Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования   
«Алтайский государственный техничеcкий университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной работе № 2

" Решение систем линейных уравнений методами Якоби и Зейделя "

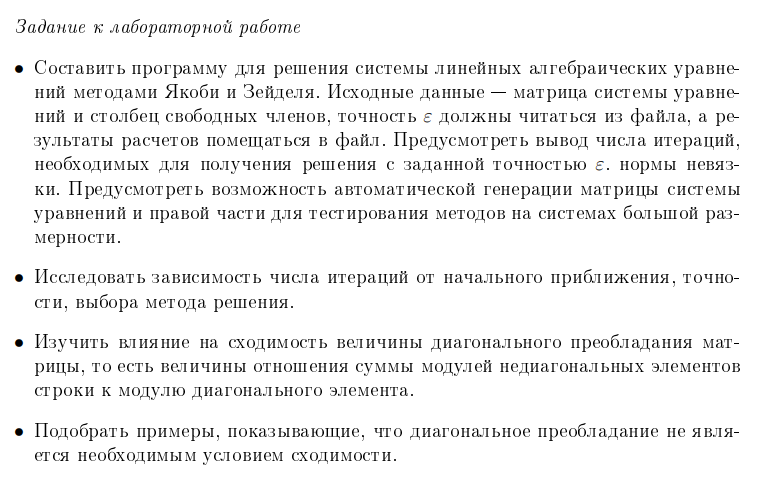
по дисциплине «Вычислительные алгоритмы»

Студент группы ПИ-92 Шинтяпин И. И.

Преподаватель доцент, к.ф.-м.н. Проскурин А. В.

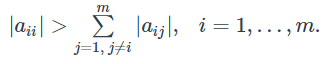
Барнаул 2022

**Задание**



**Краткое описание алгоритма и расчетные формулы**

Изначально производится проверка достаточного условия сходимости в функции bool is\_diagonal\_preopladanie(double \*\*matrix, int \*n) для методов Якоби и Зейделя. Условие сходимости, заключается в том, что диагональное преобладание матрицы A должно быть меньше 1, то есть сумма модулей не диагональных элементов строки должна быть меньше модуля диагонального элемента строки в каждой строке.

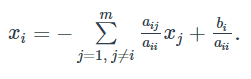


**Метод Якоби**

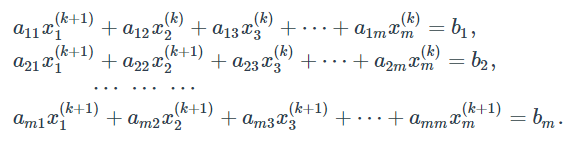
Поиск решения происходит до тех пор, пока выполняется условие:

где x^(k+1) и x^(k) – решение системы на k+1 и k итерации соответственно, а e – точность.

На k итерации в текущую строку системы подставляется значение X с k-1 итерации, причем в первой итерации подставляем первоначальное приближение, кроме диагональных элементов. После чего из получившихся уравнений находим x на главной диагонали, разделив полученное значение разности на коэффициент при текущем X.



Метод Зейделя

Аналогичен методу Якоби, разница в том, что в каждой итерации, до диагонального элемента, X подставляется с текущей итерации, после диагонального элемента – с прошлой итерации. 

**Текст программы с комментариями**

#include "stdafx.h"

#include <iostream>

#include <Conio.h>

#include <Windows.h>

#include <vector>

#include <fstream>

using namespace std;

const int max\_count = 100;// максимальная размерность матрицы

void read(double \*\*matrix, int \*n, ifstream \*fin);// функция чтения матрицы из файла

void print(double \*\*matrix, int \*n, double \*e, FILE \*file, bool flag);// функция вывода расширенной матрицы

bool is\_diagonal\_preopladanie(double \*\*matrix, int \*n);// функция, проверяющая диагональное преобладание

void search\_nevyzka(double \*\*matrix, int \*n, FILE \*file, vector<double> \*X);// функция поиска невязки

void first\_method(double \*\*matrix, int \*n, double \*e, FILE \*file);// решение системы линейных уравнений методом Якоби

void second\_method(double \*\*matrix, int \*n, double \*e, FILE \*file);// решение системы линейных уравнений методом Зейделя

void random(double \*\*matrix, int \*n);// генерация случайной матрицы

// функция чтения расширенной матрицы из файла

void read(double \*\*matrix, int \*n, ifstream \*fin)

{

for (int i = 0; i < \*n; i++)

{

for (int j = 0; j < \*n + 1; j++)

{

\*fin >> matrix[i][j];

}

}

}

// функция вывода расширенной матрицы в консоль и в файл

void print(double \*\*matrix, int \*n, double \*e, FILE \*file, bool flag)

{

if (!flag)

{

for (int i = 0; i < \*n; i++)

{

for (int j = 0; j < \*n + 1; j++)

{

printf("%10lf ", matrix[i][j]);

fprintf(file, "%10lf ", matrix[i][j]);

}

printf("\n\n");

fprintf(file, "\n\n");

}

}

else

{

for (int i = 0; i < \*n; i++)

{

for (int j = 0; j < \*n; j++)

{

printf("%10lf ", matrix[i][j]);

fprintf(file, "%10lf ", matrix[i][j]);

}

printf("\n\n");

fprintf(file, "\n\n");

}

}

cout << "Точность: " << \*e << endl;

fprintf(file, "Точность: %lf", \*(e));

cout << endl;

fprintf(file, "\n");

}

// функция, проверяющая диагональное преобладание системы

bool is\_diagonal\_preopladanie(double \*\*matrix, int \*n)

{

double sum = 0;

for (int i = 0; i < \*n; i++)// прогонка по строкам

{

sum = 0;

for (int j = 0; j < \*n; j++)// сложение всех элементов в строке, кроме диагонального

{

if (i != j)

{

sum += abs(matrix[i][j]);

}

}

if (abs(matrix[i][i]) <= sum)// проверка на условие диагонального преобладания

{

cout << "Ошибка, достаточное условие сходимости не выполнятеся" << endl;

cout << "Продолжить вычисления?\n1) да\n2) нет";

if (\_getch() == '2')

{

cout << "Завершение работы программы!" << endl;

return false;// сходимость нарушена

}

else

{

cout << endl << endl;

break;

}

}

}

return true;

}

//решение системы линейных уравнений методом Якоби

void first\_method(double \*\*matrix, int \*n, double \*e, FILE \*file)

{

vector<double> X(\*n);// текущие корни уравнения

vector<double> X\_back(\*n);// корни уравнения прошлой итерации

bool e\_flag;// флаг проверки, что необходимая точность достигнута

double current\_e;// точность в очередной итерации

int k = 0;// кол-во итераций

double sum;// временная переменная для нахождения корней

printf("| K |");

fprintf(file, "| K |");

for (int i = 1; i < \*n+1; i++)

{

printf(" X%d |", i);

fprintf(file, " X%d |", i);

}

for (int i = 1; i < \*n + 1; i++)

{

printf(" e%d |", i);

fprintf(file, " e%d |", i);

}

printf("\n");

fprintf(file, "\n");

// цикл с постусловием, пока не будет найдено решение системы с заданной точностью

do{

e\_flag = true;

// цикл по строкам матрицы,

//за каждую итерацию нахождение одного корня на главной диагонали текущей строки

for (int i = 0; i < \*n; i++)

{

sum = matrix[i][\*n];

// цикл по столбцам

for (int j = 0; j < \*n; j++)

{

if (i != j)

{

sum -= matrix[i][j] \* X\_back[j];

}

}

X\_back[i] = X[i];// запоминание предыдущего корня

X[i] = sum / matrix[i][i];// вычисление нового корня

}

// вывод номера итерации и значения корней

printf("|%3d |", k);

fprintf(file, "|%3d |", k);

for (int i = 0; i < \*n; i++)

{

printf("%10lf |",X[i]);

fprintf(file, "%10lf |", X[i]);

}

if (k != 0)

{

// нахождение текущей точности

for (int i = 0; i < \*n; i++)

{

current\_e = abs(X[i] - X\_back[i]); // abs(X[i]);

if (current\_e >= \*e)// если хотя бы одна точность не соответствует необходимой, обнуление флага

{

e\_flag = false;

}

printf("%10lf |", current\_e);

fprintf(file, "%10lf |", current\_e);

}

}

else

{

for (int i = 0; i < \*n; i++)

{

printf(" - |");

fprintf(file, " - |");

}

e\_flag = false;

}

printf("\n");

fprintf(file, "\n");

k++;// переход к следуещей итерации

if (k > 500)

{

cout << "k>500, решение системы расходится!" << endl;

break;

}

} while (!e\_flag);

search\_nevyzka(matrix, n, file, &X);// поиск невязки

}

// решение системы линейных уравнений методом Зейделя

void second\_method(double \*\*matrix, int \*n, double \*e, FILE \*file)

{

vector<double> X(\*n);// текущие корни уравнения

vector<double> X\_back(\*n);// корни уравнения прошлой итерации

bool e\_flag;// флаг проверки, что необходимая точность достигнута

double current\_e;// точность при вычилении текущего корня

int k = 0;// кол-во итераций

double sum;// временная переменная для нахождения промежуточных вычилений

printf("| K |");

fprintf(file, "| K |");

for (int i = 1; i < \*n + 1; i++)

{

printf(" X%d |", i);

fprintf(file, " X%d |", i);

}

for (int i = 1; i < \*n + 1; i++)

{

printf(" e%d |", i);

fprintf(file, " e%d |", i);

}

printf("\n");

fprintf(file, "\n");

// цикл с постусловием, пока не будет достигнута необходимая точность

do{

e\_flag = true;

// цикл по строкам системы

//за каждую итерацию нахождение одного корня на главной диагонали текущей строки

for (int i = 0; i < \*n; i++)

{

sum = 0;

for (int j = 0; j < \*n; j++)

{

if (i != j)

{

sum -= matrix[i][j] \* X[j];// разность произведения элементов строки на соответствующий корень

}

}

sum += matrix[i][\*n];

X\_back[i] = X[i];// запоминания предыдущего корня для поиска точности

X[i] = sum / matrix[i][i];// вычисление очередного корня

}

printf("|%3d |", k);

fprintf(file, "|%3d |", k);

for (int i = 0; i < \*n; i++)

{

printf("%10lf |", X[i]);

fprintf(file, "%10lf |", X[i]);

}

if (k != 0)

{

// вычисление невязки

for (int i = 0; i < \*n; i++)

{

current\_e = abs(X[i] - X\_back[i]); // abs(X[i]);

if (current\_e >= \*e)// если хотя бы одна точность не соответствует необходимой, обнуление флага

{

e\_flag = false;

}

printf("%10lf |", current\_e);

fprintf(file, "%10lf |", current\_e);

}

}

else

{

for (int i = 0; i < \*n; i++)

{

printf(" - |");

fprintf(file, " - |");

}

e\_flag = false;

}

printf("\n");

fprintf(file, "\n");

k++;// переход к следующей итерации

if (k > 500)

{

cout << "k>500, решение системы расходится!" << endl;

break;

}

} while (!e\_flag);

search\_nevyzka(matrix, n, file, &X);// поиск невязки

}

// функция поиска невязки и нормы невязки

void search\_nevyzka(double \*\*matrix, int \*n, FILE \*file, vector<double> \*X)

{

//нахождение и вывод невязки

double \*nevyzka = new double;// значение невязки

double \*max = new double;

cout << endl << "Невязка равна:" << endl;

fprintf(file, "Невязка равна:\n");

// цикл по строкам матрицы, за каждую итерацию нахождения невязки для текущей строки

for (int i = 0; i < \*n; i++)

{

\*nevyzka = 0;

for (int j = 0; j < \*n; j++)

{

\*nevyzka += matrix[i][j] \* X->at(j);

}

\*nevyzka -= matrix[i][\*n];

// нахождение нормы невязки

if (i == 0)

{

\*max = abs(\*nevyzka);// начальное значение нормы

}

else if (\*max < abs(\*nevyzka))// если найдено большее значение

{

\*max = abs(\*nevyzka);

}

// вывод невязки в консоль и в файл

cout << "Строка " << i + 1 << " = " << \*nevyzka << endl;

fprintf(file, "Строка %d = %lf\n", i + 1, \*nevyzka);

}

cout << "Норма невязки: " << \*max << endl;

delete nevyzka;

delete max;

}

// генерация случайной матрицы

void random(double \*\*matrix, int \*n)

{

for (int i = 0; i < \*n; i++)

{

for (int j = 0; j < \*n + 1; j++)

{

matrix[i][j] = (rand() % 50) + 1;

}

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

SetConsoleCP(1251);// подключение русскоязычного ввода/вывода

SetConsoleOutputCP(1251);

system("Color F0");

int n;// размерность матрицы

double e;// точность вычислений

double \*\*matrix = nullptr;// указатель на динамический массив расширенной матрицы

cout << "Выберите откуда брать данные:" << endl;

cout << "1) из файла input.dat" << endl;

cout << "2) сгенерировать автоматически" << endl;

char c = \_getch();

if (c == '1')

{

system("cls");

ifstream fin("input.dat");

if (fin.is\_open() == false)

{

cout << "Ошибка открытия файла input.dat\nзавершение работы" << endl;

\_getch();

return 1;

}

// чтение данных из файла

fin >> e;

fin >> n;

matrix = new double \*[n];

for (int i = 0; i < n; i++)

{

matrix[i] = new double[n + 1];

}

read(matrix, &n, &fin);

fin.close();

}

else

{

system("cls");

cout << "Введите размерность матрицы: ";

cin >> n;

cout << "Введите необходимую точность: ";

cin >> e;

matrix = new double \*[n];

for (int i = 0; i < n; i++)

{

matrix[i] = new double[n + 1];

}

random(matrix, &n);

}

FILE \*file = fopen("output.dat", "w");

if (file == NULL)

{

cout << "Ошибка открытия файла output.dat\nзавершение работы" << endl;

\_getch();

return 1;

}

cout << "Расширенная матрица:" << endl;

fprintf(file, "Расширенная матрица:\n");

print(matrix, &n, &e, file, 0);// вывод расширенной матрицы в консоль

// проверка на диагональное преобладание

if (!is\_diagonal\_preopladanie(matrix, &n))

{

system("pause");

return 1;

}

// решение системы методом Якоби

first\_method(matrix, &n, &e, file);

cout << endl;

fprintf(file, "\n");

// решение системы методом Зейделя

second\_method(matrix, &n, &e, file);

fclose(file);

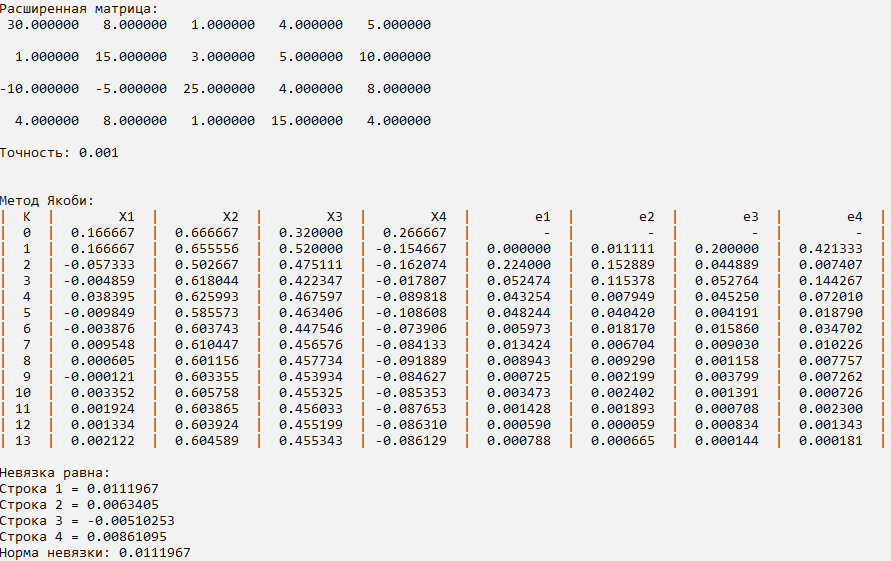
system("pause");

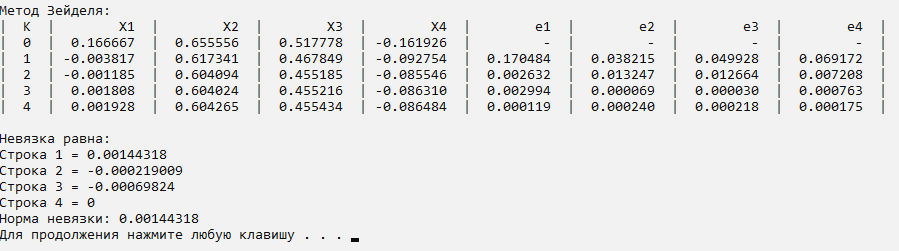
return 0;

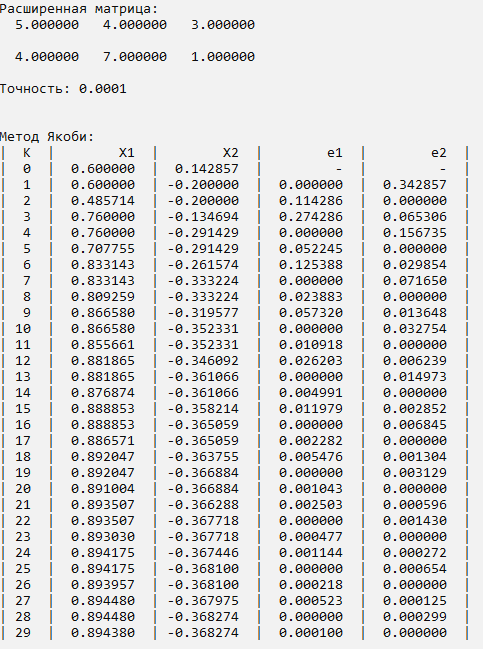
}

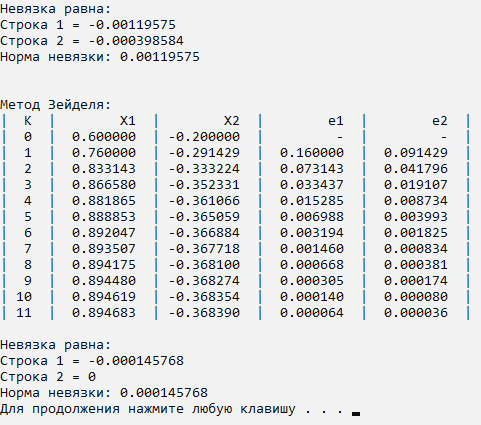
**Тесты программы:**

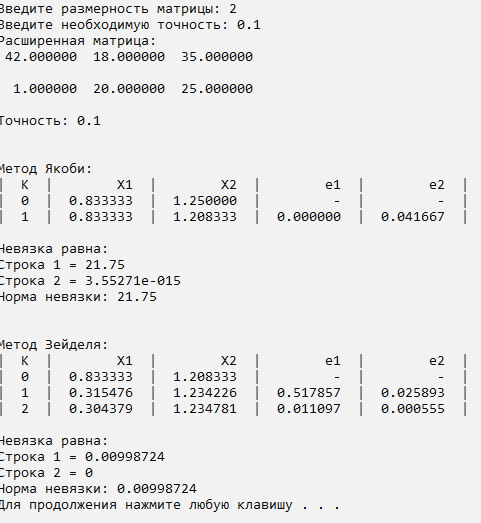






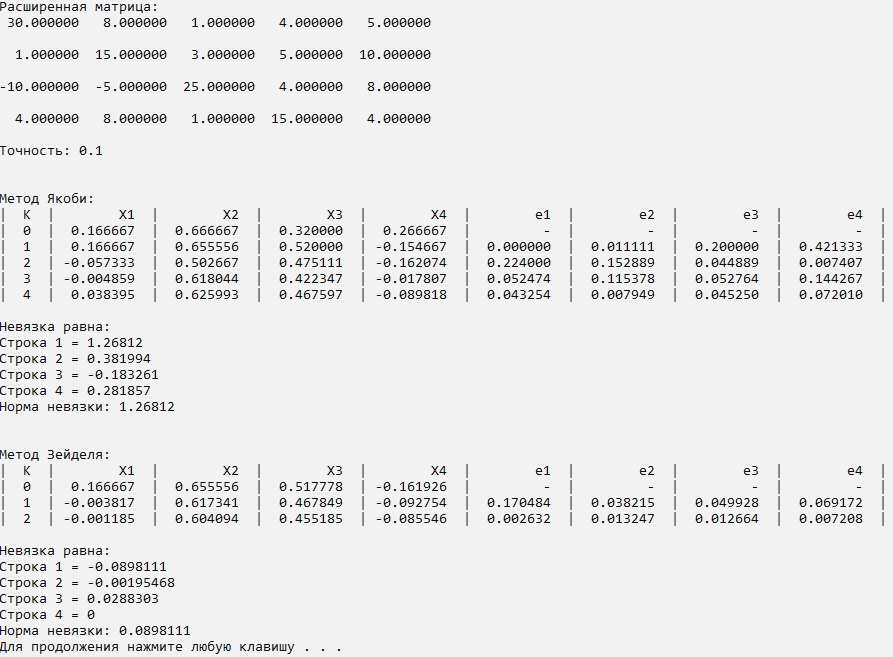




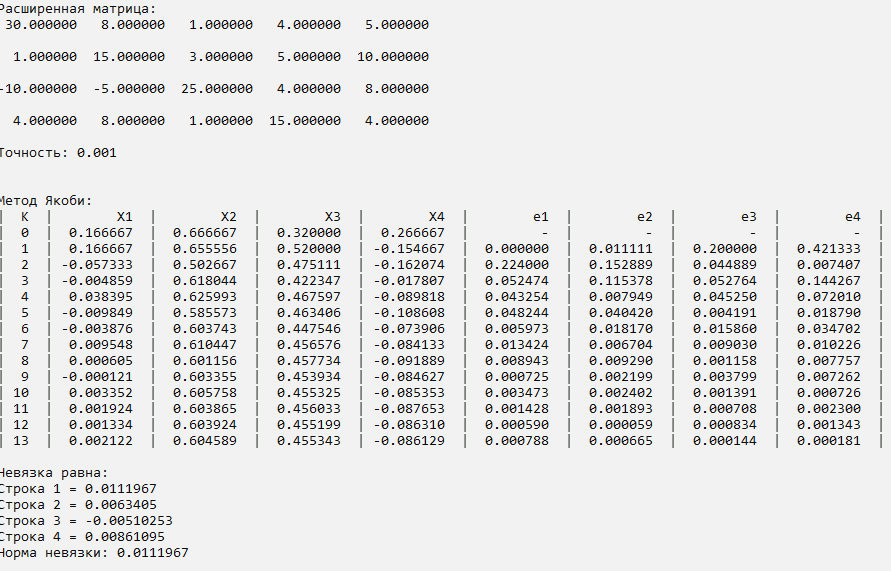


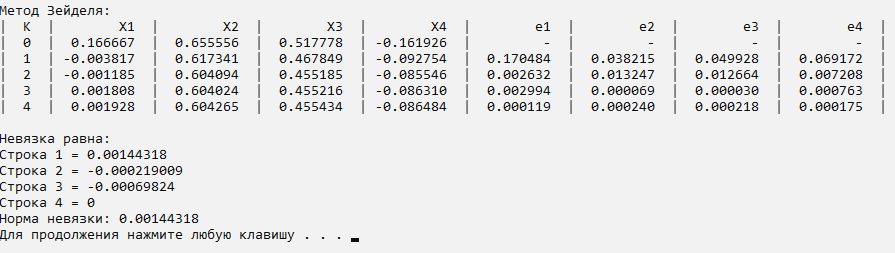
1. Проверка влияния точности на число итераций: при каждом тесте изменяем только значение точности е

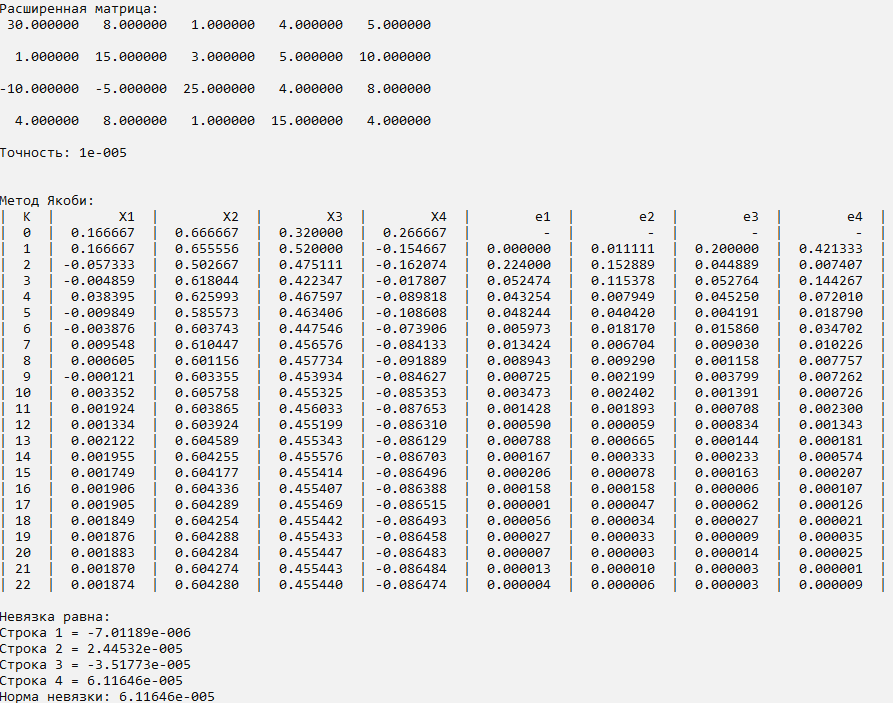
Изначально берем небольшую точность

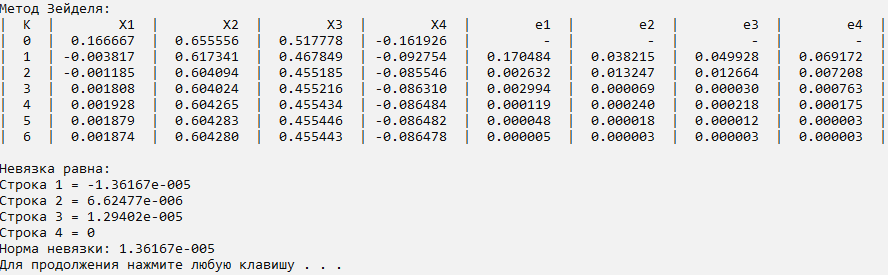


Далее увеличиваем точность вычислений



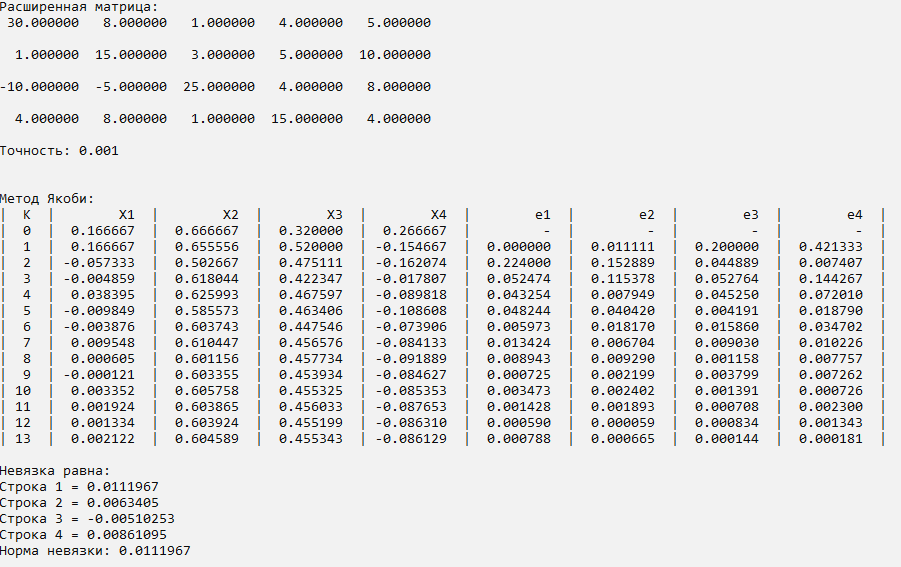


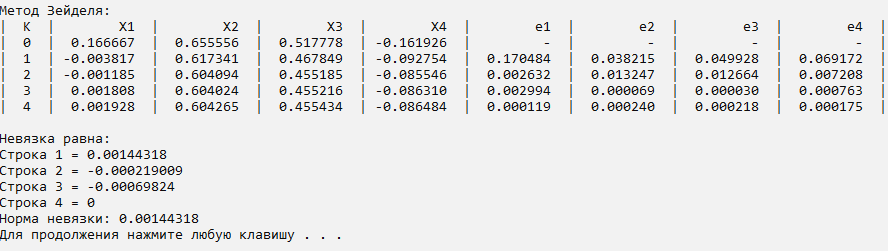


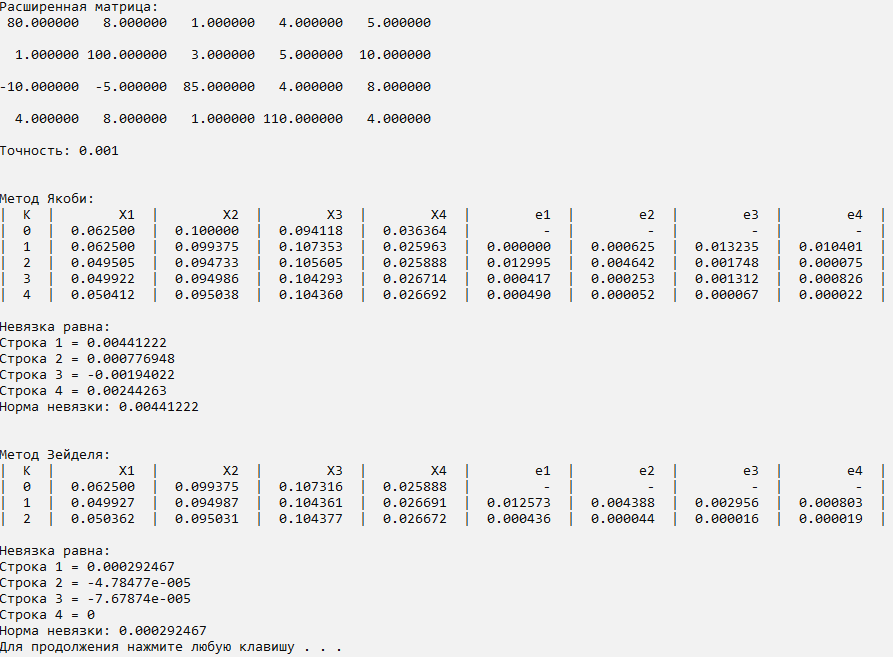


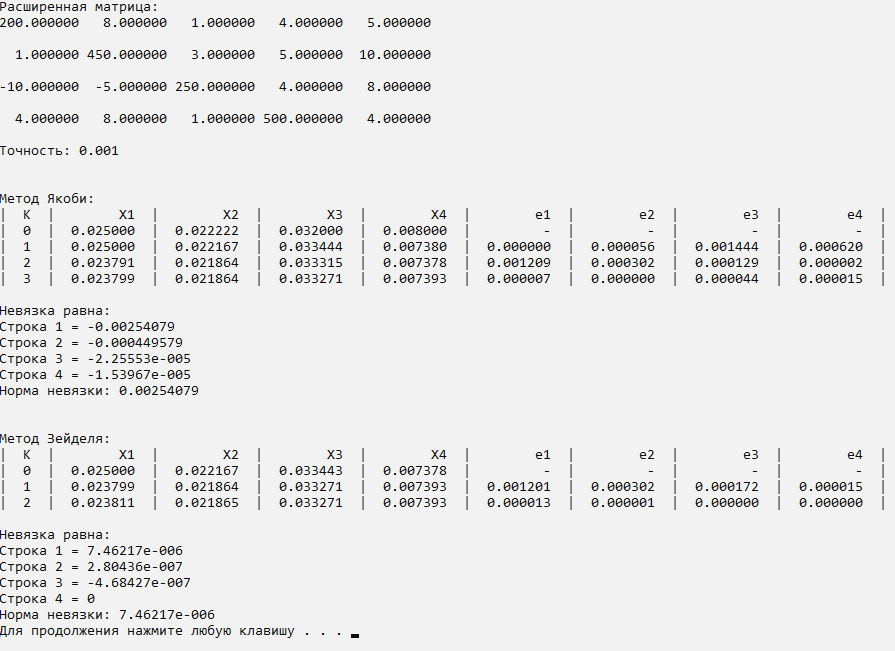
Вывод: видно, что при увеличении точности, кол-во итераций значительно возрастает.

1. Сравнение методов: метод Зейделя является более рациональным, так как при одинаковых входных данных требует значительно меньшего кол-ва итераций для достижения заданной точности вычислений. Стоит заметить, что в методе Зейделя, изменения значения точности, влияет на кол-во итераций в меньшей степени, нежели в методе Якоби.
2. Проверка влияния диагонального преобладания точности на кол-во итераций: в каждом примере будем увеличивать значения диагональных элементов, остальные данные оставляем без изменений.









Вывод: с увеличением значения диагональных элементов кол-во итераций значительно уменьшается.