Numerical Analysis Code

Omm Satyakam Behera
Roll No: 424MA5005
NIT Rourkela

Course Details

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

Numerical Methods

Bisection Method

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

#define EP 0.0001  // Smaller epsilon for better precision

// A function with a real root
double solution(double x) {
   return x*x*x + x - 1;  // Example: f(x) = x^3 + x - 1
}

void bisection(double a, double b) {
   if (solution(a) * solution(b) >= 0) {
      cout << "You have not assumed correct a and b\n";
      return;
   }

   double c;
   int iterations = 0;

   while ((b - a) >= EP) {
      c = (a + b) / 2;
      double fc = solution(c);

      if (fc == 0.0)
         break;

      if (solution(a) * fc < 0)
         b = c;
      else
         a = c;

      iterations++;
   }

   cout << "The value of root is : " << c << endl;
   cout << "Found in " << iterations << " iterations." << endl;
}

int main() {
   double a = 0, b = 1;
   bisection(a, b);
   return 0;
}

/*
```

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

Secant Method

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

// Define the function
float f(float x) {
    return pow(x, 3) + x - 1; // Equation: x^3 + x - 1
}

void secant(float x1, float x2, float E) {
    float x0, f1, f2;
    int n = 0;

    do {
        f1 = f(x1);
        f2 = f(x2);

        if (f2 - f1 == 0) {
            cout << "Division by zero error in secant formula." << endl;
            return;
        }

        x0 = (x1 * f2 - x2 * f1) / (f2 - f1);

        // Update values
        x1 = x2;
        x2 = x0;
        n++;

    } while (fabs(f(x2) - f(x1)) >= E);

    cout << fixed << setprecision(10);
    cout << "Root of the equation = " << x0 << endl;
    cout << "Number of iterations = " << n << endl;
}

int main() {
    float x1 = 0, x2 = 1, E = 1e-7;
    secant(x1, x2, E);
    return 0;
}

/*
This program does not take any user input. The initial values for the secant method are hardcoded in the main function.
x1 = 0
x2 = 1
E = 1e-7
*/
```

Newton-Raphson Method

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

// Define functions
double f1(double x, double y) { return pow(x, 3) + y - 1; }
double f2(double x, double y) { return pow(y, 3) - x + 1; }

```

```cpp
// Partial derivatives (Jacobian)
double f1x(double x, double y) { return 3 * x * x; }  //   f1  /  x
double f1y(double x, double y) { return 1; }          //   f1  /  y
double f2x(double x, double y) { return -1; }         //   f2  /  x
double f2y(double x, double y) { return 3 * y * y; }  //   f2  /  y

int main() {
    cout << "Newton's Method for solving system of equations\n";

    double x, y;
    cout << "Enter initial guesses for x and y: ";
    cin >> x >> y;

    int maxIter = 20;
    double tol = 1e-6;

    for (int iter = 1; iter <= maxIter; iter++) {
        // Function values
        double F1 = f1(x, y);
        double F2 = f2(x, y);

        // Jacobian matrix
        double J11 = f1x(x, y), J12 = f1y(x, y);
        double J21 = f2x(x, y), J22 = f2y(x, y);

        // Determinant
        double det = J11 * J22 - J12 * J21;
        if (fabs(det) < 1e-12) {
            cout << "Jacobian is singular. Stopping.\n";
            break;
        }

        // Inverse of 2x2 Jacobian * F
        double dx = (-F1 * J22 + F2 * J12) / det;
        double dy = (-J11 * F2 + J21 * F1) / det;

        // Update guesses
        x += dx;
        y += dy;

        cout << "Iteration " << iter << ": x = "
             << fixed << setprecision(6) << x
             << ", y = " << y
             << " | dx = " << dx << ", dy = " << dy << endl;

        // Check convergence
        if (fabs(dx) < tol && fabs(dy) < tol) {
            cout << "Converged to solution.\n";
            break;
        }
    }

    cout << "Final solution: x = " << x << ", y = " << y << endl;
    return 0;
}

/*
This program expects the user to input initial guesses for x and y.
For example:
-0.5
0.5
*/
```

Fixed Point Method

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

double iterat(double x1, double x2, double x3, int i) {
    i++;

    // g1(x) = (x^2 - 3)/2
    double y1 = (x1 * x1 - 3) / 2.0;

    // g2(x) = 3 / (x - 2)
    double y2 = 3.0 / (x2 - 2.0);

    // g3(x) = sqrt(2x + 3)
    double y3 = sqrt(2.0 * x3 + 3.0);

    double d = y3 - x3;

    cout << "After iteration no. " << i << ":\t"
         << y1 << "\t\t" << y2 << "\t\t" << y3 << endl;

    if (fabs(d) > 1e-5)
        return iterat(y1, y2, y3, i);  // continue iteration
    else
        return y3;  // converged
}

int main() {
    int i = 0;
    double x = 2.5, y;

    cout << "After iteration no. " << i
         << ":\tg1(x)\t\tg2(x)\t\tg3(x)\n";

    y = iterat(x, x, x, i);

    cout << "The answer is x = " << y << endl;
    return 0;
}

/*
This program does not take any user input. The initial value for the fixed-point iteration is hardcoded in the main function.
x = 2.5
*/
```

Gauss Elimination Method

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

int main() {
    int n;
    cout << "GAUSS ELIMINATION METHOD\n";
    cout << "Enter the number of equations: ";
    cin >> n;

    double coeff[20][20], var[20], temp, pivratio;

    // Input augmented matrix
    cout << "\nEnter the augmented matrix (coefficients + constants):\n";
    for (int i = 1; i <= n; i++) {
        cout << "Equation " << i << ": ";
        for (int j = 1; j <= n + 1; j++) {
            cin >> coeff[i][j];
        }
    }

    // Display augmented matrix
    cout << "\nThe augmented matrix is:\n";
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n + 1; j++) {
            cout << setw(10) << coeff[i][j] << " ";
        }
        cout << "\n";
    }

    // Forward elimination
    for (int i = 1; i <= n - 1; i++) {
        for (int j = i + 1; j <= n; j++) {
            if (coeff[i][i] == 0) {
                cout << "Mathematical Error: Zero pivot element.\n";
                return 1;
            }
            pivratio = coeff[j][i] / coeff[i][i];
            for (int k = i; k <= n + 1; k++) {
                coeff[j][k] -= pivratio * coeff[i][k];
            }
        }
    }

    // Display upper triangular matrix
    cout << "\nUpper Triangular Matrix after elimination:\n";
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n + 1; j++) {
            cout << setw(10) << coeff[i][j] << " ";
        }
        cout << "\n";
    }

    // Back substitution
    for (int i = n; i >= 1; i--) {
        temp = coeff[i][n + 1];
        for (int j = i + 1; j <= n; j++) {
            temp -= coeff[i][j] * var[j];
        }
        var[i] = temp / coeff[i][i];
    }

    // Output solution
    cout << "\nSolution:\n";
    for (int i = 1; i <= n; i++) {
        cout << "x" << i << " = " << fixed << setprecision(6) << var[i] << endl;
    }

    return 0;
}

/*
This program expects the user to input the number of equations and the coefficients of the augmented matrix.
For example, for a 2x2 system:
2
2 1 5
3 2 8
*/
```

Lagrange Interpolation

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

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

    float x[50], f[50];  // simple static arrays (could use vector too)
    cout << "Enter x and f(x) values:\n";
    for (int i = 0; i < n; i++) {
        cout << "x[" << i << "]:";
        cin >> x[i];
        cout << "f(x[" << i << "]): ";
        cin >> f[i];
    }

    float y;
    cout << "Enter the value of y to interpolate f(y): ";
    cin >> y;

    float l[50];   // Lagrange basis coefficients
    for (int j = 0; j < n; j++) {
        float num = 1.0, den = 1.0;
        for (int i = 0; i < n; i++) {
            if (i != j) {
                num *= (y - x[i]);
                den *= (x[j] - x[i]);
            }
        }
        l[j] = num / den;
        cout << "L[" << j << "] = " << l[j] << endl;
    }

    float p = 0.0f; // Interpolated result
    for (int i = 0; i < n; i++) {
        p += l[i] * f[i];
    }

    cout << "\nf(" << y << ") = " << p << endl;

    return 0;
}

/*
This program expects the user to input the number of data points, the data points themselves (x and f(x) values), and the value of y to interpolate.
For example:
4
0 0
1 1
2 8
3 27
2.5
*/
```

Newton’s Divided Difference

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

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

    float x[50], y[50][50];

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

    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]) / (x[i + j] - x[i]);
        }
    }

    cout << "\nDivided Difference Table:\n";
    for (int i = 0; i < n; i++) {
        cout << x[i] << "\t";
        for (int j = 0; j < n - i; j++) {
            cout << y[i][j] << "\t";
        }
        cout << endl;
    }

    float value;
    cout << "\nEnter value of x to interpolate f(x): ";
    cin >> value;

    float result = y[0][0];
    float term = 1.0;

    for (int j = 1; j < n; j++) {
        term *= (value - x[j - 1]);
        result += term * y[0][j];
    }

    cout << "\nf(" << value << ") = " << result << endl;

    return 0;
}

/*
This program expects the user to input the number of data points, the data points themselves (x and f(x) values), and the value of x to interpolate.
For example:
4
0 0
1 1
2 8
3 27
2.5
*/
```

Newton’s Forward Difference

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

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

    float x[50], y[50][50];

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

    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];
        }
    }

    cout << "\nForward Difference Table:\n";
    for (int i = 0; i < n; i++) {
        cout << x[i] << "\t";
        for (int j = 0; j < n - i; j++) {
            cout << y[i][j] << "\t";
        }
        cout << endl;
    }

    float value;
    cout << "\nEnter value of x to interpolate f(x): ";
    cin >> value;

    float h = x[1] - x[0];
    float p = (value - x[0]) / h;
    float result = y[0][0];
    float term = 1.0;

```

```cpp
    for (int j = 1; j < n; j++) {
        term = term * (p - (j - 1)) / j;
        result += term * y[0][j];
    }

    cout << "\nf(" << value << ") = " << result << endl;

    return 0;
}

/*
This program expects the user to input the number of data points, the data points themselves (x and f(x) values), and the value of x to interpolate.
For example:
4
0 0
1 1
2 8
3 27
2.5
*/
```

Power Method

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

// Function to multiply a matrix with a vector
vector<double> multiplyMatrixVector(const vector<vector<double>> &A, const vector<double> &x) {
    int n = A.size();
    vector<double> result(n, 0.0);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i] += A[i][j] * x[j];
    return result;
}

// Function to compute the Euclidean norm (magnitude) of a vector
double norm(const vector<double> &v) {
    double sum = 0.0;
    for (double val : v)
        sum += val * val;
    return sqrt(sum);
}

int main() {
    int n;
    cout << "Enter the order of the square matrix: ";
    cin >> n;

    vector<vector<double>> A(n, vector<double>(n));
    cout << "Enter the elements of the matrix A (" << n << "x" << n << "):\n";
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> A[i][j];

    vector<double> x(n);
    cout << "Enter the initial guess vector (size " << n << "):\n";
    for (int i = 0; i < n; ++i)
        cin >> x[i];

    int maxIter = 1000;
    double tol = 1e-3;
    double lambda_old = 0.0, lambda_new = 0.0;

    cout << "\nIter\tEigenvalue\n";
    for (int iter = 1; iter <= maxIter; ++iter) {
        // Multiply A * x
        vector<double> y = multiplyMatrixVector(A, x);

        // Compute new eigenvalue approximation (Rayleigh quotient)
        lambda_new = y[0] / x[0];

        // Normalize y to avoid overflow/underflow
        double y_norm = norm(y);
        for (int i = 0; i < n; ++i)
            x[i] = y[i] / y_norm;

        cout << iter << "\t" << lambda_new << endl;

        // Check convergence
        if (fabs(lambda_new - lambda_old) < tol)
            break;

        lambda_old = lambda_new;
    }

    cout << "\nDominant Eigenvalue     " << lambda_new << endl;
    cout << "Corresponding Eigenvector     [ ";
    for (double val : x)
        cout << val << " ";
    cout << "]\n";

    return 0;
}

/*
This program expects the user to input the order of the square matrix, the elements of the matrix, and the initial guess vector.
For example:
3
1 2 0
2 1 0
0 0 5
1 1 1
*/
```

Euler's Method

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

// Example differential equation: dy/dx = x + y
double f(double x, double y) {
    return x + y;
}

int main() {
    double x0, y0, h, xn;

    cout << "Enter initial x0 and y0: ";
    cin >> x0 >> y0;
    cout << "Enter step size h: ";
    cin >> h;
    cout << "Enter x at which to find y (xn): ";
    cin >> xn;

    cout << fixed << setprecision(6);
    cout << "\nEulers Method:\n";
    cout << "x\t\ty\n";

    while (x0 < xn) {
        y0 = y0 + h * f(x0, y0);
        x0 = x0 + h;
        cout << x0 << "\t" << y0 << endl;
    }

    cout << "\nApproximate solution at x = " << xn << " is y = " << y0 << endl;
    return 0;
}

/*
This program expects the user to input the initial x0 and y0 values, the step size h, and the xn value at which to find y.
For example:
0 1
0.1
0.5
*/
```

RK2 System

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

// Example system:
// dy1/dx = y2
// dy2/dx = -y1
// (Simple harmonic oscillator)

double f1(double x, double y1, double y2) {
    return y2;
}

double f2(double x, double y1, double y2) {
    return -y1;
}

int main() {
    double x0, y1, y2, h, xn;
    cout << "Enter initial x0, y1(0), y2(0): ";
    cin >> x0 >> y1 >> y2;
    cout << "Enter step size h: ";
    cin >> h;
    cout << "Enter final x value xn: ";
    cin >> xn;

    cout << fixed << setprecision(6);
    cout << "\n2nd Order RungeKutta (System of 2 ODEs)\n";
    cout << "x\t\ty1\t\ty2\n";

    while (x0 < xn) {
        // Compute k-values for RK2 (Heuns method)
        double k1_1 = h * f1(x0, y1, y2);
        double k1_2 = h * f2(x0, y1, y2);

        double k2_1 = h * f1(x0 + h, y1 + k1_1, y2 + k1_2);
        double k2_2 = h * f2(x0 + h, y1 + k1_1, y2 + k1_2);

        // Update y1, y2
        y1 = y1 + 0.5 * (k1_1 + k2_1);
        y2 = y2 + 0.5 * (k1_2 + k2_2);
        x0 += h;

        cout << x0 << "\t" << y1 << "\t" << y2 << endl;
    }

    cout << "\nApproximate solution at x = " << xn << ":\n";
    cout << "y1 = " << y1 << ", y2 = " << y2 << endl;
    return 0;
}

/*
This program expects the user to input the initial x0, y1(0), y2(0) values, the step size h, and the final xn value.
For example:
0 0 1
0.1
0.5
*/
```