БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №3

**Итерационные методы решения СЛАУ**

**Выполнил:**

Крючков Василий

2 курс 9 группа

**Преподаватель:**

Горбачева Ю.Н.

Минск, 2021

**Постановка задачи**

Написать и отладить программу численного решения систем линейных алгебраических уравнений с квадратной матрицей порядка n

1) методом градиентного спуска

2) методом релаксации.

Предусмотреть сообщение о выходе из итерационного процесса из-за превышения допустимого максимального количества итераций (). В качестве критерия остановки итерационного процесса использовать .

**Краткие теоретические сведения**

Метод градиентного спуска:

–невязка

Метод релаксации:

Теорема:

Если и , то метод релаксации сходится.

Абсолютная погрешность вычислялась по формуле:

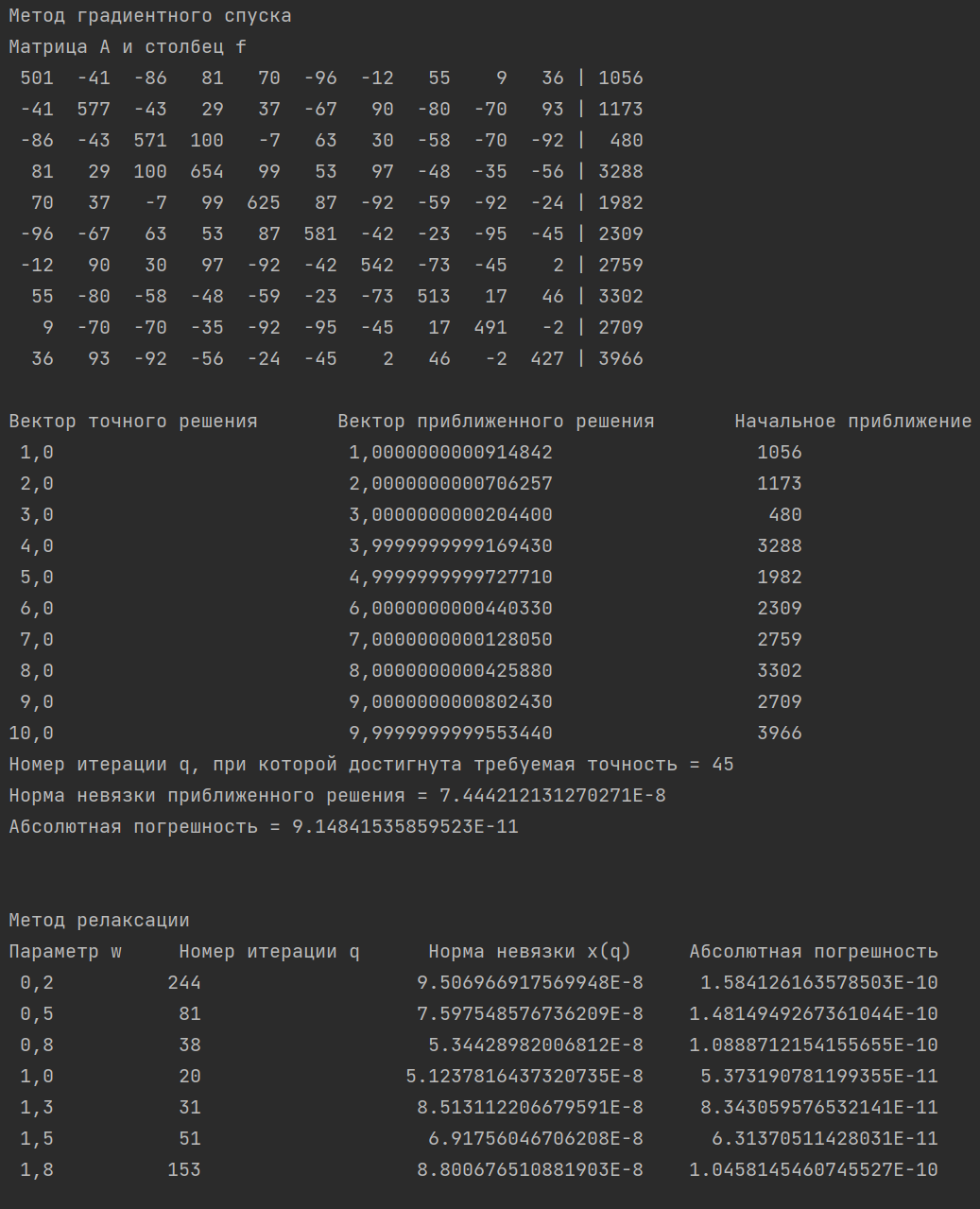
, где, , - полученное приближенное решение.

**Листинг программы**

**Lab.java**

import java.util.Formatter;  
import java.util.Random;  
  
public class Lab {  
 public static void main(String[] args) {  
 Program pr = new Program();  
 try {  
 pr.base();  
 }catch (Error e){  
 System.*out*.println(e.getMessage());  
 }  
 }  
}  
class Program {  
 static int *kIt* = 0;  
 void base() throws Error {  
 final double e = 1e-7;  
 final int kMax = 5000;  
 final int n = 10;  
 double[][] mtrA = generateMatrix(n);  
 double [] exactColumnX = new double[n];  
  
 //Точное решение слау  
 for (int i=0; i<n; i++)  
 exactColumnX[i]=i+1;  
  
 double [] columnF = generateColumnF(mtrA,exactColumnX);  
 double [] initialColumnX = columnF.clone();  
 double [] columnX = gradientDescentMethod(mtrA,columnF,initialColumnX,kMax,e);  
  
  
 //Абсолютная погрешность  
 double[] tempColumn = new double[columnX.length];  
 for (int i = 0;i<n; i++)  
 tempColumn[i]= exactColumnX[i]-columnX[i];  
 double accuracy = norm(tempColumn);  
  
 //Ввывод результатов  
 System.*out*.println("Метод градиентного спуска");  
 System.*out*.println("Матрица А и столбец f");  
 for (int i=0; i< mtrA.length; i++) {  
 Formatter f = new Formatter();  
 for (int j=0; j<mtrA.length; j++) {  
 f.format("%4.0f ", mtrA[i][j]);  
 }  
 f.format("| %4.0f%n", columnF[i]);  
 System.*out*.print(f);  
 }  
 System.*out*.println();  
 System.*out*.println("Вектор точного решения Вектор приближенного решения Начальное приближение");  
 Formatter f = new Formatter();  
 for (int j=0; j<mtrA.length; j++)  
 f.format("%4.1f %43.16f %20.0f%n", exactColumnX[j], columnX[j],initialColumnX[j]);  
 System.*out*.print(f);  
 System.*out*.println("Номер итерации q, при которой достигнута требуемая точность = " +*kIt*);  
 System.*out*.println("Норма невязки приближенного решения = "+secondNorm(discrepancy(mtrA,columnF,columnX)));  
 System.*out*.println("Абсолютная погрешность = " +accuracy);  
 System.*out*.println();  
 System.*out*.println();  
 System.*out*.println("Метод релаксации");  
 System.*out*.println("Параметр w Номер итерации q Норма невязки x(q) Абсолютная погрешность");  
 f = new Formatter();  
 *kIt*=0;  
 double w = 0.2;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 0.5;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 0.8;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 1.;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 1.3;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 1.5;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 *kIt*=0;  
 w = 1.8;  
 columnX = relaxationMethod(w, mtrA, columnF, initialColumnX, kMax, e);  
 for (int i = 0; i < n; i++)  
 tempColumn[i] = exactColumnX[i] - columnX[i];  
 accuracy = norm(tempColumn);  
 f.format("%4.1f %12d %37s %23s%n", w, *kIt*, secondNorm(discrepancy(mtrA, columnF, columnX)), accuracy);  
  
 System.*out*.print(f);  
 }  
  
 //Генерация матрицы  
 double[][] generateMatrix(int n){  
 final int k = 6;  
 double[][] mtrA = new double[n][n];  
 Random rand = new Random();  
 for (int i = 0;i<n; i++)  
 for (int j = 0; j < n; j++) {  
 if (i < j) {  
 mtrA[i][j] = -100 + rand.nextInt(200+1);  
 mtrA[j][i] = mtrA[i][j];  
 }  
 }  
 for (int i = 0;i<n; i++) {  
 double dTemp = 0.;  
 for (int j = 0; j < n; j++) {  
 dTemp += Math.*abs*(mtrA[i][j]);  
 }  
 mtrA[i][i] = dTemp + k + rand.nextInt(9\*k+1);  
 }  
 return mtrA;  
 }  
  
 //Вектор f  
 double[] generateColumnF(double[][] mtrA,double [] exactColumnX){  
 return multiplicationMatrixColumn(mtrA,exactColumnX);  
 }  
  
 //Метод градиентного спуска  
 double[] gradientDescentMethod(double[][] mtrA,double[] columnF,double [] initialColumnX, int kMax, double e) throws Error {  
 double [] columnX = initialColumnX.clone();  
 int k =0;  
 double[] r = discrepancy(mtrA,columnF,columnX);  
 while (secondNorm(r)>e){  
 k++;  
 *kIt*++;  
 if(k>kMax)  
 throw new Error("Привышен параметр k max");  
 double gradF = scalarMultiplication(r,r)/scalarMultiplication(multiplicationMatrixColumn(mtrA,r),r);  
 for(int i=0;i<columnX.length;i++)  
 columnX[i]= columnX[i] - r[i]\*gradF;  
 r = discrepancy(mtrA,columnF,columnX);  
 }  
 return columnX;  
 }  
  
  
 //Метод релаксации  
 double[] relaxationMethod(double w,double[][] mtrA,double[] columnF,double [] initialColumnX, int kMax, double e) throws Error{  
 double [] columnX = initialColumnX.clone();  
 int k = 0;  
 while (secondNorm(discrepancy(mtrA,columnF,columnX))>e){  
 if(k>kMax)  
 throw new Error("Привышен параметр k max");  
 for(int i=0;i<columnX.length;i++) {  
 double dTemp = 0.;  
 for(int j=0; j<columnX.length;j++) {  
 if(i!=j)  
 dTemp +=mtrA[i][j]\*columnX[j];  
 }  
 columnX[i]=(1-w)\*columnX[i]+(w/mtrA[i][i])\*(columnF[i]-dTemp);  
 }  
 *kIt*++;  
 k++;  
 }  
 return columnX;  
 }  
  
  
 //Подсчет второй нормы  
 double secondNorm(double []columnX){  
 double dTemp =0.;  
 for (double x : columnX) dTemp += x \* x;  
 return Math.*sqrt*(dTemp);  
 }  
 //Подсчет нормы  
 double norm(double []columnX){  
 double dTemp =0.;  
 for (double x : columnX)  
 if(Math.*abs*(x)>dTemp)  
 dTemp = Math.*abs*(x);  
 return dTemp;  
 }  
  
 //Подсчет скалярного произведение векторов  
 double scalarMultiplication(double []columnX1,double []columnX2){  
 double temp =0.;  
 for (int i = 0;i<columnX1.length;i++){  
 temp += columnX1[i]\*columnX2[i];  
 }  
 return temp;  
 }  
  
 //Подсчет невязки  
 double[] discrepancy(double[][] mtrA,double[] columnF,double [] columnX){  
 double[] r = multiplicationMatrixColumn(mtrA,columnX);  
 for(int i=0;i<r.length;i++)  
 r[i]= r[i] - columnF[i];  
 return r;  
 }  
  
  
 double[] multiplicationMatrixColumn(double[][] mtrA,double[] columnF){  
 double [] columnX = new double[columnF.length];  
 for(int i = 0; i < mtrA.length; i++)  
 for(int j = 0; j < mtrA.length; j++)  
 columnX[i] += columnF[j] \* mtrA[i][j];  
 return columnX;  
 }  
}

**Результаты**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр ω | Номер итерации q, при которой достигнута требуемая точность |  |  |
| 0.2 | 244 | 9.506966917569948E-8 | 1.584126163578503E-10 |
| 0.5 | 81 | 7.597548576736209E-8 | 1.4814949267361044E-10 |
| 0.8 | 38 | 5.34428982006812E-8 | 1.0888712154155655E-10 |
| 1 | 20 | 5.1237816437320735E-8 | 5.373190781199355E-11 |
| 1.3 | 31 | 8.513112206679591E-8 | 8.343059576532141E-11 |
| 1.5 | 51 | 6.91756046706208E-8 | 6.31370511428031E-11 |
| 1.8 | 153 | 8.800676510881903E-8 | 1.0458145460745527E-10 |

**Выводы**

Итерационные методы такие, как метод градиентного спуска и метод релаксации, являются эффективными в случае положительно определённой симметрической матрицы системы линейных уравнений для нахождение приближенного решения с заранее заданной точностью. В результате исследования сходимости метода релаксации в зависимости от параметра релаксации для ω положительно определённой симметрической матрицы с диагональным преобладанием оптимальное значение ω = 1.