GOVERNMENT POLYTECHNIC, AMRAVATI

(An Autonomous Institute of Government of Maharashtra)

NBA Accredited Institute

Certificate

Name of Department: Computer Science and Engineering.

This is to certify that Mr. Ayush Shashikant Bulbule Identity Code 19CM007 has completed the practical work of the course CM3406 NUMERICAL METHODS during the Academic year.

Signature of the Teacher

Date: *who taught the examinee* *Head of Department*

Index

Practical no 1.

Aim: Find accuracy and precision of given numbers.

Question:

1)Find Accuracy:

a) 45.869 b) 0.004762 3) 0.0856000 c) 42 d) 5700
e) 4200.00

2)Find Precision:

a) 8.4612 b) 6.84 e)5.341062

Theory:

Accuracy:

Accuracy refers to the no of significant digits in a value.

E.g. 57.396 Accuracy = 5;

Precision:

Precision refers to the number of decimal position in a value.

E.g. 57.396 Precision = 10^{-3};

Solution:

Find Accuracy:

1) 45.869 = 5 Significant Digits So, Accuracy = 5
2) 0.004762 = 4 Significant Digits So, Accuracy = 4
3) 0.0856000 =6 Significant Digits So, Accuracy = 6
4) 42 = 2 Significant Digits So, Accuracy = 2
5) 5700 = 2 Significant Digits So, Accuracy = 2
6) 4200.00 = 6 Significant Digits So, Accuracy = 6

Find Precision:

1) 8.4612 = 4 Decimal Positions = Precision = 10^{-4}
2) 6.84= 4 Decimal Positions = Precision = 10^{-2}
3) 5.341062= 4 Decimal Positions = Precision = 10^{-6}

Practical no 2.

Aim: Find truncation error from given series.

Question: Find the Truncation Error in the result of following function for x = 1/3 when we use 1st term 2nd term ..and 5th term-

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

Theory:

Find truncation error in the result of following function for x=⅓ when we use:

$e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + x^5/5!$

1) First three term -

Truncation Error = $x^3/3! + x^4/4! + x^5/5!$

$= (0.3)^3/3! + (0.3)^4/4! + (0.3)^5/5!$
$= (0.3)^3/6 + (0.3)^4/24 + (0.3)^5/120$
$= 0.485775 \times 10^{-4}$

2) First four term -

Truncation Error = $x^4/4! + x^5/5!$

$= (0.3)^4/4! + (0.3)^5/5!$
$= (0.3)^4/24 + (0.3)^5/120$
$= 0.00035775$

3) First five term –

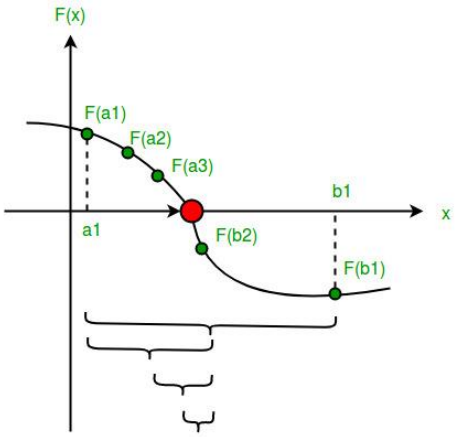
Truncation Error = $x^5/5!$

$= (0.3)^5/5!$
$= (0.3)^5/120$
$= 0.00002025$

Practical no 3.

Aim: Write C program to Determine roots of given algebraic equations using Bisection method.

Theory: Bisection method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which f(x) = 0.



The method is based on **The Intermediate Value Theorem** which states that if f(x) is a continuous function and there are two real numbers a and b such that f(a)*f(b) 0 and f(b) < 0), then it is guaranteed that it has at least one root between them.

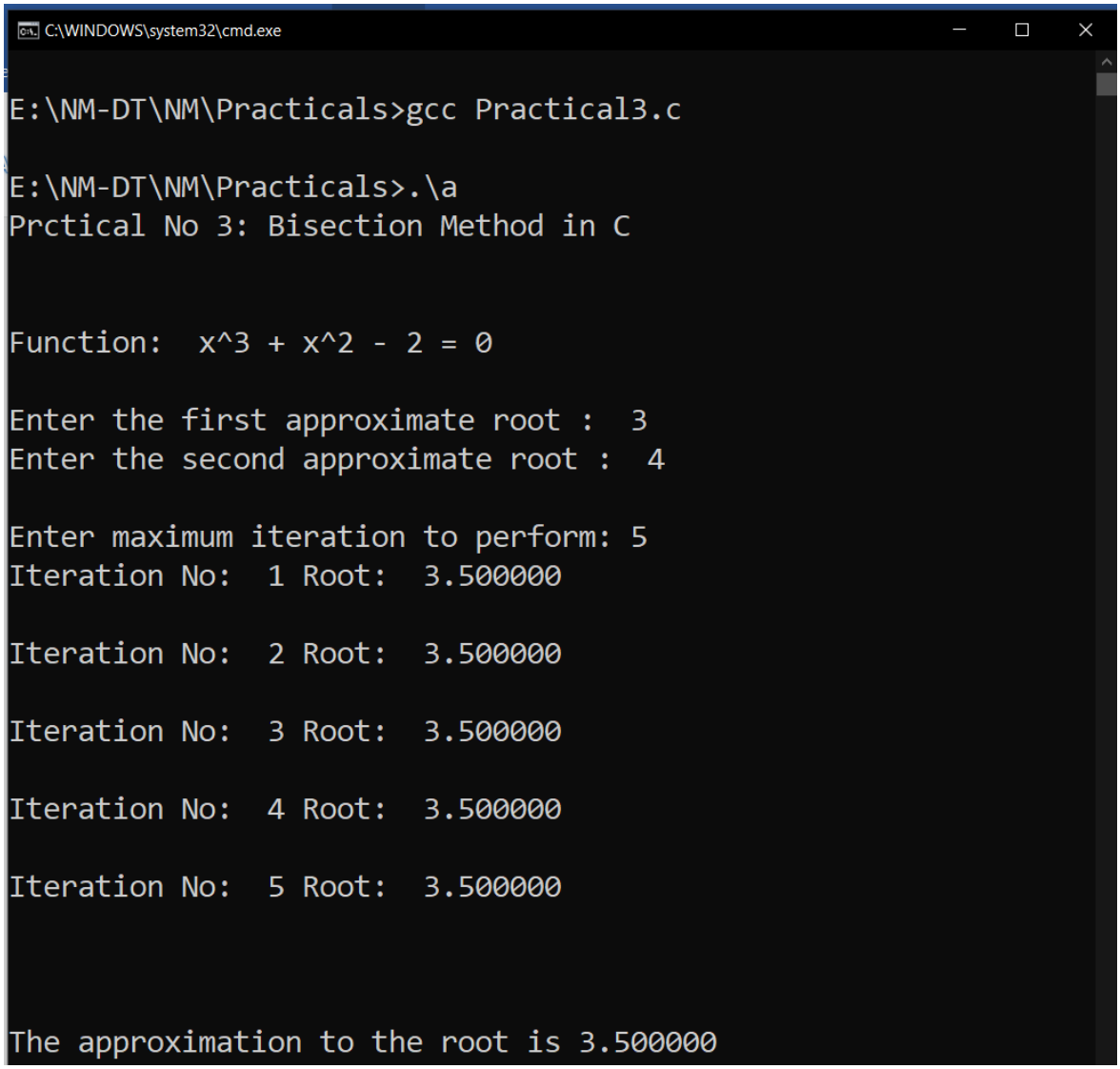
Program In C :

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

double F(double x)
{
    return (pow(x, 3) + pow(x, 2) - 2);
}
int main()
{
    double x0, x1;
    double r, f1, f2, f3;
    int i = 1;
    printf("Prctical No 3: Bisection Method in C\n\n\n");
    printf("Function:  x^3 + x^2 - 2 = 0\n\n");

    printf("Enter the first approximate root :  ");
```

```
    scanf("%lf", &x0);

    printf("Enter the second approximate root :  ");
    scanf("%lf", &x1);

    int iter;
    printf("\nEnter maximum iteration to perform: ");
    scanf("%d", &iter);

    double l1 = x0;
    double l2 = x1;

    if (F(l1) == 0)
        r = l1;
    else if (F(l2) == 0)
        r = l2;
    else
    {
        while (i <= iter)
        {
            f1 = F(l1);
            r = (l1 + l2) / 2.0;
            f2 = F(r);
            f3 = F(l2);

            if (f2 == 0)
            {
                r = f2;
                break;
            }
            printf("Iteration No:  %d Root:  %lf \n\n", i, r);
            if (f1 * f2 < 0)
                l2 = r;
            else if (f2 * f3 < 0)
                l1 = r;
            i++;
        }
    }
```

```
    printf("\n\nThe approximation to the root is %lf\n", r);
    return 0;
}
```

Output:

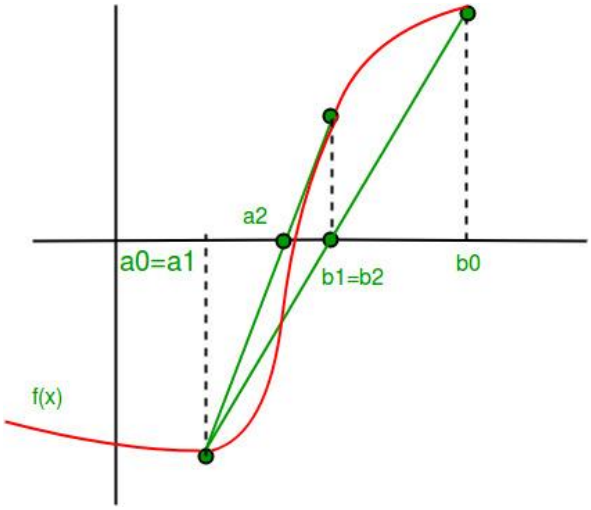


Practical no 4.

Aim: Write algorithm to obtain solution of given algebraic equation using Regula Falsi method.

Theory:

Regula Falsi method, also known as the false position method, is the oldest approach to find the real root of a function. It is a closed bracket method and closely resembles the **bisection method**.



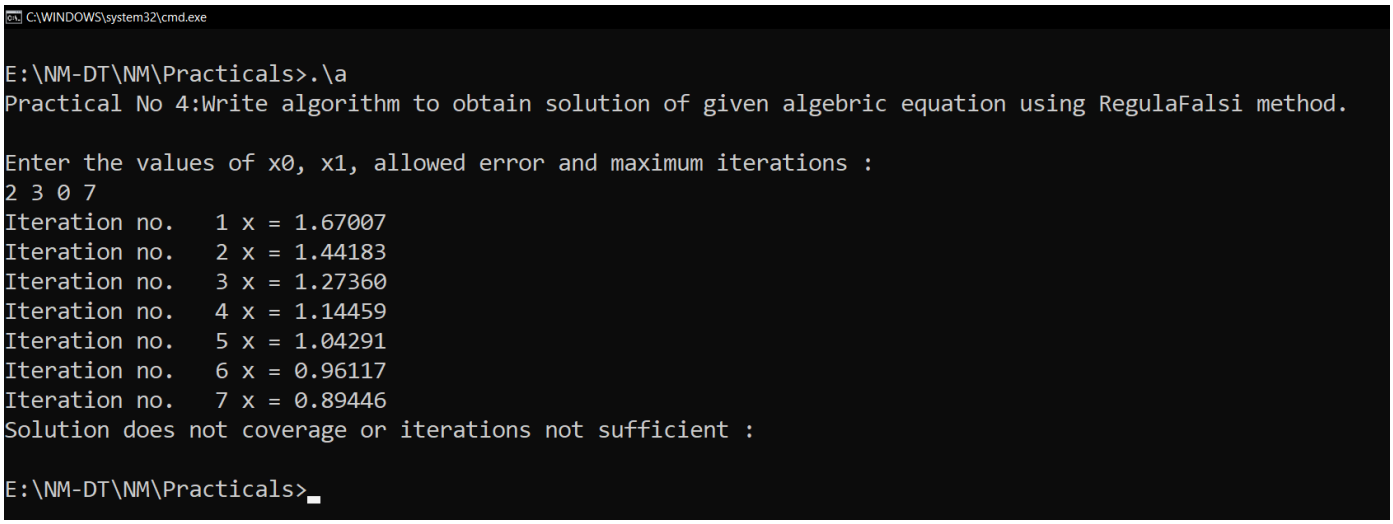
Program in C:

```
//Write algorithm to obtain solution of given algebric equation
 using RegulaFalsi method.
```

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

float f(float x)
{
    return cos(x) - x * exp(x);
}

void regula(float *x, float x0, float x1, float fx0, float fx1,
 int *itr)
{
    *x = x0 - ((x1 - x0) / (fx1 - fx0)) * fx0;
    ++(*itr);
```

```
    printf("Iteration no. %3d x = %7.5f\n", *itr, *x);
}

int main()
{
    int itr = 0, maxmitr;
    float x0, x1, x2, x3, allerr;

    printf("Practical No 4:Write algorithm to obtain solution o
f given algebric equation using RegulaFalsi method.");

    printf("\nEnter the values of x0, x1, allowed error and max
imum iterations :\n");
    scanf("%f %f %f %d", &x0, &x1, &allerr, &maxmitr);

     regula(&x2, x0, x1, f(x0), f(x1), &itr);

    do
    {
        if (f(x0) * f(x2) < 0)
            x1 = x2;
        else
            x0 = x2;
        regula(&x3, x0, x1, f(x0), f(x1), &itr);

        if (fabs(x3 - x2) < allerr)
        {
            printf("Iteration No: %d, Root = %6.4f\n", itr, x3)
;
            return 0;
        }
        x2 = x3;
    } while (itr < maxmitr);
    printf("Solution does not coverage or iterations not suffic
ient : \n");
    return 1;
}
```

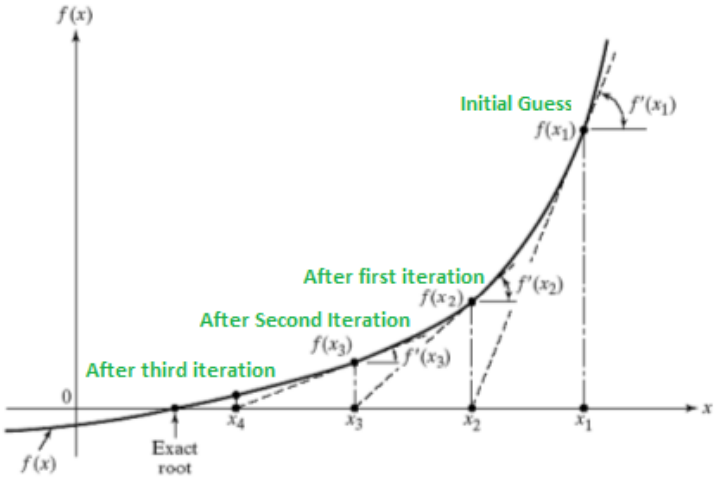
Output:


C:\WINDOWS\system32\cmd.exe
E:\NM-DT\NM\Practicals>.\a
Practical No 4:Write algorithm to obtain solution of given algebric equation using RegulaFalsi method.
Enter the values of x0, x1, allowed error and maximum iterations :
2 3 0 7
Iteration no. 1 x = 1.67007
Iteration no. 2 x = 1.44183
Iteration no. 3 x = 1.27360
Iteration no. 4 x = 1.14459
Iteration no. 5 x = 1.04291
Iteration no. 6 x = 0.96117
Iteration no. 7 x = 0.89446
Solution does not coverage or iterations not sufficient :
E:\NM-DT\NM\Practicals>

Practical no 5.

Aim: Write C program to solve the given system of linear equation using Newton – Raphson Method

Theory: Newton's method is often used to improve the result or value of the root obtained from other methods. This method is more useful when the first derivative of f(x) is a large value. The programming effort for Newton Raphson Method in C language is relatively simple and fast.



Program in C:

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

#define e 0.001
#define f(x) pow(x, 3) - 4 * x + 1
#define df(x) 3 * x *x - 4

int main()
{
    float x0, x1, f0, f1, df0;
    int i = 0;

    printf("Practical No.5:Write C program to solve the given s
ystem of linear equation using Newton – Raphson Method\n");
    printf("\nEnter the value of x0: ");
```

```
    scanf("%f", &x0);

    do
    {
        f0 = f(x0);
        df0 = df(x0);
        x1 = x0 - (f0 / df0);
        f1 = f(x1);
        x0 = x1;
        i++;

        printf("Iteration No: %d\t", i);
        printf("Root = %f\t", x1);
        printf("Value of Function = %f\n", f1);
    } while (fabs(f1) > e);

    return 0;
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe

E:\NM-DT\NM\Practicals>.\a

Practical No.5:Write C program to solve the given system of linear equation using Newton Raphson Method

Enter the value of x0: 3.2
Iteration No: 1 Root = 2.415270 Value of Function = 5.428463
Iteration No: 2 Root = 2.013179 Value of Function = 1.106473
Iteration No: 3 Root = 1.877559 Value of Function = 0.108588
Iteration No: 4 Root = 1.861046 Value of Function = 0.001532
Iteration No: 5 Root = 1.860806 Value of Function = 0.000000

E:\NM-DT\NM\Practicals>
```

Practical no 6.

Aim: Write algorithm to find solution of given Simultaneous Equations using Gauss Elimination method.

Theory:

Gauss Elimination method can be adopted to find the solution of linear simultaneous equations arising in engineering problems. In the method, equations are solved by elimination procedure of the unknowns successively.

The method overall reduces the system of linear simultaneous equations to an upper triangular matrix. Then backward substitution is used to derive the unknowns. This is the key concept in writing an algorithm or program, or drawing a flowchart for Gauss Elimination.

Program in C:

```
#include <stdio.h>
int main()
{
    int i, j, k, n;
    float A[20][20], c, x[10], sum = 0.0;

    printf("\nPractical 6:Write algorithm to find solution of given
Simultaneous Equations using GaussElimination method\n");
    printf("\nEnter the order of matrix: ");
    scanf("%d", &n);
    printf("\nEnter the elements of augmented matrix row-
wise:\n\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= (n + 1); j++)
        {
            printf("A[%d][%d] : ", i, j);
            scanf("%f", &A[i][j]);
        }
    }
    for (j = 1; j <= n; j++) //Generation of upper triangular Matrix
    {
```

```
        for (i = 1; i <= n; i++)
        {
            if (i > j)
            {
                c = A[i][j] / A[j][j];
                for (k = 1; k <= n + 1; k++)
                {
                    A[i][k] = A[i][k] - c * A[j][k];
                }
            }
        }
    }
    x[n] = A[n][n + 1] / A[n][n];

    for (i = n - 1; i >= 1; i--) //backward substitution
    {
        sum = 0;
        for (j = i + 1; j <= n; j++)
        {
            sum = sum + A[i][j] * x[j];
        }
        x[i] = (A[i][n + 1] - sum) / A[i][i];
    }
    printf("\nThe solution is: \n");
    for (i = 1; i <= n; i++)
    {
        printf("\nx%d=%f\t", i, x[i]);
    }
    return (0);
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe

E:\NM-DT\NM\Practicals>gcc Practical6.c

E:\NM-DT\NM\Practicals>.\a

Practical 6:Write algorithm to find solution of given Simultaneous Equations using GaussElimination method

Enter the order of matrix: 3

Enter the elements of augmented matrix row-wise:

A[1][1] : 1
A[1][2] : 4
A[1][3] : 9
A[1][4] : 16
A[2][1] : 2
A[2][2] : 1
A[2][3] : 1
A[2][4] : 10
A[3][1] : 3
A[3][2] : 2
A[3][3] : 3
A[3][4] : 18

The solution is:

x1=6.999992
x2=-8.999968
x3=4.999987
E:\NM-DT\NM\Practicals>
```

Practical no 7.

Aim: Write algorithm to find solution of given Simultaneous Equations using Gauss Elimination method.

Theory: In numerical linear algebra, the **Jacobi method** is an **algorithm** for determining the solutions of a diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges.

Program in C:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define ESP 0.001
#define X1(x2, x3) ((17 - 20 * (x2) + 2 * (x3)) / 20)
#define X2(x1, x3) ((-18 - 3 * (x1) + (x3)) / 20)
#define X3(x1, x2) ((25 - 2 * (x1) + 3 * (x2)) / 20)

void main()
{
    double x1 = 0, x2 = 0, x3 = 0, y1, y2, y3;
    int i = 0;
    printf("\nPractical 7:Write algorithm to find solution of given
Simultaneous Equations using Gauss Elimination method.");
    printf("\n_____\n");
    printf("\n      x1\t      x2\t      x3\n");
    printf("\n_____\n");
    printf("\n%f\t%f\t%f", x1, x2, x3);

    do
    {
        y1 = X1(x2, x3);
        y2 = X2(x1, x3);
        y3 = X3(x1, x2);

        if (fabs(y1 - x1) < ESP && fabs(y2 - x2) < ESP && fabs(y3 -
x3) < ESP)
        {
```

```
            printf("\n_____________________________________________\n"
);
            printf("\n\nx1 = %.3lf", y1);
            printf("\n\nx2 = %.3lf", y2);
            printf("\n\nx3 = %.3lf", y3);
            i = 1;
        }
        else
        {
            x1 = y1;
            x2 = y2;
            x3 = y3;
            printf("\n%f\t%f\t%f", x1, x2, x3);
        }
    } while (i != 1);
}
```

Output:

```
Select C:\WINDOWS\system32\cmd.exe
e:\NM-DT\NM\Practicals>.\a

Practical 7:Write algorithm to find solution of given Simultaneous Equations using Gauss Elimination method.
_____________________________________________

      x1              x2              x3

_____________________________________________

0.000000        0.000000        0.000000
0.850000        -0.900000       1.250000
1.875000        -0.965000       1.030000
1.918000        -1.129750       0.917750
2.071525        -1.141812       0.888738
2.080686        -1.166292       0.871576
2.103449        -1.168524       0.866988
2.105223        -1.172168       0.864376
2.108606        -1.172565       0.863653
_____________________________________________


x1 = 2.109

x2 = -1.173

x3 = 0.863
e:\NM-DT\NM\Practicals>
```

Practical no 8.

Aim: Write C program to Find solution of given Simultaneous Equations Using Gauss–seidel iterative method.

Theory:

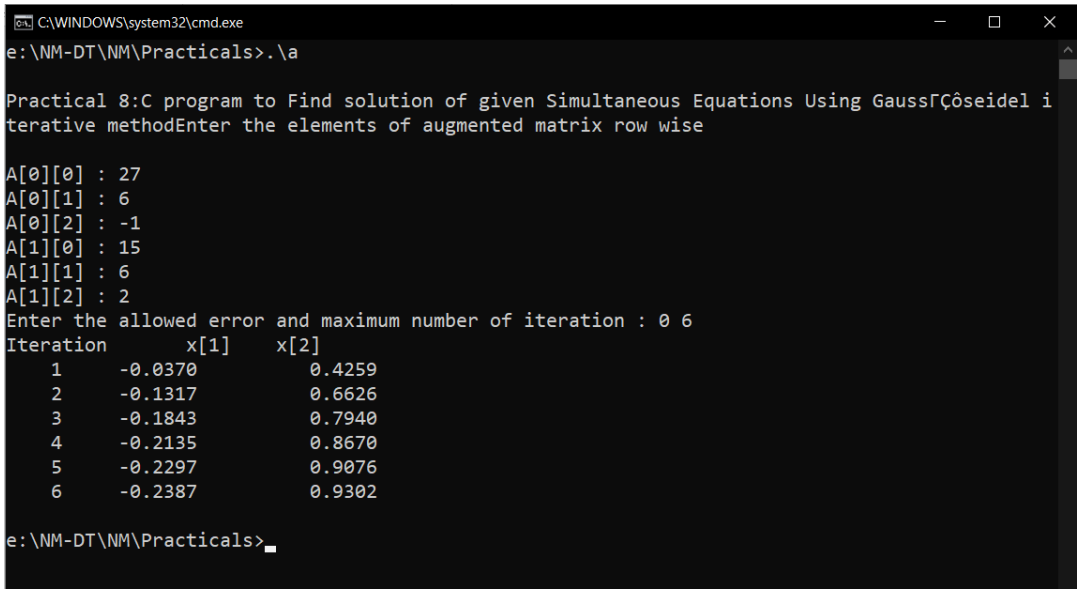
Gauss Seidel Method is an iterative method in numerical analysis, every **solution** attempt is started with an approximate **solution** of an equation and iteration is performed until the desired accuracy is obtained. In Gauss-Seidel method, the most recent values are used in successive iterations.

Program in C:

```
#include <stdio.h>
#include <math.h>
#define X 2
int main()
{
    float x[X][X + 1], a[X], ae, max, t, s, e;
    int i, j, r, mxit;

    printf("\nPractical 8:C program to Find solution of given Simult
aneous Equations Using Gauss-seidel iterative method");
    for (i = 0; i < X; i++)
        a[i] = 0;
    puts("Enter the elements of augmented matrix row wise\n");
    for (i = 0; i < X; i++)
    {
        for (j = 0; j < X + 1; j++)
        {
            printf("A[%d][%d] : ", i, j);
            scanf("%f", &x[i][j]);
        }
    }

    printf("Enter the allowed error and maximum number of iteration
: ");
    scanf("%f%d", &ae, &mxit);
    printf("Iteration\tx[1]\tx[2]\n");
    for (r = 1; r <= mxit; r++)
    {
```

```
        max = 0;
        for (i = 0; i < X; i++)
        {
            s = 0;
            for (j = 0; j < X; j++)
                if (j != i)
                {
                    s += x[i][j] * a[j];
                    t = (x[i][X] - s) / x[i][i];
                    e = fabs(a[i] - t);
                    a[i] = t;
                }
        }
        printf("%5d\t", r);
        for (i = 0; i < X; i++)
            printf("%9.4f\t", a[i]);
        printf("\n");
        if (max < ae)
        {
            printf("Converses in %3d iteration\n", r);
            for (i = 0; i < X; i++)
                printf("a[%3d] = %7.4f\n", i + 1, a[i]);
        }
    }
    return 0;
}
```

Output:



Practical no 9.

Aim: Write C program to implement Newtons divided difference.

Theory:

Interpolation is an estimation of a value within two known values in a sequence of values.
Newton's divided difference interpolation formula is a interpolation technique used when the interval difference is not same for all sequence of values.

Program in C:

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

void main()
{
    int x[10], y[10], p[10];
    int k, f, n, i, j = 1, f1 = 1, f2 = 0;
    printf("Practical 9: Write C program to implement Newtons divide
d difference");
    printf("\nEnter the number of observations : \n");
    scanf("%d", &n);

    printf("\nEnter the different values of x : \n");
    for (i = 1; i <= n; i++)
        scanf("%d", &x[i]);

    printf("\nThe corresponding value of y are : \n");
    for (i = 1; i <= n; i++)
        scanf("%d", &y[i]);

    f = y[1];
    printf("\nEnter the value of 'k' in f(k) you want to evaluate :
\n");
    scanf("%d", &k);

    do
    {
```

```
        for (i = 1; i <= n - 1; i++)
        {
            p[i] = ((y[i + 1] - y[i]) / (x[i + j] - x[i]));
            y[i] = p[i];
        }

        f1 = 1;
        for (i = 1; i <= j; i++)
        {
            f1 *= (k - x[i]);
        }

        f2 += (y[1] * f1);
        n--;
        j++;
    }

    while (n != 1);
    f += f2;
    printf("\nf(%d) = %d", k, f);
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Practical 9: Write C program to implement Newtons divided difference
Enter the number of observations :
4

Enter the different values of x :
0
1
2
3

The corresponding value of y are :
1
2
1
10

Enter the value of 'k' in f(k) you want to evaluate :
4

f(4) = 41
e:\NM-DT\NM\Practicals>
```

Practical no 10.

Aim: Write C program to implement Lagrange's interpolation method for finding x.

Theory: In Lagrange interpolation in **C** language, x and y are defined as arrays so that a number of data can be stored under a single variable name. ... At this step, the value of 'y' is computed in loops using Lagrange interpolation formula.

Program in C:

```
#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;

    printf("\nPractical 10:Write C program to implement lagrange's i
nterpolation method for finding x. ");
    printf("\nHow many record you will write : ");
    scanf("%d", &n);
    printf("\n");

    for (i = 0; i < n; i++)
    {
        printf("\nEnter value of x%d: ", i);
        scanf("%f", &x[i]);
        printf("\nEnter the value of f(x%d): ", i);
        scanf("%d", &y[i]);
    }

    printf("\n\nEnter f(x) for finding 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 - y[j]) / (y[k] - y[j]));
            }
        }
        f[i] = x[i] * temp;
    }

    for (i = 0; i < n; i++)
    {
        sum = sum + f[i];
    }
    printf("\n\n    x = %f   *", sum);
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe

E:\NM-DT\NM\Practicals>.\a

Practical 10:Write C program to implement lagrange's interpolation method for finding x.
How many record you will write : 3


Enter value of x0: 0

Enter the value of f(x0): 648

Enter value of x1: 2

Enter the value of f(x1): 704

Enter value of x2: 3

Enter the value of f(x2): 729


Enter f(x) for finding x: 4


    x = -1.#IND00
E:\NM-DT\NM\Practicals>
```

Practical no 11.

Aim: Write C program for trapezoidal method

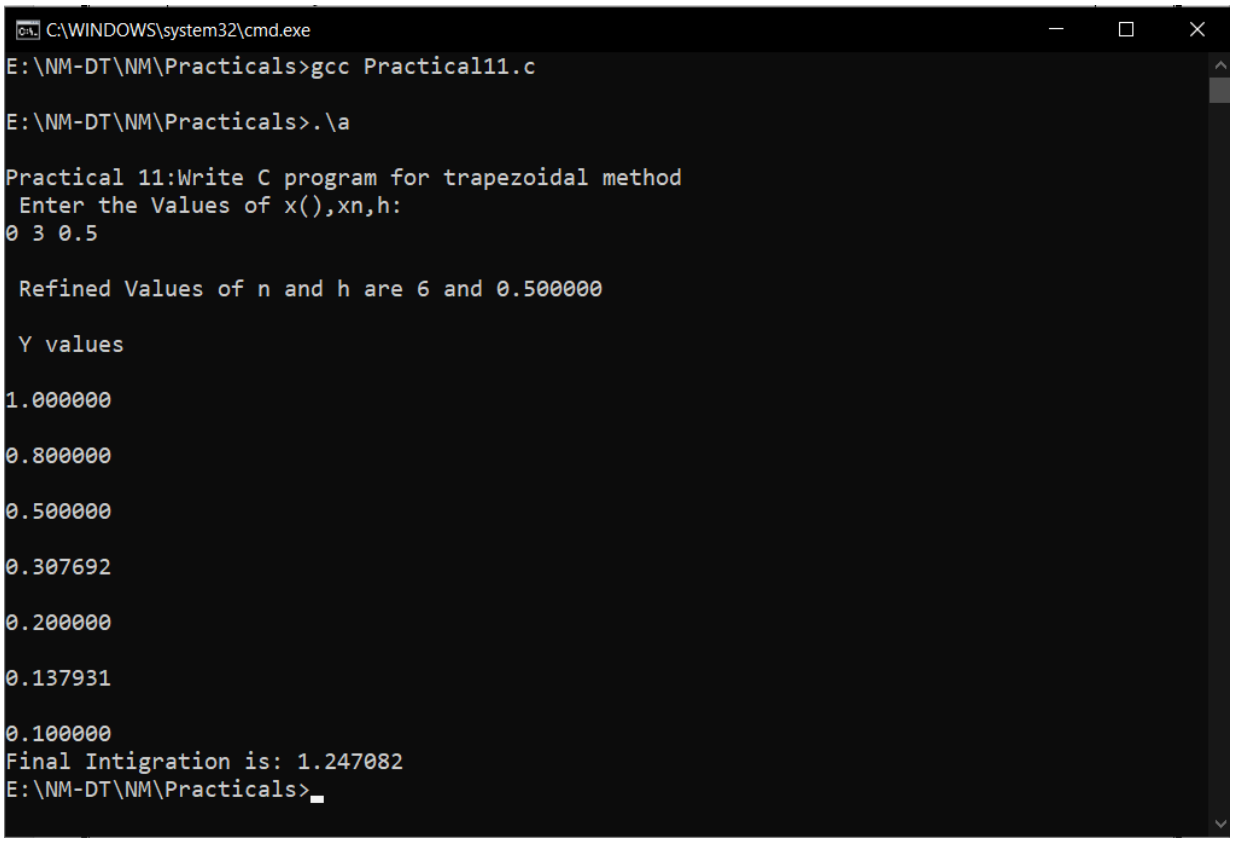
Theory: Trapezoidal Rule is a Numerical technique to find the definite integral of a function. Then the area of trapeziums is calculated to find the integral which is basically the area under the curve. The more is the number of trapeziums used, the better is the approximation.

Program in C:

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

float f(float x) { return (1 / (1 + pow(x, 2))); }

int main()
{

    int i, n;
    float x0, xn, h, y[20], s0, se, ans, x[20];
    printf("\nPractical 11:Write C program for trapezoidal method");
    printf("\n Enter the Values of x(),xn,h:\n");
    scanf("%f%f%f", &x0, &xn, &h);
    n = (xn - x0) / h;

    if (n % 2 == 1)
    {
        n = n + 1;
    }
    h = (xn - x0) / n;

    printf("\n Refined Values of n and h are %d and %f \n", n, h);
    printf("\n Y values \n");

    for (i = 0; i <= n; i++)
    {
        x[i] = x0 + i * h;
        y[i] = f(x[i]);
```

```
        printf("\n%f\n", y[i]);
    }
    s0 = 0;
    se = 0;

    for (i = 1; i < n; i++)
    {
        if (i % 2 == 1)
        {
            s0 = s0 + y[i];
        }
        else
        {
            se = se + y[i];
        }
    }
    ans = h / 3 * (y[0] + y[n] + 4 * s0 + 2 * se);
    printf("Final Intigration is: %f", ans);
    return 0;
}
```

Output:



Practical no 12.

Aim: Write C program for Sympson’s 1/3 rd rule.

Theory: **Simpson's Rule** is a Numerical technique to find the definite integral of a function within a given interval. And the area is then calculated to find the integral. ... The more is the number of sub-intervals used, the better is the approximation.

Program in C:

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

float f(float x)
{
    return (1 / (1 + x));
}
int main()
{
    int i, n;
    float x0, xn, h, y[20], s0, se, ans, x[20];
    printf("Practical 12:Write C program for Sympson’s 1/3 rd rule")
;
    printf("\n Enter  values of x0,xn,h:");
    scanf("%f%f%f", &x0, &xn, &h);

    n = (xn - x0) / h;
    if (n % 2 == 1)
    {
        n = n + 1;
    }

    h = (xn - x0) / h;

    printf("\n Refined Value of n and h are %d%f\n", n, h);
    printf("\n Y values: \n");

    for (i = 0; i <= n; i++)
    {
```

```
        x[i] = x0 + i * h;
        y[i] = f(x[i]);
        printf("\n %f\n", y[i]);
    }
    s0 = 0;
    se = 0;

    for (i = 1; i < n; i++)
    {
        if (i % 2 == 1)
        {
            s0 = s0 + y[i];
        }
        else
        {
            se = se + y[i];
        }
    }

    ans = h / 3 * (y[0] + y[n] + 4 * s0 + 2 * se);
    printf("\n Final Integration is %f", ans);

    return 0;
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe

E:\NM-DT\NM\Practicals>gcc Practical12.c

E:\NM-DT\NM\Practicals>.\a
Practical 12:Write C program for SympsonΓÇÖs 1/3 rd rule
 Enter  values of x0,xn,h:0 3 0.5

 Refined Value of n and h are 66.000000

 Y values:

 1.000000

 0.142857

 0.076923

 0.052632

 0.040000

 0.032258

 0.027027

 Final Integration is 4.343720
E:\NM-DT\NM\Practicals>
```