Numerical Method Lab

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

Solution:

Algorithm of Bisection Method for finding root:

1. **Input**: Function $f(x)$, interval $[a, b]$, and a tolerance value.
2. Check if $f(a) \times f(b) \geq 0$. If true, stop: no root exists within this interval.
3. Set $c = \frac{a+b}{2}$.
4. While $|b - a| \geq \text{tolerance}$:

   a. Evaluate $f(c)$.

   b. If $f(c) = 0$, then c is the root. Stop the process.

   c. If $f(c) \times f(a) < 0$, set $b = c$.

   d. If $f(c) \times f(b) < 0$, set $a = c$.

   e. Update $c = \frac{a+b}{2}$.
5. Output the final value of c as the approximate root.

Suppose we have a function: $f(x) = 3x - cos(x) - 1$

Now we need a and b. [0,1]
এখানে a এবং b এর মান নেয়ার সময় একটা কন্ডিশন মাথায় রাখতে হবে। তা হলোঃ $f(a) * f(b) < 0;$

এখানে আমরা a এবং b এর জন্য এমন মান নিবো যাতে ২টা ফাংশন গুন করলে ০ এর ছোট হয়।
এ জন্য আমরা [0,1] এটা না হলে [1,2] এটা না হলে [2,3] এভাবে চলতে থাকবে । তাও না হলে মাইনেস মান দিয়েও আমরা চেক করে দেখবো।

$a = 0 \qquad f(a) = f(0) = 3 * 0 - cos\,0 - 1 = -2$

$b = 1 \qquad f(b) = f(1) = 3 * 1 - cos\,1 - 1 = 1.46$

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

$\Rightarrow -2 * 1.46$

$\Rightarrow -2.92$ [Note: এখানে আমরা দেখতে পারছি শর্ত মেনেছে তাই আমরা ধরে নইতে পারি আমাদের রুট ০ আর ১ এর মাঝে আছে]

Let's Find the root:

Befor jump we need to know 1 thing:
$If\ f(a) * f(c) = positive\ value\ then\ a = c;$

$If\ f(a) * f(c) = negative\ value\ then\ b = c;$

Now Lets go:

Iteration	a	b	$f(a)$	$f(b)$	$c = \frac{a+b}{2}$	$f(c)$
1	0	1	-2	1.4597	0.5	-0.377583
2	0.5	1	-0.377583	1.4597	0.75	0.518311
3	0.5	0.75	-0.377583	0.518311	0.625	0.0640369
4	0.5	0.625	-0.377583	0.0640369	0.5625	-0.158424
5	0.5625	0.625	-0.158424	0.0640369	0.59375	-0.0475985
6	0.59375	0.625	-0.0475985	0.0640369	0.609375	0.0081191
7	0.59375	0.609375	-0.0475985	0.0081191	0.601562	-0.0197649
8	0.601562	0.609375	-0.0197649	0.0081191	0.605469	-0.00582915
9	0.605469	0.609375	-0.00582915	0.0081191	0.607422	0.00114341
10	0.605469	0.607422	-0.00582915	0.00114341	0.606445	-0.00234326
11	0.606445	0.607422	-0.00234326	0.00114341	0.606445	-0.00234326

Solve with iteration :

```
#include <bits/stdc++.h>

using namespace std;

double equation(double x) {

    // Define your equation here

    // For example, let's solve 3*x - cos(x)  - 1

    return 3*x - cos(x)  - 1;

}


double bisectionMethod(double a, double b, double tolerance) {

    double c;

int n=1;

    while (fabs(b - a    ) >= tolerance) {




        c = (a + b) / 2;

cout<<"Iteration: "<<n<< " a = " << a << " b = " << b << " f(a) "<<equation(a)<< " f(b)
"<<equation(b)<<" c = "<<c<<" f(c) "<<equation(c)<<endl;




        if (equation(c) == 0.0)

            return c;


        if (equation(c) * equation(a) < 0)

            b = c;
```

```
        else
            a = c;
n++;
    }
    cout<<"Iteration: "<<n<< " a = " << a << " b = " << b << " f(a) "<<equation(a)<< " f(b) 
"<<equation(b)<<" c = "<<c<<" f(c) "<<equation(c)<<endl;



    return c;
}

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

    cout << "Enter the interval [a, b]: ";
    cin >> a >> b;

    cout << "Enter the tolerance: ";
    cin >> tolerance;

    double root = bisectionMethod(a, b, tolerance);

    cout << "Approximate root: " << root << endl;

    return 0;
}
```

Additional info:

The condition `while (fabs(b - a) >= tolerance)` in the code ensures that the bisection method keeps running until the interval between the two values (let's call them `a` and `b`) becomes smaller than the desired level of accuracy, which is defined as `tolerance`.

Imagine you're trying to find where a number is on a line, but you can only see a range on that line (from `a` to `b`). To determine the number more precisely, you need to keep reducing the range until it's very small. The `while` condition does just that – it keeps the method running until the range (the difference between `a` and `b`) is tinier than what you consider acceptable (tolerance). This helps to pinpoint the location of the number you're seeking.

So, the smaller the `tolerance`, the more precise the final result will be, because it forces the method to keep refining the range until it's very, very small, giving a more accurate approximation of the number you're looking for.

Certainly! Let's consider the quadratic function:

$f(x) = (x - a)(x - b)$

where a and b are the roots of the function. You can choose any values for a and b. For example:

$f(x) = (x - 2)(x + 3)$

In this case, the roots are $x = 2$ and $x = -3$. You can replace 2 and -3 with any other values to create a quadratic function with different roots.

MD

multiply (x−2)(x+3)

Sure, let's multiply $(x - 2)(x + 3)$:

$$(x - 2)(x + 3) = x \cdot x + x \cdot 3 - 2 \cdot x - 2 \cdot 3$$

Simplifying the expression:

$$= x^2 + 3x - 2x - 6$$

Combining like terms:

$$= x^2 + x - 6$$

So, the product of $(x - 2)(x + 3)$ is $x^2 + x - 6$.

Regenerate

2. To find the roots of non-linear equation using False-Position method.

Algorithm:

1. **Define the function:**
   - $f(x)$ is the function for which the root is to be found.
2. **Initialize variables:**
   - a and b are the endpoints of the interval.
   - TOL is the tolerance for the solution.
   - c is the current approximation to the root.
3. **Iterate until convergence:**
   - Repeat until $|f(c)| < TOL$:
     - Calculate the function values at the endpoints: $fa = f(a)$ and $fb = f(b)$.
     - Calculate the approximate root using the false position formula: $c = \frac{a \cdot fb - b \cdot fa}{fb - fa}$
     - If $f(c) = 0.0$, return c as the exact root.
     - Update the interval based on the signs of function values:
       - If $f(c) \cdot fa < 0$, set $b = c$.
       - Else, set $a = c$.
4. **Output the result:**
   - Return c as the final approximation to the root of $f(x)$ within the specified tolerance.

Iteration	a	b	$f(a)$	$f(b)$	$c = \frac{a*f(b)-b*f(a)}{f(b)-f(a)}$	$f(c)$
1	0	1	-2	1.4597	0.578085	-0.103255
2	0.578085	1	-0.103255	1.4597	0.605959	-0.0040808

3	0.605959	1	-0.0040808	1.4597	0.607057	-0.000159047
4	0.607057	1	-0.000159047	1.4597	0.607057	-0.000159047

Solution:

```
#include <bits/stdc++.h>

using namespace std;


double equation(double x) {

   // Define your equation here

   // For example, let's solve 3x-cos(x)-1

   return pow(x,2)+x-6;

}


double falsePositionMethod(double a, double b, double tolerance) {

   double c;


  while (fabs(equation(c)) >= tolerance){

      // Calculate the function values at the endpoints

      double fa = equation(a);

      double fb = equation(b);


      // Calculate the approximate root using the false position formula

      c = (a * fb - b * fa) / (fb - fa);

cout<<"Iteration: "<<1<< " a = " << a << " b = " << b << " f(a) "<<equation(a)<< " f(b) "<<equation(b)<<" c = "<<c<<" f(c) "<<equation(c)<<endl;

      // Check if c is the root
```

```
        if (equation(c) == 0.0){
            return c;
        }
        // Update the interval based on the signs of function values
        if (equation(c) * fa < 0)
            b = c;
        else
            a = c;

    }

    return c;
}

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

    cout << "Enter the interval [a, b]: ";
    cin >> a >> b;

    cout << "Enter the tolerance: ";
    cin >> tolerance;

    double root = falsePositionMethod(a, b, tolerance);

    cout << "Approximate root: " << root << endl;
```

```
    return 0;
}
```

3. To find the roots of non-linear equation using Newton's method.

$f(x) = 3x - cosx - 1$

Newton Raphson Method:

Tangent formula: $y - y' = \frac{dy}{dx}(x - x_1)$

Now: $y - f(x_0) = f'(x_0)(x_1 - x_0)$

$\Rightarrow 0 - f(x_0) = f'(x_0)(x_1 - x_0)$

$\Rightarrow x_1 - x_0 = -\frac{f(x_0)}{f'(x_0)}$

$\Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

So,That , $x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$

Given function:

$f(x) = 3x - cosx - 1$

$f'(x) = 3 + sinx$

Iteration	x_n	$f(x_n)$	$f'(x_n)$	$x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$
1	0	0.214113	3	0.666667
2	0.666667	0.00139686	3.61837	0.607493
3	0.607493	6.28295e-08	3.57081	0.607102

So, The Approximater root is: 0.607102

<u>Solution:</u>

```
#include<bits/stdc++.h>


using namespace std;


// Define your function here 3x-cos(x)-1

double equation(double x) {

    return 3*x-cos(x)-1;

}


// Define the derivative of your function here

double derivative(double x) {

    return 3+sin(x);

}
```

```
double newtonRaphson(double x0, double epsilon, int maxIterations) {
    double x = x0;
    int iterations = 0;

    while (fabs(equation(x)) > epsilon ) {
        cout<<"x = "<<x;
        x = x - (equation(x) / derivative(x));
        cout<< " f(x) = "<<equation(x)<<"  f'(x) = "<<derivative(x)<<"  Xn = "<<x<<endl;
        iterations++;
    }

    return x;
}

int main() {
    double initialGuess = 0;
    double epsilon = 0.001;
    int maxIterations = 100;

    double root = newtonRaphson(initialGuess, epsilon, maxIterations);

    cout << "Approximate root: " << root << endl;

    return 0;
}
```

version-2:

```
#include <bits/stdc++.h>

using namespace std;

double equation(double x) {
    return 3 * x - cos(x) - 1; // Change this function as needed
}

double numericalDerivative(double x, double h) {
    return (equation(x + h) - equation(x - h)) / (2 * h);
}

double newtonRaphson(double x0, double epsilon, int maxIterations, double h) {
    double x = x0;
    int iterations = 0;

    while (fabs(equation(x)) > epsilon && iterations < maxIterations) {
        cout << "x = " << x;
        double derivative = numericalDerivative(x, h);
        x = x - (equation(x) / derivative);
```

```
        cout << " f(x) = " << equation(x) << "  f'(x) = " << derivative << "  Xn = " << x <<
endl;

        iterations++;

    }

    return x;

}


int main() {

    double initialGuess = 0;

    double epsilon = 0.001;

    int maxIterations = 100;

    double h = 0.0001; // Step size for numerical derivative


    double root = newtonRaphson(initialGuess, epsilon, maxIterations, h);


    cout << "Approximate root: " << root << endl;


    return 0;

}
```

5. To solve problems using Newton's forward difference method of interpolation.

Year	1931	1941	1951	1961	1971	1981
Sale	12	15	20	27	39	52

Newton’s forward difference formula:

$$f(a + uh) = f(x) + u\Delta f(x) + \frac{u(u-1)}{2!}\Delta^2 f(x) + \frac{u(u-1)(u-2)}{3!}\Delta^3 f(x) + \ldots\ldots$$

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$	$\Delta^4 f(x)$	$\Delta^5 f(x)$
1931	12					
		3				
1941	15		2			
				0		
		5				
					3	
			2			
1951	20					
				3		-10

		7				
			5		-7	
1961	27					
		12		-4		
1971	39		1			
		13				
1981	52					

Here, a=1931

h=10;

Now , a+uh=1934

$\Rightarrow 1931 + u * 10 = 1934$

$\Rightarrow 10u = 1934 - 1931$

$\Rightarrow u = 0.3$

SO,

$$f(1934) = 12 + 0.3(3) + \frac{0.3(0.3-1)}{2}(2) + 0 + \frac{0.3(0.3-1)(0.3-2)(0.3-3)}{24}(3)$$

$$+ \frac{0.3(0.3-1)(0.3-2)(0.3-3)(0.3-4)}{120}(-10)$$

$$= 12.27231$$

Solution:

```
#include <iostream>

#include <vector>
```

```
using namespace std;


double forwardInterpolation(vector<double>& x, vector<double>& y, double targetX)
{

    int n = x.size();

    double result = 0;


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

        double term = y[i]; // Initialize term with the value from y


        double numerator = 1.0;

        double denominator = 1.0;


        for (int j = 0; j < n; ++j) {

            if (j != i) {

                numerator *= (targetX - x[j]);

                denominator *= (x[i] - x[j]);

            }

        }


        term *= numerator / denominator; // Calculate the term for this point

        result += term; // Add the calculated term to the final result

    }


    return result;
```

```
}

int main() {
    vector<double> x = {1931, 1941, 1951, 1961, 1971, 1981};   // x values
    vector<double> y = {12, 15, 20, 27, 39, 52}; // corresponding f(x) values

    double targetX = 1934;
    double estimatedValue = forwardInterpolation(x, y, targetX);

    cout << "Estimated value at x = " << targetX << " is: " << estimatedValue << endl;

    return 0;
}
```

6.To solve problems using Lagrange method of interpolation.
Question:

Ans:

The following table gives the normal weight of a baby during the six month of life.

	x_0	x_1	x_2	x_3	x_4
Age in month	**0**	**2**	**3**	**5**	**6**
Weight in month	**5**	**7**	**8**	**10**	**12**
	y_0	y_1	y_2	y_3	y_4

Estimate the weight of the baby at the age of 4 month. x = 4.

Solution:

$$y = \frac{(x-x_1)(x-x_2).....(x-x_n)}{(x_0-x_1)(x_0-x_2).....(x_0-x_n)}\, y_0$$

$$+ \frac{(x-x_0)(x-x_2).....(x-x_n)}{(x_1-x_0)(x_1-x_2).....(x_0-x_n)}\, y_1$$

$$+ \frac{(x-x_0)(x-x_1)(x-x_3).....(x-x_n)}{(x_2-x_0)(x_2-x_1)(x_2-x_3).....(x_2-x_n)}\ y_2$$

$$+...$$

$$+ \frac{(x-x_0)(x-x_1)(x-x_2).....(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)(x_n-x_2).....(x_n-x_{n-1})}\ y_n$$

$$\Rightarrow y = \frac{(4-2)(4-3)(4-5)(4-6)}{(0-2)(0-3)(0-5)(0-6)} \times (5)$$

$$+ \frac{(4-0)(4-3)(4-5)(4-6)}{(2-0)(2-3)(2-5)(2-6)} \times (7)$$

$$+ \frac{(4-0)(4-2)(4-5)(4-6)}{(3-0)(3-2)(3-5)(2-6)} \times (8)$$

$$+ \frac{(4-0)(4-2)(4-3)(4-6)}{(5-0)(5-2)(5-3)(5-6)} \times (10)$$

$$+ \frac{(4-0)(4-2)(4-3)(4-5)}{(6-0)(6-2)(6-3)(6-5)} \times (12)$$

$$\Rightarrow\ y = \frac{(2)(1)(-1)(-2)}{(-2)(-3)(-5)(-6)} \times (5)$$

$$+ \frac{(4)(1)(-1)(-2)}{(2)(-1)(-3)(-4)} \times (7)$$

$$+\frac{(4)(2)(-1)(-2)}{(3)(1)(-2)(-3)} \times (8)$$

$$+\frac{(4)(2)(1)(-2)}{(5)(3)(2)(-1)} \times (10)$$

$$+\frac{(4)(2)(1)(-1)}{(6)(4)(3)(1)} \times (10)$$

$$\Rightarrow \left(\frac{20}{180}\right) + \left(\frac{56}{-24}\right) + \left(\frac{128}{18}\right) + \left(\frac{-160}{-30}\right) + \left(\frac{-96}{72}\right)$$

$$\Rightarrow y = 0.111 - 2.333 + 7.111 + 5.333 - 1.333$$

$\Rightarrow y = 8.889$ Ans.

<u>Solution:</u>

```
#include <iostream>

#include <vector>


using namespace std;


double lagrangeInterpolation(vector<double> x, vector<double> y, double targetX) {

    double result = 0.0;


    for (int i = 0; i < x.size(); i++) {

        double term = y[i];

        for (int j = 0; j < x.size(); j++) {

            if (j != i) {

                term = term * (targetX - x[j]) / (x[i] - x[j]);
```

```
            }
        }
        result += term;
    }

    return result;
}

int main() {
    // Given data
    vector<double> x = {0, 2, 3, 5, 6};
    vector<double> y = {5, 7, 8, 10, 12};

    // Test with a specific value (replace with desired input)
    double targetX = 4;

    // Perform Lagrange's interpolation
    double result = lagrangeInterpolation(x, y, targetX);

    // Output the result
    cout << "Interpolated value at x=" << targetX << ": " << result << endl;

    return 0;
}
```

11. To integrate numerically using trapezoidal rule.

Question:

Evaluate the integral $I = \int_0^1 \frac{dx}{\sqrt{1+x^2}}$ by trapezoidal rule dividing the integral [0,1] into 5 equal parts. Compute upto 5 decimals.

Formula: $I = \frac{h}{2}[y_1 + (2y_2 + 2y_3 + 2y_4 + \dots\dots\dots 2y_n) + y_{n+1}]$

Now, $n = 5, \quad h = \frac{UpperLimit - lowerLimit}{n} = \frac{1-0}{5} = 0.2$

x	0	0.2	0.4	0.6	0.8	1.0
$y = \frac{1}{\sqrt{1+x^2}}$	1.0	0.98058	0.92848	0.85749	0.78087	0.70711

$I = \frac{0.2}{2}[1.0 + 2(0.98058 + 0.92848 + 0.85749 + 0.78087) + 0.70711]$

$= 0.88020 \qquad Ans.$

Solution:

```
#include <iostream>
#include <cmath>

using namespace std;

double func(double x) {
    return 1.0 / sqrt(1.0 + x * x);
}
```

```
double trapezoidalRule(double a, double b, int n) {
    double h = (b - a) / n;

    double result = (func(a) + func(b)) / 2.0;
    for (int i = 1; i < n; i++) {
        double x_i = a + i * h;
        double y_i = func(x_i);
        result += func(x_i);
    }

    return h * result;
}

int main() {
    double a = 0.0;
    double b = 1.0;
    int n = 5;

    double integral = trapezoidalRule(a, b, n);

    cout << "The approximate integral value: " << integral << endl;

    return 0;
}
```

Or,

```
#include <iostream>
#include <cmath>

using namespace std;

double func(double x) {
    return 1.0 / sqrt(1.0 + x * x);
}

double trapezoidalRule(double a, double b, int n) {
    double h = (b - a) / n;

    double result = (func(a) + func(b)) / 2.0;
    cout<<"beginning result: "<<result<<endl;
    for (int i = 1; i < n; i++) {
        double x_i = a + i * h;
        double y_i = func(x_i);
        cout<<"iteration ---- x_i: "<<x_i<<" __ and __ "<< "y_i: "<<y_i<<endl;
        result += func(x_i);
        cout<<"iteration result: "<<result<<endl;
    }

    return h * result;
```

```
}

int main() {
    double a = 0.0;
    double b = 1.0;
    int n = 5;

    double integral = trapezoidalRule(a, b, n);

    cout << "The approximate integral value: " << integral << endl;

    return 0;
}
```

4. To find the roots of a non-linear equation using the Secant method. (anisul islam)

$f(x) = 3x - cosx - 1$

Solution:

Formula for second method:

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \times f(x_i)$$

$x_0 = 0 \qquad f(x_0) = f(0) = 3 * 0 - cos\,0 - 1 = -2$

$x_1 = 1 \qquad f(x_1) = f(1) = 3 * 1 - cos\,1 - 1 = 1.46$

$\therefore f(x_0) * f(x_1) < 0$

Iteration-1:

$x_{1+1} = x_1 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} \times f(x_1)$ [using the formula]

$\therefore x_2 = 1 - \frac{1-0}{1.46 - (-2)} \times 1.46$

$= 0.58$

Iteration-2:

$x_{2+1} = x_2 - \frac{x_2 - x_1}{f(x_2) - f(x_1)} \times f(x_2)$ [using the formula]

$\therefore x_3 = 0.58 - \frac{0.58-1}{-0.10 - 1.46} \times (-0.10)$

$= 0.64$

Iteration-3:

$x_{3+1} = x_3 - \frac{x_3 - x_2}{f(x_3) - f(x_2)} \times f(x_3)$ [using the formula]

$\therefore x_4 = 0.64 - \frac{0.64-0.58}{0.12 - (-0.10)} \times 0.12$

$= 0.57$

Iteration-4:

$x_{4+1} = x_4 - \frac{x_4 - x_3}{f(x_4) - f(x_3)} \times f(x_4)$ [using the formula]

$\therefore x_5 = 0.57 - \frac{0.57-0.64}{-0.13 - 0.12} \times (-0.13)$

$= 0.61$

Iteration-5:

$x_{5+1} = x_5 - \frac{x_5 - x_4}{f(x_5) - f(x_4)} \times f(x_5)$ [using the formula]

$\therefore x_6 = 0.61 - \frac{0.61-0.57}{0.01 - (-0.13)} \times 0.01$

$= 0.61$

So the root in 0.61

Code:

```
#include <iostream>
```

```
#include <cmath>    // For the cos() function and the fabs() function

using namespace std;

// Define the function f(x) = 3x - cos(x) - 1
double f(double x) {
    return 3 * x - cos(x) - 1;
}

// Implement the Secant method
double secant(double x0, double x1, double e) {
    double x2, f0, f1, f2;
    int iteration = 0;

    for (;;) {
        f0 = f(x0);
        f1 = f(x1);
        // Avoid division by zero
        if (f1 == f0) {
            cerr << "Mathematical Error.";
            return -1;
        }

        // Apply the Secant formula
        x2 = x1 - (f1 * (x1 - x0)) / (f1 - f0);

        x0 = x1;  // Update x0
        x1 = x2;  // Update x1
        f2 = f(x2);

        iteration++;

        // Display each iteration
        cout << "Iteration " << iteration << ": x = " << x2 << endl;

        if (fabs(f2) <= e) // Check if the result is accurate enough
            break;
    }

    return x2;  // Return the root
}

int main() {
    double x0 = 0, x1 = 1, e = 0.001;  // Initial guesses and tolerance

    double root = secant(x0, x1, e);

    if (root != -1) {
```

```
        cout << "The root is " << root << endl;
    }

    return 0;
}
```

12. To integrate numerically using the trapezoidal rule.

$\int_1^2 \frac{1}{x} dx \quad and \quad n = 10$

Solution:

Formula:

$i) \int_a^b f(x)dx = \frac{\Delta x}{2} = \left[f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + 2f(x_4) + ... + 2f(x_{n-1}) + f(x_n)\right]$

$ii) \Delta x = \frac{b-a}{n} \quad note: \left[\frac{upper\ limit - Lower\ limit}{n}\right]$

$ii) x_i = a + i\Delta x$

Here,$\Delta x = \frac{b-a}{n} = \frac{2-1}{10} = 0.1$

$x_i = a + i\Delta x$

So,

$x_0 = 1 + 0 \times 0.1 = 1$

$x_1 = 1 + 1 \times 0.1 = 1.1$

$x_2 = 1 + 2 \times 0.1 = 1.2$

$x_3 = 1 + 3 \times 0.1 = 1.3$

$x_4 = 1 + 4 \times 0.1 = 1.4$

$x_5 = 1 + 5 \times 0.1 = 1.5$

$x_6 = 1 + 6 \times 0.1 = 1.6$

$x_7 = 1 + 7 \times 0.1 = 1.7$

$x_8 = 1 + 8 \times 0.1 = 1.8$

$x_9 = 1 + 9 \times 0.1 = 1.9$

$x_{10} = 1 + 10 \times 0.1 = 2$

Now,

$\int_a^b f(x)dx = \frac{\Delta x}{2} = \left[f(x_0) + 2f(x_2) + 2f(x_3) + 2f(x_3) + 2f(x_4) + ... + 2f(x_{n-1}) + f(x_n)\right]$

$= \int_1^2 \frac{1}{x}dx = \frac{0.1}{2}[\frac{1}{1} + 2 \times \frac{1}{1.1} + 2 \times \frac{1}{1.2} + 2 \times \frac{1}{1.3} + 2 \times \frac{1}{1.4} + 2 \times \frac{1}{1.5} + 2 \times \frac{1}{1.6} + 2 \times \frac{1}{1.7}$

$+ 2 \times \frac{1}{1.8} + 2 \times \frac{1}{1.9} + \frac{1}{2}]$

= $\frac{0.1}{2}[1 + 12.374 + 0.5]$

= 0.6937

Code:

```
#include <bits/stdc++.h>

using namespace std;

// Define the function to integrate, f(x) = 1/x
double f(double x) {
    return 1/x;
}

// Implement the trapezoidal rule
double trapezoidalRule(double a, double b, int n) {
    double h = (b - a) / n;  // Step size
    double integral = f(a) + f(b);  // Sum the first and last terms

    for (int i = 1; i < n; i++) {  // Using post-increment here
        integral += 2 * f(a + i * h);  // Sum the interior terms with weight 2
    }

    integral *= h/2;  // Multiply by the step size divided by 2

    return integral;
}

int main() {
    double a = 1, b = 2;  // Limits of integration
    int n = 10;  // Number of subdivisions

    double result = trapezoidalRule(a, b, n);

    // Set precision for output to 5 decimal places
    cout << fixed << setprecision(5);
    cout << "The integral is approximately: " << result << endl;

    return 0;
}
```

13. To integrate numerically using Simpson's 1/3 rule.

$\int_1^2 \frac{1}{x} dx \quad and \quad n = 10$

Solution:

Formula:

i) $\int_a^b f(x)dx = \frac{\Delta x}{3} = \left[f(x_0) + 4f(x_1) + 2f(x_2) + 2f(x_3) + 2f(x_4) + ... + 4f(x_{n-1}) + f(x_n)\right]$

ii) $\Delta x = \frac{b-a}{n} \quad note: \left[\frac{upper\ limit - Lower\ limit}{n}\right]$

ii) $x_i = a + i\Delta x$

Here,$\Delta x = \frac{b-a}{n} = \frac{2-1}{10} = 0.1$

$x_i = a + i\Delta x$

So,

$x_0 = 1 + 0 \times 0.1 = 1$

$x_1 = 1 + 1 \times 0.1 = 1.1$

$x_2 = 1 + 2 \times 0.1 = 1.2$

$x_3 = 1 + 3 \times 0.1 = 1.3$

$x_4 = 1 + 4 \times 0.1 = 1.4$

$x_5 = 1 + 5 \times 0.1 = 1.5$

$x_6 = 1 + 6 \times 0.1 = 1.6$

$x_7 = 1 + 7 \times 0.1 = 1.7$

$x_8 = 1 + 8 \times 0.1 = 1.8$

$x_9 = 1 + 9 \times 0.1 = 1.9$

$x_{10} = 1 + 10 \times 0.1 = 2$

Now,

$\int_a^b f(x)dx = \frac{\Delta x}{3} = \left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + ... + 4f(x_{n-1}) + f(x_n)\right]$

$= \int_1^2 \frac{1}{x} dx = \frac{0.1}{3} [\frac{1}{1} + 4 \times \frac{1}{1.1} + 2 \times \frac{1}{1.2} + 2 \times \frac{1}{1.3} + 2 \times \frac{1}{1.4} + 2 \times \frac{1}{1.5} + 2 \times \frac{1}{1.6} + 2 \times \frac{1}{1.7}$

$+ 2 \times \frac{1}{1.8} + 4 \times \frac{1}{1.9} + \frac{1}{2}]$

$= \frac{0.1}{3} \times 20.7947$

= 0.69315

Code:

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

// Define the function to be integrated
double f(double x) {
```

```
    if (x == 0) {
        return numeric_limits<double>::infinity(); // Avoid division by zero
    }
    return 1 / x;
}

// Implement Simpson's 1/3 rule
double simpsonsOneThirdRule(double a, double b, int n) {
    double h = (b - a) / n; // Calculate the interval size
    double sum = f(a) + f(b); // f(x_0) + f(x_n)

    // Apply Simpson's 1/3 rule
    for (int i = 1; i < n; i++) {
        double x_i = a + i * h;
        if (i % 2 == 0) {
            sum += 2 * f(x_i); // Even index terms are multiplied by 2
        } else {
            sum += 4 * f(x_i); // Odd index terms are multiplied by 4
        }
    }
    return (h / 3) * sum;
}

int main() {
    double lower_limit = 1;
    double upper_limit = 2;
    int n = 10; // Number of intervals

    // Calculate the integral
    double result = simpsonsOneThirdRule(lower_limit, upper_limit, n);

    // Output the result
    cout << " using Simpson's 1/3 rule is: "
         << setprecision(5) << fixed << result << endl;

    return 0;
}
```

4. To find the roots of a non-linear equation using the Secant method. (RKR)

$f(x) = 3x - cosx - 1$

Solution:

Formula for second method:

$$x_{i+1} = \frac{x_{i-1}f_i - x_i f_{i-1}}{f_i - f_{i-1}}$$

$$x_1 = \frac{x_{-1}f_0 - x_0 f_{-1}}{f_0 - f_{-1}}$$

$$x_2 = \frac{x_0 f_1 - x_1 f_0}{f_1 - f_0}$$

$$x_3 = \frac{x_1 f_2 - x_2 f_1}{f_2 - f_1}$$

$$x_4 = \frac{x_2 f_3 - x_3 f_2}{f_3 - f_2}$$

Code:

```
#include <bits/stdc++.h>

using namespace std;

// Define the function for which we are finding the root
double f(double x) {
    // Replace with the actual function
    return 3 * x - cos(x) - 1;
}

// Implementing the Secant Method
double secantMethod(double x0, double x1, double tol, int maxIter) {
    double x2, fx0, fx1;
    for (int i = 0; i < maxIter; i++) {
        fx0 = f(x0);
        fx1 = f(x1);
        x2 = x1 - (fx1 * (x1 - x0)) / (fx1 - fx0);

        // Check for convergence
        if (fabs(x2 - x1) < tol) {
            return x2;
        }

        // Update the values
        x0 = x1;
        x1 = x2;
    }
    return x2; // Return the root approximation
}
```

```
int main() {
    double x0 = 1; // Initial guess
    double x1 = 2; // Second guess
    double tol = 0.001; // Tolerance
    int maxIter = 100; // Maximum number of iterations

    double root = secantMethod(x0, x1, tol, maxIter);
    cout << "The root is: " << root << endl;

    return 0;
}
```

11. To find the numerical solution of Newton's divided difference interpolation formula.

Use newton's divided difference formula evaluate f(6) from following data:

x	5	7	11	13	21
f(x)	150	392	1452	2366	9702

Formula:

$$f(x) = f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + (x - x_0)(x - x_1)(x - x_2) f(x_0, x_1, x_2, x_3) + ...$$

Solve:

Newton's divided difference table is

x	y	1^{st} order	2^{nd} order	3^{rd} order	4^{th} order
5	150				
		$\frac{392-150}{7-5} = 121$			
7	392		$\frac{265-121}{11-5} = 24$		
		$\frac{1452-392}{11-7} = 265$		$\frac{32-24}{13-5} = 1$	
11	1452		$\frac{457-265}{13-7} = 32$		$\frac{1-1}{21-5} = 0$
		$\frac{2366-1452}{13-11} = 457$		$\frac{46-32}{21-7} = 1$	
13	2366		$\frac{917-457}{21-11} = 46$		
		$\frac{9702-2366}{21-13} = 917$			
21	9702				

Here,

$f(x_0) = 150$

$f(x_0, x_1) = 121$

$f(x_0, x_1, x_2) = 24$

$f(x_0, x_1, x_2, x_3) = 1$

And,

$x = 6$, $x_0 = 5$, $x_1 = 7$, $x_2 = 11$, $x_3 = 13$, $x_4 = 21$

From Newton's divided difference formula we get,

$$f(x) = f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + (x - x_0)(x - x_1)(x - x_2) f(x_0, x_1, x_2, x_3)$$

$$\therefore f(6) = 150 + (6 - 5) \times 121 + (6 - 5)(6 - 7) \times 24 + (6 - 5)(6 - 7)(6 - 11) \times 1$$
$$= 150 + 121 - 24 + 5$$
$$= 252$$

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

// Function to find the product term
float proterm(int i, float value, float x[])
{
	float pro = 1;
	for (int j = 0; j < i; j++) {
		pro = pro * (value - x[j]);
	}
	return pro;
}

// Function for calculating
// divided difference table
void dividedDiffTable(float x[], float y[][10], int n)
{
	for (int i = 1; i < n; i++) {
		for (int j = 0; j < n - i; j++) {
			y[j][i] = (y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]);
		}
	}
}

// Function for applying Newton's
// divided difference formula
float applyFormula(float value, float x[],
					float y[][10], int n)
{
	float sum = y[0][0];

	for (int i = 1; i < n; i++) {
	sum = sum + (proterm(i, value, x) * y[0][i]);
	}
	return sum;
}

// Function for displaying
// divided difference table
void printDiffTable(float y[][10],int n)
{
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n - i; j++) {
			cout << setprecision(4) << y[i][j] << "\t ";
		}
		cout << "\n";
	}
```

```
}

// Driver Function
int main()
{
	// number of inputs given
	int n = 5;
	float value, sum, y[10][10];
	float x[] = { 5, 7, 11, 13,21 };

	// y[][] is used for divided difference
	// table where y[][0] is used for input
	y[0][0] = 150;
	y[1][0] = 392;
	y[2][0] = 1452;
	y[3][0] = 2366;
   y[4][0] = 9702;

	// calculating divided difference table
	dividedDiffTable(x, y, n);

	// displaying divided difference table
	printDiffTable(y,n);

	// value to be interpolated
	value = 6;

	// printing the value
	cout << "\nValue at " << value << " is "
			<< applyFormula(value, x, y, n) << endl;
	return 0;
}
```

10. To find the numerical solution of Lagrange interpolation formula.
https://youtu.be/dcHPhLDWmZE?si=TGnVmhKsJVqt35Uo
Problem:
The following table gives the normal weight of a baby during the six month of life.

Age in month	0	2	3	5	6
Weight in lbs	5	7	8	10	12

Estimate the weight of the baby at the age of 4 month.

Solve:
The lagrange's interpolation formula is:

$$y = \frac{(x-x_1)(x-x_2)........(x-x_n)}{(x_0-x_1)(x_0-x_2)....(x_0-x_n)} y_0 + \frac{(x-x_0)(x-x_2)........(x-x_n)}{(x_1-x_0)(x_1-x_2)....(x_1-x_n)} y_1 + \frac{(x-x_0)(x-x_1)(x-x_3)........(x-x_n)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)....(x_2-x_n)} y_2$$

$$+ .... + \frac{(x-x_0)(x-x_1)(x-x_2)........(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)(x_n-x_2)....(x_n-x_{n-1})} y_n$$

Code:

```
#include <bits/stdc++.h> // Add this header for using namespace std

using namespace std; // Add this line to use the std namespace

int main() {
   // Given data points
   float age[] = {0, 2, 3, 5, 6}; // x values
   float weight[] = {5, 7, 8, 10, 12}; // y values
   float x = 4; // the point at which we want to estimate the value of y
   float y = 0; // this will hold the result
   int n = 5; // number of data points

   // Apply Lagrange interpolation formula
   for (int i = 0; i < n; i++) {
      // Calculate the Lagrange polynomial for the i-th data point
      float li = 1;
      for (int j = 0; j < n; j++) {
         if (j != i) {
            li *= (x - age[j]) / (age[i] - age[j]);
         }
      }
      // Add the current term to the result
      y += li * weight[i];
   }

   // Display the result
   cout << "The estimated weight of the baby at 4 months is: " << y << " lbs" << endl;

   return 0;
}
```

14. To find the numerical solution of ordinary differential equations by Euler's method.
https://youtu.be/zD-Mg4ZUGsE?si=G0JTVDPHnbInGn39

Problem:

Given $\frac{dy}{dx} = \frac{y-x}{y+x}$ with initial condition y=1 at x=0. Find y for x=0.1 by Euler's Method.

Solution:

Given y=1 and x =0.

Let,

$y_0 = 1 \ \ and\, x_0 = 0$

Let our step size = 5.

So, h=$\frac{0-0.1}{5} = 0.02$

So,

$x_0 = 0\ ,\ \ x_1 = 0.02,\, x_2 = 0.04,\ x_3 = 0.06,\ x_4 = 0.08,\ x_5 = 0.1$

Now we need to find the value of y

So,

The formula is: $y_n = y_{n-1} + hf(x_{n-1}, y_{n-1})$

Now,

$y_1 = y_{1-1} + hf(x_{1-1}, y_{1-1})$

$= y_0 + hf(x_0, y_0)$

$= y_0 + 0.02(\frac{y_0 - x_0}{y_0 + x_0})$

$= \ 1 + 0.02(\frac{1-0}{1+0})$

$= \ 1.02$

$y_2 = y_{2-1} + hf(x_{2-1}, y_{2-1})$

$= y_1 + hf(x_1, y_1)$

$= 1.02 + 0.02(\frac{y_1 - x_1}{y_1 + x_1})$

$= 1.02 + 0.02(\frac{y_1 - x_1}{y_1 + x_1})$

$= 1.02 + 0.02(\frac{1.02-0.02}{1.02+0.02})$

$= 1.0392$

Continue…….

$y_3 = 1.0578$

$y_4 = 1.0757$

$y_5 = 1.0929$

Code:

```
#include <iostream>
#include <bits/stdc++.h> // Add this header for using namespace std

using namespace std; // Add this line to use the std namespace
```

```
// Define the derivative of y with respect to x as given in the problem
float dydx(float x, float y) {
    return (y - x) / (y + x);
}

int main() {
    // Initial condition
    float x0 = 0.0;
    float y0 = 1.0;
    float h = 0.02; // Step size
    float x = 0.1; // The value at which we want to estimate y

    // Number of steps to reach x = 0.1 with step size h
    int n = (int)((x - x0) / h);

    // Euler's method
    for (int i = 0; i < n; i++) {
        y0 = y0 + h * dydx(x0, y0);
        x0 = x0 + h;
    }

    // Output the estimate for y at x = 0.1
    cout << "The estimated value of y at x = " << x << " is: " << y0 << endl;

    return 0;
}
```

15. Implement appropriate numerical methods to calculate a definite integral.

1. **Riemann Sum**
2. **Trapezoidal Rule**
3. **Simpson's Rule**
4. **Gaussian Quadrature**

7. To solve the system of linear equations using Gauss Elimination method.
https://www.youtube.com/watch?v=f3ZvVWUdgxc

Problem:

Solve the following equation with gauss elimination method.

$2x + y + 4z = 12$

$4x + 11y - z = 33$

$8x - 3y + 2z = 20$

Solution:

$2x + y + 4z = 12$ ....(1)

$4x + 11y - z = 33$ ....(2)

$8x - 3y + 2z = 20$ ....(3)

Step-1:

Consider eq.(1) and eq.(2)

$r_1 = \frac{coefficient\ of\ x\ in\ eq.(2)}{coefficient\ of\ x\ in\ eq.(1)}$

$= \frac{4}{2}$

$= 2$

Now,

$eq.(2) - r_1(eq.1)$

So, $4x + 11y - z - 2(2x + y + 4z) = 33 - 2(12)$

$= 4x + 11y - z - 4x - 2y - 8z = 32 - 24$

$= 11y - z - 2y - 8z = 9$

$= 9y - 9z = 9$

$= y - z = \frac{9}{9}$

$= y - z = 1$ ....(4)

Step-2:

Consider eq.(1) and eq.(2)

$r_2 = \frac{coefficient\ of\ x\ in\ eq.(3)}{coefficient\ of\ x\ in\ eq.(1)}$

$= \frac{8}{2}$

$= 4$

Now,

$eq.(3) - r_2(eq.1)$

So, $8x - 3y + 2z - 4(2x + y + 4z) = 20 - 4(12)$

$= 8x - 3y + 2z - 8x - 4y - 16z = 20 - 48$

$=- 3y + 2z - 4y - 16z =- 28$

$=- 7y - 14z =- 28$

$= 7y - 14z = 28$

$= 7(y - 2z) = 28$

$= y + 2z = 4$ ……….(5)

Step-3:

Consider eq.(4) and eq.(5)

$r_3 = \frac{coefficient\ of\ y\ \ in\ eq.(5)}{coefficient\ of\ y\ in\ eq.(4)}$

$= \frac{1}{1}$

$= 1$

Now,

$eq.(5) - r_2(eq.4)$

So,$y + 2z - 1(y - z) = 4 - 1(1)$

$= y + 2z - y + z = 4 - 1$

$= 3z = 3$

$= z = 1$

Put value of z in eq(5)

$y + 2z = 4$

$\Rightarrow y + 2(1) = 4$

$\Rightarrow y = 2$

Now put y and z in eq(1)

$2x + y + 4z = 12$

$\Rightarrow 2x + 2 + 4(1) = 12$

$\Rightarrow 2x + 6 = 12$

$\Rightarrow 2x = 12 - 6$

$\Rightarrow 2x = 6$

$\Rightarrow x = 3$

So, final answer is : x = 3, y = 2 and z = 1.

Code:

```
#include <iostream>
#include <vector>

int main() {
    // Coefficient matrix
    std::vector<std::vector<double>> A = {
        {2, 1, 4},
        {4, 11, -1},
        {8, -3, 2}
    };

    // Right-hand side vector
    std::vector<double> b = {12, 33, 20};
```

```
    // Number of equations
    const int n = A.size();

    // Forward elimination
    for (int i = 0; i < n; ++i) {
        // Make the first element of the row 'i' unity, and scale the rest of the row.
        double pivot = A[i][i];
        for (int j = 0; j < n; ++j) {
            A[i][j] /= pivot;
        }
        b[i] /= pivot;

        // Eliminate the first element in all rows below i.
        for (int k = i + 1; k < n; ++k) {
            double factor = A[k][i];
            for (int j = 0; j < n; ++j) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
    }

    // Back substitution
    std::vector<double> x(n);
    for (int i = n - 1; i >= 0; --i) {
        x[i] = b[i];
        for (int j = i + 1; j < n; ++j) {
            x[i] -= A[i][j] * x[j];
        }
    }

    // Output the results
    std::cout << "The solution is:" << std::endl;
    for (int i = 0; i < n; ++i) {
        std::cout << "x" << i + 1 << " = " << x[i] << std::endl;
    }

    return 0;
}
```

7. To solve the system of linear equations using Gauss-elimination method.

Solving process:

Working rule:

Consider the system of equation:

$a_{11}x + a_{12}y + a_{13}z = b_1$

$a_{21}x + a_{22}y + a_{23}z = b_2$

$a_{31}x + a_{32}y + a_{33}z = b_3$

In matrix form AX = B

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Augmented matrix, C = [A:B] = $\begin{bmatrix} a_{11} & a_{12} & a_{13} & : & b_1 \\ a_{21} & a_{22} & a_{23} & : & b_2 \\ a_{31} & a_{32} & a_{33} & : & b_3 \end{bmatrix}$

Reduce the augmented matrix to echelon form using elementary row transformations,

C = $\begin{bmatrix} c_{11} & c_{12} & c_{13} & : & d_1 \\ c_{21} & c_{22} & c_{23} & : & d_2 \\ c_{31} & c_{32} & c_{33} & : & d_3 \end{bmatrix}$

The corresponding system of equation are :

$c_{11}x + c_{12}y + c_{13}z = d_1$

$\qquad + c_{22}y + c_{23}z = d_2$

$\qquad\qquad + c_{33}z = d_3$

The solution of system is obtained by solving these equation by back substitution.

Code:

```
#include <iostream>
#include <vector>
#include <cmath>

const int N = 3; // Assuming it's a 3x3 system of equations

// Function to perform row reduction to echelon form
void gaussianElimination(std::vector<std::vector<double>>& mat) {
    for (int i = 0; i < N; i++) {
        // Partial pivoting
        for (int k = i + 1; k < N; k++) {
            if (abs(mat[k][i]) > abs(mat[i][i])) {
                std::swap(mat[k], mat[i]);
            }
```

```
        }

        // Making elements below the pivot equal to 0
        for (int k = i + 1; k < N; k++) {
            double t = mat[k][i] / mat[i][i];
            for (int j = i; j <= N; j++) {
                mat[k][j] -= t * mat[i][j]; // Subtract the multiplied row from current row
            }
        }
    }
}

// Function to perform back substitution
std::vector<double> backSubstitution(const std::vector<std::vector<double>>& mat) {
    std::vector<double> x(N); // Solution vector

    for (int i = N - 1; i >= 0; i--) {
        x[i] = mat[i][N];
        for (int j = i + 1; j < N; j++) {
            x[i] -= mat[i][j] * x[j];
        }
        x[i] = x[i] / mat[i][i];
    }
    return x;
}

int main() {
    // Matrix representation of augmented matrix
    // Replace with the actual matrix from your problem
    std::vector<std::vector<double>> mat = {
        {2, 1, -1, 8},
        {-3, -1, 2, -11},
        {-2, 1, 2, -3}
    };

    gaussianElimination(mat);

    std::vector<double> x = backSubstitution(mat);

    std::cout << "The solution is: \n";
    for (int i = 0; i < N; i++) {
        std::cout << "x" << i + 1 << " = " << x[i] << std::endl;
    }

    return 0;
}
```

8.

To solve the system of linear equations using Gauss-Jordan method.
https://youtu.be/6n7tub_20J0?si=QbCNAbb7OIXu26VY

Solving process:

Working rule:

Consider the system of equation:

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}z = b_3$$

In matrix form AX = B

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Augmented matrix, C = [A:B] = $\begin{bmatrix} a_{11} & a_{12} & a_{13} & : & b_1 \\ a_{21} & a_{22} & a_{23} & : & b_2 \\ a_{31} & a_{32} & a_{33} & : & b_3 \end{bmatrix}$

Applying elementary row transformations to augmented matrix to reduce coefficient matrix to unit matrix.

[A:B] $\xrightarrow[row\ transformation]{by\ elemtary} \begin{bmatrix} 1 & 0 & 0 & : & d_1 \\ 0 & 1 & 0 & : & d_2 \\ 0 & 0 & 1 & : & d_3 \end{bmatrix}$

Corresponding system of equations:

$$x = d_1$$
$$y = d_2$$
$$z = d_3$$

Code:

```
#include <iostream>
#include <vector>
#include <cmath>

// Function to perform Gauss-Jordan elimination
void gaussJordan(std::vector<std::vector<double>>& mat) {
    int n = mat.size();
    for (int i = 0; i < n; ++i) {
        // Make the diagonal element 1.
        double diag = mat[i][i];
```

```
        for (int j = 0; j <= n; ++j) {
            mat[i][j] /= diag;
        }

        // Make the rest of the column 0.
        for (int k = 0; k < n; ++k) {
            if (k != i) {
                double factor = mat[k][i];
                for (int j = 0; j <= n; ++j) {
                    mat[k][j] -= factor * mat[i][j];
                }
            }
        }
    }
}

// Function to print the solution
void printSolution(const std::vector<std::vector<double>>& mat) {
    for (int i = 0; i < mat.size(); ++i) {
        std::cout << "x" << i + 1 << " = " << mat[i][mat.size()] << "\n";
    }
}

int main() {
    // Example of a 3x3 system's augmented matrix
    // Replace with your actual augmented matrix
    std::vector<std::vector<double>> mat = {
        {2, 1, -1, 8},
        {-3, -1, 2, -11},
        {-2, 1, 2, -3}
    };

    gaussJordan(mat);
    printSolution(mat);

    return 0;
}
```

5. To implement the Least Square Method for curve fitting.

Problem:Find the curve of best fit $y = ae^{bx}$ to the following data by using method of Least Square Method.

x	1	5	7	9	12
y	10	15	12	15	21

$y = ae^{bx}$

Taking log on both side

$log\, y = log\, a + log\, e^{bx}$

$\Rightarrow log\, y = log\, a + bx.\, log\, e$

$\Rightarrow Y = A + Bx$

$\Sigma Y = nA + B\Sigma x$ .....(i)

$\Sigma xY = A\Sigma x + B\Sigma x^2$ .....(ii)

x	**y**	**Y = log y**	x^2	**xY**
1	10	1	1	1
5	15	1.18	25	5.9
7	12	1.08	49	7.56
9	15	1.18	81	10.62
12	21	1.32	144	15.84
Σx		Σy	Σx^2	ΣxY
34		5.76	300	40.92

5.76=5A+b.34 A=0.98 ⇒ log a = A
⇒ a = anti log (A)
⇒ a = 9.55

40.92=34A+300.B B=0.025 ⇒ log e b = B
⇒ b= anti log(B)
⇒ b = 1.06

$y = ae^{bx}$

$=9.5e^{1.06x}$ (Ans)

Code:

```
#include <iostream>
#include <cmath>
#include <vector>

int main() {
    // Given data points
    std::vector<double> x = {1, 5, 7, 9, 12};
    std::vector<double> y = {10, 15, 12, 15, 21};
```

```
    // Number of data points
    int n = x.size();

    // Variables to store the sums needed for least squares
    double sumX = 0, sumY = 0, sumX2 = 0, sumXY = 0;

    // Calculate the sums
    for (int i = 0; i < n; ++i) {
        sumX += x[i];
        sumY += log(y[i]); // Note that we're using log(y), not y
        sumX2 += x[i] * x[i];
        sumXY += x[i] * log(y[i]);
    }

    // Calculate the coefficients A and B from the normal equations
    double B = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    double A = (sumY - B * sumX) / n;

    // Calculate a and b for the original exponential equation
    double a = exp(A);
    double b = B;

    // Output the results
    std::cout << "The best-fit curve is y = " << a << "e^(" << b << "x)" << std::endl;

    return 0;
}
```

6. To implement the Polynomial Method for curve fitting.