Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Численные методы»

Лабораторные работы

Студент: Хренов Геннадий

Группа: 80-307Б

Преподаватель: Ревизников Д. Л.

Дата:

Оценка:

Москва, 2021

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

input matrix:

2.000 -7.000 8.000 -4.000

0.000 -1.000 4.000 -1.000

3.000 -4.000 2.000 -1.000

-9.000 1.000 -4.000 6.000

LU-decomposition:

L =

1.000 0.000 0.000 0.000

-0.222 1.000 0.000 0.000

-0.333 0.541 1.000 0.000

-0.000 0.148 -0.928 1.000

U =

-9.000 1.000 -4.000 6.000

0.000 -6.778 7.111 -2.667

0.000 -0.000 -3.180 2.443

0.000 0.000 0.000 1.660

L * U =

-9.000 1.000 -4.000 6.000

2.000 -7.000 8.000 -4.000

3.000 -4.000 2.000 -1.000

0.000 -1.000 4.000 -1.000

system solution:

x = (1 -5 7 9)

determinant of matrix:

det = -322

reverse matrix:

Arev=

-0.062 0.099 0.009 -0.056

-0.130 -0.391 -0.130 -0.217

-0.099 0.559 0.115 0.311

0.248 0.602 0.463 0.224

```
#include <iostream>
#include <vector>
#include "TMatrix.cpp"

using namespace std;



double Udet(TMatrix &U) {
    double det = 1;
    for (int i = 0; i < U.Size(); i++) {
        det *= U[i][i];
    }
    return pow(-1, U.swaps.size()) * det;
}


void ReadFromFile(vector<vector<double>>& vec, vector<double>& b) {

    double c;
    int n;
    cin >> n;
    vec.resize(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> c;
            vec[i].push_back(c);
        }
    }
    for (int i = 0; i < n; i++) {
        cin >> c;
        b.push_back(c);
    }
    return;
}

int main() {

    vector<vector<double>> vec;
```

```
    vector<double> b;
    ReadFromFile(vec, b);

    TMatrix A(vec);
    cout << "input matrix:\n";
    Print(A);
    TMatrix U(A.Size()), L(A.Size());
    LU(A, L, U);

    cout << "LU-decomposition:\n L =\n";
    Print(L);
    cout <<" U =\n";
    Print(U);

    cout << "L * U =\n";
    TMatrix res = L * U;
    Print(res);

    for (int i = 0; i < U.swaps.size(); i++) {
        double tmp = b[U.swaps[i].first];
        b[U.swaps[i].first] = b[U.swaps[i].second];
        b[U.swaps[i].second] = tmp;
    }
   vector<double> res2 = solveOfSystem(L, U, b);

    cout << "system solution:\n\n x = (  ";
    for (int i = 0; i < res2.size(); i++) {
        cout  << res2[i] << "  ";
    }
    cout << ")\n\n";


    double det = Udet(U);
    cout << "determinant of matrix:\ndet = " << det << "\n\n";

    TMatrix Arev = reverse(L,U);
    cout << "reverse matrix:\n Arev= \n";
    Print(Arev);

   //TMatrix res3 = Arev * A;
    //cout << "Arev * A\n";
    //Print(res3);


    return 0;
}
```

1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

run factors :

P = (0.714286 0.61165 0.488482 -0.138702 0)

Q = (5.42857 3.16505 4.46545 -4.24832 -9)

answer

X = (9 5 3 -3 -9)

```
#include <iostream>
#include <vector>
#include "TMatrix.cpp"

using namespace std;

void ReadFromFile(vector<double>& a, vector<double>& b, vector<double>& c,
vector<double>& q) {

    double c1, c2, c3, size;
    cin >> size;
    for (int i = 0; i < size; i++) {
        if (i == 0) {
            cin >> c2 >> c3;
            c1 = 0;
        } else if (i == size - 1){
            cin >> c1 >> c2;
            c3 = 0;
        } else {
            cin >> c1 >> c2 >> c3;
        }
        a.push_back(c1);
        b.push_back(c2);
        c.push_back(c3);
    }

    for (int i = 0; i < size; i++) {
        cin >> c1;
        q.push_back(c1);
    }
}

int main() {

    vector<double> d, a, b, c;
    ReadFromFile(a, b, c, d);

    vector<double> p(d.size());
    vector<double> q(d.size());

    p[0] = -c[0] / b[0];
    q[0] = d[0] / b[0];

    for (int i = 1; i < p.size(); i++) {
        p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
        q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
    }

    cout << "run factors :\n P = ( ";
    for(int i = 0; i < p.size(); i++) {
        cout << p[i] << " ";
    }
    cout << ")\n\n Q = ( ";
    for(int i = 0; i < q.size(); i++) {
        cout << q[i] << " ";
    }
    cout << ")\n\n";

    vector<double> x(d.size());
```

```
    x[x.size() - 1] = q[q.size() - 1];

    for (int i = x.size() - 2; i >= 0; i--) {
        x[i] = x[i + 1] * p[i] + q[i];
    }

    cout << "answer\nX = ( ";
    for (int i = 0; i < x.size(); i++) {
        cout << x[i] << ' ';
    }
    cout << ")\n";

    return 0;
}
```

1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

Rewriting the system to equivalent form x = beta + alfa * x :

alfa =

0.000 -0.250 0.167 0.292

0.381 0.000 -0.190 0.095

-0.375 -0.375 0.000 -0.000

0.292 0.292 -0.208 0.000

beta = (-5.41667 6.61905 -5.25 -6.875)

Simple iterations (eps = 0.001) :

X = (-9 3.00001 -3.00001 -7.99998)

number of iterations = 12

Zeidel method (eps = 0.001) :

X = (-9 2.99999 -2.99999 -8.00001)

number of iterations = 7

```
#include <iostream>
```

```cpp
#include <vector>
#include "TMatrix.cpp"

using namespace std;

void ReadFromFile(vector<vector<double>>& vec, vector<double>& b, double&
eps) {

    double c;
    int n;
    cin >> n;
    vec.resize(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> c;
            vec[i].push_back(c);
        }
    }
    for (int i = 0; i < n; i++) {
        cin >> c;
        b.push_back(c);
    }
    cin >> eps;
    return;
}

void Jacobi(TMatrix& a, vector<double>& b, TMatrix& alf, vector<double>& bet)
{

    for (int i = 0; i < bet.size(); i++) {
        bet[i] = b[i] / a[i][i];
    }

    for (int i = 0; i < bet.size(); i++) {
        for (int j = 0; j < bet.size(); j++) {
            if (i == j) {
                alf[i][j] = 0;
            } else {
                alf[i][j] = -a[i][j] / a[i][i];
            }
        }
    }
    return;
}

double VecNorm(vector<double>& a) {
    double norm = 0;
    for (int i = 0; i < a.size(); i++) {
        norm += pow(a[i], 2);
    }
    return sqrt(norm);
}

vector<double> SimpleItSolution(TMatrix& alf, vector<double>& bet, double
eps, int& iter)  {

    vector<double> x = bet;
    double epsK = eps + 1;
    double matNorm = alf.SqNorm();

    while (epsK > eps) {
        vector<double> x1 = alf * x;
        for (int i = 0; i < x1.size(); i++) {
            x1[i] += bet[i];
```

```
        }

        vector<double> dif = x1;
        for (int i = 0; i < x.size(); i++) {
            dif[i] -= x[i];
        }
        epsK = VecNorm(dif) * matNorm / (1 - matNorm);
        x = x1;
        iter++;
    }
    return x;
}

TMatrix FindUpTriangle (TMatrix& alf) {
    TMatrix C(alf.Size());
    for (int i = 0; i < C.Size(); i++) {
        for (int j = 0; j < C.Size(); j++) {
            if (i > j) {
                C[i][j] = 0;
            } else {
                C[i][j] = alf[i][j];
            }
        }
    }
    return C;
}

vector<double> ZeidelSolution(TMatrix& alf, vector<double>& bet, double eps,
int& iter) {

    vector<double> x = bet;
    double epsK = eps + 1;
    double matNorm = alf.SqNorm();
    TMatrix C = FindUpTriangle(alf);
    double cNorm = C.SqNorm();

    while (epsK > eps) {
        vector<double> x1(x.size());

        for (int i = 0; i < x1.size(); i++) {
            for (int j = 0; j < x1.size(); j++) {
                if (i > j) {
                    x1[i] += alf[i][j] * x1[j];
                } else {
                    x1[i] += alf[i][j] * x[j];
                }
            }
            x1[i] += bet[i];
        }

        vector<double> dif = x1;
        for (int i = 0; i < x.size(); i++) {
            dif[i] -= x[i];
        }

        epsK = VecNorm(dif) * cNorm / (1 - matNorm);
        x = x1;
        iter++;

    }

    return x;
}
```

```
int main() {

    double eps;
    int iter = 0;
    vector<vector<double>> vec;
    vector<double> b;
    ReadFromFile(vec, b, eps);
    TMatrix a(vec);
    TMatrix alf (a.Size());

    vector<double> bet(b.size());
    cout << "Rewriting the system to equivalent form x = beta + alfa * x
:\nalfa = \n";
    Jacobi(a, b, alf, bet);
    Print(alf);
    cout << "beta = ( ";
    for (int i = 0; i < bet.size(); i++) {
        cout << bet[i] << ' ';
    }
    cout << ")\n\n";


    vector<double> answer = SimpleItSolution(alf, bet, eps, iter);
    cout << "Simple iterations (eps = " << eps << ") :\n\n";

    cout << "X = ( ";
    for (int i = 0; i < answer.size(); i++) {
        cout << answer[i] << ' ';
    }
    cout << ")\n number of iterations = " << iter << "\n";
    iter = 0;
    cout << "-----------------------------\n";

    answer = ZeidelSolution(alf, bet, eps, iter);
    cout << "Zeidel method (eps = " << eps << ") :\n\n";

    cout << "X = ( ";
    for (int i = 0; i < answer.size(); i++) {
        cout << answer[i] << ' ';
    }
    cout << ")\nnumber of iterations = " << iter << "\n";

    return 0;
}
```

1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

eigenvalues:

l1 = 11.8956

l2 = 7.2349

l3 = -12.1305

eigenvectors:

x1 x2 x3

0.937 0.045 0.347

-0.252 0.774 0.581

-0.242 -0.632 0.737

number of iterations: 4 (eps = 0.01)

Check: Ax == lx

(11.146 -3.00088 -2.87537) (11.1445 -3.00337 -2.87856)

(0.328736 5.59955 -4.56955) (0.328754 5.59955 -4.56955)

(-4.20223 -7.04555 -8.93597) (-4.20627 -7.04446 -8.93492)

```
#include <iostream>
#include <vector>
#include "TMatrix.cpp"

using namespace std;

void ReadFromFile(vector<vector<double>>& vec, double& eps) {

    double c;
    int n;
    cin >> n;
    vec.resize(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> c;
            vec[i].push_back(c);
        }
    }
    cin >> eps;
    return;
}

pair<int, int> MaxInd(TMatrix &a) {
    double value = a[0][1];
    int l = 0, k = 1;
    for (int i = 0; i < a.Size(); i++) {
        for (int j = i + 1; j < a.Size(); j++) {
            if (abs(a[i][j]) > value) {
                value = abs(a[i][j]);
                l = i;
                k = j;
            }
```

```
            }
        }
        return make_pair(l, k);
    }

    TMatrix FindU(pair<int, int> ind, TMatrix& a) {
        TMatrix U(a.Size());
        for (int i = 0; i < a.Size(); i++) {
            U[i][i] = 1;
        }

        int i = ind.first;
        int j = ind.second;

        double phi = 0;
        if (a[i][i] != a[j][j]) {
            phi = 0.5 * atan( 2 * a[i][j] / (a[i][i] - a[j][j]));
        } else {
            phi = M_PI / 4;
        }

        U[i][i] = cos(phi);
        U[i][j] = -sin(phi);
        U[j][i] = sin(phi);
        U[j][j] = cos(phi);
        return U;
    }

    double SqSum(TMatrix& a) {
        double norm = 0;
        for (int i = 0; i < a.Size(); i++) {
            for (int j = i + 1; j < a.Size(); j++) {
                norm += pow(a[i][j], 2);
            }
        }
        return sqrt(norm);
    }

    TMatrix RotationMeth(TMatrix &a, double eps, vector<double>& v, int& iter) {
        double epsk = eps + 1;
        TMatrix A = a;

        TMatrix Ucount(a.Size());
        for (int i = 0; i < Ucount.Size(); i++) {
            Ucount[i][i] = 1;
        }

        while (epsk > eps) {
            pair<int, int> indxs = MaxInd(A);
            TMatrix U = FindU(indxs, A);
            Ucount = Ucount * U;
            A = U.Transp() * A * U;
            epsk = SqSum(A);
            iter++;
        }
        v.resize(A.Size());
        for (int i = 0; i < A.Size(); i++) {
            v[i] = A[i][i];
        }
        return Ucount;
    }

    int main() {
```

```
    double eps;
    int iter = 0;
    vector<vector<double>> vec;
    ReadFromFile(vec, eps);
    TMatrix A(vec);
    vector<double> eigenvalues;
    TMatrix res = RotationMeth(A, eps, eigenvalues, iter);

    cout << "eigenvalues:\n\n";
    for (int i = 0; i < eigenvalues.size(); i++) {
        cout << "l" << i + 1 << " = " << eigenvalues[i] << '\n';
    }
    cout << "\neigenvectors:\n\n";
    cout << "x1\tx2\tx3\n";
    Print(res);

    cout << "number of iterations: " << iter << " (eps = " << eps << ")\n\n";

    cout << "Check:\t\t\tAx\t==\tlx\n";
    vector<vector<double>> v(res.Size());
    for (int i = 0; i < res.Size(); i++) {
        v[0].push_back(res[i][0]);
        v[1].push_back(res[i][1]);
        v[2].push_back(res[i][2]);
    }

    for (int k = 0; k < res.Size(); k ++) {
        vector<double> pr = A * v[k];
        cout << "( ";
        for (int i = 0; i < res.Size(); i++) {
            cout << pr[i] << ' ';
        }
        cout << " )  (";
        for (int i = 0; i < res.Size(); i++) {
            cout << v[k][i] * eigenvalues[k] << ' ';

        }
        cout << " )\n";
    }
    return 0;
}
```

1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

input matrix:

1.000 5.000 -6.000

9.000 -7.000 -9.000

6.000 -1.000 -9.000

example of decomposition:

Q =

-0.092 0.873 -0.479

-0.829 -0.334 -0.450

-0.552 0.355 0.754

R =

-10.863 5.892 12.980

-0.000 6.347 -5.431

0.000 0.000 0.131

Q * R =

1.000 5.000 -6.000

9.000 -7.000 -9.000

6.000 -1.000 -9.000

eigenvalues by QR (eps = 0.01):

l1 = -12.7863

l2 = -1.82935

l3 = -0.384385

```
#include <iostream>
#include <vector>
#include "TMatrix.cpp"

using namespace std;

const int ITER_MAX = 100;

void ReadFromFile(vector<vector<double>>& vec, double& eps) {

    double c;
    int n;
    cin >> n;
    vec.resize(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> c;
            vec[i].push_back(c);
        }
    }
    cin >> eps;
```

```
    return;
}

int sign(double a) {
    return a >= 0 ? 1 : -1;
}
/*
vector<double> VectorMult(const vector<double> &a, const vector<double> &b) {
    vector<double> res(a.size());
    for (int i = 0; i < res.size(); ++i) {
        res[i] = a[i] * b[i];
    }
    return res;
}

vector<double> VectorDiv(const vector<double> &a, const vector<double> &b) {
    vector<double> res(a.size());
    for (int i = 0; i < res.size(); ++i) {
        if (b[i] != 0) {
            res[i] = a[i] / b[i];
        } else {
            cerr << "0 element in division!\n";
        }
    }
    return res;
}
*/
double VecNorm(const vector<double>& a) {
    double norm = 0;
    for (int i = 0; i < a.size(); i++) {
        norm += pow(a[i], 2);
    }
    return sqrt(norm);
}

bool CheckEnd (TMatrix& A, double eps) {
    double sq1 = 0, lastsq1 = 10000, lastsq2 = 0,
     sq2 = 0;
    for (int i = 0; i < A.Size() - 1; i++) {
        sq1 = 0;
        sq2 = 0;
        for (int j = i + 1; j < A.Size(); ++j) {
            sq1 += pow(A[j][i], 2);
            if (j < A.Size() - 1) {
                sq2 += pow(A[j + 1][i], 2);
            }
        }
        if (sqrt(sq1) > eps && (sqrt(sq2) > eps || sqrt(lastsq1) > eps)) {
            return false;
        }
        lastsq1 = sq1;
        lastsq2 = sq2;
    }
    return true;
}



vector<pair<double, double>> solveQv(double a, double b , double c) {
    vector<pair<double, double>> answ(2);
    double d = pow(b, 2) - 4 * a * c;
    if (d < 0) {
        answ[0].first = -b/(2*a);
        answ[1].first = -b/(2*a);
```

```
        answ[0].second = sqrt(abs(d))/(2*a);
        answ[1].second = -sqrt(abs(d))/(2*a);
    } else {
        cerr << "no complex solution\n";
    }
    return answ;
}


TMatrix GetH(vector<double>& v) {
    TMatrix E;
    E.GetE(v.size());
    TMatrix mat(v.size());
    for (int i = 0; i < v.size(); ++i) {
        for (int j = 0; j < v.size(); ++j) {
            mat[i][j] = v[i] * v[j];
        }
    }

    double scal = 0;
    for (int i = 0; i < v.size(); ++i) {
        scal += v[i] * v[i];
    }
    return E -  mat / scal * 2.0;
}


TMatrix HouseHolder(TMatrix& A, TMatrix& Q) {
    TMatrix H(A.Size());
    TMatrix A0 = A;
    vector<double> v(A.Size());

    for (int i = 0; i < v.size() - 1; ++i) {
        for(int k = 0; k < v.size(); ++k) {
            if (k < i) {
                v[k] = 0;
            } else if (k == i) {
                double sq = 0;
                for (int j = k; j < v.size(); ++j) {
                    sq += pow(A0[j][k], 2);
                }
                sq = sqrt(sq);
                v[k] = A0[k][k]+ sign(A0[k][k]) * sq;
            } else {
                v[k] = A0[k][i];
            }
        }
        H = GetH(v);
        A0 = H * A0;
        Q = Q * H;
    }
    return A0;
}


void QRMethod (TMatrix &A0, double& eps) {
    int iter = 0;
    TMatrix Q;
    TMatrix A = A0;
    while (iter < ITER_MAX) {
        Q.GetE(A0.Size());
        A = HouseHolder(A, Q);
        A = A * Q;
    //  Print(A);
```

```
        iter++;
        if (CheckEnd(A, eps)) {
            break;
        }

    }

    //Print(A);
    for (int i = 0; i < A.Size(); ++i) {
        double sq1 = 0;
        for (int j = i + 1; j < A.Size(); ++j) {
            sq1 += pow(A[j][i], 2);
        }
        if (sqrt(sq1) <= eps) {
            cout << "l" << i + 1 << " = " << A[i][i] << '\n';
        } else {
            vector<pair<double, double>> ans = solveQv(1.0, -
A[i][i]*A[i+1][i+1], -A[i][i+1]*A[i+1][i]);
            cout << "l" << i+1 << " = " << ans[0].first << " + " <<
ans[0].second << "i\n";
            cout << "l" << i+2 << " = " << ans[1].first << " - " <<
abs(ans[1].second) << "i\n";
            ++i;
        }
    }
    return;
}

int main() {

    double eps;
    vector<vector<double>> vec;
    ReadFromFile(vec, eps);
    TMatrix A(vec);
    cout << "input matrix: \n";
    Print(A);
    cout << "example of decomposition:\n";
    TMatrix Q;
    Q.GetE(A.Size());
    TMatrix R = HouseHolder(A, Q);
    cout << " Q = \n";
    Print(Q);
    cout << "R =\n";
    Print(R);
    cout << "Q * R = \n";
    R = Q * R;
    Print(R);
    cout << "eigenvalues by QR (eps = " << eps << "):\n";
    QRMethod(A, eps);
}
```

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

eps = 0.001

start segment [1.0 1.5]

Simple Itarations:

x0 = 1.25

x = 1.13235

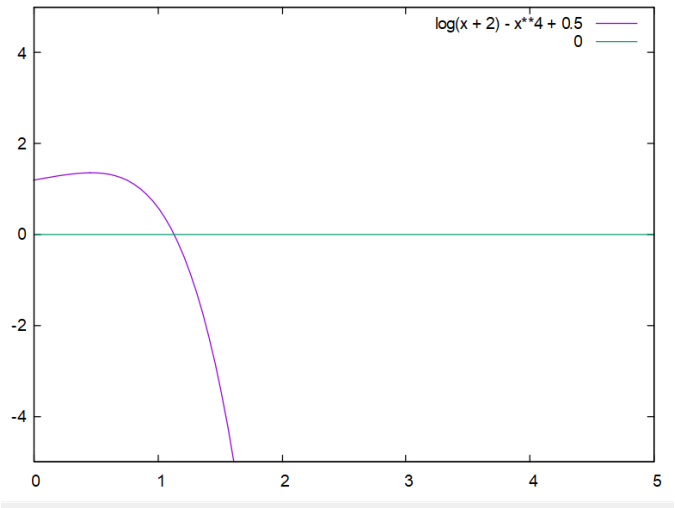
number of iterations: 10

Newton:

x0 = 1.5

x = 1.13193

number of iterations: 4




```
#include <iostream>
#include <cmath>
#include "gnuplot.h"

using namespace std;

int sign(double x) {
    return x < 0 ? -1 : 1;
}

double f(double x) {
    return log(x + 2) - pow(x, 4) + 0.5;
}

double phi(double x, double lambda) {
    return x - lambda * f(x);
}

double f1(double x) {
    return 1/(x + 2) - 4 * pow(x, 3);
}
```

```
double phi1(double x, double lambda) {
    return 1 - lambda * f1(x);
}

double f2(double x) {
    return 1/pow(x + 2, 2) - 12 * pow(x, 2);
}

double Newton(double a, double b, double eps, int& iter) {
    double epsk = eps + 1;

    double x = f(a)* f2(a) > 0 ? a : b;

    if (f(x) * f2(x) <= 0) {
        cerr << "f(x) * f''(x) <= 0\n";
        exit(1);
    }

    cout << "x0 = " << x << endl;
    double x1;
    while (epsk > eps) {
        x1 = x - f(x) / f1(x);
        epsk = abs(x1 - x);
        x = x1;
        iter++;
    }
    return x;
}

double SimpleIt(double a, double b, double eps, int& iter) {

    double x1, x = (a + b) / 2;
    double ex = sign(f1(a));
    for (double i = a; i <= b; i += 0.1) {
        if (sign(f1(i)) != ex) {
            cerr << "sign f' != const\n";
            exit(2);
        }
    }
    double ex2 = sign(f2(a));
    for (double i = a; i <= b; i += 0.1) {
        if (sign(f2(i)) != ex2) {
            cerr << "sign f'' != const\n";
            exit(3);
        }
    }
    double maxF2 = max(abs(f1(a)), abs(f1(b)));
    double lambda = ex / maxF2;
    double q = max(phi1(a, lambda), phi1(b, lambda));
    double epsk = eps + 1;
    while (epsk > eps) {
        x1 = phi(x, lambda);
        epsk = q / (1 - q) * abs(x1 - x);
        x = x1;
        iter++;
    }
    return x;
}

int main() {

    double eps;
    int iter = 0;
```

```
    cin >> eps;
    cout << "eps = " << eps << endl << endl;
    double x1 = SimpleIt(1.0, 1.5, eps, iter);
    cout << "start segment [1.0 1.5]\n";
    cout << "Simple Itarations:\nx0 = 1.25\nx = " << x1 << "\nnumber of
iterations: " << iter << "\n\n";
    iter = 0;
    cout << "Newton:\n";
    double x = Newton(1.0, 1.5, eps, iter);
    cout << "x = " << x << "\nnumber of iterations: " << iter << "\n\n";



    Gnuplot plot;
    plot("set xrange [0:+5]");
    plot("set yrange [-5:+5]");
    plot("plot log(x + 2) - x**4 + 0.5, 0");
    std::cin.get();
    return 0;
}
```

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

eps = 0.001

X0 = [0.75 0.75]

Simple Itarations:

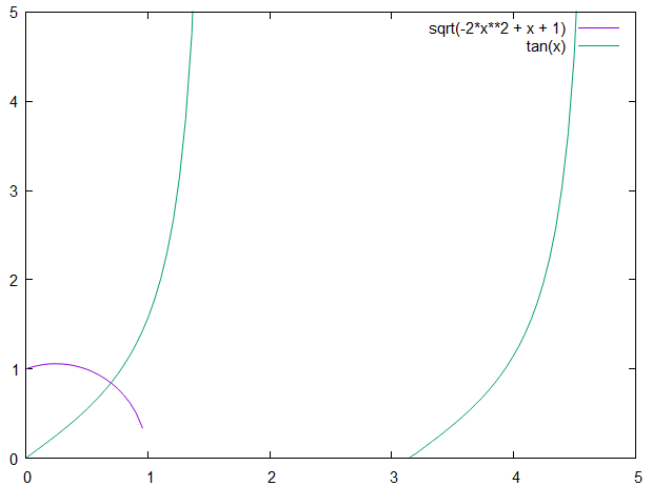
x = 0.70223 0.846191

number of iterations: 7

Newton:

x = 0.702501 0.845865

number of iterations: 10




```cpp
#include <iostream>
#include <cmath>
#include "gnuplot.h"
#include "..\lab1\TMatrix.cpp"

using namespace std;

double f(double x1, double x2) {
    return 2 * pow(x1,2) - x1 + pow(x2,2) - 1;
}

double g(double x1, double x2) {
    return x2 - tan(x1);
}

vector<double> phi(vector<double> x, TMatrix J1) {
    vector<double> F(2);
    F[0] = f(x[0], x[1]);
    F[1] = g(x[0], x[1]);
    vector<double> sub = J1 * F;
    for (int i = 0; i < x.size(); i++) {
        x[i] -= sub[i];
    }
    return x;
}


double f11(double x1, double x2) {
    return 4*x1 - 1;
}

double f12(double x1, double x2) {
    return 2*x2;
}

double g11(double x1, double x2) {
    return -1/(1 + pow(x1,2));
}

double g12(double x1, double x2) {
    return 1.0;
}

double phi1Max(double x1, double x2, TMatrix J1) {
    vector<double> x(2);
    vector<double> phi1(2);
    vector<double> phi2(2);
```

```
    x[0] = 1;
    x[1] = x2;
    vector<double> F(2);
    F[0] = f11(x1, x2);
    F[1] = g11(x1, x2);
    vector<double> sub = J1 * F;
    for (int i = 0; i < x.size(); i++) {
            phi1[i] = x[i] - sub[i];
        }

    x[0] = x1;
    x[1] = 1;
    F[0] = f12(x1, x2);
    F[1] = g12(x1, x2);
    sub = J1 * F;
    for (int i = 0; i < x.size(); i++) {
            phi2[i] = x[i] - sub[i];
        }
    double m1 = max(abs(phi1[0]) + abs(phi2[0]), abs(phi1[1]) +
abs(phi2[1]));

    return m1;
}

TMatrix GetJ (double x1, double x2) {
    TMatrix J(2);
    J[0][0] = f11(x1, x2);
    J[0][1] = f12(x1, x2);
    J[1][0] = g11(x1, x2);
    J[1][1] = g12(x1, x2);
    return J;
}


vector<double> Newton(double a1, double b1, double a2, double b2, double eps,
int& iter) {
    double epsk = eps + 1;

    vector<double> X;

    vector<double> b(2);
    vector<double> delta;
    X.push_back((a1 + b1) / 2);
    X.push_back((a2 + b2) / 2);

    while (epsk > eps){
        //cout << X[0] << " " << X[1] << '\n';
        TMatrix J = GetJ(X[0], X[1]);
        b[0] = -f(X[0], X[1]);
        b[1] = -g(X[0], X[1]);

        TMatrix U(J.Size()), L(J.Size());
        LU(J, L, U);
        delta = solveOfSystem(L, U, b);

        for (int i = 0; i < delta.size(); ++i) {
            X[i] += delta[i];
        }

        abs(delta[0]) > abs(delta[1]) ? epsk = abs(delta[0]) : epsk =
abs(delta[1]);
        iter++;
    }
    return X;
```

```
}

vector<double> SimpleIt(double a1, double a2, double eps, int& iter) {

    double epsk = eps + 1;
    TMatrix J = GetJ(a1, a2);
    TMatrix U(J.Size()), L(J.Size());
    TMatrix J1 = reverse2x2(J);
    vector<double> v(2);
    v[0] = a1;
    v[1] = a2;
    double q = phi1Max(a1,a2,J1);
    if (q > 1) {
        cerr << "q > 1\n";
    }
    while (epsk > eps) {
        vector<double> p = phi(v, J1);
        epsk = q/(1-q) *max(abs(p[0] - v[0]), abs(p[1] - v[1]));
        v = p;
        iter++;
    }
    return v;
}

int main() {

    double eps;
    int iter = 0;
    cin >> eps;
    cout << "eps = " << eps << endl << endl;
    vector<double> x = SimpleIt(0.75, 0.75, eps, iter);
    cout << "X0 = [0.75 0.75]\n";
    cout << "Simple Itarations:\nx = " << x[0] << "  " << x[1] << "\nnumber
of iterations: " << iter << "\n\n";
    iter = 0;
    cout << "Newton:\n";
    x = Newton(0.5, 1, 0.5, 1, eps, iter);
    cout << "x = " << x[0] << " " << x[1] << "\nnumber of iterations: " <<
iter << "\n\n";



    Gnuplot plot;
    plot("set xrange [0:+5]");
    plot("set yrange [0:+5]");
    plot("plot sqrt(-2*x**2 + x + 1), tan(x)");
    std::cin.get();
    return 0;
}
```

3.1. Используя таблицу значений Yi функции y = f x)(, вычисленных в точках X , i = 0,...,3 i построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки {X , . Yii } Вычислить значение погрешности интерполяции в точке * X .

Lagrange method:

i	xi	fi	w(xi)	fi/w(xi)
0	0.1	10	-0.384	-26.0417
1	0.5	2	0.128	15.625

2 0.9 1.11111 -0.128 -8.68056

3 1.3 0.769231 0.384 2.00321

lagrange polynomial:

L3(x) = -26.0417(x - 0.5)(x - 0.9)(x - 1.3)

+ 15.625(x - 0.1)(x - 0.9)(x - 1.3)

-8.68056(x - 0.1)(x - 0.5)(x - 1.3)

+ 2.00321(x - 0.1)(x - 0.5)(x - 0.9)

X* = 0.8

L3(0.8) = 1.02564

F(0.8) = 1.25

absolute error: 0.224359

Newton method:

f1: 10 2 1.11111 0.769231

f2: -20 -2.22222 -0.854701

f3: 22.2222 1.7094

f4: -17.094

newton polynomial:

N3(x) = 10 + (x - 0.1) * -20

+ (x - 0.1)(x - 0.5) * 22.2222

+ (x - 0.1)(x - 0.5)(x - 0.9) * -17.094

X* = 0.8

N3(0.8) = 1.02564

F(0.8) = 1.25

absolute error: 0.224359

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


using namespace std;

vector<double> ReadX () {
    int num;
    cin >> num;
    vector<double> v(num);
    for (int i = 0; i < num; i++) {
        cin >> v[i];
    }
    return v;
}

double f(double x) {
    return 1 / x;
}

double f(double x1, double x2) {
    return (f(x1) - f(x2)) / (x1 - x2);
}

double f(double x1, double x2, double x3) {
    return (f(x1, x2) - f(x2, x3)) / (x1 - x3);
}

double f(double x1, double x2, double x3, double x4) {
    return (f(x1, x2, x3) - f(x2, x3, x4)) / (x1 - x4);
}

double NewtPol (double X, vector<vector<double>>& vals, vector<double>& x) {
    return vals[0][0] + (X - x[0]) * vals[1][0] + (X - x[0])*(X - x[1]) *
vals[2][0] +
        (X - x[0])*(X - x[1])*(X - x[2]) * vals[3][0];
}

double LagPol (double X, vector<double>& x, vector<double>& wi) {
    return f(x[0]) / wi[0] * (X - x[1]) * (X - x[2])*(X - x[3]) +
            f(x[1]) / wi[1] * (X - x[0])*(X - x[2])*(X - x[3]) +
            f(x[2]) / wi[2] * (X - x[0])*(X - x[1])*(X - x[3]) +
            f(x[3]) / wi[3] * (X - x[0])*(X - x[1])*(X - x[2]);
}

void Lagrange(vector<double> x) {
    vector<double> fi(x.size());
    vector<double> wi(x.size());

    for (int i = 0; i < x.size(); ++i) {
        fi[i] = f(x[i]);
        double count = 1;
        for (int j = 0; j < x.size(); ++j) {
            if (i != j) {
                count *= x[i] - x[j];
            }
        }
        wi[i] = count;
        count = 1;
    }
```

```
    cout << "i" << "\t" << "xi" << "\t" << "fi" << "\t" << "w(xi)"<< "\t" <<
"fi/w(xi)\n";
    for (int i = 0; i < x.size(); i++) {
        cout << i << "\t" << x[i] << "\t" << f(x[i]) << "\t" << wi[i] << "\t"
<< f(x[i]) / wi[i] << '\n';
    }
    cout << "\n lagrange polynomial:\n L" << x.size()-1 <<"(x) = "
    << f(x[0]) / wi[0] << "(x - " << x[1] << ")(x - " << x[2] << ")( x - " <<
x[3] << ") \n"
    << "\t+ " << f(x[1]) / wi[1] << "(x - " << x[0] << ")(x - " << x[2] <<
")(x - " << x[3] << ") \n"
    <<'\t' <<f(x[2]) / wi[2] << "(x - " << x[0] << ")(x - " << x[1] << ")(x -
" << x[3] << ") \n"
    << "\t+ " << f(x[3]) / wi[3] << "(x - " << x[0] << ")(x - " << x[1] <<
")(x - " << x[2] << ")\n\n";

    double xStar = 0.8;
    cout << "X* = " << xStar << '\n';
    double lg = LagPol(xStar, x, wi);
    double trueVal = f(xStar);

    cout << "L3(" << xStar << ") = " << lg << '\n';
    cout << "F(" << xStar << ") = " << trueVal << '\n';
    cout << "absolute error: " << abs(lg - trueVal) << "\n\n";
}

void Newton (vector<double> x) {
    vector<vector<double>> vals(4);
    for (int i = 0; i < x.size(); i++) {
        vals[0].push_back(f(x[i]));
    }
    for (int i = 1; i < x.size(); i++) {
        vals[1].push_back(f(x[i - 1], x[i]));
    }
    for (int i = 2; i < x.size(); i++) {
        vals[2].push_back(f(x[i - 2], x[i - 1], x[i]));
    }
    for (int i = 3; i < x.size(); i++) {
        vals[3].push_back(f(x[i - 3], x[i - 2], x[i - 1], x[i]));
    }


    for (int i = 0; i < vals.size(); i++) {
        cout << "f" << i+1 << ": ";
        for (int j = 0; j < vals[i].size(); j++) {
            cout << vals[i][j] << "  ";
        }
        cout << '\n';
    }
    cout << endl;

    cout << "\n newton polynomial:\n";
    cout << "N3(x) = " << vals[0][0] << " + (x - " << x[0] << ") * " <<
vals[1][0] << '\n'
    <<'\t' << " + (x - " << x[0] << ")(x - " << x[1] << ") * " << vals[2][0]
<< '\n'
    <<'\t' << " + (x - " << x[0] << ")(x - " << x[1] << ")(x - " << x[2] <<
") * " << vals[3][0] << "\n\n";

    double xStar = 0.8;
    cout << "X* = " << xStar << '\n';
    double np = NewtPol(xStar, vals, x);
    double trueVal = f(xStar);
    cout << "N3(" << xStar << ") = " << np << '\n';
```

```
    cout << "F(" << xStar << ") = " << trueVal << '\n';
    cout << "absolute error: " << abs(np - trueVal) << "\n";


}
int main() {
    vector<double> x = ReadX();
    cout << "Lagrange method:\n";
    Lagrange(x);
    cout << "Newton method:\n";
    Newton(x);

    return 0;
}
```

3.2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при 0 x = x и 4 x = x . Вычислить значение функции в точке ∗ x = X .

x	a	b	c	d
[0.10-0.50]	10.000000	-24.671396	0.000000	29.196222
[0.50-0.90]	2.000000	-10.657209	35.035467	-34.870173
[0.90-1.30]	1.111100	0.633481	-6.808741	7.720876
[1.30-1.70]	0.769230	-1.107491	2.456310	-2.046925

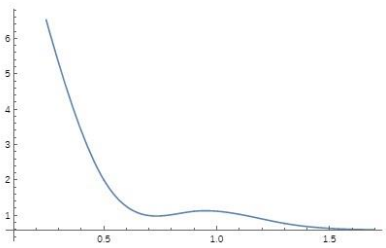
X* = 0.8

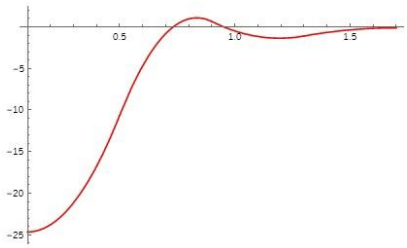
f(X*) = 2 + -10.6572(x - 0.5) + 35.0355(x - 0.5)^2 + -34.8702(x - 0.5)^3

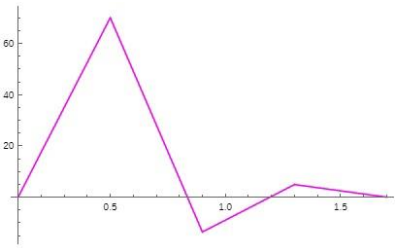
f(0.8) = 1.01453

CHECK:

	f(x)		f'(x)		f''(x)	
x1:	2.000000	2.000000	-10.657209	-10.657209	70.070933	70.070933
x2:	1.111100	1.111100	0.633481	0.633481	-13.617482	-13.617482
x3:	0.769230	0.769230	-1.107491	-1.107491	4.912621	4.912621








```
#include <iostream>
```

```cpp
#include <vector>
#include <cmath>
#include <iomanip>

using namespace std;

void ReadFromFile(vector<double>& x, vector<double>& f, double& xStar) {
    int c;
    double val;
    cin >> c;
    x.resize(c);
    f.resize(c);
    for (int i = 0; i < c; ++i) {
        cin >> val;
        x[i] = val;
    }
    for (int i = 0; i < c; ++i) {
        cin >> val;
        f[i] = val;
    }
    cin >> xStar;
    return;
}


double h(int i, vector<double> x) {
    return x[i] - x[i - 1];
}


vector<double> getC(vector<double>& x, vector<double>& f) {
    vector<double> a, b, c, d;
    a.push_back(0);
    b.push_back(2 * (h(1,x) + h(2,x)));
    c.push_back(h(2,x));

    a.push_back(h(2,x));
    b.push_back(2 * (h(2,x) + h(3,x)));
    c.push_back(h(4,x));

    a.push_back(h(3,x));
    b.push_back(2 * (h(3,x) + h(4,x)));
    c.push_back(0);

    for (int i = 2; i < x.size(); ++i) {
        d.push_back(3 * ((f[i] - f[i-1]) / h(i,x) - (f[i-1] - f[i-2]) / h(i-
1,x)));
    }

    vector<double> p(d.size());
    vector<double> q(d.size());

    p[0] = -c[0] / b[0];
    q[0] = d[0] / b[0];

    for (int i = 1; i < p.size(); i++) {
        p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
        q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
    }

    vector<double> answ(d.size());
    answ[answ.size() - 1] = q[q.size() - 1];

    for (int i = answ.size() - 2; i >= 0; i--) {
```

```
        answ[i] = answ[i + 1] * p[i] + q[i];
    }

    answ.push_back(0);
    for (int i = answ.size()-2; i >= 0; i--) {
        answ[i+1] = answ[i];
    }
    answ[0] = 0;
    return answ;
}

double fi(double xStar, int i, int xi, vector<vector<double>>& coefs,
vector<double>& x) {
    return  coefs[0][i] + coefs[1][i]*(xStar-x[xi]) + coefs[2][i]*pow((xStar-
x[xi]),2) +
            coefs[3][i]*pow((xStar-x[xi]),3);
}

double fi1(double xStar, int i, int xi, vector<vector<double>>& coefs,
vector<double>& x) {
    return  coefs[1][i] + coefs[2][i]*2*(xStar-x[xi]) +
            coefs[3][i]*3*pow((xStar-x[xi]),2);
}

double fi2(double xStar, int i, int xi, vector<vector<double>>& coefs,
vector<double>& x) {
    return coefs[2][i] * 2 + coefs[3][i]*6*(xStar-x[xi]);
}

void Spline(vector<double>& x, vector<double>& f, double xStar) {
    vector<double> c = getC(x, f);
    vector<double> a(c.size()), b(c.size()), d(c.size());
    for (int i = 1; i < c.size(); ++i) {
        a[i-1] = f[i-1];
        b[i-1] = (f[i]-f[i-1])/ h(i, x) - 1.0/3.0 * h(i, x) * (c[i] + 2*c[i-
1]);
        d[i-1] = (c[i] - c[i-1]) / (3 * h(i,x));
    }
    int n = b.size() - 1;
    b[n] = (f[n+1] - f[n])/h(n,x) - 2.0/3.0 * h(n,x)*c[n];
    d[n] = -c[n]/ (3*h(n,x));
    a[n] = f[n];

    vector<vector<double>> coefs;
    coefs.push_back(a);
    coefs.push_back(b);
    coefs.push_back(c);
    coefs.push_back(d);

    printf("x\t\ta\t\tb\t\tc\t\td\n");
    for (int i = 0; i < a.size(); i++) {
        printf("[%.2f-%.2f]\t%f\t%f\t%f\t%f\n",x[i], x[i+1], a[i], b[i],
c[i], d[i]);
    }

    cout << "\n X* = " << xStar << '\n';
    cout << "f(X*) = " << a[1] << " + " << b[1] << "(x - " << x[1] << ")"
        << " + " << c[1] << "(x - " << x[1] << ")^2"
        << " + " << d[1] << "(x - " << x[1] << ")^3\n";

    double res = fi(xStar, 1, 1, coefs, x);
    cout << "f(" << xStar << ") = " << res << '\n';

//check
```

```
    printf("\nCHECK:\n\tf(x)\t\t\tf'(x)\t\t\tf''(x)\n");
    for (int i = 1; i < 4; i++) {
        double res = fi(x[i], i, i, coefs, x);
        double res1 = fi(x[i], i-1, i-1, coefs, x);
        printf("x%d: %f %f\t", i ,res, res1);

        res = fi1(x[i], i, i, coefs, x);
        res1 = fi1(x[i], i-1, i-1, coefs, x);
        printf("%f %f\t",res, res1);


        res = fi2(x[i], i, i, coefs, x);
        res1 = fi2(x[i], i-1, i-1, coefs, x);
        printf("%f %f\n",res, res1);
    }
}

int main() {

    vector<double> x, f;
    double xStar;
    ReadFromFile(x, f, xStar);
    Spline(x,f, xStar);

    return 0;
}
```

3.3. Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

polynomial of the first degree:

F1(x) = 6.59192 + x * -3.7283

xi	F1(xi)
0.100000	6.219093
0.500000	4.727773
0.900000	3.236453
1.300000	1.745133
1.700000	0.253813
2.100000	-1.237507

sum of squared errors: 30.2541

polynomial of the second degree:

F2(x) = 10.0988 + x * -14.1074 + x^2 * 4.71779

xi F2(xi)

0.100000 8.735248

0.500000 4.224542

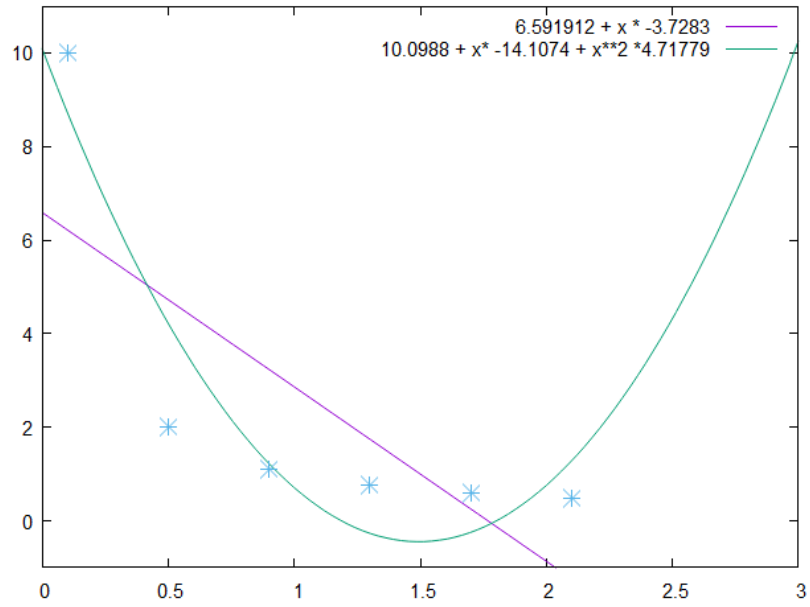
0.900000 1.223530

1.300000 -0.267790

1.700000 -0.249418

2.100000 1.278648

sum of squared errors: 8.98185




```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include "..\lab2\gnuplot.h"
#include "..\lab1\TMatrix.cpp"

using namespace std;

void ReadFromFile(vector<double>& x, vector<double>& y) {
    int c;
    double val;
    cin >> c;
    x.resize(c);
    y.resize(c);
    for (int i = 0; i < c; ++i) {
        cin >> val;
        x[i] = val;
    }
    for (int i = 0; i < c; ++i) {
        cin >> val;
        y[i] = val;
```

```
    }
    return;
}

double F1(vector<double>& a, double x) {
    return a[0] + a[1] * x;
}

double F2(vector<double>& a, double x) {
    return a[0] + a[1] * x + a[2] * pow(x, 2);
}

void MNK1 (vector<double>& x, vector<double>& y) {
    vector<double> b(2,0);
    double xSum = 0, xSqSum = 0;
    for (int i = 0; i < x.size(); i++) {
        b[0] += y[i];
        b[1] += x[i] * y[i];
        xSqSum += pow(x[i], 2);
        xSum += x[i];
    }

    TMatrix A(2), L(2), U(2);
    A[0][0] = x.size();
    A[0][1] = xSum;
    A[1][0] = xSum;
    A[1][1] = xSqSum;

    LU(A, L, U);
    for (int i = 0; i < U.swaps.size(); i++) {
        double tmp = b[U.swaps[i].first];
        b[U.swaps[i].first] = b[U.swaps[i].second];
        b[U.swaps[i].second] = tmp;
    }
    vector<double> a = solveOfSystem(L, U, b);

    cout << "polynomial of the first degree:\nF1(x) = " << a[0] << " + x * "
<< a[1] << '\n';
    printf("xi\t\tF1(xi)\n");
    for (int i = 0; i < x.size(); ++i) {
        printf("%f\t%f\n", x[i], F1(a, x[i]));
    }

    double sse = 0;
    for (int i = 0; i < y.size(); i++) {
        sse += pow(F1(a, x[i]) - y[i], 2);
    }
    cout << "sum of squared errors: " << sse << "\n\n";
}

void MNK2 (vector<double>& x, vector<double>& y) {
    vector<double> b(3,0);
    TMatrix A(3), L(3), U(3);
    A[0][0] = x.size();
    double xSum = 0, xSqSum = 0, xTrSum = 0, xQSum = 0;
    for (int i = 0; i < x.size(); i++) {
        xSum += x[i];
        xSqSum += pow(x[i], 2);
        xTrSum += pow(x[i],3);
        xQSum += pow(x[i],4);
        b[0] += y[i];
        b[1] += y[i]*x[i];
        b[2] += y[i]* pow(x[i],2);
    }
```

```
    A[0][1] = xSum;
    A[0][2] = xSqSum;
    A[1][0] = xSum;
    A[1][1] = xSqSum;
    A[1][2] = xTrSum;
    A[2][0] = xSqSum;
    A[2][1] = xTrSum;
    A[2][2] = xQSum;
    LU(A, L, U);
    for (int i = 0; i < U.swaps.size(); i++) {
        double tmp = b[U.swaps[i].first];
        b[U.swaps[i].first] = b[U.swaps[i].second];
        b[U.swaps[i].second] = tmp;
    }
    vector<double> a = solveOfSystem(L, U, b);

    cout << "polynomial of the second degree:\nF2(x) = " << a[0] << " + x * "
<< a[1] << " + x^2 * " << a[2] << '\n';
    printf("xi\t\tF2(xi)\n");
    for (int i = 0; i < x.size(); ++i) {
        printf("%f\t%f\n", x[i], F2(a, x[i]));
    }

    double sse = 0;
    for (int i = 0; i < y.size(); i++) {
        sse += pow(F2(a, x[i]) - y[i], 2);
    }
    cout << "sum of squared errors: " << sse << '\n';
}

int main() {
    vector<double> x, y;
    ReadFromFile(x, y);
    MNK1(x, y);
    MNK2(x, y);

    Gnuplot plot;
    plot("set xrange [0:+3]");
    plot("set yrange [-1:+11]");
    plot("set pointsize 2");
    plot("plot 6.591912 + x * -3.7283, 10.0988 + x* -14.1074 + x**2 *4.71779,
'test.dat' notitle with points");

    return 0;
}
```

3.4. Вычислить первую и вторую производную от таблично заданной функции = (i), = ,0 1,2,3,4 i i y f x в точке ∗ x = X .

X* = 2

first-order precision first derivative: 0.6666

second-order precision first derivative: 0.7333

second derivative: 0.2668

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


using namespace std;

void ReadFromFile(vector<double>& x, vector<double>& y, double& xStar) {
    int c;
    double val;
    cin >> c;
    x.resize(c);
    y.resize(c);
    for (int i = 0; i < c; ++i) {
        cin >> val;
        x[i] = val;
    }
    for (int i = 0; i < c; ++i) {
        cin >> val;
        y[i] = val;
    }

    cin >> val;
    xStar = val;
    return;
}

int findInd (const vector<double> x, double a) {
    for (int i = 0; i < x.size(); i++) {
        if (a <= x[i]){
            return i;
        }
    }
    cerr << "x out of range\n";
}

double derivativeA1 (const vector<double>& x, const vector<double>&y, double
xStar) {
    int i = findInd(x, xStar);
    if (i == 0) {
        i++;
    }
    return (y[i] - y[i-1]) / (x[i] - x[i-1]);
}

double derivativeA1Right (const vector<double>& x, const vector<double>&y,
double xStar) {
    int i = findInd(x, xStar);
    if (i == x.size()-1) {
        i--;
    }
    return (y[i+1] - y[i]) / (x[i+1] - x[i]);
}


double derivativeA2 (const vector<double>& x, const vector<double>&y, double
xStar) {
    int i = findInd(x, xStar);
    if (i == 0) {
        i++;
    }

    if (i == x.size()){
        cerr << "x out of range\n";
```

```
    }


    return (y[i] - y[i-1]) / (x[i] - x[i-1]) +
        ((y[i+1] - y[i]) / (x[i+1] - x[i]) - (y[i]-y[i-1]) / (x[i]-x[i-1])) /
            (x[i+1]-x[i-1]) * (2*xStar - x[i-1] - x[i]);
}

double derivative2 (const vector<double>& x, const vector<double>&y, double
xStar) {
    int i = findInd(x, xStar);
    if (i == 0) {
        i++;
    }

    if (i == x.size()){
        cerr << "x out of range\n";
    }


    return ((y[i+1] - y[i]) / (x[i+1] - x[i]) - (y[i]-y[i-1]) / (x[i]-x[i-
1])) /
            (x[i+1]-x[i-1]) * 2;
}


int main(){

    vector<double> x, y;
    double xStar;
    ReadFromFile(x, y, xStar);

    cout << "X* = " << xStar << '\n';
    cout << "first-order precision first derivative: " << derivativeA1(x, y,
xStar) << "\n\n";
    cout << "second-order precision first derivative: " << derivativeA2(x, y,
xStar) << "\n\n";
    cout << "second derivative: " << derivative2(x, y, xStar) << "\n\n";

    return 0;
}
```

3.5. Вычислить определенный интеграл ∫ = 1 0 X X F dxy , методами прямоугольников, трапеций, Симпсона с шагами 1 2 h , h . Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

step h1 = 1

rect 0.414533

trape 0.425023

simpson 0.418623

step h2 = 0.5

rect 0.417075

trape 0.419778

simpson 0.41803

Runge-Romberg method: inaccuracy

h2

rect -0.000847368

trape 0.00174845

simpson 3.95634e-005

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


using namespace std;

void ReadFromFile(double& x0, double& xk, double& h1, double& h2) {
    cin >> x0;
    cin >> xk;
    cin >> h1;
    cin >> h2;
    return;
}

double f(double x) {
    return 1 / sqrt((2*x+7)*(3*x+4));
}

double rect(vector<double>& x, double h1) {
    double res = 0;
    for (int i = 0; i < x.size() - 1; i++) {
        res += f((x[i] + x[i+1]) / 2);
    }
    return res*h1;
}

double trape(vector<double>& x, double h1) {
    double res = 0;
    for (int i = 0; i < x.size(); i++) {
        if (i == 0 || i == x.size() - 1) {
            res += f(x[i]) / 2;
        } else {
            res += f(x[i]);
        }
    }
    return res*h1;
}

double simpson(vector<double>& x, double h1) {
    double res = 0;
    for (int i = 0; i < x.size(); i++) {
        if (i == 0 || i == x.size() - 1) {
            res += f(x[i]);
        } else if (i % 2){
            res += f(x[i])*4;
        } else {
            res += f(x[i])*2;
```

```
        }
    }
    return res*h1/3;
}

double RRR(double fh, double fkh, double h, double kh, double p) {
    double k = kh/h;
    return (fkh- fh)/(pow(k,p)-1);
}

int main() {
    vector<double> x1,x2;
    double h1,h2,x0,xk;
    ReadFromFile(x0, xk, h1, h2);
    for (double i = x0; i <= xk; i+= h1) {
        x1.push_back(i);
    }
    for (double i = x0; i <= xk; i+= h2) {
        x2.push_back(i);
    }
    cout << "step h1 = " << h1 << '\n';
    cout << "rect " << rect(x1, h1) << '\n';
    cout << "trape " << trape(x1, h1) << '\n';
    cout << "simpson " << simpson(x1, h1) << '\n';

    cout << "\nstep h2 = " << h2 << '\n';
    cout << "rect " << rect(x2, h2) << '\n';
    cout << "trape " << trape(x2, h2) << '\n';
    cout << "simpson " << simpson(x2, h2) << '\n';

    cout << "\nRunge-Romberg method: inaccuracy\n";
    cout << "\t    h2\n";
    cout <<"rect\t"<<RRR(rect(x2,h2), rect(x1,h1), h2, h1, 2) << '\n';
    cout <<"trape\t"<< RRR(trape(x2,h2), trape(x1,h1), h2, h1, 2) << '\n';
    cout <<"simpson\t"<< RRR(simpson(x2,h2), simpson(x1,h1), h2, h1, 4) <<
'\n';
    return 0;
}
```

4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением

x^2*y'' + xy' - y - 3x^2 = 0

replacement: z = y'

x^2*z' + xz - y - 3x^2 = 0

Euler:

Y = [3 3.2 3.44 3.71463 4.02026 4.35436 4.71511 5.10116 5.5115 5.94536 6.40212]

Boosted Euler:

Y = [3 3.22 3.47469 3.76076 4.07581 4.41807 4.78622 5.17922 5.59627 6.03674 6.50012]

RungeKutta:

Y = [3 3.21909 3.47334 3.75923 4.07429 4.41667 4.785 5.17824 5.59556 6.03632 6.50001]

Adams:

Y = [3 3.21909 3.47334 3.75923 4.15064 4.59244 5.07381 5.6065 6.19636 6.84809 7.56824]

Runge-Romberg: inaccuracy

Euler: -1.76101

Boosted Euler: -2.16671

RungeKutta: -0.182718

Adams: -0.50455

absolute error:

Euler: 0.0150586

Boosted Euler: 1.89269e-005

RungeKutta: 8.97901e-007

Adams: 0.164345

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

using namespace std;

void ReadFromFile(double& y0, double& yx0, double& x0, double& x1, double& h)
{
    cin >> y0;
    cin >> yx0;
    cin >> x0;
    cin >> x1;
    cin >> h;
    return;
}

double f(double x, double y, double z) {
    return (-x*z + y + 3*pow(x,2)) / pow(x, 2);
}
```

```
double RRR(double fh, double fkh, double h, double kh, double p) {
    double k = kh/h;
    return (fkh- fh)/(pow(k,p)-1);
}

vector<double> Euler(vector<double>& x, double y0, double z0, double h) {
    vector<double> y(x.size()), z(x.size());
    y[0] = y0;
    z[0] = z0;

    for (int i = 1; i < y.size(); i++) {
        z[i] = z[i-1] + h*f(x[i-1], y[i-1], z[i-1]);
        y[i] = y[i-1] + h*z[i-1];
    }
    return y;
}

vector<double> BoostedEuler (vector<double>& x, double y0, double z0, double
h) {

    vector<double> y(x.size()), z(x.size());
    y[0] = y0;
    z[0] = z0;

    for (int i = 1; i < y.size(); i++) {

        double xhalh = x[i-1] + h/2;
        double yhalf = y[i-1] + h/2*z[i-1];
        double zhalf = z[i-1] + h/2*f(x[i-1], y[i-1], z[i-1]);

        z[i] = z[i-1] + h*f(xhalh, yhalf, zhalf);
        y[i] = y[i-1] + h*zhalf;
    }
    return y;

}

vector<double> RungeKutta(vector<double>& x, double y0, double z0, double h)
{
    vector<double> y(x.size()), z(x.size());
    y[0] = y0; z[0] = z0;
    vector<double> k1(x.size()), k2(x.size()), k3(x.size()), k4(x.size());
    vector<double> l1(x.size()), l2(x.size()), l3(x.size()), l4(x.size());

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

            z[i] = z[i-1] + (l1[i-1] + 2*l2[i-1] + 2*l3[i-1] + l4[i-1])/6.0;
            y[i] = y[i-1] + (k1[i-1] + 2*k2[i-1] + 2*k3[i-1] + k4[i-1])/6.0;
        }

        l1[i] = h*f(x[i], y[i], z[i]);
        k1[i] = h*z[i];

        l2[i] = h*f(x[i] + h/2.0, y[i] + k1[i]/2.0, z[i] + l1[i]/2.0);
        k2[i] = h*(z[i] + l1[i]/2.0);

        l3[i] = h*f(x[i] + h/2.0, y[i] + k2[i]/2.0, z[i] + l2[i]/2.0);
        k3[i] = h*(z[i] + l2[i]/2.0);

        l4[i] = h*f(x[i] + h, y[i] + k3[i], z[i] + l3[i]);
        k4[i] = h*(z[i] + l3[i]);
```

```
        double theta1 = abs((k2[i] - k3[i])/(k1[i]-k2[i]));
        double theta2 = abs((l2[i] - l3[i])/(l1[i]-l2[i]));
        //cout << theta1 << " " << theta2 << '\n';
        //if (theta1 > 0.1 || theta2 > 0.1) {
        //  h /= 2.0;
        //} else if (theta1 < 0.01 || theta2 < 0.01) {
    //      h *= 2.0;
    //  }
    }

    return y;
}

vector<double> Adams(vector<double>& x, vector<double>& yStart, double h) {
    vector<double> y(x.size());
    for (int i = 0; i < 4; i++) {
        y[i] = yStart[i];
    }

    for (int i = 4; i < y.size(); i++) {
        y[i] = y[i-1] + h/24*(55*y[i-1] - 59*y[i-2] + 37*y[i-3] - 9*y[i-4]);
    }

    return y;

}


int main() {
    vector<double> x1,x2;
    double h,y0,yx0,x0,xk;
    ReadFromFile(y0, yx0, x0, xk, h);
    for (double i = x0; i <= xk+h; i += h) {
        x1.push_back(i);
    }

    cout << "x^2*y'' + xy' - y - 3x^2 = 0\n replacement: z = y'\n";
    cout << "x^2*z'  + xz  - y - 3x^2 = 0\n\n";

    vector<double> answE = Euler(x1, y0, yx0, h);
    cout << "Euler:\n";
    cout << "Y = [ ";
    for (int i = 0; i < answE.size(); i++) {
        cout << answE[i] << " ";
    }
    cout << "]\n\n";

    vector<double> answBoost = BoostedEuler(x1, y0, yx0, h);
    cout << "Boosted Euler:\n";
    cout << "Y = [ ";
    for (int i = 0; i < answBoost.size(); i++) {
        cout << answBoost[i] << " ";
    }
    cout << "]\n\n";

    vector<double> answR = RungeKutta(x1, y0, yx0, h);
    cout << "RungeKutta:\n";
    cout << "Y = [ ";
    for (int i = 0; i < answR.size(); i++) {
        cout << answR[i] << " ";
    }
    cout << "]\n\n";

    vector<double> answA = Adams(x1, answR, h);
```

```
    cout << "Adams:\n";
    cout << "Y = [ ";
    for (int i = 0; i < answA.size(); i++) {
        cout << answA[i] << " ";
    }
    cout << "]\n";


    for (double i = x0; i <= xk+h; i+= h*2) {
        x2.push_back(i);
    }
    vector<double> answE2 =  Euler(x2, y0, yx0, 2*h);
    vector<double> answBoost2 =  Euler(x2, y0, yx0, 2*h);
    vector<double> answR2 =  RungeKutta(x2, y0, yx0, 2*h);
    vector<double> answA2 =  Adams(x2,answR, 2*h);

    cout << "\nRunge-Romberg: inaccuracy\n";
    cout <<"Euler:\t"<<RRR(answE[10], answE2[10], h, h*2, 1) << '\n';
    cout <<"Boosted Euler:\t"<<RRR(answBoost[10], answBoost2[10], h, h*2, 2)
<< '\n';
    cout <<"RungeKutta:\t"<<RRR(answR[10], answR2[10], h, h*2, 4) << '\n';
    cout <<"Adams:\t"<<RRR(answA[10], answA2[10], h, h*2, 4) << '\n';

    cout << "\nabsolute error:\n";
    cout << "Euler:\t" << abs(answE[10] - 6.5) / 6.5 << '\n';
    cout << "Boosted Euler:\t" << abs(answBoost[10] - 6.5) / 6.5 << '\n';
    cout << "RungeKutta:\t" << abs(answR[10] - 6.5) / 6.5 << '\n';
    cout << "Adams:\t" << abs(answA[10] - 6.5) / 6.5 << '\n';

    return 0;
}
```

4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Runge-Romberg: inaccuracy

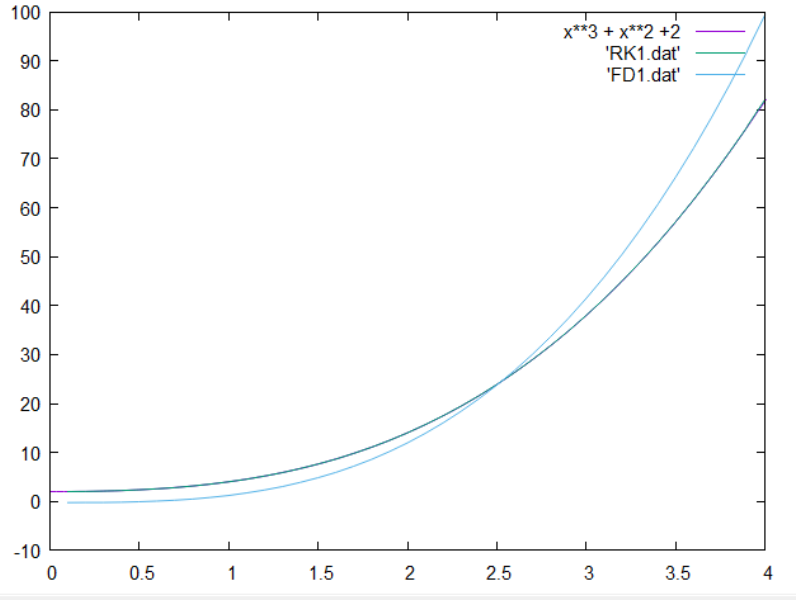
Shooting method: -0.604555

Finite Difference method: 0.187984

absolute error:

Shooting method: 0.000688429

Finite Difference method: 0.211859




```
#include <iostream>
#include <vector>
#include <cmath>
#include "..\lab2\gnuplot.h"

using namespace std;

void ReadFromFile(double& z0, double& kr2, double& x0, double& x1, double& h)
{
    cin >> z0;
    cin >> kr2;
    cin >> x0;
    cin >> x1;
    cin >> h;
    return;
}

double NewNu (double nu0, double nu1, double F0, double F1) {
    return nu1 - (nu1-nu0)/(F1-F0)*F1;
}

double f(double x, double y, double z) {
    return (4*(pow(x,2)+3)*z - 6*x*y) / x / (pow(x,2)+6);
}

double RRR(double fh, double fkh, double h, double kh, double p) {
    double k = kh/h;
    return (fkh- fh)/(pow(k,p)-1);
}

pair<vector<double>,vector<double>> RungeKutta(vector<double>& x, double y0,
double z0, double h) {
    vector<double> y(x.size()), z(x.size());
    y[0] = y0; z[0] = z0;
    vector<double> k1(x.size()), k2(x.size()), k3(x.size()), k4(x.size());
    vector<double> l1(x.size()), l2(x.size()), l3(x.size()), l4(x.size());

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

            z[i] = z[i-1] + (l1[i-1] + 2*l2[i-1] + 2*l3[i-1] + l4[i-1])/6.0;
            y[i] = y[i-1] + (k1[i-1] + 2*k2[i-1] + 2*k3[i-1] + k4[i-1])/6.0;
        }
```

```
        l1[i] = h*f(x[i], y[i], z[i]);
        k1[i] = h*z[i];

        l2[i] = h*f(x[i] + h/2.0, y[i] + k1[i]/2.0, z[i] + l1[i]/2.0);
        k2[i] = h*(z[i] + l1[i]/2.0);

        l3[i] = h*f(x[i] + h/2.0, y[i] + k2[i]/2.0, z[i] + l2[i]/2.0);
        k3[i] = h*(z[i] + l2[i]/2.0);

        l4[i] = h*f(x[i] + h, y[i] + k3[i], z[i] + l3[i]);
        k4[i] = h*(z[i] + l3[i]);

    }
    return make_pair(y,z);
}

double Fi(double nu, vector<double> y, vector<double> z, double kr) {
    return abs(y[y.size()-1] - z[z.size()-1] - kr);
}

double Shooting(vector<double> x, double z0, double h, double kr) {
    double oldNu = 0;
    double nNu = 5;
    double nu;
    double res = 100.0;
    while(res > 0.0001) {
        pair<vector<double>,vector<double>> yz1 = RungeKutta(x, oldNu, z0,
h);
        pair<vector<double>,vector<double>> yz2  = RungeKutta(x, nNu, z0, h);
        double F1 = Fi(oldNu, yz1.first, yz1.second, kr);
        double F2 = Fi(nNu, yz2.first, yz2.second, kr);
        nu = NewNu(oldNu, nNu, F1, F2);
        pair<vector<double>,vector<double>> yz3  = RungeKutta(x, nu, z0, h);
        res = Fi(nu, yz3.first, yz3.second, kr);
        oldNu = nNu;
        nNu=nu;

    }
    return nu;
}

double q(double x) {
    return 6/(pow(x,2)+6);
}
double p(double x) {
    return -4*(pow(x,2)+3)/x/(pow(x,2)+6);
}

double f(double x) {
    return 0.0;
}

vector<double> FiniteDifference(vector<double> x, double z0, double h, double
kr) {

    vector<double> a(x.size()), b(x.size()), c(x.size()), d(x.size());
    a[0] = 0;
    b[0] = - 1 + pow(h,2)*p(x[0]) - p(x[0])*h/2;
    c[0] = 1 + p(x[0])*h/2;
    d[0] = h*z0*(1-p(x[0])*h/2);
    for (int k = 1; k < x.size()-1; ++k) {
        a[k] = (1-p(x[k])*h/2);
        b[k] = (-2 + pow(h,2)*q(x[k]));
        c[k] = (1 + p(x[k])*h/2);
```

```
        d[k] = f(x[k])*pow(h,2);
    }
    a[x.size()-1] = 1;
    b[x.size()-1] = h-1;
    c[x.size()-1] = 0;
    d[x.size()-1] = kr*h;


    vector<double> p(d.size());
    vector<double> q(d.size());

    p[0] = -c[0] / b[0];
    q[0] = d[0] / b[0];

    for (int i = 1; i < p.size(); i++) {
        p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
        q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
    }

    vector<double> y(d.size());
    y[x.size() - 1] = q[q.size() - 1];

    for (int i = y.size() - 2; i >= 0; i--) {
        y[i] = y[i + 1] * p[i] + q[i];
    }
    return y;
}

int main () {
    vector<double> x1,x2;
    double h,z0,kr2,x0,xk;
    ReadFromFile(z0, kr2, x0, xk, h);
    for (double i = x0; i <= xk+h; i += h) {
        x1.push_back(i);
    }
    double y0 = Shooting(x1, z0, h, kr2);
    vector<double> answS1 = RungeKutta(x1, y0, z0, h).first;
    vector<double> answFD1 = FiniteDifference(x1, z0, h, kr2);

    for (double i = x0; i <= xk+h; i += h*2) {
        x2.push_back(i);
    }

    vector<double> answS2 = RungeKutta(x2, y0, z0, h*2).first;
    vector<double> answFD2 = FiniteDifference(x2, z0, h*2, kr2);

    cout << "\nRunge-Romberg: inaccuracy\n";
    cout << "Shooting method: " << RRR(answS1[answS1.size()-1],
answS2[answS2.size()-1], h, h*2, 4) << '\n';
    cout << "Finite Difference method: " << RRR(answFD1[answS1.size()-1],
answFD2[answS2.size()-1], h, h*2, 1) << '\n';
    cout << "\nabsolute error:\n";
    cout << "Shooting method: " << abs(answS1[answS1.size()-1] - 82.0) / 82.0
<< '\n';
    cout << "Finite Difference method: " << abs(answFD1[answS1.size()-1] -
82.0) / 82.0;

    Gnuplot plot;
    plot("set xrange [0:+4]");
    plot("plot x**3 + x**2 +2, 'RK1.dat'  with lines, 'FD1.dat'  with
lines");
    //plot("plot x**3 + x**2 +2, 'RK2.dat'  with lines, 'FD2.dat'  with
lines");
```

```
    return 0;
}
```