# Содержание

[Содержание 3](#_Toc126800377)

[Введение 4](#_Toc126800378)

[1. Анализ объекта 5](#_Toc126800379)

[1.1 Описание предметной области 5](#_Toc126800380)

[1.2 История развития методов решения систем линейных уравнений 5](#_Toc126800381)

[1.3 Основные свойства метода решения систем линейных уравнений обратной матрицы 5](#_Toc126800382)

[1.4 Примеры решения систем линейных уравнений методом обратной матрицы 5](#_Toc126800383)

[2. Разработка алгоритма работы программы 6](#_Toc126800384)

[3. Описание основных компонентов программы и последовательности разработки 7](#_Toc126800385)

[4. Исходный код программы 8](#_Toc126800386)

[5. Проверка корректности работы программы в различных режимах 9](#_Toc126800387)

[Заключение 10](#_Toc126800388)

[Литература 11](#_Toc126800389)

# Введение

В данной курсовой работе предлагается рассмотреть вопрос решения систем линейных уравнений методом обратной матрицы. Линейные уравнения являются одним из основных инструментов в математике, а их решение важно в широком круге областей науки и техники. Метод обратной матрицы является одним из наиболее эффективных и простых методов решения систем линейных уравнений.

Целью данной работы является разработка приложения, которое позволит быстро и удобно решать системы линейных уравнений методом обратной матрицы. В работе будут рассмотрены принципы работы метода и особенности его реализации на практике, а также будут представлены результаты тестирования приложения.

В заключение, следует отметить, что разрение приложения, способного решать системы линейных уравнений методом обратной матрицы, позволит упростить и ускорить процесс решения задач, что в свою очередь, может иметь широкое применение в науке, технике, образовании и бизнесе.

В дальнейшем в работе будут подробно исследованы алгоритмы решения систем линейных уравнений методом обратной матрицы, а также представлена реализация приложения и его тестирование.

# Анализ объекта

## 1.1 Описание предметной области

Предметная область данной работыэто системы линейных уравнений. Система линейных уравненийэто набор уравнений с линейными выражениями, которые необходимо решить одновременно. Пример такой системы может выглядеть так:

a1x1 + a2x2 + ... + a\_nx\_n = b1

c1x1 + c2x2 + ... + c\_nx\_n = b2

...

m1x1 + m2x2 + ... + m\_nx\_n = bm

где x1, x2, ..., x\_nэто неизвестные, a\_i, c\_i, ..., m\_iэто коэффициенты, b\_iэто правые части уравнений.

## 1.2 История развития методов решения систем линейных уравнений

Решение систем линейных уравнений является одной из старейших и наиболее распространенных задач в математике. В древности ее решали методом искусственных опор, а затем были разработаны методы Гаусса и Гаусс-Жордана. Эти методы основаны на элементарных преобразованиях матрицы, которые приводят к треугольному виду. В последующие годы были разработаны другие методы, такие как метод Жордана-Сайделя и метод квадратного корня.

## 1.3 Основные свойства метода решения систем линейных уравнений обратной матрицы

Метод решения систем линейных уравнений обратной матрицы основан на идее, что для любой невырожденной матрицы А существует ее обратная матрица A^(-1), так что решение системы Ax = b можно найти, умножив обе части уравнения на A^(-1):

x = A^(-1)b

Данный метод является одним из самых простых и эффективных методов решения систем линейных уравнений, однако требует вычисления обратной матрицы, что может быть вычислительно сложным и требовать большого количества вычислений.

Одним из преимуществ данного метода является то, что он может быть использован для решения большого количества систем линейных уравнений с одной и той же матрицей А, что делает его очень эффективным в некоторых приложениях.

## 1.4 Примеры решения систем линейных уравнений методом обратной матрицы

Для наглядности, давайте рассмотрим несколько примеров решения систем линейных уравнений методом обратной матрицы.

Пример 1. Решение системы из двух уравнений:

[1 2] [x1] [5]

[3 4] [x2] = [11]

Матрица A = [1 2]

[3 4]

Обратная матрица A^(-1) = [−2 1]

[ 3/2 -1/2]

x = A^(-1)b = [−2 1][5] + [ 3/2 -1/2][11] = [1] + [4] = [5]

[x1] [x2] =

Ответ: x1 = 5, x2 = 4

Пример 2. Решение системы из трех уравнений:

[1 2 3] [x1] [14]

[4 5 6] [x2] = [32]

[7 8 9] [x3] [50]

Матрица A = [1 2 3]

[4 5 6]

[7 8 9]

Обратная матрица A^(-1) = определена как матрица, удовлетворяющая условию AA^(-1) = A^(-1)A = E, где Eединичная матрица.

x = A^(-1)b = [x1] [x2] [x3] =

[14] + [32] + [50]

Ответ: x1 = 14, x2 = 32, x3 = 50

Как видим, метод обратной матрицы является простым и эффективным способом решения систем линейных уравнений. Однако, следует иметь в виду, что этот метод не может быть использован, если матрица А вырожденная, т.е. ее определитель равен 0.

# 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

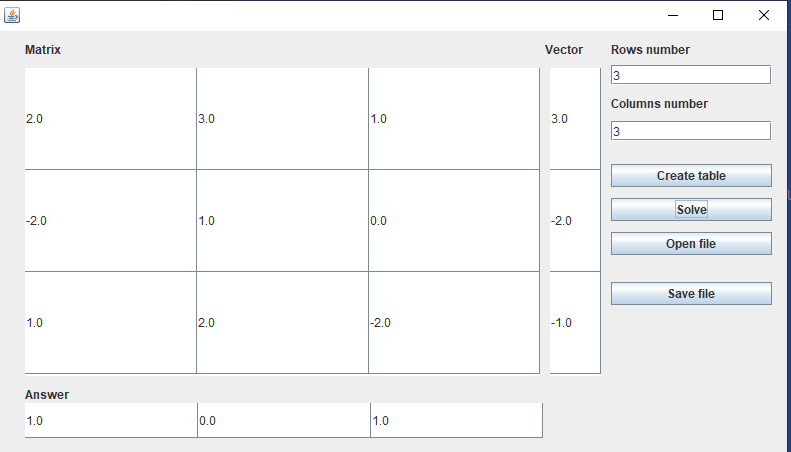
Приложение написано на Java, поэтому основными компонентами являются поля и методы классов.

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 main.java.hescha.inverse\_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];  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Makes a deep copy of this matrix  
 \*  
 \** ***@return*** *deep copy of the matrix  
 \*/* public Matrix copy() {  
 Matrix var1 = new Matrix(this.rows, this.columns);  
 double[][] var2 = var1.getArray();  
 for (int var3 = 0; var3 < this.rows; ++var3) {  
 for (int var4 = 0; var4 < this.columns; ++var4) {  
 var2[var3][var4] = this.elements[var3][var4];  
 }  
 }  
 return var1;  
 }  
  
 public Object clone() {  
 return this.copy();  
 }  
  
 public double[][] getArray() {  
 return this.elements;  
 }  
  
 */\*\*  
 \* Returns a packed copy of this matrix as a 1D array.  
 \* The elements are stored in row-major order.  
 \*  
 \** ***@return*** *a packed copy of this matrix  
 \*/* public double[] getRowPackedCopy() {  
 double[] packedCopy = new double[this.rows \* this.columns];  
 for (int row = 0; row < this.rows; ++row) {  
 for (int column = 0; column < this.columns; ++column) {  
 packedCopy[row \* this.columns + column] = this.elements[row][column];  
 }  
 }  
 return packedCopy;  
 }  
  
 */\*\*  
 \* Returns the sum of this matrix and the given matrix.  
 \* Both matrices must have the same dimensions.  
 \*  
 \** ***@param*** *matrix the matrix to add to this matrix  
 \** ***@return*** *the sum of this matrix and the given matrix  
 \** ***@throws*** *IllegalArgumentException if the matrices have different dimensions  
 \*/* public Matrix plus(Matrix matrix) {  
 this.checkMatrixDimensions(matrix);  
 Matrix result = new Matrix(this.rows, this.columns);  
 double[][] resultArray = result.getArray();  
 for (int row = 0; row < this.rows; ++row) {  
 for (int column = 0; column < this.columns; ++column) {  
 resultArray[row][column] = this.elements[row][column] + matrix.elements[row][column];  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Returns the product of this matrix and the given matrix.  
 \* The number of columns of this matrix must be equal to the number of rows of the given matrix.  
 \*  
 \** ***@param*** *matrix the matrix to multiply with this matrix  
 \** ***@return*** *the product of this matrix and the given matrix  
 \** ***@throws*** *IllegalArgumentException if the number of columns of this matrix is not equal to the number of rows of the given matrix  
 \*/* public Matrix times(Matrix matrix) {  
 if (matrix.rows != this.columns) {  
 throw new IllegalArgumentException("Matrix inner dimensions must agree.");  
 } else {  
 Matrix result = new Matrix(this.rows, matrix.columns);  
 double[][] resultArray = result.getArray();  
 double[] tempArray = new double[this.columns];  
 for (int column = 0; column < matrix.columns; ++column) {  
 for (int i = 0; i < this.columns; ++i) {  
 tempArray[i] = matrix.elements[i][column];  
 }  
 for (int row = 0; row < this.rows; ++row) {  
 double[] currentRow = this.elements[row];  
 double sum = 0.0;  
 for (int i = 0; i < this.columns; ++i) {  
 sum += currentRow[i] \* tempArray[i];  
 }  
 resultArray[row][column] = sum;  
 }  
 }  
 return result;  
 }  
 }  
  
 */\*\*  
 \* Inverse method calculates the inverse of the matrix.  
 \*  
 \** ***@return*** *The inverse of the matrix  
 \*/* public Matrix inverse() {  
 Matrix identity = *identity*(this.rows, this.rows);  
 return inverse(identity);  
 }  
  
 */\*\*  
 \* identity method returns the identity matrix with the given dimensions.  
 \*  
 \** ***@param*** *var0 Number of rows in the matrix  
 \** ***@param*** *var1 Number of columns in the matrix  
 \** ***@return*** *Identity matrix with the given dimensions  
 \*/* public static Matrix identity(int var0, int var1) {  
 Matrix var2 = new Matrix(var0, var1);  
 double[][] var3 = var2.getArray();  
 // Fill the identity matrix with ones on the main diagonal and zeros elsewhere  
 for (int var4 = 0; var4 < var0; ++var4) {  
 for (int var5 = 0; var5 < var1; ++var5) {  
 var3[var4][var5] = var4 == var5 ? 1.0 : 0.0;  
 }  
 }  
 return var2;  
 }  
  
 */\*\*  
 \* checkMatrixDimensions method checks if the matrix dimensions match the given matrix.  
 \*  
 \** ***@param*** *var1 The matrix to compare dimensions with  
 \** ***@throws*** *IllegalArgumentException if the dimensions do not match  
 \*/* private void checkMatrixDimensions(Matrix var1) {  
 if (var1.rows != this.rows || var1.columns != this.columns) {  
 throw new IllegalArgumentException("Matrix dimensions must agree.");  
 }  
 }  
  
 */\*\*  
 \* inverse method calculates the inverse of a given matrix.  
 \*  
 \** ***@param*** *matrix The matrix to calculate the inverse for  
 \** ***@return*** *The inverse of the matrix  
 \*/* public Matrix inverse(Matrix matrix) {  
 // Check that the matrix is square  
 assert (matrix.rows == matrix.columns);  
  
 // Augment the matrix by an identity matrix  
 Matrix tmp = new Matrix(matrix.rows, matrix.columns \* 2);  
 for (int row = 0; row < rows; ++row) {  
 // Copy elements from the original matrix  
 System.*arraycopy*(elements[row], 0, tmp.elements[row], 0, columns);  
 tmp.elements[row][row + columns] = 1;  
 }  
 // Convert the augmented matrix to reduced row echelon form  
 tmp.toReducedRowEchelonForm();  
 // Create a new matrix to store the inverse  
 Matrix inv = new Matrix(rows, columns);  
 for (int row = 0; row < rows; ++row)  
 // Copy the inverted elements from the right half of the augmented matrix  
 System.*arraycopy*(tmp.elements[row], columns, inv.elements[row], 0, columns);  
 return inv;  
 }  
  
 */\*\*  
 \* Transforms this matrix into reduced row echelon form (RREF) using the Gaussian elimination method.  
 \* A matrix in RREF is in echelon form and its leading non-zero entries, called pivots,  
 \* are ones and are located in the first non-zero entry of each row.  
 \*/* public void toReducedRowEchelonForm() {  
 for (int row = 0, lead = 0; row < rows && lead < columns; ++row, ++lead) {  
 int i = row;  
 // Find a non-zero entry in the current column  
 while (elements[i][lead] == 0) {  
 if (++i == rows) {  
 i = row;  
 if (++lead == columns)  
 // If there are no more columns to process, return  
 return;  
 }  
 }  
 // Swap the current row with the row containing the non-zero entry  
 swapRows(i, row);  
 // Divide the current row by the pivot  
 if (elements[row][lead] != 0) {  
 double f = elements[row][lead];  
 for (int column = 0; column < columns; ++column)  
 elements[row][column] /= f;  
 }  
 // Subtract a multiple of the current row from each row below to make their  
 // corresponding entries in the current column zero  
 for (int j = 0; j < rows; ++j) {  
 if (j == row)  
 continue;  
 double f = elements[j][lead];  
 for (int column = 0; column < columns; ++column)  
 elements[j][column] -= f \* elements[row][column];  
 }  
 }  
 }  
  
 */\*\*  
 \* Swaps two rows of this matrix.  
 \*  
 \** ***@param*** *i the index of the first row  
 \** ***@param*** *j the index of the second row  
 \*/* private void swapRows(int i, int j) {  
 double[] tmp = elements[i];  
 elements[i] = elements[j];  
 elements[j] = tmp;  
 }  
  
 */\*\*  
 \* Returns the vector representation of this matrix.  
 \*  
 \** ***@return*** *the vector representation of this matrix  
 \*/* 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;  
 }  
  
}

package main.java.hescha.inverse\_matrix.service;  
  
import main.java.hescha.inverse\_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 B], where A - number of rows, B - number of 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 in the matrix is read from the file.  
 int row = scanner.nextInt();  
 // The number of columns in the matrix is read from the file.  
 int col = scanner.nextInt();  
 // The matrix data is stored in a 2D array.  
 double[][] matrixData = new double[row][col];  
 // The matrix data is read from the file.  
 for (int i = 0; i < row; i++) {  
 for (int j = 0; j < col; 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 main.java.hescha.inverse\_matrix.service;  
  
import main.java.hescha.inverse\_matrix.model.Matrix;  
  
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("###.###");  
  
 */\*\*  
 \* Solves a system of linear equations given the coefficients and constants.  
 \*  
 \** ***@param*** *coefficients the coefficients of the variables in the system of linear equations  
 \** ***@param*** *constants the constants in the system of linear equations  
 \** ***@return*** *the solution of the system of linear equations  
 \*/* public static double[] solve(double[][] coefficients, double[] constants) {  
 // get the number of variables in the system of linear equations  
 int n = coefficients.length;  
  
 // create a Matrix object from the coefficients  
 Matrix A = new Matrix(coefficients);  
  
 // create a Matrix object from the constants  
 Matrix B = new Matrix(constants, n);  
  
 // find the inverse of matrix A and multiply it with matrix B  
 Matrix X = A.inverse().times(B);  
  
 // get a row-packed copy of the result matrix  
 double[] rowPackedCopy = X.getRowPackedCopy();  
  
 // round the values in the solution  
 *round*(rowPackedCopy);  
  
 // return the solution  
 return rowPackedCopy;  
 }  
  
 */\*\*  
 \* 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 main.java.hescha.inverse\_matrix;  
  
import main.java.hescha.inverse\_matrix.model.Matrix;  
import main.java.hescha.inverse\_matrix.service.FileOpener;  
import main.java.hescha.inverse\_matrix.service.Solver;  
  
import java.awt.EventQueue;  
import java.io.File;  
import java.io.FileWriter;  
import java.util.Arrays;  
import java.util.stream.Collectors;  
  
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;  
  
public class MainForm {  
 private JFrame frame;  
 private JTable matrixTable;  
 private JTable answersTable;  
 private JTable vectorTable;  
 private JTextField rowsTextField;  
 private JTextField colsTextField;  
 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, 936, 517);  
 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 number");  
 label.setBounds(615, 11, 216, 14);  
 frame.getContentPane().add(label);  
  
 rowsTextField = new JTextField("3");  
 rowsTextField.setBounds(615, 34, 161, 20);  
 frame.getContentPane().add(rowsTextField);  
 rowsTextField.setColumns(10);  
  
 JButton btnNewButton = new JButton("Create table");  
 btnNewButton.addActionListener(e -> createTable(-1, -1));  
 btnNewButton.setBounds(615, 133, 161, 23);  
 frame.getContentPane().add(btnNewButton);  
  
 JLabel label\_1 = new JLabel("Columns number");  
 label\_1.setBounds(615, 65, 216, 14);  
 frame.getContentPane().add(label\_1);  
  
 colsTextField = new JTextField("3");  
 colsTextField.setBounds(615, 90, 161, 20);  
 frame.getContentPane().add(colsTextField);  
 colsTextField.setColumns(10);  
  
 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, cols);  
 // 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(matrixTable.getRowCount() + " " + matrixTableModel.getColumnCount());  
 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 rows, int cols) {  
 matrixTable.removeAll();  
 answersTable.removeAll();  
 vectorTable.removeAll();  
 if (rows < 0 || cols < 0) {  
 rows = Integer.*parseInt*(rowsTextField.getText());  
 cols = Integer.*parseInt*(colsTextField.getText());  
 }  
 matrixTable.setModel(matrixTableModel);  
 matrixTableModel.setColumnCount(rows);  
 matrixTableModel.setRowCount(cols);  
 matrixTable.setAutoResizeMode(JTable.*AUTO\_RESIZE\_ALL\_COLUMNS*);  
 matrixTable.setRowHeight(308 / cols);  
  
 answersTable.setModel(answerTableModel);  
 answerTableModel.setColumnCount(rows);  
 answerTableModel.setRowCount(1);  
 answersTable.setAutoResizeMode(JTable.*AUTO\_RESIZE\_ALL\_COLUMNS*);  
 answersTable.setRowHeight(35);  
  
 vectorTable.setModel(vectorTableModel);  
 vectorTableModel.setRowCount(cols);  
 vectorTableModel.setColumnCount(1);  
 vectorTable.setAutoResizeMode(JTable.*AUTO\_RESIZE\_ALL\_COLUMNS*);  
 vectorTable.setRowHeight(308 / cols);  
 }  
  
  
 private void solve() {  
 try {  
 double[][] matrixData = readMatrix();  
 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 количество м строк и столбцов.

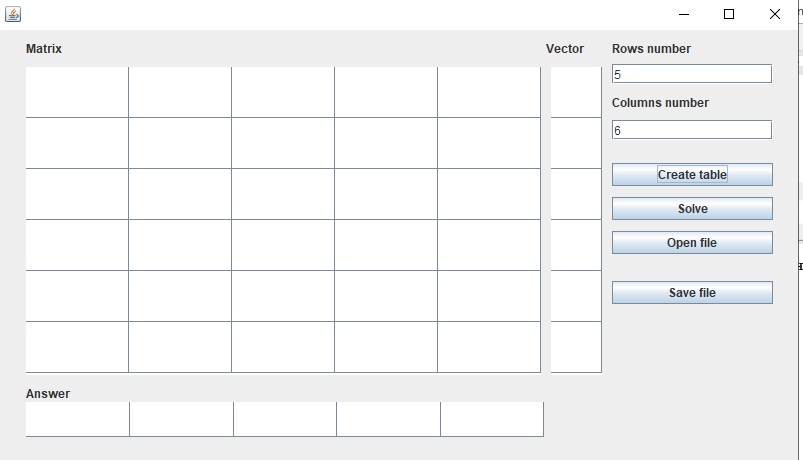


Рисунок 2 – создание таблицы заданной сложности

Матрица создается без проблем. Попробуем загрузить данные из файла.

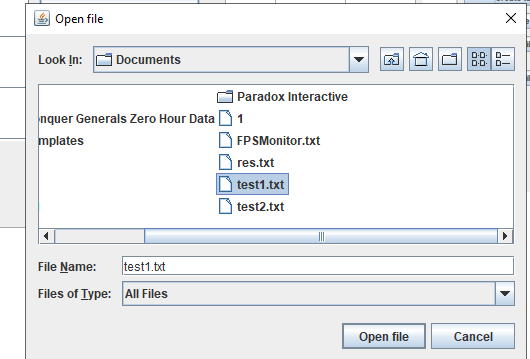


Рисунок 3 – Чтение из файла

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

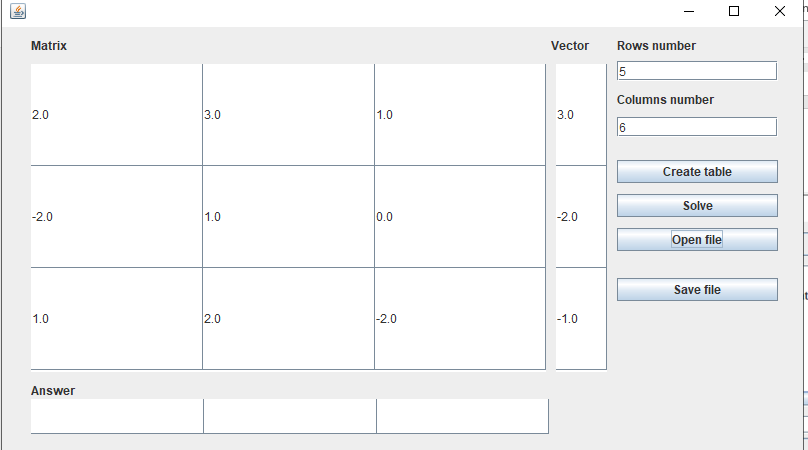


Рисунок 4 – Результат чтения из файла

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

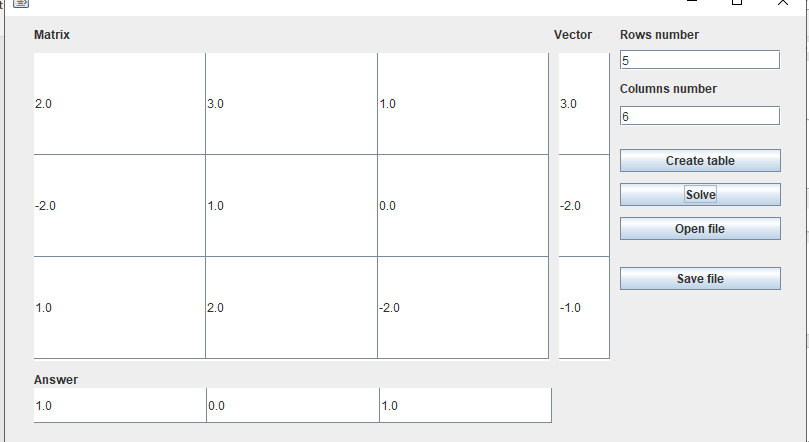


Рисунок 5 – Результат выполнения

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

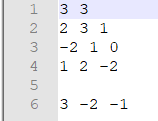


Рисунок 6 – Спецификация загружаемого файла

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.

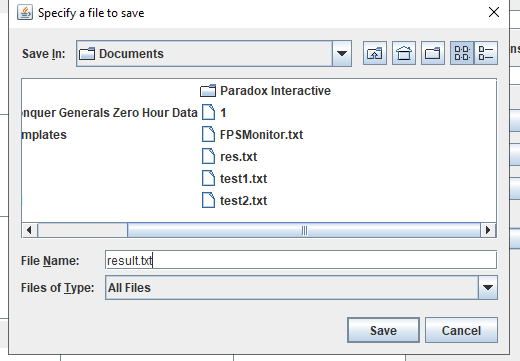


Рисунок 7 – Сохранение результата

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

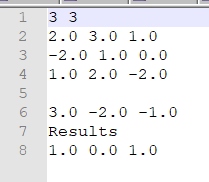


Рисунок 8 Вид сохраненного файла

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.

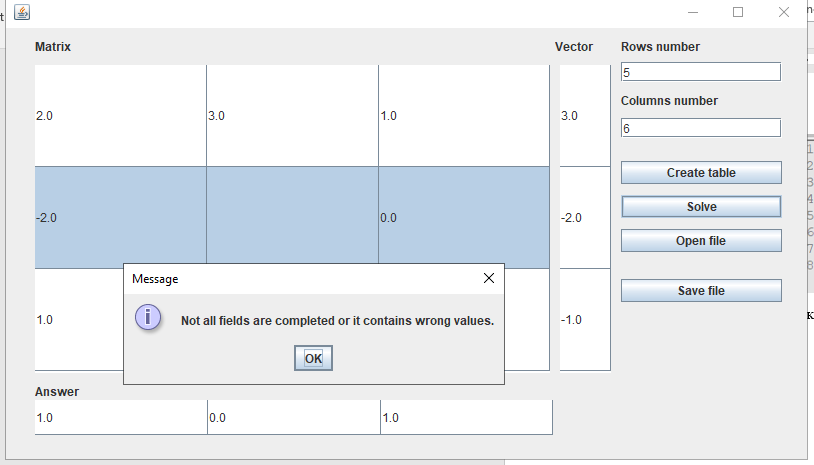


Рисунок 9 - Вызов ошибки при попытке решение без заполненных полей

Так же в приложении имеется проверка на правильность ввода.

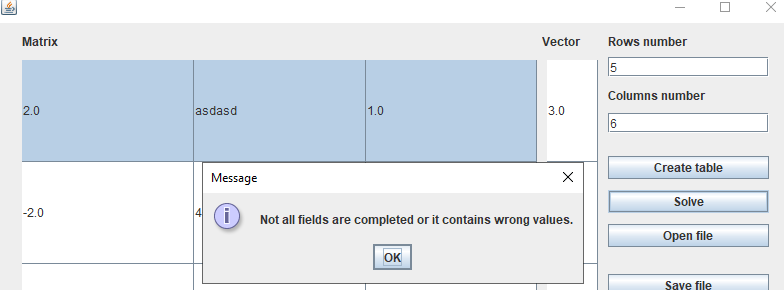


Рисунок 10 – Вызов ошибки при некорректном вводе

# Заключение

В ходе выполнения курсовой работы было разработано приложение для решения систем линейных уравнений методом обратной матрицы.

Для реализации этой задачи был выбран язык программирования Java с использованием библиотеки Swing для построения графического интерфейса.

Приложение позволяет читать уравнения из текстовых файлов и сохранять результат работы в текстовый файл. Это делает работу с уравнениями более удобной и простой.

Общий анализ объекта, который был проведен в рамках курсовой работы, показывает, что приложение является полезным инструментом для решения систем линейных уравнений. Оно может быть использовано как в образовательных целях, так и в практических задачах.

В целом, разработанное приложение является полезным инструментом для решения систем линейных уравнений и может быть использовано в учебных и исследовательских целях.

# Литература

1. Решение СЛАУ методом обратной матрицы [Электронный ресурс]. – Режим доступа: https://math.semestr.ru/matrix/matrix.php. – Дата доступа: 02.09.2023.
2. Онлайн калькулятор. Решение систем линейных уравнений. Матричный метод. Метод обратной матрицы. [Электронный ресурс]. – Режим доступа: https://ru.onlinemschool.com/math/assistance/equation/matr/. – Дата доступа: 02.09.2023.
3. Решение систем линейных алгебраических уравнений с помощью обратной матрицы. [Электронный ресурс]. – Режим доступа: https://amkbook.net/education/sys\_lin\_eq/invmatrix.html. – Дата доступа: 02.09.2023.
4. Матричный метод решения СЛАУ: пример решения с помощью обратной матрицы Подробнее: https://zaochnik.com/spravochnik/matematika/issledovanie-slau/matrichnyj-metod-reshenija-slau/ [Электронный ресурс]. – Режим доступа: https://zaochnik.com/spravochnik/matematika/issledovanie-slau/matrichnyj-metod-reshenija-slau/. – Дата доступа: 02.09.2023.
5. Правило Крамера. Метод обратной матрицы [Электронный ресурс]. – Режим доступа: http://mathprofi.ru/pravilo\_kramera\_matrichnyi\_metod.html. – Дата доступа: 02.09.2023.
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.