МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа № 1

Метод Ньютона решения систем нелинейных уравнений

Вариант 7

Выполнил

Ульяницкий Владимир

2 курс 3 группа

Преподаватель:

Бондарь И. В.

Минск, 2018

Постановка задачи

Имеется система нелинейных уравнений вида

$$A_0 x_0^i + \ldots + A_m x_m^i, i = 0, \ldots 2m+1. \quad (1)$$

Здесь $\{A_k\}_{k=0}^m, \{x_k\}_{k=0}^m$ — неизвестные величины, $\{g_i\}_{i=0}^{2m+1}$ — числовые коэффициенты.

Для варианта 7 $g_i = \int_{-1}^{1}(1-x)^5 (1+x)^3 x^i dx; m = 1.$

Задание:

- Реализовать метод Ньютона для решения системы.
- Провести вычислительный эксперимент: взяв несколько начальных приближений, при которых итерационный процесс сходится, найти решение системы с точностью 10^{-10}.
- Построить логарифмические диаграммы сходимости

Теория

Метод Ньютона

$$\boldsymbol{\varepsilon^k = x^* - x^k}$$

$$\boldsymbol{f(x^k + \varepsilon^k) = 0}$$

Разложим f в ряд Тейлора в окрестности x^k и оставим только линейную часть.

$$f(x^k) + \frac{\partial f(x^k)}{\partial x}\varepsilon^k \approx 0$$

Пусть $\Delta x^k \approx \varepsilon^k$.

$$\frac{\partial f(x^k)}{\partial x}\Delta x^k = -f(x^k)$$

Последнее выражение – система линейных алгебраических выражений для определения поправок Δx^k. Матрица этой системы – матрица Якоби – имеет вид

$$\frac{\partial f(x^k)}{\partial x} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ x_0 & A_0 & x_1 & A_1 \\ x_0^2 & 2A_0x_0 & x_1^2 & 2A_1x_1 \\ x_0^3 & 3A_0x_0^2 & x_1^3 & 3A_1x_1^2 \end{bmatrix}$$

Следующее приближение может быть найдено

$$x^{k+1} = x^k + \Delta x^k \Leftrightarrow x^{k+1} = x^k - \left(\frac{\partial f(x^k)}{\partial x}\right)^{-1} f(x^k),\ k = 0,1,\ldots$$

Итерационный процесс продолжается до тех пор, пока $\|x^{k+1} - x^k\| > \varepsilon$.

Поиск начального приближения

Из (1) можно получить

$$\begin{cases} A_0 = h(A_1) = g_0 - A_1 \\ f_0(A_0, A_1) = f_0(h(A_1), A_1) = A_0 + A_1 - g_0 = 0 \end{cases}; \quad (2)$$

$$\begin{cases} x_0 = t(x_1) = (g_1 - A_1 x_1)/(g_0 - A_1) \\ f_1(A_0, A_1, x_0, x_1) = f_1(h(A_1), A_1, t(x_1), x_1) = A_0 x_0 + A_1 x_1 - g_1 = 0 \end{cases}. \quad (3)$$

причем в (3) рассмотрим A_0, A_1 как константы

Используя метод деления отрезка пополам найдем начальное приближение. Необходимо найти такой отрезок, на котором функции f_0, f_1 непрерывны, монотонны и принимают значения разного знака на концах отрезка. Тогда на таком отрезке функция имеет ровно один корень, и в качестве начального приближения для дальнейшего решения можно взять некоторую точку этого отрезка.

Используем следующие начальные приближения A_0, x_0, A_1, x_1:

- [10, 30, 50, 60]
- [50, 60, 10, 30]
- [0.5, 0.1, 0.6, -0.5]
- [1000, 1, 1, 1000]
- [-1000000, 1, -1, 1000000]

Решение

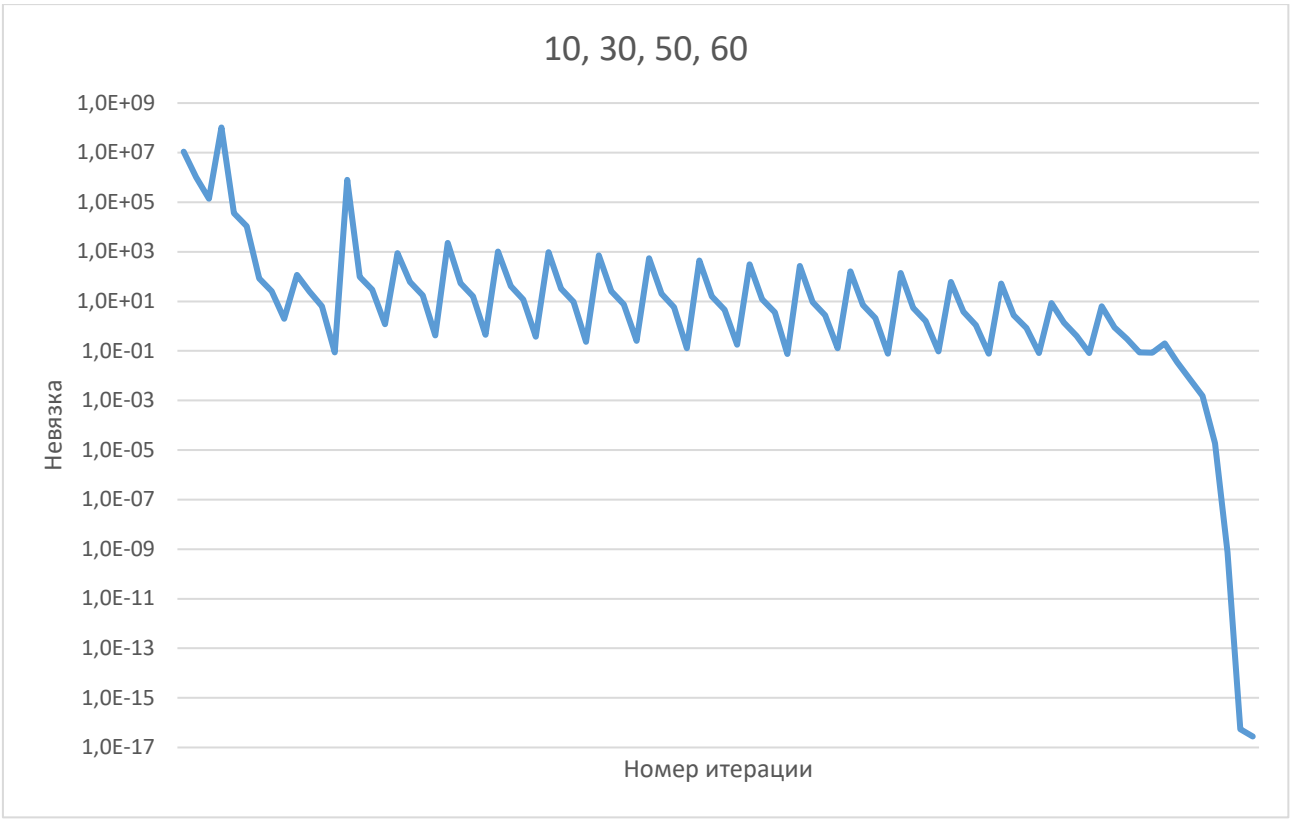
Вычислим значения g_i:

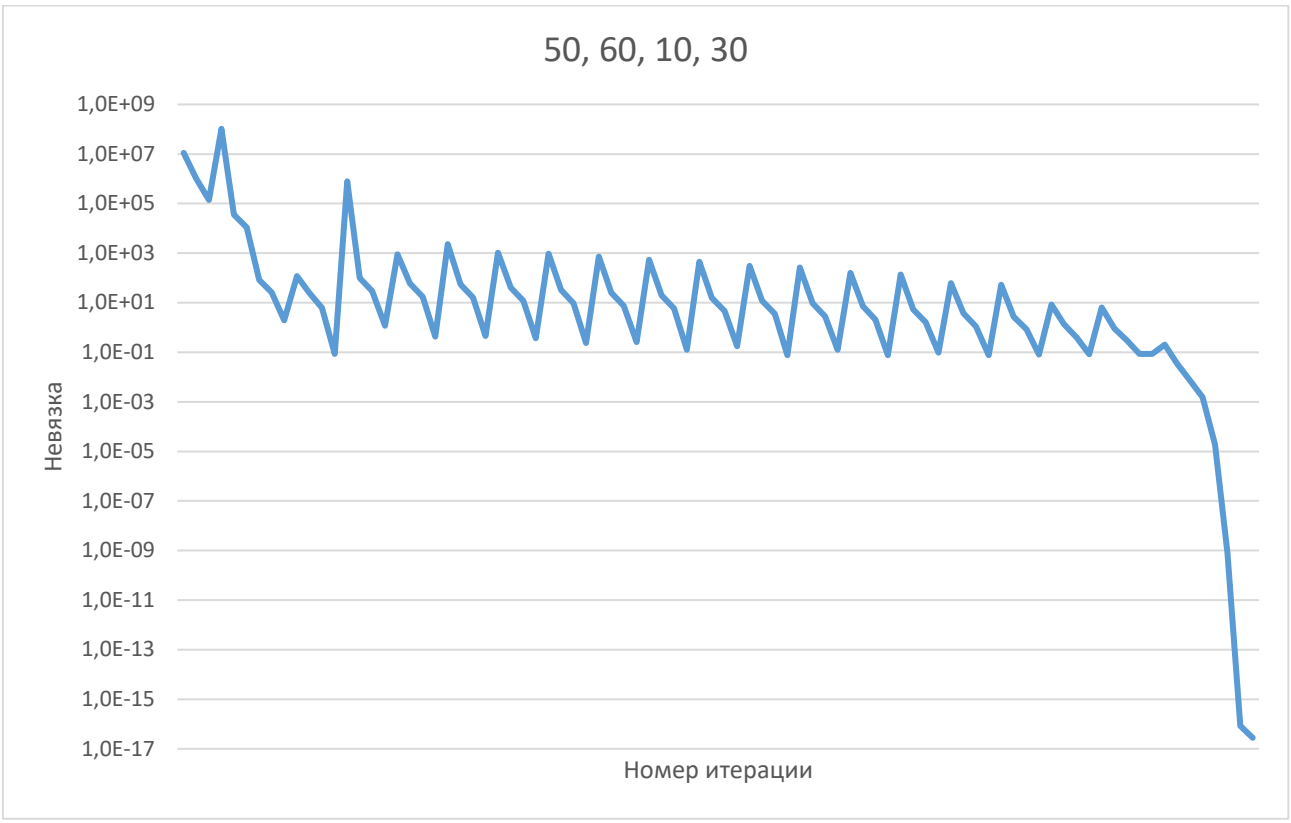
$$\int_{-1}^{1} (1-x)^5 (1+x)^3 x^i dx = \frac{96(195 + 573(-1)^i + 2i(10+i)(5 + (-1)^i(35 + i(10+i))))}{(1+i)(2+i)(3+i)(4+i)(6+i)(7+i)(8+i)(9+i)}$$

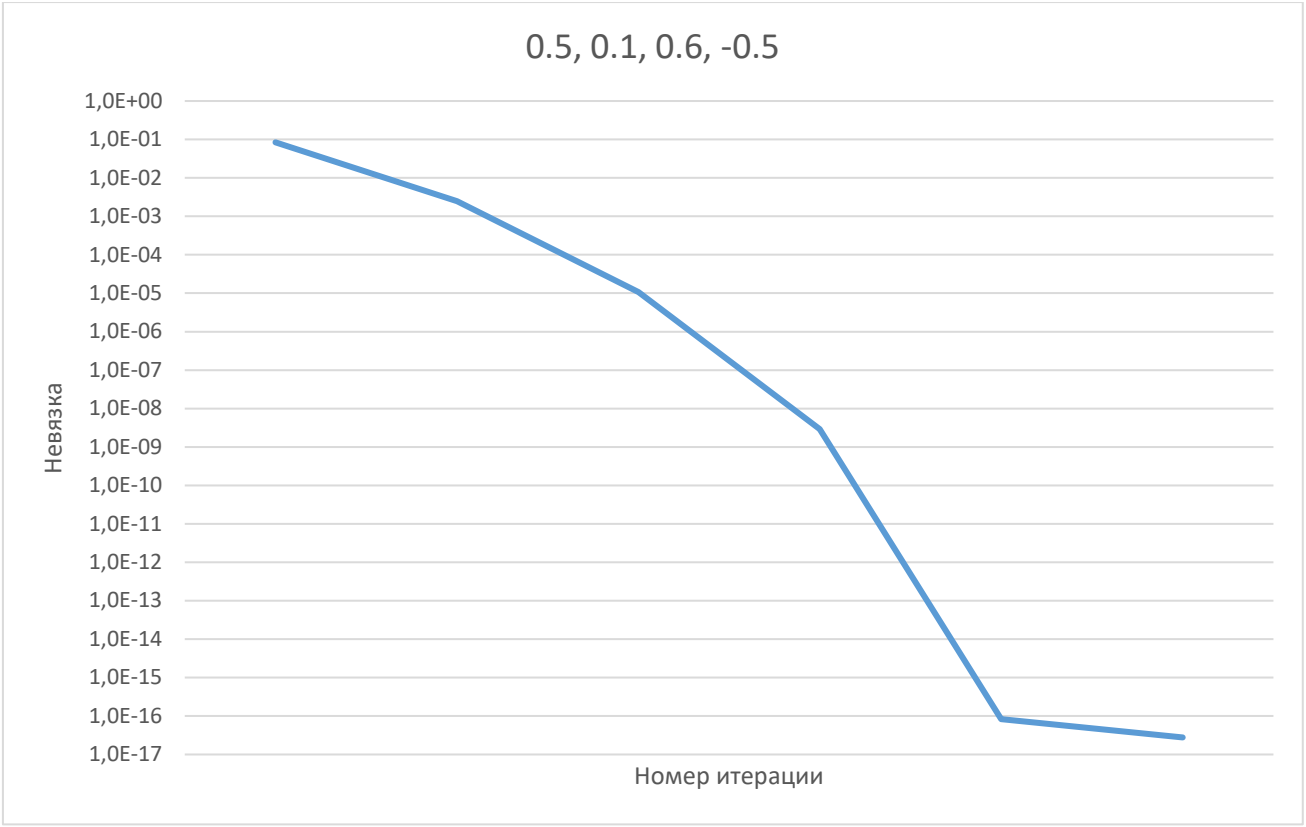
Используя полученные начальные приближения и числовые коэффициенты, решим систему (1) методом Ньютона.

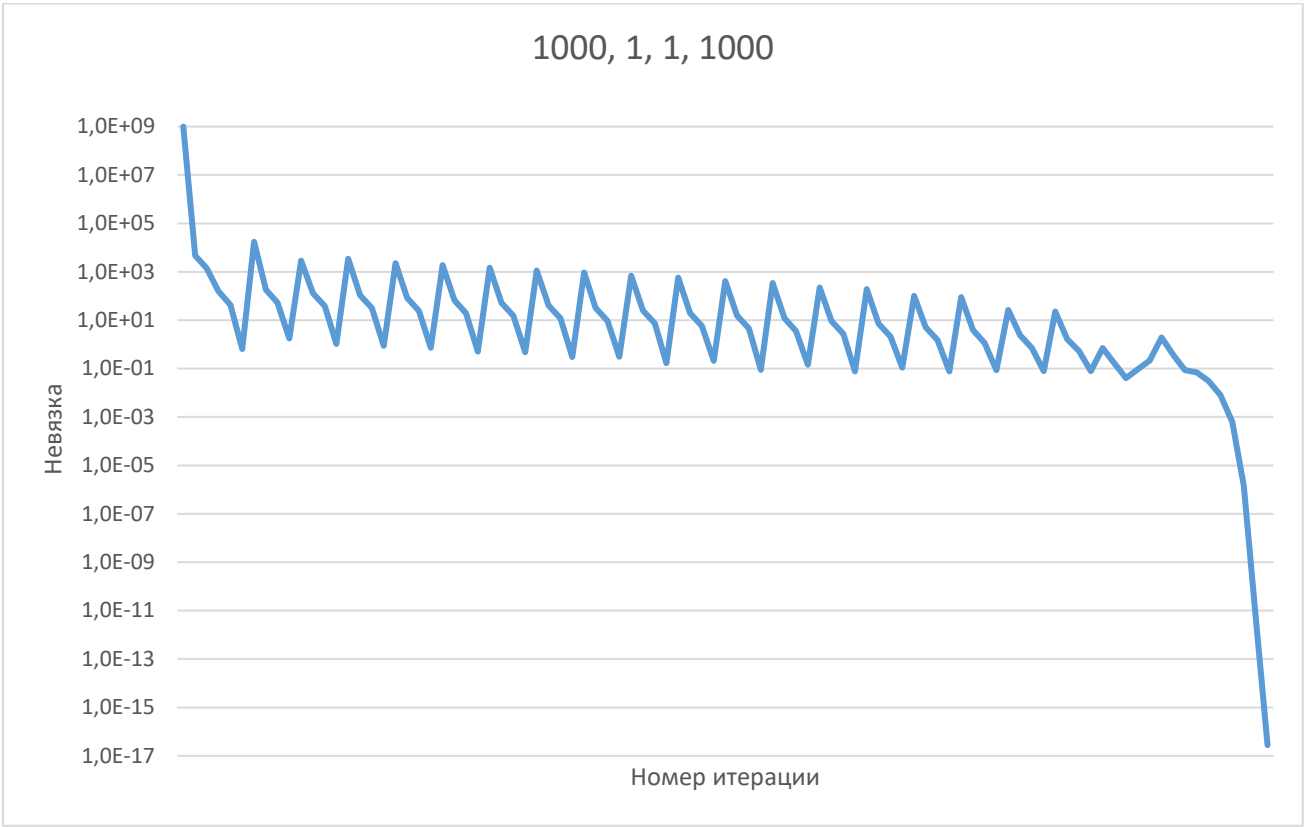
В зависимости он использованного начального приближения вектор решения $\mathrm{A}_0, \mathrm{x}_0, \mathrm{A}_1, \mathrm{x}_1$ принимает значения [0.450985456703, 0.130627528339, 0.564887559170, -0.463960861672] или [0.564887559170, -0.463960861672, 0.450985456703, 0.130627528339].

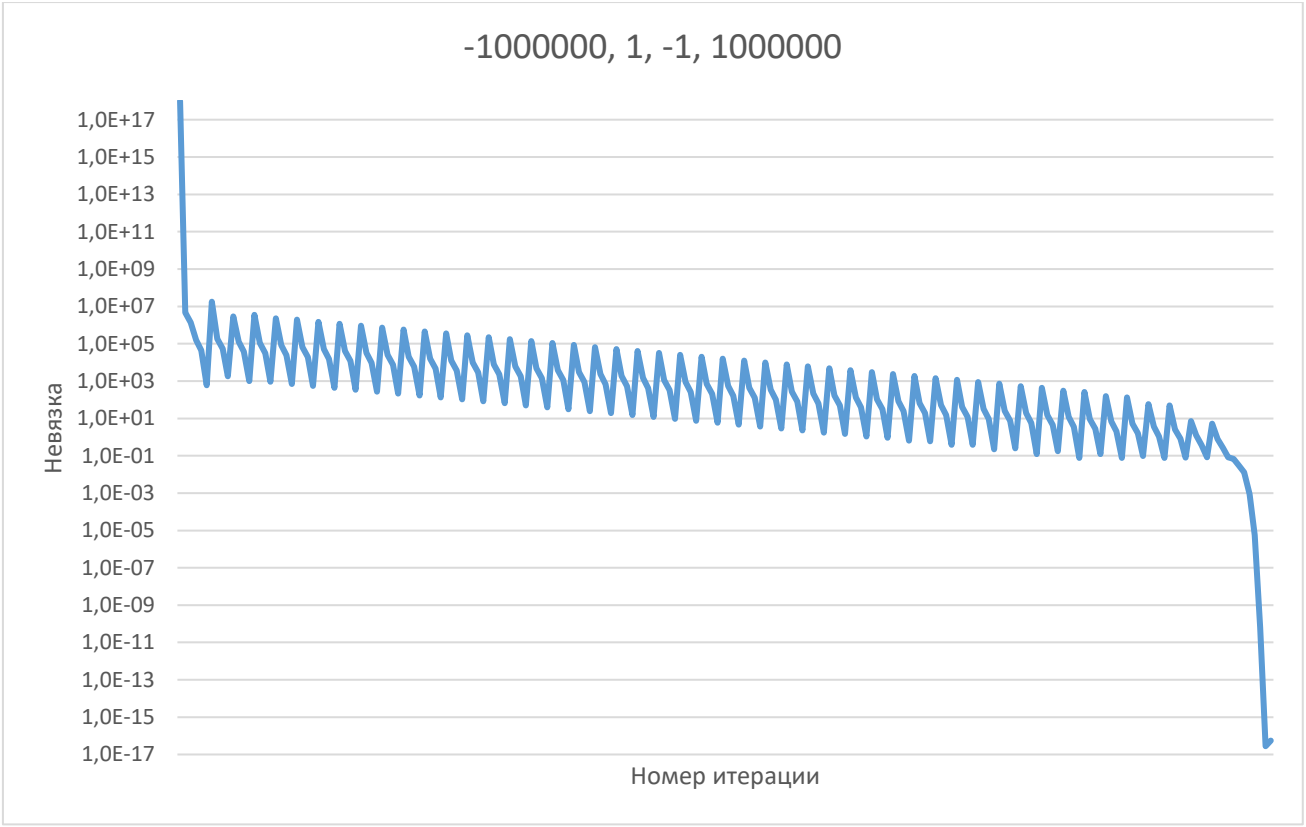
Диаграммы сходимости






0.5, 0.1, 0.6, -0.5
1,0E+00
1,0E-01
1,0E-02
1,0E-03
1,0E-04
1,0E-05
1,0E-06
1,0E-07
1,0E-08
1,0E-09
1,0E-10
1,0E-11
1,0E-12
1,0E-13
1,0E-14
1,0E-15
1,0E-16
1,0E-17
Невязка
Номер итерации


1000, 1, 1, 1000
1,0E+09
1,0E+07
1,0E+05
1,0E+03
1,0E+01
1,0E-01
1,0E-03
1,0E-05
1,0E-07
1,0E-09
1,0E-11
1,0E-13
1,0E-15
1,0E-17
Невязка
Номер итерации


-1000000, 1, -1, 1000000
Невязка
1,0E+17
1,0E+15
1,0E+13
1,0E+11
1,0E+09
1,0E+07
1,0E+05
1,0E+03
1,0E+01
1,0E-01
1,0E-03
1,0E-05
1,0E-07
1,0E-09
1,0E-11
1,0E-13
1,0E-15
1,0E-17
Номер итерации

Исходный код

```
/*Newton.cpp
C++14, Cygwin 2.11.2 compiler*/

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <string>

using namespace std;

double integral(int i)
{
    return 1.0 * (96 * (195 + 573 * ((i % 2 == 0) ? 1 : -1) +
                        2 * i * (10 + i) * (5 + ((i % 2 == 0) ? 1 : -1) * (35 + i
* (10 + i))))) /
           ((1 + i) * (2 + i) * (3 + i) * (4 + i) * (6 + i) * (7 + i) * (8 + i) *
(9 + i));
}

double determine(vector<vector<double>> &matrix, int size)
{
    int i, j;
    double det = 0;
    if (size == 1)
    {
        det = matrix[0][0];
    }
    else if (size == 2)
    {
        det = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
    }
    else
    {
        vector<vector<double>> temp(size - 1);
        for (i = 0; i < size; ++i)
        {
            for (j = 0; j < size - 1; ++j)
            {
                if (j < i)
                    temp[j] = matrix[j];
                else
                    temp[j] = matrix[j + 1];
            }
            det += (((i + j) % 2 == 0) ? 1.0 : -1.0) * determine(temp, size - 1)
* matrix[i][size - 1];
        }
    }
    return det;
}


vector<vector<double>> inv(vector<vector<double>> &matrix)
```

```
{
    vector<vector<double>> extended(matrix.size(), vector<double>(2 *
matrix.size(), 0));
    for (int i = 0; i < matrix.size(); i++)
    {
        extended[i][i + matrix.size()] = 1;
        for (int j = 0; j < matrix.size(); j++)
        {
            extended[i][j] = matrix[i][j];
        }
    }

    //directStep(extended);
    double koef;
    for (int i = 0; i < extended.size(); i++)
    {//от i-ой строки
        if (extended[i][i] == 0)
        {
            int j = 0;
            while ((j < extended.size()) && (extended[j][i] == 0))
            {
                j++;
            }
            for (int k = 0; k < 2 * extended.size(); k++)
            {
                extended[i][k] += extended[j][k];
            }
        }
        for (int j = 0; j < i; j++)
        {//отнимаем все предыдущие j-ые
            if (extended[i][j] != 0)
            {
                koef = extended[i][j] / extended[j][j];
                for (int k = j; k < 2 * extended.size(); k++)
                {
                    extended[i][k] -= extended[j][k] * koef;
                }
            }
        }
    }
    //reverseStep(extended);
    for (int i = extended.size() - 1; i >= 0; i--)
    {
        for (int j = i + 1; j < extended.size(); j++)
        {
            if (extended[i][j] != 0)
            {
                koef = extended[i][j] / extended[j][j];
                for (int k = j; k < 2 * extended.size(); k++)
                {
                    extended[i][k] -= extended[j][k] * koef;
                }
            }
        }
        for (int j = extended.size(); j < 2 * extended.size(); j++)
        {
            extended[i][j] /= extended[i][i];
        }
        extended[i][i] = 1;
    }
```

```
    vector<vector<double>> temp(matrix.size(), vector<double>(matrix.size(), 0));
    for (int i = 0; i < matrix.size(); i++)
    {
        for (int j = 0; j < matrix.size(); j++)
        {
            temp[i][j] = extended[i][j + matrix.size()];
        }
    }
    return temp;
}

vector<vector<double>> J_inv(vector<double> &x) //J^-1
{
    vector<vector<double>> J(4, vector<double>(4));

    J[0][0] = 1;
    J[1][0] = x[1];
    J[2][0] = x[1] * x[1];
    J[3][0] = x[1] * x[1] * x[1];

    J[0][2] = 1;
    J[1][2] = x[3];
    J[2][2] = x[3] * x[3];
    J[3][2] = x[3] * x[3] * x[3];

    J[0][1] = 0;
    J[1][1] = x[0];
    J[2][1] = 2 * x[0] * x[1];
    J[3][1] = 3 * x[0] * x[1] * x[1];

    J[0][3] = 0;
    J[1][3] = x[2];
    J[2][3] = 2 * x[2] * x[3];
    J[3][3] = 3 * x[2] * x[3] * x[3];

    double det = determine(J, 4);
    if (det == 0)
        throw invalid_argument("singular Jacobi matrix");

    return inv(J);
}

double norm(vector<double> &a, vector<double> &b)
{
    double maximal = 0, diff;
    for (int i = 0; i < a.size(); ++i)
    {
        diff = abs(a[i] - b[i]);
        maximal = max(maximal, diff);
    }
    return maximal;
}

vector<double> F(vector<double> &x, vector<double> &g)
{
    vector<double> f(4);
    f[0] = x[0] + x[2] - g[0];
    f[1] = x[0] * x[1] + x[2] * x[3] - g[1];
    f[2] = x[0] * x[1] * x[1] + x[2] * x[3] * x[3] - g[2];
    f[3] = x[0] * x[1] * x[1] * x[1] + x[2] * x[3] * x[3] * x[3] - g[3];
    return f;
```

```
}

vector<double> vector_minus(vector<double> &a, vector<double> &b)
{
    vector<double> f(a.size());
    for (int i = 0; i < a.size(); ++i)
        f[i] = a[i] - b[i];
    return f;
}

vector<double> multiply(vector<vector<double>> &a, vector<double> &b)
{
    vector<double> f(b.size(), 0);
    for (int i = 0; i < b.size(); ++i)
        for (int j = 0; j < b.size(); ++j)
            f[i] += a[i][j] * b[j];
    return f;
}

double discrepancy(vector<double> &x)
{
    return max(max(abs(x[0]), abs(x[1])), max(abs(x[2]), abs(x[3])));
}

int main()
{
    double eps = 1e-10;
    vector<double> x_prev(4), x_cur(4); //A0, x0, A1, x1
    vector<double> g(4);
    for (int i = 0; i < 4; ++i)
        g[i] = integral(i);
    cout << "input filename" << endl;
    string filename;
    cin >> filename;
    ofstream fout("out" + filename + ".txt");
    fout << "g" << endl;
    for (int i = 0; i < 4; ++i)
        fout << g[i] << " ";
    fout << endl;
    fout << endl;
    cout << "input initial approximation A0, x0, A1, x1" << endl;
    for (int i = 0; i < 4; ++i)
        cin >> x_cur[i];
    fout << "initial approximation A0, x0, A1, x1" << endl;
    for (int i = 0; i < 4; ++i)
        fout << x_cur[i] << " ";
    fout << endl;
    fout << endl;
    vector<vector<double>> W;
    vector<double> f, mult;
    int iteration = 0;
    double norma;
    try
    {
        do
        {
            x_prev = x_cur;
            W = J_inv(x_prev);
            f = F(x_prev, g);
            mult = multiply(W, f);
            x_cur = vector_minus(x_prev, mult);
```

```
            norma = norm(x_cur, x_prev);
            fout << iteration << '\t' << fixed << setprecision(20) <<
discrepancy(f) << endl;
            iteration++;
        } while ((norma > eps) && (iteration < 1000));
        f = F(x_cur, g);
        fout << iteration << '\t' << fixed << setprecision(20) << discrepancy(f)
<< endl;
        fout << endl;
        fout << "A0, x0, A1, x1" << endl;
        for (int i = 0; i < x_cur.size(); ++i)
        {
            fout << setprecision(12) << x_cur[i] << " ";
        }
    }
    catch (invalid_argument e)
    { cout << e.what() << endl; }
    fout.close();
    return 0;
}
```