**Решение систем уравнений методом простой итерации. Погрешность вычисления.**

*Метод простой итерации, называемый также методом последовательного приближения, - это математический алгоритм нахождения значения неизвестной величины путем постепенного ее уточнения. Суть этого метода в том, что, как видно из названия, постепенно выражая из начального приближения последующие, получают все более уточненные результаты. Этот метод используется для поиска значения переменной в заданной функции, а также при решении систем уравнений, как линейных, так и нелинейных.*

**Рассмотрим, как данный метод реализуется при решении СЛАУ. Метод простой итерации имеет следующий алгоритм:**

1. Проверка выполнения условия сходимости в исходной матрице. Теорема о сходимости: если исходная матрица системы имеет диагональное преобладание (т.е, в каждой строке элементы главной диагонали должны быть больше по модулю, чем сумма элементов побочных диагоналей по модулю), то метод простых итераций - сходящийся.

2. Матрица исходной системы не всегда имеет диагональное преобладание. В таких случаях систему можно преобразовать. Уравнения, удовлетворяющие условию сходимости, оставляют нетронутыми, а с неудовлетворяющими составляют линейные комбинации, т.е. умножают, вычитают, складывают уравнения между собой до получения нужного результата.

Если в полученной системе на главной диагонали находятся неудобные коэффициенты, то к обеим частям такого уравнения прибавляют слагаемые вида сi\*xi, знаки которых должны совпадать со знаками диагональных элементов.

3. Преобразование полученной системы к нормальному виду: x-=β-+α\*x- Это можно сделать множеством способов, например, так: из первого уравнения выразить х1 через другие неизвестные, из второго- х2, из третьего- х3 и т.д.

При этом используем формулы: αij= -(aij / aii) i= bi/aii

Следует снова убедиться, что полученная система нормального вида соответствует условию сходимости: ∑ (j=1) |αij|≤ 1, при этом i= 1,2,...n

4. Начинаем применять, собственно, сам метод последовательных приближений. x(0)- начальное приближение, выразим через него х(1), далее через х(1) выразим х(2).

Общая формула а матричном виде выглядит так: x(n)= β-+α\*x(n-1) Вычисляем, пока не достигнем требуемой точности: max |xi(k)-xi(k+1) ≤ ε –

**КОД:**

|  |
| --- |
| **package** gaus;  **import** java.util.Scanner;  **public** **class** simple\_iter {  **public** **final** **static** **double** eps = 0.001;  **public** **static** **double**[][] A;  **public** **static** **double**[] B;  **public** **static** **int** n;  **public** **static** **void** main(String[] args) {    n = 3;  A = **new** **double**[][] {{10,2,-1},{-2,-6,-1},{1,-3,12}}; // матрица левых частей уравнения  B = **new** **double**[] {5,24.42,36}; // правая часть соотв.    // проверка главной диагонали матрицы А на наличие нулей  Normalization(); // выражаем x    **for**(**int** i = 0; i < n; i++) {  **for**(**int** j = 0; j < n; j++) {  System.out.print(A[i][j]+" ");  }  System.out.println(B[i]);  }    **double**[] nextB = **new** **double**[n];  **double**[] lastB = **new** **double**[n];  nextB = B;  lastB = B;    **if** (CheckNorm()) {  **do** {  lastB = nextB;  nextB = Step(lastB);  } **while**(Measure(lastB,nextB)>=eps) ;    **for**(**int** i = 0; i < n; i++) {  System.out.printf("x%d = %f" + System.lineSeparator(),i+1,nextB[i]);  }  }  }    **private** **static** **void** Normalization() {  **for**(**int** i = 0; i<n; i++) {  **if**(A[i][i]!=0) {  **double** aii = A[i][i];  **for**(**int** j = 0; j<n; j++) {  A[i][j] = A[i][j] / aii;  }  B[i] = B[i]/aii;  A[i][i] = 0;  }**else** {  System.out.println("На главной диагонали не могуть быть нули.");  **break**;  }  }  }  // проверка на сходимость  **private** **static** **boolean** CheckNorm() {  **for**(**int** i = 0; i<n; i++) {  **double** sum = 0;  **for**(**int** j = 0; j<n; j++) {  sum += A[i][j];  }  **if**(sum>1) {  System.out.println("Достаточное условие сходимости метода простых итераций НЕ выполняется");  System.out.println(sum+" > 1");  **return** **false**;  }  }  **return** **true**;  }  // сходимость  **private** **static** **double**[] Step(**double**[] X) {  **double**[] nextX = **new** **double**[n];  **for**(**int** i = 0; i < n; i++) {  **for**(**int** j = 0; j < n; j++) {  nextX[i]+=A[i][j]\*X[j];  }  nextX[i] =B[i] - nextX[i];  }  **return** nextX;  }  **private** **static** **double** Measure(**double**[] lastX, **double**[] X) {  **double** sum = 0;  **for**(**int** i = 0; i < n; i++) {  sum+= Math.abs(X[i]-lastX[i]);  }  **return** sum;  }  } |