Белорусский государственный университет

Факультет прикладной математики и информатики

Кафедра технологий программирования

Стефанович Константин Андреевич

**Лабораторная работа №2**

**Решение СЛАУ методом Квадратного корня(метод Холецкого)**

студента 2 курса 6 группы

**Преподаватель**

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

Минск, 2016

Оглавление

[1. Техническое задание 3](#_Toc463559007)

[2.Алгоритм решения и формулы 3](#_Toc463559008)

[3. Листинг программы 4](#_Toc463559009)

[4.Результаты и вывод 7](#_Toc463559010)

# 1. Техническое задание

1. Методом Квадратного корня найти решение СЛАУ
2. Вычислить определитель матрицы, используя данный метод

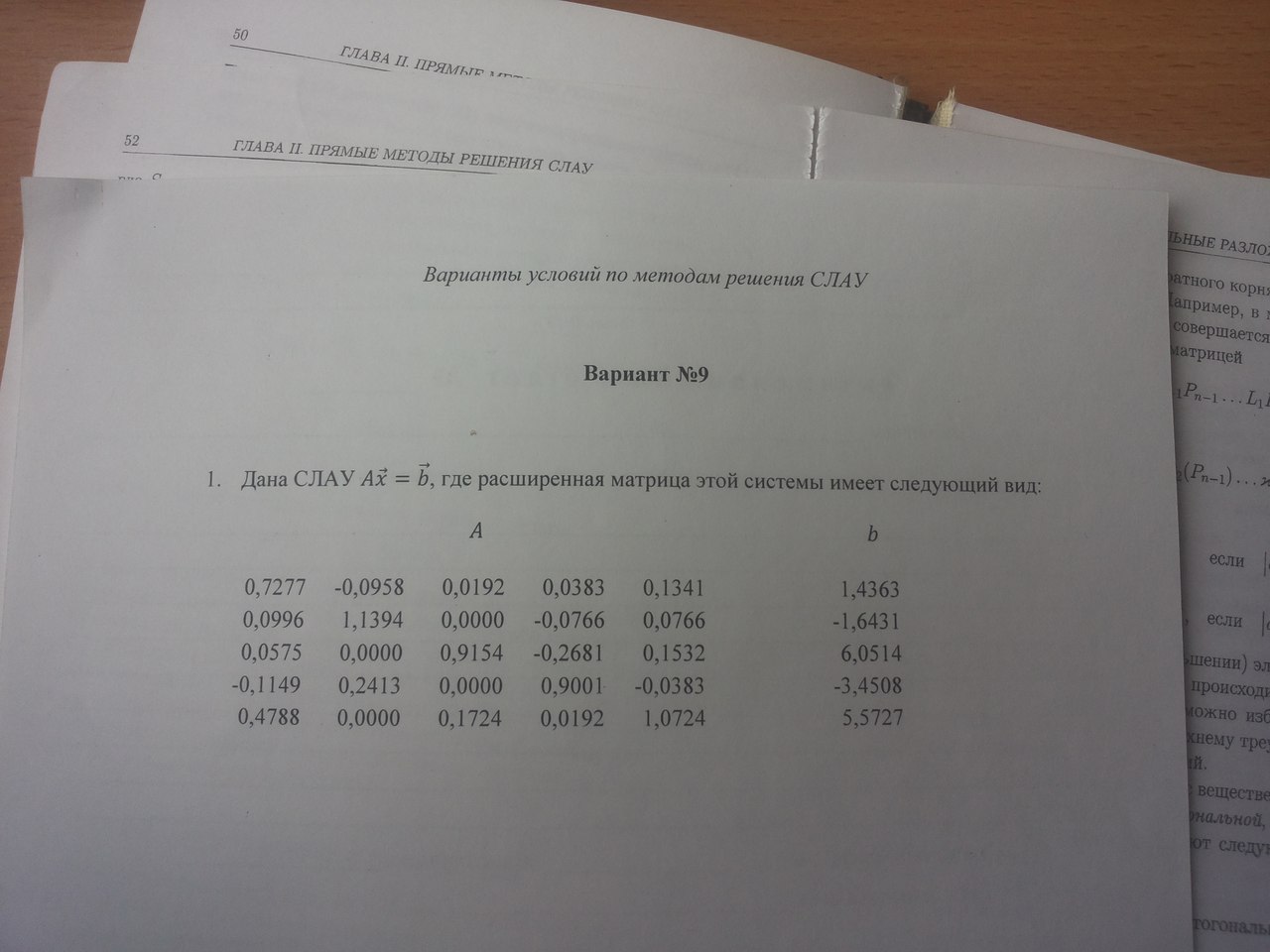


Рис.1. Данные для ввода.

# 2.Алгоритм решения и формулы

Метод квадратного корня по своему идейному содержанию близок к LU-методу. Основное отличие в том, что он дает упрощение для симметричных матриц.ID\_1

Этот метод основан на разложении матрицы *А* в произведение



где S–верхняя треугольная матрица с положительными элементами на главной 





Из условия (1) получаются уравнения



Так как матрица А симметричная, не ограничивая общности, можно считать, что в системе (2) выполняется неравенство *i≤j*. Тогда (2) можно переписать в виде







В частности, при *i=j* получится



 (3)

Далее, при *i<j* получим



По формулам (4) и (5) находятся рекуррентно все ненулевые элементы матрицы S.

Обратный ход метода квадратного корня состоит в последовательном решении двух систем уравнений с треугольными матрицами.



Решения этих систем находятся по рекуррентным формулам





# 3. Листинг программы

Class Main

**package** com.company;  
**public class** Main {  
 **static** Double[][] *mas*=**new** Double[][]  
 {{0.7277,-0.0958,0.0192, 0.0383,0.1341},  
 {0.0996, 1.1394,0.0000,-0.0766,0.0766},  
 {0.0575, 0.0000,0.9154,-0.2681,0.1532},  
 {-0.1149,0.2413,0.0000,0.9001,-0.0383},  
 {0.4788,0.0000,0.1724,0.0192,1.0724}};  
 **static** Double[][] *matrB*=**new** Double[][]  
 {{1.4363}, {-1.6431}, {6.0514}, {-3.4508}, {5.5727}};  
 **public static void** printMatrix(Double[][] x){  
 **for** (**int** i = 0; i < x.**length**; i++){  
 String s = **""**;  
 **for** (**int** j = 0; j < x[i].**length**; j++){  
 s += String.*format*(**"%.4f; %s"**, x[i][j], **"\t"**);  
 }  
 System.***out***.println(s);  
 }System.***out***.println(**""**);  
 }  
 **public static void** printMatrix2(Double[][] x){  
 **for** (**int** i = 0; i < x.**length**; i++){  
 String s = **""**;  
 **for** (**int** j = 0; j < x[i].**length**; j++){  
 s += String.*format*(**"%.4e; %s"**, x[i][j], **"\t"**);  
 }  
 System.***out***.println(s);  
 }System.***out***.println(**""**);  
 }  
 **public static** Double[][] MatrixMultiplection(Double[][] mA,Double[][] mB) {  
 **int** m = mA.**length**;  
 **int** n = mB[0].**length**;  
 **int** o = mB.**length**;  
 Double[][] res = **new** Double[m][n];  
 **for** (**int** i = 0; i < res.**length**; i++) {  
 **for** (**int** j = 0; j < res[0].**length**; j++) {  
 res[i][j]=0.0;  
 }  
 }  
 **for** (**int** i = 0; i < m; i++) {  
 **for** (**int** j = 0; j < n; j++) {  
 **for** (**int** k = 0; k < o; k++) {  
 res[i][j] += mA[i][k] \* mB[k][j];  
 }  
 }  
 }  
 **return** res;  
 }  
 **public static** Double[][] MatrixSub(Double[][] mA,Double[][] mB) {  
 Double[][] res = **new** Double[5][1];  
 **for** (**int** i = 0; i < res.**length**; i++) {  
 res[i][0]=mA[i][0]-mB[i][0];  
 }  
 **return** res;  
 }  
 **public static** Double[][] MatrixTransport(Double[][] m) {  
 Double[][] res = **new** Double[m.**length**][m.**length**];  
 **for** (**int** i = 0; i < res.**length**; i++) {  
 **for** (**int** j = 0; j < res.**length**; j++) {  
 res[i][j]=m[j][i];  
 }  
 }  
  
 **return** res;  
 }  
 **public static void** Calc(Double[][] a,Double[][] b) {  
 a[0][0]=Math.*sqrt*(a[0][0]);  
 **for**(**int** i=1;i<a.**length**;i++)  
 a[0][i]/= a[0][0];  
 b[0][0]/= a[0][0];  
 **for**(**int** i=1;i<a.**length**;i++) {  
 **for** (**int** j = 0; j < a.**length**; j++) {  
 **if** (j < i) a[i][j] = 0.0;  
 **else if** (j == i) {  
 **double** sum = 0.0;  
 **for** (**int** k = 0; k < i; k++)  
 sum += (a[k][i] \* a[k][i]);  
 a[i][i] = Math.*sqrt*(a[i][i] - sum);  
 } **else** {  
 **double** sum = 0.0;  
 **for** (**int** k = 0; k < i; k++)  
 sum += (a[k][i] \* a[k][j]);  
 a[i][j] = (a[i][j] - sum) / a[i][i];  
 }  
  
 }  
 **double** sum = 0.0;  
 **for** (**int** k = 0; k < i; k++)  
 sum += (a[k][i] \* b[k][0]);  
 b[i][0] = (b[i][0] - sum) / a[i][i];  
 }  
 }  
 **public static void** main(String args[]){  
 System.***out***.println(**"Matrix:"**);  
 *printMatrix*(*mas*);  
 Double[][] res=*MatrixTransport*(*mas*);  
 Double[][] y=*MatrixMultiplection*(res,*mas*);  
 System.***out***.println(**"Symmetric Matrix:"**);  
 *printMatrix*(y);  
 Double[][]copy =y.clone();  
 Double[][] y2=*MatrixMultiplection*(res,*matrB*);  
 System.***out***.println(**"Vector Y:"**);  
 *printMatrix*(y2);  
 Double[][] clone=y2.clone();  
 *Calc*(y,y2);  
 System.***out***.println(**"S = "**);  
 *printMatrix*(y);  
 Double [][] x = **new** Double[5][1];  
 **int** j;  
 **for**(**int** i = 4; i >= 0; i--) {  
 Double sum = 0.0;  
 **for**(j = 4; j > i; j--) {  
 sum += y[i][j] \* x[j][0];  
 }  
 x[i][0] = (y2[i][0] - sum) / y[i][j];  
 }  
 System.***out***.println(**"Vector X:"**);  
 *printMatrix*(x);  
 Double det=1.0;  
 **for**(**int** i=0;i<5;i++)  
 det\*=y[i][i];  
 System.***out***.println(**"Determinate :"**);  
 System.***out***.println(det);  
 System.***out***.println();  
 Double[][] AX=*MatrixMultiplection*(y,x);  
 System.***out***.println(**"AX-B:"**);  
 *printMatrix2*(*MatrixSub*(AX,y2));  
 Double[][] SX=*MatrixMultiplection*(copy,x);  
 System.***out***.println(**"SX-F:"**);  
 *printMatrix2*(*MatrixSub*(SX,clone));  
  
 }  
}

# 4.Результаты и 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 методом квадратного корня работает намного быстрее, чем метод Гаусса, и использовать его намного более выгодно.

В результате работы программы получается следующий вывод:

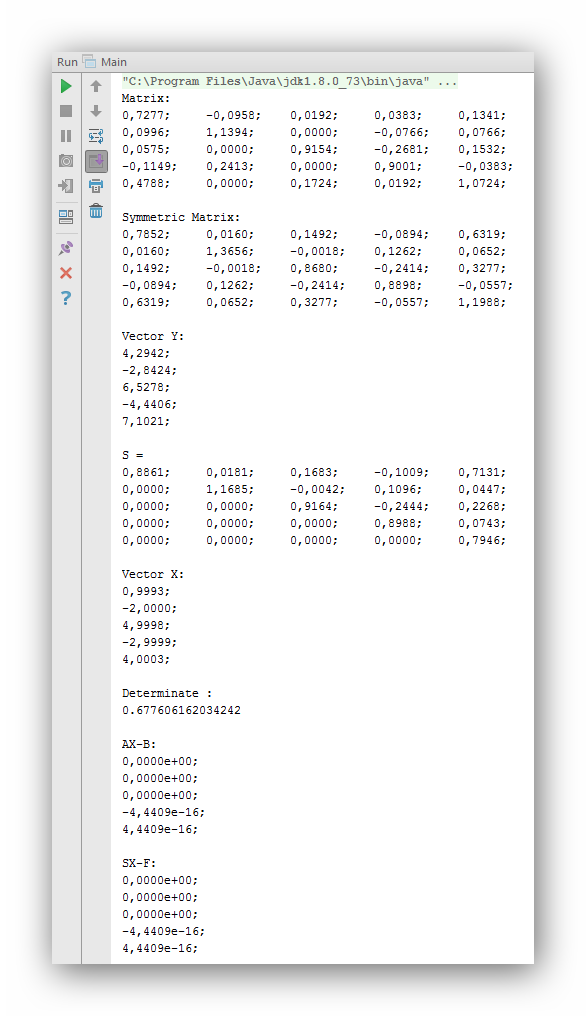


Рис.2. Окно вывода