DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI DISTRICT NAINITAL-263132

2024-2025

Term work of

CBNST Lab

(PMA - 502)

Submitted in partial fulfillment of the requirement for the V semester

Bachelor of Technology

By

Varun Kharkwal

University Roll No

2261649

Under the Guidance of

Mr. Parthak Mehra

Assistant Professor

Dept. of CSE

INDEX

SR. No.	Objective	Date	Sign
01.	WAP to find the roots of non-linear equation using Bisection method.		
02.	WAP to find the roots of non-linear equation using False position method.		
03.	WAP to find the roots of non-linear equation using Newton’s Raphson method.		
04.	WAP to find the roots of non-linear equation using Iteration method.		
05.	WAP to interpolate numerically using Newton’s forward difference method.		
06.	WAP to interpolate numerically using Newton’s backward difference method.		
07.	WAP to interpolate numerically using Lagrange’s method.		
08.	WAP to Integrate numerically using Trapezoidal rule.		
09.	WAP to Integrate numerically using Simpson’s 1/3 rules.		
10.	WAP to Integrate numerically using Simpson’s 3/8 rules.		

1. WAP to find the roots of non-linear equation using Bisection method.

```
#include<bits/stdc++.h>
using namespace std;

double fun(double x)
{
   return x*x*x - 4*x - 9;
}

int main()
{
   double x0, x1, x2;
   cout << "Enter the interval for your function: ";
   cin >> x0 >> x1;

   if(fun(x0) * fun(x1) > 0) {
      cout << "No root in this interval." << endl;
      return 0;
   }

   int i;
   for(i = 0; i < 10; i++)
   {
      x2 = (x0 + x1) / 2;

      if(fun(x2) == 0)
      {
         cout << "Exact root: " << x2 << endl;
         return 0;
      }
      else if(fun(x0) * fun(x2) < 0)
      {
         x1 = x2;  // Root lies in left subinterval
      }
      else
      {
         x0 = x2;  // Root lies in right subinterval
      }
   }

   cout << "Root is approximately: " << x2 << endl;
   return 0;
}
```

2. WAP to find the roots of non-linear equation using False position method.

```
#include<bits/stdc++.h>
using namespace std;

double fun(double x)
{
	return x*x*x  - x - 1;
}

int main()
{
	double x0,x1,x2;
	cout<<"enter the interval for your function "<< endl;
	cin>>x0>>x1;

	int i;
	for(i=0;i<10;i++)
	{
		x2=(x0*fun(x1) - x1*fun(x0))/(fun(x1) - fun(x0));

		if(fun(x2)==0)
		{
			cout<<x2;
			return 0;
		}
		else if(fun(x0)*fun(x2)<0)
		{
			x1=x2;
		}
		else if(fun(x1)*fun(x2)<0)
		{
			x0=x2;
		}
	}
cout<<x2;
return 0;
}
```

3. WAP to find the roots of non-linear equation using Newton's Raphson method.

```
#include<bits/stdc++.h>
using namespace std;

double fun(double x)
{
   return x*x*x - 4*x - 9;
}

double derivative(double x)
{
   return 3*x*x - 1;
}

int main()
{
   double x0, x1;
   cout << "Enter the initial guess: ";
   cin >> x0;

   int i;
   for(i = 0; i < 10; i++)
   {
      x1 = x0 - fun(x0)/derivative(x0);

      if(fun(x1) == 0)  // If root is found
      {
         cout << "Exact root: " << x1 << endl;
         return 0;
      }

      x0 = x1;  // Update x0 for next iteration
   }

   cout << "Root is approximately: " << x1 << endl;
   return 0;
}
```

4. WAP to find the roots of non-linear equation using Iteration method.

```
#include<bits/stdc++.h>
using namespace std;

double g(double x)
{
   return cbrt(x + 1);  // Rearranged function x = g(x)
}

int main()
{
   double x0, x1;
   cout << "Enter the initial guess: ";
   cin >> x0;

   int i;
   for(i = 0; i < 10; i++)  // Adjust the number of iterations or use tolerance
   {
      x1 = g(x0);  // Fixed point iteration formula: x1 = g(x0)

      if(fabs(x1 - x0) < 1e-6)  // If the difference is small enough, stop
      {
         cout << "Root is approximately: " << x1 << endl;
         return 0;
      }

      x0 = x1;  // Update x0 for the next iteration
   }

   cout << "Root after iterations is approximately: " << x1 << endl;
   return 0;
}
```

5. WAP to interpolate numerically using Newton's forward difference method.

```
#include <iostream>
#include <iomanip>
using namespace std;

// Function to calculate factorial
int factorial(int n) {
   int fact = 1;
   for (int i = 2; i <= n; i++) {
      fact *= i;
   }
   return fact;
}

int main() {
   int n;
   cout << "Enter the number of data points: ";
   cin >> n;

   double x[n], y[n][n]; // y is the forward difference table

   // Input data points
   cout << "Enter the x and y values:\n";
   for (int i = 0; i < n; i++) {
      cout << "x[" << i << "]: ";
      cin >> x[i];
      cout << "y[" << i << "]: ";
      cin >> y[i][0];
   }

   // Calculate the forward difference table
   for (int j = 1; j < n; j++) {
      for (int i = 0; i < n - j; i++) {
         y[i][j] = y[i + 1][j - 1] - y[i][j - 1];
      }
   }

   // Display the forward difference table
   cout << "\nForward Difference Table:\n";
   for (int i = 0; i < n; i++) {
      cout << setw(10) << x[i];
      for (int j = 0; j < n - i; j++) {
         cout << setw(10) << y[i][j];
      }
      cout << endl;
   }
```

```
    double xp;
    cout << "\nEnter the x value to interpolate: ";
    cin >> xp;

    // Newton Forward Interpolation
    double h = x[1] - x[0]; // Assuming equal spacing
    double u = (xp - x[0]) / h;
    double yp = y[0][0]; // Initial value of interpolated result

    for (int i = 1; i < n; i++) {
        double term = y[0][i];
        for (int j = 0; j < i; j++) {
            term *= (u - j);
        }
        term /= factorial(i);
        yp += term;
    }

    // Display the result
    cout << "\nInterpolated value at x = " << xp << " is " << yp << endl;

    return 0;
}
```

6. WAP to interpolate numerically using Newton's backward difference method.

```
#include <iostream>
#include <iomanip>
using namespace std;

// Function to calculate factorial
int factorial(int n) {
    int fact = 1;
    for (int i = 2; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

int main() {
    int n;
    cout << "Enter the number of data points: ";
    cin >> n;

    double x[n], y[n][n]; // y is the backward difference table

    // Input data points
    cout << "Enter the x and y values:\n";
    for (int i = 0; i < n; i++) {
        cout << "x[" << i << "]: ";
        cin >> x[i];
        cout << "y[" << i << "]: ";
        cin >> y[i][0];
    }

    // Calculate the backward difference table
    for (int j = 1; j < n; j++) {
        for (int i = n - 1; i >= j; i--) {
            y[i][j] = y[i][j - 1] - y[i - 1][j - 1];
        }
    }

    // Display the backward difference table
    cout << "\nBackward Difference Table:\n";
    for (int i = 0; i < n; i++) {
        cout << setw(10) << x[i];
        for (int j = 0; j <= i; j++) {
            cout << setw(10) << y[i][j];
        }
        cout << endl;
    }
```

```
    double xp;
    cout << "\nEnter the x value to interpolate: ";
    cin >> xp;

    // Newton Backward Interpolation
    double h = x[1] - x[0]; // Assuming equal spacing
    double u = (xp - x[n - 1]) / h;
    double yp = y[n - 1][0]; // Initial value of interpolated result

    for (int i = 1; i < n; i++) {
        double term = y[n - 1][i];
        for (int j = 1; j <= i; j++) {
            term *= (u + j - 1);
        }
        term /= factorial(i);
        yp += term;
    }

    // Display the result
    cout << "\nInterpolated value at x = " << xp << " is " << yp << endl;

    return 0;
}
```

7. WAP to interpolate numerically using Lagrange's method.

```
#include <iostream>
using namespace std;

// Function to perform Lagrange Interpolation
double lagrangeInterpolation(double x[], double y[], int n, double xp) {
   double yp = 0;  // Initial value of interpolated result

   // Loop through each term of the Lagrange formula
   for (int i = 0; i < n; i++) {
      double term = y[i];
      for (int j = 0; j < n; j++) {
         if (j != i) {
            term *= (xp - x[j]) / (x[i] - x[j]);
         }
      }
      yp += term;  // Add term to final result
   }

   return yp;
}

int main() {
   int n;
   cout << "Enter the number of data points: ";
   cin >> n;

   double x[n], y[n];
   cout << "Enter the x and y values:\n";
   for (int i = 0; i < n; i++) {
      cout << "x[" << i << "]: ";
      cin >> x[i];
      cout << "y[" << i << "]: ";
      cin >> y[i];
   }
   double xp;
   cout << "Enter the x value to interpolate: ";
   cin >> xp;

   // Call the Lagrange interpolation function
   double yp = lagrangeInterpolation(x, y, n, xp);

   // Display the result
   cout << "\nInterpolated value at x = " << xp << " is " << yp << endl;
   return 0;
}
```

8. WAP to Integrate numerically using Trapezoidal rule.

```
#include <iostream>
#include <cmath>
using namespace std;

// Define the function to integrate
double f(double x) {
    return x * x; // Example: f(x) = x^2
}

// Function to calculate the integral using the Trapezoidal Rule
double trapezoidalRule(double a, double b, int n) {
    double h = (b - a) / n; // Calculate the width of each subinterval
    double sum = f(a) + f(b); // Add the first and last terms

    // Calculate the sum of the middle terms
    for (int i = 1; i < n; i++) {
        double x = a + i * h;
        sum += 2 * f(x);
    }

    // Apply the trapezoidal rule formula
    return (h / 2) * sum;
}

int main() {
    double a, b;
    int n;

    // Input the limits of integration and number of subintervals
    cout << "Enter the lower limit (a): ";
    cin >> a;
    cout << "Enter the upper limit (b): ";
    cin >> b;
    cout << "Enter the number of subintervals (n): ";
    cin >> n;

    // Calculate the integral
    double result = trapezoidalRule(a, b, n);

    // Display the result
    cout << "\nThe integral value using Trapezoidal Rule is: " << result << endl;

    return 0;
}
```

9. WAP to Integrate numerically using Simpson’s 1/3 rules.

```
#include <iostream>
#include <cmath>
using namespace std;

// Define the function to integrate
double f(double x) {
  return x * x; // Example: f(x) = x^2
}

// Function to calculate the integral using Simpson's 1/3 Rule
double simpsonsRule(double a, double b, int n) {
  // Check if n is even
  if (n % 2 != 0) {
    cout << "Error: Number of subintervals (n) must be even.\n";
    return -1;
  }

  double h = (b - a) / n; // Calculate the width of each subinterval
  double sum = f(a) + f(b); // Add the first and last terms

  // Calculate the sum of odd terms (4 * f(x_i))
  for (int i = 1; i < n; i += 2) {
    double x = a + i * h;
    sum += 4 * f(x);
  }

  // Calculate the sum of even terms (2 * f(x_i))
  for (int i = 2; i < n; i += 2) {
    double x = a + i * h;
    sum += 2 * f(x);
  }

  // Apply the Simpson's 1/3 Rule formula
  return (h / 3) * sum;
}

int main() {
  double a, b;
  int n;

  // Input the limits of integration and number of subintervals
  cout << "Enter the lower limit (a): ";
  cin >> a;
  cout << "Enter the upper limit (b): ";
  cin >> b;
```

```
    cout << "Enter the number of subintervals (n): ";
    cin >> n;

    // Calculate the integral
    double result = simpsonsRule(a, b, n);

    // Display the result if n is valid
    if (result != -1) {
        cout << "\nThe integral value using Simpson's 1/3 Rule is: " << result << endl;
    }

    return 0;
}
```

10. WAP to Integrate numerically using Simpson's 3/8 rules.

```
#include <iostream>
#include <cmath>
using namespace std;

// Define the function to integrate
double f(double x) {
    return x * x; // Example: f(x) = x^2
}

// Function to calculate the integral using Simpson's 3/8 Rule
double simpsons38Rule(double a, double b, int n) {
    // Check if n is a multiple of 3
    if (n % 3 != 0) {
        cout << "Error: Number of subintervals (n) must be a multiple of 3.\n";
        return -1;
    }

    double h = (b - a) / n; // Calculate the width of each subinterval
    double sum = f(a) + f(b); // Add the first and last terms

    // Calculate the sum of terms multiplied by 3 (odd and most middle points)
    for (int i = 1; i < n; i++) {
        double x = a + i * h;
        if (i % 3 == 0) {
            sum += 2 * f(x); // Every 3rd term gets multiplied by 2
        } else {
            sum += 3 * f(x); // All other terms get multiplied by 3
        }
    }

    // Apply the Simpson's 3/8 Rule formula
    return (3 * h / 8) * sum;
}

int main() {
    double a, b;
    int n;

    // Input the limits of integration and number of subintervals
    cout << "Enter the lower limit (a): ";
    cin >> a;
    cout << "Enter the upper limit (b): ";
    cin >> b;
    cout << "Enter the number of subintervals (n): ";
    cin >> n;
```

```
    // Calculate the integral
    double result = simpsons38Rule(a, b, n);

    // Display the result if n is valid
    if (result != -1) {
        cout << "\nThe integral value using Simpson's 3/8 Rule is: " << result << endl;
    }

    return 0;
}
```