Numerical Methods

Bisection Method

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

double f(double x) {
    return cos(x) - x * exp(x);
}

void bisection(double a, double b, int maxIterations, double tolerance) {
    double mid;
    for (int i = 0; i < maxIterations; i++) {
        mid = (a + b) / 2.0;
        if (f(mid) == 0 || (b - a) / 2 < tolerance) {
            cout << "Root found: " << mid << endl;
            return;
        } else if (f(mid) * f(a) < 0) {
            b = mid;
        } else {
            a = mid;
        }
    }
    cout << "Root after " << maxIterations << " iterations: " << mid << endl;
}

int main() {
    double a = 0.0, b = 1.0;
    int maxIterations = 20;
    double tolerance = 1e-6;

    bisection(a, b, maxIterations, tolerance);

    return 0;
}
```

Regula Falsi Method

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

double f(double x) {
    return cos(x) - x * exp(x);
}

void regulaFalsi(double a, double b, int maxIterations, double tolerance) {
    double c;
    for (int i = 0; i < maxIterations; i++) {
        c = (a * f(b) - b * f(a)) / (f(b) - f(a));
        if (f(c) == 0 || fabs(f(c)) < tolerance) {
            cout << "Root found: " << c << endl;
            return;
        } else if (f(c) * f(a) < 0) {
            b = c;
        } else {
            a = c;
        }
    }
    cout << "Root after " << maxIterations << " iterations: " << c << endl;
}

int main() {
    double a = 0.0, b = 1.0;
    int maxIterations = 20;
    double tolerance = 1e-6;

    regulaFalsi(a, b, maxIterations, tolerance);

    return 0;
}
```

Secant Method

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

double f(double x) {
    return cos(x) - x * exp(x);
}

void secant(double x0, double x1, int maxIterations, double tolerance) {
    double x2;
    for (int i = 0; i < maxIterations; i++) {
        x2 = x1 - (f(x1) * (x1 - x0)) / (f(x1) - f(x0));
        if (fabs(f(x2)) < tolerance) {
            cout << "Root found: " << x2 << endl;
            return;
        }
        x0 = x1;
        x1 = x2;
    }
    cout << "Root after " << maxIterations << " iterations: " << x2 << endl;
}

int main() {
    double x0 = 0.0, x1 = 1.0;
    int maxIterations = 10;
    double tolerance = 1e-6;

    secant(x0, x1, maxIterations, tolerance);

    return 0;
}
```

Newton-Raphson

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

double f(double x) {
    return x * exp(x) - 1;
}

double df(double x) {
    return exp(x) + x * exp(x);
}

void newtonRaphson(double x0, int maxIterations, double tolerance) {
    double x1;
    for (int i = 0; i < maxIterations; i++) {
        x1 = x0 - f(x0) / df(x0);
        if (fabs(f(x1)) < tolerance) {
            cout << "Root found: " << x1 << endl;
            return;
        }
        x0 = x1;
    }
    cout << "Root after " << maxIterations << " iterations: " << x1 << endl;
}

int main() {
    double x0 = 0.5;
    int maxIterations = 10;
    double tolerance = 1e-6;

    newtonRaphson(x0, maxIterations, tolerance);

    return 0;
}
```

Gauss-Elimination

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

void gaussianElimination(vector<vector<double>> &A, vector<double> &B) {
    int n = A.size();

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            double factor = A[j][i] / A[i][i];
            for (int k = i; k < n; k++) {
                A[j][k] -= factor * A[i][k];
            }
            B[j] -= factor * B[i];
        }
    }
    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];
        }
        X[i] /= A[i][i];
    }

    cout << "Solution using Gaussian Elimination: ";
    for (int i = 0; i < n; i++) {
        cout << fixed << setprecision(4) << "x" << i + 1 << " = " << X[i] << " ";
    }
    cout << endl;
}

int main() {
    vector<vector<double>> A = {
        {1.19, 2.11, -100.0, 1.0},
        {14.2, -0.122, 12.2, -1.0},
        {0.0, 100.0, -99.9, 1.0},
        {15.3, 0.11, -13.1, 0.0}
    };

    vector<double> B = {1.12, 3.44, 2.15, 4.16};

    gaussianElimination(A, B);

    return 0;
}
```

Gauss-Jordan

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

void gaussJordan(vector<vector<double>> &A, vector<double> &B) {
    int n = A.size();

    for (int i = 0; i < n; i++) {
        // Make the diagonal element 1
        double diag = A[i][i];
        for (int j = 0; j < n; j++) {
            A[i][j] /= diag;
        }
        B[i] /= diag;
        for (int j = 0; j < n; j++) {
            if (j != i) {
                double factor = A[j][i];
                for (int k = 0; k < n; k++) {
                    A[j][k] -= factor * A[i][k];
                }
                B[j] -= factor * B[i];
            }
        }
    }

    cout << "Solution using Gauss-Jordan Elimination: ";
    for (int i = 0; i < n; i++) {
        cout << fixed << setprecision(4) << "x" << i + 1 << " = " << B[i] << " ";
    }
    cout << endl;
}

int main() {
    vector<vector<double>> A = {
        {1.19, 2.11, -100.0, 1.0},
        {14.2, -0.122, 12.2, -1.0},
        {0.0, 100.0, -99.9, 1.0},
        {15.3, 0.11, -13.1, 0.0}
    };

    vector<double> B = {1.12, 3.44, 2.15, 4.16};

    gaussJordan(A, B);
    return 0;
}
```

Gauss-Jacobi

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

void gaussJacobi(vector<vector<double>>& a, vector<double>& b, int n, int max_iter = 1
00, double tol = 1e-5) {
    vector<double> x(n, 0), x_new(n, 0);
    for (int iter = 0; iter < max_iter; iter++) {
        for (int i = 0; i < n; i++) {
            double sum = 0;
            for (int j = 0; j < n; j++) {
                if (i != j) {
                    sum += a[i][j] * x[j];
                }
            }
            x_new[i] = (b[i] - sum) / a[i][i];
        }
        double error = 0;
        for (int i = 0; i < n; i++) {
            error += fabs(x_new[i] - x[i]);
            x[i] = x_new[i];
        }
        if (error < tol) break;
    }
    for (int i = 0; i < n; i++) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
}

int main() {
    int n = 3;
    vector<vector<double>> a = {
        {5, -2, 3},
        {-3, 9, 1},
        {2, -1, -7}
    };
    vector<double> b = {-1, 2, 3};
    gaussJacobi(a, b, n);
    return 0;
}
```

Gauss-Siedel

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

void gaussSeidel(vector<vector<float> >& a, vector<float>& b, int n, int max_iter = 10
0, float tol = 1e-5) {
    vector<float> x(n, 0);
    for (int iter = 0; iter < max_iter; iter++) {
        float error = 0;
        for (int i = 0; i < n; i++) {
            float sum = 0;
            for (int j = 0; j < n; j++) {
                if (i != j) {
                    sum += a[i][j] * x[j];
                }
            }
            float x_new = (b[i] - sum) / a[i][i];
            error += fabs(x_new - x[i]);
            x[i] = x_new;
        }
        if (error < tol) break;
    }
    for (int i = 0; i < n; i++) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
}

int main() {
    int n = 3;
    vector<vector<float> > a = {
        {5, -2, 3},
        {-3, 9, 1},
        {2, -1, -7}
    };
    vector<float> b = {-1, 2, 3};

    gaussSeidel(a, b, n);

    return 0;
}
```

Lagranges-Interpolation

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

double lagrangesInterpolation(double x[], double y[], int n, double xp) {
    double yp = 0;
    for (int i = 0; i < n; i++) {
        double p = y[i];
        for (int j = 0; j < n; j++) {
            if (i != j) {
                p = p * (xp - x[j])/(x[i] - x[j]);
            }
        }
        yp += p;
    }
    return yp;
}

int main () {
    int n; cout << "Enter n: "; cin >> n;
    double x[n-1], y[n-1];
    for (int i = 0; i < n-1; i++) {
        cout << "x[" << i << "]: ";
        cin >> x[i];
    }
    for (int i = 0; i < n-1; i++) {
        cout << "y[" << i << "]: ";
        cin >> y[i];
    }
    double xp; cout << "Enter xp: " ; cin >> xp;
    cout << "Interpolated Value at x = " << xp << " is " << lagrangesInterpolation(x,
y,n,xp) << endl;
    return 0;
}
```

Newton-Forward Interpolation

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

void newtonForwardInterpolation(double x[], double y[], int n, double xp) {
    double h = x[1]-x[0];
    double diff[n][n];
    for (int i = 0; i < n; i++) {
        diff[i][0] = y[i];
    }
    for (int j = 1; j < n; j++) {
        for (int i = 0; i < n-j; i++) {
            diff[i][j] = diff[i+1][j-1] - diff[i][j-1];
        }
    }
    double s = diff[0][0];
    double u = (xp - x[0])/h;
    double uterm = u;
    for (int i = 1; i < n; i++) {
        s += (uterm * diff[0][i]/i);
        uterm *= (u-i);
    }
    cout << "The interpolated value at x = " << xp << " is " << s << endl;
}

int main() {
    int n = 7;
    double x[] = {1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3};
    double y[] = {5.474, 6.050, 6.686, 7.389, 8.166, 9.025, 9.974};
    double xp = 1.85;
    newtonForwardInterpolation(x,y,n,xp);
    return 0;
}
```

Newton Backward-Interpolation

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

void newtonForwardInterpolation(double x[], double y[], int n, double xp) {
    double h = x[1]-x[0];
    double diff[n][n];
    for (int i = 0; i < n; i++) {
        diff[i][0] = y[i];
    }
    for (int j = 1; j < n; j++) {
        for (int i = n-1; i >= j; i--) {
            diff[i][j] = diff[i][j-1] - diff[i-1][j-1];
        }
    }
    double s = diff[n-1][0];
    double u = (xp - x[n-1])/h;
    double uterm = u;
    for (int i = 1; i < n; i++) {
        s += (uterm * diff[n-1][i]/i);
        uterm *= (u+i);
    }
    cout << "The interpolated value at x = " << xp << " is " << s << endl;
}

int main() {
    int n = 7;
    double x[] = {1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3};
    double y[] = {5.474, 6.050, 6.686, 7.389, 8.166, 9.025, 9.974};
    double xp = 2.25;
    newtonForwardInterpolation(x,y,n,xp);
    return 0;
}
```

Trapezoidal, Simpson’s 1/3 & Simpson’s 3/8

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

double f(double x) {
    return 1.0/(1+x*x);
}

double trapezoidalRule(double a, double b, int n) {
    double h = (b-a)/n, x = (f(a)+f(b))/2.0;
    for (int i = 1; i < n; i++) {
        x += f(a+i*h);
    }
    return x*h;
}

double simpsonsOneThirdRule(double a, double b, int n) {
    if (n % 2 != 0) {
        cout << "Error Even Interval";
        return -1;
    }
    double h = (b-a)/n, x = f(a)+f(b);
    for (int i = 1; i < n; i += 2) {
            x += 4*f(a+i*h);
    }
    for (int i = 2; i < n-1; i += 2) {
        x += 2*f(a+i*h);
    }
    return h/3*x;
}

double simpsonsThreeEighthRule(double a, double b, int n) {
    if (n % 3 != 0) {
        cout << "Error Interval is not a multiple of 3";
        return -1;
    }
    double h = (b-a)/n, x = f(a)+f(b);
    for (int i = 1; i < n; i++) {
        if (i % 3 == 0) {
            x += 2*f(a+i*h);
        } else {
            x += 3*f(a+i*h);
        }
    }
    return 3*h/8*x;
}


int main() {
    double a = 0.0, b = 1.0;
    int intervals[] = {10,12,15,18,20};
    cout << fixed << setprecision(8);
    for (int n: intervals) {
        double approx = trapezoidalRule(a,b,n);
        double exact = atan(1);
        double error = fabs(exact-approx);
        cout << "Trapezoidal Rule with n = " << n << " : " << approx << ", Absolute er
ror = " << error << endl;
    }
    for (int n: intervals) {
        if (n % 2 == 0) {
            double approx = simpsonsOneThirdRule(a,b,n);
            double exact = atan(1);
            double error = fabs(exact-approx);
            cout << "Simpson's 1/3 with n = " << n << " : " << approx << ", Absolute e
rror = " << error << endl;
        }
    }
    for (int n: intervals) {
        if (n % 3 == 0) {
            double approx = simpsonsThreeEighthRule(a,b,n);
            double exact = atan(1);
            double error = fabs(exact-approx);
            cout << "Simpson's 3/8 with n = " << n << " : " << approx << ", Absolute e
rror = " << error << endl;
        }
    }
    return 0;
}
```

Euler Method

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

double f(double x, double y) {
    return x+y;
}

void eulerMethod(double x0, double y0, double h, double xf) {
    double x = x0;
    double y = y0;
    while (x < xf-h) {
        if (x == 1.0) break;
        y += h*f(x,y);
        x += h;
    }
    cout << "The value of y at x = " << xf << " is " << y << endl;
}

int main () {
    double x0 = 0.0;
    double y0 = 1.0;
    double h = 0.1;
    double xf = 1.0;
    eulerMethod(x0, y0,h,xf);
    return 0;
}
```

Runge Kutta 4

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

double f(double x, double y) {
    return (x*x)+(y*y);
}

void rungeKutta(double x0, double y0, double h, double xf) {
    double x = x0;
    double y = y0;
    while (x < xf) {
        double k1 = h*f(x,y);
        double k2 = h*f(x+h/2,y+k1/2);
        double k3 = h*f(x+h/2,y+k2/2);
        double k4 = h*f(x+h,y+k3);

        y += (k1+2*k2+2*k3+k4)/6;
        x += h;
    }
    cout << "The value of y at x = " << xf << " is " << y << endl;
}

int main () {
    double x0 = 1.0;
    double y0 = 1.2;
    double h = 0.05;
    double xf = 1.05;
    rungeKutta(x0, y0,h,xf);
    return 0;
}
```