i.e. Difference tables for each and use it to find the unknown from the known data.



To begin with let me demonstrate you with the program for Newton's Forward Difference Formula, which you had gone through in your BCS054. Just read the text given in BCS054 for the Interpolation with Equal intervals and apply the same to the code given below, you will understand the logical execution.

**/\*PROGRAM 3 : NEWTONS FORWARD DIFFERENCE FORMULA \*/**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x[10],y[10][10],sum,p,u,temp;
    int i,n,j,k=0,f,m;
    float fact(int);
    clrscr();
    printf("\nhow many record you will be enter: ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("\n\nenter the value of x%d: ",i);
        scanf("%f",&x[i]);
        printf("\n\nenter the value of f(x%d): ",i);
        scanf("%f",&y[k][i]);
    }
    printf("\n\nEnter X for finding f(x): ");
    scanf("%f",&p);
    for(i=1;i<n;i++)
    {
        for(j=0;j<n-i;j++)
        {
            y[i][j]=y[i-1][j+1]-y[i-1][j];
        }
    }
    printf("\n_____ \n");
    printf("\n x(i)\t y(i)\t y1(i)  y2(i)  y3(i)  y4(i)");
    printf("\n_____ \n");
    for(i=0;i<n;i++)
    {
        printf("\n %.3f",x[i]);
        for(j=0;j<n-i;j++)
        {
            printf(" ");
            printf(" %.3f",y[j][i]);
        }
    }
    printf("\n");
}
```

```
}
i=0;
do
{
    if(x[i]<p && p<x[i+1])
        k=1;
    else
        i++;
} while(k != 1);
f=i;
u=(p-x[f])/(x[f+1]-x[f]);
printf("\n\n u = %.3f ",u);
n=n-i+1;
sum=0;
for(i=0;i<n-1;i++)
{
    temp=1;
    for(j=0;j<i;j++)
    {
        temp = temp * (u - j);
    }
    m=fact(i);
    sum = sum + temp*(y[i][f]/m);
}
printf("\n\n f(%.2f) = %f ",p,sum);
getch();
}

float fact(int a)
{
    float fac = 1;
    if (a == 0)
        return (1);
    else
        fac = a * fact(a-1);
    return(fac);
}
```

## OUTPUT: PROGRAM-3 NEWTON'S FORWARD DIFFERENCE FORMULA

how many record you will be enter: 4

enter the value of x0: 2

enter the value of f(x0): 9

enter the value of x1: 2.25

enter the value of f(x1): 10.06

enter the value of x2: 2.5

enter the value of f(x2): 11.25

enter the value of x3: 2.75

enter the value of f(x3): 12.56\_

Enter X for finding f(x): 2.35

x(i)	y(i)	y1(i)	y2(i)	y3(i)	y4(i)
2.000	9.000	1.060	0.130	-0.010	
2.250	10.060	1.190	0.120		
2.500	11.250	1.310			
2.750	12.560				
u = 0.400					
f(2.35) = 10.521600					

### 👉 Lab Session 2

- 1) Modify the Linear Interpolation program (i.e. program-2) for the calculation of the actual data, such that the function values are calculated at run time and not given explicitly as it is done in the above program.

Hint : See the use of #define in Program-1

- 2) Modify program-3 for the demonstration of Newton's Backward Interpolation Formula.
- 3) Write the program in C for Central Difference formula
- 4) Write program in C for Bessel's Formula
- 5) Write program in C for Stirling's Formula

Hint: Refer to BCS054

---

## 1.4 INTERPOLATION WITH UNEQUAL INTERVALS

---

You had already learned about the techniques lying under this category of interpolation in BCS054, its time to implement them in C. Further, after executing through the tasks given in Lab Session 1 & 2, you are equipped enough to complete the tasks to be assigned to you in subsequent Lab Session-3.

We will start this section with a sample program-4 for Lagrange method to find  $F(x)$  for a given value of  $X$

**/\*PROGRAM-4 :LAGRANGE'S INTERPOLATION METHOD FOR FINDING F(X) FOR A GIVEN X\*/**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
float x[10],y[10],temp=1,f[10],sum,p;
int i,n,j,k=0,c;
clrscr();
printf("\nhow many record you will be enter: ");
scanf("%d",&n);
for(i=0; i<n; i++)
{
printf("\n\nenter the value of x%d: ",i);
scanf("%f",&x[i]);
printf("\n\nenter the value of f(x%d): ",i);
scanf("%f",&y[i]);
}
printf("\n\nEnter X for finding f(x): ");
scanf("%f",&p);
for(i=0;i<n;i++)
{
temp = 1;
k = i;
for(j=0;j<n;j++)
{ if(k==j)
{ continue; }
else
{ temp = temp * ((p-x[j])/(x[k]-x[j])); }
}
```

```
} f[i]=y[i]*temp;
}
for(i=0;i<n;i++)
{ sum = sum + f[i]; }
printf("\n\n f(%.1f) = %f ",p,sum);
getch();
}
```

OUTPUT: PROGRAM-4 LAGRANGE'S INTERPOLATION

```
how many record you will be enter: 4
```

```
enter the value of x0: 2
```

```
enter the value of f(x0): 7
```

```
enter the value of x1: 2.25
```

```
enter the value of f(x1): 10.29
```

```
enter the value of x2: 2.70
```

```
enter the value of f(x2): 11.5
```

```
enter the value of x3: 2.95
```

```
enter the value of f(x3): 13.6_
```

```
enter the value of f(x0): 7
```

```
enter the value of x1: 2.25
```

```
enter the value of f(x1): 10.29
```

```
enter the value of x2: 2.7
```

```
enter the value of f(x2): 11.5
```

```
enter the value of x3: 2.95
```

```
enter the value of f(x3): 13.6
```

```
Enter X for finding f(x): 2.3
```

```
f(2.3) = 10.577614 _
```

Now, you may perform the following lab session.

### ☞ Lab Sessions 3

- 1) Modify the program-4 ( i.e. Lagrange's Interpolation Method for finding  $F(x)$  for a given  $X$ ), such that it can determine the value of  $X$  for a given  $F(X)$ .
- 2) Write a program in C to demonstrate the Newton's Divided Difference Method, which you learned in BCS054.

---

## 1.5 SUMMARY

---

The content and codes given in this section, equipped you with the double edge sword for your victory in both domains i.e. Computer Science and Mathematics. You learned here, how to apply the theoretical understanding to get the techniques implemented in programming language, which is C in our case. The techniques you learned in BCS054 and here i.e. BCSL058 has wide area of application; be it graphics, image processing, calibration, quality assurance etc.

Through the simple concept of linear interpolation, you learned the meaning of “Knowing the unknown from the known”, i.e. interpolation. This understanding was extended to different techniques related to the operators for interpolation, interpolation with equal intervals and interpolation with unequal intervals. To understand the concept of interpolation from the point of view of computer science, you must perform all the suggested lab sessions. They will help you in learning the concept of interpolation and its related methods in a practical way. Further, it will also enhance your programming skills.

---

## 1.6 FURTHER READINGS

---

- Numerical Methods With Application, Authors: Autar K Kaw, Co-Author: Egwu E Kalu, Duc Nguyen and Contributors: Glen Besterfield, Sudeep Sarkar, Henry Welch, Ali Yalcin, Venkat Bhethanabotla, Dedicated Website for book is: [http://mathforcollege.com/nm/topics/textbook\\_index.html](http://mathforcollege.com/nm/topics/textbook_index.html)
- [www.wikipedia.org](http://www.wikipedia.org)