Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №1**

по дисциплине

“Вычислительная математика”

Метод Гаусса

**Выполнил:**

Кислицин Алексей Андреевич

**Группа:**

P3231

**Преподаватель:**

Перл Ольга Вячеславовна



Санкт-Петербург, 2022

***Метод Гаусса***

Метод Гаусса позволяет найти неизвестные элементы СЛАУ, если ни один из ведущих элементов матрицы коэффициентов не равен нулю. Суть метода заключается к сведению расширенной матрицы СЛАУ к треугольному виду, чтобы потом пройтись в обратном порядке по уравнениям и рекуррентно найти неизвестные.

***Листинг программы***

public double calculateDeterminant(Matrix matrix) {  
 Matrix convertedMatrix = convertMatrixForDetermination(matrix);  
 double determinant = 0;  
 if (convertedMatrix!=null) {  
 determinant = convertedMatrix.getCoefficientsOfUnknowns()[0][0];  
 for (int i = 1; i<convertedMatrix.getNumberOfUnknowns() ; ++i) {  
 determinant \*= convertedMatrix.getCoefficientsOfUnknowns()[i][i];  
 }  
 }  
 return determinant;  
}  
  
private Matrix convertMatrixForDetermination(final Matrix matrix) {  
 Matrix copiedMatrix = matrix.getCopyOfSquareMatrix();  
 for (int i=0; i<matrix.getNumberOfUnknowns()-1 ; ++i) {  
 double maxElement = 0;  
 int lineIndex = i;  
 for (int j=i; j<matrix.getNumberOfUnknowns() ; ++j) {  
 if (copiedMatrix.getCoefficientsOfUnknowns()[j][i]!=0 && Math.*abs*(copiedMatrix.getCoefficientsOfUnknowns()[j][i])>Math.*abs*(maxElement)) {  
 maxElement = copiedMatrix.getCoefficientsOfUnknowns()[j][i];  
 lineIndex = j;  
 }  
 }  
 if (maxElement == 0 && i==0)  
 return null;  
 else if (maxElement!=0) {  
 double[] firstLine = new double[matrix.getNumberOfUnknowns()];  
 double freeCofFirstLine = copiedMatrix.getFreeCoefficients()[i];  
 for (int z = 0; z < matrix.getNumberOfUnknowns(); ++z)  
 firstLine[z] = copiedMatrix.getCoefficientsOfUnknowns()[i][z];  
 copiedMatrix.getCoefficientsOfUnknowns()[i] = copiedMatrix.getCoefficientsOfUnknowns()[lineIndex];  
 copiedMatrix.getCoefficientsOfUnknowns()[lineIndex] = firstLine;  
 copiedMatrix.getFreeCoefficients()[i] = copiedMatrix.getFreeCoefficients()[lineIndex];  
 copiedMatrix.getFreeCoefficients()[lineIndex] = freeCofFirstLine;  
 double[] divideCoefficients = new double[matrix.getNumberOfUnknowns() - 1 - i];  
 int arrIndex = 0;  
 for (int s = i + 1; s < matrix.getNumberOfUnknowns(); ++s) {  
 divideCoefficients[arrIndex] = -copiedMatrix.getCoefficientsOfUnknowns()[s][i] / maxElement;  
 arrIndex++;  
 }  
 arrIndex = 0;  
 for (int k = i + 1; k < matrix.getNumberOfUnknowns(); ++k) {  
 for (int l = 0; l < matrix.getNumberOfUnknowns(); ++l) {  
 copiedMatrix.getCoefficientsOfUnknowns()[k][l] += divideCoefficients[arrIndex] \* copiedMatrix.getCoefficientsOfUnknowns()[i][l];  
 }  
 copiedMatrix.getFreeCoefficients()[k] += divideCoefficients[arrIndex] \* copiedMatrix.getFreeCoefficients()[i];  
 arrIndex++;  
 }  
 }  
 }  
 return copiedMatrix;  
}  
  
public Matrix convertToTriangularMatrix(final Matrix matrix) {  
 Matrix copiedMatrix = matrix.getCopyOfSquareMatrix();  
 for (int i = 0; i < matrix.getNumberOfUnknowns()-1; ++i) {  
 double leadingElement = copiedMatrix.getCoefficientsOfUnknowns()[i][i];  
 for (int j = i; j < matrix.getNumberOfUnknowns(); ++j) {  
 copiedMatrix.getCoefficientsOfUnknowns()[i][j] /= leadingElement;  
 }  
 copiedMatrix.getFreeCoefficients()[i] /= leadingElement;  
 for (int k = i+1; k < matrix.getNumberOfUnknowns(); ++k) {  
 double leadingElementOfCurrentEquation = copiedMatrix.getCoefficientsOfUnknowns()[k][i];  
 for (int j = i; j < matrix.getNumberOfUnknowns(); ++j)  
 copiedMatrix.getCoefficientsOfUnknowns()[k][j] -= leadingElementOfCurrentEquation\*copiedMatrix.getCoefficientsOfUnknowns()[i][j];  
 copiedMatrix.getFreeCoefficients()[k] -= leadingElementOfCurrentEquation\*copiedMatrix.getFreeCoefficients()[i];  
 }  
 }  
 return copiedMatrix;  
}  
  
public double[] findUnknownsOfEquations(Matrix triangularMatrix) {  
 double[] definitionOfUnknowns = new double[triangularMatrix.getNumberOfUnknowns()];  
 final double[][] coefficientsOfUnknowns = triangularMatrix.getCoefficientsOfUnknowns();  
 final double[] freeCoefficients = triangularMatrix.getFreeCoefficients();  
 for (int i = triangularMatrix.getNumberOfUnknowns()-1; i>=0 ; --i) {  
 double subtrahend = 0;  
 for (int j = i+1; j<triangularMatrix.getNumberOfUnknowns(); ++j) {  
 subtrahend += coefficientsOfUnknowns[i][j]\*definitionOfUnknowns[j];  
 }  
 definitionOfUnknowns[i] = (freeCoefficients[i]-subtrahend) / coefficientsOfUnknowns[i][i];  
 }  
 return definitionOfUnknowns;  
}  
  
public double[] calculateFaults(Matrix matrix, double[] definitionOfUnknowns) {  
 double[] faults = new double[matrix.getNumberOfUnknowns()];  
 for (int i = 0 ; i<matrix.getNumberOfUnknowns() ; ++i) {  
 double leftSum = 0;  
 for (int j = 0 ; j<matrix.getNumberOfUnknowns() ; ++j)  
 leftSum += matrix.getCoefficientsOfUnknowns()[i][j]\*definitionOfUnknowns[j];  
 faults[i] = Math.*abs*(leftSum-matrix.getFreeCoefficients()[i]);  
 }  
 return faults;  
}

***Блок-схема методов, которые находят неизвестные СЛАУ***

(без нахождения определителя и невязок)

***Примеры работы программы***

***Изображение выглядит как текст

Автоматически созданное описание***

***Изображение выглядит как текст

Автоматически созданное описание***

***Изображение выглядит как текст

Автоматически созданное описание***

***Вывод***

В результате проделанной работы я исследовал прямые и итерационные методы для нахождения неизвестных СЛАУ. У каждого метода есть свои преимущества и недостатки. Так, например, метод Гаусса с выбором главного элемента высчитывает с меньшими погрешностями, чем обычный метод Гаусса, и имеет алгоритмическую сложность O(n3).

Если же рассматривать прямые и итерационные методы в целом, то можно сделать следующие выводы:

1. Прямые методы позволяют найти ответ за точное количество итераций, когда итерационные не имеют начального значения.
2. Погрешность прямых методов зависит от количества промежуточных вычислений, а у итерационных методов точность задается.
3. При использовании прямых методов необходимо хранить полную матрицу, а при итерационных данное условие не нужно.
4. Итерационные методы используются больше для решения больших СЛАУ, а прямые – для маленьких и средних.