

Numerical Analysis Code

Omm Satyakam Behera
Roll No: 424MA5005
NIT Rourkela

Course Details

- **Course Code:** MA2071
- **Instructor:** Prof. Bikash Sahoo

Numerical Methods

Bisection Method

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 #define EP 0.0001 // Smaller epsilon for better precision
6
7 // A function with a real root
8 double solution(double x) {
9     return x*x*x + x - 1; // Example:  $f(x) = x^3 + x - 1$ 
10 }
11
12 void bisection(double a, double b) {
13     if (solution(a) * solution(b) >= 0) {
14         cout << "You have not assumed correct a and b\n";
15         return;
16     }
17
18     double c;
19     int iterations = 0;
20
21     while ((b - a) >= EP) {
22         c = (a + b) / 2;
23         double fc = solution(c);
24
25         if (fc == 0.0)
26             break;
27
28         if (solution(a) * fc < 0)
29             b = c;
30         else
31             a = c;
32
33         iterations++;
34     }
35
36     cout << "The value of root is : " << c << endl;
37     cout << "Found in " << iterations << " iterations." << endl;
38 }
39
40 int main() {
41     double a = 0, b = 1;
42     bisection(a, b);
43     return 0;
44 }
45
46 /*
```

```

47 This program does not take any user input. The initial values for the bisection method
    are hardcoded in the main function.
48 a = 0
49 b = 1
50 */

```

```

~/github/ma2071_numerical_analysis_lab main 8ms
15:30:06 ./class_1/bisection-out
The value of root is : 0.682312
Found in 14 iterations.

```

Secant Method

```

1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4 using namespace std;
5
6 // Define the function
7 float f(float x) {
8     return pow(x, 3) + x - 1; // Equation: x^3 + x - 1
9 }
10
11 void secant(float x1, float x2, float E) {
12     float x0, f1, f2;
13     int n = 0;
14
15     do {
16         f1 = f(x1);
17         f2 = f(x2);
18
19         if (f2 - f1 == 0) {
20             cout << "Division by zero error in secant formula." << endl;
21             return;
22         }
23
24         x0 = (x1 * f2 - x2 * f1) / (f2 - f1);
25
26         // Update values
27         x1 = x2;
28         x2 = x0;
29         n++;
30
31     } while (fabs(f(x2) - f(x1)) >= E);
32
33     cout << fixed << setprecision(10);
34     cout << "Root of the equation = " << x0 << endl;
35     cout << "Number of iterations = " << n << endl;
36 }
37
38 int main() {
39     float x1 = 0, x2 = 1, E = 1e-7;
40     secant(x1, x2, E);
41     return 0;
42 }
43
44 /*
45 This program does not take any user input. The initial values for the secant method are
    hardcoded in the main function.
46 x1 = 0
47 x2 = 1
48 E = 1e-7
49 */

```

```

~/github/ma2071_numerical_analysis_lab main 5ms
15:31:56 ./class_1/secant-out
Root of the equation = 0.6823278069
Number of iterations = 7

```

Newton-Raphson Method

```
1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4 using namespace std;
5
6 // Define functions
7 double f1(double x, double y) { return pow(x, 3) + y - 1; }
8 double f2(double x, double y) { return pow(y, 3) - x + 1; }
9
10 // Partial derivatives (Jacobian)
11 double f1x(double x, double y) { return 3 * x * x; } // \textdegree f1/\textdegree x
12 double f1y(double x, double y) { return 1; } // \textdegree f1/\textdegree y
13 double f2x(double x, double y) { return -1; } // \textdegree f2/\textdegree x
14 double f2y(double x, double y) { return 3 * y * y; } // \textdegree f2/\textdegree y
15
16 int main() {
17     cout << "Newton's Method for solving system of equations\n";
18
19     double x, y;
20     cout << "Enter initial guesses for x and y: ";
21     cin >> x >> y;
22
23     int maxIter = 20;
24     double tol = 1e-6;
25
26     for (int iter = 1; iter <= maxIter; iter++) {
27         // Function values
28         double F1 = f1(x, y);
29         double F2 = f2(x, y);
30
31         // Jacobian matrix
32         double J11 = f1x(x, y), J12 = f1y(x, y);
33         double J21 = f2x(x, y), J22 = f2y(x, y);
34
35         // Determinant
36         double det = J11 * J22 - J12 * J21;
37         if (fabs(det) < 1e-12) {
38             cout << "Jacobian is singular. Stopping.\n";
39             break;
40         }
41
42         // Inverse of 2x2 Jacobian * F
43         double dx = (-F1 * J22 + F2 * J12) / det;
44         double dy = (-J11 * F2 + J21 * F1) / det;
45
46         // Update guesses
47         x += dx;
48         y += dy;
49
50         cout << "Iteration " << iter << ": x = "
51              << fixed << setprecision(6) << x
52              << ", y = " << y
53              << " | dx = " << dx << ", dy = " << dy << endl;
54
55         // Check convergence
56         if (fabs(dx) < tol && fabs(dy) < tol) {
57             cout << "Converged to solution.\n";
58             break;
59         }
60     }
61
62     cout << "Final solution: x = " << x << ", y = " << y << endl;
63     return 0;
64 }
65
66 /*
67 This program expects the user to input initial guesses for x and y.
68 For example:
69 -0.5
70 0.5
71 */
```

```

~/.github/ma2071_numerical_analysis_lab main 7ms
15:32:26 ./class_2/newton_raphson_mthd
Newton's Method for solving system of equations
Enter initial guesses for x and y: -1
1
Iteration 1: x = -0.400000, y = 0.200000 | dx = 0.600000, dy = -0.800000
Iteration 2: x = 1.029349, y = 0.377912 | dx = 1.429349, dy = 0.177912
Iteration 3: x = 0.954776, y = 0.146389 | dx = -0.074574, dy = -0.231524
Iteration 4: x = 0.994989, y = 0.019654 | dx = 0.040214, dy = -0.126735
Iteration 5: x = 0.999985, y = 0.000120 | dx = 0.004996, dy = -0.019534
Iteration 6: x = 1.000000, y = 0.000000 | dx = 0.000015, dy = -0.000120
Iteration 7: x = 1.000000, y = 0.000000 | dx = 0.000000, dy = -0.000000
Converged to solution.
Final solution: x = 1.000000, y = 0.000000

```

Fixed Point Method

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 double iterat(double x1, double x2, double x3, int i) {
6     i++;
7
8     // g1(x) = (x^2 - 3)/2
9     double y1 = (x1 * x1 - 3) / 2.0;
10
11     // g2(x) = 3 / (x - 2)
12     double y2 = 3.0 / (x2 - 2.0);
13
14     // g3(x) = sqrt(2x + 3)
15     double y3 = sqrt(2.0 * x3 + 3.0);
16
17     double d = y3 - x3;
18
19     cout << "After iteration no. " << i << ":\t"
20          << y1 << "\t\t" << y2 << "\t\t" << y3 << endl;
21
22     if (fabs(d) > 1e-5)
23         return iterat(y1, y2, y3, i); // continue iteration
24     else
25         return y3; // converged
26 }
27
28 int main() {
29     int i = 0;
30     double x = 2.5, y;
31
32     cout << "After iteration no. " << i
33          << ":\tg1(x)\t\tg2(x)\t\tg3(x)\n";
34
35     y = iterat(x, x, x, i);
36
37     cout << "The answer is x = " << y << endl;
38     return 0;
39 }
40
41 /*
42 This program does not take any user input. The initial value for the fixed-point
43 iteration is hardcoded in the main function.
44 x = 2.5
45 */

```

```

~/github/ma2071_numerical_analysis_lab main 5ms
15:33:46 ./class_3/fix_point_mthd
After iteration no. 0: g1(x)          g2(x)          g3(x)
After iteration no. 1: 1.625          6             2.82843
After iteration no. 2: -0.179688      0.75          2.94225
After iteration no. 3: -1.48386       -2.4          2.98069
After iteration no. 4: -0.399085      -0.681818     2.99356
After iteration no. 5: -1.42037       -1.11864     2.99785
After iteration no. 6: -0.491281      -0.961957     2.99928
After iteration no. 7: -1.37932       -1.01284     2.99976
After iteration no. 8: -0.548736      -0.995737     2.99992
After iteration no. 9: -1.34944       -1.00142     2.99997
After iteration no. 10: -0.5895       -0.999526     2.99999
After iteration no. 11: -1.32624      -1.00016     3
The answer is x = 3

```

Gauss Elimination Method

```

1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4 using namespace std;
5
6 int main() {
7     int n;
8     cout << "GAUSS ELIMINATION METHOD\n";
9     cout << "Enter the number of equations: ";
10    cin >> n;
11
12    double coeff[20][20], var[20], temp, pivratio;
13
14    // Input augmented matrix
15    cout << "\nEnter the augmented matrix (coefficients + constants):\n";
16    for (int i = 1; i <= n; i++) {
17        cout << "Equation " << i << ": ";
18        for (int j = 1; j <= n + 1; j++) {
19            cin >> coeff[i][j];
20        }
21    }
22
23    // Display augmented matrix
24    cout << "\nThe augmented matrix is:\n";
25    for (int i = 1; i <= n; i++) {
26        for (int j = 1; j <= n + 1; j++) {
27            cout << setw(10) << coeff[i][j] << " ";
28        }
29        cout << "\n";
30    }
31
32    // Forward elimination
33    for (int i = 1; i <= n - 1; i++) {
34        for (int j = i + 1; j <= n; j++) {
35            if (coeff[i][i] == 0) {
36                cout << "Mathematical Error: Zero pivot element.\n";
37                return 1;
38            }
39            pivratio = coeff[j][i] / coeff[i][i];
40            for (int k = i; k <= n + 1; k++) {
41                coeff[j][k] -= pivratio * coeff[i][k];
42            }
43        }
44    }
45
46    // Display upper triangular matrix
47    cout << "\nUpper Triangular Matrix after elimination:\n";
48    for (int i = 1; i <= n; i++) {
49        for (int j = 1; j <= n + 1; j++) {
50            cout << setw(10) << coeff[i][j] << " ";
51        }
52        cout << "\n";
53    }

```

```

54
55 // Back substitution
56 for (int i = n; i >= 1; i--) {
57     temp = coeff[i][n + 1];
58     for (int j = i + 1; j <= n; j++) {
59         temp -= coeff[i][j] * var[j];
60     }
61     var[i] = temp / coeff[i][i];
62 }
63
64 // Output solution
65 cout << "\nSolution:\n";
66 for (int i = 1; i <= n; i++) {
67     cout << "x" << i << " = " << fixed << setprecision(6) << var[i] << endl;
68 }
69
70 return 0;
71 }
72
73 /*
74 This program expects the user to input the number of equations and the coefficients of
75 the augmented matrix.
76 For example:
77 2
78 2 1 5
79 3 2 8
80 */

```

```

15:34:14 ~/github/ma2071_numerical_analysis_lab main 5ms
./class_4/gauss_elimination_mthd
GAUSS ELIMINATION METHOD
Enter the number of equations: 2

Enter the augmented matrix (coefficients + constants):
Equation 1: 2 1 5
Equation 2: 3 2 8

The augmented matrix is:
    2    1    5
    3    2    8

Upper Triangular Matrix after elimination:
    2    1    5
    0    0.5  0.5

Solution:
x1 = 2.000000
x2 = 1.000000

```

Lagrange Interpolation

```

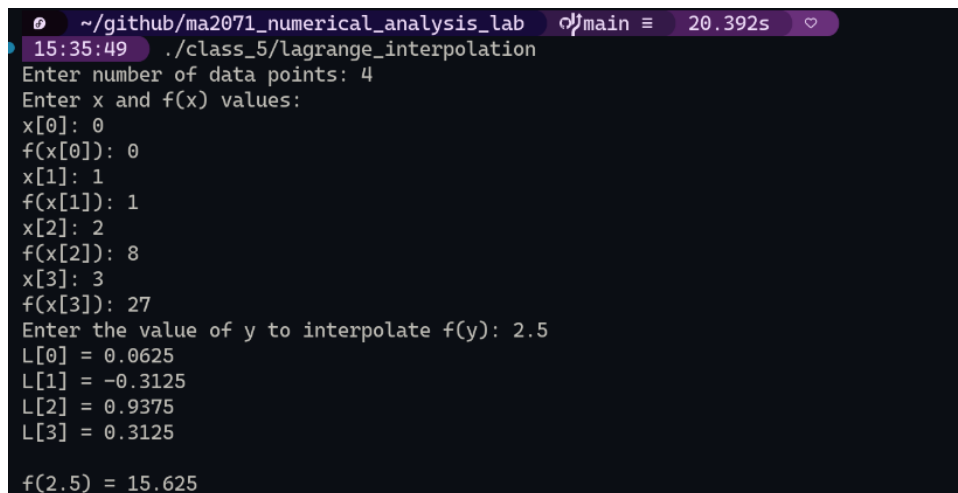
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cout << "Enter number of data points: ";
7     cin >> n;
8
9     float x[50], f[50]; // simple static arrays (could use vector too)
10    cout << "Enter x and f(x) values:\n";
11    for (int i = 0; i < n; i++) {
12        cout << "x[" << i << "]: ";
13        cin >> x[i];
14        cout << "f(x[" << i << "]): ";
15        cin >> f[i];
16    }
17
18    float y;
19    cout << "Enter the value of y to interpolate f(y): ";

```

```

20     cin >> y;
21
22     float l[50];    // Lagrange basis coefficients
23     for (int j = 0; j < n; j++) {
24         float num = 1.0, den = 1.0;
25         for (int i = 0; i < n; i++) {
26             if (i != j) {
27                 num *= (y - x[i]);
28                 den *= (x[j] - x[i]);
29             }
30         }
31         l[j] = num / den;
32         cout << "L[" << j << "] = " << l[j] << endl;
33     }
34
35     float p = 0.0f; // Interpolated result
36     for (int i = 0; i < n; i++) {
37         p += l[i] * f[i];
38     }
39
40     cout << "\nf(" << y << ") = " << p << endl;
41
42     return 0;
43 }
44
45 /*
46 This program expects the user to input the number of data points, the data points
47 themselves (x and f(x) values), and the value of y to interpolate.
48 For example:
49 4
50 0 0
51 1 1
52 2 8
53 3 27
54 2.5
55 */

```



```

~/github/ma2071_numerical_analysis_lab main 20.392s
15:35:49 ./class_5/lagrange_interpolation
Enter number of data points: 4
Enter x and f(x) values:
x[0]: 0
f(x[0]): 0
x[1]: 1
f(x[1]): 1
x[2]: 2
f(x[2]): 8
x[3]: 3
f(x[3]): 27
Enter the value of y to interpolate f(y): 2.5
L[0] = 0.0625
L[1] = -0.3125
L[2] = 0.9375
L[3] = 0.3125
f(2.5) = 15.625

```

Newton's Divided Difference

```

1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cout << "Enter number of data points: ";
7     cin >> n;
8
9     float x[50], y[50][50];
10
11     cout << "Enter x and f(x) values:\n";

```

```

12     for (int i = 0; i < n; i++) {
13         cout << "x[" << i << "]: ";
14         cin >> x[i];
15         cout << "f(x[" << i << "]): ";
16         cin >> y[i][0];
17     }
18
19     for (int j = 1; j < n; j++) {
20         for (int i = 0; i < n - j; i++) {
21             y[i][j] = (y[i + 1][j - 1] - y[i][j - 1]) / (x[i + j] - x[i]);
22         }
23     }
24
25     cout << "\nDivided Difference Table:\n";
26     for (int i = 0; i < n; i++) {
27         cout << x[i] << "\t";
28         for (int j = 0; j < n - i; j++) {
29             cout << y[i][j] << "\t";
30         }
31         cout << endl;
32     }
33
34     float value;
35     cout << "\nEnter value of x to interpolate f(x): ";
36     cin >> value;
37
38     float result = y[0][0];
39     float term = 1.0;
40
41     for (int j = 1; j < n; j++) {
42         term *= (value - x[j - 1]);
43         result += term * y[0][j];
44     }
45
46     cout << "\nf(" << value << ") = " << result << endl;
47
48     return 0;
49 }
50
51 /*
52 This program expects the user to input the number of data points, the data points
53 themselves (x and f(x) values), and the value of x to interpolate.
54 For example:
55 4
56 0 0
57 1 1
58 2 8
59 3 27
60 2.5
61 */

```



```

~/github/ma2071_numerical_analysis_lab main 5ms
15:36:54 ./class_6/newton_divided
Enter number of data points: 4
Enter x and f(x) values:
x[0]: 0
f(x[0]): 0
x[1]: 1
f(x[1]): 1
x[2]: 2
f(x[2]): 8
x[3]: 3
f(x[3]): 27

Divided Difference Table:
0      0      1      3      1
1      1      7      6
2      8      19
3      27

Enter value of x to interpolate f(x): 2.5

f(2.5) = 15.625

```

Newton's Forward Difference

```

1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cout << "Enter number of data points: ";
7     cin >> n;
8
9     float x[50], y[50][50];
10
11     cout << "Enter x and f(x) values:\n";
12     for (int i = 0; i < n; i++) {
13         cout << "x[" << i << "]: ";
14         cin >> x[i];
15         cout << "f(x[" << i << "]): ";
16         cin >> y[i][0];
17     }
18
19     for (int j = 1; j < n; j++) {
20         for (int i = 0; i < n - j; i++) {
21             y[i][j] = y[i + 1][j - 1] - y[i][j - 1];
22         }
23     }
24
25     cout << "\nForward Difference Table:\n";
26     for (int i = 0; i < n; i++) {
27         cout << x[i] << "\t";
28         for (int j = 0; j < n - i; j++) {
29             cout << y[i][j] << "\t";
30         }
31         cout << endl;
32     }
33
34     float value;
35     cout << "\nEnter value of x to interpolate f(x): ";
36     cin >> value;
37
38     float h = x[1] - x[0];
39     float p = (value - x[0]) / h;
40     float result = y[0][0];
41     float term = 1.0;
42
43     for (int j = 1; j < n; j++) {
44         term = term * (p - (j - 1)) / j;
45         result += term * y[0][j];
46     }

```

```

47     cout << "\n(" << value << ") = " << result << endl;
48
49     return 0;
50 }
51
52 /*
53  This program expects the user to input the number of data points, the data points
54 themselves (x and f(x) values), and the value of x to interpolate.
55 For example:
56 4
57 0 0
58 1 1
59 2 8
60 3 27
61 2.5
62 */

```

```

~/.github/ma2071_numerical_analysis_lab  main 28.557s
15:37:37 ./class_6/newton_forward
Enter number of data points: 4
Enter x and f(x) values:
x[0]: 0
f(x[0]): 0
x[1]: 1
f(x[1]): 1
x[2]: 2
f(x[2]): 8
x[3]: 3
f(x[3]): 27

Forward Difference Table:
0      0      1      6      6
1      1      7      12
2      8      19
3      27

Enter value of x to interpolate f(x): 2.5
f(2.5) = 15.625

```

Power Method

```

1 #include <iostream>
2 #include <cmath>
3 #include <vector>
4 using namespace std;
5
6 // Function to multiply a matrix with a vector
7 vector<double> multiplyMatrixVector(const vector<vector<double>> &A, const vector<double>
8 > &x) {
9     int n = A.size();
10    vector<double> result(n, 0.0);
11    for (int i = 0; i < n; ++i)
12        for (int j = 0; j < n; ++j)
13            result[i] += A[i][j] * x[j];
14    return result;
15 }
16
17 // Function to compute the Euclidean norm (magnitude) of a vector
18 double norm(const vector<double> &v) {
19     double sum = 0.0;
20     for (double val : v)
21         sum += val * val;
22     return sqrt(sum);
23 }
24
25 int main() {
26     int n;
27     cout << "Enter the order of the square matrix: ";

```

```

27     cin >> n;
28
29     vector<vector<double>> A(n, vector<double>(n));
30     cout << "Enter the elements of the matrix A (" << n << "x" << n << "):\n";
31     for (int i = 0; i < n; ++i)
32         for (int j = 0; j < n; ++j)
33             cin >> A[i][j];
34
35     vector<double> x(n);
36     cout << "Enter the initial guess vector (size " << n << "):\n";
37     for (int i = 0; i < n; ++i)
38         cin >> x[i];
39
40     int maxIter = 1000;
41     double tol = 1e-3;
42     double lambda_old = 0.0, lambda_new = 0.0;
43
44     cout << "\nIter\tEigenvalue\n";
45     for (int iter = 1; iter <= maxIter; ++iter) {
46         // Multiply A * x
47         vector<double> y = multiplyMatrixVector(A, x);
48
49         // Compute new eigenvalue approximation (Rayleigh quotient)
50         lambda_new = y[0] / x[0];
51
52         // Normalize y to avoid overflow/underflow
53         double y_norm = norm(y);
54         for (int i = 0; i < n; ++i)
55             x[i] = y[i] / y_norm;
56
57         cout << iter << "\t" << lambda_new << endl;
58     }
59
60     cout << "\nDominant Eigenvalue      " << lambda_new << endl;
61     cout << "Corresponding Eigenvector      [ ";
62     for (double val : x)
63         cout << val << " ";
64     cout << "]\n";
65
66     return 0;
67 }
68
69 /*
70 This program expects the user to input the order of the square matrix, the elements of
71 the matrix, and the initial guess vector.
72 For example:
73 3
74 1 2 0
75 2 1 0
76 0 0 5
77 1 1 1
78 */

```

```

~/github/ma2071_numerical_analysis_lab main 6ms
15:39:18 ./class_8/power_mthd
Enter the order of the square matrix: 3
Enter the elements of the matrix A (3x3):
1 2 0
2 1 0
0 0 5
Enter the initial guess vector (size 3):
1
1
1

Iter    Eigenvalue
1       3
2       3

Dominant Eigenvalue ≈ 3
Corresponding Eigenvector ≈ [ 0.320815 0.320815 0.891154 ]

```

Euler's Method

```

1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4 using namespace std;
5
6 // Example differential equation:  $dy/dx = x + y$ 
7 double f(double x, double y) {
8     return x + y;
9 }
10
11 int main() {
12     double x0, y0, h, xn;
13
14     cout << "Enter initial x0 and y0: ";
15     cin >> x0 >> y0;
16     cout << "Enter step size h: ";
17     cin >> h;
18     cout << "Enter x at which to find y (xn): ";
19     cin >> xn;
20
21     cout << fixed << setprecision(6);
22     cout << "\nEulers Method:\n";
23     cout << "x\t\ty\n";
24
25     while (x0 < xn) {
26         y0 = y0 + h * f(x0, y0);
27         x0 = x0 + h;
28         cout << x0 << "\t" << y0 << endl;
29     }
30
31     cout << "\nApproximate solution at x = " << xn << " is y = " << y0 << endl;
32     return 0;
33 }
34
35 /*
36 This program expects the user to input the initial x0 and y0 values, the step size h,
37 and the xn value at which to find y.
38 For example:
39 0 1
40 0.1
41 0.5
42 */

```

```

~/github/ma2071_numerical_analysis_lab 15:41:57 main 6ms
./class_9/euler_mthd
Enter initial x0 and y0: 0 1
Enter step size h: 0.1
Enter x at which to find y (xn): 0.5

Euler's Method:
x          y
0.100000   1.100000
0.200000   1.220000
0.300000   1.362000
0.400000   1.528200
0.500000   1.721020

Approximate solution at x = 0.500000 is y = 1.721020
~/github/ma2071_numerical_analysis_lab 15:42:10 main 11.133s
15:42:10

```

RK2 System

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
```

```

4
5 // Example system:
6 //  $dy_1/dx = y_2$ 
7 //  $dy_2/dx = -y_1$ 
8 // (Simple harmonic oscillator)
9
10 double f1(double x, double y1, double y2) {
11     return y2;
12 }
13
14 double f2(double x, double y1, double y2) {
15     return -y1;
16 }
17
18 int main() {
19     double x0, y1, y2, h, xn;
20     cout << "Enter initial x0, y1(0), y2(0): ";
21     cin >> x0 >> y1 >> y2;
22     cout << "Enter step size h: ";
23     cin >> h;
24     cout << "Enter final x value xn: ";
25     cin >> xn;
26
27     cout << fixed << setprecision(6);
28     cout << "\n2nd Order Runge Kutta (System of 2 ODEs)\n";
29     cout << "x\t\t y1\t\t y2\n";
30
31     while (x0 < xn) {
32         // Compute k-values for RK2 (Heuns method)
33         double k1_1 = h * f1(x0, y1, y2);
34         double k1_2 = h * f2(x0, y1, y2);
35
36         double k2_1 = h * f1(x0 + h, y1 + k1_1, y2 + k1_2);
37         double k2_2 = h * f2(x0 + h, y1 + k1_1, y2 + k1_2);
38
39         // Update y1, y2
40         y1 = y1 + 0.5 * (k1_1 + k2_1);
41         y2 = y2 + 0.5 * (k1_2 + k2_2);
42         x0 += h;
43
44         cout << x0 << "\t" << y1 << "\t" << y2 << endl;
45     }
46
47     cout << "\nApproximate solution at x = " << xn << ":\n";
48     cout << "y1 = " << y1 << ", y2 = " << y2 << endl;
49     return 0;
50 }
51
52 /*
53 This program expects the user to input the initial x0, y1(0), y2(0) values, the step
54 size h, and the final xn value.
55 For example:
56 0 0 1
57 0.1
58 0.5
59 */

```

```
~/github/ma2071_numerical_analysis_lab main 11.133s
15:42:10 ./class_9/Rk2_system
Enter initial x0, y1(0), y2(0): 0 0 1
Enter step size h: 0.1
Enter final x value xn: 0.5

2nd Order Runge-Kutta (System of 2 ODEs)
x          y1          y2
0.100000   0.100000   0.995000
0.200000   0.199000   0.980025
0.300000   0.296007   0.955225
0.400000   0.390050   0.920848
0.500000   0.480185   0.877239

Approximate solution at x = 0.500000:
y1 = 0.480185, y2 = 0.877239
```