Университет ИТМО  
Факультет программной инженерии   
и компьютерной техники

Кафедра Вычислительной Техники

Лабораторная работа №1

По курсу вычислительной математики.

"Решение СЛАУ методом Гаусса."

Выполнил: Беспалов В.Д.

Студент первого курса группы P3211

Преподаватель: Петрова М.М.

Задание:

Составить программу, обеспечивающую решение системы линейных алгебраических уравнений порядка n (n <=20) методом Гаусса.

Описание метода:

**Метод Гаусса** основан на приведении матрицы системы к треугольному виду. Это достигается последовательным исключением неизвестных из уравнений системы. Сначала с помощью первого уравнения исключается *x1* из всех последующих уравнений системы. Затем с помощью второго уравнения исключается х2 из третьего и всех последующих уравнений. Этот процесс, называемый *прямым ходом метода Гаусса,* продолжается до тех пор, пока в левой части последнего уравнения не останется лишь один член с неизвестным *хп,* т. е. матрица системы будет приведена к треугольному виду (к такому виду приводится лишь невырожденная матрица, в противном случае метод Гаусса неприменим). Когда матрица системы примет треугольный вид закончится прямой ход метода Гаусса.

*Обратный ход метода Гаусса* состоит в последовательном вычислении искомых неизвестных: решая последнее сравнение, находим единственное неизвестное *хп.* Далее, используя это значение, из предыдущего уравнения вычисляем *хп-1* и т. д. Последним найдем *x1* из первого уравнения.

Расчетные формулы:

* mat – матрица коэффициентов СЛАУ.
* b – матрица (вектор) свободных членов СЛАУ.
* i – номер строки.
* j – номер столбца.
* v – невязки.
* x – матрица решений системы.
* k - доп. переменная для нахождения решений в цикле.

1. i от 0 до n-1, j от 0 до n;
2. mat[j][k] += -mat[j][i] /mat[i][i]\* mat[i][k];
3. x[i] -= matr[n-1-j][n]\*matr[n-1-k][n-1-j];
4. v = b - mat[i][j] \* x[i];

Код класса Gauss:

**import** java.util.\*;

**public class** Gauss {

**private int size**;

**private double**[] **rightColumn**;

**private double**[] **rightAnswer**;

**private double**[] **innacurasy**;

**private double**[][] **matrix**;

**private double**[][] **startMatrix**;

**private** Stack<Integer> **firstSwapped** = **new** Stack<>();

**private** Stack<Integer> **secondSwapped** = **new** Stack<>();

**public** Gauss(**int** s){

init(generate(s));

}

**public** Gauss(**double**[][] mat){

**if**(mat.**length** == 1)

init(generate((**int**)mat[0][0]));

**else**

init(mat);

}

**public double**[] getInnacurasy() {

**return innacurasy**;

}

**public double**[][] getStartMatrix() {

**return startMatrix**;

}

**public double**[] getRightAnswer() {

**return rightAnswer**;

}

**public double**[][] getMatrix() {

**return matrix**;

}

**private void** init(**double**[][] mat){

**rightColumn** = **new double**[mat.**length**];

**for**(**int** i = 0;i < mat.**length**;i++){

**rightColumn**[i] = mat[i][mat.**length**];

}

**startMatrix** = copyMatrix(mat);

**for**(**int** i = 0;i < mat.**length**-1;i++){

**if**(mat[i][i]!=0) {

**double** firstDiv = mat[i][i];

**for** (**int** j = i; j <= mat.**length**; j++) {

mat[i][j] /= firstDiv;

}

**for** (**int** j = i + 1; j < mat.**length**; j++) {

**double** temp = -mat[j][i] /mat[i][i] ;

**for** (**int** k = i; k <= mat.**length**; k++) {

mat[j][k] += temp \* mat[i][k];

}

}

}

**else**{

**boolean** b = **false**;

**for**(**int** h = i+1;h < mat.**length**;h++){

**if**(mat[h][i]!=0){

mat=swap(mat,i,h);

i--;

b=**true**;

**break**;

}

}

**if**(!b) {

System.***out***.println(**"Решений нет!"**);

**return**;

}

}

}

**if**(mat[mat.**length**-1][mat.**length**-1]==0){

System.***out***.println(**"Решений нет!"**);

**return**;

}

mat[mat.**length**-1][mat.**length**] /= mat[mat.**length**-1][mat.**length**-1];

mat[mat.**length**-1][mat.**length**-1] = 1;

**matrix** = mat;

System.***out***.println();

goBack();

}

**public double**[][] generate(**int** size){

Random random = **new** Random();

**double**[][] matrix = **new double**[size][size+1];

**for** (**int** i = 0; i < size; i++){

**for**(**int** j =0;j <=size;j++) {

matrix[i][j] = random.nextDouble()\*100;

}

}

**return** matrix;

}

**private double**[][] copyMatrix(**double**[][] matrix){

**double**[][]matr = **new double**[matrix.**length**][matrix.**length**+1];

**for**(**int** i = 0;i < matrix.**length**;i++) {

**for** (**int** j = 0; j <= matrix.**length**; j++) {

matr[i][j] = matrix[i][j];

}

}

**return** matr;

}

**private void** goBack(){

**double**[][] matr = copyMatrix(**matrix**);

**rightAnswer** = **new double**[**matrix**.**length**];

**rightAnswer**[matr.**length**-1] = matr[matr.**length**-1][matr.**length**];

**size** = matr.**length**;

**int** k = 1;

**for**(**int** i = matr.**length**-2;i >= 0;i--){

**for**(**int** j = 0;j<k;j++){

matr[i][**size**] -= matr[**size**-1-j][**size**]\*matr[**size**-1-k][**size**-1-j];

matr[**size**-1-k][**size**-1-j] = 0;

}

**rightAnswer**[i] = matr[i][**size**];

k++;

}

setInnacurasy();

}

**private void** setInnacurasy(){

**innacurasy**= **new double**[**size**];

**for**(**int** i = 0;i < **size**;i++){

**double** temp = **rightColumn**[i];

**for**(**int** j = 0;j < **size**; j++){

temp-=**startMatrix**[i][j]\***rightAnswer**[j];

}

**innacurasy**[i] = temp;

}

System.***out***.println(**"Матрица обработана."**);

}

**private double**[][] swap(**double**[][] mat,**int** i,**int** h){

**firstSwapped**.push(i);

**secondSwapped**.push(h);

**for**(**int** j = i;j<=mat.**length**;j++){

**double** temp = mat[i][j];

mat[i][j] = mat[h][j];

mat[h][j] = temp;

}

**return** mat;

}

**public double** CalculateMatrix(**double**[][] matrix){

**double** calcResult=0;

**if** (matrix.**length**==2){

calcResult=matrix[0][0]\*matrix[1][1]-matrix[1][0]\*matrix[0][1];

}

**else**{

**int** koeff;

**for**(**int** i=0; i<matrix.**length**; i++){

**if**(i%2==1){

koeff=-1;

}

**else**{

koeff=1;

};

calcResult += koeff\*matrix[0][i]\***this**.CalculateMatrix(**this**.GetMinor(matrix,0,i));

}

}

**return** calcResult;

}

**private double**[][] GetMinor(**double**[][] matrix, **int** row, **int** column){

**double**[][] minor = **new double**[matrix.**length**-1][matrix.**length**-1];

**int** dI=0;

**for**(**int** i=0; i<matrix.**length**; i++){

**int** dJ=0;

**for**(**int** j=0; j<matrix.**length**; j++){

**if**(i==row){

dI=1;

}

**else**{

**if**(j==column){

dJ=1;

}

**else**{

minor[i-dI][j-dJ] = matrix[i][j];

}

}

}

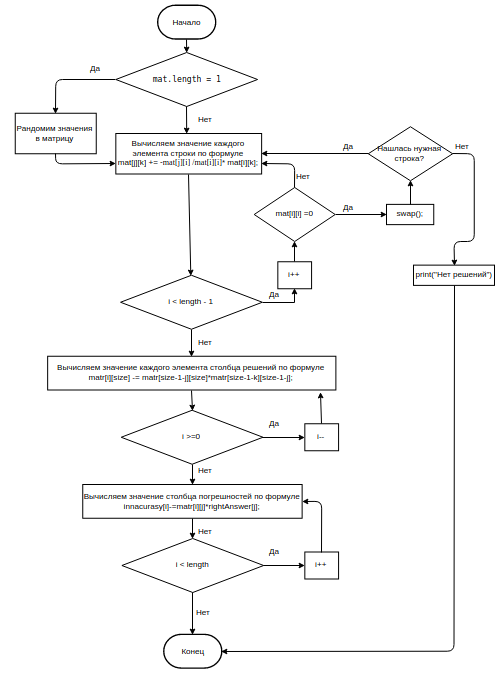
}

**return** minor;

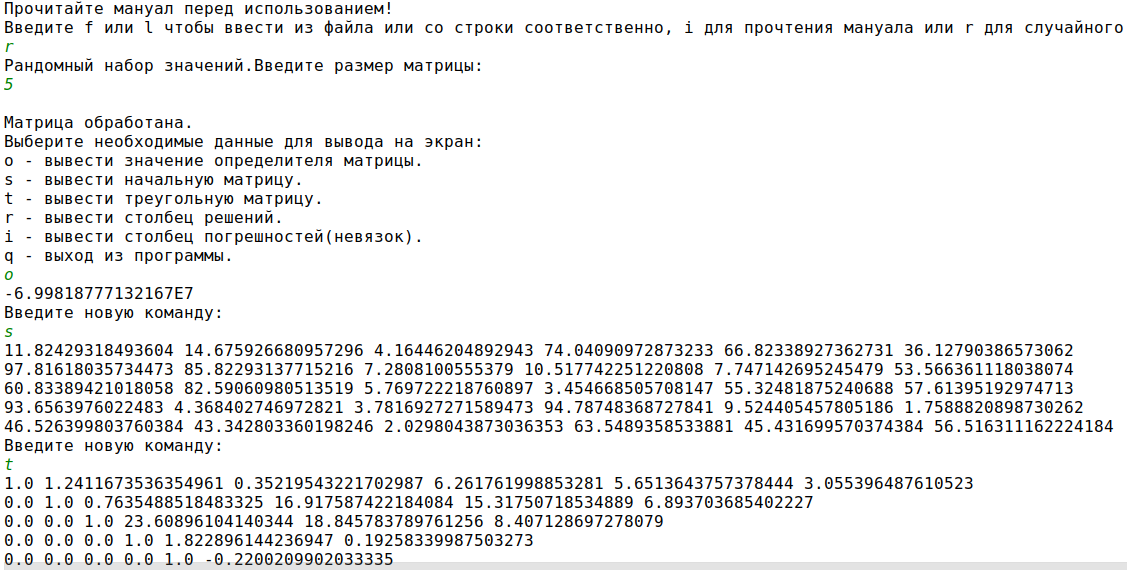
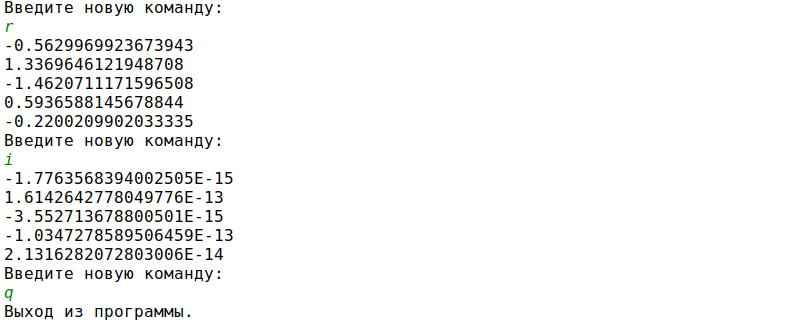
}

}

Блок-схема класса Gauss:



Результат:



**Выводы:**

Для проверки работоспособности программы подавались значения, в которых были решения и не было, результат должен был быть дробным и целочисленным, матрицы были разных размеров, а также на вход подавались всевозможные неправильные варианты заполнения матрицы.

Метод Гаусса эффективен, обладает высокой точностью и довольно прост, что позволяет несложно реализовать его программным путём. Единственная проблема - это вычисление определителя матрицы при размерности более 10, т.к. занимает очень продолжительное время.