ИТМО

ПИИКТ

**Лабораторная работа №1**

**“Вычислительная математика”**

Группа Р3201

Метод простых итераций

Выполнил: Братчиков Иван Станиславович

Приняла: Перл Ольга Вячеславовна уанщк

Санкт-Петербург

2020

**Описание метода:**Метод простых итераций – один из приближенных численных методов.

Пусть дана линейная система

Рассмотрим матрицы:

Тогда систему можно записать в виде уравнения

Разрешим первое уравнение системы относительно x1, второе – относительно x2 и тд:

Где

Получаются матрицы

Тогда систему можно записать в виде уравнения

Будем решать систему методом последовательных приближений. За нулевое приближение можно взять произвольные числа

И так далее

Если последовательность приближений имеет предел, то этот предел является решением системы

При этом процесс итерации сводится к единственному решению этой системы только при выполнении по меньшей мере одного из условий

Или

IterationMatrix.java  
*public class* IterationMatrix {  
 *private double*[][] matrix;  
 *private double* maxDeviation = 0;  
 *private int* iterations = 1;  
 *private double*[] approximation;  
 *private double*[] previousApproximation;  
  
 *public* IterationMatrix(*double*[][] initialMatrix) {  
 *this*.matrix = initialMatrix;  
 }  
  
 */\*  
 \* Applies the formula Cij = -Aij/Aii (if i != j) or Cij = 0 (if i == j)  
 \*/  
 public void* transformMatrixToXFormed() {  
 *// Loop through every row in the array  
 for* (*int* i = 0; i < matrix.length; i++) {  
 *double* Aii = matrix[i][i];  
 *// Loop through every element in the row  
 for* (*int* j = 0; j < matrix[i].length; j++) {  
 *if* (j != i) {  
 *if* (matrix[i].length - 1 != j)  
 matrix[i][j] = matrix[i][j] / -Aii;  
 *else* matrix[i][j] = matrix[i][j] / Aii;  
 } *else* matrix[i][j] = 0;  
 }  
 }  
 }  
 */\*  
 \* Returns an array of constants terms of the matrix a.k.a Bn.  
 \*/  
 public double*[] getConstantTermsVector() {  
 *double*[] constants = *new double*[matrix.length];  
 *for* (*int* i = 0; i < matrix.length; i++) {  
 constants[i] = matrix[i][matrix[i].length-1];  
 }  
 *return* constants;  
 }  
  
 */\*  
 \* Computes the X values (aka approximation) on the basis of the previously computed approximation. Also, sets the maximum deviation.  
 \*/  
 public double*[] computeXUsingPreviousApproximation(*final double*[] previousApproximation) {  
 *double*[] answer = *new double*[matrix.length];  
 *this*.maxDeviation = 0;  
 *for* (*int* i = 0; i < matrix.length; i++) {  
 answer[i] = 0;  
 *//compute the Xk terms values  
 for* (*int* j = 0; j < matrix[i].length-1; j++) {  
 answer[i] += matrix[i][j] \* previousApproximation[j];  
 }  
 *//compute the final Xk value of the row* answer[i] += matrix[i][matrix[i].length-1];  
  
 *//Search for the absolute deviation criteria  
 double* deviation = Math.*abs*(answer[i] - previousApproximation[i]);  
 *if* (deviation > *this*.maxDeviation)  
 *this*.maxDeviation = deviation;  
 }  
 *return* answer;  
 }  
   
 *public void* iterateToTheGivenEpsilon() {  
 *// if it's the first iteration then use the Constant Terms Vector as an approximation  
 if* (iterations == 1) {  
 approximation = computeXUsingPreviousApproximation(getConstantTermsVector());  
 previousApproximation = approximation;  
 } *else* {  
 *// otherwise use the previously computed approximation* previousApproximation = approximation;  
 approximation = computeXUsingPreviousApproximation(approximation);  
 }  
 iterations++;  
 }  
  
 */\*\*  
 \* Check whether the matrix is Diagonally Dominant, if not makes it so.  
 \*/  
 public boolean* transformToDominant(*int* r, *boolean*[] V, *int*[] R) {  
 *int* n = matrix.length;  
 *// if moved all of the rows then change initial matrix  
 if* (r == matrix.length) {  
 *double*[][] T = *new double*[n][n + 1];  
 *for* (*int* i = 0; i < R.length; i++) {  
 *for* (*int* j = 0; j < n + 1; j++)  
 T[i][j] = matrix[R[i]][j];  
 }  
 matrix = T;  
 *return true*;  
 }  
 *//use recursion to move through all of the rows and search for the dominant elements  
 for* (*int* i = 0; i < n; i++) {  
 *if* (V[i]) *continue*;  
  
 *double* sum = 0;  
  
 *for* (*int* j = 0; j < n; j++)  
 sum += Math.*abs*(matrix[i][j]);  
  
 *if* (2 \* Math.*abs*(matrix[i][r]) > sum) {  
 V[i] = *true*;  
 R[r] = i;  
 *if* (transformToDominant(r + 1, V, R))  
 *return true*;  
 V[i] = *false*;  
 }  
 }  
 *return false*;  
 }  
  
 */\*\*  
 \* Check whether the matrix is Diagonally Dominant, if not makes it so.  
 \*/  
 public boolean* makeDominant() {  
 *//boolean array for highlighting moved rows  
 boolean*[] visited = *new boolean*[matrix.length];  
 *int*[] rows = *new int*[matrix.length];  
  
 Arrays.*fill*(visited, *false*);  
  
 *return* transformToDominant(0, visited, rows);  
 }  
  
 *public double* getMaxDeviation() {  
 *return* maxDeviation;  
 }  
 *public int* getIterations() {  
 *return* iterations;  
 }  
 *public double*[] getApproximation() {  
 *return* approximation;  
 }  
 *public double*[] getPreviousApproximation() {  
 *return* previousApproximation;  
 }  
}

**Блок схема:**  
A close up of a map

Description automatically generated

**Пример:**

A close up of a keyboard

Description automatically generated

Точность – 1.0E-9

Количество итераций: 32

Результат:

x1: 0.9888792527697281

x2: 0.990114891303152

x3: 0.9912132366582859

x4: -0.24958526033103978

x5: 0.31452493120620395

Вектор погрешностей:

x1^(32)-x1^(31): 5.147853254783286E-10

x2^(32)-x2^(31): 6.001125152366171E-10

x3^(32)-x3^(31): 6.995828361056056E-10

x4^(32)-x4^(31): 3.7679981357285897E-10

x5^(32)-x5^(31): 2.6882840398201324E-10

A screenshot of a cell phone

Description automatically generated

Точность – 1.0E-16

Количество итераций: 20

Результат:

x1: 0.04575092285414497

x2: 0.043537399584321554

x3: 0.04277650096031975

x4: 0.04239206788588952

x5: 0.042160277355718354

Вектор погрешностей:

x1^(20)-x1^(19): 1.3877787807814457E-17

x2^(20)-x2^(19): 2.0816681711721685E-17

x3^(20)-x3^(19): 2.7755575615628914E-17

x4^(20)-x4^(19): 2.7755575615628914E-17

x5^(20)-x5^(19): 2.7755575615628914E-17

**Вывод:**

Метод простых итераций подходит для систем любого размера, в том числе при n>200, так как количество вычислений относительно невелико для каждой итерации. Также методом достигается низкая погрешность за счет итераций, по сравнению с прямыми методами, из-за большого количества последовательных вычислений и ограниченности разрядной сетки, в которых, итоговый результат получается неточным. Однако для выполнения метода должны выполняться строгие условия сходимости, поэтому не для всех систем данный метод будет работать. Также преимуществом метода можно считать его понятность и простоту реализации.