Department of Computer Science

COMPUTER LINEAR ALGEBRA SYSTEM

Natalia Kosaretskaya
BSc (Hons) in Computer Software Theory

2005

COMPUTER LINEAR AGLEBRA SYSTEM

submitted by Natalia Kosaretskaya

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specially acknowledged, it is the work of the author.

Abstract

Linear algebra is branch of mathematics widely used for solving various problems in mathematics, computer science, physics, engineering, economics etc. It is concerned with matrices, systems of linear equations, linear transformations, vectors, vector spaces and is closely related to geometry and analysis. A large number of compute linear algebra systems have been developed over the years to ease solving algebraic problems and thus enable their users to perform larger computations. These systems provide a vast range of operations and cover most subjects of linear algebra.

The aim of this project is to design a basic computer algebra system as a simplified analogy of systems like REDUCE, Maple, Mathematica etc. The system developed as the result of this project will mostly perform basic operations over matrices and solve linear equations using various algorithms, as the amount of time given is not enough for a profound research and implementation of a more sophisticated system. However, bounding the research to a very narrow subject allows concentrating on specific methods of solving problems of linear algebra and thus providing a better functionality in this particular area of mathematics.

The project requires some background work with literature on linear algebra and investigation of available systems. A deep research of matrix manipulation methods, ways of solving linear equations and their implementation algorithms is needed is needed to enable the system choose the appropriate methods of dealing with a particular problem according to the size of the input and complexity of the problem. The main functions the system would provide are matrix addition, multiplication, finding determinants and inverse matrices; solving systems of linear equations using Gauss-Jordan elimination, Cramer's method etc.

Analysis of the way the existing systems work can also help to increase functionality of the system and design convenient user interface. As the system is aimed to perform a limited range of operations it is possible to create a convenient interface reflecting main functions of the system. Another aim is to create an interface that would allow the user to a choose an algorithm for a specific operation.

Acknowledgements

I would like to thank my family friends who supported me though writing this project.

Special thanks to Dr. Nicolai Vorobjov for his guidance and my friends Gabriella, Alex and Hithion for keeping my morale up.

Overview

1. Introduction

This section contains the result of the research made as preparatory work for the development of this project. It includes an overview of some existing computer linear algebra systems and mathematical foundation of the algorithms used for the implementation of this instance of a computer linear algebra system.

2. Requirements analysis

This section describes basic requirements for the development of a computer linear algebra systems. It also describes the possible structure of the system to be developed. It lists various alternatives and justifies choices the developer has to make while designing a computer linear algebra system.

3. Requirements specification

This section briefly describes the choices made for designing this particular instance of a computer algebra system. It lists basic operations that will be implemented as the result of the project.

4. System design

This section contains formal description of the operations implemented in the system and algorithms used for implementation.

5. Testing

This section names the operations the system is capable of performing an shows results of test computations carried out after the system was fully implemented.

6 Appendixes.

Appendices contain the user manual for the system, full code listings and results of white-box testing.

7 Critical evaluation and further improvements

This section sums up the development process and list ideas that were not implemented due to time constraints but could be used for future improvement of the system.

Contents.

1. Introduction

1.1 Computer Algebra and Existing Computer Algebra Systems

If one follows the development of programming languages from a computer algebra point of view, one comes to realize that many issues modern programming languages try to address, have been encountered in the area of computer algebra first , and that those problems were – more or less successively resolved for that area. [9]

A large number of computer algebra systems have been developed over the years for solving algebraic problems in different areas of science. Most of these systems aim to help the user avoid mistakes and ease large computations that can be too complicated or are too time-consuming if done in pen and paper.

In general Computer algebra systems are aimed to:

- provide a set of basic pre-programmed commands, which will hand over to the machine the wearisome calculations which occur in a process run completely by the user.
- To offer a programming language, which lets the user to define higher-level commands or procedures to enlarge the original set of commands.[19]

Computer algebra systems can be classified as general purpose systems, special purpose systems and packages. However, the distinction among these types of computer algebra systems is not so evident.

Special-purpose computer algebra systems were developed first. However, their use was limited to some certain areas of application. Next generations of computer algebra systems tended to be general-purpose.

Most of the existing systems are general-purpose i.e. they can be used on very large range of applications. These systems usually reveal common characteristics:

- not restricted to any particular application
- better user interface
- wide range of built-in functions
- run-time systems that can be used interactively
- available on a range of hardware platforms

- usually provide a high-level programming language that allows users to implement their own algorithms.[11]

The built-in functions usually include functions for simplifying algebraic expressions, graphing facilities, typesetting for mathematical expressions, packages of methods of different areas of mathematics

Though existing general-purpose computer algebra systems are powerful and flexible computation tools, they are not universal means of solving mathematical problems. Some of the disadvantages of general-purpose computer systems are:

- Some computer systems may require the user to learn their specific syntax.
- The system that has to cover different areas of mathematics must be able to perform a large variety of operations. Thus the number of algorithms the system can use to perform a specific computation is limited.
- The user has very little (if any) control of what methods the system is using to perform certain computations

The aim of this project is to develop a simplified computer algebra system capable of solving linear equations and carrying out matrix-related operations. Thus it is essential to analyze some of the existing systems and their ways of carrying out linear algebra computations.

Maple

Maple represents a command-line language for symbolic, numeric and graphical computations that provides the users with a list of pre-defined functions and allows defining their own procedures. It contains the number of packages which may be loaded to work in different fields of mathematics.

The linalg package contains functions for manipulating vectors and matrices including matrix inversion, finding eigenvalues, finding rank of a matrix, solving linear systems etc. Some versions of Maple allow performing step-by-stem Gaussian or Gauss-Jordan elimination.

REDUCE

REDUCE is a general purpose computer algebra system designed to perform large scale computations over symbolic and numeric expressions. REDUCE offers a number of powerful operators which often give an immediate answer to a given problem. It allows combination of these operators thus enabling the user to create complicated evaluation and computation sequences. Rule formulation with subsequent pattern matching is also possible in REDUCE so it is a powerful tool for simplifying algebraic expressions.

Mathematica

Mathematica performs a wide range of mathematical computations using a programming language that is based on term re-writing and supports functional and procedural programming. This language contains a set of commands for solving mathematical problems. Internal functions can be called by external programs and data can be output to these programmers. It is peculiar for Mathematica to use the currently stored transformation rule as long as possible.

Matrix representation and manipulation

Most computer algebra systems use two-dimensional arrays to represent matrices (sometimes internally represented by a vector of vectors). Users can define matrix size and assign and access matrix elements in the same way as most languages operate arrays. Elements of matrices can be numeric as well as algebraic expressions or identifiers. After the matrix has been defined it can be passed as argument to procedures and functions and returned as a procedure value. Some systems can evaluate expressions built up from or containing matrices.

REDUCE performs algebraic operations over matrices in the same way it operates scalars. Initially matrices are filled with zeros. Matrices can be manipulated as object as well as element-by-element. However, internally they are operated element-by-element, so all elements must be defined before the matrix is used. REDUCE does

not recognize neither matrices of special types nor vectors (these are treated as n-by-1 or 1-by-n matrices).

Maple on the other hand allows the user to declare whether the matrix is of a special type (e.g. sparse, symmetric, triangular etc) and makes use of the properties of special type matrices. There are conventional ways of representing special matrices: symmetric matrices only need declaration of one triangle, sparse matrices can be represented as a vector of lists of pairs, containing indexes of the element etc.

Some general operations performed over large numeric matrices and especially matrices containing algebraic expressions as their elements can be tedious, complicated and time-consuming. A number of techniques have been introduced over the years to handle such operations and decrease their complexity (e.g. Strassen's algorithm for matrix multiplication). Block algorithms, for example, are widely used to improve system's performance (e.g. for calculating determinants, matrix multiplication, solving linear equations etc.). These algorithms are based on partitioning matrices into square or rectangular sub-blocks and applying functions to these blocks rather than individual matrix elements.

Matrix inversion

The classical method of finding the inverse is based on calculating matrix determinant and adjugate matrix. This method involves excessive number of computations as every element of the inverse comprises the determinant of the original matrix. However it is suitable computing inverses of sparse matrices.

Dense matrices can be effectively inverted by applying Gaussian elimination to an augmented matrix, formed of the original matrix and identity matrix.

Linear Equations

Computer algebra systems solve systems of linear equations taking their matrix representation as input.

The emphasis in computer algebra is primarily on the exact solution of systems over such domains as

- the integers (Diophantine systems)
- the field of rational numbers
- a finite field
- the rational numbers extended by some algebraic or transcendental elements. [11]

Some algorithmic developments widely used for solving linear systems are mostly based on elimination methods, determinant-based methods, parallel algorithms, black-box methods, canonical forms, Dophinante solutions etc. The methods that would be implemented in this project are described below.

Solution can be found by performing Gaussian elimination. This method is the most commonly used for dense matrices, but it is quite inefficient for sparse matrices, since the inverses of sparse matrices can be dense.

Matrix inversion is another way of solving linear equations. It is comparatively ineffective because evaluation of the inverse involves a large number of operations. Nevertheless, some systems (e.g. REDUCE) use this method as they are not equipped with row reduction tools.

Cramer's rule is also effective way of solving systems of linear equations, provided the matrix of coefficients is sparse. It can also be used with small matrices, but with great caution as it involves multiple calculations of determinant and that increases computational complexity immensely.

Alternative way of solving sparse matrix is by LU factorization used solely or as a part of a more complicated algorithm (Mathematica's way).

Therefore the choice of what heuristics to use for solving a given system of equation should be made according to the size and type of the matrix.

User interface

In current computer algebra systems, there exist three different types of interfaces for linking them to numerical packages. The first type allows the user to generate expressions in another programming language. The second type connects object modules, originally created by a compiler of an arbitrary programming language, to computer algebra systems. With the third type, numerical and symbolic software are seamlessly integrated into a common environment.[9]

Most computer algebra systems provide graphical representation of their output expressions. This output can either be used as a text document or inserted into a text document by copy-paste techniques.

1.2 Mathematics

After analyzing existing systems it is necessary to give an overview of linear algebra concepts that will be implemented within the system.

Matrices

A **matrix** $A=[a_{ij}]_{m\times n}$ of size m by n (or an *m* by *n* matrix) is a rectangular array of m rows and n columns and consisting of m*n numbers.

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{2m} & \dots & a_{mn} \end{bmatrix}$$

The **element** a_{ij} (or the ij^{th} **entry**) appears in row *i* and column *j*.

Special cases:

If every element of a matrix is zero, it is called a **zero matrix**.

A $n\times 1$ matrix is called a **column vector** of dimension n.

A $1\times n$ matrix is called a **row vector** of dimension n.

If *m* = *n* the matrix A is called **square matrix**.

A square matrix A whose elements a_{ij} = 0 for $i > j$ is called **upper triangular.**
A square matrix A whose elements a_{ij} = 0 for $i < j$ is called **lower triangular**.

A square matrix $A=[a_{ij}]_{m\times n}$ is said to be **diagonal** if d_{ij} = 0 whenever $i\neq j$ (i.e. when all the entries off the main diagonal are 0)

The matrix with 1's on the main diagonal and 0 elsewhere is called the **identity** matrix.

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Two matrices $A=[a_{ij}]$ and $B=[b_{ij}]$ are said to be **equal** if they gave the same size (i.e. the same number of rows and columns) and their corresponding entries are equal (i.e. $a_{ij}= b_{ij}\ \forall i, j \in Z$)

Transpose matrix

Given an $m \times n$ matrix $A=[a_{ij}]$, the **transpose** of A (denoted A^T) is an $n \times m$ matrix such as $A^T=[a_{ji}]$ (i.e. columns of A^T are formed of corresponding rows of A).

Scalar multiplication

Given a matrix $A=[a_{ij}]$and a number λ (scalar) we define the product of A by a to be the matrix, denoted by $\lambda A=[\lambda a_{ij}]$ that is obtained of multiplying every element of A by λ.

Matrix addition (subtraction)

Let $A=[a_{ij}]$and $B=[b_{ij}]$be two $m \times n$ matrices. Then their sum (difference) is defined by an $m \times n$ matrix $C=[c_{ij}]$where c_{ij} is obtained by adding (subtracting) corresponding elements of A and B. For matrices of different order addition (subtraction) is undefined

Matrix multiplication

Let $A=[a_{ij}]\ m \times n$ and $B=[b_{ij}]\ n \times p$ be two matrices. Then their product is defined by an $m \times p$ matrix $C=[c_{ij}]$where c_{ij} is obtained by multiplying the i^{th} row of A by j^{th} column of B:

$$C = [c_{ij}] = \left[\sum_{k=1}^{n} a_{ik} b_{kj} \right] \forall i \in [1, m], j \in [1, p]$$

In general $AB \neq BA$. If $AB = BA$ then A and B are said to **commute**.

Properties of matrix multiplication

Let $A=[a_{ij}]\, m \times n$, $B=[c_{ij}]\, n \times p$ and $C=[c_{ij}]\, p \times k$ be matrices and λ be a scalar. Then

1. $(A \times B) \times C = A \times (B \times C)$
2. $A \times (B + C) = A \times B + A \times C$
3. $(A + B) \times C = A \times C + B \times C$
4. $\lambda \times (A \times B) = (\lambda \times A) \times B = A \times (\lambda \times B)$
5. $I_n \times A = A \times I_n = A$

Strassen Multiplication

Let $A=[a_{ij}]\, 2^n \times 2^n$ and $B=[b_{ij}]\, 2^n \times 2^n$ be two matrices. Then their product is defined by an $2^n \times 2^n$ matrix $C=[c_{ij}]\, 2^n \times 2^n$. Using the following matrix representation:

$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & A_{22} \end{pmatrix}$ the product is given by $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Components of C can be computed by:

$$p_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$p_2 = (A_{21} + A_{22})B_{11}$$

$$p_3 = A_{11}(B_{12} - B_{22})$$

$$p_4 = A_{22}(B_{21} - B_{11})$$

$$p_5 = (A_{11} + A_{12})B_{22}$$

$$p_6 = (A_{21} - A_{11})(B_{11} + B_{21})$$

$$p_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = p_1 + p_4 - p_5 + p_7$$

$$C_{12} = p_3 + p_5$$

$$C_{21} = p_2 + p_4$$

$$C_{22} = p_1 + p_3 - p_2 + p_6$$

If $A=[a_{ij}]\ m\times m$ and $B=[b_{ij}]\ m\times m$ with m odd, then their product is defined by an $m\times m$ matrix $C=[c_{ij}]\ m\times m$. Using the following matrix representation:

$A = \begin{pmatrix} A_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} B_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$ the product is given by $C = \begin{pmatrix} C_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$

Where A_{11}, B_{11} and C_{11} are square $m-1\times m-1$ matrices; a_{12}, b_{12} and c_{12} are $m-1\times 1$ column vectors; a_{21}, b_{21} and c_{21} are $1\times m-1$ row vectors; a_{22}, b_{22} and c_{22} are 1×1 matrices. Then matrix $C=[c_{ij}]$ is computed as:

$$\begin{pmatrix} C_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + a_{21}b_{21} & A_{11}b_{12} + a_{21}b_{22} \\ a_{21}B_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Powers of matrices

Given a square matrix $A=[a_{ij}]$, powers of A are given by:

$$A^0 = I_n$$

$$A^n = \underbrace{AA...A}_{n}$$

Matrix determinant

Let $A=[a_{ij}]$ be a 2×2 matrix.

$$A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Then the **determinant** of A is denoted by $|A|$ or $\det A$ is a scalar

$$|A| = a_{11}a_{22}-a_{12}a_{21}.$$

For n ≥ 2 the determinant of $n\times n$ matrix $A=[a_{ij}]$ is:

$$\det A=\sum_{j=1}^{n}(-1)^{i+j}a_{ij}\det A_{ij} \text{ for } \forall i\in[1,n]$$

Minors, cofactors and ajugate matrices

Given a matrix $A=[a_{ij}]\ n\times n$ the ij^{th} **minor** of A (denoted M_{ij}) is the determinant of $(n-1)\times(n-1)$ matrix obtained by eliminating the i^{th} row and the j^{th} column in A.

Given a matrix $A=[a_{ij}]\ n\times n$ the ij^{th} **cofactor** of A (denoted A_{ij}) is given by

$$C_{ij}=(-1)^{i+j}M_{ij}$$

Given a matrix $A=[a_{ij}]\ n\times n$ the **adjugate** matrix of A (denoted adj$A$) is the transpose matrix of cofactors of A:

$$adjA=\left[A_{ij}\right]^{T}$$

Inverse matrix

Let A be a $m \times n$ matrix. Then $m \times n$ matrix X is said to be a **left inverse** of A if it satisfies the equation $X \times A = I_n$. Y is said to be a **right inverse** of A if it satisfies the equation $A \times Y = I_n$. If a matrix A has both left inverse X and right inverse Y then A is square, $X = Y$ and A is called **invertible** (or **non-singular**). The inverse of A is denoted by A^{-1}:

$$A^{-1} = \frac{1}{\det A} adjA$$

If $A=[a_{ij}]\, n \times n$ is a diagonal matrix and $[a_{ij}] = [k_i],\ \forall i = j$, then B A^{-1} is a diagonal matrix $[a_{ij}] = \left[1 \diagup k_i \right],\ \forall i = j$

Elementary Row (Column) Operations

Given a matrix $A=[a_{ij}]$with rows $R_1 \ldots R_n$ and columns $C_1 \ldots C_n$ elementary row (column) operations are:

1. Interchange of i^{th} and j^{th} row (column) denoted by $R_i \leftrightarrow R_j$ $(C_i \leftrightarrow C_j)$
2. Multiplication of all elements of i^{th} row (column) by a non-zero scalar λ denoted by λR_i (λC_i)
3. Addition to all elements of the i^{th} row (column) the corresponding elements of the j^{th} row (column) multiplied by a scalar λ, denoted by $R_i + \lambda R_j$ $(C_i + \lambda C_i)$

Elementary matrices

Let e be an elementary row operation. Then the **elementary matrix** E is obtained by applying e to the identity matrix:

$$E = e(I_n)$$

Any elementary row operation over a matrix $A=[a_{ij}]$ is equivalent to pre-multiplication of A by the corresponding elementary matrix

Row equivalence

If a matrix B is obtained by applying any number of elementary row (column) operations it is said to be **row equivalent** to *A*.

A matrix a matrix $A=[a_{ij}]\ n \times n$ is row-equivalent to I_n if there exists a finite number of elementary matrices $E_1, E_2 \ldots E_m$ such that

$$E_1E_2 \ldots E_mA = I_n$$

If A is row-equivalent to I_n then A is invertible and A^{-1} is obtained by applying the same elementary row operations to I_n:

$$A^{-1} = E_m \ldots E_2E_1I_n$$

Alternative method of computing the inverse

Given invertible matrix $A=[a_{ij}]\ n \times n$ its inverse A^{-1} can be computed by forming an **augmented matrix**

$$[A \mid I_n]$$

of size a matrix $n \times 2n$ and applying elementary row operations so that the augmented matrix takes the form

$$[I_n \mid A^{-1}]$$

Echelon Forms

The matrix A is in **row echelon form** if:

1. All rows consisting entirely of zeros are in the bottom of the matrix.
2. Each leading entry of a row is in a column to the right of the leading entry of the each row above it.
3. All entries if a column below a leading entry are zeros.

The matrix A is in **reduced row echelon form** if it is in row echelon form and:

1. The leading entry in each nonzero row is 1
2. Each leading 1 is the only nonzero entry in its column

Each matrix is equivalent to only one reduced echelon matrix.

Linear Equations

A **linear equation** in *n* unknowns is an equation of the form

$$a_1x_1 + a_2x_2 + ... + a_nx_n = b$$

where coefficients $a_1 \ldots a_n$ and *b* are real or complex numbers and subscript *n* may be any positive integer.

A system of linear equations is a collection of two or more linear equations with the same unknowns.

A system of linear equations can be represented in a matrix form: $AX = B$ where $A=[a_{ij}]\, m \times n$ is a matrix of coefficients, $X=[x_j]$ is a vector of unknowns and $B=[b_i]$:

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{2m} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix}$$

A **solution** of a system is a vector $S=[s_i]$ of numbers such that each equation becomes a true statement when the values of $X=[x_i]$ are substituted by corresponding elements of S.

A **solution set** of the system of linear equations is the set of all possible solutions.

If the matrix of coefficients $A=[a_{ij}]$ is invertible the system can be solved by pre-multiplying by A^{-1}:

$$A^{-1}AX = A^{-1}B$$

$$I_n X = A^{-1}B$$

$$X = A^{-1}B$$

Cramer's Rule

Given a system of equations represented in matrix form $AX = B$, where determinant of A is not zero the solution can be found as follows:

$$x_i = \frac{|A_i|}{|A|} \quad \forall x_i \in X$$

where A_i is obtained by replacing the i^{th} column by B.

Gauss elimination

Gauss elimination is one of the quite powerful methods of solving linear equations. In consists of several steps. Given a system of equations represented in matrix form $AX = B$

1. Form the augmented matrix

$$[A \mid B]$$

2. Using only elementary row (column) operations on the augmented matrix change it so that the coefficient matrix takes the row echelon form
3. If the system has solution (or solutions) solve the last non-trivial equation
4. Substitute the obtained value into the next equation

Gauss-Jordan elimination

Gauss-Jordan elimination extends Gauss elimination. Given a system of linear equations $AX = B$:

1. Form the augmented matrix

$$[A \mid B]$$

2. By elementary row operations the augmented matrix is converted into reduced row echelon form

2. Requirements Analysis

Main goals

The main goal of this project is to design a basic computer algebra system capable of performing operations over matrices and solve systems of simultaneous linear equations. The program will execute basic matrix operations and calculations using some existing methods (addition, multiplication, exponentiation, inversion, transposition, elementary row and column operations, finding determinant, minors, cofactors and adjugate matrix) and apply those methods to the systems of linear equations to obtain solutions. The choice of the algorithm to be applied to the input will be made either by the system itself, depending on the size of the input, or specified by the user. Eigenvalue and eigenvector calculation may also be implemented if possible within given time constraints.

The program will be written in Java programming language in object oriented fashion.

The system will be developed according to the spiral systems development model and the number of functions implemented will be determined by the time constraints.

This section describes system requirements, names options encountered on different stages of system development and justifies choices made.

Functional Requirements

System requirements

- Memory requirements – the system must be able to store data entered by user and manipulate that data throughout the run of the program. The system may be able to store the data during the run-time so that the data can be called the next time the user uses the system. However, the usability of interactive storage of the data depends on the way the user interface design is implemented and on user preferences.

- Speed requirements – the system should be a run-time program and provide answers to the given problems in sufficiently small time. In order to do so it should be able to choose appropriate algorithms depending on the size of the given problem.

Fundamental Requirements

The basic matrix operations the system should be able to perform:

- Matrix addition
- Matrix subtraction
- Scalar multiplication
- Matrix multiplication, using various algorithms
- Matrix transposition
- Matrix exponentiation
- Calculate matrix determinant
- Calculate matrix minor
- Calculate matrix cofactors
- Calculate a matrix of cofactors
- Matrix inversion, using various algorithms
- Perform elemetary row operations
- Perform elementary column operations
- Create augmented matrix
- Solve a system of linear equations, using various algorithms

Auxiliary Requirements

In order to perform the operations listed above the system should be able to retrieve information about the input data of the data used internally. Therefore it should be able to perform the following actions:

- Take a matrix as input
- Compute the number of rows of the matrix

- Compute the number of columns of the matrix
- Check if the number of rows of the matrix is even
- Check if the number of columns of the matrix is even
- Retrieve an element of the matrix
- Retrieve a row of the matrix
- Retrieve a column of the matrix
- Perform computations over a retrieved row (column)
- Interchange rows (columns) of the matrix
- Augment two matrices
- Augment two matrices vertically
- Output the result of performed calculations
- Be robust
- Be able to tell if the specified action can be performed over a given inp

Data input requirements

The data should be input in the form of a matrix . However the size of the input is limited regarding the system's computational abilities. The maximum input data size must be fixed to avoid excessive excessive computations. The maximum size will be set to 15 rows or/and columns in a matrix.

Another issue connected with data input is the way the user provides data to the system. One of the possible way is to pass the system a text-file containing all the data. This may be useful when the input is large. However, the interactive way of inputting data is more convenient for performing a series of different manipulations over the input. Therefore the system will receive the input data interactively.

User Interface Requirements

The major user interface requirement is that the interface must be convenient for performing all the stated functionality. There are two obvious alternative ways of designing the user interface for the system. One is implementing a command-line

environment, another – creating a windows-based system with buttons and menu bars corresponding to specific functions.

The system should be able to respond to user actions. It should check the validity of input an inform the user of any disparity

It should be robust and inform the user of any run-time errors stating their reasons and prompting possible solutions.

The system should also have an embedded help directory containing guidelines and trouble shooting instructions.

Non-functional requirements

System Constraints

- The system will be programmed in Java programming language
- The system will be compatible with Windows
- The deadline for the system to be handed in is May 16 but the implementation should be finished earlier in order to be able to produce appropriate documentation. Thus the number of options to be implemented is limited by the time constraints
- The system is implemented by one person. This also limits the possible functionality of the system

Testing Requirements

- The appropriate testing plan should be created
- The system is not a safety-critical application, so it is only the main functionality that needs to be properly tested
- All units should be thoroughly tested to make sure all algorithms work properly. However, exhaustive testing is not always possible
- The system will be tested using JUnit testing facility

Maintenance Requirements

The system should be portable

The system will be presented as a Jva jar file and will not need any installation procedures

Documentation Requirements

The system should be well-documented and have a detailed user manual.

The system is aimed for users familiar with computer linear algebra systems or with concepts of linear algebra so the users will not need any training

3. Requirements specification

Main goal

The system created as the result of this project will be a simplified computer linear algebra system performing basic matrix operations and solving systems of linear equations.

Functional requirements

System requirements

The system will be compatible with one operational system.

The system will take the user input and manipulate it in order to produce appropriate computational result.

The system will chose a computational algorithm according to the size of the input in order to perform the computations efficiently.

Fundamental Requirements

The system should be able to perform the following functions:

- Matrix addition
- Matrix subtraction
- Scalar multiplication
- Matrix multiplication
- Matrix multiplication using Strassen algorithm
- Matrix transposition
- Matrix exponentiation
- Calculate matrix determinant
- Calculate matrix minor
- Calculate matrix cofactors
- Calculate a matrix of cofactors
- Matrix inversion
- Perform elemetary row operations
- Perform elementary column operations

- Create augmented matrix using the identity matrix
- Create augmented matrix using column vector
- Solve a system of linear equations using the inverse
- Solve a system of linear equations using the Cramer's Rule
- Solve a system of linear equations using Gauss elimination
- Solve a system of linear equations using Gauss-Jordan Elimination

Auxiliary Requirements

In order to comply with the fundamental requirements the system should:

- Take a matrix as input
- Compute the number of rows of the matrix
- Compute the number of columns of the matrix
- Check if the number of rows of the matrix is even
- Check if the number of columns of the matrix is even
- Retrieve an element of the matrix
- Retrieve a row of the matrix
- Retrieve a column of the matrix
- Perform computations over a retrieved row (column)

Data input requirements

The system must be able to read in a matrix in the form of a two–dimensional array.

The system should be able to decline user input if the input size exceeds the set limit of 15 rows and/of columns .

User interface requirements

The system will have a window-based graphical user interface with drop-down menus and buttons for the choice of operations and algorithms.

The user interface will have a help text field giving the user directions of how to use the system.

4. System design

Data representation

The system is implemented in Java programming language in the object-oriented fashion. The main data type is the Matrix object.

Internally the Matrix object uses a two-dimensional array of doubles to represent a matrix. Elements of the matrix are accessed as entries in the array and all operations are performed over the internal array.

Constructors

A few constructors construct matrix objects of different dimensions, shape and contents to be used by the system interface or for internal use of some methods.

Constructors take matrix dimensions (and in some cases some data to be entered into the matrix) to form a two-dimensional array that would be accessed by matrix methods.

Matrix(int r)

Constructs a square zero matrix

Parameters:

r - number of rows and columns of the matrix

This constructor takes the number of rows as input. Assuming that the number of columns is equal to the number of rows it fills the matrix with zeros.

```
FOR all rows
        FOR all columns
                each element is set to zero
```

Matrix(int r, int c)

Constructs a rectangular zero matrix

Parameters:

r - rows of the matrix

c - columns of the matrix

Another zero-matrix constructor. Taking the number of rows and columns as input it creates a rectangular matrix and fills it with zeros.

FOR all rows
    FOR all columns
        each element is set to zero

Matrix(int r, double element)

Constructs a square matrix filled with one element

Parameters:

r - rows and columns of the matrix

element - element of the matrix

The constructor used for internally by some methods. Taking the number of rows of the matrix and the element value as input it assumes the number of rows is equal to the number of columns and creates a square matrix setting every element to the input element value.

FOR all rows
    FOR all columns
        each element is set to the input value

Matrix(int r, int c, double element)

Constructs a rectangular matrix filled with the same element

Parameters:

r - rows of the matrix

c - columns of the matrix

element - elements of the matrix

This constructor creates a rectangular matrix filled with the same value. Taking the number of rows, the number of columns and the element value as input it creates a rectangular matrix and sets each element to the element value

FOR all rows
    FOR all columns
        each element is set to the input value

Matrix(int r, double[] d)

Constructs a square matrix of size out of the array of doubles

Parameters:

r - rows and columns of the matrix

d - elements of the matrix

The constructor takes the number of rows and an array of doubles and, assuming the number of columns is equal to the number of rows, sets each element of the matrix to each subsequent value in the input array.

FOR all rows
    FOR all columns
        every element is set to every subsequent element of the array

Matrix(int r, int c, double[] d)

Constructs a rectangular matrix out of the array of doubles

Parameters:

r - rows of the matrix

c - columns of the matrix

d - elements of the matrix

The constructor takes the number of rows, the number of columns and an array of doubles as input and creates a rectangular matrix setting each element of the matrix to the value of each subsequent element of the array

FOR all rows

FOR all columns

every element is set to every subsequent element of the array

Matrix(int r, int c, double[][] da)

Constructs a rectangular matrix out of the array of doubles

Parameters:

r - rows of the matrix

c - columns of the matrix

da - elements of the matrix

The constructor creates a rectangular (or a square matrix if the input values of the number of rows and the number of columns are equal) matrix taking the number of rows, the number of columns and a two-dimensional array of doubles as input and setting each element of the matrix to the corresponding element of the input array.

FOR all rows

FOR all columns

each element is set to the value of the corresponding element of the input array

IdentityMatrix(int r)

Constructs an Identity matrix object

Parameters:

r - rows, columns

The Identity matrix extends the matrix object and inherits all methods connected with it. The constructor creates a matrix that reveals properties of the inverse matrix. It is a square matrix with ones on the main diagonal and zeros elsewhere.

FOR all rows

FOR all columns

IF the row number equals to the column number

element is set to one

ELSE

Element is set to zero

Methods for auxilary operations

Some matrix operations require implementation of auxiliary functions for performing necessary checks, retrieving matrix characteristics etc.

int **getRows**()

Returns the number of rows of the matrix

Returns: Integer.

The method returns the number of rows of the corresponding matrix object. Mostly used for performing checks within methods that operate matrices of particular types or dimensions.

int **getColumns**()

Returns the number of columns of the matrix

Returns: Integer

The method returns the number of columns of the corresponding matrix object. Mostly used for performing checks within methods that operate matrices of particular types or dimensions.

double **getElement**(int i, int j)

Returns the ij-th element of the matrix

Parameters:

i - row

j - column

Returns: Double

The method retrieves the element of the matrix at the specified position taking input of the number of row and the number of column of the element.

double[][] **getElements**()

Returns elements of the matrix arranged into a 2D array

Returns: two-dimensional array of doubles

Returns the elements of the matrix in the form of the internal array.

boolean **rowsEven**()

Checks if the matrix has even number of rows

Returns: true if the matrix has even number of rows

Gets the number of rows of the matrix and checks whether this value is even. The method is used internally inside matrix manipulation methods that can only be performed over the matrices with even number of rows.

boolean **isSquare**()

Checks if the matrix is square

Returns: true if the matrix is square

Compares the number of rows with the number of columns of the matrix. The method is used internally inside methods that can only be performed over square matrices.

boolean **isInvertible**()

Returns true if the matrix is invertible

Returns: true if the matrix is invertible

Determinant-based check used by determinant-based methods that require the matrix to be invertible.

IF the determinant of the matrix is not equal to zero
RETURN true

matrix **matrixAugment**()

Creates augmented matrix using identity matrix

Returns: matrix

The method takes a square matrix and creates a new matrix with the same number of rows and twice the same number of columns. The extra columns are filled with the elements of the newly created identity matrix of the same dimensions as the input matrix

IF the matrix is square
create new matrix(rows, columns*2)
FOR each row of the new matrix
FOR each column of the new matrix
IF the column number is smaller than the number of columns in the original matrix
set the element to the value of the corresponding value of the original matrix
ELSE
set the element to the value of the corresponding value of the identity matrix

matrix **matrixAugmentVector**(double[] vect)

Creates augmented matrix with the vector taking an array of doubles as an input

Parameters:

vect - vecor

Returns: matrix

The method takes a matrix and an array of doubles and creates a new matrix with the same number of rows and the number and an extra column added. The extra column is filled with subsequent elements of the vector.

IF the number of rows of the matrix is equal to the number of elements in the vector

create new matrix(rows, columns*2)

FOR each row of the new matrix

FOR each column of the new matrix

IF the column number is smaller than the number of columns in the original matrix

set the element to the value of the corresponding value of the original matrix

ELSE

set the element to the value of each subsequent element of the vector

matrix **matrixAugmentVector**(Matrix vect)

Creates augmented matrix with the vector using matrix representation of a vector

Parameters:

vect - vector

Returns: matrix

The method takes a matrix and a matrix representation of a column vector and creates a new matrix with the same number of rows and the number and an extra column added. The extra column is filled with subsequent elements of the vector.

IF the number of rows of the matrix is equal to the number of elements in the vector

create new matrix(rows, columns*2)

FOR each row of the new matrix

FOR each column of the new matrix

IF the column number is smaller than the number of columns in the original matrix

set the element to the value of the corresponding value of the original matrix

ELSE

set the element to the value of each subsequent element of the vector

matrix **matrixAugment**(Matrix A)

Creates augmented matrix using matrix A

Parameters:

A - matrix

Returns: matrix

The method takes two matrices and creates a new matrix with the same number of rows and twice the same number of columns. The extra columns are filled with the corresponding elements of the input matrix

IF the matrix is square

create new matrix(rows, columns*2)

FOR each row of the new matrix

FOR each column of the new matrix

IF the column number is smaller than the number of columns in the original matrix

set the element to the value of the corresponding value of the original matrix

ELSE

set the element to the value of the corresponding value of the input matrix

matrix **matrixVertAugment**(Matrix A)

Performs vertical augmentation using matrix A

Parameters:

A - matrix

Returns: matrix

The method takes two matrices and creates a new matrix with twice the same number of rows and the same number of columns. The extra rows are filled with the

corresponding elements of the input matrix. The method is only used inside methods that are based on partitioning matrices.

```
IF the matrix is square
    create new matrix(rows, columns*2)
    FOR each row of the new matrix
        FOR each column of the new matrix
            IF the row number is smaller than the number of columns in
            the original matrix
                set the element to the value of the corresponding
                value of the original matrix
            ELSE
                set the element to the value of the corresponding
                value of the input matrix
```

matrix **deleteRowCol**(int r, int c)

Returns the matrix with the row r and column c deleted

Parameters:

r - row

c - column

Returns: matrix

This method takes the row and columns indexes as input and creates a matrix with the number of rows and columns one less that the dimensions of the original matrix. The method is used inside determinant-based methods.

```
new Matrix (rows-1, columns-1)
FOR all rows of the original matrix
    FOR all columns of the original matrix
        IF the element indexes are not equal to the input indexes
            set the corresponding element in the new matrix
            to the value of the element
```

String **matrixWrite**()

Returns String representation of the matrix

Returns: String

The String representation of a matrix is used to print out the matrix. Used by the user interface.

Methods for major matrix operations

Arithmetic operations

matrix **transpose**()

Returns a transposed matrix

Returns: matrix

This method transposes the matrix. It returns the matrix with the number of rows the same as the number of columns of the original matrix and the number of columns the same as the number of rows of the original matrix. The elements are swapped accordingly.

```
new Matrix (columns, rows)
FOR all rows of the original matrix
    FOR all columns of the original matrix
        set the element of the new matrix to the value of the corresponding
        element of the original matrix
```

matrix **scalarMult**(double scalar)

Returns this matrix multiplied by a scalar

Parameters:

scalar - scalar

Returns: matrix

The method performs scalar multiplication. Taking a scalar as input it returns a matrix with each element multiplied by the scalar.

```
FOR all rows of the original matrix
        FOR all columns of the original matrix
                multiply each element of the matrix by the scalar
```

matrix **matrixAdd**(MatrixA)

Retuns sum of this matrix and matrix A

Parameters:

A - matrix

Returns: matrix

Taking two matrices as input the method performs matrix addition. If the matrices are of the same dimensions it returns the matrix with each element set to the sum of the corresponding elements of the matrices.

```
IF matrices have the same number of rows and columns
        FOR all rows of the original matrix
                FOR all columns of the original matrix
                        add each element of the matrix to the corresponding element
                        of the other matrix
```

matrix **matrixSubtract**(Matrix A)

Returns difference of this matrix and matrix A

Parameters:

A - matrix

Returns: matrix

Taking two matrices as input the method performs matrix subtraction. If the matrices are of the same dimensions it returns the matrix with each element set to the difference of the corresponding elements of the matrices.

IF matrices have the same number of rows and columns

FOR all rows of the original matrix

FOR all columns of the original matrix

subtract each element of the matrix from the corresponding element of the other matrix

matrix **matrixMult**(Matrix A)

Returns the product of this matrix and matrix A

Parameters:

A - matrix

Returns: matrix

This method takes two matrices as input and performs matrix multiplication. If the matrices are of appropriate dimensions the method returns a matrix with the number of rows of the first matrix and the number of columns of the second matrix with the elements set to sums of products of elements of corresponding rows of the first matrix and corresponding columns of the second matrix.

new Matrix(rows of the first matrix, columns of the second matrix)

IF the number of columns of the first matrix equals to the number of rows of the second matrix

FOR all rows of the original matrix

FOR all columns of the second matrix

FOR all columns of the original matrix

set the element of the new matrix to the sum of the products of the elements of corresponding rows of the first matrix and corresponding columns of the second matrix

matrix **matrixStrassenMult**(Matrix B)

Performs Strassen multiplication. Returns product of this matrix and matrix A

Parameters:

B - matrix

Returns: matrix

This method performs matrix multiplication using Strassen's algorithm. The major condition is that both matrices multiplied should be square matrices of the same dimensions. The output matrix is square matrix of the same dimensions as the input matrices. The method is based on partitioning the input matrices and manipulating their parts rather than just elements or the whole matrices.

IF both matrices are square

IF matrices are square and the number of rows is even

new Matrix A11(rows/2, columns/2)
new Matrix A12(rows/2, columns/2)
new Matrix A21(rows/2, columns/2)
new Matrix A22(rows/2, columns/2)
new Matrix B11(rows/2, columns/2)
new Matrix B12(rows/2, columns/2)
new Matrix B21(rows/2, columns/2)
new Matrix B22(rows/2, columns/2)
new Matrix p1 = (A11 + A22) * (B11 + B22)
new Matrix p2 = (A21 + A22) * B11
new Matrix p3 = A11 * (B12 – B22)
new Matrix p4 = A22 * (B21 – B11)
new Matrix p5 = (A11 + A12) * B22
new Matrix p6 = (A21 – A11) * (B11 + B12)
new Matrix p7 = (A12 – A11) * (B21 + B22)
new Matrix C11 = p1 + p4 – p5 + p7
new Matrix C12 = p3 + p5
new Matrix C21 = p2 + p4
new Matrix C22 = p1 + p3 – p2 + p6
augment C11 and 12
augment C21 and C22
vertically augment augmented matrices

```
            output result
        IF the number of rows is odd
            new Matrix A11(rows-1, columns-1)
            new Matrix A12(1, columns-1)
            new Matrix A21(rows-1, 1)
            new Matrix A22(1, 1)
            new Matrix B11(rows-1, columns-1)
            new Matrix B12(1, columns-1)
            new Matrix B21(rows-1, 1)
            new Matrix B22(1, 1)
            new Matrix C11 = strassenMult(A11,B11) + A12 * B21
            new Matrix C12 = A11 * B21 + A21 + A12 * B22
            new Matrix C21 = A21 * B11 + A22 * B21
            new Matrix C22 = A21 * B12 + A22*B22
            augment C11 and 12
            augment C21 and C22
            vertically augment augmented matrices
            output result
```

matrix **matrixExp**()

Returns a squared matrix

Returns: matrix

This method squares the matrix. Taking the input matrix it multiplies it on itself.

matrix **matrixExp**(int degree)

Returns matrix to the power degree

Parameters:

degree - degree

Returns: matrix

This method takes an integer as input and performs matrix exponentiation to the power of the input value. It post-multiplies matrix on itself a number of times.

Elementary Row Operations

matrix **multRow**(int row, double scalar)

Multiplies a row with a scalar

Parameters:

row - row

scalar - scalar

Returns: matrix

This method takes the row index and a scalar as input and multiplies the specified row by the scalar

FOR every element of the specified row

set the element to the product of the element and the scalar

matrix **addRowMult**(int row1, int row2, double scalar)

Adds a multiple of one row to another row

Parameters:

row1 - frist row

row2 - second row

scalar - scalar

Returns: matrix

This method takes two row indices and a scalar as input and adds the scalar multiple of the second row to the first row

FOR every element of the first row

add to the element the product of the element of the second row and the scalar

matrix **swapRow**(int row1, int row2)

Swaps two rows

Parameters:

row1 - first row

row2 - second row

Returns: matrix

This method takes two row indices as input and swaps the specified rows

FOR the number of columns

temporary variable set to the value of the element of the first row

the element of the first row set to the value of the corresponding element of the second row

the element of the second row set to the value of the temporary variable

Elementary Column Operations

matrix **multCol**(int col, double scalar)

Multiplies a column with a scalar

Parameters:

col - columns

scalar - scalar

Returns: matrix

This method takes the column index and a scalar as input and multiplies the specified column by the scalar

FOR every element of the specified column

set the element to the product of the element and the scalar

matrix **addColMult**(int col1, int col2, double scalar)

Adds a multiple of one column to another column

Parameters:

col1 - first columns

col2 - second column

scalar - scalar

Returns: matrix

This method takes two column indices and a scalar as input and adds the scalar multiple of the second column to the first row

FOR every element of the first column

add to the element the product of the element of the second column and the scalar

matrix **swapCol**(int col1, int col2)

Swaps two columns

Parameters:

col1 - first column

col2 - second column

Returns: matrix

This method takes two column indices as input and swaps the specified column

FOR the number of rows

temporary variable set to the value of the element of the first column

the element of the first column set to the value of the corresponding element of the second column

the element of the second column set to the value of the temporary variable

Determinant

double **det**()

Computes determinant of the matrix

Returns: double

This method computes the determinant of the matrix. If the matrix is a two-by-two matrix the determinant equals the difference of products of elements on the matrix diagonals. If the matrix if bigger that two-by-two its determinant equals the sum of the appropriate multiple of products of the elements of the first row and the determinant of the matrix formed by eliminating the first row and the corresponding column from the original matrix.

IF the matrix is square
    IF the number of rows is two
        RETURN the difference of the products of the elements on the diagonal
    ELSE
        FOR every element in the first row
            delete the corresponding columns
            determinant equals the sum of the determinant and the element multiplied by the determinant of the matrix after columns elimination an a (-1) to the power of the element coordinates

Minor, Cofactor, Adjugate

double **minor**(int r, int c)

Taking two indices as input this method computes the ij-th minor of the matrix

Parameters:

r - row

c - column

Returns: double

This method takes two indices as input and returns the ij-th minor of the matrix.

IF the matrix is square

delete specified row and column

RETURN determinant

double **cofactor**(int r, int c)

Taking two indices as input this method computes the ij-th cofactor of the matrix

Parameters:

r - row

c - column

Returns: double

This method takes two indices as input and returns the ij-th cofactor of the matrix.

IF the matrix is square

RETURN the specified minor multiplied by (-1) to the power of the sum if indices

matrix **adjugate**()

Returns the adjugate of this matrix

Returns: matrix

This method computes the adjugate of the matrix.

IF the matrix is square

new Matrix(rows, columns)

FOR all rows

FOR all columns

compute cofactor

set the corresponding new matrix element to the value of the cofactor

Inverse

matrix **inverse**()

Calculates the inverse of the matrix

Returns: matrix

This method uses the classical way of computing the inverse of the matrix using determinant and adjugate matrix.

IF the matrix is square
    IF the matrix is invertible
        compute the adjugate
        RETURN the adjugate scalar multiplied by the determinant to the power (-1)

matrix **gaussElimInvert**()

Computes the matrix inverse using Gauss-Jordan Elimination

Returns: matrix

This method uses Gauss-Jordan elimination to compute the inverse of the matrix. First it creates an augmented matrix and applies Elementary Row Operations to create an upper-triangular matrix. Then it uses backward substitution to turn the upper-triangular matrix into identity.

IF the matrix is square
    create augmented matrix using the inverse
    FOR number of columns of input matrix
        FOR number of rows of input matrix
            find the maximum element in the column
            swap rows so that the biggest element in the column is in the first row
            subtract from each row the appropriate multiple of the first row
            divide each row by the value of the diagonal element of the row

subtract from each row appropriate multiples of other rows
form a new matrix out of the augmented part of the matrix
used for computation

Systems of equations

matrix **cramerSolve**(Matrix B)

Solves a system of equations using Cramer's rule

Parameters:

B - column vector

Returns: column vector

This method uses Cramer's rule to solve systems of linear equations. Taking a matrix representation of the column vector as input if produces the column vector of unknowns. Unknowns are computed by performing operations over matrixes, obtained by substituting appropriate column by the input vector.

IF the matrix is square and invertible
IF the number of rows is equal to the number of rows of the input vector
new Matrix (rows, 1)
FOR number of rows
element of the vector set to the product of the corresponding cofactor and input vector element divided by matrix determinant

matrix **inverseSolve**(Matrix B)

Solves a system of equations multiplying by the inverse

Parameters:

B - column vector

Returns: column vector

This method uses inverse of the original matrix to solve systems of linear equations. Taking a matrix representation of the column vector as input if produces the column vector of unknowns. Unknowns are computed by pre-multiplying the input vector by the inverse of the original matrix.

matrix **gaussElimSolve**(Matrix B)

Solves a system of equations using Gauss-Jordan elimination

Parameters:

B - column vector

Returns: column vector

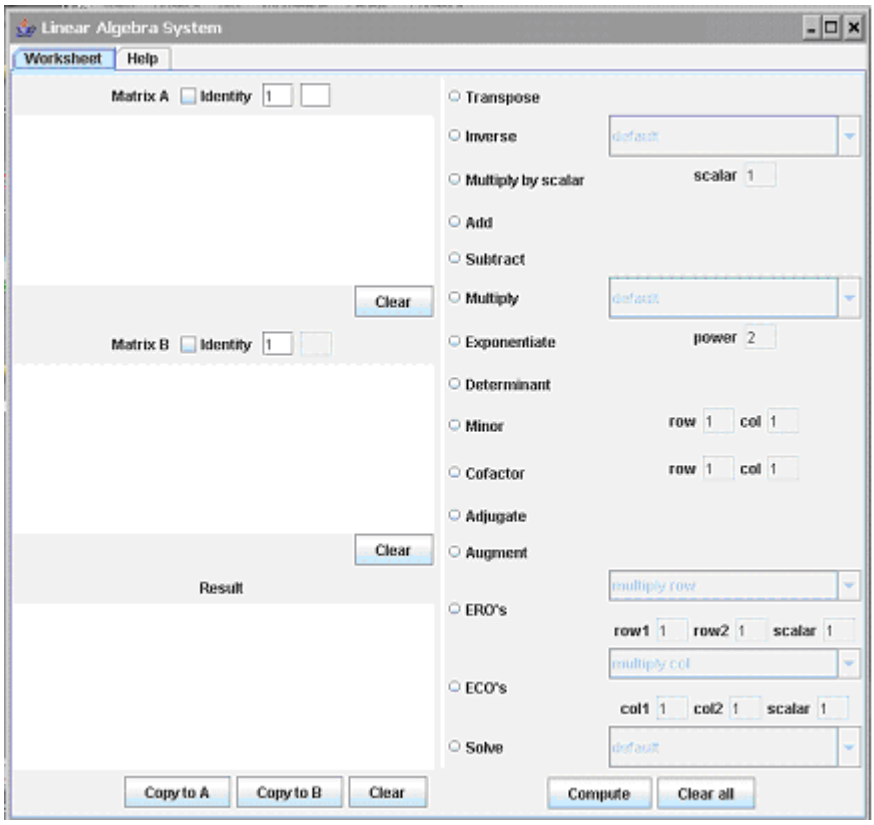
This method uses Gauss-Jordan elimination to solve systems of linear equations. Taking a matrix representation of the column vector as input if produces the column vector of unknowns. First it creates an augmented matrix and applies Elementary Row Operations to create an upper-triangular matrix. Then it uses backward substitution to turn the upper-triangular matrix into identity. The vector of answers is created out of the last column.

```
IF the matrix is square
        create augmented matrix with the input vector
                FOR number of columns of input matrix
                        FOR number of rows of input matrix
                                find the maximum element in the column
                                swap rows so that the biggest element in the column
                                is in the first row
                                subtract from each row the appropriate multiple of the
                                first row
                                divide each row by the value of the diagonal element
                                of the row
                                subtract from each row appropriate multiples of other
                                rows
                                form a new column vector out of the last column of the
                                matrix used for computation
```

User Interface Design

This instance of a computer linear algebra system is equipped with an application implementing one of the possible graphical user interface solutions for such a system.

User interface is represented by a JFrame with windows, drop-down menus and buttons.



The user enters values into appropriate fields and selects options. These parameters are then passed on to the actual methods.

If the operation is represented by several algorithms the user is given a choice of which to choose. If the user does not specify the algorithm, the interface application makes the choice according to the size of the input.

As the size of the input is limited to 15 rows and/or columns in a matrix, the choice of a default algorithm is made on the following basis:

Matrix Inversion

Classical determinant-based inversion algorithm is used if the matrix has less then 6 rows. Jauss-Jordan elimination is used otherwise.

Matrix multiplication

If the input matrix has less than 6 rows the multiplication is carried out using the classical algorithm. Strassen multiplication is used otherwise.

Linear systems

The system offers three ways of solving linear systems. If the input matrix has less than 5 rows the Cramer's rule is used. Systems with number of rows greater or equal to 5 but less than 8 the inverse pre-multiplication method is used. Otherwise the system is solved by Gauss-Jordan elimination.

5. Testing

Testing was carried out through the whole development process. This section lists the results of Black-box tests performed after the system was fully implemented. These include only test of the main matrix functions. (Results of White-box tests, including auxiliary operations tests, and the code of the testing unit are listed in the *Appendix*).

See table on the next page.

Operation under test	Condition under test	Input	System respond	Pass/ fail
Arithmetic operations				
Transposition	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix 2.0 1.0 2.0 1.0 0.0 3.0 4.0 2.0 1.0	P
Scalar multiplication	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 Scalar 2.0	matrix 4.0 2.0 8.0 2.0 0.0 4.0 4.0 6.0 2.0	P
Matrix addition	Dimension check, addition	matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix 2.0 0.0 3.0 3.0 -1.0 4.0	No respond	P
		matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix -1.0 1.0 0.0 -3.0 3.0 0.0 2.0 1.0	matrix 1.0 1.0 3.0 -4.0 -2.0 3.0 3.0 5.0	P

Matrix subtraction	Dimension check, subtraction	matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix 2.0 0.0 3.0 3.0 -1.0 4.0	No respond	P
		matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix -1.0 1.0 0.0 -3.0 3.0 0.0 2.0 1.0	matrix 3.0 -1.0 3.0 2.0 -8.0 3.0 -1.0 3.0	P
Matrix multiplication	Dimension check, multiplication	matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix 0.0 1.0	No respond	P
		matrix 2.0 0.0 3.0 -1.0 -5.0 3.0 1.0 4.0 matrix 0.0 1.0 -1.0 0.0	matrix 0.0 2.0 1.0 3.0 -3.0 -5.0 -4.0 1.0	P

Strassen multiplication	Dimension check, Strassen multiplication	matrix 0.0 -3.0 -6.0 -9.0 4.0 1.0 -2.0 -5.0 8.0 5.0 2.0 -1.0 12.0 9.0 6.0 3.0 matrix 0.0 2.0 4.0 1.0 3.0 5.0 2.0 4.0 6.0	No respond	P
		matrix 0.0 -3.0 -6.0 -9.0 4.0 1.0 -2.0 -5.0 8.0 5.0 2.0 -1.0 12.0 9.0 6.0 3.0 matrix 0.0 3.0 6.0 9.0 2.0 5.0 8.0 11.0 4.0 7.0 10.0 13.0 6.0 9.0 12.0 15.0	matrix -84.0 -138.0 -192.0 -246.0 -36.0 -42.0 -48.0 -54.0 12.0 54.0 96.0 138.0 60.0 150.0 240.0 330.0	P
		matrix 0.0 1.0 2.0 2.0 3.0 4.0 4.0 5.0 6.0 matrix 0.0 2.0 4.0 1.0 3.0 5.0 2.0 4.0 6.0	matrix 5.0 11.0 17.0 11.0 29.0 47.0 17.0 47.0 77.0	P
Matrix exponentiation	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix 13.0 14.0 14.0 6.0 7.0 6.0 9.0 5.0 15.0	P
Matrix exponentiation	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix 68.0 55.0 94.0 31.0 24.0 44.0 53.0 54.0 61.0	P

Augmented matrix (identity)	Dimension check	matrix 2.0 1.0 4.0 1.0 0.0 2.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix 2.0 1.0 4.0 1.0 0.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 3.0 1.0 0.0 0.0 1.0	P
Augmented matrix (vector)	Dimension check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0 -6.0	matrix 2.0 1.0 4.0 2.0 1.0 0.0 2.0 3.0 2.0 3.0 1.0 -6.0	P
Elementary Row operations				
Multiplying row by a scalar	Row index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row index 4 scalar 2.0	No respond	P

		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row index 3 scalar 2.0	matrix 2.0 1.0 4.0 1.0 0.0 2.0 4.0 6.0 2.0	P
Add a multiple of one row to another row	Row index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row 1 index 3 row 2 index 4 scalar 2.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row 1 index 1 row 2 index 3 scalar 2.0	matrix 6.0 7.0 6.0 1.0 0.0 2.0 2.0 3.0 1.0	P

Swap two rows	Row index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row 1 index 3 row 2 index 4	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 row 1 index 1 row 2 index 3	matrix 2.0 3.0 1.0 1.0 0.0 2.0 2.0 1.0 4.0	P
Elementary column operations				
Multiplying column by a scalar	Column index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column index 4 scalar 2.0	No respond	P

		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column index 3 scalar 2.0	matrix 2.0 1.0 8.0 1.0 0.0 4.0 2.0 3.0 2.0	P
Add a multiple of one column to another column	Column index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column 1 index 3 column 2 index 4 scalar 2.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column 1 index 1 column 2 index 3 scalar 2.0	matrix 10.0 1.0 4.0 5.0 0.0 2.0 4.0 3.0 1.0	P

Swap two rows	Column index check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column 1 index 3 column 2 index 4	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 column 1 index 1 column 2 index 3	matrix 4.0 1.0 2.0 2.0 0.0 1.0 1.0 3.0 2.0	P
Determinant				
Computing determinant	Shape check	matrix 2.0 1.0 4.0 1.0 0.0 2.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	determinant 3.0	P
Minors, cofactors, adjugate				

Computing a minor	Shape check, index check	2.0 5.0 -3.0 -7.0 row index 3 column index 3	No respond	P
		2.0 5.0 -3.0 -7.0 row index 1 column index 2	minor -3.0	P
Computing cofactor	Shape check, index check	2.0 5.0 -3.0 -7.0 row index 3 column index 3	No respond	P
		2.0 5.0 -3.0 -7.0 row index 2 column index 2	cofactor 2.0	P
Computing adjugate matrix	Shape check	matrix 2.0 1.0 1.0 0.0 2.0 3.0	No respond	P

		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix -6.0 11.0 2.0 3.0 -6.0 -0.0 3.0 -4.0 -1.0	P
Inverse				
Computing inverse using determinant	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix -2.0 3.6 0.6 1.0 -2.0 -0.0 1.0 -1.3 -0.3	P
Computing inverse using Gauss-Jordan elimination	N/A	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0	matrix -2.0 3.6 0.6 1.0 -2.0 -0.0 1.0 -1.3 -0.3	P
Systems of linear equations				
Solve a system of equations using Cramer's rule	Dimension check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0 -6.0	vector 3.0 -4.0 0.0	P

Solve a system of equations using inverse pre-multiplication	Dimension check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0 -6.0	vector 3.0 -4.0 0.0	P
Solve a system of equations using Gauss-Jordan elimination	Dimension check	matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0	No respond	P
		matrix 2.0 1.0 4.0 1.0 0.0 2.0 2.0 3.0 1.0 vector 2.0 3.0 -6.0	vector 3.0 -4.0 0.0	P

6. Critical evaluation and further improvements.

Critical evaluation.

The evaluation of this project took for major stages – research, implementation, testing and documentation write-up.

During the research stage major matrix operations and their implementation algorithms were explored. A few existing computer algebra systems were analyzed to understand the underlying principles of internal organization of a computer algebra system and possible implementation solutions of the main goals and objectives of this project.

The second stage took two parts. One for implementing main system functions, another for creating a graphical user interface.

During the first part of the implementation stage functions for carrying out the basic matrix operations were implemented. However not all planned functionality could be implemented within the given time constraints.

The graphical user interface was implemented after major operations and was designed to comply with the implemented functions in the best way possible.

The testing stage named separately was, in fact, carried out through the whole development of the system. Each object and method were tested thoroughly during implementation. The last tests were performed after the system was fully implemented. The results of these tests were described it the *Testing* section and the full code of the testing units and their output are provided in the *Appendix*.

The write-up stage was also carried out through the whole development process except for the *User manual* (provided in the *Appendix*) which was written after the system was fully implemented and tested.

System design improvements

Some improvements can be made over the data input and the output of the computations results. The system can be enabled to read in data from the text file and to output the results into a text file.

The run-time backup can also be implemented. The system currently does not back up neither the data the user enters nor the result of the computations.

Functional improvements

The created system is a simplified analog of existing computer algebra systems and only implements the basic matrix operations. Therefore it is possible to introduced a great variety of additional features.

The system may be enabled to recognize matrices of special types (triangular, tirdiagonal, Sylvester matrix, etc).

Different computational approaches can be made to sparse and dense matrices.

Implemented functions can be extended by introducing other algorithm to increase the system's computational power. (E.g. Bareiss algorithm for computing determinant, LU-factorization for matrix inversion etc)

The system can be extended by implementing other linear algebra concepts and functions (eigenvalues, eigenvectors, matrix factorization etc)

The system is currently performing operations over matrices with maximum size of 15 rows and/or columns and chooses the algorithm according to the size of the input, though the maximum size is small enough to allow performing any algorithm disregarding complexity. However, increasing the maximum size of input may require reconsidering the choice of the algorithm used by default.

As the system can perform one computation via a choice of algorithm it might be useful to add a counter, indicating computational time (number of operations made during the computation) and complexity of the problem.

Another functional improvement worth making is introduction of ways to manipulate algebraic expressions as well as numerical ones.

User interface improvements

The system has a window-based user interface with drop-down menus, checkboxes radio buttons etc. that correspond to the functions the system is capable of performing. However it might be useful to add a console and a parser, enabling the user to enter assignments and compound commands in the form of regular expressions and add new functions without changing the internal structure of the system.

Some graphics can also be introduced to make the system look more professional.

Bibliography

1. Ayres, F., 1974. *Schaum's outline of theory and problems of matrices*. SI (metric) ed. New York : McGraw-Hill.
2. Aho, Alfred V. (Alfred Vaino), 1983. *Data structures and algorithms* Addison-Wesley
3. Barnett S., 1990. *Matrices : methods and applications*. Oxford : Clarendon Press.
4. Blyth T. S., 1998. *Basic linear algebra*. Berlin : Springer
5. Chudnovsky D. V. 1989. *Computer algebra*. New York : Dekker.
6. Demmel J. W. 1997. *Applied numerical linear algebra*. Philadelphia, Pa. : Society for Industrial and Applied Mathematics.
7. Faddeev D.K, Faddeeva V.N. 1963. *Computational methods of linear algebra*. San Francisco, London: W.H. Freeman and company.
8. Gantmacher F.R. 1960. *The theory of Matrices*. New York: Chelsea Publishing Company New York.
9. Grabmeier, J. 2003**.** *Computer algebra handbook : foundations, applications, systems*. Berlin ; London : Springer.
10. Hamilton, A.G.,1989. *Linear algebra an introduction with concurrent examples. Cambridge* : Cambridge University Press.
11. Harper, D., 1991*. A guide to computer algebra system*. Chichester : Wiley.
12. Higham N.J., Higham D.J.2000 *Matlab guide.* Society for Industrial and applied mathematics.
13. Lay David C. 1997. *Linear algebra and its applications*. 2nd ed. Reading, Mass. ; Harlow : Addison-Wesley.
14. Lipschutz S. 1974. *Schaum's outline of theory and problems of linear algebra*. SI (metric ed.). New York : McGraw-Hill.
15. Mignotte, M. 1992. ***Mathematics for computer algebra***. New York ; London : Springer-Verlag.
16. Nering, E D. 1963. *Linear algebra and matrix theory*. New York ; London : Wiley.
17. Towers, D. A., 1988. ***Guide to linear algebra***. London : Macmillan.
18. Williams, G.2001. ***Linear algebra with applications***. 4th ed., Sudbury, Mass. ; London : Jones and Bartlett.

19. Davenport, Siret, Tournier. *Computer Algebra* [online]. Addison-Wesley. Available from:

http://staff.bath.ac.uk/masjhd/masternew.pdf

18. Hearn A.C. 1995. *Reduce user's manual* [online]. Avaliable from:

http://www.uni-koeln.de/REDUCE/3.6/doc/reduce/

19. Huss-Lederman S, Jacobson E.M., Tsao A, Turnbull T., Johnson J.R. *Implementation of Strassen's algorithm for matrix multiplication* [online] Available from:

http://portal.acm.org/citation.cfm?id=369096&coll=GUIDE&dl=GUIDE&CFID=43825267&CFTOKEN=45666301&ret=1#Fulltext

19. Mathematica. *What is Mathematica* [online]. Avaliable from:

http://www.wolfram.com/products/mathematica/introduction.html

20. Schreiner, W. *Distributed Maple* – User and Reference manual [online]. Avaliable from:

http://www.risc.uni-linz.ac.at/software/distmaple/report/

21. Wikipedia. *Computer algebra system* [online]. Avaliable from:

http://en.wikipedia.org/wiki/Computer_algebra_system

22. Wikipedia. *Linear algebra* [online]. Avaliable from:

http://en.wikipedia.org/wiki/Linear_algebra

8 Appendices

8.1 Appendix A. User Manual

Installation

To install this linear algebra system on your computer simply copy the CAsystem.jar file from the CD to the directory where you would like to keep your linear algebra system.

Using linear algebra system

To start working with the linear algebra system simply double-click oh the CAsystem.jar file.

To create a matrix enter the number of rows (and columns for a rectangular matrix) in the Matrix A 'rows' (and 'cols') fields and enter the matrix in the form of a two-dimensional array, separating elements in each row by pressing TAB and separating rows by pressing ENTER. For Example:

1 2 3
4 5 6
7 8 9

Choose an action by selection one of the radio buttons.

Some options (scalar, minor etc) may require entering some values or element coordinates

If the action is performed over using two matrices, the 'Matrix B' text field would be activated. Create the second matrix the same way you have created matrix A.

If the system can offer more than one way of performing a computation you can select the algorithm you wish the system to choose. Otherwise the system will choose the algorithm itself.

If you need to create an identity matrix simply enter the number of rows your matrix should have and tick the box 'Identity' next to corresponding text field.

After selecting an operation press COMPUTE

You can copy the result into 'Matrix A' or 'Matrix B' fields by pressing 'Copy to A' or 'Copy to B' respectively.

To clear a text field press 'Clear' at the bottom of the text field.

To clear all text field press 'Clear All'

8.2 White-box testing

```
      Creating an empty square matrix of size 4 by 4

0.0   0.0   0.0   0.0
0.0   0.0   0.0   0.0
0.0   0.0   0.0   0.0
0.0   0.0   0.0   0.0


       Creating an empty rectangular matrix of size 2 by 4

0.0   0.0   0.0   0.0
0.0   0.0   0.0   0.0


       Creating a matrix of size 3 by 3 out of a double 2.3

2.3   2.3   2.3
2.3   2.3   2.3
2.3   2.3   2.3


       Creating a matrix of size 2 by 4 out of a double 5.7

5.7   5.7   0.0   0.0
5.7   5.7   0.0   0.0


       Creating a square matrix of size 3 by 3 given an array of doubles

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


       Creating a rectangular matrix of size 4 by 2 given an array of doubles

2.0   0.0
3.0   -1.0
-5.0  3.0
1.0   4.0


Creating a rectangular matrix of size 3 by 3 given a 2D array of doubles

0.0   1.0   2.0
2.0   3.0   4.0
4.0   5.0   6.0
```

Testing method getRows

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

3 rows

Testing method getColumns

```
2.0   0.0
3.0   -1.0
-5.0  3.0
1.0   4.0
```

2 columns

Testing method getElement.

```
0.0   1.0
-1.0  0.0
```

Element index [2][1] is -1.0

Testing method getElements

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Elements

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Testing method RowsEven

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Checking if number of rows is even

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Is the number of rows even?

No

Checking if number of rows is even

-1.0 1.0
0.0   -3.0
3.0   0.0
2.0   1.0

Is the number of rows even?

Yes

Checking if number of columns is even

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0

Is the number of columns even?

No

Checking if number of columns is even

-1.0 1.0
0.0   -3.0
3.0   0.0
2.0   1.0

Is the number of columns even?

Yes

Testing shape

2.0
3.0
-6.0

Non-square matrix

Testing shape

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0

Square matrix

Invertability test

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Non-singular (invertible) matrix

Testing transposition

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Transposed:

```
2.0   1.0   2.0
1.0   0.0   3.0
4.0   2.0   1.0
```

Testing scalar multiplicaiton.

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Testing scalar multiplication. Multiplying by 2

```
4.0   2.0   8.0
2.0   0.0   4.0
4.0   6.0   2.0
```

Testing matrix addition.

```
2.0   0.0
3.0   -1.0
-5.0  3.0
1.0   4.0
```

```
-1.0  1.0
0.0   -3.0
3.0   0.0
2.0   1.0
```

Adding matrices:

```
1.0   1.0
```

```
3.0   -4.0
-2.0  3.0
3.0   5.0
```

Testing matrix subtraction.

```
2.0   0.0
3.0   -1.0
-5.0  3.0
1.0   4.0



-1.0  1.0
0.0   -3.0
3.0   0.0
2.0   1.0
```

Subtracting matrices

```
3.0   -1.0
3.0   2.0
-8.0  3.0
-1.0  3.0
```

Testing matrix multiplication.

```
2.0   0.0
3.0   -1.0
-5.0  3.0
1.0   4.0



0.0   1.0
-1.0  0.0
```

Multiplying matrices

```
0.0   2.0
1.0   3.0
-3.0  -5.0
-4.0  1.0
```

Testing Strassen matrix multiplication.

```
0.0   -3.0  -6.0  -9.0
4.0   1.0   -2.0  -5.0
8.0   5.0   2.0   -1.0
12.0  9.0   6.0   3.0



0.0   3.0   6.0   9.0
2.0   5.0   8.0   11.0
```

```
4.0   7.0   10.0  13.0
6.0   9.0   12.0  15.0


       Multiplying matrices

-84.0 -138.0      -192.0      -246.0
-36.0 -42.0 -48.0 -54.0
12.0  54.0  96.0  138.0
60.0  150.0 240.0 330.0



       Testing Strassen matrix multiplication.

0.0   1.0   2.0
2.0   3.0   4.0
4.0   5.0   6.0



0.0   2.0   4.0
1.0   3.0   5.0
2.0   4.0   6.0


       Multiplying matrices

5.0   11.0  17.0
11.0  29.0  47.0
17.0  47.0  77.0



       Testing matrix exponentiation to the power of two.

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


       Squared matrix:

13.0  14.0  14.0
6.0   7.0   6.0
9.0   5.0   15.0



       Testing matrix exponentiation to the power of tree.

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


       Matrix to the power of 3

68.0  55.0  94.0
31.0  24.0  44.0
53.0  54.0  61.0
```

Creating augmented matrix (with identity matrix)

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Augmented matrix:

```
2.0   1.0   4.0   1.0   0.0   0.0
1.0   0.0   2.0   0.0   1.0   0.0
2.0   3.0   1.0   0.0   0.0   1.0
```

Creating augmented matrix (with a vector).

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0



2.0
3.0
-6.0
```

Augmented

```
2.0   1.0   4.0   2.0
1.0   0.0   2.0   3.0
2.0   3.0   1.0   -6.0
```

Creating augmented matrix (vertical augmentation).

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0



0.0   1.0   2.0
2.0   3.0   4.0
4.0   5.0   6.0
```

Augmented

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
0.0   1.0   2.0
2.0   3.0   4.0
4.0   5.0   6.0
```

Creating augmented matrix with a column vector

```
2.0   1.0   4.0
```

```
1.0   0.0   2.0
2.0   3.0   1.0


2.0
3.0
-6.0

       Augmented matrix:

2.0   1.0   4.0   2.0
1.0   0.0   2.0   3.0
2.0   3.0   1.0   -6.0


       Creating augmented matrix with a vector

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


2.0
3.0
-6.0

       Augmented matrix:

2.0   1.0   4.0   2.0
1.0   0.0   2.0   3.0
2.0   3.0   1.0   -6.0


       Testing elementary row operations

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


       Multiplying row 3 by 2.0

2.0   1.0   4.0
1.0   0.0   2.0
4.0   6.0   2.0


       Testing elementary row operations

2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0


       Row1 + 2.0*row3

6.0   7.0   6.0
1.0   0.0   2.0
```

```
2.0   3.0   1.0
```

Testing elementary row operations

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Swap row 1 and row 3

```
2.0   3.0   1.0
1.0   0.0   2.0
2.0   1.0   4.0
```

Testing elementary column operations

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Multiplying column 3 by 2.0

```
2.0   1.0   8.0
1.0   0.0   4.0
2.0   3.0   2.0
```

Testing elementary column operations

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Column1 + 2.0*column3

```
10.0  1.0   4.0
5.0   0.0   2.0
4.0   3.0   1.0
```

Testing elementary column operations

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Swap column 1 and column 3

```
4.0   1.0   2.0
2.0   0.0   1.0
1.0   3.0   2.0
```

Eliminating a row and a column

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Eliminate row 1 and column 3

```
1.0   0.0
2.0   3.0
```

Computing determinant

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Determinant equals 3.0

Computing minor

```
2.0   5.0
-3.0  -7.0
```

1-2 minor of the matrix equals -3.0

Computing cofactor

```
2.0   5.0
-3.0  -7.0
```

2-2 cofactor of the matrix equals 2.0

Computing adjugate of the matrix

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Adjugate of the matrix

```
-6.0  11.0  2.0
3.0   -6.0  -0.0
3.0   -4.0  -1.0
```

Computing inverse of the matrix

```
2.0   1.0   4.0
1.0   0.0   2.0
```

```
2.0   3.0   1.0
```

Inverse of the matrix

```
-2.0  3.6666666666666665      0.6666666666666666
1.0   -2.0                    -0.0
1.0   -1.3333333333333333     -0.3333333333333333
```

Find the inverse

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0
```

Inverse using Gaussian Elimination

```
-2.0  3.6666666666666665      0.6666666666666666
1.0   -2.0                    0.0
1.0   -1.3333333333333333     -0.3333333333333333
```

Solve the system

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0



2.0
3.0
-6.0
```

using Cramer's rule

```
3.0
-4.0
0.0
```

Solve the system

```
2.0   1.0   4.0
1.0   0.0   2.0
2.0   3.0   1.0



2.0
3.0
-6.0
```

using inverse

```
3.0
```

-4.0
0.0

Solve the system

2.0 1.0 4.0
1.0 0.0 2.0
2.0 3.0 1.0

2.0
3.0
-6.0

using Gaussian Elimination

3.0
-4.0
0.0

8.3 Appendix C: System source code.

```
/**
 * @author cs2nk
 * 06.05.2005
 *
 * This Computer Linear Algebra System is designed for sumission to the
University of Bath
 * as a part of the final year project
 */
package matrix;


public class Matrix {

    /*
     * Constructs a Matrix object
     */

    /**
     *  <code>rows</code> of the matrix
     */

    /**
     *  <code>columns</code> of the matrix
     */
    public int rows, columns;


      /**
       *  <code>data</code> array of doubles the constructor turns
       * into the elements of the matrix
       */
      public double[] data;
      /**
       * <code>data_</code> a 2D array of doubles the constructor turns
       * into the elements of the matrix
       */
      /**
       * <code>_matrix</code> elements of the matrix
       */
      public double[][] data_, _matrix;


      /**
       * Constructs a square zero matrix
       * @param r  number of rows and columns of the matrix
       */
      public Matrix(int r) {
            rows = r;
            columns = r;
            _matrix = new double[rows][columns];

    }

      /**
       * Constructs a rectangular zero matrix
       * @param r rows of the matrx
       * @param c columns of the matrix
       */
      public Matrix(int r, int c) {
            rows = r;
```

```
        columns = c;
        _matrix = new double[rows][columns];
}

 /**
 *  Constructs a square matrix filled with one element
 * @param r rows and columns of the matrix
 * @param element element of the matrix
 */
public Matrix(int r, double element) {
     rows = r;
     columns = r;
     _matrix = new double[rows][columns];
     for(int i = 0; i < rows; i++) {
        for (int j = 0; j < rows; j++) {
                    _matrix[i][j] = element;
        }
        }
 }

 /**
 * Constructs a rectangular matrix filled with the same element
 * @param r rows of the matrix
 * @param c columns of the matrix
 * @param element elements of the matrix
 */
public Matrix(int r, int c, double element) {
     rows = r;
     columns = c;
     _matrix = new double[rows][columns];
     for(int i = 0; i < rows; i++) {
        for (int j = 0; j < rows; j++) {
                    _matrix[i][j] = element;
        }
        }
 }

 /**
 * Constructs a square matrix of size out of the array of doubles
 * @param r rows and columns of the matrix
 * @param d elements of the matrix
 */
public Matrix(int r, double[] d) {
        rows = r;
        columns = r;
        data = d;
        int index = 0;
        _matrix = new double[rows][columns];
        for(int i = 0; i < rows; i++) {
        for (int j = 0; j < rows; j++) {
                    _matrix[i][j] = data[index];
                    index++;
        }
        }
  }

/**
 * Constructs a rectangular matrix out of the array of doubles
 * @param r rows of the matirx
 * @param c columns of the matrix
 * @param d elements of the matrix
 */
public Matrix(int r, int c, double[] d) {
        rows = r;
```

```
        columns = c;
        data = d;
        int index = 0;
        _matrix = new double[rows][columns];
        for(int i = 0; i < rows; i++) {
              for (int j = 0; j < columns; j++) {
                    _matrix[i][j] = data[index];
                    index++;
              }
        }
}

  /**
   * Constructs a rectangular matrix out of the array of doubles
   * @param r rows of the matrix
   * @param c columns of the matrix
   * @param da elements of the matrix
   */
  public Matrix(int r, int c, double[][] da) {
        rows = r;
        columns = c;
        data_ = da;
        int index = 0;
        _matrix = new double[rows][columns];
        for(int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
              _matrix[i][j] = data_[i][j];
              index++;
            }
        }
  }

  /**
   * Returns the number of rows of the matrix
   * @return number of rows of the matrix
   */
  public int getRows() {
        return rows;
}

/**
 * Returns the number of columns of the matrix
 * @return number of columns of the matrix
 */
public int getColumns() {
        return columns;
}

/**
 * Returns the ij-th element of the matrix
 * @param i row
 * @param j column
 * @return ij-th element of the matrix
 */
public double getElement(int i, int j) {
  if (i < 0 || i >= this.getRows() || j < 0 || j >= this.getColumns()) {
        throw new IllegalArgumentException("Array out of bounds");
        }
        return _matrix[i][j];
}

/**
 * Returns elements of the matrix arranged into a 2D array
 * @return array of doubles
```

```
     */
    public double[][] getElements() {
            return _matrix;
    }

    /**
     * Returns String representation of the matrix
     * @return String representation of the matrix
     */
    public String matrixWrite() {
      String str = "";
        for(int i = 0; i < this.getRows(); i++) {
            for (int j = 0; j < this.getColumns(); j++) {
                double d = this.getElement(i,j);
                        str =
str.concat(String.valueOf(this.getElement(i,j))+"\t");
            }
            str = str.concat("\n");
            }
        return str;
    }

    /**
     * Returns true if the matrix has even number of rows
     * @return true if the matrix has even number of rows
     */
    public boolean rowsEven() {
        return ((this.getRows()/2)*2 == this.getRows());
    }

    /**
     * Returns true if the matrix has even number of columns
     * @return true if the matrix has even number of columns
     */
    public boolean colsEven() {
        return ((this.getColumns()/2)*2 == this.getColumns());
    }

     /**
     * Returns true if the matrix is square
     * @return true if the matrix is square
     */
    public boolean isSquare() {
      return (this.getRows() == this.getColumns());
    }

    /**
     * Returns true if the matrix is invertible
     * @return True if the matrix is invertible
     */
    public boolean isInvertible() {
            return(this.det() != 0);
    }

    /**
     * Returns a transposed matrix
     * @return a transposed of this matrix
     */
    public Matrix transpose() {
            double[][] matrix = new
double[this.getColumns()][this.getRows()];
            for(int i = 0; i < this.getRows(); i++) {
               for (int j = 0; j < this.getColumns(); j++) {
                  matrix[i][j] = this.getElement(j,i);
```

```
            }
          }
          return new Matrix(this.getColumns(), this.getRows(), matrix);
    }


    /**
     * Returns this matrix multiplied by a scalar
     * @param scalar scalar
     * @return matrix multiplied by a scalar
     */
    public Matrix scalarMult(double scalar) {
          double[][] matrix = new
double[this.getRows()][this.getColumns()];
          for(int i = 0; i < this.getRows(); i++) {
           for (int j = 0; j < this.getColumns(); j++) {
                matrix[i][j] = this.getElement(i,j)*scalar;
              }
          }
          return new Matrix(this.getRows(), this.getColumns(), matrix);
      }

     /**
     * Retuns sum of this matrix and matrix A
     * @param A matrix
     * @return matrix sum of this matrix and matrix A
     */
    public Matrix matrixAdd(Matrix A) {
      if (this.getRows()!= A.getRows() || this.getColumns()!= A.getColumns())
{
          throw new IllegalArgumentException("Action not defined: Can not
add matrices. Wrong column size");
          }
                double[][] matrix = new
double[this.getRows()][this.getColumns()];
                for(int i = 0; i < this.getRows(); i++) {
                for (int j = 0; j < this.getColumns(); j++) {
                           matrix[i][j] =
this.getElement(i,j)+A.getElement(i,j);
                     }
                }
                return new Matrix(this.getRows(), this.getColumns(),
matrix);
          }

     /**
     * Returns difference of this matrix and matrix A
     * @param A matrix
     * @return difference of this matrix and matrix A
     */
    public Matrix matrixSubtract(Matrix A) {
      if (this.getRows()!= A.getRows() || this.getColumns()!= A.getColumns())
{
          throw new IllegalArgumentException("Action not defined: Can not
subtract matrices. Wrong column size");
          }
                double[][] matrix = new
double[this.getRows()][this.getColumns()];
                for(int i = 0; i < this.getRows(); i++) {
                for (int j = 0; j < this.getColumns(); j++) {
                           matrix[i][j] =this.getElement(i,j)-
A.getElement(i,j);
                     }
                }
```

```
                return new Matrix(this.getRows(), this.getColumns(),
matrix);
            }

     /**
     * Returns the product of this matrix and matrix A
     * @param A matrix
     * @return matrix product of this matrix and matrix A
     */
    public Matrix matrixMult(Matrix A) {
      if (this.getColumns()!= A.getRows()) {
            throw new IllegalArgumentException("Action not defined: Can not
multiply matrices. Wrong dimentions");
            }
                double[][] matrix = new
double[this.getRows()][A.getColumns()];
                for(int i = 0; i < this.getRows(); i++) {
                      for(int j = 0; j < A.getColumns(); j++) {
                      for(int k = 0; k < this.getColumns(); k++) {
                            matrix[i][j] +=
this.getElement(i,k)*A.getElement(k,j);

                      }
                      }
                }
                return new Matrix(this.getRows(), A.getColumns(), matrix);
            }


      /**
     * Performs Strassen multiplication. Returns product of this matrix and
matrix A
     * @param B matrix
     * @return matrix product of this matrix and matrix A
     */
    public Matrix matrixStrassenMult(Matrix B) {
          if (!this.isSquare() || !B.isSquare()) {
                  throw new IllegalArgumentException("Action not defined: Can
not multiply matrices using Strassen algorithm. Matrices not square");
          }
          if (this.getRows()!= B.getRows()) {
            throw new IllegalArgumentException("Action not defined: Can not
multiply matrices. Wrong dimentions");
          }
          if (this.rowsEven()){
              double[][] _A11 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _A12 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _A21 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _A22 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _B11 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _B12 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _B21 = new
double[this.getRows()/2][this.getRows()/2];
              double[][] _B22 = new
double[this.getRows()/2][this.getRows()/2];

              for (int i = 0; i < this.getRows()/2; i++) {
                  for (int j = 0; j < this.getRows()/2; j++) {
```

```
                    _A11[i][j] = this.getElement(i,j);
                    _A12[i][j] = this.getElement(i,j + this.getRows()/2);
                    _A21[i][j] = this.getElement(i + this.getRows()/2,j);
                    _A22[i][j] = this.getElement(i + this.getRows()/2,j+
this.getRows()/2);
                    _B11[i][j] = B.getElement(i,j);
                    _B12[i][j] = B.getElement(i,j + this.getRows()/2);
                    _B21[i][j] = B.getElement(i + this.getRows()/2,j);
                    _B22[i][j] = B.getElement(i + this.getRows()/2,j+
this.getRows()/2);
                }
            }

            Matrix A11 = new Matrix (this.getRows()/2,
this.getRows()/2,_A11);
            Matrix A12 = new Matrix (this.getRows()/2,
this.getRows()/2,_A12);
            Matrix A21 = new Matrix (this.getRows()/2,
this.getRows()/2,_A21);
            Matrix A22 = new Matrix (this.getRows()/2,
this.getRows()/2,_A22);
            Matrix B11 = new Matrix (this.getRows()/2,
this.getRows()/2,_B11);
            Matrix B12 = new Matrix (this.getRows()/2,
this.getRows()/2,_B12);
            Matrix B21 = new Matrix (this.getRows()/2,
this.getRows()/2,_B21);
            Matrix B22 = new Matrix (this.getRows()/2,
this.getRows()/2,_B22);


            Matrix P1 =
(A11.matrixAdd(A22)).matrixMult(B11.matrixAdd(B22));
            Matrix P2 = (A21.matrixAdd(A22)).matrixMult(B11);
            Matrix P3 = A11.matrixMult(B12.matrixSubtract(B22));
            Matrix P4 = A22.matrixMult(B21.matrixSubtract(B11));
            Matrix P5 = (A11.matrixAdd(A12)).matrixMult(B22);
            Matrix P6 =
(A21.matrixSubtract(A11)).matrixMult(B11.matrixAdd(B12));
            Matrix P7 =
(A12.matrixSubtract(A22)).matrixMult(B21.matrixAdd(B22));


            Matrix C11 =
((P1.matrixAdd(P4)).matrixSubtract(P5)).matrixAdd(P7);
            Matrix C12 = P3.matrixAdd(P5);
            Matrix C21 = P2.matrixAdd(P4);
            Matrix C22 =
((P1.matrixAdd(P3)).matrixSubtract(P2)).matrixAdd(P6);

            Matrix C =
(C11.matrixAugment(C12)).matrixVertAugment(C21.matrixAugment(C22));

            return C;
        }

        else {
            double _a22 = this.getElement(this.getRows()-1, this.getRows()-
1);
            double _b22 = this.getElement(this.getRows()-1, this.getRows()-
1);
            double[][] _a12 = new double[this.getRows()-1][1];
            double[][] _a21 = new double[1][this.getRows()-1];
            double[][] _b12 = new double[this.getRows()-1][1];
```

```
            double[][] _b21 = new double[1][this.getRows()-1];
            for(int i = 0; i < this.getRows()-1; i++ ) {
                _a12[i][0] = this.getElement(i,this.getRows()-1);
                _a21[0][i] = this.getElement(this.getRows()-1,i);
                _b12[i][0] = B.getElement(i,this.getRows()-1);
                _b21[0][i] = B.getElement(this.getRows()-1,i);
            }

            Matrix a12 = new Matrix(this.getRows()-1,1,_a12);
            Matrix a21 = new Matrix(1, this.getRows()-1,_a21);
            Matrix a22 = new Matrix(1,_a22);
            Matrix b12 = new Matrix(this.getRows()-1,1,_b12);
            Matrix b21 = new Matrix(1,this.getRows()-1,_b21);
            Matrix b22 = new Matrix(1,_b22);

            Matrix A11 = this.deleteRowCol(this.getRows()-1,this.getRows()-
1);
            Matrix B11 = B.deleteRowCol(this.getRows()-1,this.getRows()-1);

            Matrix C11 =
(A11.matrixStrassenMult(B11)).matrixAdd(a12.matrixMult(b21));
            Matrix C12 =
(A11.matrixMult(b12)).matrixAdd(a12.scalarMult(_b22));
            Matrix C21 =
(a21.matrixMult(B11)).matrixAdd(b21.scalarMult(_a22));
            Matrix C22 =
(a21.matrixMult(b12)).matrixAdd(a22.matrixMult(b22));

            Matrix C =
(C11.matrixAugment(C12)).matrixVertAugment(C21.matrixAugment(C22));
            return C;
          }
      }

     /**
     * Returns a squared matrix
     * @return squared matrix
     */
    public Matrix matrixExp() {
      if (!this.isSquare()) {
            throw new IllegalArgumentException("Can not exponentiate non-
square matrix");
            }
                  return this.matrixMult(this);
    }

    /**
     * Returns matrix to the power degree
     * @param degree degree
     * @return matrix to the power degree
     */
    public Matrix matrixExp(int degree) {
      if (!this.isSquare()) {
            throw new IllegalArgumentException("Can not exponentiate non-
square matrix");
            }
                  Matrix A = this;
                  for(int i = 1; i < degree; i++){
                        A = A.matrixMult(this);
                  }
            return A;
    }

    /**
```

```
     * Creates augmented matrix using identity matrix
     * @return matrix augmented with identity
     */
   public Matrix matrixAugment() {
      if (!this.isSquare()) {
            throw new IllegalArgumentException("Can not augment non-square
matrix");
            }
                  IdentityMatrix ident = new
IdentityMatrix(this.getColumns());
                  double[][] matrix = new
double[this.getRows()][2*this.getColumns()];
                  for(int i = 0; i < this.getRows(); i++) {
                  for (int j = 0; j < 2*this.getColumns(); j++) {
                              if(j < this.getColumns()) {
                              matrix[i][j] = this.getElement(i,j);

                              }
                              else {
                              matrix[i][j] = ident.getElement(i,j -
this.getColumns());
                              }
                  }
                  }

            return new Matrix(this.getRows(),2*this.getColumns(), matrix);
    }

    /**
     * Creates augmented matrix with the vector taking an array of doubles as
an input
     * @param vect vecor
     * @return matrix augmeted with a vector
     */
    public Matrix matrixAugmentVector(double[] vect) {
      if (this.getRows()!= vect.length) {
            throw new IllegalArgumentException("Can not create augmented
matrix. Wrong dimentions");
            }
                  double[][] matrix = new
double[this.getRows()][this.getColumns()+1];
                  for(int i = 0; i < this.getRows(); i++) {
                  for (int j = 0; j < this.getColumns()+1; j++) {
                              if(j < this.getColumns()) {
                              matrix[i][j] = this.getElement(i,j);

                              }
                              else {
                                    matrix[i][j] = vect[i];
                              }
                  }
                  }
                  return new Matrix(this.getRows(),this.getColumns()+1,
matrix);
    }

    /**
     * Creates augmented matrix with the vector using matirx representation
of a vetor
     * @param vect vector
     * @return matrix augmented with a vector
     */
    public Matrix matrixAugmentVector(Matrix vect) {
      if (this.getRows()!= vect.getRows() || vect.getColumns() != 1) {
```

```
            throw new IllegalArgumentException("Can not create augmented
matrix. Wrong dimentions");
            }
                  double[][] matrix = new
double[this.getRows()][this.getColumns()+1];
                  for(int i = 0; i < this.getRows(); i++) {
                  for (int j = 0; j < this.getColumns()+1; j++) {
                              if(j < this.getColumns()) {
                              matrix[i][j] = this.getElement(i,j);

                              }
                              else {
                                    matrix[i][j] = vect.getElement(i,0);
                              }
                  }
                  }
                  return new Matrix(this.getRows(),this.getColumns()+1,
matrix);
    }

    /**
     * Creates augmented matrix using matrix A
     * @param A matrix
     * @return matrix augmented with A
     */
    public Matrix matrixAugment(Matrix A) {
      if (this.getRows() != A.getRows()) {
            throw new IllegalArgumentException("Can not augment matrices of
wrong row dimentions");
            }
                  double[][] matrix = new
double[this.getRows()][this.getColumns()+ A.getColumns()];
                  for(int i = 0; i < this.getRows(); i++) {
                  for (int j = 0; j < this.getColumns() + A.getColumns();
j++) {
                              if(j < this.getColumns()) {
                              matrix[i][j] = this.getElement(i,j);

                              }
                              else {
                              matrix[i][j] = A.getElement(i,j -
this.getColumns());
                              }
                  }
                  }

                  return new
Matrix(this.getRows(),this.getColumns()+A.getColumns(), matrix);
    }

    /**
     * Performs vertical augmentation using matrix A
     * @param A matrix
     * @return matrix vertically augmented with A
     */
    public Matrix matrixVertAugment(Matrix A) {
      if (this.getColumns() != A.getColumns()) {
            throw new IllegalArgumentException("Can not augment matrices of
wrong column dimentions");
            }
                  double[][] matrix = new double[this.getRows() +
A.getRows()][this.getColumns()];
                  for(int i = 0; i < this.getRows() + A.getRows(); i++) {
                  for (int j = 0; j < this.getColumns(); j++) {
```

```
                            if(i < this.getRows()) {
                            matrix[i][j] = this.getElement(i,j);

                            }
                            else {
                            matrix[i][j] = A.getElement(i -
this.getRows(),j);
                            }
                    }
                    }

                    return new
Matrix(this.getRows()+A.getRows(),this.getColumns(), matrix);
    }

    /**
     * Multiplies a row with a scalar
     * @param row row
     * @param scalar scalar
     * @return matrix with the row multiplied wby the scalar
     */
    public Matrix multRow(int row, double scalar) {
            double[][] matrix = this.getElements();
            for(int i = 0; i < this.getColumns(); i ++) {
            matrix[row][i] *= scalar;
            }
            return new Matrix(this.getRows(),this.getColumns(), matrix);
    }

     /**
     * Adds a multiple of one row to another row
     * @param row1 frist row
     * @param row2 second row
     * @param scalar scalar
     * @return matrix with the multiple of the second row added to the first
row
     */
    public Matrix addRowMult(int row1, int row2, double scalar) {
            double[][] matrix = this.getElements();
            for(int j = 0; j < this.getColumns(); j ++) {
                matrix[row1][j] +=this.getElement(row2,j)*scalar;
            }
            return new Matrix(this.getRows(),this.getColumns(), matrix);
    }

    /**
     * Swaps two rows
     * @param row1 first row
     * @param row2 second row
     * @return matrix with the first and the second row swapped
     */
    public Matrix swapRow(int row1, int row2) {
            double[][] matrix = new
double[this.getRows()][this.getColumns()];
            for(int i = 0; i < this.getRows(); i ++) {
                for(int j = 0; j < this.getColumns(); j++) {
                        if(i == row1){
                            matrix[i][j] = this.getElement(row2, j);
                        }
                        else if(i == row2) {
                              matrix[i][j] = this.getElement(row1, j);
                        }
                        else{
                          matrix[i][j] = this.getElement(i,j);
```

```
                        }
                    }
                }
                return new Matrix(this.getRows(),this.getColumns(), matrix);
        }

        /**
         * Multiplies a column with a scalar
         * @param col columns
         * @param scalar scalar
         * @return matrix with the columns multiplied by the scalar
         */
        public Matrix multCol(int col, double scalar) {
                double[][] matrix = this.getElements();
                for(int i = 0; i < this.getRows(); i ++) {
                matrix[i][col] *= scalar;
                }
                return new Matrix(this.getRows(),this.getColumns(), matrix);
    }

        /**
         * Adds a multiple of one column to another column
         * @param col1 first columns
         * @param col2 second column
         * @param scalar scalar
         * @return matrix with the first column multiplied by the second column
         */
    public Matrix addColMult(int col1, int col2, double scalar) {
                double[][] matrix = this.getElements();
                for(int i = 0; i < this.getRows(); i ++) {
                    matrix[i][col1] +=this.getElement(i,col2)*scalar;
                }
                return new Matrix(this.getRows(),this.getColumns(), matrix);
    }

    /**
         * Swaps two columns
         * @param col1 first column
         * @param col2 second column
         * @return matrix with the first column swapped with the second column
         */
    public Matrix swapCol(int col1, int col2) {
                double[][] matrix = new
double[this.getRows()][this.getColumns()];
                for(int i = 0; i < this.getRows(); i ++) {
                    for(int j = 0; j < this.getColumns(); j++) {
                            if(j == col1){
                                matrix[i][j] = this.getElement(i, col2);
                            }
                            else if(j == col2) {
                                  matrix[i][j] = this.getElement(i, col1);
                            }
                            else{
                              matrix[i][j] = this.getElement(i,j);
                            }
                    }
                }
                return new Matrix(this.getRows(),this.getColumns(), matrix);
        }

        /**
         * Returns the matrix with the row r and column c deleted
         * @param r row
         * @param c column
```

```
       * @return matrix with the row r and column c deleted
       */
      public Matrix deleteRowCol(int r, int c) {
            double[][] matrix = new double[this.getRows()-
1][this.getColumns()-1];
            int a,b;
            for(int i = 0; i < this.getRows()-1; i++) {
            for (int j = 0; j < this.getColumns()-1; j++) {
                        if (i < r) {
                        a = i;
                  }
                  else {
                        a = i+1;
                  }
                  if (j < c) {
                              b = j;
                        }
                        else  {
                        b = j+1;
                        }
                  matrix[i][j] = this.getElement(a,b);
                  }
            }
            return new Matrix(this.getRows()-1,this.getColumns()-1, matrix);
    }

      /**
       * Computes determinant of the matrix
       * @return determinant of the matrix
       */
      public double det() {
            if(!this.isSquare()) {
                  throw new IllegalArgumentException("Can not find
determinant of non-square matrix");
            }
            double det = 0;
          if (this.getRows() == 1) {
            det = this.getElement(0,0);
          }
          else if (this.getRows() == 2) {
                  det =  this.getElement(0,0)*this.getElement(1,1)-
this.getElement(0,1)*this.getElement(1,0);
          }
          else {
                for(int j = 0; j < this.getRows(); j++) {
                        Matrix matrix_ = this.deleteRowCol(0,j+1);
                        det = det + Math.pow((-
1),(2+j))*this.getElement(0,j)*matrix_.det();
                 }
          }
          return det;
    }

    /**
     * Computes the ij-th minor of the matrix
     * @param r row
     * @param c column
     * @return the ij-th minor of the matrix
     */
    public double minor(int r, int c) {
      if(!this.isSquare()) {
                  throw new IllegalArgumentException("Can not find minor of
non-square matrix");
            }
```

```
            Matrix matrix_ = this.deleteRowCol(r,c);
            double minor = matrix_.det();
                  return minor;
    }

    /**
     * Computes the ij-th cofactor of the matrix
     * @param r row
     * @param c column
     * @return the ij-th cofactor of the matrix
     */
    public double cofactor(int r, int c) {
            if(!this.isSquare()) {
                  throw new IllegalArgumentException("Can not compute
cofactor of a non-square matrix");
            }
                  double cofactor = Math.pow((-1), (r+c))*this.minor(r,c);
                  return cofactor;
    }

    /**
     * Returns the adjugate of this matrix
     * @return the adjugate of this matrix
     */
    public Matrix adjugate() {
      if (!this.isSquare()) {
        throw new IllegalArgumentException("Can not compute adjugate of a
non-square matrix");
      }
       double[][] matrix = new double[this.getRows()][this.getColumns()];
      for (int i = 0; i < this.getRows(); i++) {
        for (int j = 0; j < this.getColumns(); j++) {
          matrix[j][i] = this.cofactor(i, j);
        }
      }
      return new Matrix(this.getRows(), this.getColumns(), matrix);
    }

    /**
     * Calculates the inverse of the matrix
     * @return the inverse of this matrix
     */
    public Matrix inverse() {
            if (!this.isSquare()) {
        throw new IllegalArgumentException("Can not compute inverse of a non-
square matrix");
            }
            if (!this.isInvertible()) {
        throw new IllegalArgumentException("Can not compute adjugate of a
singular matrix");
            }
         Matrix matrix_ = this.adjugate();
         Matrix inverse = matrix_.scalarMult(1/this.det());
         return inverse;
    }

    /**
     * Computes the matrix inverse using Gauss-Jordan Elimination
     * @return the inverse of this matrix
     */
    public Matrix gaussElimInvert() {
          if (!this.isSquare()) {
          throw new IllegalArgumentException("Can not find the inverse " +
                  "(Gauss Elimination). The matrix is not square");
```

```
            }
        double[][] matrix = new double[this.getRows()][this.getColumns()];
        Matrix gauss = this.matrixAugment();
        int max = 0;
        for(int j = 0; j < this.getColumns(); j++) {
            for(int i = j+1; i < this.getRows(); i++) {
                if(gauss.getElement(i,j) > gauss.getElement(max,j)) {
                    max = i;
                }
            }
            if (max != j) {
                gauss = gauss.swapRow(j,max);
            }
            for(int i = j+1; i < this.getRows(); i++) {
                gauss = gauss.addRowMult(i,j,(-1*gauss.getElement(i,j)/
                        gauss.getElement(j,j)));
            }
        }
        for(int i = 0; i < this.getRows(); i++) {
            gauss = gauss.multRow(i,(1/gauss.getElement(i,i)));
        }
        for(int j = this.getColumns()-1; j > 0; j--) {
            for(int i = 0; i < j; i++) {
                gauss = gauss.addRowMult(i,j,(-1*gauss.getElement(i,j)));
            }
        }
        for(int i = 0; i < this.getRows(); i++) {
            for(int j = 0; j < gauss.getColumns(); j ++) {
                if(j >= this.getColumns()) {
                    matrix[i][j-this.getColumns()] = gauss.getElement(i,j);
                }
            }
        }
        return new Matrix(this.getRows(),this.getColumns(), matrix);
      }

    /**
     * Solves a system of equations using Cramer's rule
     * @param B column vector
     * @return vector of answers
     */
    public Matrix cramerSolve(Matrix B) {
      if (!this.isSquare()) {
        throw new IllegalArgumentException("Can not solve the system(Cramer).
The matrix is not square");
            }
            if (!this.isInvertible()) {
        throw new IllegalArgumentException("Can not solve the system(Cramer).
Tne matrix is singular");
            }
            if(this.getRows()!= B.getRows() || B.getColumns()!= 1) {
                  throw new IllegalArgumentException("Can not solve the
system(Cramer). Wrong dimentions");
            }
      Matrix matrix_ = this.adjugate();
      double[][] matrix = new double[this.getRows()][1];
            for(int i = 0; i < this.getRows(); i++){
                  for(int j = 0; j < this.getColumns(); j++) {
                        matrix[i][0] +=
((matrix_.getElement(i,j))/this.det())*B.getElement(j,0);
                  }
            }
      return new Matrix(this.getRows(),1, matrix);
      }
```

```
      /** Solves a system of equations multiplying by the inverse
       * @param B column vector
       * @return vector of answers
       */
      public Matrix inverseSolve(Matrix B) {
          if (!this.isSquare()) {
                  throw new IllegalArgumentException("Can't solve the
system(Cramer). The matrix is not square");
          }
          if (!this.isInvertible()) {
                  throw new IllegalArgumentException("Can't solve the
system(Cramer). Tne matrix is singular");
          }
          if(this.getRows()!= B.getRows() || B.getColumns()!= 1) {
              throw new IllegalArgumentException("Can't solve the
system(Cramer). Wrong dimentions");
          }
          Matrix inverse = this.inverse();
          Matrix matrix = inverse.matrixMult(B);
          return matrix;
      }

      /**
       * Solves a system of equations using Gauss-Jordan elimination
       * @param B column vector
       * @return vector of answers
       */
      public Matrix gaussElimSolve(Matrix B) {
          if (!this.isSquare()) {
                throw new IllegalArgumentException("Can't solve the
system(Cramer). The matrix is not square");
            }
          double[] x = new double[this.getRows()];
          Matrix gauss = this.matrixAugment(B);
          int max = 0;
          for(int j = 0; j < this.getColumns(); j++) {
                              for(int i = j+1; i < this.getRows(); i++) {
                  if(gauss.getElement(i,j) > gauss.getElement(max,j)) {
                      max = i;
                  }
              }
              if (max != j) {
                  gauss = gauss.swapRow(j,max);
              }
              for(int i = j+1; i < this.getRows(); i++) {
                  gauss = gauss.addRowMult(i,j,(-
1*gauss.getElement(i,j)/gauss.getElement(j,j)));
              }
          }
          double sum = 0.0;
          for(int i = this.getRows()-1; i >= 0; i--) {
              for(int j = i+1; j < this.getRows(); j++) {
                    x[i] = (gauss.getElement(i, gauss.getColumns()-1)-
sum)/gauss.getElement(i,i);
                    sum += x[j]*gauss.getElement(i,j);
              }
          }
          return new Matrix(this.getRows(),1,x);
      }
}

/**
 * @author cs2nk
```

```
 * 06.05.2005
 *
 * This Computer Linear Algebra System is designed for sumission to the
University of Bath
 * as a part of the final year project
 */
package matrix;


public class IdentityMatrix extends Matrix {

    /**
     * Elements of Identity Matrix <code>matrix</code>
     */
    public double[][] matrix;

    /**
     * Constructs an Identity matrix object
     * @param r rows, columns
     */
    public IdentityMatrix(int r) {
            super(r);
            matrix = new double[rows][rows];
            for(int i = 0; i < rows; i++) {
            for (int j = 0; j < rows; j++) {
                        if (i == j) {
                        matrix[i][j] = 1;
                        }
            }
            }
            this._matrix = matrix;
      }
}

/*
 * @author cs2nk
 * 06.05.2005
 *
 * This Computer Linear Algebra System is designed for sumission to the
University of Bath
 * as a part of the final year project
 */
package matrix;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
/*
 * An interface frame for the Linear Algebra System
 */
public class CASystem extends JFrame implements ItemListener, ActionListener
{

    private Font f10 = new Font("Courier", Font.PLAIN, 10);
    private Font f14 = new Font("Courier", Font.PLAIN, 14);

    private int action;

    private String matrix1, matrix2, output;
    private String helpStirng = "\n \t To create a matrix enter the number of
rows " +
            "(and columns for a rectangular matrix) in the Matrix A 'rows' "
+
            "(and 'cols') fields and enter the matrix " +
```

```
            "in the form of a two-dimentional array, separating elements in
each" +
            "row by pressing TAB and separating rows by pressing ENTER." +
            " For Example: \n \n 1 \t 2 \t 3 \n 4 \t 5 \t 6 \n 7 \t 8 \t 9 \n
\n " +
            "\t Choose an action by selection one of the radio buttons. \n
\n" +
            "\t Some options (scalar,minor etc) may require enteirng some
values or element " +
            "coordinates \n \n" +
            "\t If the action is performed over using two matrices, the
'Matrix B' text field " +
            "would be activated. Create the second matrix the same way you
have created matrix A. \n \n" +
            "\t If the system can offer more than one way of perfoming a
computation you can select" +
            "the algorithm you wish the system to choose. Otherwise the
system will choose the algorithm" +
            "itself. \n \n" +
            "\t If you need to create an identity matrix simply enter the
number of rows your matirx" +
            "should have and tick the box 'Identity' next to corresponding
text field. \n \n" +
            "\t After selecting an operaiton press COMPUTE \n \n" +
            "\t You can copy the result into 'Matrix A' or 'Matrix B' fields
by pressing " +
            "'Copy to A' or 'Copy to B' respectively. \n \n" +
            "\t To clear a text field press 'Clear' at the bottom of the
textfield. \n \n" +
            "\t To clear all textfield press 'Clear All'";
    private String [] optionNames = {"Transpose", "Inverse", "Multiply by
scalar", "Add", "Subtract",
            "Multiply","Exponentiate","Determinant", "Minor", "Cofactor",
            "Adjugate","Augment","ERO's","ECO's","Solve"};
    private String [] invert = {"default", "standard", "Gauss-Jordan"};
    private String [] multiply = {"default", "standard", "Strassen"};
    private String [] eRoperations = {"multiply row", "add row multiple",
"swap rows"};
    private String [] eCoperations = {"multiply col", "add col multiple",
"swap cols"};
    private String [] solve = {"default", "Inverse", "Cramer", "Gauss-
Jordan"};

      private JTabbedPane mainPane = new JTabbedPane();
    private JPanel worksheetPanel = new JPanel(new GridLayout(1,2,7,7));
    private JPanel matrixPanel = new JPanel(new GridLayout(3,1,5,5));
    private JPanel matrixOne = new JPanel(new BorderLayout());
    private JPanel matrixTwo = new JPanel(new BorderLayout());
    private JPanel matrixThree = new JPanel(new BorderLayout());
    private JPanel matrixOneTitle = new JPanel(new FlowLayout());
    private JPanel matrixTwoTitle = new JPanel(new FlowLayout());
    private JPanel matrixThreeTitle = new JPanel(new FlowLayout());
    private JPanel matrixOneBottom = new JPanel(new BorderLayout());
    private JPanel matrixTwoBottom = new JPanel(new BorderLayout());
    private JPanel matrixThreeBottom = new JPanel(new BorderLayout());
    private JPanel matirxThreeBot = new JPanel(new FlowLayout());
    private JPanel matirxThreeBotR = new JPanel(new BorderLayout());
    private JPanel optionsBottom = new JPanel(new FlowLayout());
    private JPanel optionsPanel = new JPanel(new BorderLayout());
    private JPanel optionsList = new JPanel(new GridBagLayout());
    private JPanel scalarOpns = new JPanel (new FlowLayout());
    private JPanel powerOpns = new JPanel (new FlowLayout());
    private JPanel minorOpns = new JPanel (new FlowLayout());
    private JPanel cofactorOpns = new JPanel (new FlowLayout());
```

```
    private JPanel eroOpns = new JPanel (new BorderLayout());
    private JPanel ecoOpns = new JPanel (new BorderLayout());
    private JPanel eroRows = new JPanel (new FlowLayout());
    private JPanel ecoCols = new JPanel (new FlowLayout());

    private GridBagConstraints constraints = new GridBagConstraints();

    private JCheckBox identity1 = new JCheckBox("Identity");
    private JCheckBox identity2 = new JCheckBox("Identity");

    private boolean editable1 = true;
    private boolean editable2 = true;
    private boolean useB;
    private boolean[] enabled = new boolean[13];

    private JTextField rows1 = new JTextField ("1",2);
    private JTextField cols1 = new JTextField (2);
    private JTextField rows2 = new JTextField ("1",2);
    private JTextField cols2 = new JTextField (2);
    private JTextField scalar = new JTextField ("1",2);
    private JTextField power = new JTextField ("2",2);
    private JTextField minorRow = new JTextField ("1",2);
    private JTextField minorCol = new JTextField ("1",2);
    private JTextField cofactorRow = new JTextField ("1",2);
    private JTextField cofactorCol = new JTextField ("1",2);
    private JTextField eroRow1 = new JTextField ("1",2);
    private JTextField eroRow2 = new JTextField ("1",2);
    private JTextField eroScalar = new JTextField ("1",2);
    private JTextField ecoCol1 = new JTextField ("1",2);
    private JTextField ecoCol2 = new JTextField ("1",2);
    private JTextField ecoScalar = new JTextField ("1",2);

    private JButton clear1 = new JButton("Clear");
    private JButton clear2 = new JButton("Clear");
    private JButton clear3 = new JButton("Clear");
    private JButton copy21 = new JButton("Copy to A");
    private JButton copy22 = new JButton("Copy to B");
    private JButton compute = new JButton("Compute");
    private JButton clearAll = new JButton("Clear all");

    private TextArea help = new
TextArea(helpStirng,1,30,TextArea.SCROLLBARS_VERTICAL_ONLY);
    private JTextArea input1 = new JTextArea(matrix1, 15,15);
    private JTextArea input2 = new JTextArea(matrix2, 15,15);
    private JTextArea result = new JTextArea(output, 15,15);

    private JLabel matrixA = new JLabel("Matrix A");
    private JLabel matrixB = new JLabel("Matrix B");
    private JLabel matrixR = new JLabel("Result");
    private JLabel scalarLabel = new JLabel("scalar");
    private JLabel powerLabel = new JLabel("power");
    private JLabel minorRowLabel = new JLabel("row");
    private JLabel minorColLabel = new JLabel("col");
    private JLabel cofRowLabel = new JLabel("row");
    private JLabel cofColLabel = new JLabel("col");
    private JLabel eroRow1Label = new JLabel("row1");
    private JLabel eroRow2Label = new JLabel("row2");
    private JLabel eroScalarLabel = new JLabel("scalar");
    private JLabel ecoCol1Label = new JLabel("col1");
    private JLabel ecoCol2Label = new JLabel("col2");
    private JLabel ecoScalarLabel = new JLabel("scalar");

    private JComboBox inverse = new JComboBox(invert);
    private JComboBox multiplication = new JComboBox(multiply);
```

```
    private JComboBox eROperations = new JComboBox(eRoperations);
    private JComboBox eCOperations = new JComboBox(eCoperations);
    private JComboBox systemSolve = new JComboBox(solve);

    private JRadioButtonMenuItem[] option = new
JRadioButtonMenuItem[optionNames.length];
    private ButtonGroup options = new ButtonGroup();

    /**
     * Constructs a Linear Algebra system
     */
    public CASystem() {

        Container canvas = getContentPane();

        help.setFont(f14);
        help.setBackground(Color.white);
        help.setEditable(false);

        setFont(f10);
        canvas.add(mainPane);
        mainPane.addTab("Worksheet", null, worksheetPanel, "Current
Worksheet");
        mainPane.addTab("Help", null, help, "Help");
        worksheetPanel.add(matrixPanel);
        worksheetPanel.add(optionsPanel);

        matrixPanel.add(matrixOne);
        matrixPanel.add(matrixTwo);
        matrixPanel.add(matrixThree);

        matrixOne.add(matrixOneTitle, BorderLayout.NORTH);
        matrixOne.add(input1, BorderLayout.CENTER);
        matrixOne.add(matrixOneBottom, BorderLayout.SOUTH);
            matrixTwo.add(matrixTwoTitle, BorderLayout.NORTH);
            matrixTwo.add(input2, BorderLayout.CENTER);
            matrixTwo.add(matrixTwoBottom, BorderLayout.SOUTH);
            matrixThree.add(matrixThreeTitle, BorderLayout.NORTH);
            matrixThree.add(result, BorderLayout.CENTER);
            matrixThree.add(matirxThreeBotR, BorderLayout.SOUTH);

            input1.requestFocus();
            input2.requestFocus();
            input1.setBackground(Color.white);
            input2.setBackground(Color.white);
            result.setBackground(Color.white);
            result.setEditable(false);
            input1.setEditable(editable1);
            input2.setEditable(useB);
            cols1.setEditable(editable1);
            cols2.setEditable(useB);

            matrixOneTitle.add(matrixA);
            matrixOneTitle.add(identity1);
            matrixOneTitle.add(rows1);
            matrixOneBottom.add(clear1,BorderLayout.EAST);
            matrixOneTitle.add(cols1);
            matrixTwoTitle.add(matrixB);
            matrixTwoTitle.add(identity2);
            matrixTwoTitle.add(rows2);
            matrixTwoTitle.add(cols2);
            matrixTwoBottom.add(clear2,BorderLayout.EAST);
            matrixThreeTitle.add(matrixR);
            matirxThreeBot.add(copy21);
```

```
matirxThreeBot.add(copy22);
matirxThreeBot.add(clear3);
matirxThreeBotR.add(matirxThreeBot,BorderLayout.EAST);

identity1.addItemListener(this);
identity2.addItemListener(this);
clear1.addActionListener(this);
clear2.addActionListener(this);
clear3.addActionListener(this);
copy21.addActionListener(this);
copy22.addActionListener(this);
clearAll.addActionListener(this);

optionsPanel.add(optionsBottom, BorderLayout.SOUTH);
optionsPanel.add(optionsList,BorderLayout.CENTER);

optionsBottom.add(compute);
compute.addActionListener(this);
optionsBottom.add(clearAll);

constraints.weightx = 1;
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.fill = GridBagConstraints.BOTH;
for(int i = 0; i < option.length; i++) {
    option[i] = new JRadioButtonMenuItem(optionNames[i]);
    options.add(option[i]);
    option[i].addItemListener(this);
    constraints.gridy = i;
    optionsList.add(option[i],constraints);
}

constraints.gridx = 1;

constraints.gridy = 1;
inverse.setEditable(false);
inverse.setEnabled(enabled[0]);
inverse.addItemListener(this);
optionsList.add(inverse,constraints);

scalarOpns.add(scalarLabel);
scalarOpns.add(scalar);
scalar.setEditable(enabled[1]);
constraints.gridy = 2;
optionsList.add(scalarOpns,constraints);

constraints.gridy = 5;
multiplication.setEditable(false);
multiplication.setEnabled(enabled[2]);
multiplication.addItemListener(this);
optionsList.add(multiplication,constraints);

powerOpns.add(powerLabel);
powerOpns.add(power);
power.setEditable(enabled[3]);
constraints.gridy = 6;
optionsList.add(powerOpns,constraints);

minorOpns.add(minorRowLabel);
minorOpns.add(minorRow);
minorRow.setEditable(enabled[4]);
minorOpns.add(minorColLabel);
```

```
        minorOpns.add(minorCol);
        minorCol.setEditable(enabled[4]);
        constraints.gridy = 8;
        optionsList.add(minorOpns,constraints);

        cofactorOpns.add(cofRowLabel);
        cofactorOpns.add(cofactorRow);
        cofactorRow.setEditable(enabled[5]);
        cofactorOpns.add(cofColLabel);
        cofactorOpns.add(cofactorCol);
        cofactorCol.setEditable(enabled[5]);
        constraints.gridy = 9;
        optionsList.add(cofactorOpns,constraints);

        eROperations.setEditable(false);
        eROperations.setEnabled(enabled[7]);
        eROperations.addItemListener(this);
        eroOpns.add(eROperations, BorderLayout.NORTH);
        eroRows.add(eroRow1Label);
        eroRows.add(eroRow1);
        eroRow1.setEditable(enabled[6]);
        eroRows.add(eroRow2Label);
        eroRows.add(eroRow2);
        eroRow2.setEditable(enabled[7]);
        eroRows.add(eroScalarLabel);
        eroRows.add(eroScalar);
        eroScalar.setEditable(enabled[8]);
        eroOpns.add(eroRows, BorderLayout.SOUTH);
        constraints.gridy = 12;
        optionsList.add(eroOpns,constraints);

        eCOperations.setEditable(false);
        eCOperations.setEnabled(enabled[9]);
        eCOperations.addItemListener(this);
        ecoOpns.add(eCOperations, BorderLayout.NORTH);
        ecoCols.add(ecoCol1Label);
        ecoCols.add(ecoCol1);
        ecoCol1.setEditable(enabled[12]);
        ecoCols.add(ecoCol2Label);
        ecoCols.add(ecoCol2);
        ecoCol2.setEditable(enabled[9]);
        ecoCols.add(ecoScalarLabel);
        ecoCols.add(ecoScalar);
        ecoScalar.setEditable(enabled[10]);
        ecoOpns.add(ecoCols, BorderLayout.SOUTH);
        constraints.gridy = 13;
        optionsList.add(ecoOpns,constraints);

        systemSolve.setEditable(false);
        systemSolve.setEnabled(enabled[11]);
        systemSolve.addItemListener(this);
        constraints.gridy = 14;
        optionsList.add(systemSolve,constraints);

        this.setSize(710,670);
        this.setLocation(50,45);
        this.setTitle("Linear Algebra System");
        this.show();
}

public void itemStateChanged(ItemEvent evt) {
  Object source = evt.getSource();
  for(int i = 0; i < enabled.length; i++) {
          enabled[i] = false;
```

```
    }
if (evt.getSource() == option[0]) {
    useB = false;
    input2.setText("");
    action = 1;
}
if (evt.getSource() == option[1]) {
    enabled[0] = true;
    useB = false;
    input2.setText("");
    action = 2;
}
else if (evt.getSource() == option[2]) {
    enabled[1] = true;
    useB = false;
    input2.setText("");
    action = 5;
}
else if (evt.getSource() == option[3]) {
    useB = true;
    action = 6;
}
else if (evt.getSource() == option[4]) {
    useB = true;
    action = 7;
}
else if (evt.getSource() == option[5]) {
    useB = true;
    enabled[2] = true;
    action = 8;
}
else if (evt.getSource() == option[6]) {
    useB = false;
    input2.setText("");
    action = 11;
}
else if (evt.getSource() == option[7]) {
    useB = false;
    input2.setText("");
    action = 12;
}
else if (evt.getSource() == option[8]) {
    useB = false;
    enabled[4] = true;
    input2.setText("");
    action = 13;
}
else if (evt.getSource() == option[9]) {
    useB = false;
    input2.setText("");
    enabled[5] = true;
    action = 14;
}
else if (evt.getSource() == option[10]) {
    useB = false;
    input2.setText("");
    action = 15;
}
else if (evt.getSource() == option[11]) {
    useB = true;
    input2.setText("");
    action = 16;
}
else if (evt.getSource() == option[12]) {
```

```
    useB = false;
    enabled[6] = true;
        enabled[7] = false;
      enabled[8] = true;
      action = 17;
    input2.setText("");
}
else if (evt.getSource() == option[13]) {
    useB = false;
    enabled[9] = false;
      enabled[10] = true;
      enabled[12] = true;
      action = 20;
    input2.setText("");
}
else if (evt.getSource() == option[14]) {
    useB = false;
    input2.setText("");
    enabled[11] = true;
    action = 23;
}

else if (evt.getSource() == inverse) {
    switch(inverse.getSelectedIndex()) {
    case 0:
        enabled[0] = true;
          action = 2;
          break;
    case 1:
        enabled[0] = true;
          action = 3;
          break;
    case 2:
        enabled[0] = true;
          action = 4;
          break;
    }
}

else if (evt.getSource() == multiplication) {
    switch(multiplication.getSelectedIndex()) {
    case 0:
        enabled[2] = true;
          action = 8;
          break;
    case 1:
        enabled[2] = true;
          action = 9;
          break;
    case 2:
        enabled[2] = true;
          action = 10;
          break;
    }
}

else if (evt.getSource() == eROperations) {
    switch(eROperations.getSelectedIndex()) {
    case 0:
        enabled[6] = true;
        enabled[7] = false;
          enabled[8] = true;
          action = 17;
          break;
```

```
    case 1:
        enabled[6] = true;
        enabled[7] = true;
          enabled[8] = true;
          action = 18;
          break;
    case 2:
        enabled[6] = true;
        enabled[7] = true;
          enabled[8] = false;
          action = 19;
          break;
    }
}
else if (evt.getSource() == eCOperations) {
    switch(eCOperations.getSelectedIndex()) {
    case 0:
        enabled[9] = false;
          enabled[10] = true;
          enabled[12] = true;
          action = 20;
          break;
    case 1:
        enabled[9] = true;
          enabled[10] = true;
          enabled[12] = true;
          action = 21;
          break;
    case 2:
        enabled[9] = true;
          enabled[10] = false;
          enabled[12] = true;
          action = 22;
          break;
    }
}

else if (evt.getSource() == systemSolve) {
    switch(systemSolve.getSelectedIndex()) {
    case 0:
        enabled[11] = true;
          action = 23;
          break;
    case 1:
        enabled[11] = true;
          action = 24;
          break;
    case 2:
        enabled[11] = true;
          action = 2;
          break;
    case 3:
        enabled[11] = true;
          action = 26;
          break;
    }
}
else if (evt.getSource() == identity1) {
    editable1 = !editable1;
    if(!editable1) {
        cols1.setText("");
        if (Integer.parseInt(rows1.getText()) >= 15) {
                input1.setText("The matrix is too big");
                throw new IllegalArgumentException();
```

```
            }
            IdentityMatrix identMatrix = new IdentityMatrix(
                    Integer.parseInt(rows1.getText()));
            input1.setText(identMatrix.matrixWrite());
        }
        else {
            input1.setText("");
        }
        cols1.setEditable(editable1);
            input1.setEditable(editable1);
    }
    else  {
        if (useB) {
            editable2 = !editable2;
            if(!editable2) {
                  cols1.setText("");
                  if (Integer.parseInt(rows2.getText()) >= 15) {
                    input2.setText("The matrix is too big");
                    throw new IllegalArgumentException();
                }
                    IdentityMatrix identMatrix = new IdentityMatrix(
                            Integer.parseInt(rows2.getText()));
                    input2.setText(identMatrix.matrixWrite());
              }
            else {
                  input2.setText("");
              }
              cols2.setEditable(editable2);
            input2.setEditable(editable2);
        }
        else {
            identity2.setSelected(useB);
            input2.setText("");
        }
    }
    inverse.setEnabled(enabled[0]);
    scalar.setEnabled(enabled[1]);
    multiplication.setEnabled(enabled[2]);
        power.setEditable(enabled[3]);
        minorRow.setEditable(enabled[4]);
        minorCol.setEditable(enabled[4]);
        cofactorRow.setEditable(enabled[5]);
        cofactorCol.setEditable(enabled[5]);
        eROperations.setEnabled(enabled[7]);
        eroRow1.setEditable(enabled[6]);
        eroRow2.setEditable(enabled[7]);
        eCOperations.setEnabled(enabled[9]);
        ecoCol1.setEditable(enabled[12]);
        ecoCol2.setEditable(enabled[9]);
        systemSolve.setEnabled(enabled[11]);
        eroScalar.setEditable(enabled[8]);
        ecoScalar.setEditable(enabled[10]);
        cols2.setEditable(useB);
      input2.setEditable(useB);
      scalar.setText("1");
      power.setText("2");
      minorRow.setText("1");
      minorCol.setText("1");
      cofactorRow.setText("1");
      cofactorCol.setText("1");
      eroRow1.setText("1");
      eroRow2.setText("1");
      eroScalar.setText("1");
      ecoCol1.setText("1");
```

```
     ecoCol2.setText("1");
     ecoScalar.setText("1");
 }

public void actionPerformed (ActionEvent evt) {
    if (evt.getSource() == clear1) {
        input1.setText(matrix1);
        identity1.setSelected(false);
    }
    else if (evt.getSource() == clear2) {
        input2.setText(matrix2);
        identity2.setSelected(false);
    }
    if (evt.getSource() == copy21) {
        input1.setText(result.getText());
        result.setText("");
        identity1.setSelected(false);
    }
    else if (evt.getSource() == copy22) {
        input2.setText(result.getText());
        result.setText("");
        identity2.setSelected(false);
    }
    else if (evt.getSource() == clear3) {
        result.setText(output);
    }
    else if (evt.getSource() == clearAll) {
        identity1.setSelected(false);
        identity2.setSelected(false);
        input1.setText(matrix1);
        input2.setText(matrix2);
        result.setText(output);
        rows1.setText("1");
        cols1.setText("");
        rows2.setText("1");
        cols2.setText("");
        scalar.setText("1");
        power.setText("1");
        minorRow.setText("1");
        minorCol.setText("1");
        cofactorRow.setText("1");
        cofactorCol.setText("1");
        eroRow1.setText("1");
        eroRow2.setText("1");
        eroScalar.setText("1");
        ecoCol1.setText("1");
        ecoCol2.setText("1");
        ecoScalar.setText("1");
    }

    else {
         int matrixARows = Integer.parseInt(rows1.getText());

         if (matrixARows >= 15) {
          result.setText("The matrix is too big");
          throw new IllegalArgumentException("Invalid iput size");
         }
         int matrixACols;
         if (!(cols1.getText()).equals("")) {
           matrixACols = Integer.parseInt(cols1.getText());
         }
         else {
             matrixACols = matrixARows;
         }
```

```
            double[] data1 = new double[matrixARows*matrixACols];
            String[] inputString1 = (input1.getText()).split("\\s+");
            for(int i = 0; i < matrixARows*matrixACols; i++) {
                data1[i] = Double.parseDouble(inputString1[i]);
            }
            Matrix newMatrixA = new Matrix (matrixARows, matrixACols, data1);
            if (action == 0 || action == 1 ||action == 2 ||action == 3
||action == 4 ||
                    action == 5 || action == 11 ||action == 12 || action ==
13 ||
                    action == 14 || action == 15 ||action == 17 || action ==
18 ||
                    action == 19 || action == 20 ||action == 21 ||action ==
22) {
                switch(action) {
                case 0:
                    result.setText("Action not selected");
                    break;
                case 1:
                    result.setText((newMatrixA.transpose()).matrixWrite());
                    break;
                case 2:
                    if(matrixARows <6) {
                        result.setText((newMatrixA.inverse()).matrixWrite());
                    }
                    else {

result.setText((newMatrixA.gaussElimInvert()).matrixWrite());
                    }
                    break;
                case 3:
                    result.setText((newMatrixA.inverse()).matrixWrite());
                    break;
                case 4:

result.setText((newMatrixA.gaussElimInvert()).matrixWrite());
                    break;
                case 5:
                    result.setText((newMatrixA.scalarMult(

Double.parseDouble(scalar.getText()))).matrixWrite());
                  break;
                case 11:
                    result.setText((newMatrixA.matrixExp(

Integer.parseInt(power.getText()))).matrixWrite());
                    break;
                case 12:
                    result.setText(String.valueOf(newMatrixA.det()));
                    break;
                case 13:
                    if(Integer.parseInt(minorRow.getText()) > matrixARows ||
                         Integer.parseInt(minorCol.getText()) > matrixACols )
{
                        result.setText("Index out of bounds");
                    }
                    else{
                        result.setText(String.valueOf(newMatrixA.minor(
                            Integer.parseInt(minorRow.getText())-1,
                            Integer.parseInt(minorCol.getText())-1)));
                    }
                    break;
                case 14:
                 if(Integer.parseInt(cofactorRow.getText()) > matrixARows ||
```

```
                        Integer.parseInt(cofactorCol.getText()) >
matrixACols ) {
                        result.setText("Index out of bounds");
                    }
                    else{
                        result.setText(String.valueOf(newMatrixA.cofactor(
                            Integer.parseInt(cofactorRow.getText())-1,
                            Integer.parseInt(cofactorCol.getText())-1)));
                    }
                    break;
                case 15:
                    result.setText((newMatrixA.adjugate()).matrixWrite());
                    break;
                case 17:
                 if(Integer.parseInt(eroRow1.getText()) > matrixARows) {
                        result.setText("Index out of bounds");
                    }
                    else{

result.setText((newMatrixA.multRow(Integer.parseInt(eroRow1.getText())-1,

Double.parseDouble(eroScalar.getText()))).matrixWrite());
                    }
                    break;
                case 18:
                 if(Integer.parseInt(eroRow1.getText()) > matrixARows ||
                         Integer.parseInt(eroRow2.getText()) > matrixARows )
{
                        result.setText("Index out of bounds");
                    }
                    else{

result.setText((newMatrixA.addRowMult(Integer.parseInt(eroRow1.getText())-1,
                             Integer.parseInt(eroRow2.getText())-1,

Double.parseDouble(eroScalar.getText()))).matrixWrite());
                    }
                    break;
                case 19:
                 if(Integer.parseInt(eroRow1.getText()) > matrixARows ||
                         Integer.parseInt(eroRow2.getText()) > matrixARows )
{
                        result.setText("Index out of bounds");
                    }
                    else{

result.setText((newMatrixA.swapRow(Integer.parseInt(eroRow1.getText())-1,
                            Integer.parseInt(eroRow2.getText())-
1)).matrixWrite());
                    }
                    break;
                case 20:
                 if(Integer.parseInt(ecoCol1.getText()) > matrixACols) {
                        result.setText("Index out of bounds");
                    }
                    else{

result.setText((newMatrixA.multCol(Integer.parseInt(ecoCol1.getText())-1,

Double.parseDouble(ecoScalar.getText()))).matrixWrite());
                    }
                    break;
                case 21:
                    if(Integer.parseInt(ecoCol1.getText()) > matrixACols ||
```

```
                        Integer.parseInt(ecoCol2.getText()) > matrixACols )
{
                       result.setText("Index out of bounds");
                    }
                    else{

      result.setText((newMatrixA.addColMult(Integer.parseInt(ecoCol1.getText(
))-1,
                              Integer.parseInt(ecoCol2.getText())-1,

      Double.parseDouble(ecoScalar.getText()))).matrixWrite());
                    }
                    break;
                case 22:
                 if(Integer.parseInt(ecoCol1.getText()) > matrixACols ||
                        Integer.parseInt(ecoCol2.getText()) > matrixACols )
{
                       result.setText("Index out of bounds");
                    }
                    else{

result.setText((newMatrixA.swapCol(Integer.parseInt(ecoCol1.getText())-1,
                           Integer.parseInt(ecoCol2.getText())-
1)).matrixWrite());
                    }
                    break;
                }
            }
            else {
                int matrixBRows = Integer.parseInt(rows2.getText());

                  if (matrixBRows >= 15) {
                      result.setText("The matrix is too big");
                      throw new IllegalArgumentException("Invalid iput
size");
                  }
                  int matrixBCols;
                  if (!(cols1.getText()).equals("")) {
                    matrixBCols = Integer.parseInt(cols2.getText());
                  }
                  else {
                      matrixBCols = matrixBRows;
                  }
                  double[] data2 = new double[matrixBRows*matrixBCols];
                  String[] inputString2 = (input2.getText()).split("\\s+");
                  for(int i = 0; i < matrixBRows*matrixBCols; i++) {
                      data2[i] = Double.parseDouble(inputString2[i]);
                  }
                  Matrix newMatrixB = new Matrix (matrixBRows, matrixBCols,
data2);
                switch(action) {
                case 6:

result.setText((newMatrixA.matrixAdd(newMatrixB)).matrixWrite());
                    break;
                case 7:

result.setText((newMatrixA.matrixSubtract(newMatrixB)).matrixWrite());
                    break;
                case 8:
                    if(matrixARows < 6) {

result.setText((newMatrixA.matrixMult(newMatrixB)).matrixWrite());
                    }
```

```
                    else {

result.setText((newMatrixA.matrixStrassenMult(newMatrixB)).matrixWrite());
                    }
                    break;
                case 9:

result.setText((newMatrixA.matrixMult(newMatrixB)).matrixWrite());
                    break;
                case 10:

result.setText((newMatrixA.matrixStrassenMult(newMatrixB)).matrixWrite());
                    break;
                case 16:

result.setText((newMatrixA.matrixAugment(newMatrixB)).matrixWrite());
                    break;
                case 23:
                    if(matrixARows < 6) {

result.setText((newMatrixA.cramerSolve(newMatrixB)).matrixWrite());
                    }
                    else if (matrixARows < 8) {

result.setText((newMatrixA.inverseSolve(newMatrixB)).matrixWrite());
                    }
                    else {

result.setText((newMatrixA.gaussElimSolve(newMatrixB)).matrixWrite());
                    }
                    break;
                case 24:

result.setText((newMatrixA.inverseSolve(newMatrixB)).matrixWrite());
                    break;
                case 25:

result.setText((newMatrixA.cramerSolve(newMatrixB)).matrixWrite());
                    break;
                case 26:

result.setText((newMatrixA.gaussElimSolve(newMatrixB)).matrixWrite());
                    break;
                }
            }
        }
    }

    public static void main(String[] args) {

        CASystem window = new CASystem();
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Tester.

```
/**
 * @author cs2nk
 *07.05.2005
 *
 * This Computer Linear Algebra System is designed for sumission to the
University of Bath
 * as a part of the final year project
 */
```

```
package matrix;

import junit.framework.TestCase;

/*
 * Constructs a MatrixTest object
 * @
 */
public class MatrixTest extends TestCase {

    public static void main(String[] args) {
        int n = 4;
      int m = 2;
      int k = 3;
      double[] a = {2,1,4,1,0,2,2,3,1};
      double[] b = {2,3,-6};
      double[] c = {0,1,-1,0};
      double[] d = {2,0,3,-1,-5,3,1,4};
      double[] e = {6,7,3};
      double[] f = {-1,1,0,-3,3,0,2,1};
      double[] g = {2,5,-3,-7};
      double[] a_ = {2,1,4,0,1,0,2,0,2,3,1,0,0,0,0,0};
      double[] b_ = {2,3,-6,0};
      double[][] h = new double[4][4];
        double[][] l = new double[4][4];
        double[][] o = new double[3][3];
        double[][] p = new double[3][3];

        for(int i = 0; i < k; i++) {
            for (int j = 0; j < k; j++) {
                        o[i][j] = 2*i+j;
                        p[i][j] = i+2*j;
            }
            }
        for(int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                        h[i][j] = 4*i-3*j;
                        l[i][j] = i*2+j*3;
            }
            }

        Matrix A = new Matrix(k,a);
            Matrix A_ = new Matrix(n, a_);
            Matrix B = new Matrix(k,1,b);
            Matrix B_ = new Matrix(n,1,b_);
            Matrix C = new Matrix(m,c);
            Matrix D = new Matrix(n,m,d);
            Matrix F = new Matrix(n,m,f);
            Matrix G = new Matrix(m,g);
            Matrix H = new Matrix(n,n,h);
        Matrix L = new Matrix(n,n,l);
        Matrix O = new Matrix(k,k,o);
        Matrix P = new Matrix(k,k,p);

            MatrixTest tester = new MatrixTest();

            tester.testMatrixint(n);
            tester.testMatrixintint(m,n);
            tester.testMatrixintdouble(k,2.3);
            tester.testMatrixintintdouble(m,n,5.7);
            tester.testMatrixintdoubleArray(3,a);
            tester.testMatrixintintdoubleArray(n,m,d);
            tester.testMatrixintintdoubleArrayArray(k,k,o);
            tester.testGetRows(A);
```

```
            tester.testGetColumns(D);
            tester.testGetElement(C,1,0);
            tester.testGetElements(A);
            tester.testMatrixWrite(A);
            tester.testRowsEven(A);
            tester.testRowsEven(F);
            tester.testColsEven(A);
            tester.testColsEven(F);
            tester.testIsSquare(B);
            tester.testIsSquare(A);
            tester.testIsInvertible(A);
            tester.testTranspose(A);
            tester.testScalarMult(A,m);
            tester.testMatrixAdd(D,F);
            tester.testMatrixSubtract(D,F);
            tester.testMatrixMult(D,C);
            tester.testMatrixStrassenMult(H,L);
            tester.testMatrixStrassenMult(O,P);
            tester.testMatrixExp(A);
            tester.testMatrixExpint(A,3);
            tester.testMatrixAugment(A);
            tester.testMatrixAugmentMatrix(A,B);
            tester.testMatrixVertAugment(A,O);
            tester.testMatrixAugmentVectorMatrix(A,B);
            tester.testMatrixAugmentVector(A,b);
            tester.testMultRow(A,m,m);
            tester.testAddRowMult(A,0,2,2);
            tester.testSwapRow(A,0,2);
            tester.testMultCol(A,m,m);
            tester.testAddColMult(A,0,2,2);
            tester.testSwapCol(A,0,2);
            tester.testDeleteRowCol(A,0,2);
            tester.testDet(A);
            tester.testMinor(G,0,1);
            tester.testCofactor(G,1,1);
            tester.testAdjugate(A);
            tester.testInverse(A);
            tester.testGaussElimInvert(A);
            tester.testCramerSolve(A,B);
            tester.testInverseSolve(A,B);
            tester.testGaussElimSolve(A,B);
    }



    /**
     * Constructor for MatrixTest.
     * @param
     */
    public MatrixTest() {
    }

      public void testMatrixint(int n) {
            System.out.println(" \n \t Creating an empty square matrix of
size "+n+" by "+n+"\n \t");
            Matrix matrix = new Matrix(n);
            System.out.println(matrix.matrixWrite());
            }

      public void testMatrixintint(int n, int m) {
            System.out.println("\n \t Creating an empty rectangular matrix of
size "+n+" by "+m+"\n \t");
            Matrix matrix = new Matrix(n,m);
            System.out.println(matrix.matrixWrite());
```

```
      }

    public final void testMatrixintdouble(int n, double d) {
        System.out.println("\n \t Creating a matrix of sise " +
                  + n + " by " + n + " out of a double " + d + " \n \t");
            Matrix matrix = new Matrix(n,d);
            System.out.println(matrix.matrixWrite());
    }

    public final void testMatrixintintdouble(int n, int m, double d) {
        System.out.println("\n \t Creating a matrix of sise " +
                  + n + " by " + m + " out of a double " + d + " \n \t");
            Matrix matrix = new Matrix(n,m,d);
            System.out.println(matrix.matrixWrite());
    }


      public void testMatrixintdoubleArray(int n, double[] array) {
            System.out.println("\n \t Creating a square matrix of size "+n+
                    " by "+n+" given an array of doubles"+"\n \t");
            Matrix matrix = new Matrix(n, array);
            System.out.println(matrix.matrixWrite());
      }

      public void testMatrixintintdoubleArray(int n, int m, double[] array) {
            System.out.println("\n \t Creating a rectangular matrix of size
"+n+
                    " by "+m+" given an array of doubles"+"\n \t");
            Matrix matrix = new Matrix(n,m, array);
            System.out.println(matrix.matrixWrite());
      }

      public void testMatrixintintdoubleArrayArray(int n, int m, double[][]
array) {
            System.out.println("\n \t Creating a rectangular matrix of size
"+n+
                    " by "+m+" given a 2D array of doubles"+"\n \t");
            Matrix matrix = new Matrix(n,m, array);
            System.out.println(matrix.matrixWrite());
      }

      public void testGetRows(Matrix matrix) {
            System.out.println("\n \t Testing method getRows \n");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t" + matrix.getRows()+ " rows \n");

      }

      public void testGetColumns(Matrix matrix) {
            System.out.println("\n \t Testing method getColumns \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t" + matrix.getColumns()+ " columns \n");
      }

      public void testGetElement(Matrix matrix, int row, int column) {
            System.out.println("\n \t Testing method getElement. \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Element index
"+"["+(row+1)+"]["+(column+1)+
                    "] is "+matrix.getElement(row, column)+"\n \t");
      }

      public void testGetElements(Matrix matrix) {
            System.out.println("\n \t Testing method getElements \n \t");
```

```
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Elements \n \t");
            double[][] elements = matrix.getElements();
            for(int i = 0; i < matrix.getRows(); i++) {
            for (int j = 0; j < matrix.getColumns(); j++") {
                        System.out.print(elements[i][j]+"\t");
            }
            System.out.print("\n");
            }
      }

      public void testMatrixWrite(Matrix matrix) {
          double[][] elements = matrix.getElements();
            for(int i = 0; i < matrix.getRows(); i++) {
            for (int j = 0; j < matrix.getColumns(); j++) {
                        System.out.print(elements[i][j]+"\t");
            }
            System.out.print("\n");
            }
        System.out.println("\n \t Testing method"+"\n \t");
        System.out.println(matrix.matrixWrite());
    }

      public void testRowsEven(Matrix matrix) {
        System.out.println("\n \t Checking if number of rows is even \n \t
");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Is the number of rows even? \n \t ");
            if(matrix.rowsEven()) {
                System.out.println("Yes");
            }
            else {
                System.out.println("No");
            }
    }

      public void testColsEven(Matrix matrix) {
        System.out.println("\n \t Checking if number of columns is even \n \t
");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Is the number of columns even? \n \t
");
            if(matrix.colsEven()) {
                System.out.println("Yes");
            }
            else {
                System.out.println("No");
            }
    }

      public void testIsSquare(Matrix matrix) {
            System.out.println("\n \t Testing shape \n \t");
            System.out.println(matrix.matrixWrite());
            if(matrix.isSquare()) {
                  System.out.println("Square matrix \n \t");
            }
            else {
                  System.out.println("Non-square matrix \n \t");
            }
      }

      public void testIsInvertible(Matrix matrix) {
            System.out.println("\n \t Invertability test \n \t");
            System.out.println(matrix.matrixWrite());
```

```
            if(matrix.isInvertible()) {
                  System.out.println("\n \t Non-singular (invertible) matrix
\n \t");
            }
            else {
                  System.out.println("\n \t Singular matrix \n \t");
            }
      }

      public void testTranspose(Matrix matrix) {
            System.out.println("\n \t Testing transposition \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Transposed: \n \t");
            Matrix transpose = matrix.transpose();
            System.out.println(transpose.matrixWrite());
      }

      public void testScalarMult(Matrix matrix, int scalar) {
            System.out.println(" \n \t Testing scalar multiplicaiton. \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Testing scalar multiplicaiton.
Multiplying by " +
                    scalar + "\n \t");
            Matrix mult = matrix.scalarMult(scalar);
            System.out.println(mult.matrixWrite());
      }

      public void testMatrixAdd(Matrix A, Matrix B) {
            System.out.println(" \n \t Testing matrix addition. \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println(" \n \t Adding matrices: \n \t");
            Matrix C = A.matrixAdd(B);
            System.out.println(C.matrixWrite());
      }

      public void testMatrixSubtract(Matrix A, Matrix B) {
            System.out.println(" \n \t Testing matrix subtraction. \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println(" \n \t Subtracting matrices \n \t");
            Matrix C = A.matrixSubtract(B);
            System.out.println(C.matrixWrite());
      }

      public void testMatrixMult(Matrix A, Matrix B) {
            System.out.println("\n \t Testing matrix multiplication. \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println("\n \t Multiplying matrices \n \t");
            Matrix C = A.matrixMult(B);
            System.out.println(C.matrixWrite());
      }

      public void testMatrixStrassenMult(Matrix A, Matrix B) {
        System.out.println("\n \t Testing Strassen matrix multiplication. \n
\t");
        System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
```

```
            System.out.println("\n \t Multiplying matrices \n \t");
            Matrix C = A.matrixStrassenMult(B);
            System.out.println(C.matrixWrite());
    }

      public void testMatrixExp(Matrix matrix) {
            System.out.println("\n \t Testing matrix exponentiation to the
power of two." +
                        " \n \t ");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Squared matrix: \n \t ");
            Matrix exp = matrix.matrixExp();
            System.out.println(exp.matrixWrite());
      }


      public void testMatrixExpint(Matrix matrix, int power) {
            System.out.println("\n \t Testing matrix exponentiation to the
power of two. " +
                        "\n \t ");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Matrix to the power of " + power+"\n
\t");
            Matrix exp = matrix.matrixExp(power);
            System.out.println(exp.matrixWrite());
      }

      public void testMatrixAugment(Matrix matrix) {
            System.out.println("\n \t Creating augmented matrix (with
identity matrix) " +
                        "\n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Augmented matrix: \n \t");
            Matrix augmented = matrix.matrixAugment();
            System.out.println(augmented.matrixWrite());
      }

      public void testMatrixAugmentVector(Matrix matrix, double[] vector) {
            System.out.println("\n \t Creating augmented matrix with a vector
\n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println(" \n \t ");
            for(int i = 0; i < vector.length; i++) {
                  System.out.println(vector[i]);
            }
            System.out.println("\n \t Augmented matrix: \n \t");
            Matrix augmented = matrix.matrixAugmentVector(vector);
            System.out.println(augmented.matrixWrite());
      }

      public void testMatrixAugmentVectorMatrix(Matrix matrix, Matrix vector)
{
            System.out.println("\n \t Creating augmented matrix with a column
vector " +
                        "\n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println(" \n \t ");
            for(int i = 0; i < vector.getRows(); i++) {
                  System.out.println(vector.getElement(i,0));
            }
            System.out.println("\n \t Augmented matrix: \n \t");
            Matrix augmented = matrix.matrixAugmentVector(vector);
            System.out.println(augmented.matrixWrite());
      }
```

```
    public void testMatrixAugmentMatrix(Matrix A, Matrix B) {

        System.out.println("\n \t Creating augmented matrix. \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println("\n \t Augmented \n \t");
            Matrix C = A.matrixAugment(B);
            System.out.println(C.matrixWrite());

    }

    public void testMatrixVertAugment(Matrix A, Matrix B) {
        System.out.println("\n \t Creating augmented matrix. \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println("\n \t Augmented \n \t");
            Matrix C = A.matrixVertAugment(B);
            System.out.println(C.matrixWrite());
    }

    public void testMultRow(Matrix matrix, int row, double scalar) {
            System.out.println("\n \t Testing elementrary row operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Multiplying row "+(row+1)+" by
"+scalar+"\n \t");
            Matrix mult = matrix.multRow(row,scalar);
            System.out.println(mult.matrixWrite());
      }

      public void testAddRowMult(Matrix matrix, int row1, int row2, double
scalar) {
            System.out.println("\n \t Testing elementrary row operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Row"+(row1+1)+" +
"+scalar+"*row"+(row2+1)+"\n \t");
            Matrix addmult = matrix.addRowMult(row1,row2,scalar);
            System.out.println(addmult.matrixWrite());
      }

      public void testSwapRow(Matrix matrix, int row1, int row2) {
            System.out.println("\n \t Testing elementrary row operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Swap row "+(row1+1)+" and row
"+(row2+1)+"\n \t");
            Matrix swap = matrix.swapRow(row1,row2);
            System.out.println(swap.matrixWrite());
      }

      public void testMultCol(Matrix matrix, int col, double scalar) {
            System.out.println("\n \t Testing elementrary column operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Multiplying column "+(col+1)+" by
"+scalar+"\n \t");
            Matrix mult = matrix.multCol(col,scalar);
            System.out.println(mult.matrixWrite());
      }
```

```
      public void testAddColMult(Matrix matrix, int col1, int col2, double
scalar) {
            System.out.println("\n \t Testing elementrary column operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Column"+(col1+1)+" +
"+scalar+"*column"+(col2+1)+
                  "\n \t");
            Matrix addmult = matrix.addColMult(col1,col2,scalar);
            System.out.println(addmult.matrixWrite());;
      }

      public void testSwapCol(Matrix matrix, int col1, int col2) {
            System.out.println("\n \t Testing elementrary column operatons \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Swap column "+(col1+1)+" and column
"+(col2+1)+
                  "\n \t");
            Matrix swap = matrix.swapCol(col1,col2);
            System.out.println(swap.matrixWrite());
      }

      public void testDeleteRowCol(Matrix matrix, int row, int col) {
            System.out.println("\n \t Eliminating a row and a column \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Eliminate row "+(row+1)+" and column
"+(col+1)+"\n \t");
            Matrix elim = matrix.deleteRowCol(row,col);
            System.out.println(elim.matrixWrite());
      }

      public void testDet(Matrix matrix) {
            System.out.println(" \n \t Computing determinant \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Determinant eqials " + matrix.det() +
"\n \t");
      }

      public void testMinor(Matrix matrix, int row, int col) {
            System.out.println(" \n \t Computing minor \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println((row+1)+"-"+(col+1)+" minor of the matrix
eqials "
                  + matrix.minor(row,col) + "\n \t");
      }

      public void testCofactor(Matrix matrix, int row, int col) {
            System.out.println(" \n \t Computing cofactor \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println((row+1)+"-"+(col+1)+" cofactor of the matrix
eqials "
                  + matrix.cofactor(row,col) + "\n \t");
      }

      public void testAdjugate(Matrix matrix) {
            System.out.println("\n \t Computing adjugate of the matrix \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Adjugate of the matrix \n \t");
            Matrix adjugate = matrix.adjugate();
            System.out.println(adjugate.matrixWrite());
      }
```

```
      public void testInverse(Matrix matrix) {
            System.out.println("\n \t Computing inverse of the matrix \n
\t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Inverse of the matrix \n \t");
            Matrix inverse = matrix.inverse();
            System.out.println(inverse.matrixWrite());
      }

      public final void testGaussElimInvert(Matrix matrix) {
            System.out.println("\n \t Find the inverse \n \t");
            System.out.println(matrix.matrixWrite());
            System.out.println("\n \t Inverse using Gaussian Elimination \n
\t");
            Matrix gauss = matrix.gaussElimInvert();
            System.out.println(gauss.matrixWrite());
    }

      public void testCramerSolve(Matrix A, Matrix B) {
            System.out.println("\n \t Solve the system \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println(" \n \t \t using Cramer's rule \n \t");
            Matrix X = A.cramerSolve(B);
            System.out.println(X.matrixWrite());
      }

      public void testInverseSolve(Matrix A, Matrix B) {
            System.out.println("\n \t Solve the system \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println(" \n \t \t using inverse \n \t");
            Matrix X = A.inverseSolve(B);
            System.out.println(X.matrixWrite());
      }

      public final void testGaussElimSolve(Matrix A, Matrix B) {
        System.out.println("\n \t Solve the system \n \t");
            System.out.println(A.matrixWrite());
            System.out.println(" \n \t ");
            System.out.println(B.matrixWrite());
            System.out.println(" \n \t \t using Gaussian Elimination \n \t");
            Matrix gauss = A.gaussElimSolve(B);
            System.out.println(gauss.matrixWrite());

    }

}
```