**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Численные методы

**Отчет по лабораторной работе № 5**

«Итерационные методы решения   
систем линейных алгебраических уравнений»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-353 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Шамаев И.Р |  |  |  |
| Принял |  |  |  |  |

**Уфа 2021**

**Цель работы:** получить навык проведения вычислительного эксперимента, направленного на исследование свойств итерационных методов решения СЛАУ.

**Теоретическая часть**

***Задача 1. Генерация СЛАУ***

Генерируемая матрица должна быть ленточной, симметричной, положительно определенной и обладать диагональным преобладанием. Размерность (N×N) матрицы системы и параметр l, определяющий ширину ленты, указаны в индивидуальном задании. Генерация включает несколько этапов.

1. Случайным образом генерируются внедиагональные элементы ленточной матрицы А:
2. Генерируются диагональные элементы таким образом, чтобы обеспечить диагональное преобладание:

Необходимо сгенерировать три различных ленточных матрицы А1, А2, А3, соответствующие *q =* {1.1, 2, 10}, и отличающиеся, таки образом, только диагональными элементами. Ширина ленты *2l+1.*

1. Сгенерировать случайный вектор *x* размерности *N с элементами*

Этот вектор будет представлять собой вектор точного решения СЛАУ.

1. По известным А1, А2, А3 и *x* вычислить три различных вектора правой части системы
2. Выполнить симметризацию системы путем умножения слева на транспонированную матрицу АТ:

Полученная в результате матрица новой системы А\*= АТА будет симметричной, положительно определенной, ленточной (с шириной ленты 4l+1) и обладать диагональным преобладанием.

Таким образом, в следующих задачах будут решаться СЛАУ

с тремя различными матрицами и тремя векторами правых частей

***Задача 2. Метод Якоби***

Пусть даны вещественная *n×n* матрица *А* и вещественный вектор *b* размерности *n*. Рассматриваем следующую задачу: найти вектор *x* из такой, что

Метод предусматривает переход от одного приближения к другому посредством изменения компоненты текущего приближения. Такой подход вполне естественен, поскольку имеются простые критерии изменения компонент, позволяющих улучшить приближение. Одной из возможностей является аннулирование какой-то компоненты вектора невязки

В методе Якоби *i*-компонента следующего приближения выбирается так, чтобы аннулировать *i*-ю компоненту невязки. В дальнейшем будем обозначать *i*-ю компоненту приближения через а *i*-ю компоненту правой части *b* через Таким образом, можем записать

где обозначение используется для *i*-й компоненты вектора *y.* Отсюда получаем

или

***Задача 3. Метод SOR***

Метод Гаусса-Зейделя, как и метод Якоби, тоже подправляет i-ю компоненту текущего приближения с тем, чтобы аннулировать i-ю компоненту невязки, и тоже придерживается порядка Однако здесь приближенное решение перестраивается сразу вслед за тем, как определена новая компонента. Пересчет компонент может выполняться в рабочем векторе, переопределяемом на каждом шаге релаксации. Поскольку принят порядок то i-й шаг описывается равенством

что приводит к формуле

Метод последовательной верхней релаксации (SOR) соответствует релаксационной последовательности

где – параметр релаксации. Если значение параметра релаксации равно 1, приходим к методу Гаусса-Зейделя, описанному выше.

***Задача 4. Метод PCGM***

Вектор можно записать в виде

Векторы невязок должны удовлетворять рекурсии

Из попарной ортогональности векторов необходимо вытекает соотношениеКак следствие,

Известно также, что следующее направление спускаесть линейная комбинация векторов *и .* После соответствующего масштабирования векторов имеем

В качестве первого следствия этого соотношения получаем

так как вектор ортогонален к *.* Теперь формулу для можно переписать в виде

Кроме того, ортогональность вектора к вектору *,* дает равенство

Заметим, что из рекурсии для следует

а потому

Объединяя эти соотношения, получаем искомый алгоритм*.*

**Практическая часть**

***Задача 1. Генерация СЛАУ***

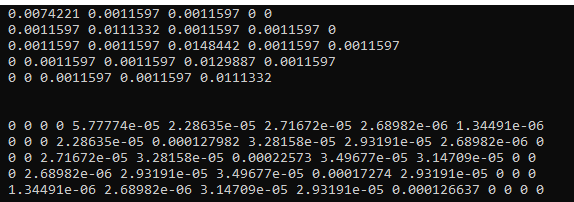


Рисунок 1. Пример тестового варианта работы генерации

В результате, на данном этапе работы программа выводит сгенерированную матрицу А, её представление в ленточном виде.

***Задача 2. Метод Якоби***

1) Написать вычислительную программу на языке программирования C++ для решения СЛАУ с указанной в индивидуальном задании точностью методом Якоби, являющегося частным случаем метода простых итераций.

2) С использованием написанной программы исследовать зависимость числа итераций метода Якоби, необходимых для достижения заданной точности, от величины параметра q, определяющего степень диагонального преобладания

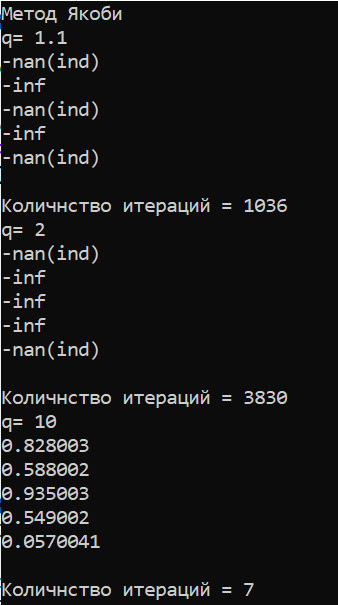


Рисунок 2. Пример выполнения программы метода Якоби

***Задача 3. Метод SOR***

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ с указанной в индивидуальном задании точностью методом последовательной верхней релаксации (SOR) с параметром релаксации ω∈(0,2).
2. С использованием написанной программы исследовать зависимость числа итераций метода SOR от параметров *q* и ω. При сравнении предусмотреть частный случай ω=1, соответствующий методу Гаусса-Зейделя.

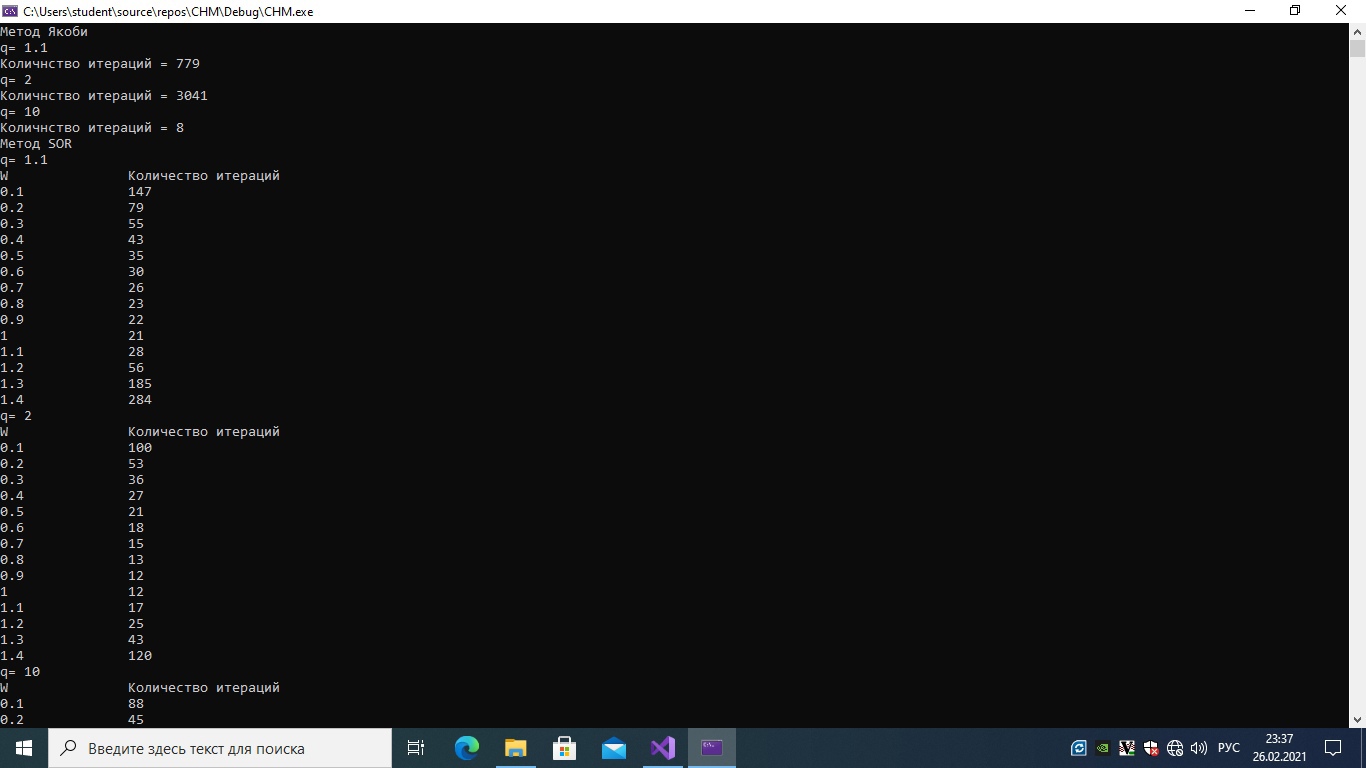


Рисунок 3. Пример выполнения программы метода SOR для матрицы q=1.1

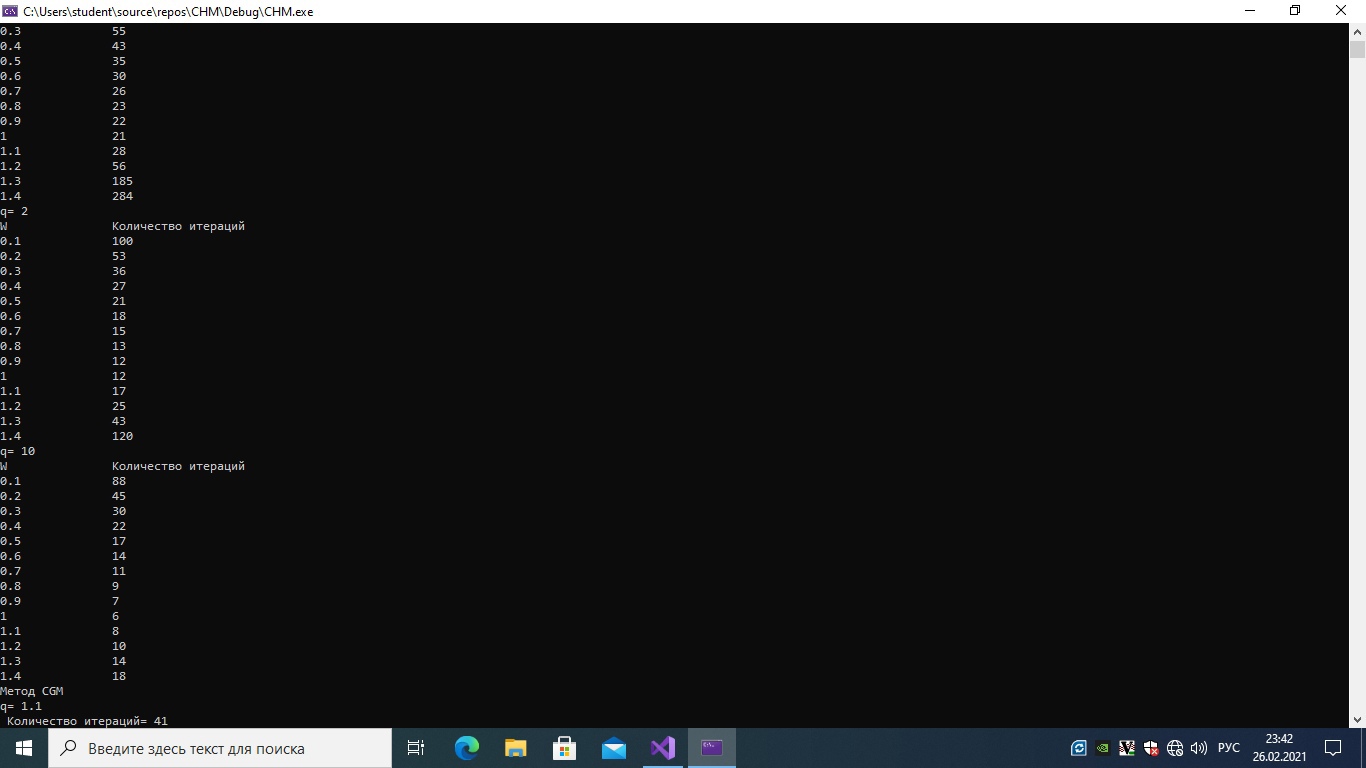


Рисунок 4. Пример выполнения программы метода SOR для матрицы q=2, q=10

***Задача 4. Метод PCGM***

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ с указанной в индивидуальном задании точностью методом сопряженных градиентов (CGM).
2. С использованием написанной программы исследовать зависимость числа итераций метода сопряженных градиентов от параметра *q*.
3. Выполнить модификацию написанной программы путем введения предобуславливателя в виде m-шагового метода Якоби.
4. Для системы с матрицей Ai, требующей наибольшего числа итераций метода сопряженных градиентов, с использованием написанной программы исследовать зависимость числа итераций метода сопряженных градиентов с предобуславливателем (PCGM) от количества шагов m метода Якоби, используемого в качестве предобуславливателя.

Результат: На рисунках 5, 6 показана корректность выполнение программы.

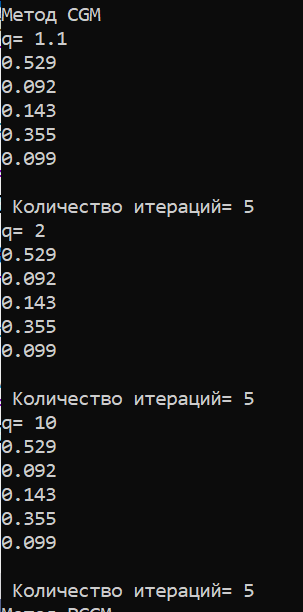


Рисунок 5. Решения для матриц методом CGM

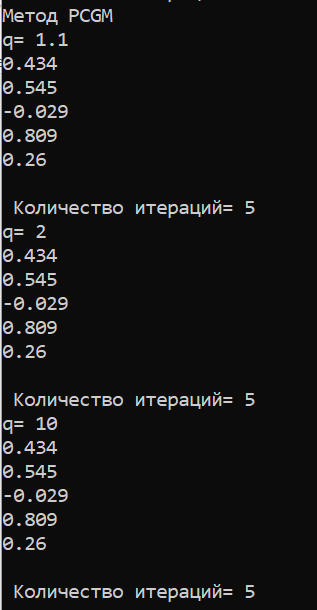


Рисунок 6. Решения для матриц методом PCGM

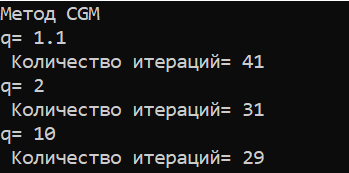


Рисунок 7. Решения для соответствующих матриц методом CGM

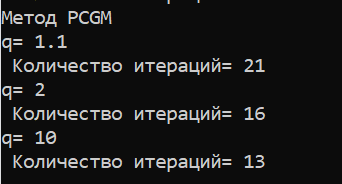


Рисунок 8. Решения для соответствующих матриц методом PCGM

**Вывод**

В результате проделанной лабораторной работы был изучен теоретический материал необходимый для решения поставленных задач по решению систем линейных уравнений итерационными методами и получен навык проведения вычислительного эксперимента, направленного на исследование свойств итерационных методов СЛАУ.

**Список литературы**

1. Юсеф Саад. Итерационные методы для разреженных линейных систем: Учеб. пособие. – В 2-х томах. Том 1 / Пер. с англ.: Х.Д.Икрамов. – М.: Издательство Московского университета, 2013. – 344 с.
2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы: Бином, 2018. – 636 с.
3. Самарский А.А., Гулин А. В. Численные методы: Учеб, пособие для вузов, — М.: Наука. Гл. ред. физ-мат. лит., 1989.— 432 с.

**Приложение**

***Листинг программы к задачам 1-4***

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <fstream>

using namespace std;

void Out(vector<vector<double>> vec)//Вывод матрици

{

for (int i = 0; i < vec.size(); i++)

{

for (int j = 0; j < vec[i].size(); j++)

{

cout << vec[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

void Out(vector<double> vec)//вывод вектора

{

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << "\n";

}

cout << endl;

}

vector<double> Zap(int N)//Заполнить вектор начального значения

{

vector<double> vec; vec.resize(N);

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

{

vec[i] = 0.5;

}

return vec;

}

double Norm(vector<double> vec, vector<double> vec1)//в цикле 2 вектора минус и его норма

{

double norm = 0;

for (int i = 0; i < vec.size(); i++)

{

norm += pow((vec[i] - vec1[i]), 2);

}

return sqrt(norm);

}

vector<vector<double>> Lent(int N, int l)//Матрици 3 ленточные

{

vector<double> q;

q = { 1.1,2,10 };

vector<vector<double>> vec;

vec.resize(N);

for (int i = 0; i < N; i++) vec[i].resize(3 \* N + 1);

for (int i = 0; i < N; i++)//заполнение побочных диаг

{

int left = i - l;

if (left < 0) left = 0;

int right = i + l + 1;

if (right > N) right = N;

for (int j = left; j < right; j++)

{

if (i == j) continue;

vec[i][j] = (rand() % 1001 - 50) / 1000.0;

}

}

//Out(vec);

for (int z = 0; z < q.size(); z++)//заполнение диагоналейй с q

{

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

{

for (int j = z \* N; j < z \* N + N; j++)

{

if ((i + z \* N) == j)

{

double Sum = 0;

for (int shg = 0; shg < N; shg++)

{

if (shg == i) continue;

Sum += abs(vec[i][shg]);

}

vec[i][j] = q[z] \* Sum;

continue;

}

vec[i][j] = vec[i][j - z \* N];

}

}

}

//Out(vec);

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

{

vec[i][N \* 3] = (rand() % 1001 - 50) / 1000.0;

}

//Out(vec);

return vec;

}

vector<vector<double>> Vprav(vector<vector<double>> vec, int N)//Вектора правые для соответсвующей ленточной матрици

{

vector<vector<double>> b;

b.resize(N);

for (int i = 0; i < N; i++) b[i].resize(3);

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < z \* N + N; j++)

{

b[i][z] += vec[i][j] \* vec[j - z \* N][3 \* N];

}

}

}

return b;

}

vector<vector<double>> Transp(vector<vector<double>> vec, int N)//транспонирование 3 матриц

{

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

{

for (int j = i; j < N; j++)

{

if (i == j) continue;

double per = vec[i][j];

vec[i][j] = vec[j][i];

vec[i][j + N] = vec[j][i];

vec[i][j + 2 \* N] = vec[j][i];

vec[j][i] = per;

vec[j][i + N] = per;

vec[j][i + 2 \* N] = per;

}

}

//Out(vec);

return vec;

}

vector<vector<double>> Proizved(vector<vector<double>> vec, vector<vector<double>> vect, int N)//произведение матриц для придания симметричности

{

vector<vector<double>> Rezvec;

Rezvec.resize(N);

for (int i = 0; i < N; i++) Rezvec[i].resize(3 \* N);

for (int z = 0; z < 3; z++)

{

for (int i = 0; i < N; i++)//

{

for (int j = z \* N; j < z \* N + N; j++)//По координатам результата

{

double Sum = 0;

for (int jy = z \* N; jy < z \* N + N; jy++)

{

Sum += vect[i][jy] \* vec[jy - z \* N][j];

}

Rezvec[i][j] = Sum;

}

}

}

//Out(Rezvec);

return Rezvec;

}

vector<vector<double>> proizvedB(vector<vector<double>> vect, vector<vector<double>> b, int N)//соответствующие правые вектора для соответствующих симметричых матриц

{

vector<vector<double>> B;

B.resize(N);

for (int i = 0; i < N; i++) B[i].resize(3);

for (int z = 0; z < 3; z++)

{

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

{

double Sum = 0;

for (int j = z \* N; j < z \* N + N; j++)

{

Sum += vect[i][j] \* b[j - z \* N][z];

}

B[i][z] = Sum;

}

}

//Out(B);

return B;

}

double NormMbesc(vector<vector<double>> vec, int N)//Норма бесконечности матрици

{

double max = 0, sum = 0;

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

{

sum = 0;

for (int j = 0; j < N; j++)

{

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

}

if (sum > max) max = sum;

}

//cout << max << endl;

return max;

}

int Icobi(vector<vector<double>> vec, vector<double> nach, int N, double t)//Метод Якоби

{

int it = 0;

//Преобразование B

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

{

for (int j = 0; j < N; j++)

{

if (i == j) continue;

vec[i][j] /= ((-1) \* vec[i][i]);

}

vec[i][N] /= vec[i][i];

vec[i][i] = 0;

}

//t = (1.0 - NormMbesc(vec, N)) / NormMbesc(vec, N)\*t;

vector<double> rez; rez = nach;

do

{

nach = rez;

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

{

double sum = 0;

for (int j = 0; j < N; j++)

{

sum += vec[i][j] \* nach[j];

}

rez[i] = sum + vec[i][N];

}

it++;

} while (Norm(nach, rez) > t);

//Out(rez);

ofstream Outv;

Outv.open("Vec.txt", ios::app);

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

Outv << rez[i] << " ";

Outv << endl;

return it;

}

int SOR(vector<vector<double>> vec, vector<double> nach, int N, double t, double w)

{

int it = 0;

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

{

for (int j = 0; j < N; j++)

{

if (i == j) continue;

vec[i][j] /= ((-1) \* vec[i][i]);

}

vec[i][N] /= vec[i][i];

vec[i][i] = 0;

}

vector<double> rez; rez = nach;

do

{

rez = nach;

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

{

double sum = 0;

for (int j = 0; j < N; j++)

{

sum += vec[i][j] \* nach[j];

}

nach[i] = sum + vec[i][N];

}

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

{

nach[i] = nach[i] \* w + (1 - w) \* rez[i];

}

it++;

} while (Norm(nach, rez) > t);

//if (abs(w - 1) < 0.01) { cout << endl; cout << "Вектор при w = 1" << endl; Out(nach); }

return it;

}

//Задание 4

vector<double> Zapb(int n)

{

vector<double> vec; vec.resize(n);

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

vec[i] = rand() % 1000 / 100.0;

return vec;

}

double Norm(vector<double> vec)//Норма вектора

{

double sum = 0;

for (int i = 0; i < vec.size(); i++)

sum += vec[i] \* vec[i];

return sqrt(sum);

}

double Skp(vector<double> vec1, vector<double> vec2)//Скалярное произведение векторов

{

double sum = 0;

for (int i = 0; i < vec1.size(); i++)

sum += vec1[i] \* vec2[i];

return sum;

}//Cr

vector<double> Umnog(vector<vector<double>> vec, vector<double> vecb)//Умножение матрицы на вектор

{

vector<double> rez; rez = vecb;

//Out(rez); Out(vec);

for (int i = 0; i < vecb.size(); i++)

{

rez[i] = 0;

for (int j = 0; j < vecb.size(); j++)

rez[i] += vec[i][j] \* vecb[j];

}

return rez;

}

vector<double> umchvec(double ch, vector<double> vec)//Умножение число на вектор

{

for (int i = 0; i < vec.size(); i++)

{

vec[i] \*= ch;

}

return vec;

}

vector<vector<double>> umchmat(double ch, vector<vector<double>> vec)//Умножение число на матрицу

{

for (int i = 0; i < vec.size(); i++)

{

for (int j = 0; j < vec.size(); j++)

vec[i][j] \*= ch;

}

return vec;

}

vector<double> Minus(vector<double> vec, vector<double> vec1)//вычетание векторов

{

vector<double> rez; rez = vec;

for (int i = 0; i < vec.size(); i++)

rez[i] = vec[i] - vec1[i];

return rez;

}

vector<double> Plus(vector<double> vec, vector<double> vec1)//сумма векторов

{

vector<double> rez; rez = vec;

for (int i = 0; i < vec.size(); i++)

rez[i] = vec[i] + vec1[i];

return rez;

}

int CGM(vector<vector<double>> vec, vector<double> vecb, vector<double> vecx, double t)

{

int i = 0;

vector<double> r, z, rp; r.resize(vecx.size());

double L, B;

r = Minus(vecb, Umnog(vec, vecx));

z = r;

while (/\*Norm(r) / Norm(vecb) > t\*/Norm(z) > t)

{

rp = r;

L = Skp(r, r) / Skp(Umnog(vec, z), z);

vecx = Plus(vecx, umchvec(L, z));

r = Minus(r, umchvec(L, Umnog(vec, z)));

B = Skp(r, r) / Skp(rp, rp);

z = Plus(r, umchvec(B, z));

i++;

}

//Out(vecx);

return i;

}

vector<vector<double>> ObrD(vector<vector<double>> D)

{

for (int i = 0; i < D.size(); i++)

{

D[i][i] = 1 / sqrt(D[i][i]);

}

return D;

}

vector<vector<double>> UmMatr(vector<vector<double>> D, vector<vector<double>> vec)

{

vector<vector<double>> rez; rez = D;

for (int i = 0; i < D.size(); i++)

{

for (int j = 0; j < D.size(); j++)

{

rez[i][j] = 0;

for (int z = 0; z < D.size(); z++)

{

rez[i][j] += D[i][z] \* vec[z][j];

}

}

}

return rez;

}

int PCGM(vector<vector<double>> vec, vector<vector<double>> D, vector<double> vecb, vector<double> vecx, double t)

{

//обратаня d

vector<vector<double>> Do; Do = D;

Do = ObrD(D);

vec = UmMatr(UmMatr(Do, vec), Do);

vecb = Umnog(Do, vecb);

//vecx = Umnog(D, vecx);

int i = 0;

vector<double> r, z, rp; r.resize(vecx.size());

double L, B;

vector<vector<double>> M; M = UmMatr(D, D);

M = ObrD(M);

r = Minus(Umnog(vec, vecx), vecb);

z = Umnog(umchmat(-1, M), r);

//vecx = Umnog(D, vecx);

while (/\*Norm(r) / Norm(vecb) > t\*/Norm(z) > t)

{

rp = r;

L = Skp(r, Umnog(M, r)) / Skp(Umnog(vec, z), z);

vecx = Plus(vecx, umchvec(L, z));

r = Plus(r, umchvec(L, Umnog(vec, z)));

B = Skp(r, Umnog(M, r)) / Skp(rp, Umnog(M, rp));

z = Plus(Umnog(umchmat(-1, M), r), umchvec(B, z));

i++;

}

vecx = Umnog(Do, vecx);

//Out(vecx);

return i;

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

int N = 800, l = 7; double t = 0.0001;

vector<vector<double>>vec, vect, vecRez;

vector<vector<double>>b;

vector<double> q;

q = { 1.1,2,10 };

vec = Lent(N, l);

//Out(vec);

b = Vprav(vec, N);

vect = Transp(vec, N);

vecRez = Proizved(vec, vect, N);

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

{

vecRez[i].push\_back(vec[i][3 \* N]);

}

b = proizvedB(vect, b, N);

//Вывод 3 матриц

vector<vector<double>> Rez; Rez.resize(N); for (int i = 0; i < N; i++)Rez[i].resize(N + 1);

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < N + z \* N; j++)

{

Rez[i][j - z \* N] = vecRez[i][j];

}

Rez[i][N] = b[i][z];

}

//cout << q[z] << endl;

//Out(Rez);

}

cout << "Метод Якоби" << endl;

//Задание 2

vector<double> nach;

nach = Zap(N);// cout << "Начальный вектор: " << endl; Out(nach);

ofstream Outt;

Outt.open("Matrix.txt");

ofstream Outv; Outv.open("Vec.txt");

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < N + z \* N; j++)

{

Rez[i][j - z \* N] = vecRez[i][j];

Outt << Rez[i][j - z \* N] << " ";

}

Rez[i][N] = b[i][z];

Outt << Rez[i][N] << endl;

}

Outt << endl;

cout << "q= " << q[z] << endl;

cout << "Количнство итераций = " << Icobi(Rez, nach, N, t) << endl;

}

Outt.close();

////Задание 3

cout << "Метод SOR" << endl;

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < N + z \* N; j++)

{

Rez[i][j - z \* N] = vecRez[i][j];

}

Rez[i][N] = b[i][z];

}

cout << "q= " << q[z] << endl;

cout << "W Количество итераций" << endl;

for (double w = 0.1; w < 1.5; w += 0.1)

{

cout << w << " " << SOR(Rez, nach, N, t, w) << endl;

}

}

//Задание 4 пункт 1

cout << "Метод CGM" << endl;

vector<double> vecb; vecb.resize(N);

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < N + z \* N; j++)

{

Rez[i][j - z \* N] = vecRez[i][j];

}

vecb[i] = b[i][z];

}

cout << "q= " << q[z] << endl;

cout << " Количество итераций= " << CGM(Rez, vecb, nach, t) << endl;

}

//пункт 3

cout << "Метод PCGM" << endl;

vector<vector<double>> D; D.resize(N);

for (int i = 0; i < N; i++)//заполнение предобуславлетеля

{

D[i].resize(N);

}

//Out(D);

for (int z = 0; z < 3; z++)

{

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

{

for (int j = z \* N; j < N + z \* N; j++)

{

Rez[i][j - z \* N] = vecRez[i][j];

}

D[i][i] = Rez[i][i];

vecb[i] = b[i][z];

}

cout << "q= " << q[z] << endl;

cout << " Количество итераций= " << PCGM(Rez, D, vecb, nach, t) << endl;

}

}