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: $a_{ij} \in [-1,1](i \neq j, max(0, i-j) \leq j \leq min(i+l, N))$.
2. Генерируются диагональные элементы таким образом, чтобы обеспечить диагональное преобладание:

$$a_{ii} = q \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}|, q > 1.$$

Необходимо сгенерировать три различных ленточных матрицы A_1, A_2, A_3, соответствующие q = {1.1, 2, 10}, и отличающиеся, таки образом, только диагональными элементами. Ширина ленты *2l+1*.

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

$$x_i \in [-1,1], i = 1, 2, \ldots, N.$$

Этот вектор будет представлять собой вектор точного решения СЛАУ.

4. По известным A_1, A_2, A_3 и *x* вычислить три различных вектора правой части системы

$$b_k = A_k x.$$

5. Выполнить симметризацию системы путем умножения слева на транспонированную матрицу A^T:

$$A^T A x = A^T b.$$

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

Таким образом, в следующих задачах будут решаться СЛАУ

$$A^{¿} x = b^{¿}, A^{¿} = A^T A, b^{¿} = A^T b.$$

с тремя различными матрицами $A_1^{¿}, A_2^{¿}, A_3^{¿}$ и тремя векторами правых частей $b_1^{¿}, b_2^{¿}, b_3^{¿}$.

Задача 2. Метод Якоби

Пусть даны вещественная $n\times n$ матрица A и вещественный вектор b размерности n. Рассматриваем следующую задачу: найти вектор x из R^n такой, что

$$Ax=b.$$

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 $b-Ax$.

В методе Якоби i-компонента следующего приближения выбирается так, чтобы аннулировать i-ю компоненту невязки. В дальнейшем будем обозначать i-ю компоненту приближения x_k через $\xi_i^{(k)}$, а i-ю компоненту правой части b через β_i. Таким образом, можем записать

$$(b-A\,x_{k+1})_i=0,$$

где обозначение $(y)_i$ используется для i-й компоненты вектора y. Отсюда получаем

$$a_{ii}\xi_i^{(k+1)}=-\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\xi_j^{(k)}+\beta_i,$$

или

$$\xi_i^{(k+1)}=\frac{1}{a_{ii}}\left(\beta_i-\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\xi_j^{(k)}\right),\, i=1,\ldots,n.$$

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 $i=1,\dots,n$. Однако здесь приближенное решение перестраивается сразу вслед за тем, как определена новая компонента. Пересчет компонент $\xi_i^{(k)}(i=1,\dots,n)$ может выполняться в рабочем векторе, переопределяемом на каждом шаге релаксации. Поскольку принят порядок $i=1,2,\dots,$ то i-й шаг описывается равенством

$$\beta_i-\sum_{j=1}^{i-1}a_{ij}\xi_j^{(k+1)}-a_{ii}\xi_i^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}\xi_j^{(k)}=0,$$

что приводит к формуле

$$\xi_i^{(k+1)}=\frac{1}{a_{ii}}\left(-\sum_{j=1}^{i-1}a_{ij}\xi_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}\xi_j^{(k)}+\beta_i\right),i=1,\dots,n.$$

Метод последовательной верхней релаксации (SOR) соответствует релаксационной последовательности

$$\xi_i^{(k+1)}=\omega\xi_i^{GS}+(1-\omega)\xi_i^{(k)},i=1,\dots,n,$$

где $\omega\in(0,2)$ – параметр релаксации. Если значение параметра релаксации равно 1, приходим к методу Гаусса-Зейделя, описанному выше.

Задача 4. Метод PCGM

Вектор x_{j+1} можно записать в виде

$$x_{j+1}=x_j+\alpha_j p_j.$$

Векторы невязок должны удовлетворять рекурсии

$$r_{j+1}=r_j-\alpha_j A p_j.$$

Из попарной ортогональности векторов r_j необходимо вытекает соотношение $(r_j-\alpha_j A p_j, r_j)=0$. Как следствие,

$$\alpha_j=\frac{(r_j,r_j)}{(A p_j,r_j)}.$$

Известно также, что следующее направление спуска p_{j+1} есть линейная комбинация векторов r_{j+1} *и* p_j. После соответствующего масштабирования векторов p имеем

$$p_{j+1}=r_{j+1}+\beta_j p_j.$$

В качестве первого следствия этого соотношения получаем

$$(A p_j, r_j)=(A p_j, p_j-\beta_{j-1} p_{j-1})=(A p_j, p_j),$$

так как вектор $A p_j$ ортогонален к p_{j-1}. Теперь формулу для α_j можно переписать в виде

$$\alpha_j=\frac{(r_j,r_j)}{(A p_j,p_j)}.$$

Кроме того, ортогональность вектора p_{j+1} к вектору $A p_j$, дает равенство

$$\beta_j=\frac{-(r_{j+1},A p_j)}{(p_j,A p_j)}.$$

Заметим, что из рекурсии для r_{j+1} следует

$$A p_j=\frac{-1}{\alpha_j}(r\text{¿¿}\,j+1-r_j),\text{¿}$$

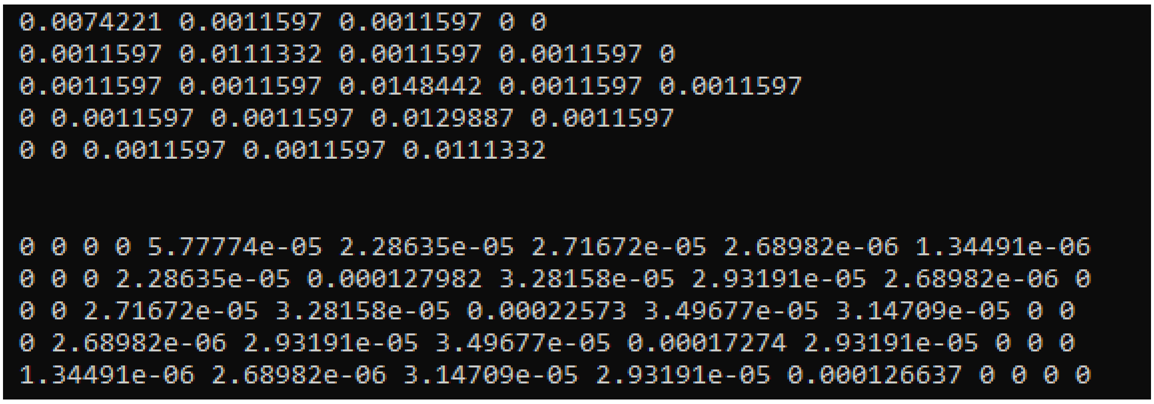
а потому

$$\beta_j=\frac{1}{\alpha_j}\frac{(r_{j+1},(r_{j+1}-r_j))}{(A p_j,p_j)}=\frac{(r_{j+1},r_{j+1})}{(r_j,r_j)}.$$

Объединяя эти соотношения, получаем искомый алгоритм.

Практическая часть

Задача 1. Генерация СЛАУ



```
0.0074221 0.0011597 0.0011597 0 0
0.0011597 0.0111332 0.0011597 0.0011597 0
0.0011597 0.0011597 0.0148442 0.0011597 0.0011597
0 0.0011597 0.0011597 0.0129887 0.0011597
0 0 0.0011597 0.0011597 0.0111332


0 0 0 0 5.77774e-05 2.28635e-05 2.71672e-05 2.68982e-06 1.34491e-06
0 0 0 2.28635e-05 0.000127982 3.28158e-05 2.93191e-05 2.68982e-06 0
0 0 2.71672e-05 3.28158e-05 0.00022573 3.49677e-05 3.14709e-05 0 0
0 2.68982e-06 2.93191e-05 3.49677e-05 0.00017274 2.93191e-05 0 0 0
1.34491e-06 2.68982e-06 3.14709e-05 2.93191e-05 0.000126637 0 0 0 0
```

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

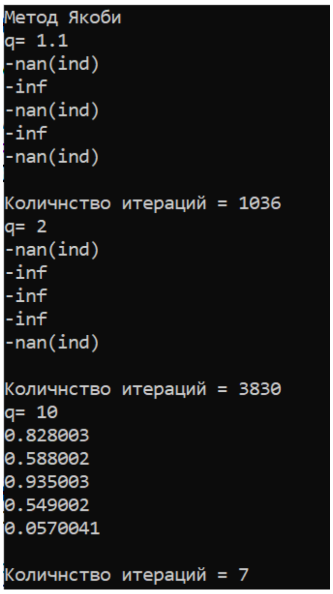


Рисунок 2*. Пример выполнения программы метода Якоби*

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 $\omega \in (0,2)$.
2) С использованием написанной программы исследовать зависимость числа итераций метода SOR от параметров *q* и ω. При сравнении предусмотреть частный случай $\omega=1$, соответствующий методу Гаусса-Зейделя.

```
Метод SOR
q= 1.1
W               Количество итераций
0.1             147
0.2             79
0.3             55
0.4             43
0.5             35
0.6             30
0.7             26
0.8             23
0.9             22
1               21
1.1             28
1.2             56
1.3             185
1.4             284
```

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

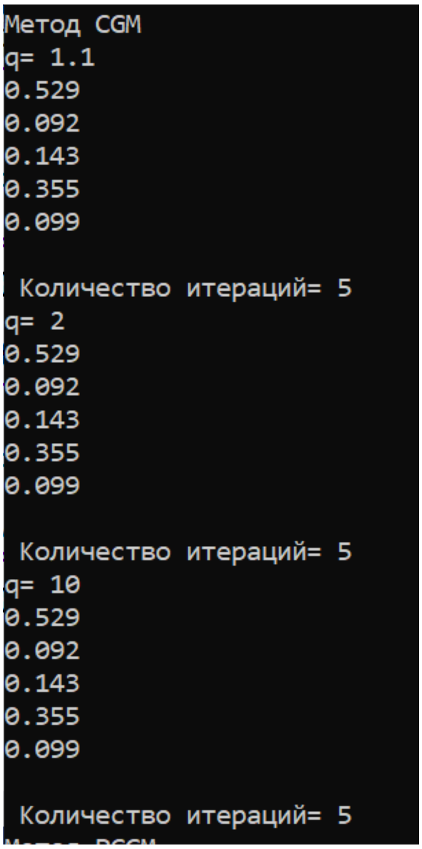
```
q= 2
W               Количество итераций
0.1             100
0.2             53
0.3             36
0.4             27
0.5             21
0.6             18
0.7             15
0.8             13
0.9             12
1               12
1.1             17
1.2             25
1.3             43
1.4             120
q= 10
W               Количество итераций
0.1             88
0.2             45
0.3             30
0.4             22
0.5             17
0.6             14
0.7             11
0.8             9
0.9             7
1               6
1.1             8
1.2             10
1.3             14
1.4             18
```

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 A_i, 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.



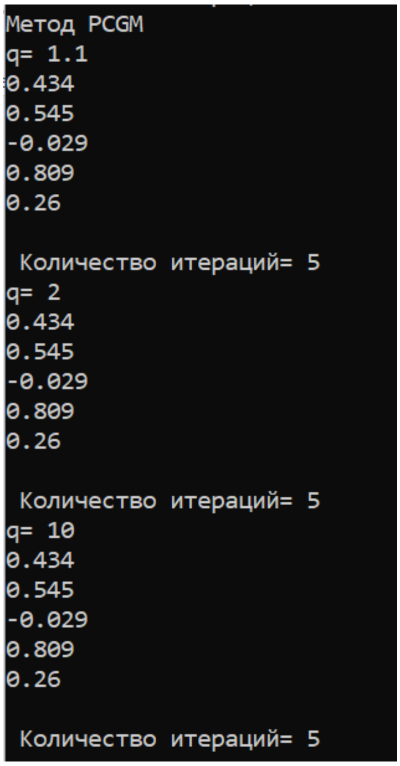
```
Метод CGM
q= 1.1
0.529
0.092
0.143
0.355
0.099

 Количество итераций= 5
q= 2
0.529
0.092
0.143
0.355
0.099

 Количество итераций= 5
q= 10
0.529
0.092
0.143
0.355
0.099

 Количество итераций= 5
```

Рисунок 5. Решения для матриц методом CGM


```
Метод PCGM
q= 1.1
0.434
0.545
-0.029
0.809
0.26

 Количество итераций= 5
q= 2
0.434
0.545
-0.029
0.809
0.26

 Количество итераций= 5
q= 10
0.434
0.545
-0.029
0.809
0.26

 Количество итераций= 5
```

Рисунок 6. Решения для матриц методом PCGM

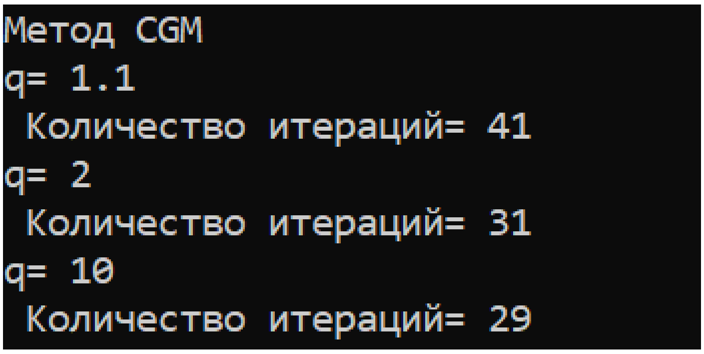

```
Метод CGM
q= 1.1
 Количество итераций= 41
q= 2
 Количество итераций= 31
q= 10
 Количество итераций= 29
```

Рисунок 7. *Решения для соответствующих матриц методом CGM*

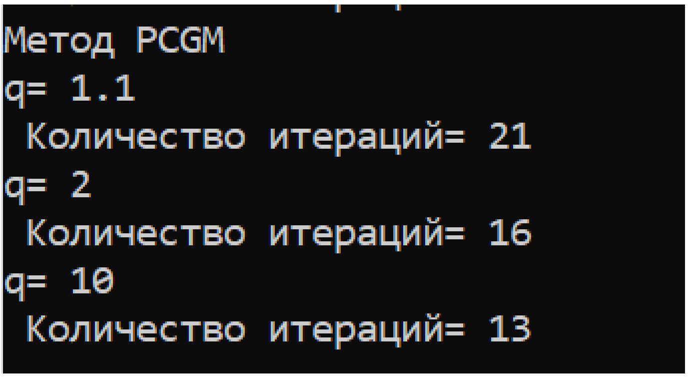

```
Метод PCGM
q= 1.1
 Количество итераций= 21
q= 2
 Количество итераций= 16
q= 10
 Количество итераций= 13
```

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;


        }
}
```