**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Численные методы

**Отчет по лабораторной работе № 5**

«Итерационные методы решения   
систем линейных алгебраических уравнений»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмурзин М.Э. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

**Уфа 2023**

**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*.*

***Индивидуальное задание № 1***

Задание: написать вычислительную программу на языке программирования C++ для генерации ленточной, симметричной, положительно определенной и обладающей диагональным преобладанием матрицы СЛАУ, а также векторов точного решения и правой части. Размерность (NxN) матрицы системы и параметр l, определяющий ширину ленты, указаны в индивидуальном задании. Генерация включает несколько этапов.

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

Необходимо сгенерировать три различных ленточных матрицы A1, A2, A3, соответствующие q={1.8, 2, 10}, и отличающиеся, таким образом, только диагональными элементами. В результате будут сгенерированы три ленточные (ширина ленты 2l+1) матрицы с диагональным преобладанием, не являющиеся, в общем случае, ни симметричными, ни положительно определенными. Эти свойства будут им приданы на последнем этапе.

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

Этот вектор будет представлять собой вектор точного решения СЛАУ.

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

Описание: сначала генерируется маленькая, тестовая матрица, которая выводится, а потом выводится ее ленточный вид.

при запуске программы сначала генерируется тестовый вариант работы с маленькими матрицами для проверки правильности генерации матрицы, перевода её в ленточный вид и получение вектора правой части.

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

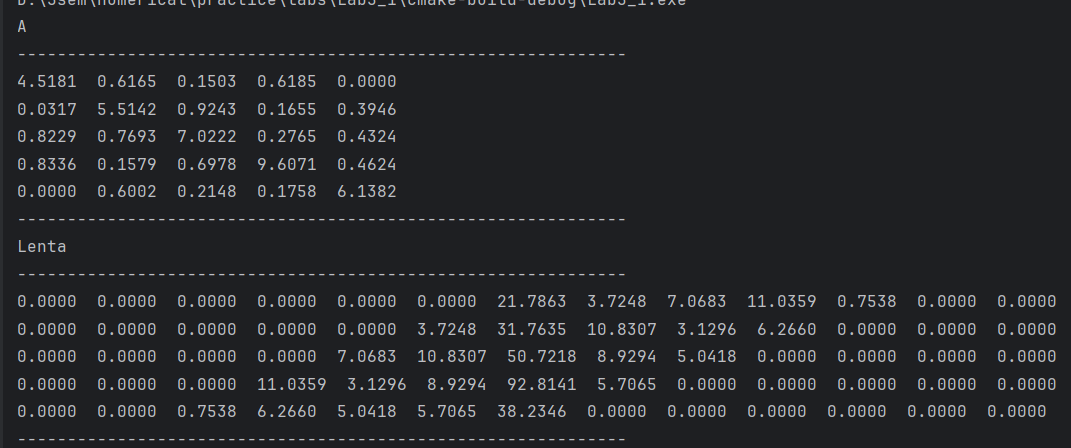


Рисунок 1 - пример работы генерации

***Индивидуальное задание № 2***

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

Описание: при запуске выводится количество понадобившихся итераций и вектор решения при разных q.

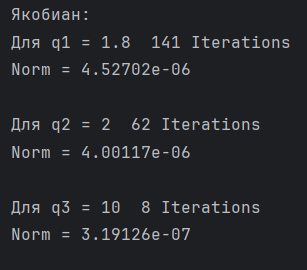


Рисунок 2. Пример выполнения программы метода Якоби

При значении параметра q равного 10, метод сходится всего за 8 итераций.

***Индивидуальное задание № 3***

Задание:

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.\

Описание: при запуске выводится количество понадобившихся итераций и вектор решения при разных значениях параметра релаксации на отрезке [0,2] с шагом 0.1.

Результат:

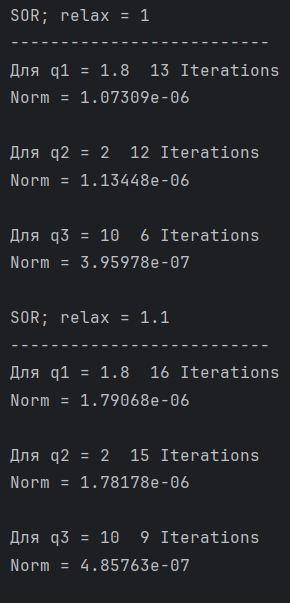


Рисунок 3. Пример работы метода SOR для ленточных матриц большой размерности

После выполнения метода на ленточной матрице большой размерности при разных значениях параметров q и , программа выводит на экран количество понадобившихся итераций для каждых значений параметров, а также разность между двумя последними приближениями.

Если коэффициент релаксации равен 1, то метод SOR сводится к методу Гаусса – Зейделя. Максимальное число итераций в данном случае достигает 13.

Быстрее всего метод сходится при коэффициенте релаксации равному 0.9.

***Индивидуальное задание № 4***

Задание:

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 , 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.

Описание: при запуске выводится количество понадобившихся итераций и вектор решения при разных значениях q.

Результат:

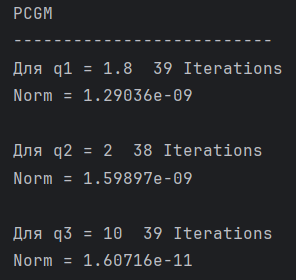


Рисунок 4 - пример работы метода CG для ленточных матриц большой размерности

После выполнения метода на ленточной матрице большой размерности при разных значениях параметра q, программа выводит на экран количество понадобившихся итераций для каждого значения параметра, а также разность между двумя последними приближениями.

При значении параметра q равного 2, метод сходится за 38итераций, что немного лучше варианта с q = 1.8.

**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**

Main.cpp

#include <stdio.h>

#include<iostream>

#include<stdlib.h>

#include<conio.h>

#include <time.h>

#include <vector>

#include <windows.h>

#include <cmath>

#include <iomanip>

#include "Methods.h"

const int N = 750;

const int l = 34;

void PrintMatrix(vector<vector<double>> Matrix)

{

cout << fixed << std::setprecision(4);

cout << "-------------------------------------------------------------" << endl;

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

{

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

{

cout << setw(5) << Matrix[i][j] << " ";

}

cout << endl;

}

cout << "-------------------------------------------------------------" << endl;

}

void PrintVector(vector<double> Vector)

{

cout << fixed << std::setprecision(4);

cout << "-------------------------------------------------------------" << endl;

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

{

cout << setw(5) << Vector[i] << " ";

}

cout << endl << "-------------------------------------------------------------" << endl;

}

void

task2(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3);

void

task3(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3);

void

task4(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3);

int main() {

SetConsoleOutputCP(CP\_UTF8);

srand((unsigned int) time(0));

vector<vector<double>> A1(N, vector<double>(N, 0));

vector<vector<double>> A2(N, vector<double>(N, 0));

vector<vector<double>> A3(N, vector<double>(N, 0));

vector<vector<double>> AT1(N, vector<double>(N, 0));

vector<vector<double>> AT2(N, vector<double>(N, 0));

vector<vector<double>> AT3(N, vector<double>(N, 0));

vector<vector<double>> M1(N, vector<double>(N, 0));

vector<vector<double>> M2(N, vector<double>(N, 0));

vector<vector<double>> M3(N, vector<double>(N, 0));

double q1 = 1.8;

double q2 = 2;

double q3 = 10;

//рнадомное заполнение

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

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

if ((l + 1) - abs(i - j) > 0)

A1[i][j] = randomValue(-1, 1);

A2[i][j] = A1[i][j];

A3[i][j] = A1[i][j];

}

}

//диагональное преобладание

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

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

A1[i][i] += fabs(A1[i][j]);

A2[i][i] += fabs(A2[i][j]);

A3[i][i] += fabs(A3[i][j]);

}

A1[i][i] \*= q1;

A2[i][i] \*= q2;

A3[i][i] \*= q3;

}

// cout << "A" << endl;

// PrintMatrix(A1);

vector<double> sol(N, 0);

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

sol[i] = randomValue(-1, 1);

}

vector<double> rhs1 = MatrixVectorMultiply(N, A1, sol);

vector<double> rhs2 = MatrixVectorMultiply(N, A2, sol);

vector<double> rhs3 = MatrixVectorMultiply(N, A3, sol);

AT1 = MatrixTranspose(N, A1);

AT2 = MatrixTranspose(N, A2);

AT3 = MatrixTranspose(N, A3);

// cout << "A" << endl;

// PrintMatrix(AT1);

M1 = MatrixMatrixmult(N, AT1, A1);

M2 = MatrixMatrixmult(N, AT2, A2);

M3 = MatrixMatrixmult(N, AT3, A3);

// cout << "A\*" << endl;

// PrintMatrix(M1);

vector<double> b1 = MatrixVectorMultiply(N, AT1, rhs1);

vector<double> b2 = MatrixVectorMultiply(N, AT2, rhs2);

vector<double> b3 = MatrixVectorMultiply(N, AT3, rhs3);

vector<double> x1(N, 0);

vector<double> x2(N, 0);

vector<double> x3(N, 0);

// создание ленточных матриц

vector<vector<double>> Lenta1(N, vector<double>(4 \* l + 1, 0));

vector<vector<double>> Lenta2(N, vector<double>(4 \* l + 1, 0));

vector<vector<double>> Lenta3(N, vector<double>(4 \* l + 1, 0));

Lenta1 = MatrixtoLenta(N, 2 \* l, M1, Lenta1);

Lenta2 = MatrixtoLenta(N, 2 \* l, M2, Lenta2);

Lenta3 = MatrixtoLenta(N, 2 \* l, M3, Lenta3);

// cout << "Lenta" << endl;

// PrintMatrix(Lenta1);

task2(q1, q2, q3, sol, b1, b2, b3, x1, x2, x3, Lenta1, Lenta2, Lenta3);

task3(q1, q2, q3, sol, b1, b2, b3, x1, x2, x3, Lenta1, Lenta2, Lenta3);

task4(q1, q2, q3, sol, b1, b2, b3, x1, x2, x3, Lenta1, Lenta2, Lenta3);

}

void

task4(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3) {

cout << " PCGM " << endl;

cout << " -------------------------- " << endl;

cout << " Для q1 = " << q1 << " ";

x1 = CGMLenta(N, 2 \* l, Lenta1, b1);

norma(N, x1, sol);

cout << endl;

cout << " Для q2 = " << q2 << " ";

x2 = CGMLenta(N, 2 \* l, Lenta2, b2);

norma(N, x2, sol);

cout << endl;

cout << " Для q3 = " << q3 << " ";

x3 = CGMLenta(N, 2 \* l, Lenta3, b3);

norma(N, x3, sol);

cout << endl;

}

void

task3(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3) {

for (double i = 0; i < 2; i += 0.1) {//relax from 0 to 2 with 0.1 step

cout << " SOR; relax = " << i << endl;

cout << " -------------------------- " << endl;

cout << " Для q1 = " << q1 << " "; x1 = SORLenta(N, 2 \* l, Lenta1, b1, i);

norma(N, x1, sol);

cout << endl;

cout << " Для q2 = " << q2 << " "; x2 = SORLenta(N, 2 \* l, Lenta2, b2, i);

norma(N, x2, sol);

cout << endl;

cout << " Для q3 = " << q3 << " "; x3 = SORLenta(N, 2 \* l, Lenta3, b3, i);

norma(N, x3, sol);

cout << endl;

}

}

void

task2(double q1, double q2, double q3, vector<double> &sol, vector<double> &b1, vector<double> &b2, vector<double> &b3,

vector<double> &x1, vector<double> &x2, vector<double> &x3, vector<vector<double>> &Lenta1,

vector<vector<double>> &Lenta2, vector<vector<double>> &Lenta3) {

cout << " Якобиан: " << endl;

cout << " Для q1 = " << q1 << " ";

x1 = Jacobian(N, 2 \* l, Lenta1, b1);

norma(N, x1, sol);

cout << endl;

cout << " Для q2 = " << q2 << " ";

x2 = Jacobian(N, 2 \* l, Lenta2, b2);

norma(N, x2, sol);

cout << endl;

cout << " Для q3 = " << q3 << " ";

x3 = Jacobian(N, 2 \* l, Lenta3, b3);

norma(N, x3, sol);

cout << endl;

}

Methods.h

#pragma once

#include <stdio.h>

#include<iostream>

#include<stdlib.h>

#include<conio.h>

#include <time.h>

#include <vector>

using namespace std;

const double eps = 1e-5;

double randomValue(double min, double max) {

min = 0;

return (double)(rand()) / RAND\_MAX \* (max - min) + min;

}

void printVector2dDouble(vector<vector<double>> vec)

{

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 printVector1dDouble(vector<double> vec)

{

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

{

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

}

cout << endl;

}

vector<double> MatrixVectorMultiply(int N, vector<vector<double>> Matrix, vector<double> Vector)

{

vector<double> VectorSolution(N, 0);

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

{

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

{

VectorSolution[i] += Matrix[i][j] \* Vector[j];

}

}

return VectorSolution;

}

vector<double> MatrixLentaVectorMultiply(int N, int lw, vector<vector<double>> MatrixLenta, vector<double> Vector)

{

vector<double> VectorSolution(N, 0);

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

{

//cout << "string = " << i << endl;

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

{

int ind = j - i + lw;

if ((ind >= 0) && (ind < 2 \* lw + 1))

{

VectorSolution[i] += MatrixLenta[i][ind] \* Vector[j];

//cout << Vector[ind] << " ";

}

}

//cout << endl;

}

return VectorSolution;

}

double VectorVectorMult(int N, vector<double> Vector1, vector<double> Vector2)

{

double result = 0;

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

{

result += Vector1[i] \* Vector2[i];

}

return result;

}

vector<vector<double>> MatrixMatrixmult(int N, vector<vector<double>> M1, vector<vector<double>> M2)

{

vector<vector<double>> Mres(N, vector<double>(N, 0));

double num;

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

{

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

{

num = 0;

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

{

num += M1[i][k] \* M2[k][j];

}

Mres[i][j] = num;

}

}

return Mres;

}

vector<vector<double>> MatrixTranspose(int N, vector<vector<double>> M)

{

double num;

double t;

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

{

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

{

t = M[i][j];

M[i][j] = M[j][i];

M[j][i] = t;

}

}

return M;

}

vector<vector<double>> MatrixtoLenta(int N, int l, vector<vector<double>> Matrix,

vector<vector<double>> MatrixLenta)

{

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

{

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

{

if (Matrix[i][j] != 0)

MatrixLenta[i][j - i + l] = Matrix[i][j];

}

}

return MatrixLenta;

}

vector<double> Jacobian(int N, int lw, vector<vector<double>> A, vector<double> b)

{

vector<double> xprev(N, 0); // ��������� �����������

vector<double> x(N, 0); // ������� �����������

double diff; // ������� ����� ������� � ���������� ������������

int iter = 0;

do

{

iter++;

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

{

x[i] = b[i];

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

{

int ind = j - i + lw;

if ((j != i) && (ind >= 0) && (ind < 2 \* lw + 1))

x[i] -= A[i][ind] \* xprev[j];

}

x[i] /= A[i][lw];

}

diff = fabs(x[0] - xprev[0]);

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

{

if (fabs(x[i] - xprev[i]) > diff)

diff = fabs(x[i] - xprev[i]);

xprev[i] = x[i];

}

} while (diff > eps);

cout << " " << iter << " Iterations" << endl;

return x;

}

vector<double> SORLenta(int N, int l, vector<vector<double>> A, vector<double> b, double relaxation)

{

vector<double> xprev(N, 0); // ��������� �����������

vector<double> x(N, 0); // ������� �����������

double diff; // ������� ����� ������� � ���������� ������������

int iter = 0;

do

{

iter++;

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

{

x[i] = b[i];

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

{

int ind = j - i + l;

if ((j != i) && (ind >= 0) && (ind < 2 \* l + 1))

x[i] -= A[i][ind] \* x[j];

}

for (int j = i + 1; j < N; j++)

{

int ind = j - i + l;

if ((j != i) && (ind >= 0) && (ind < 2 \* l + 1))

x[i] -= A[i][ind] \* xprev[j];

}

x[i] /= A[i][l];

x[i] \*= relaxation;

x[i] += (1 - relaxation) \* xprev[i];

}

diff = fabs(x[0] - xprev[0]);

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

{

if (fabs(x[i] - xprev[i]) > diff)

diff = fabs(x[i] - xprev[i]);

xprev[i] = x[i];

}

} while (diff > eps);

cout << " " << iter << " Iterations" << endl;

return x;

}

vector<double> CGMLenta(int N, int l, vector<vector<double>> A, vector<double> b)

{

vector<double> xprev(N, 0);

vector<double> x(N, 0);

vector<double> rprev(N, 0);

vector<double> r(N, 0);

vector<double> pprev(N, 0);

vector<double> p(N, 0);

double alpha = 0;

double beta = 0;

vector<double> Ax(N, 0);

vector<double> Ap(N, 0);

Ax = MatrixLentaVectorMultiply(N, l, A, xprev);

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

{

rprev[i] = b[i] - Ax[i];

pprev[i] = rprev[i];

}

double diff; // ������� ����� ������� � ���������� ������������

int iter = 0;

do

{

iter++;

Ap = MatrixLentaVectorMultiply(N, l, A, pprev);

alpha = (double)(VectorVectorMult(N, rprev, rprev) / VectorVectorMult(N, Ap, pprev));

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

{

x[i] = xprev[i] + alpha \* pprev[i];

r[i] = rprev[i] - alpha \* Ap[i];

}

beta = (double)(VectorVectorMult(N, r, r) / VectorVectorMult(N, rprev, rprev));

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

{

p[i] = r[i] + beta \* pprev[i];

}

diff = fabs(r[0] - rprev[0]);

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

{

if (fabs(r[i] - rprev[i]) > diff)

diff = fabs(r[i] - rprev[i]);

xprev[i] = x[i];

}

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

{

pprev[i] = p[i];

xprev[i] = x[i];

rprev[i] = r[i];

}

} while (diff > eps);

cout << " " << iter << " Iterations" << endl;

return x;

}

void norma(int N, vector<double> solTrue, vector<double> solOur)

{

double max = fabs(solOur[0] - solTrue[0]);

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

{

if (fabs(solOur[i] - solTrue[i]) > max)

max = fabs(solOur[i] - solTrue[i]);

}

cout << " Norm = " << max << endl;

}