Белорусский государственный университет

Факультет прикладной математики и информатики

Кафедра технологий программирования

Стефанович Константин Андреевич

**Лабораторная работа №1**

студента 2 курса 6 группы

**Преподаватель**

***Радкевич Елена Владимировна*** Ассистент кафедры вычислительной математики ФПМИ

Минск, 2016

Оглавление

[1.Техническое задание 3](#_Toc462953523)

[2.Постановка задачи 3](#_Toc462953524)

[3.Программная реализация 3](#_Toc462953525)

[4. Листинг программы 5](#_Toc462953526)

[5.Вывод 10](#_Toc462953527)

# 1.Техническое задание

Реализовать алгоритм Гаусса для решения СЛАУ, проверить

Правильность и точность вычислений.

# 2.Постановка задачи

1. Реализовать алгоритм Гаусса с выбором главного элемента по строке для решения СЛАУ.
2. Вывести вектор решений X
3. Правильность вычислений проверить, выведя вектор AX –B
4. Найти для данной системы обратную матрицу и определитель

Выполнить задание надо было на примере следующих данных:

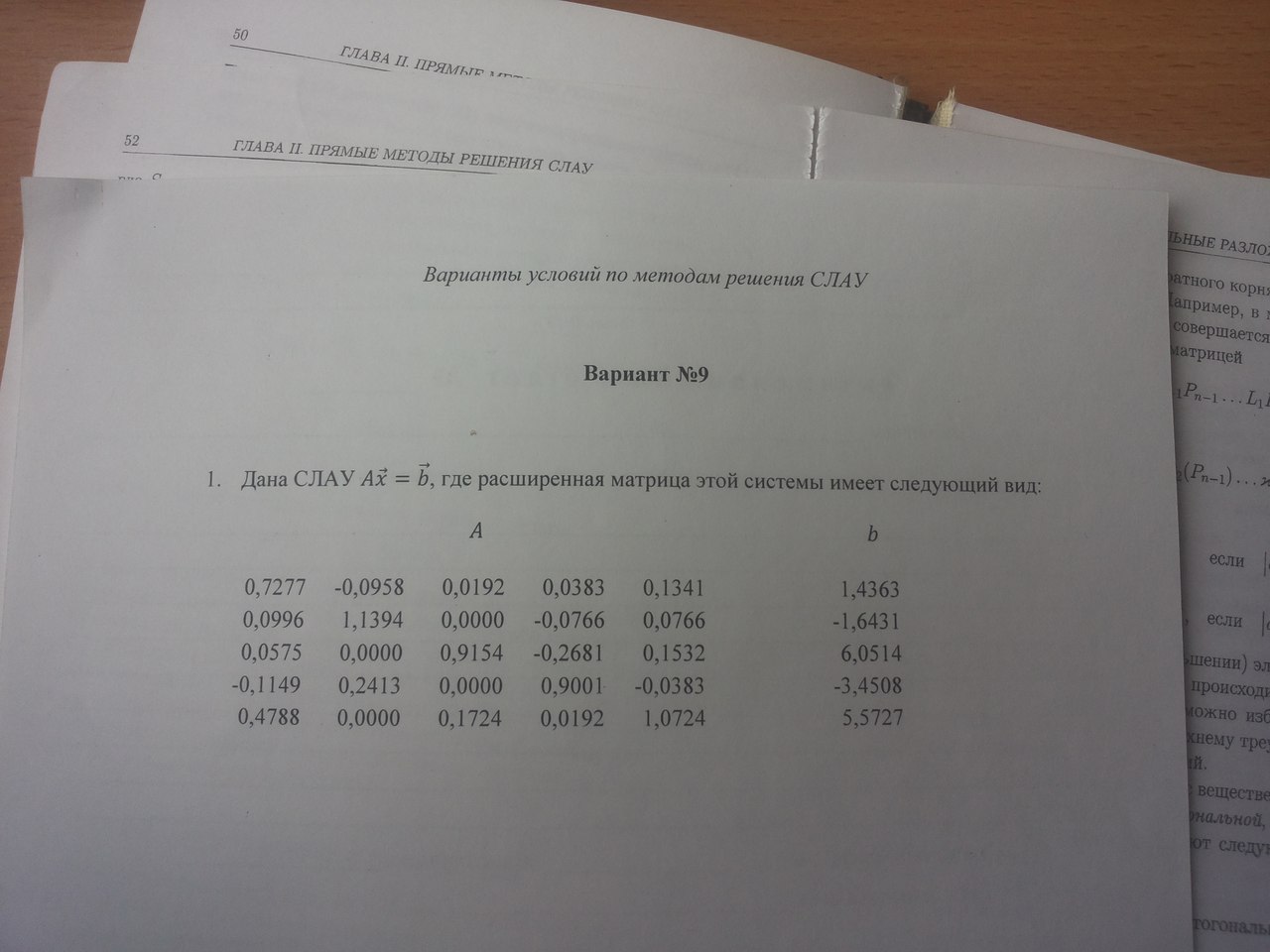


Рис.1. Данные для ввода.

# 3.Программная реализация

Для реализации данного алгоритма я написал 3 класса: MyEquation, LinearEquation и Algorithm, которые наследуются друг от друга в порядке их следования.

В первом из них хранится строчка типа Float, реализована генерация этой строчки для расширенной матрицы(из статической матрицы или заполняется случайными элементами) – функция generate. Также реализована генерация строчек для нахождения обратной матрицы по методу Гаусса – generate2. Также в этом классе реализованы простейшие функции для работы с этой строкой – умножение на число(функция mul), прибавление одной строчки к другой( функция addEquation) , нахождение коэффициента для прибавления строчек(функция findCoefficient). Помимо этого, есть индексирование(функция at), метод get для размера( функция size), а также конструкторы.

Во втором классе реализована уже настоящая матрица в виде списка из объектов прошлого класса. Добавлена функция извлечение элемента из матрицы(функция itemAt) , нахождение максимального элемента в данной строчке(функция max),добавление строчки и её возврат.  
Третий класс уже реализовывает непосредственно алгоритмы Гаусса. Первая функция – calculate – реализует прямой ход для расширенной матрицы и работает следующим образом: берём первую строчку, находим максимальный элемент, находим на какое число нужно домножить элемент, который стоит по нашим в строчке ниже, домножаем, к этой строке добавляем первую. Затем также найдём коэффиценты, домножим на них и добавим первую строчку ко все нижестоящис строчкам. После выполнения этого, перейдём ко второй строчке и проделаем то же самое. Повторять будем до последней строчки. Обратный ход реализован в классе Main в функции main.

Вторая функция выполняет поиск обратной матрицы методом Гаусса-Жордана. Вначале всё так же, как и в прошлой функции(за исключением поиска максимального элемента), далее мы получаем еденичную матрицу слева примерно также, только идём с низу матрицы. В итоге слева мы получаем еденичную матрицу, а справа обратную к данной.

Также реализован класс MatrixCalculation для поиска определителя матрицы через рекурсию. Реализуется она так: если матрица 2\*2, то считаем определитель, иначе раскладываем матрицу на миноры.  
Последний класс, Main, непосредственно реализовывает всё вышеописанное на живом примере.

Там также реализован обратный ход для нашего алгоритма Гаусса.

# 4. Листинг программы

Class MyEquation

**package** com.company;  
**import** java.util.\*;  
**class** MyEquation {  
 **private** Float[][] **mas**=**new** Float[][]  
 {{0.7277f,-0.0958f,0.0192f, 0.0383f,0.1341f,1.4363f},  
 {0.0996f, 1.1394f,0.0000f,-0.0766f,0.0766f,-1.6431f},  
 {0.0575f, 0.0000f,0.9154f,-0.2681f,0.1532f,6.0514f},  
 {-0.1149f,0.2413f,0.0000f,0.9001f,-0.0383f,-3.4508f},  
 {0.4788f,0.0000f,0.1724f,0.0192f,1.0724f,5.5727f}};  
 **private** List<Float> **equation** = **new** ArrayList<>();  
 **public** List<Float> getEquation(){  
 **return equation**;  
 }  
 **public void** generate(**int** a,**int** size){  
 **if** (size < 2) size = 2;  
 **this**.**equation**.clear();  
 **for** (**int** i = 0; i < size; i++){  
 *// Random random = new Random();  
 // this.equation.add((float) (random.nextInt() % 5) + 1);* **this**.**equation**.add(**mas**[a][i]);  
 }  
 }  
 **public void** generate2(**int** a,LinearEquations l){  
  
 **for** (**int** i = 0; i < l.size(); i++){  
 **this**.**equation**.add(l.itemAt(a,i));  
 }  
 **int** x=0;  
 **for** (**int** i = l.size(); i < l.size()\*2; i++){  
 **if**(x==a)  
 **this**.**equation**.add(1.0f);  
 **else  
 this**.**equation**.add(0.0f);  
 x++;  
 }  
 }  
 **public int** size(){  
 **return equation**.size();  
 }  
 **public void** addEquation(com.company.MyEquation item){  
 ListIterator<Float> i = **equation**.listIterator();  
 ListIterator<Float> j = item.getEquation().listIterator();  
 **for**(; i.hasNext() && j.hasNext();){  
 Float a = i.next();  
 Float b = j.next();  
 i.set(a + b);  
 }  
 }  
 **public void** mul(Float coefficient){  
 **for**(ListIterator<Float> i = **equation**.listIterator(); i.hasNext();){  
 Float next = i.next();  
 i.set(next \* coefficient);  
 }  
 }  
 **public** Float findCoefficient(Float a, Float b){  
 **if** (a == 0.0f) **return** 0.0f;  
 **return** -b/a;  
 }  
 **public** Float at(**int** index){  
 **return equation**.get(index);  
 }  
}

Class LinearEquations

**package** com.company;  
**import** java.util.\*;  
**class** LinearEquations **extends** MyEquation {  
 **private** List<MyEquation> **list** = **new** ArrayList<>();  
 **public** MyEquation get(**int** index){  
 **return list**.get(index);  
 }  
 **public void** push(MyEquation elem){  
 **list**.add(elem);  
 }  
 **public int** size(){  
 **return list**.size();  
 }  
 **public** Float itemAt(**int** i, **int** j){  
 **return list**.get(i).at(j);  
 }  
 **public** Integer max(**int** i){  
 Float MAX=(**float**)(Integer.***MIN\_VALUE***);  
 Integer pos=0;  
 **for**(**int** y=0;y<**list**.size();y++)  
 {  
 **if**(Math.*abs*(**list**.get(i).at(y))>MAX) {  
 MAX =**list**.get(i).at(y);  
 pos=y;  
 }  
 }  
 **return** pos;  
 }  
}

Class Algorithm

**package** com.company;  
**class** Algorithm **extends** LinearEquations {  
 LinearEquations **list** = **null**;  
 **public** Algorithm(LinearEquations system){  
 **list** = system;  
 }  
 **public void** calculate() {  
 **for**(**int** i = 0; i < **list**.size() - 1; i++){  
 **for**(**int** j = i +1; j < **list**.size(); j++){  
 Integer pos=**list**.max(i);  
 Float k = **list**.get(j).findCoefficient(**list**.get(j).at(pos), **list**.get(i).at(pos));  
 **if**(k==0.0f){}  
 **else** {  
 **list**.get(j).mul(k);  
 **list**.get(j).addEquation(**list**.get(i));  
 }  
 }  
 }  
 }  
 **public void** calculateInverse(){  
 **for**(**int** i = 0; i < **list**.size() - 1; i++){  
 **for**(**int** j = i + 1; j < **list**.size(); j++){  
 Float k = **list**.get(j).findCoefficient(**list**.get(j).at(i), **list**.get(i).at(i));  
 **list**.get(j).mul(k);  
 **list**.get(j).addEquation(**list**.get(i));  
 }  
 }  
 **for**(**int** i = **list**.size() - 1; i >= 0; i--){  
 Float k = **list**.get(i).at(i);  
 **list**.get(i).mul(1/k);  
 **for**(**int** j = i - 1; j>=0; j--){  
 Float l = **list**.get(j).findCoefficient(**list**.get(j).at(i), **list**.get(i).at(i));  
 **list**.get(j).mul(l);  
 **list**.get(j).addEquation(**list**.get(i));  
 }  
 }  
  
 }  
}

Class MatrixCalculation

**package** com.company;  
**class** MatrixCalculation {  
 **public** MatrixCalculation() {}  
 **public double** CalculateMatrix(**double**[][] matrix) {  
 **double** calcResult = 0.0;  
 **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) {  
 **int** minorLength = matrix.**length** - 1;  
 **double**[][] minor = **new double**[minorLength][minorLength];  
 **int** dI = 0;  
 **int** dJ = 0;  
 **for** (**int** i = 0; i <= minorLength; i++) {  
 dJ = 0;  
 **for** (**int** j = 0; j <= minorLength; j++) {  
 **if** (i == row) {  
 dI = 1;  
 } **else** {  
 **if** (j == column) {  
 dJ = 1;  
 } **else** {  
 minor[i - dI][j - dJ] = matrix[i][j];  
 }  
 }  
 }  
 }  
 **return** minor;  
 }  
}

Class Main

**package** com.company;  
**public class** Main {  
 **public static** LinearEquations generateSystem(){  
 LinearEquations list = **new** LinearEquations();  
 **int** i;  
 **for** (i = 0; i < ***WIDTH***; i++){  
 MyEquation eq = **new** MyEquation();  
 eq.generate(i,***HEIGHT*** + 1);  
 list.push(eq);  
 }  
 **return** list;  
 }  
 **public static** LinearEquations generateSystem2(LinearEquations l){  
 LinearEquations list = **new** LinearEquations();  
 **int** i;  
 **for** (i = 0; i < ***WIDTH***; i++){  
 MyEquation eq = **new** MyEquation();  
 eq.generate2(i,l);  
 list.push(eq);  
 }  
 **return** list;  
 }  
 **public static void** printSystem(LinearEquations system){  
 **for** (**int** i = 0; i < system.size(); i++){  
 MyEquation temp = system.get(i);  
 String s = **""**;  
 **for** (**int** j = 0; j < temp.size(); j++){  
 s += String.*format*(**"%f; %s"**, system.itemAt(i, j), **"\t"**);  
 }  
 System.***out***.println(s);  
 }System.***out***.println(**""**);  
 }  
 **public static void** printInvert(LinearEquations system){  
 **for** (**int** i = 0; i < system.size(); i++){  
 MyEquation temp = system.get(i);  
 String s = **""**;  
 **for** (**int** j = temp.size()/2; j < temp.size(); j++){  
 s += String.*format*(**"%f; %s"**, system.itemAt(i, j), **"\t"**);  
 }  
 System.***out***.println(s);  
 }System.***out***.println(**""**);  
 }  
 **public static void** printVector(Float [] x){  
 String s = **""**;  
 **for** (**int** i = 0; i < x.**length**; i++){  
 s += String.*format*(**"x%d = %f; "**, i, x[i]);  
 }System.***out***.println(s);  
 }  
 **private static final int *WIDTH*** = 5;  
 **private static final int *HEIGHT*** = 5;  
 **public static void** main(String args[]){  
 LinearEquations list = *generateSystem*();  
 LinearEquations copy = *generateSystem*();  
 System.***out***.println(**"Matrix:"**);  
 *printSystem*(list);  
 LinearEquations list2 = *generateSystem2*(list);  
 **int** i, j;  
 **double**[][] test = **new double**[***WIDTH***][***HEIGHT***];  
 **for**(i=0;i<***WIDTH***;i++)  
 **for**(j=0;j<***HEIGHT***;j++)  
 {  
 test[i][j]=list.itemAt(i,j);  
 }  
 MatrixCalculation mc = **new** MatrixCalculation();  
 **double** Result = mc.CalculateMatrix(test);  
 System.***out***.println(**"Determinate:"**);  
 System.***out***.println(Result);  
 Algorithm alg = **new** Algorithm(list);  
 **try**{  
 alg.calculate();  
 }**catch** (NullPointerException|ArithmeticException e){  
 System.***out***.println(e.getMessage());  
 System.*exit*(0);  
 }  
 Float []x = **new** Float[***WIDTH***];  
 **for**(i = list.size() - 1; i >= 0; i--) {  
 Float sum = 0.0f;  
 **int** pos=0;  
 **for**(j = list.size() - 1; j >=0; j--) {  
 **if**(x[j]!=**null**)  
 sum += list.itemAt(i, j) \* x[j];  
 **if**(list.itemAt(i, j)!=0.0f && x[j]==**null**) {  
 pos = j;  
 }  
 }  
 x[pos] = (list.itemAt(i, list.size()) - sum) / list.itemAt(i, pos);  
 }  
 System.***out***.println(**"System:"**);  
 *printSystem*(list);  
 System.***out***.println(**"Vector:"**);  
 *printVector*(x);  
 System.***out***.println(**"AX-B:"**);  
 **for**(i=0;i<***WIDTH***;i++)  
 {  
 Float sum=0.0f;  
 **for**(j=0;j<***WIDTH***;j++) {  
 sum+=copy.itemAt(i,j)\*x[j];  
 }  
 sum-=copy.itemAt(i,***WIDTH***);  
 System.***out***.print(String.*format*(**"%f; %s"**, sum, **"\t"**));  
 }  
 System.***out***.println(**""**);  
 System.***out***.println(**"Invertible matrix:"**);  
 Algorithm alg2 = **new** Algorithm(list2);  
 **try**{  
 alg2.calculateInverse();  
 }**catch** (NullPointerException|ArithmeticException e){  
 System.***out***.println(e.getMessage());  
 System.*exit*(0);  
 }  
 *printInvert*(list2);  
 }  
}

В результате работы программы получается следующий вывод:

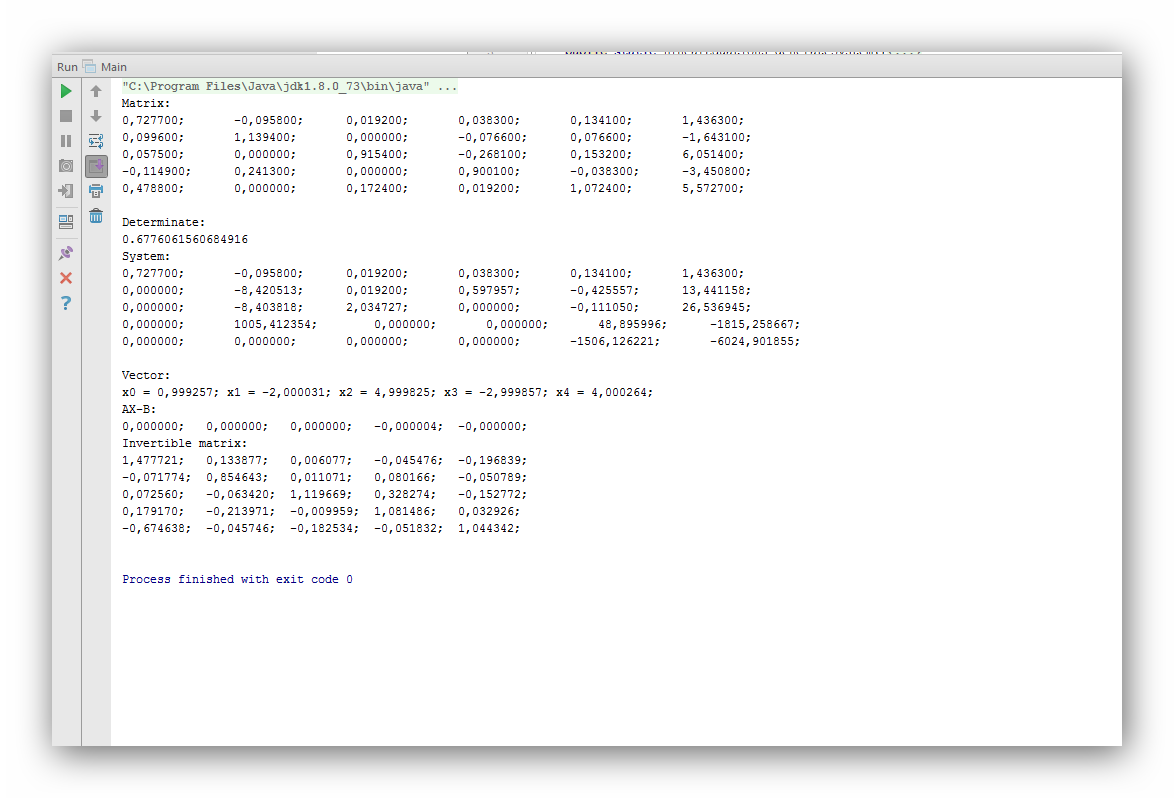


Рис.2. Окно вывода

# 5.Вывод

Несмотря на не слишком большую точность вычислений, ответ совпал с предсказуемым с точностью 3 знака после запятой, что является хорошим результатом. Также в ходе выполнения этой лабораторной мы увидели, что алгоритмы для решения СЛАУ и поиска обратной матрицы очень похожи, и их лучше реализовывать вместе, похожим образом.