Master of Science in Mathematics

(M.Sc. Mathematics)

Numerical & Statistical Techniques Lab

(DMSMCO201P24)

Self-Learning Material

(SEMESTER -II)

Jaipur National University
Directorate of Distance Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

EXPERT COMMITTEE

1. **Dr. Vikas Gupta**

Dean
Department of Mathematics
LNMIIT, Jaipur

2. Dr. Nawal Kishor Jangid

Department of Mathematics
SKIT, Jaipur

COURSE COORDINATOR

Mr. Praveen Kumar
Dept. of Basic Sciences,
JNU Jaipur

QUESTION PREPARATION

Question Writers	**Assisting & Proof Reading**	**Question Editor**
Dr. Yogesh Khandelwal Dept. of Basic Sciences JNU, Jaipur Question 1 - 8	Dr. Sanju Jangid Dept. of Basic Sciences JNU, Jaipur	Prof. Hoshiyar Singh Dept. of Basic Sciences JNU, Jaipur
Dr. Sanju Jangid Dept. of Basic Sciences JNU, Jaipur Question 9 - 15		

Secretarial Assistance:

Mr. Mukesh Sharma

COURSE INTRODUCTION

The Numerical & Statistical Techniques Lab provides practical experience in applying numerical methods and statistical techniques to solve engineering and scientific problems. This lab course is designed to complement the theoretical knowledge gained in lectures, offering hands-on opportunities to implement algorithms and analyze data using computational tools.

The course is of two credits and divided into 15 Questions.

Course Outcomes:

1. Recall the numerical methods to obtain approximate solutions of mathematical problems.
2. Explain error, source of error and its effect on any numerical computation and also
3. Analyze the efficiency of any numerical algorithm.
4. Solve a system of linear equations numerically using direct and iterative methods.
5. Analyze the accuracy of common numerical methods.
6. Evaluate numerical solution of nonlinear equations using Bisection, Newton – Raphson and fixed-point iteration methods.
7. Create interpolating polynomials with practical exposure.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Question 1: Floating Point Representation

Program Statement:

Write programs in C/C++ to implement Floating Point Representation of the following:

a) Addition
b) Subtraction
c) Multiplication
d) Division

Solution:

a) Addition

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, c;
clrscr();
printf("Enter the value of a and b");
scanf("%d%d",&a,&b);
c = a + b;
printf("the addition is %d",c);
getch();
}
```

Output:

Enter the value of a and b: 5 7

The addition is 12

b) Subtraction

```
#include<stdio.h>
#include<conio.h>
void Inain()
{
int a, b, c;
clrscr();
```

```
printf("Enter the value of a and b");
scanf("%d%d", &a,&b);
c = a - b;
printf("the substraction is %d", c);
getch();
}
```

Output:

Enter the value of a and b: 10 5

The subtraction is 5

c) **Multiplication**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, c;
cirscr():
printf("Enter the value of a and b");
scanf("%d%d",&a,&b);
c = a * b;
printf("the multiplication is % d",c);
getch();
}
```

Output:

Enter the value of a and b: 8 6

The multiplication is 48

d) Division

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, c;
cirscr():
printf("Enter the value of a and b'");
scanf("%d%d",&a,&b);
c = a/b;
printf("the division is % d",c);
getch();
}
```

Output:

Enter the value of a and b: 10 2

The division is 5

Question 2: Bisection Method

Program Statement:

WAP in “C” Language to find out the root of the Algebraic and Transcendental equations using Bisection Method.

Solution:

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

// Define the equation you want to find the root of
float equation(float x) {
  // Example equation: x^3 - 2x^2 + 3x - 6 = 0
  return x * x * x - 2 * x * x + 3 * x - 6;
}

// Function to find the root using the bisection method
float bisection(float a, float b, float tolerance) {
  if (equation(a) * equation(b) >= 0) {
    printf("Invalid interval, f(a) and f(b) must have opposite signs\n");
    return -1;
  }
```

```
 float c;
 int iterations = 0;
 while ((b - a) >= tolerance) {
   // Find the midpoint
   c = (a + b) / 2;

   // Check if the midpoint is the root
   if (equation(c) == 0.0)
     break;
   // Update the interval
   else if (equation(c) * equation(a) < 0)
     b = c;
   else
     a = c;


  }

  printf("Number of iterations: %d\n", iterations);
  return c;
}

int main() {
  float a, b, tolerance, root;

  // Input the interval [a, b] and tolerance
  printf("Enter the interval [a, b]: ");
  scanf("%f %f", &a, &b);
  printf("Enter the tolerance: ");
  scanf("%f", &tolerance);

  // Find the root using the bisection method
  root = bisection(a, b, tolerance);
```

```
  // Print the root if found
  if (root != -1)
      printf("Root: %.6f\n", root);

  return 0;
}
```

Output:

```
Enter two initial guesses:
0
1
Enter tolerable error:
0.0001

Step    x0          x1          x2          f(x2)
1       0.000000    1.000000    0.500000    0.053222
2       0.500000    1.000000    0.750000    -0.856061
3       0.500000    0.750000    0.625000    -0.356691
4       0.500000    0.625000    0.562500    -0.141294
5       0.500000    0.562500    0.531250    -0.041512
6       0.500000    0.531250    0.515625    0.006475
7       0.515625    0.531250    0.523438    -0.017362
8       0.515625    0.523438    0.519531    -0.005404
9       0.515625    0.519531    0.517578    0.000545
10      0.517578    0.519531    0.518555    -0.002427
11      0.517578    0.518555    0.518066    -0.000940
12      0.517578    0.518066    0.517822    -0.000197
13      0.517578    0.517822    0.517700    0.000174
14      0.517700    0.517822    0.517761    -0.000012

Root is: 0.517761
```

Question 3: Regula Falsi Method

Program Statement:

WAP in "C" Language to find out the root of the Algebraic and Transcendental equations using Regula Falsi Method

Solution:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
/* defining equation to be solved.
  Change this equation to solve another problem. */
#define   f(x)   x*log10(x) - 1.2

int main()
{

        float x0, x1, x2, f0, f1, f2, e;
        int step = 1;
        clrscr();
        /* Inputs */

up:
printf("\nEnter two initial guesses:\n");
scanf("%f%f", &x0, &x1);
printf("Enter tolerable error:\n");
scanf("%f", &e);
/* Calculating Functional Values */
f0 = f(x0);
f1 = f(x1);
/* Checking whether given guesses brackets the root or not. */
if( f0*f1 > 0.0)
{
        printf("Incorrect Initial Guesses.\n");
        goto up;
}
```

```
/* Implementing Regula Falsi or False Position Method */
printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");
do
{
        x2 = x0 - (x0-x1) * f0/(f0-f1);

         f2 = f(x2);
           printf("%d\t\t%f\t%f\t%f\t%f\n",step, x0, x1, x2, f2);

           if(f0*f2 < 0)
           {
                   x1 = x2;
                   f1 = f2;
           }
           else
           {
                   x0 = x2;
                   f0 = f2;
           }
           step = step + 1;

    }while(fabs(f2)>e);

    printf("\nRoot is: %f", x2);
    getch();
```

```
        return 0;
}
```

Output:

Enter two initial guesses:

2

3

Enter tolerable error:

0.000001

Step	x0	x1	x2	f(x2)
1	2.000000	3.000000	2.721014	-0.017091
2	2.721014	3.000000	2.740206	-0.000384
3	2.740206	3.000000	2.740636	-0.000009
4	2.740636	3.000000	2.740646	-0.000000

Root is: 2.740646

Question 4: Newton Raphson Method

Program Statement:

WAP in "C" Language to find out the root of the Algebraic and Transcendental equations using Newton Raphson Method

Solution:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

/* Defining equation to be solved.

   Change this equation to solve another problem. */

#define   f(x)   3*x - cos(x) -1

/* Defining derivative of g(x).

   As you change f(x), change this function also. */

#define  g(x)  3 + sin(x)

void main()

{

        float x0, x1, f0, f1, g0, e;

        int step = 1, N;

        clrscr();

    /* Inputs */

        printf("\nEnter initial guess:\n");

        scanf("%f", &x0);

        printf("Enter tolerable error:\n");

        scanf("%f", &e);

        printf("Enter maximum iteration:\n");

        scanf("%d", &N);
```

```
/* Implementing Newton Raphson Method */
    printf("\nStep\t\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
    do
    {
        g0 = g(x0);
        f0 = f(x0);
        if(g0 == 0.0)
        {
            printf("Mathematical Error.");
            exit(0);
        }
        x1 = x0 - f0/g0;
        printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,f0,x1,f1);
.
        x0 = x1;
        step = step+1;
        if(step > N)
        {
            printf("Not Convergent.");
            exit(0);
        }
        f1 = f(x1);
    }while(fabs(f1)>e);
    printf("\nRoot is: %f", x1);
    getch();
}
```

Output:-

Enter initial guess:

1

Enter tolerable error:

0.00001

Enter maximum iteration:

10

Step	x0	f(x0)	x1	f(x1)
1	1.000000	1.459698	0.620016	0.000000
2	0.620016	0.046179	0.607121	0.046179
3	0.607121	0.000068	0.607102	0.000068

Root is: 0.607102

Question 5: Newton's Forward, Backward and Central Difference Interpolation Table

Program Statement:

WAP in "C" Language to implement Newton's Forward, Backward and Central Difference Interpolation Table.

Solution:

Newton's Forward Difference Interpolation Table

```
#include<stdio.h>

#include<conio.h>


int main()

{

 float x[20], y[20][20];

 int i,j, n;

 clrscr();

 /* Input Section */

 printf("Enter number of data?\n");

 scanf("%d", &n);

 printf("Enter data:\n");

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

 {

  printf("x[%d]=", i);

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

  printf("y[%d]=", i);

  scanf("%f", &y[i][0]);
```

```
}

/* Generating Forward Difference Table */
for(i = 1; i < n; i++)
{
 for(j = 0; j < n-i; j++)
 {
  y[j][i] = y[j+1][i-1] - y[j][i-1];
 }
}

/* Displaying Forward Difference Table */
printf("\nFORWARD DIFFERENCE TABLE\n\n");
for(i = 0; i < n; i++)
{
 printf("%0.2f", x[i]);
 for(j = 0; j < n-i ; j++)
 {
  printf("\t%0.2f", y[i][j]);
 }
 printf("\n");
}

getch(); /* Holding Screen */
return 0;
```

```
}
```

Output:

```
Enter number of data?

5

Enter data:

x[0]=40

y[0]=31

x[1]=50

y[1]=73

x[2]=60

y[2]=124

x[3]=70

y[3]=159

x[4]=80

y[4]=190


FORWARD DIFFERENCE TABLE

40.00   31.00   42.00   9.00    -25.00  37.00

50.00   73.00   51.00   -16.00  12.00

60.00   124.00  35.00   -4.00

70.00   159.00  31.00

80.00   190.00
```

Newton's Backward Difference Interpolation Table

```
#include<studio.h>
```

```
#include<conio.h>
int main()
{
 float x[20], y[20][20];
 int i,j, n;
 clrscr();
 /* Input Section */
 printf("Enter number of data?\n");
 scanf("%d", &n);
 printf("Enter data:\n");
 for(i = 0; i < n ; i++)
 {
  printf("x[%d]=", i);
  scanf("%f", &x[i]);
 printf("y[%d]=", i);
 scanf("%f", &y[i][0]);
 }

 /* Generating Backward Difference Table */
 for(i = 1; i < n; i++)
 {
  for(j = n-1; j > i-1; j--)
  {
   y[j][i] = y[j][i-1]  - y[j-1][i-1];
  }
 }
```

```
/* Displaying Backward Difference Table */
printf("\nBACKWARD DIFFERENCE TABLE\n\n");
for(i = 0; i < n; i++)
{
 printf("%0.2f", x[i]);
 for(j = 0; j <= i ; j++)
 {
  printf("\t%0.2f", y[i][j]);
 }
 printf("\n");
}

getch(); /* Holding Screen */
 return 0;
}
```

Output:

```
Enter number of data?
4
Enter data:
x[0]=0
y[0]=1
x[1]=1
y[1]=2
```

x[2]=2

y[2]=1

x[3]=3

y[3]=10

BACKWARD DIFFERENCE TABLE

0.00 1.00

1.00 2.00 1.00

2.00 1.00 -1.00 -2.00

3.00 10.00 9.00 10.00 12.00

Newton's Central Difference Interpolation Table

```
#include <stdio.h>

// Function to calculate factorial

int factorial(int n) {

    if (n == 0)

        return 1;

    else

        return n * factorial(n - 1);

}


// Function to calculate central differences

void central Difference Table(float x[], float y[], int n, float diff Table[][n]) {

    int i, j;

    // Filling the first column of the difference table with y values
```

```
    for (i = 0; i < n; i++) {
        diffTable[i][0] = y[i];
    }
    // Calculating central differences
    for (j = 1; j < n; j++) {
        for (i = 0; i < n - j; i++) {
            diff Table[i][j] = diff Table[i + 1][j - 1] – diff Table[i][j - 1];
        }
    }
}

// Function to display the difference table
Void display Difference Table (int n, float diff Table[][n]) {
    int i, j;
    printf("Newton's Central Difference Table:\n");
    printf("x\t");
    for (i = 0; i < n; i++) {
        printf("Δ^%d y\t", i);
    }
    printf("\n");
    for (i = 0; i < n; i++) {
        printf("%.2f\t", diff Table[i][0]);
        for (j = 1; j < n - i; j++) {
            printf("%.2f\t", diff Table[i][j]);
```

```
        }
        printf("\n");
    }
}

int main() {
    int n, i;
    printf("Enter the number of data points: ");
    scanf("%d", &n);

    float x[n], y[n], diff Table[n][n];
    printf("Enter the data points:\n");
    for (i = 0; i < n; i++) {
        printf("x%d = ", i);
        scanf("%f", &x[i]);
        printf("y%d = ", i);
        scanf("%f", &y[i]);
    }

    centralDifferenceTable(x, y, n, diffTable);
    displayDifferenceTable(n, diffTable);

    return 0;
}
```

Output:

Enter the number of data points: 5

Enter the data points:

x0 = 0

y0 = 1

x1 = 1

y1 = 2

x2 = 2

y2 = 3

x3 = 3

y3 = 4

x4 = 4

y4 = 5

Newton's Central Difference Table:

x	Δ^0 y	Δ^1 y	Δ^2 y	Δ^3 y	Δ^4 y
1.00	1.00	1.00	1.00	1.00	
2.00	2.00	1.00	1.00		
3.00	3.00	1.00			
4.00	4.00				
5.00					

Question 6: Newton's Forward, Backward and Central Difference interpolation formula

Program Statement:

WAP in "C" Language to implement Newton's Forward, Backward and Central Difference interpolation formula.

Solution:

Newton's Forward Difference interpolation formula

```
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
   if (n == 0 || n == 1)
      return 1;
   else
      return n * factorial(n - 1);
}

// Function to calculate the forward difference table
void forward Difference Table(float x[], float y[][10], int n) {
   int i, j;
   for (i = 1; i < n; i++) {
      for (j = 0; j < n - i; j++) {
         y[j][i] = y[j + 1][i - 1] - y[j][i - 1];
```

```
        }
    }
}

// Function to calculate u value
float uValue(float u, int n) {
    float temp = u;
    for (int i = 1; i < n; i++)
        temp = temp * (u + i);
    return temp;
}

// Function to perform interpolation
float calculate Interpolation(float u, int n, float x[], float y[][10]) {
    float result = y[0][0];
    for (int i = 1; i < n; i++) {
        result = result + (uValue(u, i) * y[0][i]) / factorial(i);
    }
  return result;
}

int main() {
  int n, i, j;
  printf("Enter the number of data points: ");
  scanf("%d", &n);

  float x[n], y[n][10];

  printf("Enter the data points in the form of x y: \n");
  for (i = 0; i < n; i++) {
      scanf("%f %f", &x[i], &y[i][0]);
  }
```

```
Forward Difference Table(x, y, n);

    printf("Forward Difference Table:\n");
    for (i = 0; i < n; i++) {
        printf("%0.2f", x[i]);
        for (j = 0; j < n - i; j++) {
            printf("\t%0.2f", y[i][j]);
        }
        printf("\n");
    }

    float value;
    printf("\nEnter the value to interpolate: ");
    scanf("%f", &value);

    float sum = y[0][0];
    float u = (value - x[0]) / (x[1] - x[0]);
    sum += (u * y[0][1]);
    for (int i = 2; i < n; i++) {
        sum += (uValue(u, i) * y[0][i]) / factorial(i);
    }

    printf("\nInterpolated value at %0.2f is %0.2f\n", value, sum);

    return 0;
}
```

Output:

```
Enter the number of data points: 4
Enter the data points in the form of x y:
0 1
1 2
2 5
3 10
```

```
Forward Difference Table:
0.00    1.00    1.00    2.00    4.00
1.00    2.00    3.00    6.00
2.00    5.00    9.00
3.00    10.00

Enter the value to interpolate: 1.5

Interpolated value at 1.50 is 3.50
```

Newton's Backward Difference interpolation formula

```
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}
// Function to calculate the backward difference table
void backward Difference Table(float x[], float y[][10], int n) {
    int i, j;
    for (i = 1; i < n; i++) {
        for (j = n - 1; j >= i; j--) {
            y[j][i] = y[j][i - 1] - y[j - 1][i - 1];
        }
    }
}
```

```
// Function to calculate u value
float uValue(float u, int n) {
    float temp = u;
    for (int i = 1; i < n; i++)
        temp = temp * (u - i);
    return temp;
}

// Function to perform interpolation
float calculate Interpolation(float u, int n, float x[], float y[][10]) {
    float result = y[n - 1][0];
    for (int i = 1; i < n; i++) {
        result = result + (uValue(u, i) * y[n - 1][i]) / factorial(i);
    }
    return result;
}
```


```
// Function to calculate u value
float uValue(float u, int n) {
    float temp = u;
    for (int i = 1; i < n; i++)
        temp = temp * (u - i);
    return temp;
}

// Function to perform interpolation
float calculate Interpolation(float u, int n, float x[], float y[][10]) {
    float result = y[n - 1][0];
    for (int i = 1; i < n; i++) {
        result = result + (uValue(u, i) * y[n - 1][i]) / factorial(i);
    }
    return result;
}
```

```
int main() {
   int n, i, j;
   printf("Enter the number of data points: ");
   scanf("%d", &n);

   float x[n], y[n][10];

   printf("Enter the data points in the form of x y: \n");
   for (i = 0; i < n; i++) {
      scanf("%f %f", &x[i], &y[i][0]);
   }

   Backward Difference Table(x, y, n);

   printf("Backward Difference Table:\n");
   for (i = 0; i < n; i++) {
      printf("%0.2f", x[i]);
      for (j = 0; j <= i; j++) {
        printf("\t%0.2f", y[i][j]);
     }
     printf("\n");
  }

  float value;
  printf("\nEnter the value to interpolate: ");
  scanf("%f", &value);

  float sum = y[n - 1][0];
  float u = (value - x[n - 1]) / (x[1] - x[0]);
  sum += (u * y[n - 1][1]);

  for (int i = 2; i < n; i++) {
     sum += (uValue(u, i) * y[n - 1][i]) / factorial(i);
  }
```

```
    printf("\nInterpolated value at %0.2f is %0.2f\n", value, sum);
    return 0;
}
```

Output:

```
Enter the number of data points: 4
Enter the data points in the form of x y:
0 1
1 2
2 5
3 10
Backward Difference Table:
0.00    1.00    1.00    1.00    1.00
1.00    2.00    3.00    4.00
2.00    5.00    9.00
3.00    10.00

Enter the value to interpolate: 1.5
Interpolated value at 1.50 is 3.50
```

Newton's Central Difference interpolation formula

```
        int i, j;
        for (i = 1; i < n; i++) {
            for (j = n - 1; j >= i; j--) {
                y[j][i] = y[j][i - 1] - y[j - 1][i - 1];
            }
        }
    }

    // Function to calculate u value
    float uValue(float u, int n) {
        float temp = u / 2;
        for (int i = 1; i < n; i++)
            temp = temp * (u - i) / (i + 1);
        return temp;
    }

    // Function to perform interpolation
    float calculateInterpolation(float u, int n, float x[], float y[][10]) {
        float result = y[n / 2][0];
    for (int i = 1; i < n; i++) {
        result = result + (uValue(u, i) * y[n / 2][i]) / factorial(i);
    }
    return result;
}

int main() {
    int n, i, j;
    printf("Enter the number of data points: ");
    scanf("%d", &n);

    float x[n], y[n][10];
```

```
 float x[n], y[n][10];

 printf("Enter the data points in the form of x y: \n");
 for (i = 0; i < n; i++) {
    scanf("%f %f", &x[i], &y[i][0]);
 }
Central DifferenceTable(x, y, n);

printf("Central Difference Table:\n");
for (i = 0; i < n; i++) {
   printf("%0.2f", x[i]);
   for (j = 0; j <= i; j++) {
      printf("\t%0.2f", y[i][j]);
   }
   printf("\n");
}

float value;
printf("\nEnter the value to interpolate: ");
scanf("%f", &value);

float h = x[1] - x[0];

 float u = (value - x[n / 2]) / h;
 float sum = calculate Interpolation(u, n, x, y);

 printf("\nInterpolated value at %0.2f is %0.2f\n", value, sum);

 return 0;
}
```

Output:

Enter the number of data points: 5

Enter the data points in the form of x y:

0 1

1 2

2 5

3 10

4 17

Central Difference Table:

0.00	1.00	1.00	1.00	1.00	1.00
1.00	2.00	3.00	4.00	5.00	
2.00	5.00	9.00	14.00		
3.00	10.00	19.00			
4.00	17.00				

Enter the value to interpolate: 2.5

Interpolated value at 2.50 is 7.00

Question 7: Jacobi's interpolation Method

Program Statement:

WAP in "C" Language to implement Jacobi's interpolation Method.

Solution:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


/* Arrange systems of linear

   equations to be solved in

   diagonally dominant form

   and form equation for each

   unknown and define here

*/

/* In this example we are solving


   3x + 20y - z = -18

   2x - 3y + 20z = 25

   20x + y - 2z = 17

*/

/* Arranging given system of linear

   equations in diagonally dominant
```

```
    form:
    20x + y - 2z = 17
    3x + 20y -z = -18
    2x - 3y + 20z = 25
  */
  /* Equations:
    x = (17-y+2z)/20
  y = (-18-3x+z)/20
  z = (25-2x+3y)/20
*/
/* Defining function */
#define f1(x,y,z)  (17-y+2*z)/20
#define f2(x,y,z)  (-18-3*x+z)/20
#define f3(x,y,z)  (25-2*x+3*y)/20

/* Main function */
int main()
{
 float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3, e;
 int count=1;
```

```
clrscr();
printf("Enter tolerable error:\n");
scanf("%f", &e);

printf("\nCount\tx\ty\tz\n");
do
{
 /* Calculation */
 x1 = f1(x0,y0,z0);
 y1 = f2(x0,y0,z0);
 z1 = f3(x0,y0,z0);
  printf("%d\t%0.4f\t%0.4f\t%0.4f\n",count, x1,y1,z1);

  /* Error */
  e1 = fabs(x0-x1);
  e2 = fabs(y0-y1);
  e3 = fabs(z0-z1);
  count++;
  /* Set value for next iteration */
  x0 = x1;
  y0 = y1;
  z0 = z1;
 }while(e1>e && e2>e && e3>e);
 printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);
```

```
 getch();
 return 0;
}
```

Output:-

Enter tolerable error:

0.0001

Count	x	y	z
1	0.8500	-0.9000	1.2500
2	1.0200	-0.9650	1.0300
3	1.0013	-1.0015	1.0033
4	1.0004	-1.0000	0.9997
5	1.0000	-1.0001	1.0000

Solution: x=1.000, y=-1.000 and z = 1.000

Question 8: Gauss – Seidel interpolation Method

Program Statement:

WAP in "C" Language to implement Gauss – Seidel interpolation Method.

Solution:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


/* Arrange systems of linear

   equations to be solved in

   diagonally dominant form

   and form equation for each

   unknown and define here

*/

/* In this example we are solving

   3x + 20y - z = -18

   2x - 3y + 20z = 25
```

```
   20x + y - 2z = 17
 */
 /* Arranging given system of linear
   equations in diagonally dominant
   form:
   20x + y - 2z = 17
   3x + 20y -z = -18
   2x - 3y + 20z = 25
 */
 /* Equations:
   x = (17-y+2z)/20
   y = (-18-3x+z)/20
   z = (25-2x+3y)/20
*/
/* Defining function */
#define f1(x,y,z)  (17-y+2*z)/20
#define f2(x,y,z)  (-18-3*x+z)/20
#define f3(x,y,z)  (25-2*x+3*y)/20

/* Main function */
int main()
{
 float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3, e;
 int count=1;
```

```
clrscr();
printf("Enter tolerable error:\n");
scanf("%f", &e);

printf("\nCount\tx\ty\tz\n");
do
{
 /* Calculation */
 x1 = f1(x0,y0,z0);
 y1 = f2(x1,y0,z0);
 z1 = f3(x1,y1,z0);
 printf("%d\t%0.4f\t%0.4f\t%0.4f\n",count, x1,y1,z1);
 /* Error */
 e1 = fabs(x0-x1);
  e2 = fabs(y0-y1);
  e3 = fabs(z0-z1);
  count++;
  /* Set value for next iteration */
  x0 = x1;
  y0 = y1;
  z0 = z1;
 }while(e1>e && e2>e && e3>e);
 printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);
 getch();
 return 0;
```

}

Output:-

Enter tolerable error:

0.0001

Count	x	y	z
1	0.8500	-1.0275	1.0109
2	1.0025	-0.9998	0.9998
3	1.0000	-1.0000	1.0000
4	1.0000	-1.0000	1.0000

Solution: x=1.000, y=-1.000 and z = 1.000

Question 9: Lagrange's interpolation formula

Program Statement:

WAP in "C" Language to implement Lagrange's interpolation formula.

Solution:

```
#include<stdio.h>

#include<conio.h>


void main()

{

	float x[100], y[100], xp, yp=0, p;

	int i,j,n;

	clrscr();

	/* Input Section */

	printf("Enter number of data: ");

	scanf("%d", &n);

	printf("Enter data:\n");

	for(i=1;i<=n;i++)

	{

		printf("x[%d] = ", i);

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

		printf("y[%d] = ", i);

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

	}

	printf("Enter interpolation point: ");
```

```
	scanf("%f", &xp);
	/* Implementing Lagrange Interpolation */
	for(i=1;i<=n;i++)
	{
		p=1;
		for(j=1;j<=n;j++)
		{
			if(i!=j)
			{
			    p = p* (xp - x[j])/(x[i] - x[j]);
			}
		}
		yp = yp + p * y[i];
	}
	printf("Interpolated value at %.3f is %.3f.", xp, yp);
	getch();
}
```

C Program Output: Lagrange Interpolation

Enter number of data: 5 ↲

Enter data:

x[1] = 5 ↲

y[1] = 150 ↲

x[2] = 7 ↲

y[2] = 392 ↲

x[3] = 11 ↲

y[3] = 1452 ↲

x[4] = 13 ↲

y[4] = 2366 ↲

x[5] = 17 ↲

y[5] = 5202 ↲

Enter interpolation point: 9 ↲

Interpolated value at 9.000 is 810.000.

Question 10: Trapezoidal rule

Program Statement:

WAP in “C” Language to implement trapezoidal rule.

Solution:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

/* Define function here */

#define f(x) 1/(1+pow(x,2))

int main()

{

 float lower, upper, integration=0.0, stepSize, k;

 int i, subInterval;

 clrscr();

 /* Input */

 printf("Enter lower limit of integration: ");

 scanf("%f", &lower);

 printf("Enter upper limit of integration: ");
```

```
scanf("%f", &upper);
printf("Enter number of sub intervals: ");
scanf("%d", &subInterval);
/* Calculation */
/* Finding step size */
stepSize = (upper - lower)/subInterval;
/* Finding Integration Value */
integration = f(lower) + f(upper);
for(i=1; i<= subInterval-1; i++)
{
    k = lower + i*stepSize;
    integration = integration + 2 * f(k);
    }
    integration = integration * stepSize/2;
    printf("\nRequired value of integration is: %.3f", integration);
    getch();
    return 0;
    }
```

Output:-

Enter lower limit of integration: 0

Enter upper limit of integration: 1

Enter number of sub intervals: 6

Required value of integration is: 0.784

Question 11: Simpson 1/3 and Simpson 3/8 rule

Program Statement:

WAP in "C" Language to implement Simpson 1/3 and Simpson 3/8 rule.

Solution:

Simpson 1/3 rule:-

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


/* Define function here */

#define f(x) 1/(1+x*x)


int main()

{

 float lower, upper, integration=0.0, stepSize, k;

 int i, subInterval;

 clrscr();

 /* Input */
```

```
printf("Enter lower limit of integration: ");
 scanf("%f", &lower);
 printf("Enter upper limit of integration: ");
 scanf("%f", &upper);
 printf("Enter number of sub intervals: ");
 scanf("%d", &subInterval);

 /* Calculation */
 /* Finding step size */
 stepSize = (upper - lower)/subInterval;

 /* Finding Integration Value */
```

```
integration = f(lower) + f(upper);
 for(i=1; i<= subInterval-1; i++)
 {
  k = lower + i*stepSize;
  if(i%2==0)
  {
   integration = integration + 2 * f(k);
  }
  else
  {
   integration = integration + 4 * f(k);
  }
 }
 integration = integration * stepSize/3;
 printf("\nRequired value of integration is: %.3f", integration);
 getch();
 return 0;
}
```

Output:-

Enter lower limit of integration: 0

Enter upper limit of integration: 1

Enter number of sub intervals: 6

Required value of integration is: 0.785

Simpson 3/8 rule:-

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


/* Define function here */

#define f(x) 1/(1+x*x)


int main()

{

 float lower, upper, integration=0.0, stepSize, k;

 int i, subInterval;

clrscr();

/* Input */

printf("Enter lower limit of integration: ");

scanf("%f", &lower);

printf("Enter upper limit of integration: ");

scanf("%f", &upper);

printf("Enter number of sub intervals: ");

scanf("%d", &subInterval);


/* Calculation */

/* Finding step size */

stepSize = (upper - lower)/subInterval;
```

```
/* Finding Integration Value */

integration = f(lower) + f(upper);

for(i=1; i<= subInterval-1; i++)

{

 k = lower + i*stepSize;

 if(i%3 == 0)

 {

  integration = integration + 2 * f(k);

 }

else

{

 integration = integration + 3 * f(k);

}
```

```
 }
 integration = integration * stepSize*3/8;
 printf("\nRequired value of integration is: %.3f", integration);
 getch();
 return 0;
}
```

Output:-

```
Enter lower limit of integration: 0
Enter upper limit of integration: 1
Enter number of sub intervals: 12


Required value of integration is: 0.785
```

Question 12: Euler Method

Program Statement:

WAP in "C" Language to implement Euler Method.

Solution:

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

#define f(x,y) x+y

int main()
{
 float x0, y0, xn, h, yn, slope;
 int i, n;
 clrscr();
 printf("Enter Initial Condition\n");
 printf("x0 = ");
 scanf("%f", &x0);
printf("y0 = ");
scanf("%f", &y0);
printf("Enter calculation point xn = ");
scanf("%f", &xn);
printf("Enter number of steps: ");
scanf("%d", &n);
```

```
/* Calculating step size (h) */

h = (xn-x0)/n;


/* Euler's Method */

printf("\nx0\ty0\tslope\tyn\n");

 printf("------------------------------\n");

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

 {

  slope = f(x0, y0);

  yn = y0 + h * slope;

  printf("%.4f\t%.4f\t%0.4f\t%.4f\n",x0,y0,slope,yn);

  y0 = yn;

  x0 = x0+h;

 }


/* Displaying result */

printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);

 getch();

 return 0;

}
```

Output:-

Enter Initial Condition

x0 = 0

y0 = 1

Enter calculation point xn = 1

Enter number of steps: 10

```
x0      y0      slope   yn
------------------------------
0.0000  1.0000  1.0000  1.1000
0.1000  1.1000  1.2000  1.2200
0.2000  1.2200  1.4200  1.3620
0.3000  1.3620  1.6620  1.5282
0.4000  1.5282  1.9282  1.7210
0.5000  1.7210  2.2210  1.9431
0.6000  1.9431  2.5431  2.1974
0.7000  2.1974  2.8974  2.4872
0.8000  2.4872  3.2872  2.8159
0.9000  2.8159  3.7159  3.1875

Value of y at x = 1.00 is 3.187
```

Question 13: Euler Modified Method

Program Statement:

WAP in "C" Language to implement Euler Modified Method.

Solution:

```
#include <stdio.h>

// Function to define the ordinary differential equation (ODE)
float f(float x, float y) {
    return x + y; // Example ODE: y' = x + y
}

// Modified Euler Method (Heun's Method) for solving ODE
void modifiedEuler(float x0, float y0, float h, float xn) {
    float y = y0;
    float x = x0;
    while (x < xn) {
        float k1 = h * f(x, y);
        float k2 = h * f(x + h, y + k1);
        y = y + (k1 + k2) / 2;
        x = x + h;
    }
    printf("The value of y at x = %0.2f is %0.6f\n", xn, y);


}

int main() {
    float x0, y0, xn, h;
```

```
    // Input initial values
    printf("Enter initial value of x: ");
    scanf("%f", &x0);
    printf("Enter initial value of y: ");
    scanf("%f", &y0);

    // Input step size and desired value of x
    printf("Enter the step size (h): ");

    scanf("%f", &h);
    printf("Enter the value of x at which y is to be found: ");
    scanf("%f", &xn);

    // Applying Modified Euler Method
    modifiedEuler(x0, y0, h, xn);

    return 0;
}
```

Output:-

```
Enter initial value of x: 0
Enter initial value of y: 1
Enter the step size (h): 0.1
Enter the value of x at which y is to be found: 1
The value of y at x = 1.00 is 3.635868
```

Question 14: Runge-Kutta 2nd and Runge-Kutta 4th order Method

Program Statement:

WAP in “C” Language to implement Runge-Kutta 2nd and Runge-Kutta 4th order Method.

Solution:

Program for RK-2nd Method

```
#include <stdio.h>

// Function to define the differential equation dy/dx

float dydx(float x, float y) {

    return x + y; // Example differential equation

}

// Runge-Kutta second-order method

float rungeKutta2(float x0, float y0, float h) {

    float k1, k2;

    k1 = h * dydx(x0, y0);

    k2 = h * dydx(x0 + h, y0 + k1);

    return y0 + 0.5 * (k1 + k2);

}

int main() {

    float x0, y0, h, xn, yn;

    printf("Enter initial value of x: ");

    scanf("%f", &x0);

    printf("Enter initial value of y: ");

    scanf("%f", &y0);

    printf("Enter step size h: ");

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

```
    printf("Enter value of x at which y is to be found: ");
    scanf("%f", &xn);
    printf("Calculating using Runge-Kutta II Order Method:\n");
    printf("x\t\ty\n");
    while (x0 < xn) {
        yn = rungeKutta2(x0, y0, h);
        printf("%.2f\t\t%.6f\n", x0, y0);
        x0 += h;
        y0 = yn;
    }
    printf("%.2f\t\t%.6f\n", x0, y0);
    return 0;
}
```

Output:-

```
Enter initial value of x: 0
Enter initial value of y: 1
Enter step size h: 0.1
Enter value of x at which y is to be found: 1
Calculating using Runge-Kutta II Order Method:
x               y
0.00            1.000000
0.10            1.110500
0.20            1.242570
0.30            1.398343
```

0.40	1.580288
0.50	1.791398
0.60	2.035281
0.70	2.316446
0.80	2.640659
0.90	3.014217
1.00	3.444168

Program for RK-4th Method

```
#include<stdio.h>

#include<conio.h>

#define f(x,y) (y*y-x*x)/(y*y+x*x)

int main()

{

 float x0, y0, xn, h, yn, k1, k2, k3, k4, k;

 int i, n;

 clrscr();

 printf("Enter Initial Condition\n");

 printf("x0 = ");

 scanf("%f", &x0);

 printf("y0 = ");
```

```
scanf("%f", &y0);
printf("Enter calculation point xn = ");
scanf("%f", &xn);
printf("Enter number of steps: ");
scanf("%d", &n);
/* Calculating step size (h) */
h = (xn-x0)/n;

/* Runge Kutta Method */
printf("\nx0\ty0\tyn\n");
for(i=0; i < n; i++)
{
k1 = h * (f(x0, y0));
k2 = h * (f((x0+h/2), (y0+k1/2)));
k3 = h * (f((x0+h/2), (y0+k2/2)));
k4 = h * (f((x0+h), (y0+k3)));
k = (k1+2*k2+2*k3+k4)/6;
yn = y0 + k;
printf("%0.4f\t%0.4f\t%0.4f\n",x0,y0,yn);
x0 = x0+h;
y0 = yn;
```

```
}
 /* Displaying result */
 printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);
 getch();
 return 0;
}
```

Output:-

```
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.4
Enter number of steps: 2
x0      y0      yn
0.0000  1.0000  1.1960
0.2000  1.1960  1.3753
Value of y at x = 0.40 is 1.375
```

Question 15: Fitting a Straight Line

Program Statement:

WAP in "C" Language to implement Fitting a Straight Line.

Solution:

```
#include<stdio.h>

#include<conio.h>

#define S 50

int main()

{

 int n, i;

 float x[S], y[S], sumX=0, sumX2=0, sumY=0, sumXY=0, a, b;

 clrscr();

 /* Input */

 printf("How many data points?\n");

 scanf("%d", &n);

 printf("Enter data:\n");

 for(i=1;i<=n;i++)

  {

   printf("x[%d]=",i);

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

   printf("y[%d]=",i);

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

  }
```

```
/* Calculating Required Sum */
for(i=1;i<=n;i++)
{
 sumX = sumX + x[i];
 sumX2 = sumX2 + x[i]*x[i];
 sumY = sumY + y[i];
 sumXY = sumXY + x[i]*y[i];
}
/* Calculating a and b */
b = (n*sumXY-sumX*sumY)/(n*sumX2-sumX*sumX);
a = (sumY - b*sumX)/n;
/* Displaying value of a and b */
printf("Values are: a=%0.2f and b = %0.2f",a,b);
printf("\nEquation of best fit is: y = %0.2f + %0.2fx",a,b);
getch();
return(0);
}
```

Output:-

```
Enter data:
x[1] = 0
y[1] = -1
x[2] = 2
y[2] = 5
x[3] = 5
```

y[3] = 12

x[4] = 7

y[4] = 20

Values are: a=-1.14 and b=2.90

Equation of best fit is: y = -1.14 + 2.90x