**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Лабораторная работа №3 по курсу “ВМА”

“Метод Зейделя”

Вариант №3

Выполнил: Ёда Никита

3 курс, 6(а) группа

Преподаватель: Будник А.М.

2023

Содержание:

Постановка задачи....................................................................................2

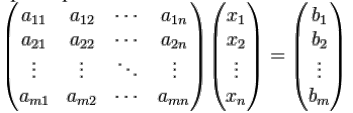
Алгоритм решения...................................................................................3

Листинг программы..................................................................................7

Результат и его анализ............................................................................18

**Постановка задачи**

В данной лабораторной работе требуется решить систему линейных алгебраических уравнений Ах=b, используя метод Зейделя



План решения:

Используя алгоритм метода Гаусса (с выбором гл. эл. по столбцам) из п.1 вычислить:

- метод Гаусса (с выбором гл. эл. по столбцам) (1)

- ­определитель матрицы A (2)

- обратную матрицу A^(-1) (3)

- число обусловленности v(A) (4)

- метод Зейделя (5)

Для метода Зейделя, найти:

- вектор невязки (6)

- матрицу невязки (7)

- количество итераций (8)

- обосновать сходимость итерационного метода (9)

- сравнить методы по точности и экономике

**Алгоритм решения**

*1. Метод Гаусса с выбором главного элемента по столбцу:*

1.1. Нахожу максимальный элемент по модулю в первом столбце. Для этого я использовал цикл от 0 до n-1, n – размер матрицы. Максимальный элемент нахожу в соответствии с номером итерации (итерация от 0 до n-1). Если 0, то первый столбец и т.д.

1.2. Переставляю строку с максимальным элементом с первой строкой (перестоновка также зависит от номера итерации, если 0, то первая строка и т.д.)

1.3. Нахожу масштабирующий множитель по формуле:

ui = , где k - шаг, i - строка, aij - матрица А

1.4. Преобразую матрицу по формуле:

ij = ij - kj \* , где k - шаг, i - строка, j - столбец, aij - матрица А

*2. Вычисление определителя матрицы A:*

Определитель mtrA (преобразованной матрицы после метода Гаусса) будет равен произведению всех элементов на диагонали. Нужно учитывать количество итераций. В моём случае, если count % 2 == 1, то умножаем на -1.

1. *Вычисление обратной матрицы A^-1:*

Для того, чтобы вычислить обратную матрицу я добавляю к исходной матрице - единичную матрицу. Таким образом:

0,3857 -0,0508 0,0102 0,0203 0,0711 | 1,0000 0,0000 0,0000 0,0000 0,0000

0,0528 0,6039 0,0000 -0,0406 0,0406 | 0,0000 1,0000 0,0000 0,0000 0,0000

0,0305 0,0000 0,4852 -0,1421 0,0812 | 0,0000 0,0000 1,0000 0,0000 0,0000

-0,0609 0,1279 0,0000 0,4711 -0,0203 | 0,0000 0,0000 0,0000 1,0000 0,0000

0,2538 0,0000 0,0914 0,0102 0,5684 | 0,0000 0,0000 0,0000 0,0000 1,0000

Теперь, необходимо привести левую матрицу, чтобы из неё получилась единичная матрица, а из единичной матрицы (та, которая справа), получится обратная матрица.

*4. Вычисление числа обусловленности v(A):*

Для того, чтобы найти число обусловленности надо умножить норму 1, изначальной матрицы, на норму 1, обратной матрицы.

Норма 1 определяется как сумма абсолютных значений элементов каждого столбца матрицы. Для матрицы размера m x n норма 1 равна максимальной сумме абсолютных значений столбцов.

1. *Метод Зейделя*

*5.1*. Первым делом я перепишу изначальную матрицу в систему уравнений, тогда система уравнений будет выглядеть следующим образом:

(0.3857\*x1) + (-0.0508\*x2) + (0.0102\*x3) + (0.0203\*x4) + (0.0711\*x5) = 0.7613

(0.0528\*x1) + (0.6039\*x2) + (0.0\*x3) + (-0.0406\*x4) + (0.0406\*x5) = -0.8709

(0.0305\*x1) + (0.0\*x2) + (0.4852\*x3) + (-0.1421\*x4) + (0.0812\*x5) = 3.2074

(-0.0609\*x1) + (0.1279\*x2) + (0.0\*x3) + (0.4711\*x4) + (-0.0203\*x5) = -1.829

(0.2538\*x1) + (0.0\*x2) + (0.0914\*x3) + (0.0102\*x4) + (0.5684\*x5) = 2.9537

*5.2*. Теперь я буду выражать х:

0.3857\*x1 = 0.7613 + (-0.0508\*x2) + (0.0102\*x3) + (0.0203\*x4) + (0.0711\*x5)

0.6039\*x2 = -0.8709 + (0.0528\*x1) + (0.0\*x3) + (-0.0406\*x4) + (0.0406\*x5)

0.4852\*x3 = 3.2074 + (0.0305\*x1) + (0.0\*x2) + (-0.1421\*x4) + (0.0812\*x5)

0.4711\*x4 = -1.829 + (-0.0609\*x1) + (0.1279\*x2) + (0.0\*x3) + (-0.0203\*x5)

0.5684\*x5 = 2.9537 + (0.2538\*x1) + (0.0\*x2) + (0.0914\*x3) + (0.0102\*x4)

*5.3*. Поделю на коэфицент перед x:

x1 = (0.7613 + (-0.0508\*x2) + (0.0102\*x3) + (0.0203\*x4) + (0.0711\*x5)) / 0.3857

x2 = (-0.8709 + (0.0528\*x1) + (0.0\*x3) + (-0.0406\*x4) + (0.0406\*x5)) / 0.6039

x3 = (3.2074 + (0.0305\*x1) + (0.0\*x2) + (-0.1421\*x4) + (0.0812\*x5)) / 0.4852

x4 = (-1.829 + (-0.0609\*x1) + (0.1279\*x2) + (0.0\*x3) + (-0.0203\*x5)) / 0.4711

x5 = (2.9537 + (0.2538\*x1) + (0.0\*x2) + (0.0914\*x3) + (0.0102\*x4)) / 0.5684

*5.4.* Вычислю нулевую норму. В нашем случае x0=b, отсюда:

x0 = 0.7613

x1 = -0.8709

x2 = 3.2074

x3 = -1.829

x4 = 2.9537

*5.5.* Теперь буду вычислять приближения:

(k) (k-1) (k-1) (k-1) (k-1)

x1 = (0.7613 + (-0.0508\*x2) + (0.0102\*x3) + (0.0203\*x4) + (0.0711\*x5)) / 0.3857

(k) (k) (k-1) (k-1) (k-1)

x2 = (-0.8709 + (0.0528\*x1) + (0.0\*x3) + (-0.0406\*x4) + (0.0406\*x5)) / 0.6039

(k) (k) (k) (k-1) (k-1)

x3 = (3.2074 + (0.0305\*x1) + (0.0\*x2) + (-0.1421\*x4) + (0.0812\*x5)) / 0.4852

(k) (k) (k) (k) (k-1)

x4 = (-1.829 + (-0.0609\*x1) + (0.1279\*x2) + (0.0\*x3) + (-0.0203\*x5)) / 0.4711

(k) (k) (k) (k) (k)

x5 = (2.9537 + (0.2538\*x1) + (0.0\*x2) + (0.0914\*x3) + (0.0102\*x4)) / 0.5684

Общая формула для метода Зейделя:

for (int i = 0; i < a.length; i++)

x2[i] = b[i]

for (int j = 0; j < a.length; j++)

i != j

x2[i] += (-1) \* a[i][j] \* x2[j], j<i

x2[i] += (-1) \* a[i][j] \* x[j], j>i

x2[i] /= a[i][j]

Где a[i][j] - изначальная матрица, x2 - приближение (k+1), x- приближение (k)

*5.6.* Считаю погрешность:

Формула для погрешности: σ = |x2 - x| / |x2|

*5.7.* Проверяю условие:

Критерий останова итерационного процесса: ||xk+1  - xk|| <= e, где e = 10^-5

d[i] = x2[i] - x[i]

||d|| =

1. *Вектор невязки:*

Вектор невязки будет равен погрешности при последнем шаге

1. *Нахождение матрицы невязки:*

Для нахождения матрицы невязки, нужно воспользоваться формулой:

R=A^-1\*A-E, где A-исходная матрица, A^-1 - обратная матрица, E - единичная матрица

1. *Количество итераций:*

Просто выведу количество итераций из метода Зейделя

1. *Обосновать сходимость итерационного метода Зейделя:*

Критерий диагонального преобладания: Проверяю матрицу системы на диагональное преобладание. Если каждый диагональный элемент матрицы по модулю больше суммы модулей всех остальных элементов в соответствующей строке, то метод Зейделя сходится для данной системы

**Листинг программы**

import java.text.DecimalFormat;  
  
public class Main {  
 public static void main(String[] args) {  
 try {  
 double[][] mtrA = {{0.3857, -0.0508, 0.0102, 0.0203, 0.0711},  
 {0.0528, 0.6039, 0.0000, -0.0406, 0.0406},  
 {0.0305, 0.0000, 0.4852, -0.1421, 0.0812},  
 {-0.0609, 0.1279, 0.0000, 0.4711, -0.0203},  
 {0.2538, 0.0000, 0.0914, 0.0102, 0.5684}};  
 int mtrALength = mtrA.length;  
  
 double[] vecB = {0.7613, -0.8709, 3.2074, -1.8290, 2.9537};  
 int vecBLength = vecB.length;;  
  
  
 // Метод Гаусса  
 System.out.println("|Метод Гаусса|");  
 double[][] copyMtrA1 = copyMtr(mtrA, mtrALength);  
 double[] copyVecB1 = copyVec(vecB, vecBLength);  
 int numOfStepsInGaussMet = 0; // Кол-во итераций  
 gaussMet(copyMtrA1, copyVecB1, numOfStepsInGaussMet, mtrALength, vecBLength);  
  
  
 // Нахождение det(mtrA)  
 System.out.println("\n|Нахождение det(mtrA)|");  
 double detMtrA = detMtr(copyMtrA1, numOfStepsInGaussMet, mtrALength);  
 System.out.println("det(mtrA) = " + detMtrA);  
  
  
 // Нахождение обратной матрицы  
 System.out.print("\n|Нахождение обратной матрицы|");  
 double[][] singleMtr = new double[mtrALength][mtrALength]; // Создание единичной м-цы  
 for (int i = 0; i < mtrALength; i++) {  
 singleMtr[i][i] = 1;  
 }  
 double[][] copyMtrA2 = copyMtr(mtrA, mtrALength);  
 double[][] reverseMtr = reverseMtr(copyMtrA2, singleMtr, mtrALength);  
  
  
 // Нахождения числа обусловленности  
 System.out.print("\n|Нахождения числа обусловленности|");  
 double[][] copyMtrA3 = copyMtr(mtrA, mtrALength);  
 System.out.println("\nЧисло обусловленности: " + (mtrNorm(copyMtrA3, mtrALength, "mtrA") \* mtrNorm(reverseMtr, mtrALength, "reverseMtr")));  
  
  
 // Метод Зейделя  
 System.out.print("\n|Метод Зейделя|");  
 double[][] copyMtrA4 = copyMtr(mtrA, mtrALength);  
 double[] copyVecB2 = copyVec(vecB, vecBLength);  
 double[] massX = seidelMet(copyMtrA4, copyVecB2, mtrALength, vecBLength);  
 int massXLength = massX.length;  
 System.out.println("\nВектор решений x:");  
 printVecWithoutRound(massX, massXLength);  
  
  
 // Матрица невязки  
 System.out.print("\n\n|Матрица невязки|");  
 double[][] copyMtrA5 = copyMtr(mtrA, mtrALength);  
 printMtrWithoutRounding(calcResMtr(copyMtrA5, reverseMtr), mtrALength);  
  
  
 // Сходимость метода зейделя  
 System.out.println("\n|Сходимость метода Зейделя|");  
 double[][] copyMtrA6 = copyMtr(mtrA, mtrALength);  
 convergence(copyMtrA6, mtrALength);  
 }  
 catch (ArithmeticException e) {  
 System.out.println("Ошибка: " + e.getMessage());  
 }  
 }  
  
  
 // Глубокое копирование м-цы  
 public static double[][] copyMtr(double[][] mtr, int mtrLength) {  
 double[][] copyMtrA = new double[mtrLength][mtrLength];  
  
 for (int i = 0; i < mtrLength; i++) {  
 for (int j = 0; j < mtrLength; j++) {  
 copyMtrA[i][j] = mtr[i][j];  
 }  
 }  
  
 return copyMtrA;  
 }  
  
  
 // Глубокое копирование в-ра  
 public static double[] copyVec(double[] vec, int vecLength) {  
 double[] copyVecB = new double[vecLength];  
  
 for (int i = 0; i < vecLength; i++) {  
 copyVecB[i] = vec[i];  
 }  
  
 return copyVecB;  
 }  
  
  
 // Метод Гаусса  
 public static void gaussMet (double[][] mtrA, double[] vecB, int numOfStepsInGaussMet, int mtrALength, int vecBLength) {  
 System.out.print("До:");  
 printMtrWithVec(mtrA, vecB, mtrALength, vecBLength);  
  
 while (numOfStepsInGaussMet != mtrALength - 1) {  
 // Поиск максимального элемента  
 double maxEl = Math.abs(mtrA[numOfStepsInGaussMet][numOfStepsInGaussMet]);  
 for (int i = numOfStepsInGaussMet; i < mtrALength; i++) {  
 if (Math.abs(mtrA[i][numOfStepsInGaussMet]) > maxEl) {  
 maxEl = Math.abs(mtrA[i][numOfStepsInGaussMet]);  
 }  
 }  
  
  
 // Перестановка строк  
 for (int i = numOfStepsInGaussMet; i < mtrALength; i++) {  
 if (Math.abs(mtrA[i][numOfStepsInGaussMet]) == maxEl) {  
 for (int j = 0; j < mtrALength; j++) {  
 // Для mtrA  
 double temp = mtrA[numOfStepsInGaussMet][j];  
 mtrA[numOfStepsInGaussMet][j] = mtrA[i][j];  
 mtrA[i][j] = temp;  
 }  
  
 // Для vecA  
 double temp = vecB[numOfStepsInGaussMet];  
 vecB[numOfStepsInGaussMet] = vecB[i];  
 vecB[i] = temp;  
 }  
 }  
  
  
 // Высчитываем масштабирующие множители  
 double[] massU = new double[mtrALength - 1];  
 for (int i = numOfStepsInGaussMet + 1; i < mtrALength; i++) {  
 massU[i - 1] = mtrA[i][numOfStepsInGaussMet] / mtrA[numOfStepsInGaussMet][numOfStepsInGaussMet];  
 }  
  
  
 // Преобразование матрицы  
 for (int i = numOfStepsInGaussMet + 1; i < mtrALength; i++) {  
 for (int j = numOfStepsInGaussMet; j < mtrALength; j++) {  
 mtrA[i][j] = mtrA[i][j] - (mtrA[numOfStepsInGaussMet][j] \* massU[i - 1]); // Для mtrA  
 }  
 vecB[i] = vecB[i] - (vecB[numOfStepsInGaussMet] \* massU[i - 1]); // Для vecA  
 }  
  
 numOfStepsInGaussMet++;  
 }  
  
 System.out.print("\nПосле:");  
 printMtrWithVec(mtrA, vecB, mtrALength, vecBLength);  
  
 System.out.println("\nМатрица приведена к треугольному виду!");  
 }  
  
  
 // Нахождение определителя м-цы (используя метод Гаусса)  
 public static double detMtr(double[][] mtrA, int numOfStepsInGaussMet, int mtrALength) {  
 double determinant = 1.0;  
 for (int i = 0; i < mtrALength; i++) {  
 determinant \*= mtrA[i][i];  
 }  
  
 if (numOfStepsInGaussMet % 2 == 1) {  
 return determinant \* -1;  
 }  
 else {  
 return determinant;  
 }  
 }  
  
  
 // Нахождение обратной м-цы  
 public static double[][] reverseMtr(double[][] copyMtr, double[][] singleMtr, int mtrALength) {  
 System.out.print("\nДо:");  
 printMtrWithMtr(copyMtr, singleMtr);  
  
 // Преобразование в единичную матрицу  
 for (int i = 0; i < mtrALength; i++) {  
 double pivot = copyMtr[i][i];  
  
 // Деление строки на pivot  
 for (int j = 0; j < mtrALength; j++) {  
 copyMtr[i][j] /= pivot; // Для cMtr  
 singleMtr[i][j] /= pivot; // Для singleMtr  
 }  
  
 // Вычитание других строк для получения нулей под главной диагональю  
 for (int k = 0; k < mtrALength; k++) {  
 if (k != i) {  
 double factor = copyMtr[k][i];  
 for (int j = 0; j < mtrALength; j++) {  
 copyMtr[k][j] -= factor \* copyMtr[i][j]; // Для cMtr  
 singleMtr[k][j] -= factor \* singleMtr[i][j]; // Для singleMtr  
 }  
 }  
 }  
 }  
  
 System.out.print("\nПосле:");  
 printMtrWithMtr(copyMtr, singleMtr);  
  
 System.out.print("\nОбратная м-ца:");  
 printMtr(singleMtr, singleMtr.length);  
  
 return singleMtr;  
 }  
  
  
 // Норма м-цы  
 public static double mtrNorm(double[][] mtrA, int mtrALength, String name) {  
 double maxSum = 0.0;  
  
 for (int j = 0; j < mtrALength; j++) {  
 double columnSum = 0.0;  
 for (int i = 0; i < mtrALength; i++) {  
 columnSum += Math.abs(mtrA[i][j]);  
 }  
 maxSum = Math.max(maxSum, columnSum);  
 }  
 System.out.print("\n||" + name + "|| = " + maxSum);  
  
 return maxSum;  
 }  
  
  
 // Вывод м-цы  
 public static void printMtr(double[][] mtrA, int mtrALength) {  
 System.out.println("\nВывод м-цы:");  
  
 DecimalFormat decimalFormat = new DecimalFormat("#.####");  
  
 for (int i = 0; i < mtrALength; i++) {  
 for (int j = 0; j < mtrALength; j++) {  
 System.out.print(decimalFormat.format(mtrA[i][j]));  
  
 for (int k = 0; k < 15 - decimalFormat.format(mtrA[i][j]).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
 System.out.println();  
 }  
 }  
  
  
 // Вывод м-цы и в-ра  
 public static void printMtrWithVec(double[][] mtrA, double[] vecB, int mtrALength, int vecBLength) {  
 System.out.println("\nВывод м-цы и в-ра:");  
  
 DecimalFormat decimalFormat = new DecimalFormat("#.####");  
  
 for (int i = 0; i < mtrALength; i++) {  
 for (int j = 0; j < mtrALength; j++) {  
 System.out.print(decimalFormat.format(mtrA[i][j]));  
  
 for (int k = 0; k < 15 - decimalFormat.format(mtrA[i][j]).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
 System.out.println("|\t" + decimalFormat.format(vecB[i]));  
 }  
 }  
  
  
 // Вывод м-цы и м-цы  
 public static void printMtrWithMtr(double[][] mtr1, double[][] mtr2) {  
 System.out.println("\nВывод м-цы и м-цы:");  
  
 DecimalFormat decimalFormat = new DecimalFormat("#.####");  
  
 for (int i = 0; i < mtr1.length; i++) {  
 for (int j = 0; j < mtr1.length; j++) {  
 System.out.print(decimalFormat.format(mtr1[i][j]));  
 for (int k = 0; k < 15 - decimalFormat.format(mtr1[i][j]).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
  
 System.out.print("|\t");  
  
 for (int j = 0; j < mtr2.length; j++) {  
 System.out.print(decimalFormat.format(mtr2[i][j]));  
 for (int k = 0; k < 15 - decimalFormat.format(mtr2[i][j]).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
  
 System.out.println();  
 }  
 }  
  
  
 // Метод Зейделя  
 public static double[] seidelMet(double[][] mtrA, double[] vecB, int mtrALength, int vecBLength) {  
 // Вывод изначальной матрицы mtrA  
 printMtrWithVec(mtrA, vecB,mtrALength, vecBLength);  
  
  
 // 1-ый шаг: Перепишем нашу матрицу в виде системы уравнений  
 System.out.println("\nВид уравнения:");  
 for (int i = 0; i < mtrALength; i++) {  
 for (int j = 0; j < mtrALength; j++) {  
 if (j != mtrALength - 1) {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ") + ");  
 } else {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ") = ");  
 }  
 }  
 System.out.println(vecB[i]);  
 }  
  
  
 // 2-ой шаг: Выразим из каждого уравнения неизвестный x, который соответствует номеру строки  
 System.out.println("\nВыразим x:");  
 for (int i = 0; i < mtrALength; i++) {  
 System.out.print(mtrA[i][i] + "\*x" + (i + 1) + " = " + vecB[i] + " + ");  
 for (int j = 0; j < mtrALength; j++) {  
 if (i != j) {  
 if (i == mtrALength - 1 && j == mtrALength - 2) {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ")");  
 } else if ((j != mtrALength - 1)) {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ") + ");  
 } else {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ")");  
 }  
 }  
 }  
 System.out.println();  
 }  
  
  
 // 3-ий шаг: Поделим на коэфицент перед x  
 System.out.println("\nПоделим на коэфицент перед x:");  
 for (int i = 0; i < mtrALength; i++) {  
 System.out.print("x" + (i + 1) + " = (" + vecB[i] + " + ");  
 for (int j = 0; j < mtrALength; j++) {  
 if (i != j) {  
 if (i == mtrALength - 1 && j == mtrALength - 2) {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ")");  
 } else if ((j != mtrALength - 1)) {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ") + ");  
 } else {  
 System.out.print("(" + mtrA[i][j] + "\*x" + (j + 1) + ")");  
 }  
 }  
 }  
 System.out.println(") / " + mtrA[i][i]);  
 }  
  
  
 // 4-ый шаг: Вычислим нулевое приближение  
 System.out.println("\nВычислим нулевое приближение:");  
 double[] massX = new double[mtrALength];  
 int massXLength = massX.length;  
  
 for (int i = 0; i < mtrALength; i++) {  
 massX[i] = vecB[i];  
 System.out.println("x" + i + " = " + vecB[i]);  
 }  
  
  
 // 4.1 шаг: Вычисляем приближения и погрешность  
 System.out.println("\nВычисляем приближения и погрешность:");  
 double[] massX2 = new double[mtrALength];  
 int countK = 0;  
  
  
 // Создание таблицы  
 createTable(massXLength);  
  
  
 // Вывод K и σ для певрого шага:  
 System.out.print(countK);  
 for (int k = 0; k < 15 - Integer.toString(countK).length(); k++) {  
 System.out.print(" ");  
 }  
 for (int i = 0; i < massXLength; i++) {  
 printMeaning(massX[i]);  
 }  
 String str = "-";  
 for (int i = 0; i < massXLength; i++) {  
 System.out.print(str);  
 for (int k = 0; k < 30 - str.length(); k++) {  
 System.out.print(" ");  
 }  
 }  
 System.out.println();  
  
 double[] massApproximation = new double[massXLength];  
 double eps = Math.pow(10, -5);  
 double[] d = new double[massXLength];  
 double normD;  
 boolean exit = false;  
  
 while (!exit) {  
 countK++;  
  
 System.out.print(countK);  
 for (int k = 0; k < 15 - Integer.toString(countK).length(); k++) {  
 System.out.print(" ");  
 }  
  
 // Считаем x  
 for (int i = 0; i < mtrALength; i++) {  
 massX2[i] = vecB[i];  
 for (int j = 0; j < mtrALength; j++) {  
 if (i != j) {  
 if (j < i) {  
 massX2[i] += (-1) \* mtrA[i][j] \* massX2[j + 1 - 1];  
 }  
 else if (j > i) {  
 massX2[i] += (-1) \* mtrA[i][j] \* massX[j + 1 - 1];  
 }  
 }  
 }  
 massX2[i] /= mtrA[i][i];  
 printMeaning(massX2[i]);  
 }  
  
  
 // 6-й шаг: Считаем погрешность σ  
 double approximation;  
 for (int i = 0; i < massXLength; i++) {  
 approximation = Math.abs(massX2[i] - massX[i]) / Math.abs(massX2[i]);  
 massApproximation[i] = approximation;  
  
 System.out.print(approximation);  
 for (int k = 0; k < 30 - Double.toString(approximation).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
  
  
 // 7-ой шаг: Проверка условия выхода  
 normD = 0;  
 for (int j = 0; j < massXLength; j++) {  
 d[j] = massX2[j] - massX[j];  
 normD += Math.pow(d[j], 2);  
 }  
 if (Math.sqrt(normD) <= eps) {  
 exit = true;  
 }  
  
  
 // Зануление м-ва  
 for (int i = 0; i < massXLength; i++) {  
 massX[i] = massX2[i];  
 massX2[i] = 0;  
 }  
  
  
 System.out.println();  
 }  
  
 System.out.println("\nВектор невязки:");  
 printVecWithoutRound(massApproximation, massApproximation.length);  
 System.out.println("\n\nКол-во ицераций k:\nk = " + (countK + 1));  
 System.out.println("\nЭпсилон E:\nE = " + eps);  
  
 return massX;  
 }  
  
  
 // Создание таблицы  
 public static void createTable(int massXLength) {  
 String str = "k";  
 System.out.print(str);  
 for (int j = 0; j < 15 - str.length(); j++) { // Вывод K  
 System.out.print(" ");  
 }  
  
 for (int i = 0; i < massXLength; i++) { // Вывод X  
 str = "x" + (i + 1);  
 System.out.print(str);  
 for (int j = 0; j < 15 - str.length(); j++) {  
 System.out.print(" ");  
 }  
 }  
  
 for (int i = 0; i < massXLength; i++) { // Вывод σ  
 str = "σ" + (i + 1);  
 System.out.print(str);  
 for (int j = 0; j < 30 - str.length(); j++) {  
 System.out.print(" ");  
 }  
 }  
 System.out.println();  
 }  
  
  
 // Вывод значений  
 public static void printMeaning(double digit) {  
 DecimalFormat decimalFormat = new DecimalFormat("#.####");  
  
 System.out.print(decimalFormat.format(digit));  
 for (int k = 0; k < 15 - decimalFormat.format(digit).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
  
  
 // Вывод вектора без округления  
 public static void printVecWithoutRound(double[] vec, int vecLength) {  
 System.out.println("Вывод в-ра:");  
  
 for (int i = 0; i < vecLength; i++) {  
 System.out.print(vec[i]);  
 for (int j = 0; j < 30 - Double.toString(vec[i]).length(); j++) {  
 System.out.print(" ");  
 }  
 }  
 }  
  
  
 // Вывод м-цы без округления  
 public static void printMtrWithoutRounding(double[][] mtr, int mtrLength) {  
 System.out.println("\nВывод м-цы");  
  
 for (int i = 0; i < mtrLength; i++) {  
 for (int j = 0; j < mtrLength; j++) {  
 System.out.print(mtr[i][j]);  
 for (int k = 0; k < 30 - Double.toString(mtr[i][j]).length(); k++) {  
 System.out.print(" ");  
 }  
 }  
 System.out.println();  
 }  
 }  
  
  
 // Матрица невязки  
 public static double[][] calcResMtr(double[][] mtrA, double[][] reverseMtr) {  
 double[][] multMtr = new double[mtrA.length][reverseMtr[0].length];  
  
 for (int i = 0; i < mtrA.length; i++) {  
 for (int j = 0; j < reverseMtr[0].length; j++) {  
 for (int k = 0; k < mtrA[0].length; k++) {  
 multMtr[i][j] += mtrA[i][k] \* reverseMtr[k][j];  
 }  
 }  
 }  
  
 double[][] singleMtr = new double[mtrA.length][mtrA.length]; // Создание единичной м-цы  
 for (int i = 0; i < mtrA.length; i++) {  
 singleMtr[i][i] = 1;  
 }  
  
 double[][] subtraction = new double[multMtr.length][multMtr[0].length];  
 for (int i = 0; i < multMtr.length; i++) {  
 for (int j = 0; j < multMtr[0].length; j++) {  
 subtraction[i][j] = multMtr[i][j] - singleMtr[i][j];  
 }  
 }  
  
 return subtraction;  
 }  
  
  
 // Сходимость метода Зейделя  
 public static void convergence(double[][] mtrA, int mtrALength) {  
 int count = 0;  
 double sum = 0.0;  
  
 for (int i = 0; i < mtrALength; i++) {  
 for (int j = 0; j < mtrALength; j++) {  
 if (i != j) {  
 sum += Math.abs(mtrA[i][j]);  
 }  
 }  
 if (Math.abs(mtrA[i][i]) > Math.abs(sum)) {  
 count++;  
 }  
 sum = 0.0;  
 }  
  
 if (count == mtrALength) {  
 System.out.println("Итерационный метод сходится");  
 }  
 else {  
 System.out.println("Итерационный метод не сходится");  
 }  
 }  
}