# Содержание

[Содержание 3](#_Toc126876926)

[Введение 4](#_Toc126876927)

[1. Анализ объекта 5](#_Toc126876928)

[1.1 Постановка задачи 5](#_Toc126876929)

[1.2 Теоретическая основа метода квадратного корня для решения линейных систем 6](#_Toc126876930)

[1.3 Реализация метода квадратного корня для решения линейных систем. 9](#_Toc126876931)

[2. Разработка алгоритма работы программы 11](#_Toc126876932)

[3. Описание основных компонентов программы и последовательности разработки 12](#_Toc126876933)

[4. Исходный код программы 13](#_Toc126876934)

[5. Проверка корректности работы программы в различных режимах 14](#_Toc126876935)

[Заключение 15](#_Toc126876936)

[Литература 16](#_Toc126876937)

# Введение

В этой курсовой работе рассматривается вопрос разработки приложения для решения систем линейных уравнений с симметричной матрицей методом квадратного корня. Целью данного проекта является разработка полезного инструмента, который может быть использован для решения сложных математических задач. Для реализации данного проекта будет использован язык Java и разработано оконное приложение.

Решение систем линейных уравнений является важным инструментом в математике и науке данных. Метод квадратного корня является одним из эффективных методов решения систем линейных уравнений с симметричной матрицей, и мы собираемся использовать его в нашем приложении.

Данный метод позволяет решать системы линейных уравнений с большими размерностями матрицы, чем методы, такие как Гауссова или Жордановы методы. Решение системы линейных уравнений с помощью метода квадратного корня является быстрым и точным, и поэтому важно реализовать данный метод в приложении.

,

# Анализ объекта

Метод квадратного корня применяется в том случае, когда матрица А симметричная, то есть:

aij = aji (i, j = 1, 2, …, n).

Кроме того, матрица должна быть невырожденной, то есть её определитель не должен равняться нулю (det(A)0). Таким образом, система будет иметь единственное решение.

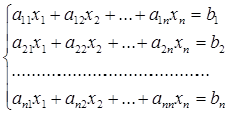
Метод квадратного корня дает большой выигрыш во времени по сравнению с другими методами (например, методом Гаусса), так как, во-первых, существенно уменьшает число умножений и делений (почти в два раза для больших n), во-вторых, позволяет накапливать сумму произведений без записи промежуточных результатов.

https://studbooks.net/imag_/43/230414/image044.png

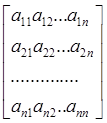
Всего метод квадратных корней требует операций умножения и деления (примерно в два раза меньше, чем метод Гаусса), а также n операций извлечения корня.

## 1.1 Постановка задачи

К решению систем линейных уравнений сводятся многочисленные практические задачи. Запишем еще раз систему n линейных алгебраических уравнений с n неизвестными:



Совокупность коэффициентов (aij), неизвестных (хi) и свободных членов (bi) этой системы запишем в виде матриц

https://studbooks.net/imag_/43/230414/image047.png

A=, X=, B=

https://studbooks.net/imag_/43/230414/image048.png

Используя понятие матрицы , систему уравнений можно записать в матричном виде:

Ax=b

Таким образом, задача состоит в том, чтобы вычислить столбец неизвестных, используя метод квадратного корня.

## 1.2 Теоретическая основа метода квадратного корня для решения линейных систем

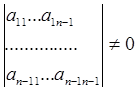
Пусть дана симметричная система линейных уравнений в матричном виде Ах=b

К ee решению может быть применена идея разложения матрицы А в произведение двух матриц специального вида. Основанием для этого служит следующая теорема:

Теорема. Какова бы ни была матрица А с отличными от нуля глав-ными минорами:

https://studbooks.net/imag_/43/230414/image049.png

, … ,

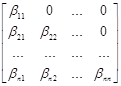
https://studbooks.net/imag_/43/230414/image050.png 

ее всегда можно разложить в произведение двух треугольных матриц:

A=BC,

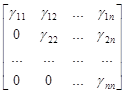
где В - левая треугольная матрица:

В=



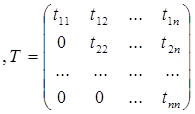
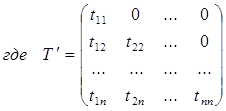
С - правая треугольная матрица:

С=



Так как данная матрица симметрична, то она раскладывается на произведение двух взаимно транспонированных треугольных матриц:

А = Т Т,



Найдем элементы tij матрицы Т. Для этого перемножим T и T' между собой и приравняем полученное к исходной матрице:

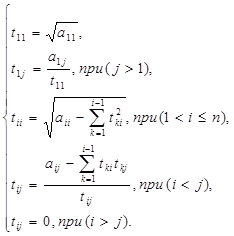
t211 = a11 , t11 t12 =a12 , … , t11 t1n = a1n ,

t212 + t222= a22 , … , t12 t1n + t22 t2n= a2n ,

…………………………………………………………………..

t21n + t22n +…+ t2nn = ann

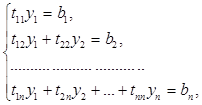
Получим следующие формулы для определения tij:

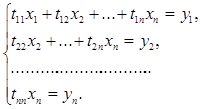


Далее, решение системы сводится к решению двух треугольных систем. Действительно, равенство равносильно двум равенствам:

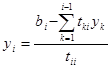
T'y=b и Tx=y.

Запишем в развернутом виде системы:

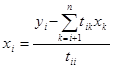




И из этих систем последовательно находим

https://studbooks.net/imag_/43/230414/image059.png

при (i>1)

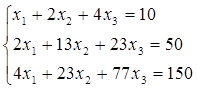
https://studbooks.net/imag_/43/230414/image061.png

при (i<n).

## 1.3 Реализация метода квадратного корня для решения линейных систем.

Для того, чтобы удостовериться, что программа будет работать правильно, решим конкретный пример вручную, а затем сравним полученный результат с результатом программы.

**Задача.Пусть дана система линейных уравнений:**



Этой системе соответствуют: матрица коэффициентов А и столбец свободных членов b:

А= b=

https://studbooks.net/imag_/43/230414/image065.png

В коде программы прописано, что пользователь может ввести матрицу размерности не более, чем 10Ч10, но, вводя коэффициенты, необходимо помнить о том, что матрица должна быть симметричной. В противном случае программа сработает неправильно.

Найдем элементы матрицы Т. Таким образом, получим:

t211 = 1 t11 = 1,

t11 t12 =2 t12 = 2,

t11 t13 = 4 t13 = 4,

t212 + t222= 13 t22 = 3,

t12 t13 + t22 t23= 23 t23 =5,

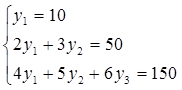
t213 + t223 +t233 = 77 t33 =6

Таким образом, матрица А раскладывается в произведение матриц T' и Т:

А=\*



Решим систему T'y=b.



Решим систему Тх=у. Суть её состоит в том, что полученные значения х подставляются в исходную матрицу и высчитывается значение столбца свободных членов b. Если оно совпадает с исходным, то решение найдено верно.

# Разработка алгоритма работы программы

Алгоритм работы приложения является простым. Первым делом пользователь запускает приложение и вводит всю необходимую информацию любым удобным для него способом. У него на выбор есть варианты вручную добавить все данные или же загрузить информацию из файлов.

После внесения всех данных пользователь нажимает на кнопку, после чего приложение приступает непосредственно к решению уравнения используя формулы из предыдущего раздела.

После решения уравнения приложение выводит на экран под таблицей матрицы ответ на решение (найденные коэфициенты для каждой неизвестной).

Теперь пользователь может либо продолжить работу с приложением внеся новые данные, либо сохранить полученный ответ в файл и выйти из приложения.

# Описание основных компонентов программы и последовательности разработки

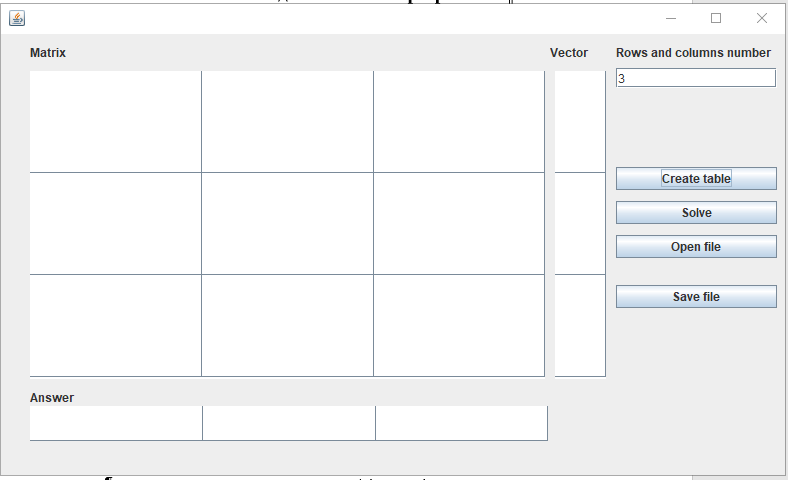
Приложение написано на Java, поэтому основными компонентами являются поля и методы классов.

Для начала был создан класс Matrix который содержит в себе данные о количество строк и столбцов, а также саму матрицу.

После этого был создан вспомогательный класс Solver, в который передавались необходимые данные и шло решение уравнения.

После этих шагов создавалась визуальная форма для отображения и взаимодействия с пользователем через Swing.

Данная форма представлена на рисунке ниже.



Данная форма доступна для редактирования, поэтому при необходимости можно задать лубой размер матрицы и вручную заполнять ее данными. Кроме того, предусмотрена загрузка и сохранение данных.

После загрузки данных из файла все данные уравнения сразу же отображаются на экране.

После нажатия на кнопку решения, все неизвестные уравнения будут отображены прямо под матрицей.

# Исходный код программы

package square\_matrix.model;  
  
public class Matrix {  
 // A two-dimensional array to store the elements of the matrix  
 private double[][] elements;  
  
 // Number of rows of the matrix  
 private int rows;  
  
 // Number of columns of the matrix  
 private int columns;  
  
 // A one-dimensional array to store the matrix elements as a packed vector  
 public double[] vectors;  
  
 /\*\*  
 \* Constructs a matrix with the given number of rows and columns  
 \*  
 \* @param rows number of rows  
 \* @param columns number of columns  
 \*/  
 public Matrix(int rows, int columns) {  
 this.rows = rows;  
 this.columns = columns;  
 this.elements = new double[rows][columns];  
 }  
  
 /\*\*  
 \* Constructs a matrix from the given two-dimensional array  
 \*  
 \* @param matrix two-dimensional array of elements  
 \*/  
 public Matrix(double[][] matrix) {  
 this.rows = matrix.length;  
 this.columns = matrix[0].length;  
  
 // Ensure all the rows of the matrix have the same length  
 for (int var2 = 0; var2 < this.rows; ++var2) {  
 if (matrix[var2].length != this.columns) {  
 throw new IllegalArgumentException("All rows must have the same length.");  
 }  
 }  
 this.elements = matrix;  
 }  
  
 /\*\*  
 \* Constructs a matrix from the given packed vector and the number of rows  
 \*  
 \* @param rows packed vector of elements  
 \* @param var2 number of rows  
 \*/  
 public Matrix(double[] rows, int var2) {  
 this.rows = var2;  
 this.columns = var2 != 0 ? rows.length / var2 : 0;  
  
 // Ensure that the array length is a multiple of the number of rows  
 if (var2 \* this.columns != rows.length) {  
 throw new IllegalArgumentException("Array length must be a multiple of m.");  
 } else {  
 this.elements = new double[var2][this.columns];  
 for (int var3 = 0; var3 < var2; ++var3) {  
 for (int var4 = 0; var4 < this.columns; ++var4) {  
 this.elements[var3][var4] = rows[var3 + var4 \* var2];  
 }  
 }  
 }  
 }  
  
 public double[] getVectors() {  
 return vectors;  
 }  
  
 /\*\*  
 \* Sets the vector representation of this matrix.  
 \*  
 \* @param vectors the new vector representation  
 \*/  
 public void setVectors(double[] vectors) {  
 this.vectors = vectors;  
 }  
  
 public double[][] getArray() {  
 return elements;  
 }  
}

package square\_matrix.service;  
  
import square\_matrix.model.Matrix;  
  
import javax.swing.JFileChooser;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Arrays;  
import java.util.Scanner;  
  
// File specification:  
// [A], where A - number of rows and columns  
// [1 -2 -3 4]  
// [-5 -6 7 8] - where numbers - matrix data  
// [] empty line  
// [9 8 7] - where number - vector data  
  
/\*\*  
 \* FileOpener is a class used for opening a file containing matrix and vector data.  
 \*\*/  
public class FileOpener {  
 /\*  
 The method openFile is used for opening a file that contains matrix and vector data.  
 @return Returns the matrix and vector data in the form of a Matrix object.  
 @throws FileNotFoundException If the selected file is not found, this exception is thrown.  
 \*/  
 public static Matrix openFile() throws FileNotFoundException {  
 // Creates an instance of JFileChooser to select a file.  
 JFileChooser fileChooser = new JFileChooser();  
 // The showDialog method returns the option selected by the user.  
 int ret = fileChooser.showDialog(null, "Open file");  
 // If the user approves the option, the selected file is processed.  
 if (ret == JFileChooser.APPROVE\_OPTION) {  
 // The selected file is stored in the file variable.  
 File file = fileChooser.getSelectedFile();  
 // A scanner is created to read the data from the file.  
 Scanner scanner = new Scanner(file);  
 // The number of rows and cols in the matrix is read from the file.  
 int rowAndCols = scanner.nextInt();  
 // The matrix data is stored in a 2D array.  
 double[][] matrixData = new double[rowAndCols][rowAndCols];  
 // The matrix data is read from the file.  
 for (int i = 0; i < rowAndCols; i++) {  
 for (int j = 0; j < rowAndCols; j++) {  
 matrixData[i][j] = scanner.nextDouble();  
 }  
 }  
 // Skipping two lines in the file.  
 scanner.nextLine();  
 scanner.nextLine();  
 // The vector data is read from the file as a string.  
 String vectorData = scanner.nextLine();  
 // The vector data is converted to an array of double values.  
 double[] vector = Arrays.stream(vectorData.split(" "))  
 .mapToDouble(Double::valueOf)  
 .toArray();  
 // A matrix object is created using the matrix data.  
 Matrix matrix = new Matrix(matrixData);  
 // The vector data is set for the matrix object.  
 matrix.setVectors(vector);  
 // The matrix object is returned.  
 return matrix;  
 }  
 // If no file is selected, null is returned.  
 return null;  
 }  
}

package square\_matrix.service;  
  
  
import java.text.DecimalFormat;  
  
/\*\*  
 \* Solver class that provides a solution for a system of linear equations.  
 \*/  
public class Solver {  
 /\*  
 Format for the rounded values in the solution.  
 \*/  
 private static final DecimalFormat decimalFormat = new DecimalFormat("###.###");  
  
 public static boolean isMatrixSymmetric(double matrix[][]) {  
 int i, j, flag = 1;  
 int row = matrix.length;  
 int col = matrix[0].length;  
  
 // Nested for loop for matrix iteration  
 // Outer loop for rows  
 for (i = 0; i < row; i++) {  
 // Inner loop for columns  
 for (j = 0; j < col; j++) {  
 // Print matrix  
 System.out.print(matrix[i][j] + "\t");  
 }  
  
 System.out.println("");  
 }  
  
 // Matrix 2  
 // Finding transpose of the matrix  
 double[][] transpose = new double[row][col];  
  
 // Again, nested for loop for matrix iteration  
 // Outer loop for rows  
 for (i = 0; i < row; i++) {  
 // Inner loop for columns  
 for (j = 0; j < col; j++) {  
 // Print matrix elements  
 transpose[j][i] = matrix[i][j];  
 }  
 }  
  
 // Condition check over Matrix 1 with Matrix 2  
 if (row == col) {  
 // Outer loop for rows  
 for (i = 0; i < row; i++) {  
 // Inner loop for columns  
 for (j = 0; j < col; j++) {  
 // Comparing two matrices  
 if (matrix[i][j] != transpose[i][j]) {  
 flag = 0;  
 break;  
 }  
 }  
  
 // Setting a flag value for symmetric matrix  
 if (flag == 0) {  
 return false;  
 } else {  
 return true;  
 }  
 }  
 }  
  
 // If it isn't a square matrix  
 // then it can't be a symmetric matrix  
 else {  
 return false;  
 }  
 return false;  
 }  
  
 public static double[] solve(double[][] A, double[] b) {  
 int n = A.length;  
 double[] x = new double[n];  
  
 // Compute the Cholesky decomposition of A  
 double[][] L = new double[n][n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j <= i; j++) {  
 double sum = 0;  
 for (int k = 0; k < j; k++) {  
 sum += L[i][k] \* L[j][k];  
 }  
 if (i == j) {  
 L[i][i] = Math.sqrt(A[i][i] - sum);  
 } else {  
 L[i][j] = (A[i][j] - sum) / L[j][j];  
 }  
 }  
 }  
  
 // Solve L \* y = b using forward substitution  
 double[] y = new double[n];  
 for (int i = 0; i < n; i++) {  
 double sum = 0;  
 for (int j = 0; j < i; j++) {  
 sum += L[i][j] \* y[j];  
 }  
 y[i] = (b[i] - sum) / L[i][i];  
 }  
  
 // Solve L^T \* x = y using backward substitution  
 for (int i = n - 1; i >= 0; i--) {  
 double sum = 0;  
 for (int j = i + 1; j < n; j++) {  
 sum += L[j][i] \* x[j];  
 }  
 x[i] = (y[i] - sum) / L[i][i];  
 }  
  
 round(x);  
 return x;  
 }  
  
 /\*\*  
 \* Rounds the values in the given array.  
 \*  
 \* @param rowPackedCopy the array of values to be rounded  
 \*/  
 private static void round(double[] rowPackedCopy) {  
 // loop through the array  
 for (int i = 0; i < rowPackedCopy.length; i++) {  
 // round each value in the array  
 rowPackedCopy[i] = Double.parseDouble(decimalFormat.format(rowPackedCopy[i]));  
 }  
 }  
}

package square\_matrix;  
  
import square\_matrix.model.Matrix;  
import square\_matrix.service.FileOpener;  
import square\_matrix.service.Solver;  
  
import javax.swing.JButton;  
import javax.swing.JFileChooser;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JOptionPane;  
import javax.swing.JTable;  
import javax.swing.JTextField;  
import javax.swing.table.DefaultTableModel;  
import java.awt.EventQueue;  
import java.io.File;  
import java.io.FileWriter;  
import java.util.Arrays;  
import java.util.stream.Collectors;  
  
public class MainForm {  
 private JFrame frame;  
 private JTable matrixTable;  
 private JTable answersTable;  
 private JTable vectorTable;  
 private JTextField rowsAndColsTextField;  
 DefaultTableModel matrixTableModel = new DefaultTableModel();  
 DefaultTableModel answerTableModel = new DefaultTableModel();  
 DefaultTableModel vectorTableModel = new DefaultTableModel();  
  
 /\*\*  
 \* Launch the application.  
 \*/  
 public static void main(String[] args) {  
 EventQueue.invokeLater(() -> {  
 try {  
 MainForm window = new MainForm();  
 window.frame.setVisible(true);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 /\*\*  
 \* Create the application.  
 \*/  
 public MainForm() {  
 initialize();  
 }  
  
 /\*\*  
 \* Initialize the contents of the frame.  
 \*/  
 private void initialize() {  
 frame = new JFrame();  
 frame.setBounds(100, 100, 800, 480);  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 frame.getContentPane().setLayout(null);  
  
 matrixTable = new JTable();  
 matrixTable.setBounds(29, 37, 515, 308);  
 frame.getContentPane().add(matrixTable);  
  
 answersTable = new JTable();  
 answersTable.setBounds(29, 372, 518, 35);  
 frame.getContentPane().add(answersTable);  
  
 vectorTable = new JTable();  
 vectorTable.setBounds(554, 37, 51, 308);  
 frame.getContentPane().add(vectorTable);  
  
 JLabel label = new JLabel("Rows and columns number");  
 label.setBounds(615, 11, 216, 14);  
 frame.getContentPane().add(label);  
  
 rowsAndColsTextField = new JTextField("3");  
 rowsAndColsTextField.setBounds(615, 34, 161, 20);  
 frame.getContentPane().add(rowsAndColsTextField);  
 rowsAndColsTextField.setColumns(10);  
  
 JButton btnNewButton = new JButton("Create table");  
 btnNewButton.addActionListener(e -> createTable(-1));  
 btnNewButton.setBounds(615, 133, 161, 23);  
 frame.getContentPane().add(btnNewButton);  
  
 JButton button = new JButton("Solve");  
 button.addActionListener(e -> solve());  
 button.setBounds(615, 167, 161, 23);  
 frame.getContentPane().add(button);  
  
 JLabel label\_2 = new JLabel("Answer");  
 label\_2.setBounds(29, 356, 147, 14);  
 frame.getContentPane().add(label\_2);  
  
 JLabel label\_3 = new JLabel("Vector");  
 label\_3.setBounds(549, 11, 77, 14);  
 frame.getContentPane().add(label\_3);  
  
 JLabel label\_4 = new JLabel("Matrix");  
 label\_4.setBounds(29, 11, 129, 14);  
 frame.getContentPane().add(label\_4);  
  
 JButton openFileButton = new JButton("Open file");  
 openFileButton.addActionListener(e -> openFile());  
 openFileButton.setBounds(615, 201, 161, 23);  
 frame.getContentPane().add(openFileButton);  
  
 JButton saveFileButton = new JButton("Save file");  
 saveFileButton.addActionListener(e -> saveFile());  
 saveFileButton.setBounds(615, 251, 161, 23);  
 frame.getContentPane().add(saveFileButton);  
 }  
  
 /\*\*  
 \* opens a file using FileOpener and sets the matrix and vector values to the table models  
 \*/  
 private void openFile() {  
 try {  
 // open the file using FileOpener and get the matrix  
 Matrix matrix = FileOpener.openFile();  
 if (matrix != null) {  
 // get the array representation of the matrix  
 double[][] array = matrix.getArray();  
 int rows = array.length;  
 int cols = array[0].length;  
 // create the table with the specified number of rows and columns  
 createTable(rows);  
 // set the number of rows in the matrix table model  
 matrixTableModel.setRowCount(rows);  
 // set the number of columns in the matrix table model  
 matrixTableModel.setColumnCount(cols);  
 // set the values of the matrix in the matrix table model  
 for (int i = 0; i < array.length; i++) {  
 for (int j = 0; j < array[0].length; j++) {  
 matrixTableModel.setValueAt(array[i][j], i, j);  
 }  
 }  
 // get the vectors from the matrix  
 double[] vectors = matrix.getVectors();  
 // set the number of rows in the vector table model  
 vectorTableModel.setRowCount(vectors.length);  
 // set the number of columns in the vector table model  
 vectorTableModel.setColumnCount(1);  
 // set the values of the vectors in the vector table model  
 for (int i = 0; i < vectors.length; i++) {  
 vectorTableModel.setValueAt(vectors[i], i, 0);  
 }  
 }  
 } catch (Exception e) {  
 // print the stack trace and show an error message  
 e.printStackTrace();  
 JOptionPane.showMessageDialog(frame, "Error while working with file. Please check your file.");  
 }  
 }  
  
 /\*\*  
 \* saves the matrix and vectors to a file  
 \*/  
 private void saveFile() {  
 try {  
 // create a file chooser to select the file to save  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setDialogTitle("Specify a file to save");  
 int userSelection = fileChooser.showSaveDialog(frame);  
  
 if (userSelection == JFileChooser.APPROVE\_OPTION) {  
 // get the selected file to save  
 File fileToSave = fileChooser.getSelectedFile();  
 // get the matrix data  
 double[][] matrix = readMatrix();  
 // get the vector data  
 double[] vector = readVector();  
 // get the results  
 double[] results = readResult();  
 System.out.println("Save as file: " + fileToSave.getAbsolutePath());  
 // create a file writer to write to the file  
 FileWriter fw = new FileWriter(fileToSave);  
 // write the number of rows and columns in the first line  
 fw.append(String.valueOf(matrixTable.getRowCount()));  
 fw.append("\r\n");  
 // write the matrix data  
 for (double[] doubles : matrix) {  
 String matrixRow = Arrays.stream(doubles)  
 .mapToObj(String::valueOf)  
 .collect(Collectors.joining(" "));  
 fw.append(matrixRow).append("\r\n");  
 }  
 fw.append("\r\n");  
  
 String vectorRow = Arrays.stream(vector)  
 .mapToObj(String::valueOf)  
 .collect(Collectors.joining(" "));  
 fw.append(vectorRow);  
 fw.append("\r\n");  
 fw.append("Results");  
 fw.append("\r\n");  
 String result = Arrays.stream(results)  
 .mapToObj(String::valueOf)  
 .collect(Collectors.joining(" "));  
 fw.append(result);  
 fw.flush();  
 fw.close();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 JOptionPane.showMessageDialog(frame, "Something goes wrong while saving file.");  
 }  
 }  
  
 private void createTable(int rowsAndCols) {  
 matrixTable.removeAll();  
 answersTable.removeAll();  
 vectorTable.removeAll();  
 if (rowsAndCols < 0) {  
 rowsAndCols = Integer.parseInt(rowsAndColsTextField.getText());  
 }  
 matrixTable.setModel(matrixTableModel);  
 matrixTableModel.setColumnCount(rowsAndCols);  
 matrixTableModel.setRowCount(rowsAndCols);  
 matrixTable.setAutoResizeMode(JTable.AUTO\_RESIZE\_ALL\_COLUMNS);  
 matrixTable.setRowHeight(308 / rowsAndCols);  
  
 answersTable.setModel(answerTableModel);  
 answerTableModel.setColumnCount(rowsAndCols);  
 answerTableModel.setRowCount(1);  
 answersTable.setAutoResizeMode(JTable.AUTO\_RESIZE\_ALL\_COLUMNS);  
 answersTable.setRowHeight(35);  
  
 vectorTable.setModel(vectorTableModel);  
 vectorTableModel.setRowCount(rowsAndCols);  
 vectorTableModel.setColumnCount(1);  
 vectorTable.setAutoResizeMode(JTable.AUTO\_RESIZE\_ALL\_COLUMNS);  
 vectorTable.setRowHeight(308 / rowsAndCols);  
 }  
  
 private void solve() {  
 try {  
 double[][] matrixData = readMatrix();  
 if(!Solver.isMatrixSymmetric(matrixData)){  
 JOptionPane.showMessageDialog(frame, "The matrix is not symmetric");  
 return;  
 }  
 double[] vector = readVector();  
 double[] result = Solver.solve(matrixData, vector);  
 printResult(result);  
 } catch (Exception e) {  
 e.printStackTrace();  
 JOptionPane.showMessageDialog(frame, "Not all fields are completed or it contains wrong values.");  
 }  
 }  
  
 private double[] readVector() {  
 int rows = matrixTableModel.getRowCount();  
 double[] vector = new double[rows];  
 for (int i = 0; i < rows; i++) {  
 vector[i] = Double.parseDouble(vectorTableModel.getValueAt(i, 0).toString());  
 }  
 return vector;  
 }  
  
 private double[] readResult() {  
 int cols = matrixTableModel.getColumnCount();  
 double[] vector = new double[cols];  
 for (int i = 0; i < cols; i++) {  
 vector[i] = Double.parseDouble(answerTableModel.getValueAt(0, i).toString());  
 }  
 return vector;  
 }  
  
 private double[][] readMatrix() {  
 int rows = matrixTableModel.getRowCount();  
 int cols = matrixTableModel.getColumnCount();  
 double[][] matrixData = new double[rows][cols];  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 matrixData[i][j] = Double.parseDouble(matrixTableModel.getValueAt(i, j).toString());  
 }  
 }  
 return matrixData;  
 }  
  
 private void printResult(double[] result) {  
 answerTableModel.setColumnCount(result.length);  
 for (int i = 0; i < result.length; i++) {  
 answerTableModel.setValueAt(result[i], 0, i);  
 }  
 }  
}

# Проверка корректности работы программы в различных режимах

Запустим приложение и посмотрим работает ли оно вообще.

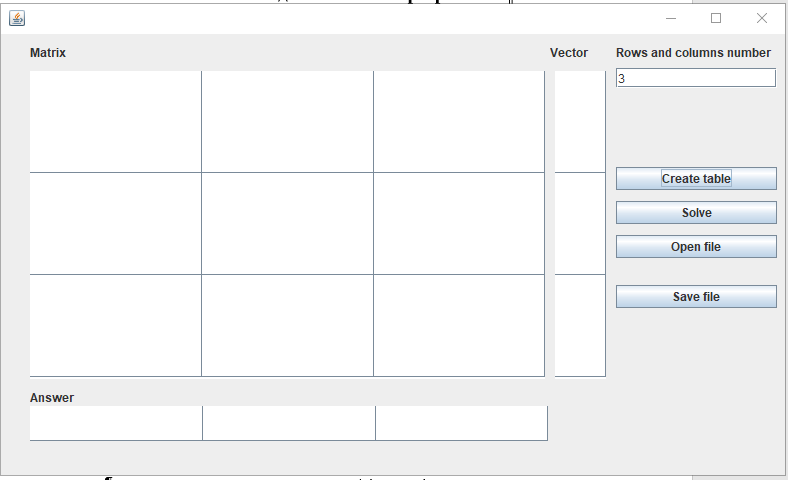


Рисунок 1 – Запуск приложения

Как видно из рисунка, приложение запускается и показывает три таблицы. Попробуем создать матрицу с выбранным количеством строк и столбцов.

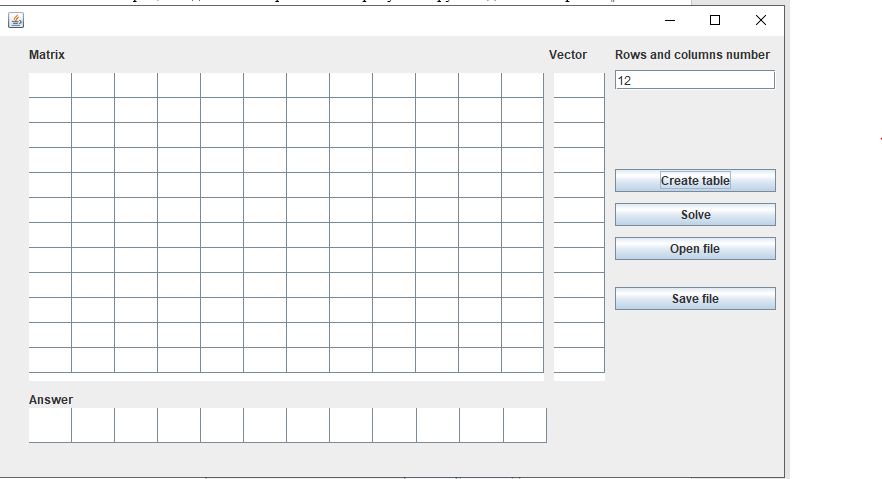


Рисунок 2 – создание таблицы заданной сложности

Матрица создается без проблем. Попробуем загрузить данные из файла.

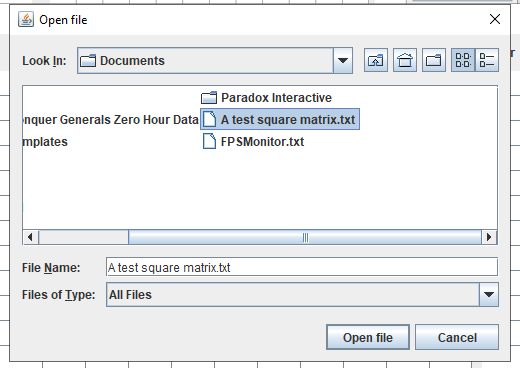


Рисунок 3 – Чтение из файла

При нажатии на кнопку загрузить мы видим окно, в котором необходимо выбрать файл с данными уравнения.

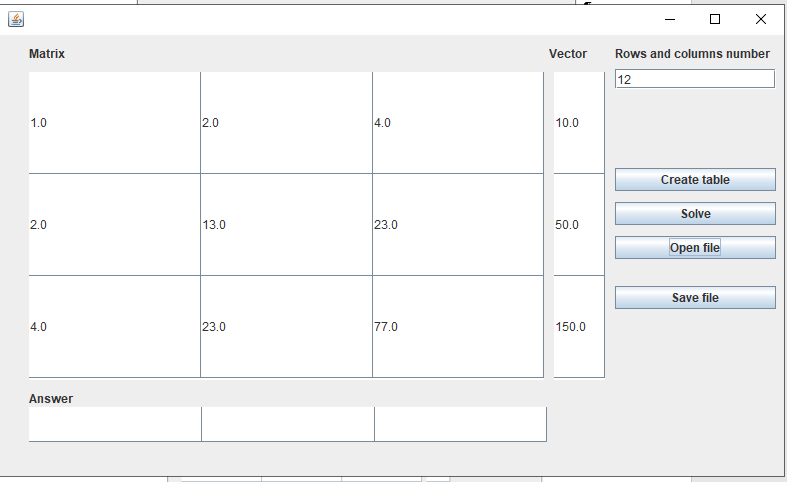


Рисунок 4 – Результат чтения из файла

После выбора файла в приложении отображается все его данные и мы можем уравнение решить.

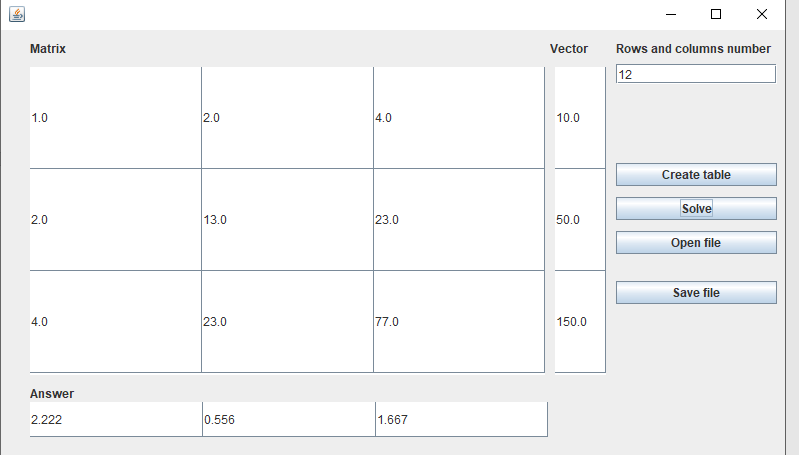


Рисунок 5 – Результат выполнения

После нажатия на кнопку решить внизу под матрицей отображаются все ответы.

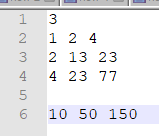


Рисунок 6 – Спецификация загружаемого файла

Файл для загрузки должен иметь следующий вид:

Первая строка – количество строк и столбцов

Вторая и последующие строки – данные матрицы

Затем пустая строка для разделения

Последняя строка – вектор

После решения уравнения его можно сохранить.

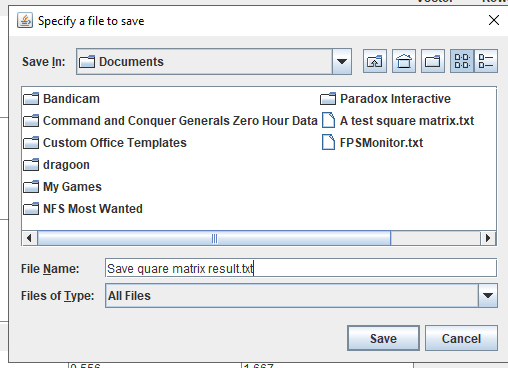


Рисунок 7 – Сохранение результата

После нажатия на кнопку сохранения нам предложат выбрать название файла и метсо его сохранения.

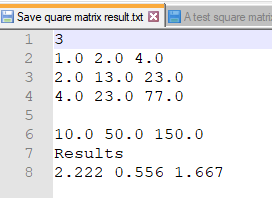


Рисунок 8 Вид сохраненного файла

После всех этих манипуляций файл сохраняется в том же виде, за исключением добавления в его конец результата выислений.

На случай незаполнения всех полей предусмотрен выброс ошибок.

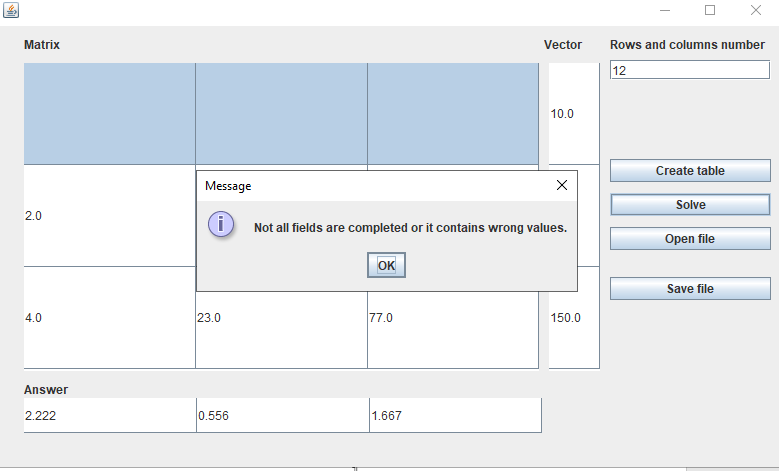


Рисунок 9 - Вызов ошибки при попытке решение без заполненных полей

Так же в приложении имеется проверка на правильность ввода.

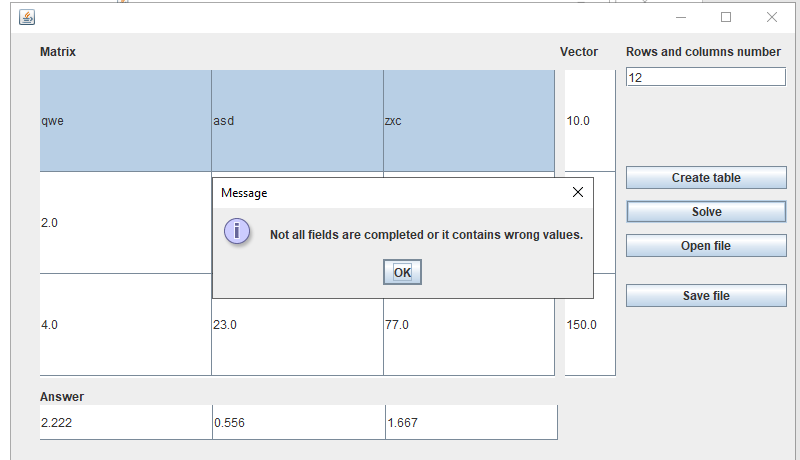


Рисунок 10 – Вызов ошибки при некорректном вводе

# Заключение

В заключение, разработанное приложение для решения систем линейных уравнений с симметричной матрицей методом квадратного корня демонстрирует высокую функциональность и практичность. Это оконное приложение написано на языке Java с использованием фреймворка Swing, позволяет пользователю выбрать размер матрицы и внести ее данные вручную или загрузить их через файл.

Приложение также включает проверки на симметричность, правильность ввода и функциональность вывода ответа на экран и сохранение его в файл. Таким образом, приложение предоставляет пользователю удобный интерфейс для решения систем линейных уравнений с симметричной матрицей, а также обеспечивает необходимую точность и безопасность решения.

Результаты работы программы показывают, что она работает быстро и эффективно, и выдает точные результаты, соответствующие ожиданиям. Тесты, проведенные на различных входных данных, показали высокую точность решения и отсутствие ошибок.

В заключение, можно сказать, что данный проект является успешным примером использования метода квадратного корня для решения систем линейных уравнений с симметричной матрицей. Это приложение может быть использовано в различных областях науки и техники, где требуется решение таких систем уравнений, и может быть дальше развиваться и улучшаться.

# Литература

1. Метод квадратного корня для решения линейных систем [Электронный ресурс]. – Режим доступа: https://studbooks.net/2257477/matematika\_himiya\_fizika/metod\_kvadratnogo\_kornya\_resheniya\_lineynyh\_sistem. – Дата доступа: 09.02.2023.
2. Еще про методы решения систем линейных алгебраических уравнений [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/418627/. – Дата доступа: 09.02.2023.
3. Метод квадратного корня [Электронный ресурс]. – Режим доступа: https://scask.ru/i\_book\_calc2.php?id=8. – Дата доступа: 09.02.2023.
4. Лекция 3: Численное решение систем линейных алгебраических уравнений [Электронный ресурс]. – Режим доступа: https://intuit.ru/studies/professional\_skill\_improvements/1374/courses/168/lecture/4592?page=4. – Дата доступа: 09.02.2023.
5. Разложение Холецкого (метод квадратного корня) [Электронный ресурс]. – Режим доступа: https://algowiki-project.org/ru/Разложение\_Холецкого\_(метод\_квадратного\_корня). – Дата доступа: 09.02.2023.
6. Java [Электронный ресурс]. – Режим доступа: https://www.java.com/ru/. – Дата доступа: 02.09.2023.
7. Java Swing Tutorial [Электронный ресурс]. – Режим доступа: https://www.javatpoint.com/java-swing. – Дата доступа: 02.09.2023.
8. Show save file dialog using JFileChooser [Электронный ресурс]. – Режим доступа: https://www.codejava.net/java-se/swing/show-save-file-dialog-using-jfilechooser. – Дата доступа: 02.09.2023.