Университет ИТМО  
Кафедра Вычислительной Техники

Лабораторная работа № 1

по предмету Вычислительная Математика

Метод Простых итераций

Выполнил:

Тарасов А.С.

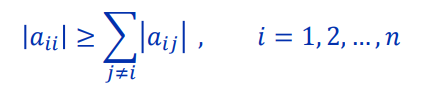
Преподаватель:

Перл О. В.

Санкт-Петербург – 2020

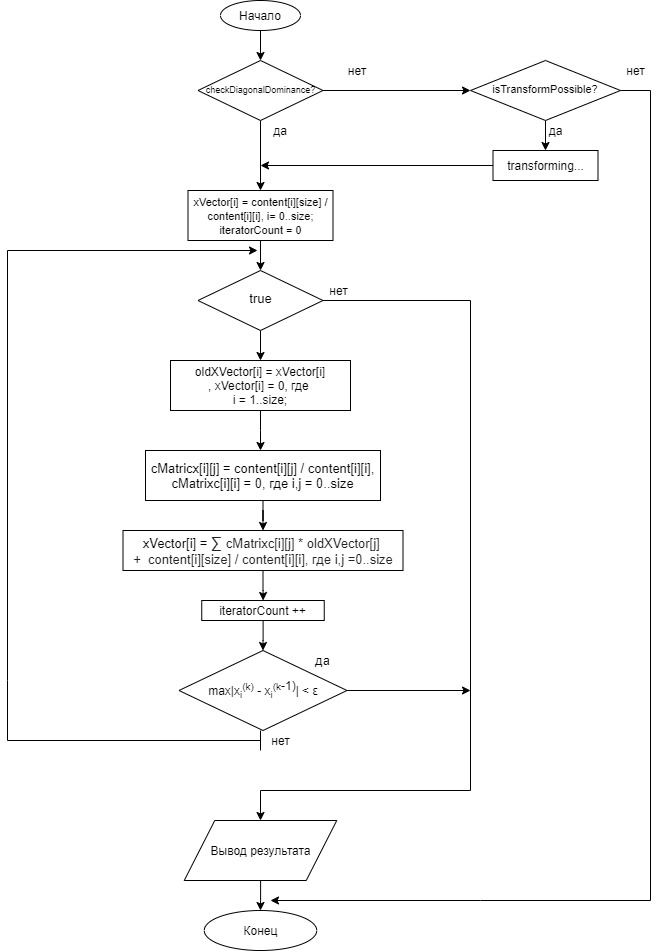
**Решение системы линейных алгебраических уравнений**

**Метод просты итераций**

**Описание метода:**   
  
 Метод простых итераций – один из классических методов решения систем линейных алгебраических уравнений. Суть метода заключается в последовательном приближении вектора корней к решению, причем каждое следующее приближение получается из предыдущего и является более точным, чем предыдущее. Точное решение СЛАУ – предел последовательности векторов 𝑥 (0) , 𝑥 (1) , … , 𝑥 (𝑘). Для возможности решения СЛАУ методом простых итераций необходимо, что бы выполнялось одно из условий сходимости.   
 В данной работе я буду пользоваться проверкой на наличие диагонального преобладания:

Как правило, за конечное число шагов (т.е.  итераций) предел не достигается, поэтому используется такое понятие, как желаемая точность ε - некоторое положительное и достаточно малое число, и процесс вычислений (итераций) проводят до тех пор, пока не будет выполнено некоторое условие, называемое критерием окончания итерационного процесса.   
 Существует несколько таки критериев: критерий по относительным разностям, критерий по невязке, и критерий по абсолютным отклонениям.  
 В данной работе я буду пользоваться критерием по абсолютным отклонениям, так как это наиболее простой способ. Суть критерия - сравнение между собой соответствующих неизвестных по двум соседним итерациям (k) и (k-1).  
Итерационный процесс продолжается до тех пор, пока не будет выполнено соотношение:  
  
 max|xi(k) - xi(k-1)| < ε

**­­­**

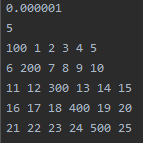
**  
Блок схема метода:  
  
  
  
  
  
  
Листинг программы(только численный метод):   
Matricx.java**

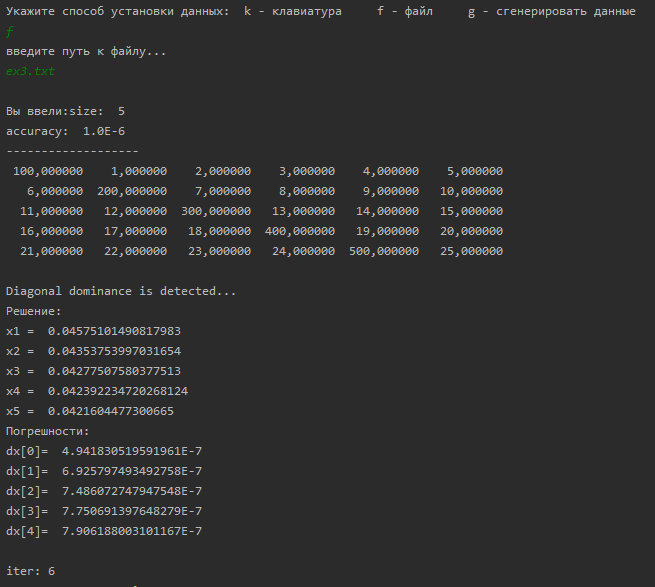
**public** String start() {  
 **oldXVector** = **new double**[**size**];  
 **xVector** = **new double**[**size**];  
 **maxValuesIndex** = **new int**[**size**];  
  
  
 **if** (checkDiagonalDominance()) {  
 **description** = **"Diagonal dominance is detected...\n"**;  
 solve();  
 } **else**{  
 **description** = **"! there is no diagonal dominance in this matricx !\n"**;  
 **if**(isTransformPossible()) {  
 **description** += **"transforming matricx...\n"**;  
 transform();  
 **description** += show() + **"\n"**;  
 solve();  
 }**else description** += **"transforming is impossible\nIt's impossible to solve this task by iteration method"**;  
 }  
 **return description**;  
}  
  
**private void** solve(){  
 initXVector();  
  
 **int** iteratorCount = 0;  
 **double**[][] cmatrix = makeCmatrix();  
 **double** xCircleTest = 0;  
  
 **while** (!isTheEnd()) {  
 **for** (**int** i = 0; i < **size**; i++) {  
 **oldXVector**[i] = **xVector**[i];  
 **xVector**[i] = 0;  
 }  
 **for** (**int** i = 0; i < **size**; i++) {  
 **for** (**int** j = 0; j < **size**; j++) {  
 **xVector**[i] += cmatrix[i][j] \* **oldXVector**[j];  
 }  
 **xVector**[i] += **content**[i][**size**] / **content**[i][i];  
 }  
 iteratorCount++;  
 **if** (iteratorCount % 3 == 0) xCircleTest = **xVector**[0];  
 **if** (iteratorCount % 3 != 0)  
 **if** (xCircleTest == **xVector**[0]) {  
 **description** += **"\n CIRCLE\n"**;  
 **break**;  
 }  
 }  
  
 **description** += **"Решение: \n"**;  
 **for** (**int** i = 0; i < **size**; i++) {  
 **description** += **"x"** + (i + 1) + **" = "** + **xVector**[i] + **"\n"**;  
 }  
 **description** += **"Погрешности: \n"**;  
 **for** (**int** i = 0; i < **size**; i++)  
 **description** += **"dx["** + i + **"]= "** + (**xVector**[i] - **oldXVector**[i]) + **"\n"**;  
  
 **description** += **"\niter: "** + iteratorCount;  
   
}

**private void** initXVector() {  
 **for** (**int** i = 0; i < **size**; i++)  
 **xVector**[i] = **content**[i][**size**] / **content**[i][i];  
}  
  
**private boolean** isTheEnd() {  
 **double** max = -999999;  
 **for** (**int** i = 0; i < **size**; i++) {  
 **if** (*abs*(**xVector**[i] - **oldXVector**[i]) > max)  
 max = *abs*(**xVector**[i] - **oldXVector**[i]);  
 }  
 **return** max < **accuracy**;  
}  
  
**private double**[][] makeCMatricx() {  
 **double**[][] fcMatricx = **new double**[**size**][**size**];  
 **for** (**int** i = 0; i < **size**; i++) {  
 **for** (**int** j = 0; j < **size**; j++) {  
 fcMatricx[i][j] = -doubleRound(**content**[i][j] / **content**[i][i], 4);  
 }  
 fcMatricx[i][i] = 0;  
 }  
 **return** fcMatricx;  
}  
  
**private double** doubleRound(**double** value, **int** places) {  
 **double** scale = Math.*pow*(10, places);  
 **return** Math.*round*(value \* scale) / scale;  
}  
  
**private boolean** checkDiagonalDominance() {  
 **boolean** isOK = **true**;  
 **for** (**int** i = 0; i < **size**; i++) {  
 **int** lineSum = 0;  
 **for** (**int** j = 0; j < **size**; j++) {  
 **if** (i != j)  
 lineSum += *abs*(**content**[i][j]);  
 }  
 isOK &= (*abs*(**content**[i][i]) >= lineSum);  
 }  
 **return** isOK;  
}  
  
**private boolean** isTransformPossible() {  
 **double** maxValue;  
 **int** indexMax;  
 **for** (**int** i = 0; i < **size**; i++) {  
 maxValue = Math.*abs*(**content**[i][0]);  
 indexMax = 0;  
 **for** (**int** j = 1; j < **size**; j++) {  
 **if** (Math.*abs*(**content**[i][j]) > maxValue) {  
 maxValue = Math.*abs*(**content**[i][j]);  
 indexMax = j;  
 }  
 }  
 **maxValuesIndex**[i] = indexMax;  
 }  
 **for** (**int** i = 0; i < **size**; i++) {  
 **for** (**int** j = 0; j < **size**; j++) {  
 **if** (i != j) {  
 **if** (**maxValuesIndex**[i] == **maxValuesIndex**[j]) {  
 **return false**;  
 }  
 }  
 }  
 }  
 **return true**;  
}  
  
**private void** transform() {  
 **double**[][] tmpContent = **new double**[**size**][**size**];  
  
 **for** (**int** i = 0; i < **size**; i++) {  
 **int** lineIndex = **maxValuesIndex**[i];  
 tmpContent[lineIndex] = **content**[i];  
 }  
 **content** = tmpContent;  
}

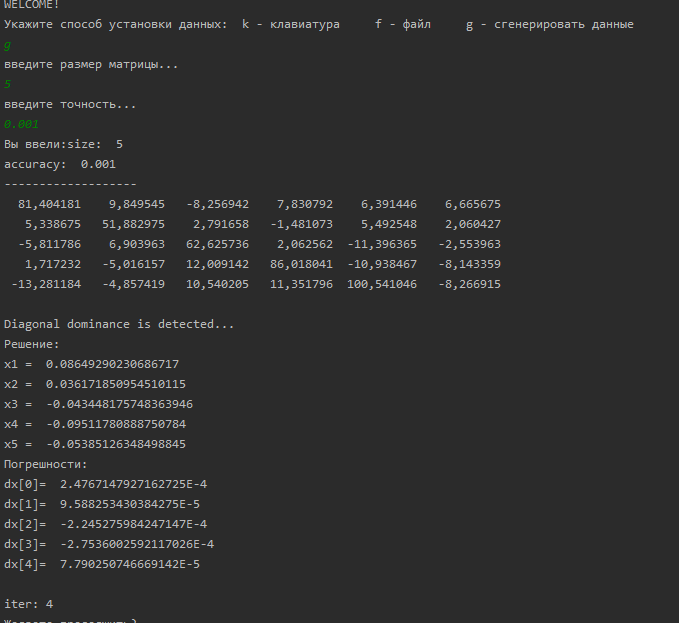
**Пример и результат работы программы:**

Пример 1. Чтение данных из файла

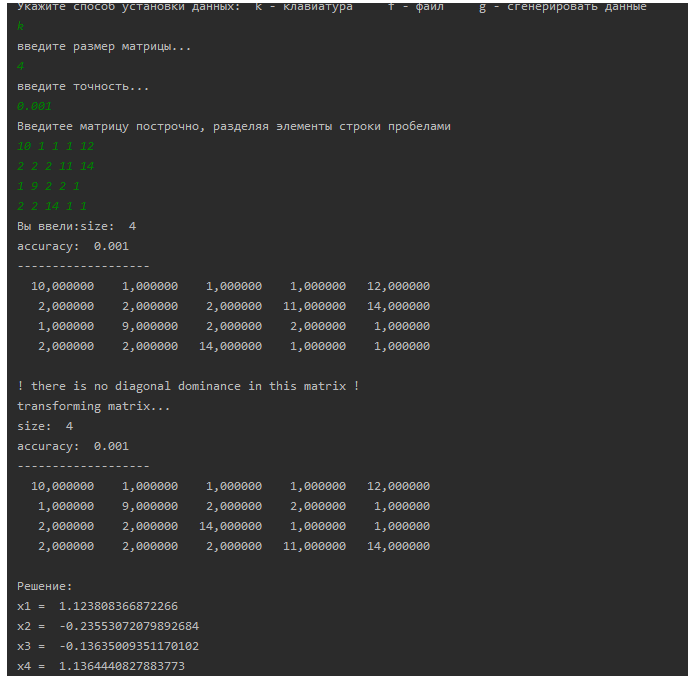
  
 (файл ex3.txt)



Пример 2. Генерация матрицы



Пример 3. Чтение данных с клавиатуры.



**Вывод:**

В данной работе реализован один из методов решения СЛАУ – метод простых итераций.

Этот метод дает худшую сходимость, чем другой итерационный метод – метод Гаусса-Зейделя, но приводит к менее объемным вычислениям. Общий недостаток итерационных методов – зависимость от начального приближения.  
Погрешности в итерационных методах не накапливаются, поскольку точность вычислений в каждой итерации определяется лишь результатами предыдущей.

Также следует сказать про прямые методы решения СЛАУ.  
Метод Гаусса - один из самых распространенных прямых методов. Он заключается в последовательном исключении переменных, когда с помощью элементарных преобразований система уравнений приводится к равносильной системе ступенчатого вида, из которого последовательно, начиная с последних по номеру переменных, находятся все остальные.

Преимущества данного метода - наличие множества модификаций, сравнительная простота, и большая универсальность, по сравнению с итерационными.   
Но данный метод менее эффективен при решении матриц больших размеров, так как алгоритмическая сложность данного метода O(n^3). Таким образом, для больших систем метод Гаусса практически не применим.