# 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

```
#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;
}
```