Университет ИТМО

ФПИиКТ

Лабораторная работа №1  
по Вычислительной математике

Выполнил: Балтабаев Дамир  
Группа: P3210  
Вариант: 4

Преподаватель: Малышева Татьяна Алексеевна

Санкт-Петербург  
2022

**Цель работы:**

Реализовать решение СЛАУ в виде отдельной подпрограммы или класса, в который входные/выходные данные передаются в качестве параметров.

**Задание:**

**Метод простых итераций**

**Для итерационных методов должно быть реализовано:**

* Точность задается с клавиатуры/файла
* Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
* Вывод вектора неизвестных:
* Вывод количества итераций, за которое было найдено решение.
* Вывод вектора погрешностей:

**Описание метода, расчетные формулы:**

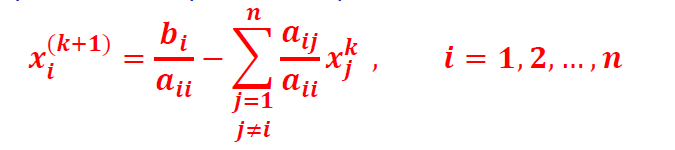
**Метод итерации** или **метод простой итерации** — [численный метод](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B) решения [системы линейных алгебраических уравнений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9). Суть метода заключается в нахождении по приближённому значению величины следующего приближения, являющегося более точным.

Метод позволяет получить значения корней системы с заданной точностью в виде [предела последовательности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%B5%D0%BB_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8) некоторых векторов (в результате итерационного процесса). Характер сходимости и сам факт сходимости метода зависит от выбора начального приближения корня.

Достаточным условием сходимости *итерационного процесса*

к решению системы при любом начальном векторе 𝑥𝑖(0)является выполнение условия преобладания диагональных элементов или доминирование диагонали:

Рабочая формула метода простой итерации:



где *k*–номер итерации.

За начальное (нулевое) приближение выбирают вектор свободных членов: 𝑥(0)=𝐷 или нулевой вектор: 𝑥(0)=0

Следующее приближение: 

Итерация продолжается до тех пор, пока погрешность не приблизится к заданной точности.

Формула:

Листинг программы:

simpleIterationMethod() - метод, отвечающий за начало работы всего алгоритма, он распоряжается порядком выполнения того или иного пункта.

public void simpleIterationMethod(MatrixPOJO userMatrix) {  
  
 MatrixPOJO finalUserMatrix;  
 if (!checkDiagonalDominance(userMatrix)) {  
 MatrixPOJO changedMatrix = changeRows(userMatrix);  
  
 if (!checkDiagonalDominance(changedMatrix)) {  
 messenger.diagonalDominatingIsMissingMessage();  
 finalUserMatrix = userMatrix;  
 } else {  
 finalUserMatrix = changedMatrix;  
 messenger.newChangedMatrixMessage();  
 messenger.introducedMatrixMessage(finalUserMatrix.getMatrix());  
 }  
  
 } else finalUserMatrix = userMatrix;  
  
 iterate(finalUserMatrix);  
  
  
}

Предыдущий метод отправляет матрицу на проверку диагонального преобладания в метод checkDiagonalDominance(), который возвращает булевое значение в зависимости от проверки.

public boolean checkDiagonalDominance(MatrixPOJO userMatrix) {  
 try {  
 double rowSum = 0;  
 int strictInequalityCounter = 0;  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 for (int j = 0; j < userMatrix.getSize(); j++) {  
 if (i != j) {  
 rowSum += Math.*abs*(userMatrix.getMatrix()[i][j]);  
 }  
 }  
 if (!(Math.*abs*(userMatrix.getMatrix()[i][i]) >= rowSum)) {  
 return false;  
 }  
 if (Math.*abs*(userMatrix.getMatrix()[i][i]) > rowSum) {  
 strictInequalityCounter++;  
 }  
 rowSum = 0;  
  
 }  
  
 if (strictInequalityCounter == 0) return false;  
 } catch (NullPointerException nullPointerException) {  
 return false;  
 }  
 return true;  
}

В случае отсутствия диагонального преобладания, основной метод отправляет матрицу в метод changeRows(), который меняет строки матрицы местами, пытаясь достичь диагонального преобладания. Если полученная матрица после выхода из метода не проходит проверку на диагональное преобладание – с-ма выводит сообщение о невозможности достижения диаг. Преобладания.

public MatrixPOJO changeRows(MatrixPOJO userMatrix) {  
 double maxElement = -10000;  
 int maxElementColumnIndex = 0;  
 Double[][] changedMatrix = new Double[userMatrix.getSize()][userMatrix.getSize() + 1];  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 for (int j = 0; j < userMatrix.getSize(); j++) {  
 if (userMatrix.getMatrix()[i][j] > maxElement) {  
 maxElement = userMatrix.getMatrix()[i][j];  
 maxElementColumnIndex = j;  
 }  
 }  
 int k = 0;  
 while (k < userMatrix.getSize() + 1) {  
 changedMatrix[maxElementColumnIndex][k] = userMatrix.getMatrix()[i][k];  
 k++;  
 }  
 maxElement = 0;  
 maxElementColumnIndex = 0;  
 }  
 return new MatrixPOJO(changedMatrix, userMatrix.getAccuracy(), userMatrix.getSize());  
}

Метод iterate() – основной метод, отвечающий за весь функционал метода простых итераций. Метод занимается поиском приближения по основной формуле и обеспечивает итерационный механизм. Также метод считает погрешность по основной формуле и в случае соблюдения условия окончания итерации – останавливает процесс.

public void iterate(MatrixPOJO userMatrix) {  
 double previousErrorEstimate = 0;  
  
 Double[] results = new Double[userMatrix.getSize()];  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 for (int j = 0; j < userMatrix.getSize() + 1; j++) {  
 results[i] = userMatrix.getMatrix()[i][j];  
 }  
 }  
  
 Double[] initialApproximation = new Double[userMatrix.getSize()];  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 initialApproximation[i] = 0.0;  
 }  
  
 int numberOfIterations = 0;  
 Double[] newInitialApproximation = new Double[userMatrix.getSize()];  
  
 while (true) {  
 messenger.numberOfIterationMessage(numberOfIterations);  
 double formula = 0;  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 for (int j = 0; j < userMatrix.getSize(); j++) {  
 if (i != j) {  
 formula = formula + (-userMatrix.getMatrix()[i][j] \* initialApproximation[j]);  
 } else continue;  
 }  
 int indexOfX = i + 1;  
 formula = (results[i] + formula) / userMatrix.getMatrix()[i][i];  
 messenger.computedXMessage(indexOfX, formula);  
 newInitialApproximation[i] = formula;  
 formula = 0;  
 }  
  
 if (numberOfIterations != 0) {  
 double maxOfErrorVector = Math.*abs*(newInitialApproximation[0] - initialApproximation[0]);  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 double def = Math.*abs*(newInitialApproximation[i] - initialApproximation[i]);  
 if (def > maxOfErrorVector) {  
 maxOfErrorVector = def;  
 }  
 }  
 messenger.currentErrorEstimateMessage(maxOfErrorVector);  
 if (previousErrorEstimate != 0 && maxOfErrorVector > previousErrorEstimate) {  
 System.*out*.println("Погрешность увеличилась!");  
 return;  
 }  
 previousErrorEstimate = maxOfErrorVector;  
 if (maxOfErrorVector <= userMatrix.getAccuracy()) return;  
 }  
 numberOfIterations++;  
 for (int i = 0; i < userMatrix.getSize(); i++) {  
 initialApproximation[i] = newInitialApproximation[i];  
 }  
  
  
 }  
  
  
}

**Пример работы программы:**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Вывод:**

В ходе выполнения данной лабораторной работы, я столкнулся с числовым методом простых итераций, позволяющим решить Систему линейных алгебраических уравнений быстрым способом. Изучил понятие диагонального преобладания, необходимое для соблюдения условия сходимости матрицы. Метод простых итераций показался мне довольно эффективным и не затрагивающим большого кол-ва памяти методом, ведь каждый фрагмент необходимых формул не сохраняется в память, а перезаписывается по мере необходимости. Из недостатков можно отметить относительную сложность метода, из-за кол-ва необходимых соблюдений условий.