Computer Linear Algebra System

Author: Xiaowei Xu
Computer Science Department

Abstract

The aim of this project is to design a basic linear algebra system as a very simplified analogy of usual computer algebra systems, like REDUCE, Maple etc. However, computer algebra systems have covered a broad area of mathematical problems. The main purpose of the dissertation is to produce computer linear algebra system which is a dedicated system and only capable of solving linear algebra problems. The operations include: matrix addition, matrix subtraction, matrix scalar, matrix multiplication, matrix transpose, computing 3 by 3 matrix determinant, computing matrix determinant, computing trace of matrix, computing matrix inverse, matrix ad-joint, and two more operations for solving linear equations which are Cramer's rule and Gauss-Jordan elimination. Those algorithms concerned have been developed by using Java codes.

Acknowledgements

Many thanks are to my supervisor Dr Nicolai Vorobjov for helping me so much. Also, I would like to thank my parents for encouraging and supporting me to finish this dissertation.

Computer Linear Algebra System

Submitted by Xiaowei Xu

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor 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 specifically acknowledged, it is the work of the author.

Signed……..

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purpose of consultation.

Signed……..

Contents

1. Introduction.....9

1.1 Brief Introduction.....9
1.2 Is it worth doing it?.....9
1.3 Structure of the dissertation.....9

2. Literature Survey.....10

2.1 Linear algebra mathematical background.....10
2.1.1 Matrix.....10
2.1.2 Identity Matrix.....11
2.1.3 Square Matrix.....11
2.1.4 Diagonal Matrix.....11
2.1.5 Matrix Addition.....11
2.1.6 Matrix Subtraction.....12
2.1.7 Scalar multiplication of matrices.....12
2.1.8 Multiplication of Matrices.....13
2.1.9 Transpose of Matrices.....14
2.1.10 Trace of Matrices.....14
2.1.11 Invertible Matrices.....14
2.1.12 (3 by 3) Determinant of matrices.....15
2.1.13 General Determinant of Matrices.....15
2.1.14 Ad-joint of matrices.....16
2.1.15 Determinant and Inverse.....16
2.1.16 Matrices and systems of linear equations.....16
2.1.17 Cramer's Rule.....19
2.2 Existing System Analysis.....20
2.2.1 Linear algebra system named Java Matrix Calculator.....20
2.2.2 Linear algebra system in Maple.....21
2.2.3 Previous Master Student's Project on Linear algebra system.....22

3. Requirements Analysis.....23

3.1 Introduction.....23
3.2 Use Cases.....23
3.2.1 Scope.....23
3.3 Main Goals of Requirements.....28
3.3.1 Functionalities should be involved in the system:.....28
3.3.2 Efficient and fast algorithms.....28
3.3.3 System response to incorrect data input.....28
3.3.4 Users of the system.....28
3.4 System constraints.....29
3.4.1 Requirements of programming language.....29
3.4.2 Requirements of operating system.....29
3.4.3 Requirements of system response time.....29
3.5 System interface constraints.....29
3.5.1 Layout.....29

3.5.2 User friendly 29
3.5.3 Error messages 29
3.5.4 Help function 29
3.5.5 Operations in a menu form 30
3.5.6 Current state will be informed 30
3.6 Requirements of data input 30
3.7 Requirements of security 30
3.8 Requirements of documentation 30
3.9 Requirements of system testing 31
3.10 Requirements of training 31
3.11 Project constraints 31
3.11.1 Deadline 31
3.11.2 Working hours 31

4. Requirements Specification 32

4.1 Introduction 32
4.2 Functional Requirements specification 32
4.2.1 Requirements for Matrix Arithmetic 32
4.2.2 Requirements for Linear equations 35
4.2.3 Requirements for other functions 35
4.3 Hardware and Software Requirements Specification 36
4.4 Non-functional Requirements Specification 37
4.5 Miscellaneous requirements 39
4.6 System Scope 41

5. System Design 42

5.1 Introduction 42
5.2 System Model 42
5.2.1 Introduction 42
5.2.2 Architectural Model 42
5.2.3 Data Flows 43
5.3 Software Model 50
5.3.1 Introduction 50
5.3.2 CRC Modelling 50
5.3.3 Components and Classes 51
5.3 User Interface Design 54
5.4.1 Introduction 54
5.4.2 User Interface Design Principles 54
5.4.3 Interface Designs 55

6. Detailed Design and Implementation 60

6.1 Introduction 60
6.2 Technologies 61
6.3 Formal specification of Matrix operations 61
6.4 Algorithm of Matrix arithmetic 63
6.4.1 Matrix Addition 63

6.4.2 Matrix Subtraction 64
6.4.3 Matrix Multiplication 65
6.4.4 Matrix Scalar 66
6.4.5 Matrix Transpose 66
6.4.6 Matrix Trace 67
6.4.7 Matrix Determinant 3 by 3 Matrices only 68
6.4.8 Matrix Determinant 69
6.4.9 Matrix Adjoint 71
6.4.10 Matrix Inverse 72
6.4.11Cramer's Rule 73
6.4.12 Gauss-Jordan Elimination 74
6.5 Implementation of Interface 77
6.5.1 How to read users' input from the interface 77

7. System Testing 78

7.1 Introduction 78
7.2 Testing Process 78
7.2.1 Unit Testing 78
7.2.2 Integration Testing 78
7.2.3 System Testing 78
7.3 Validation of Requirements 79
7.3.1 Functional Requirements Testing 79
7.4 Unit Testing 81
7.4.1 Black Box Testing 81
7.5 White Box Testing 83
7.6 Integration testing 84
7.7 System Testing 84
7.7.1 Matrix Addition Testing 84
7.7.2 Matrix Subtraction Testing 85
7.7.3 Matrix Scalar Testing 86
7.7.4 Matrix Multiplication Testing 87
7.7.5 Matrix Transpose Testing 88
7.7.6 Matrix Trace Testing 88
7.7.7 Matrix Determinant Testing 89
7.7.8 Matrix Determinant 3 by 3 Testing 89
7.7.9 Matrix Inverse Testing 90
7.7.10 Matrix Ad-joint Testing 90
7.7.11 Cramer's Rule 90
7.7.12 Gauss-Jordan Elimination 92
7.7.13 Save Function 93
7.7.14 Load Function 93
7.8 Conclusion of Testing 93

8. Critical Evaluation 94

8.1 Introduction 94
8.2 Evaluation 94
8.2.1 What has the system fulfilled? 94

8.2.2 Improvement of the system....94
8.3 Conclusion of evaluation....94

9. Conclusion and Future Work....95

9.1 Conclusion....95
9.2 Future Work on the system....95
9.2.1 Algorithms improvement....95
9.2.2 Complexity of Algorithms....96
9.2.3 Improvement of Other Functions....97
9.2.4 Function improvement....97
9.2.5 Improvement of System Interface....97

10. Bibliography....98

11. Appendices....100

11.1 A Simple User Guide....100
11.1.1 Functional Description....100
11.1.2 Installation Guide....100
11.1.3 User Manual....101
11.2 Code....111

1. Introduction

1.1 Brief Introduction

This project aims to design a basic linear algebra system as a very simplified analogy of usual computer algebra systems, like REDUCE, Maple, etc. The system should have convenient and interactive user-interface, and the design of the interface should fully follow the user interface principles, and it is based on the linear algebra mathematical background. The system could be used for computing matrix addition, matrix subtraction, matrix scalar, matrix multiplication, matrix transpose, matrix trace, matrix determinant, matrix 3 by 3 determinants, matrix inverse, matrix ad-joint, Cramer's Rule and Gauss-Jordan Elimination.

1.2 Is it worth doing it?

Currently, there are quite lot existing computer linear algebra systems. However, there are some aspects of existing systems should be improved in this project. In order to achieve a higher level in this area, some existing systems will be involved in this project for improvement purpose. In next chapter, it is going to reveal the constraints of existing systems and detail how to improve them.

1.3 Structure of the dissertation

This dissertation will include literature survey, requirement analysis, requirement specification, system design, detailed design and implementation, system testing, critical evaluation, conclusion and future work on the system.

2. Literature Survey

This project is going to design computer linear algebra system which is fully based matrix mathematical background. The background of math element which will be involved in the system should be explained precisely. The background will give users a clear clue how the internal of system works what is fully based on the math algorithms. This chapter will cover two major parts, mathematical background and existing system of computer linear algebra system.

2.1 Linear algebra mathematical background

Before designing the computer linear algebra system, the linear algebra concepts and definitions should be established and known that will be developed in computer linear algebra system. It is because linear algebra system is fully based on linear algebra mathematics. This section will illustrate the concepts of linear algebra and those will be developed.

2.1.1 Matrix

By the definition, a matrix is a rectangular array of numbers. More precisely, a matrix is with m rows and n columns, called an m × n matrix in a rectangular array of numbers.

An example of a matrix: $\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$

Let's call this matrix A. How to call each element in this matrix? For example, we would like to pick a_{ij} from this matrix, a_{ij} means this element is in the row i and column j. In a specific way, pick a_{23} from the matrix A, this means the element is in the second row and third column. It is noted that the element a_{ij} is called ij-entry or ij-component that appears in the row i_{th} and column j_{th}. A matrix with m rows and n columns is called an m by n matrix. The pair of numbers $(m \quad n)$ is called its size or shape. (Seymour Lopschutz, 1974)

To explain this idea more clearly, an illustration should be taken. A 1 by 3 matrix means it has1 row and 3 columns. It looks like: $(3 \quad 2 \quad 1)$. And a 2 by 3 matrix means it has 2 rows and 3 columns, it looks like: $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$

Its rows are $(1 \quad 2 \quad 3)$ and $(3 \quad 2 \quad 1)$; its columns are: $\begin{pmatrix} 1 \\ 3 \end{pmatrix}$, $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$ and $\begin{pmatrix} 3 \\ 1 \end{pmatrix}$.

Two matrices are equal if they have the same size that means same number of rows and columns and their corresponding elements should be equal as well.
A matrix is with one row can be referred as a row vector, and a matrix is with one column can referred as a column vector.

2.1.2 Identity Matrix

An identity matrix is a diagonal matrix of size n by n and its main diagonal is 1 and all other elements are 0. For example, 3 by 3 identity matrix should look like:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.1.3 Square Matrix

A matrix is a square matrix that its horizontal and vertical dimensions are the same (rows and columns are equal) i.e. an n by n matrix. It should look like:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

2.1.4 Diagonal Matrix

A diagonal matrix is a square matrix that its main diagonal should be non zeros and other elements are all zeros. It should look like:

$$\begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

2.1.5 Matrix Addition

Define two matrices A and B, the addition is successful when matrices A and B have the same size, which means same number of rows and columns. For example,

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad \mathrm{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

The sum of matrices A and B, it is by adding the corresponding elements:

$$A + B = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{pmatrix}$$

(Seymour Lipschutz, 1974)

Taking a specific example should be more intuitive:

$$\text{Let } A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad B = \begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix}$$

The sum of matrices and A and B, also it is by adding the corresponding elements:

$$A + B = \begin{pmatrix} 1+10 & 2+11 & 3+12 \\ 4+13 & 5+14 & 6+15 \\ 7+16 & 8+17 & 9+18 \end{pmatrix}$$

Also, taking a specific example which two matrices cannot be able to add together:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

The size of matrix A is 3 by 3 and the size of matrix B is 3 by 4. This time two matrices cannot be by adding the corresponding elements.

2.1.6 Matrix Subtraction

Before define the matrix subtraction, we first have an example.

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 4 & 1 & 2 \end{pmatrix} \qquad -B = \begin{pmatrix} -2 & -1 & -3 \\ -4 & -1 & -2 \end{pmatrix}$$

So we can give the definition of matrix subtraction in such a way that makes it compatible with addition. (Gareth Williams, 2001) In this case, we can subtract matrices A and B which should be A- B by converting it into A+ (-B). As well as matrix addition, matrix subtraction is successful if A and B have the same size, which means same number of rows and columns. Otherwise, it fails.

2.1.7 Scalar multiplication of matrices

Let c be a scalar and matrix A. The product of cA, which means each element in matrix A, denoted a_{ij} should multiply c. the matrix cA will be the same size as A.

$$cA = \begin{pmatrix} ca_{11} & ca_{12} & \cdots & ca_{1n} \\ ca_{21} & ca_{22} & \cdots & ca_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ ca_{m1} & ca_{m2} & \cdots & ca_{mn} \end{pmatrix}$$

2.1.8 Multiplication of Matrices

This section is to show multiplication of matrices of A and B. First, let us consider a row matrix and a column matrix. So the product of A and B:

$$\mathrm{A} \times \mathrm{B} = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

If the number of columns in matrix A does not equal the number of rows in B, we say that the product does not exist. (Gareth Williams, 2001)

The definition of multiplication of matrices is that let number of rows of matrix A is equal to the number of columns of matrix B. Only in this case, the product exists. In (Gareth Williams, 2001), it says the element in row i and column j of AB is obtained by multiplying the corresponding elements of row i of A and column j of B and adding the products.

In general, it could be represented like this:

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad \mathrm{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{pmatrix}$$

$$\mathrm{A} \times \mathrm{B} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1n}b_{n1} & \cdots & \cdots & a_{11}b_{1m} + a_{12}b_{2m} + \cdots + a_{1n}b_{nm} \\ a_{21}b_{11} + a_{22}b_{21} + \cdots + a_{2n}b_{n1} & \cdots & \cdots & a_{21}b_{1m} + a_{22}b_{2m} + \cdots + a_{2n}b_{nm} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1}b_{11} + a_{m2}b_{21} + \cdots + a_{mn}b_{n1} & \cdots & \cdots & a_{m1}b_{1m} + a_{m2}b_{2m} + \cdots + a_{mn}b_{nm} \end{pmatrix}$$

Take a specific example:

$$\text{Let } \mathrm{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad \mathrm{B} = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

$$\mathrm{A} \times \mathrm{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

$$= \begin{pmatrix} 1\times5+2\times8 & 1\times6+2\times9 & 1\times7+2\times10 \\ 3\times5+4\times8 & 3\times6+4\times9 & 3\times7+4\times10 \end{pmatrix}$$

$$= \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

It should be noted that matrix multiplication is not commutative, i.e. the products AB and BA of matrices need not be equal.

2.1.9 Transpose of Matrices

The transpose of a matrix A is written by A^t, which is of an m×n matrix and A is the n×m matrix obtained by interchanging the rows and columns of A. So if A = $[a_{ij}]_{m\times n}$ then $A^t = [a_{ji}]_{n\times m}$. (David Towers, 1988)

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \quad \text{then} \quad A^t = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \cdots & \cdots & \cdots & \cdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

2.1.10 Trace of Matrices

The trace of a square matrix (n by n matrix) is the sum of the diagonal elements of A. It is denoted as tr(A). So tr(A) = $a_{11} + a_{22} + \ldots\ldots + a_{nn}$ (Gareth Williams, 2001)

$$A = \begin{pmatrix} 4 & 43 & 54 \\ 34 & -5 & 65 \\ 78 & 53 & 0 \end{pmatrix} \quad \text{then} \quad tr(A) = 4 + (-5) + 0 = 1$$

2.1.11 Invertible Matrices

A square matrix A is said to be invertible if there is a matrix B such that AB = BA = I, here I is identity matrix. In this case, A is called invertible and B is called an inverse of A. It is obvious that B should be square as well: in fact B has the same size as A. Matrix B is the inverse matrix of A is denoted as A^{-1}. Apparently, if matrix B is the inverse of matrix A, and matrix A is the inverse of matrix B too. The inverse of a matrix A (if exists) is unique. (David towers. 1988 and Seymour Lipschutz. 1974) One example is illustrated below:

$$A = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix} \qquad B = \begin{pmatrix} 3 & -5 \\ -1 & 2 \end{pmatrix}$$

$$A \times B = \begin{pmatrix} 2\times 3 + 5\times(-1) & 2\times(-5) + 5\times 2 \\ 1\times 3 + 3\times(-1) & 1\times(-5) + 3\times 2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\text{And } B \times A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(Seymour Lipschutz. 1974)

2.1.12 (3 by 3) Determinant of matrices

3 by 3 determinant of matrices is using another algorithm method to gain the result. For example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$ then determinant of A:

$$|A| = a_{11}\, a_{22}\, a_{33} + a_{12} a_{23} a_{31} + a_{13} a_{21} a_{32} - a_{13} a_{22} a_{31} - a_{11} a_{23} a_{32} - a_{12} a_{21} a_{33}$$

2.1.13 General Determinant of Matrices

Let us consider the 2 by 2 matrix first. Let $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ be a 2 by 2 matrix.

The determinant of A is denoted by |A| and should be

$$|A| = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}\, a_{22} - a_{12}\, a_{21}$$

There is one thing should be emphasised, |A| here denotes the determinant of matrix A, and |A| denotes the absolute value of A if A is real number or complex number.

The definition of determinant of 3 by 3 matrix let $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$

$$\det A = |A| = a_{11} \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} - a_{12} \begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix} + a_{13} \begin{pmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

(Stanley l.Grossman. 1991)

We have to incorporate some more concepts in order to define the determinant of large matrices.

Definitions of **Minor** and **cofactor** will be required. Let A be a square matrix.

Minor: The minor of the element a_{ij} is denoted by M_{ij}, and the determinant of the matrix is the remaining by deleting the row i and column j of A.

Cofactor: The cofactor of a_{ij} is denoted C_{ij}, and given by $C_{ij} = (-1)^{i+j} M_{ij}$. The minor and cofactor differ in at most sign. $C_{ij} = \pm M_{ij}$. (Gareth Williams, 2001)

The determinant of matrix $A = a_{ij}$ is equal to the sum of the products obtained by multiplying the elements of any row (column) by their respective cofactors:

i_{th} row expansion $|A| = a_{i1} C_{i1} + a_{i2} C_{i2} + \ldots\ldots + a_{in} C_{in}$

j_{th} column expansion $|A| = a_{1j} C_{1j} + a_{2j} C_{2j} + \ldots\ldots + a_{nj} C_{nj}$

(Seymour Lipschutz 1974)

Details exposition of determinant can be found in (Stanley l.Grossman. 1991, Seymour Lipschutz. 1974, Gareth Williams 2001, David Towers 1988 and determinant available from: http://mathworld.wolfram.com/Determinant.html)

2.1.14 Ad-joint of matrices

Definition of Ad-joint of matrices is: Let A be a square matrix and C_{ij} be the cofactor of a_{ij}. The matrix whose (i, j) th element is C_{ij} is called the matrix of cofactors of A. The transpose of this matrix is called the ad-joint of A is denoted adj (A).

2.1.15 Determinant and Inverse

A square matrix A is defined to non-singular if the determinant of A is non zero and singular if the determinant of A is zero.
Only non-singular matrix is invertible. Once you know the matrix is non-singular then its inverse exist, otherwise no inverse.
The definition of inverse of square matrix is: Let A be a square matrix with |A| ≠ 0. A is invertible with $A^{-1} = \frac{1}{|A|}\text{adj(A)}$ (Gareth Williams 2001)

2.1.16 Matrices and systems of linear equations

By using matrices is to describe systems of linear equations. There are two important matrices involved with every system of linear equations. One matrix is called matrix of coefficients, the other one is called augmented matrix. There is an example below to show what coefficient matrix and augmented matrix are.

$2x_1 + 4x_2 + 4x_3 = 18$

$4x_1 + 5x_2 + 6x_3 = 24 \quad \Rightarrow \mathbf{CM} \begin{pmatrix} 2 & 4 & 4 \\ 4 & 5 & 6 \\ 3 & 1 & -2 \end{pmatrix} \Rightarrow \mathbf{AM} \begin{pmatrix} 2 & 4 & 4 & 18 \\ 4 & 5 & 6 & 24 \\ 3 & 1 & -2 & 4 \end{pmatrix}$

$3x_1 + x_2 - 2x_3 = 4$

Here **CM** and **AM** stand for coefficients matrix and augmented matrix respectively.

Elementary Row Operations

Elementary transformations can be used to change a linear equation system into another linear equation system but has the same solution. This transformation is based on eliminating variables. In terms of matrices using equivalent transformation is called elementary row operations. In this system, variables such as x_1, x_2 and x_3 need not be written down at each stage. Systems of linear equations are often written into and calculated in terms of matrices.

Elementary Transformation	**Elementary Row Operations**
1. Interchange two equations	1. Interchange two rows of a matrix
2. Multiply both sides of an equation by a non-zero constant.	2. Multiply the elements of a row by a non-zero constant

3. Add a multiple of one equation to Another equation
(Gareth Williams. 2001)

3. Add a multiple of the elements of one row to the corresponding elements of another row
(Gareth Williams. 2001)

Reduced Echelon Form

The augmented matrix is made up of a matrix of coefficients called A and a column matrix of constant called B. We write this into [A: B]. after elementary row operations, matrices will be into reduced row echelon form that is [A: B] = [I_n : X].

If [A: B] cannot be transformed in this [I_n : X], the system of equations does not have a unique solution.

A matrix A is said in reduced echelon form, if it satisfies following conditions:

(1) Every row of A which contains at least one nonzero entry, has as its first (from left to right) nonzero entry a 1. It is convention to call these 1's referred to above as the leading 1's of the rows in which they occur.
(2) Each column of A which contains a leading 1 has all other entries equal to zero. The first nonzero number in a row (if any) is called a pivot for the row.
(3) In any two nonzero rows of A, the leading 1 from the lower row must occur farther to the right than the leading 1 from the upper row.
(4) All rows of A which consist entirely of zeros are placed at the "bottom" of the matrix. (Unit 14 Matrices and Systems of Linear Equations)

$$A = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 14 \\ 0 & 1 & 0 & 25 \\ 0 & 0 & 1 & 36 \end{array}\right)$$ Matrix A is in the reduced echelon form.

Gauss-Jordan Elimination

1. Transform the system of linear equations into augmented matrix in terms of matrix.
2. Derive the reduced echelon form of the augmented matrix using elementary row operations. This is done by creating leading 1s, then zeros above and below each leading 1, column by column starting with the first column.
3. Write down the system of equations corresponding to the reduced echelon form. This system gives the solution.

(Gareth Williams, 2001)

Gauss-Jordan elimination has 4 basic steps below:

1. Interchange rows if necessary to bring a nonzero element to the top of the first nonzero column. The nonzero element is called pivot.
2. Create a 1 in the pivot location by multiplying the pivot row by 1/pivot.
3. Create zeros elsewhere in the pivot column by adding suitable multiples of the pivot row to all other rows of the matrix.
4. Cover the pivot row and all rows above it. Repeat steps 1 and 2 for the remaining sub matrix. Repeat step 3 for the whole matrix. Continue until the reduced echelon form is reached.

(Gareth Williams, 2001)

There is an example to show how it works follow 4 basic steps:

$$\begin{pmatrix} 0 & 0 & 2 & -2 & 2 \\ 3 & 3 & -3 & 9 & 12 \\ 4 & 4 & -2 & 11 & 12 \end{pmatrix}$$

1. In this example, we should interchange rows in order to bring a nonzero element to the top of the first nonzero column.

pivot

$$\begin{pmatrix} 3 & 3 & -3 & 9 & 12 \\ 0 & 0 & 2 & -2 & 2 \\ 4 & 4 & -2 & 11 & 12 \end{pmatrix}$$ Row1 interchange with Row2

2. Create a 1 in the pivot location by multiplying the pivot row by 1/pivot.

$$\begin{pmatrix} 1 & 1 & -1 & 3 & 4 \\ 0 & 0 & 2 & -2 & 2 \\ 4 & 4 & -2 & 11 & 12 \end{pmatrix}$$ (1/3) Row1

3. Create zeros elsewhere in the pivot column by adding suitable multiples of the pivot row to all other rows of the matrix.

$$\begin{pmatrix} 1 & 1 & -1 & 3 & 4 \\ 0 & 0 & 2 & -2 & 2 \\ 0 & 0 & 2 & -1 & -4 \end{pmatrix}$$ Row3+ (-4) Row1

4. Cover the pivot row and all rows above it. Repeat Steps 1 and 2 for the remaining sub matrix. Repeat Step 3 for the whole matrix. Continue until the reduced echelon form is reached.

$$\begin{pmatrix} 1 & 1 & -1 & 3 & 4 \\ 0 & 0 & 2 & -2 & 2 \\ 0 & 0 & 2 & -1 & -4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & -1 & 3 & 4 \\ 0 & 0 & 2 & -2 & 2 \\ 0 & 0 & 2 & -1 & -4 \end{pmatrix}$$

first nonzero column of the sub matrix

pivot

$$= \begin{pmatrix} 1 & 1 & -1 & 3 & 4 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 2 & -1 & -4 \end{pmatrix} \quad (1/2)\ \text{Row2}$$

$$= \begin{pmatrix} 1 & 1 & 0 & 2 & 5 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 & -6 \end{pmatrix} \quad \textbf{see Operation 1}$$

pivot

$$= \begin{pmatrix} 1 & 1 & 0 & 0 & 17 \\ 0 & 0 & 1 & 0 & -5 \\ 0 & 0 & 0 & 1 & -6 \end{pmatrix} \quad \textbf{see Operation 2}$$

This matrix is in the reduce echelon form by using the method of Gauss-Jordan elimination. [Gareth Williams, 2001]
Operation 1: Row1+Row2 and then Row3+(-2)Row2
Operation 2: Row1+(-2)Row3 and then Row2+Row3

For more detail about Gauss-Jordan elimination can be found (Gareth Williams. 2001. Stanley L. Grossman. 1991. Towers, D.1988.
Systems of Linear Equations: Gaussian Elimination is available from:
http://www.sosmath.com/matrix/system1/system1.html
Unit 14 Matrices and Systems of Linear Equations is available from:
http://v5o5jotqkgfu3btr91t7w5fhzedjaoaz8igl.unbsj.ca/~talderso/UWOnotes/UNIT14.pdf
Gauss – Jordan Elimination is available from:
http://www.aspire.cs.uah.edu/textbook/gauss.html)

2.1.17 Cramer's Rule

Definition of Cramer's rule: let Ax = B be a system of n linear equations in n variables such that |A| $\neq 0$. The system has a unique solution given

$$x_1 = \frac{|A_1|}{|A|},\ x_2 = \frac{|A_2|}{|A|},\ x_3 = \frac{|A_3|}{|A|},\ \ldots,\ x_n = \frac{|A_n|}{|A|}$$

Where A_i is the matrix obtained by replacing column I of A with B. [Gareth Williams, 2001]. Here determinant is equal to zero should be considered in the first step. If the determinant of coefficients of equations is zero, there may be many solutions or no solutions for the system equations.

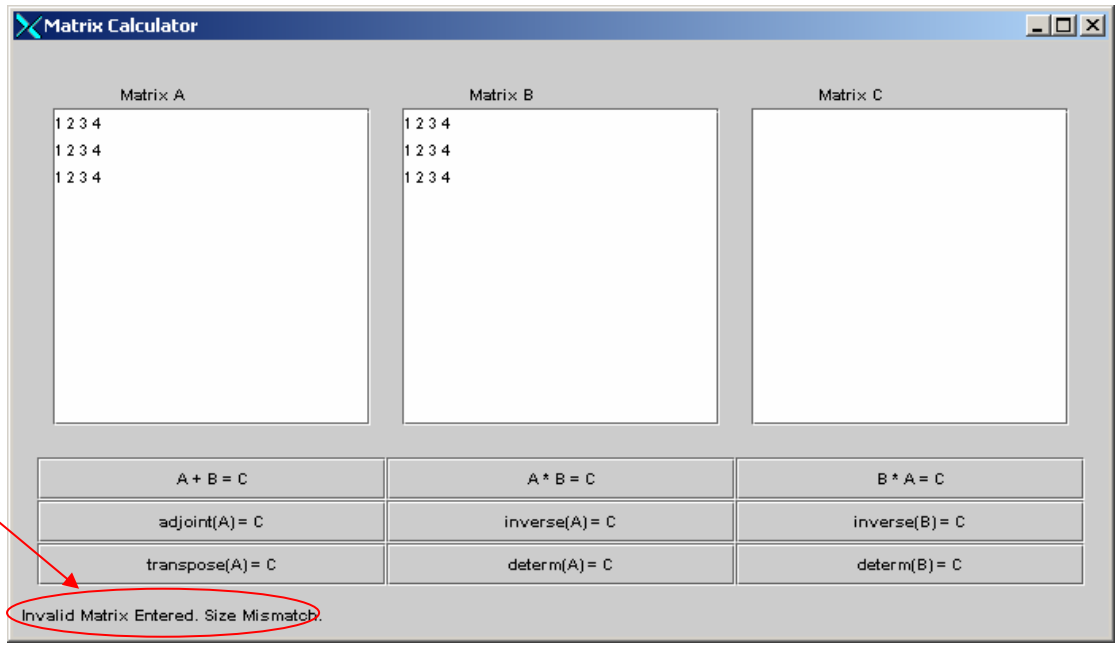
2.2 Existing System Analysis

We are going to look at existing systems which is related to linear algebra system. Many existing linear algebra system are being used for public, but there are some constraints in those systems. In this section, constraints in those systems will be demonstrated and how to improve operations provided in this project.

2.2.1 Linear algebra system named Java Matrix Calculator

The screenshot below shows a simple linear algebra system (Java Matrix Calculator).

Error Message



(Marcus Kazmierczak. 2002)

Constraints in Java Matrix Calculator:

1) The main restriction in this system is for every entry of matrix must be a square matrix. With the restriction, users are not able to manipulate some functions related to non-square matrices operations.
2) The system cannot show the current state that could be an indicator for users.
3) Users are not able to save the matrices into a file and load a file of matrix that could be manipulated they want for further operations.
4) It has not provided a help function which is for users who are not familiar with it.
5) The operation on matrix provided is not sufficient.
6) Error messages should be sent in a clear place for users easily noticing.

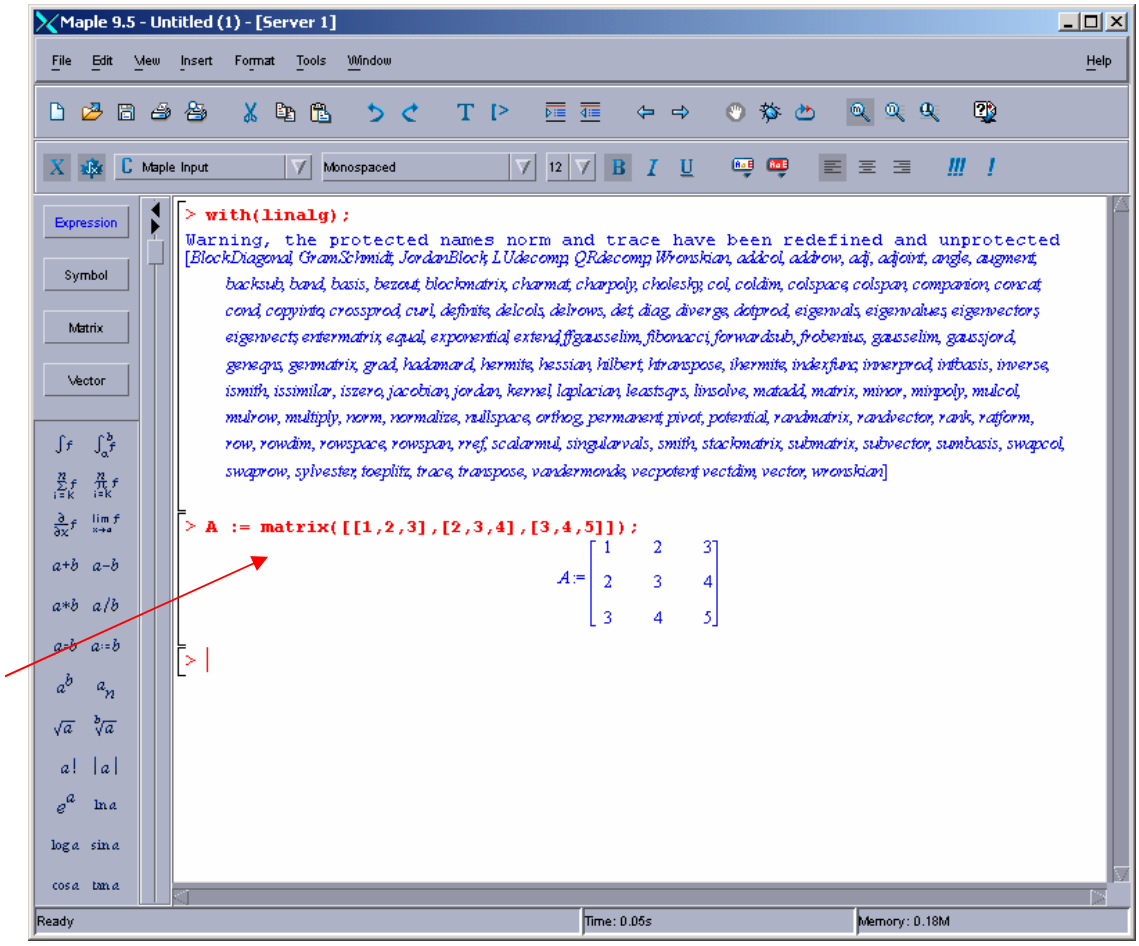
From above constraints of the system, what operations and functions should be involved in this project?

1) The system should support both square matrices and non-square matrices.
2) The system should provide current state as an indicator for users.
3) The system should have ‘save’ and ‘load’ functions for users who want for further operations.
4) The system should provide help function which will be required when he/she has problems with the system.
5) More operations on matrix should be involved.
6) Error messages should be sent at a clear place.

2.2.2 Linear algebra system in Maple

In Maple system, there is a dedicated package for linear algebra. You are able to calculate the entire linear algebra algorithm under this package. Some main features will be concerned here. In maple linear algebra package is by typing with(linalg**).** After it, maple will access linear algebra package. You can manipulate linear algebra under this platform. Below is the screenshot on Maple system:

How to represent matrix in Maple system



Constraints in the Maple system:

1) Maple users will need plenty of time to adapt the system.
2) Maple system covers really broad algorithms areas rather than a specialized linear algebra system.

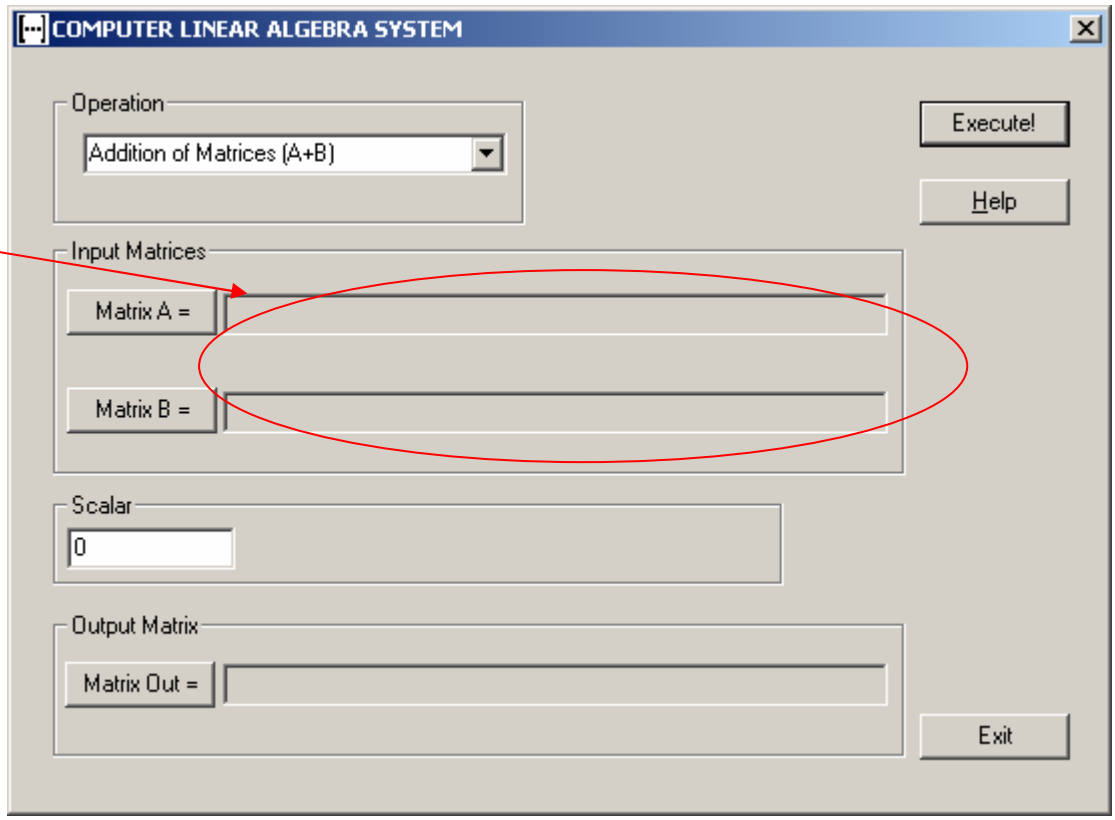
What functionalities will be provided?

1) The system will not require any training time for users. It is straightforward for users to be familiar with it. Also, the system will provide help function for user checking.
2) The system is dedicated to linear algebra mathematical background.

2.2.3 Previous Master Student's Project on Linear algebra system

This project has been produced by previous students. However, there exist some constraints in previous student's work, which should be improved in the project. The shot screen is below:

The users cannot input the matrices directly on the interface



(Santamaria-Ortega, Laia., 2002.)

Constraints:

1) Users have to input all the data into a file, and then load the file into the system. It is not convenient for the users.
2) Users may not be familiar with the terms (layout) of the system.
3) The operation on matrix provided is not sufficient.

How to improve:

1) The system could provide two ways for the users to input the data. One is that users could directly interact with the interface or they could save the data into a file for using.
2) Improve the layout under user interface design principle.
3) More operations on matrix should be involved.

3. Requirements Analysis

3.1 Introduction

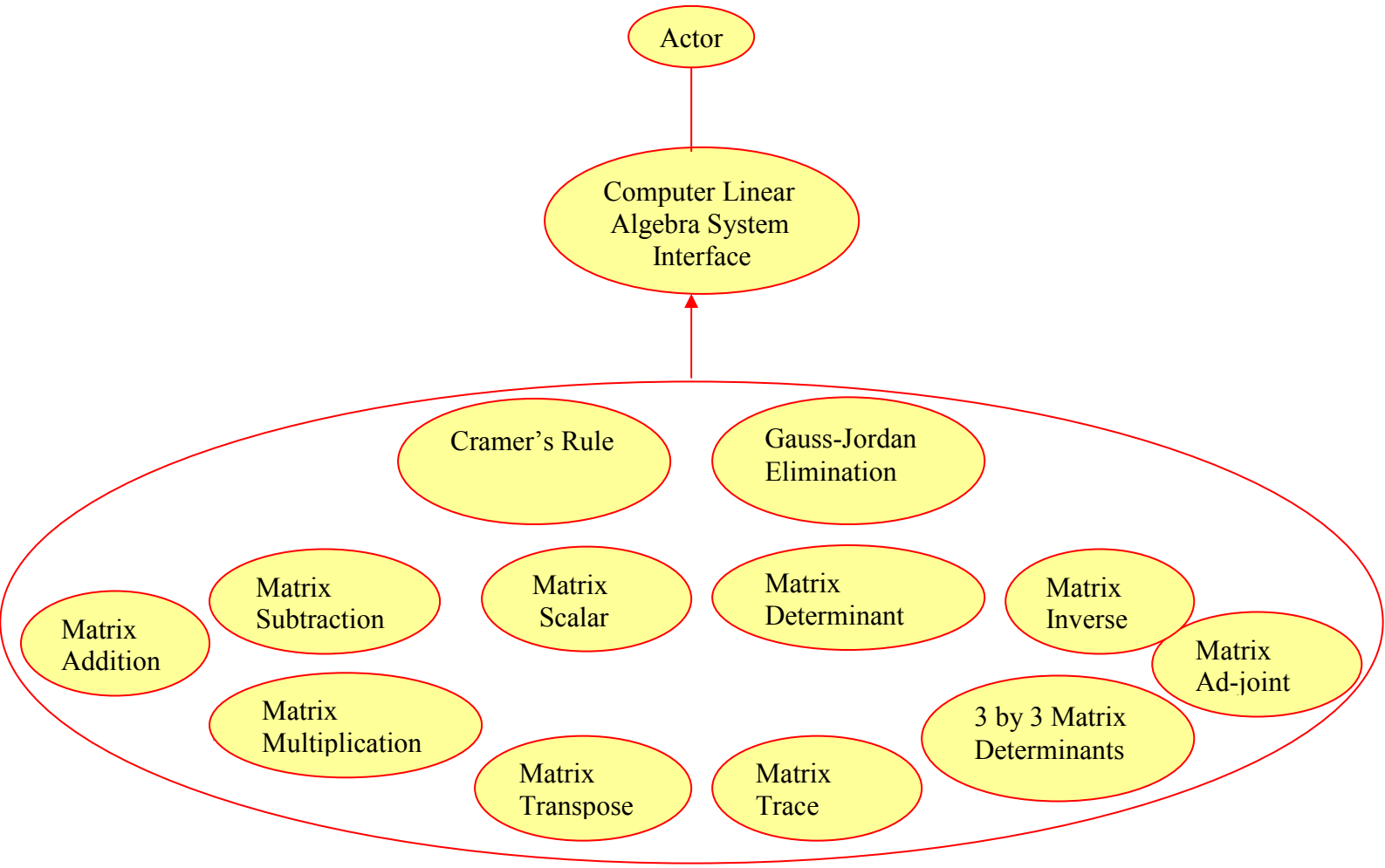
This section details the requirements for the system. It is split up into two parts, the use case study and requirements analysis. It should be mentioned here this part is likely to be changed as the system is being developed however a substantial amount of analysis has been taken place to ensure that the majority of requirements have been established.

3.2 Use Cases

This part aims to gain a better understanding of the system and the requirements that are developed will be more complete. Uses case can be used for many tasks, in this project it will be used to provide a better understanding of the system.

3.2.1 Scope

The scope of the uses cases produced is "computer linear algebra system". This is the case because they are being used to aid the requirements process rather than describe the structure of organisation. A scope diagram showing the possible use cases is found below:



User Case Diagram

Use Case 1. Matrix Addition

Scope:	Computer Linear Algebra System
Task:	Users enter two matrices
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix addition function.
Main Success Scenario:	Matrix Addition state is displayed on the screen The Users input two matrices The Users press “compute” button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrices are not complete or matrices sizes are not matched or incorrect data have been input, and wait for they input correct data. Users could load and save the matrices into a file.

Use Case 2. Matrix Subtraction

Scope:	Computer Linear Algebra System
Task:	Users enter two matrices
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix subtraction function.
Main Success Scenario:	Matrix Subtraction State is displayed on the screen The Users input two matrices The Users press “compute” button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrices are not complete or matrices sizes are not matched or incorrect data have been input, and wait for they input correct data. Users could load and save the matrices into a file.

Use Case 3. Matrix Scalar

Scope:	Computer Linear Algebra System
Task:	Users enter one matrix and one scalar number
Primary Actor:	Users
Preconditions:	Users access to the system, choose the Matrix Scalar function.
Main Success Scenario:	Matrix Scalar state is displayed on the screen The Users input one matrix and a scalar The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix or scalar are not complete or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix and the scalar number into a file.

Use Case 4. Matrix Multiplication

Scope:	Computer Linear Algebra System
Task:	Users enter two matrices
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix multiplication function.
Main Success Scenario:	Matrix multiplication state is displayed on the screen The Users input two matrices The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if matrices are not complete or the number of columns of Matrix A is not equal to number of rows of Matrix B, wait for they input two sizes of two matrices are matched or one matrix size is not complete or incorrect data input such characters, display an error message to the users, and wait for they input correct data. Users could load and save the matrices into a file.

Use Case 5. Matrix Transpose

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix transpose function.
Main Success Scenario:	Matrix Transpose state is displayed on the screen The Users input a matrix The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix is not complete or incorrect data have been input; and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 6. Matrix Trace

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix trace function.
Main Success Scenario:	Matrix Trace state is displayed on the screen The Users input a matrix. The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the user if the matrix is not complete or the matrix is not square matrix or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 7. Matrix Determinant

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix determinant function.
Main Success Scenario:	Matrix Determinant state is displayed on the screen The Users input a matrix The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix is not complete or the matrix is not square matrix or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 8. Matrix 3 by 3 Determinant

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix 3 by 3-determinant function.
Main Success Scenario:	Matrix 3 by 3 Determinant state is displayed on the screen The Users input a matrix The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix is not complete or the matrix is not 3 by 3 square matrix or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 9. Matrix Inverse

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix inverse function.
Main Success Scenario:	Matrix Inverse State is displayed on the screen. The Users input a matrix. The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix is not complete or matrix is not square matrix or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 10. Matrix Ad-joint

Scope:	Computer Linear Algebra System
Task:	Users enter a matrix
Primary Actor:	Users
Preconditions:	Users access to the system, choose the matrix ad-joint function.
Main Success Scenario:	Matrix Ad-joint State is displayed on the screen. The Users input a matrix. The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the matrix is not complete or the matrix is not square matrix or incorrect data have been input, and wait for they input correct data. Users could load and save the matrix into a file.

Use Case 11. Cramer's Rule

Scope:	Computer Linear Algebra System
Task:	Users enter coefficients and constant terms of equations
Primary Actor:	Users
Preconditions:	Users access to the system, choose the cramer's rule function.
Main Success Scenario:	Cramer's Rule State is displayed on the screen. The Users input coefficients and constant terms of equations The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the coefficient or constant terms are not complete or the size of constant term is not correct or the incorrect data have been input, and wait for they input correct data. Users could load and save the coefficients and constant terms into a file.

Use Case 12. Gauss-Jordan Elimination

Scope:	Computer Linear Algebra System
Task:	Users enter coefficients and constant terms
Primary Actor:	Users
Preconditions:	Users access to the system, choose the Gauss-Jordan function.
Main Success Scenario:	Gauss-Jordan State is displayed on the screen. The Users input coefficients and constant terms. The Users press compute button. The Users will obtain the result.
Frequency of Occurrence:	NA (Not Available)
Extensions:	Display an error message to the users if the coefficient or constant terms are not complete or the size of constant term is not correct or the incorrect data have been input, and wait for they input correct data. Users could load and save the coefficients and constant into a file.

3.3 Main Goals of Requirements

3.3.1 Functionalities should be involved in the system:

From the above use case, the following functionalities will be the core within the whole system.

1 Matrix additions
2 Matrix subtractions
3 Matrix scalars
4 Matrix multiplications
5 Computing the trace of a matrix
6 Computing the transpose of a matrix
7 Computing the determinant of a matrix
8 Computing the 3 by 3 determinant of a matrix
9 Computing the inverse of a matrix
10 Computing the matrix ad-joint
11 Solving the linear equations by Gauss-Jordan elimination
12 Solving the linear equations by Cramer's rule

3.3.2 Efficient and fast algorithms

For those operations above should be fulfilled. However, there are quite huge amount of algorithms for each operation to obtain the final solution. In this system must provide an efficient and fast algorithm for each operation. For inverting a square matrix would use an algorithm that computes the inverse of matrix by ad-joint. For computing determinant of a square matrix, the efficient algorithm should be the sum of the products of the cofactors and the corresponding elements of rows or columns.

3.3.3 System response to incorrect data input

System response to incorrect data input is crucial and it is a criterion to measure whether a system is robust. This system is fully based on linear algebra mathematics and it should be mentioned here. In matrix computation, problems will appear in multiplication of two matrices and computing the determinant of a matrix, etc. If the sizes of two matrices are not matched or the matrix is not a square matrix for matrix multiplication or computing the determinant respectively, solutions must not be offered by the system. Moreover, error messages must be provided by the system to inform users for tasks cannot be done.

3.3.4 Users of the system

The system is designed for people who are involved academic mathematical problems like undergraduate students, rather than professional people, who have much higher requirements on it.

3.4 System constraints

3.4.1 Requirements of programming language

The system could be developed by many programming languages, such as C, C++. These kinds of languages are able to deal with memory allocation for dynamic data input to make more efficient use of memory. However, the developer has a good knowledge of Java programming language, rather than C and C++. Also, Java swing could be a much better tool to develop the system interface.

3.4.2 Requirements of operating system

It must be run on either UNIX system or Windows XP system. These two systems are easily found, PCs within Bath University can run UNIX system, and my laptop supports Windows XP system.

3.4.3 Requirements of system response time

The response time of the system must not take a long time. We have mentioned above, effective and fast algorithms have been implemented in order to reduce the response time, the response time is as quick as possible.

3.5 System interface constraints

The interface is the major point for users interacting with the system. Users are able to have access all the functions provided by the system.

3.5.1 Layout

The layout of the interface should be clear to the users, and words are concise and in a straightforward manner.

3.5.2 User friendly

The interface may be used by people, who are not familiar with it. It should not contain professional words, it should be easy to follow and apprehend.

3.5.3 Error messages

Error messages providing is the most important factor, because it could lead the users to a correct solution and measure how robust of the interface. The interface will inform the users when they input an incorrect data, such as computing the determinant of a non-square matrix, matrices sizes are not matched for matrix addition and entry are not matrices at all.

3.5.4 Help function

The interface may be used by people, who have no experience on computing, the help function is to guide the users to manipulate the system.

3.5.5 Operations in a menu form

All the Operations on distinct algorithm provided by the system are contained in a menu form, which means users could be able to find all the operations in the function menu, and choose one of operations from them.

3.5.6 Current state will be informed

Current state will be kept informing to the users in order to reduce errors being made by people. It will be clear by using attractive colours.

3.6 Requirements of data input

All the functionalities will be included in different menus; the users could choose the function they wish. After operation being chosen, users are able to input the data into corresponding locations that will be indicated. Following this entering the computing button, the system will process and display the solution for users.

Also, alternative way is to input the data into files. All the files will be able to be loaded from files and then computed by the system.

3.7 Requirements of security

This system is designed for people who are involved in academic study like undergraduate students; it should be an open source for them. There is no restriction on security constraints.

3.8 Requirements of documentation

The documentation is fully following the structures offered on Dr Barry's website, and as this project is a software development project, so the structure differs from a research project. The structure is below:

1. Introduction
2. Literature survey
3. Requirement analysis
4. Requirement specification
5. Detailed design and implementation
6. System testing
7. User guide
8. Conclusion
9. Bibliography
10. Appendices

3.9 Requirements of system testing

The system testing is to test the system against the functional requirements in order to measure how robust and effective the system is. The system testing could be divided into different parts which are below:
1) Unit testing
2) Integration testing
3) System testing

The unit testing could also be divided into two parts:
1) Black box testing
2) White box testing

In this project, validation of requirements will be included in order to test all the functional requirements have been fully achieved.

3.10 Requirements of training

There are no requirements of training; user guide will fully demonstrate how to use the linear algebra system.

3.11 Project constraints

3.11.1 Deadline

The deadline of this project is set on 16th May, 2005.

3.11.2 Working hours

It is supposed to work 300 hours on the project.

4. Requirements Specification

4.1 Introduction

This part is going to state all the requirements in the system. Moreover, the requirements should be splitting into four main sections: functional requirements, non-functional requirements, hardware requirements and software requirements. The following part is going to develop a proper specification for each of these areas, the groups will be:

1) Functional requirements
2) Hardware and Software requirements
3) Non-functional requirements

4.2 Functional Requirements specification

This part is going to show all the functional requirements. A formal layout will be used for each requirement making them easy to read. They have been grouped by section so that requirements in similar areas will be grouped in one section:

4.2.1 Requirements for Matrix Arithmetic

1.

Function:	Matrix Addition
Description:	1) User will be informed if the size of two matrices is not matched or some errors in the data input. 2) User can save and load the matrix from a file that is for further use.
Inputs and source:	User inputs two matrices
Output:	Result of addition of two matrices
Obtained from:	Literature survey
Related to:	None

2.

Function:	Matrix Subtraction
Description:	1).User will be informed if the size of two matrices is not matched or some errors in the data input. 2) User can save and load the matrix from a file that is for further use.
Inputs and source:	User inputs two matrices
Output:	Result of subtraction of two matrices
Obtained from:	Literature survey
Related to:	None

3.

Function:	Matrix Scalar
Description:	1) This operation has no constraints on matrix size. 2) User can save and load the matrix that is for further use.
Inputs and source:	User inputs a single matrix and a number they want the matrix to be scaled.
Output:	Result of scalar of a matrix
Obtained from:	Literature Survey
Related to:	None

4.

Function:	Matrix Multiplication
Description:	1) Like matrix addition or subtraction, matrix multiplication concerns the size problems of two matrices. The user will be informed if two matrices cannot be multiplied or errors in the data input. 2) User can save and load the matrix that is for further use.
Inputs and source:	User inputs two matrices
Output:	Result of multiplication of two matrices
Obtained from:	Literature Survey
Related to:	None

5.

Function:	Matrix transpose
Description:	1) User is able to load the matrix from a file. 2) User can save the matrix that has been transposed for further use. 3) There are no size constraints.
Inputs and source:	User inputs a single matrix
Output:	Result of matrix being computed
Obtained from:	Literature Survey
Related to:	Ad-joint

6

Function:	Matrix trace
Description:	1) User is able to load the matrix from a file. 2) User can save the matrix that has been traced for further use. 3) Square matrix only
Inputs and source:	User inputs a single matrix
Output:	Result of matrix being computed
Obtained from:	Literature Survey
Related to:	None

7.

Function:	Matrix determinant 3 by 3
Description:	1) 3 by 3 square matrix will be informed when user has input a non-square matrix or error data or non-3 by 3 square matrices. 2) Unlike other operations, this operation will lead to a single number as a result, so the system may be not ask for saving it. However, the system could provide this function for user who wants to retain the result.
Inputs and source:	User inputs a single matrix for computing
Output:	Result of the determinant of a 3 by 3 square matrix
Obtained from:	Literature Survey
Related to:	None

8.

Function:	Matrix determinant
Description:	1) Square matrix will be informed when user has input a non-square matrix or error data. 2) Unlike other operations, this operation will lead to a single number as a result, so the system may be not ask for saving it. However, the system could provide this function for user who wants to retain the result.
Inputs and source:	User inputs a single matrix for computing
Output:	Result of the determinant of a square matrix
Obtained from:	Literature Survey
Related to:	Ad-joint, Inverse

9.

Function:	Matrix inverse
Description:	1) Again, only square matrix will be accepted by the system. The system will inform the user when user has input a non-square matrix or error data. 2) User can save and load the matrix that is for further use. 3) The system will inform the user if there is no inverse for the matrix has been input.
Inputs and source:	User inputs two matrices
Output:	Result of the inverse of a square matrix
Obtained from:	Literature Survey
Related to:	Determinant

10.

Function:	Matrix ad-joint
Description:	1) Normally, matrix ad-joint operation should be a part of computing the determinant of a matrix. User can save ad-joint matrix for further use.
Inputs and source:	User inputs a single square matrix
Output:	Result of the ad-joint of a square matrix
Obtained from:	Literature Survey
Related to:	Determinant, Cramer's Rule

4.2.2 Requirements for Linear equations

11.

Function:	Solving linear equations by Gauss-Jordan elimination
Description:	1) The system will inform the user if the answer of linear equations does not exist.
Inputs and source:	User inputs coefficient for each variable being computed
Output:	Result for each variable
Obtained from:	Literature Survey
Related to:	None

12.

Function:	Cramer's rule
Description:	1) The system will inform the user if the answer of linear equations does not exist.
Inputs and source:	User inputs coefficient for each variable being computed
Output:	Result for each variable
Obtained from:	Literature Survey
Related to:	Ad-joint, Determinant

4.2.3 Requirements for other functions

13.

Function:	Help
Description:	Help function is provided by the system for helping people who are not familiar with the system or matrix mathematical background.
Inputs and source:	None
Output:	The system will show the help.
Obtained from:	Existing system of linear algebra system
Related to:	Linear equations, Matrix Arithmetic

14.

Function:	Save
Description:	The user could click “Save” button to save the matrix into a file for further use.
Inputs and source:	None
Output:	Matrix saved into a file
Obtained from:	Literature survey
Related to:	Linear equations, Matrix Arithmetic

15.

Function:	Load
Description:	Load button is for user loading the file into the system for computing
Inputs and source:	None
Output:	The matrix in a file will be displayed on the interface
Obtained from:	Literature survey
Related to:	Linear equations, Matrix Arithmetic

16.

Function:	Error messages
Description:	When the users input incorrect data and matrices sizes are not correct, the system must be able to send error messages to the users to notify them “errors exist!”
Inputs and source:	Input incorrect data or matrices
Output:	Error messages
Obtained from:	Literature survey
Related to:	Whole system

4.3 Hardware and Software Requirements Specification

17.

Requirement:	Various components of the hardware should be provided for files backup
Description:	Floppy disk, USB and more hardware drive could be using for files backup
Metrics:	When the hardware disk of the computer is in the event of failure, backups could be used.
Obtained from:	N/A
Collaborators:	None

18.

Requirement:	The system must be easy to maintain.
Description:	Designer should near-full rights to the system, and have the ability to update the source code of the system if necessary.
Metrics:	Maintainability: duties required to sustain the system should be easily performed by the designer
Obtained from:	From the software engineering book
Collaborators:	None

19.

Requirement:	Operating system
Description:	It is with windows XP operating system or UNIX operating system or Terms application system can be found everywhere within the university of Bath
Metrics:	Reliability: Connection must be fully functioning.
Obtained from:	N/A
Collaborators:	None

20.

Requirement:	The system should be based on a reliable platform
Description:	There are external risks could affect the system, which may include power failure, loss of data, and many more risks that can take place.
Metrics:	Reliability: the platform must be fully functioning.
Obtained from:	N/A
Collaborators:	None

4.4 Non-functional Requirements Specification

21.

Requirement:	The system should be reliable.
Description:	The system should have a high level of reliability. This means that the user could be able to rely on it for computing matrix.
Obtained from:	Software engineering book
Collaborators:	None

22.

Requirement:	The system should be robust.
Description:	The system will have a low level of failures and will recover quickly from any failures that it has.
Obtained from:	Software engineering book
Collaborators:	None

23.

Requirement:	The system should be understandable.
Description:	The system will be easy to understand. This means that the code must be clear so that future developers can understand the algorithms.
Obtained from:	Software engineering book
Collaborators:	None

24.

Requirement:	The system should not be complex.
Description:	The system code should be as simple as it is possible to make it. Algorithms should be fast and efficient where possible.
Obtained from:	Software engineering book
Collaborators:	None

25.

Requirement:	The system shall efficient to use.
Description:	This means the system should be able to run efficiently. Users should be able to perform task quickly with the minimum number of key/ mouse presses.
Obtained from:	Software engineering book
Collaborators:	Whole system

26.

Requirement:	The system should be easy to learn.
Description:	The system should allow users to learn how to use all of functions in a short time. This will save users unnecessary time while trying to learn the system.
Obtained from:	Software engineering book
Collaborators:	None

27.

Requirement:	The system should be easy to maintain.
Description:	This requirement means the system should allow for new developers to carry out maintain in a fast efficient manner.
Obtained from:	Software engineering book
Collaborators:	None

28.

Requirement:	User interface should be user-friendly, concise and simple for a user. Also, the interface is straightforward to follow.
Description:	1. The layout of the interface must be clear. 2. The help function must be provided for a user who is not familiar with the system. 3. Users who are not familiar with the system could be able to manipulate it. 4. The interface must provide error message and how it is being fixed. 5. All the functionalities provided by the system will be contained in a menu form. 6. Current state of the system will be kept informing the users.
Obtained from:	Principles in software engineering book
Collaborators:	None

29.

Requirement:	The response time of the system.
Description:	Response time of the system must be as minimum time as possible by using effective algorithms.
Obtained from:	Literature survey
Collaborators:	None

30.

Requirement:	The system should be testable
Description:	The system could be able to be tested. A high level of coverage testing should be able to be gained.
Obtained from:	Principles in software engineering book
Collaborators:	None

4.5 Miscellaneous requirements

31.

Requirement:	Requirements of programming language
Description:	This linear algebra system will be divided into algorithm and interface parts. Both of them will be developed by Java programming language.
Obtained from:	N/A
Collaborators:	None

32.

Requirement:	Requirement of security
Description:	There are no Requirements for security.
Obtained from:	Principles in software engineering book
Collaborators:	None

33.

Requirement:	Requirements of documentation
Description:	The structure of documentation is fully following dissertation information provided from Dr Barry's website. 1. Introduction 2. Literature survey 4. Requirement analysis 5. Requirement specification 6. Detailed design and implementation 7. System testing 8. User guide 9. Conclusion 10. Bibliography 11. Appendices
Obtained from:	Lectures and website of Dr Barry
Collaborators:	None

34.

Requirement:	Requirements of training
Description:	There are no training requirements, all the information concern this system will be included in user guide part in this dissertation.
Obtained from:	N/A
Collaborators:	None

35.

Requirement:	Project requirements
Description:	1) The deadline of this project is 16th May 2005. 2) Time should be allocated on this project is 300 hours.
Obtained from:	Website of Project
Collaborators:	None

4.6 System Scope

Here the scope of the system should be stated as clearly as possible. The scope for the computer linear algebra system follows:

Computer linear algebra system

Internal

Algorithm arithmetic
- Matrix addition
- Matrix subtraction
- Matrix multiplication
- Matrix scalar
- Matrix transpose
- Matrix trace
- Matrix determinant 3 by 3
- Matrix determinant
- Matrix inverse
- Matrix adjoint

System of linear equations
- Gauss-Jordan Elimination
- Cramer's rule

Save function
- Save the matrix into a file

Load function
- Load the matrix from a file

Help function
- Help function provided for people who are not familiar with the system

5. System Design

5.1 Introduction

The design section is the most important section of a software development process. It consists of a number of models of the system being developed, each model viewing the system from a different perspective. The models in this design are: system model, software model.

This section lays out the different models that are involved in the design process. It starts off with the high level system model to gain an overview of the system and also includes data flow that needs to be covered in more detail as the design process. The software model, which is the second part of this section, covers the CRC process before getting lower level and more technical. The third part will look at the design from a user interface point of view. It tries to ensure that the product produced is simple to use and is effective at doing the job it is designed for. This part does not look at coding at all and is therefore at a much high level of design then the software model part.

5.2 System Model

5.2.1 Introduction

A system model is "an important bridge between the analysis and the design processes" (Sommerville, I., 2001). Its objective is to develop an understanding of the existing system and to specify the required system. It consists of models, each describing the system specification from a different perspective. There are three perspectives from which a system can be viewed and each of these has specific models that can be used depending on the system being modelled.

a. An external perspective would require an architectural model.
b. A behavioural perspective would require a data-processing model or a stimulus-response mode.
c. A structural perspective would require a composition model or a classification model.

5.2.2 Architectural Model

The goal of this part is to decide on the overall design of the system. Normally, this will involve working with the system users to distinguish what is the system and what is the system's environment. However, the development of the system architecture has been based on the knowledge from requirements analysis phase due to constraints of this project.

The computer linear algebra system comprises a number of components in the context model below:

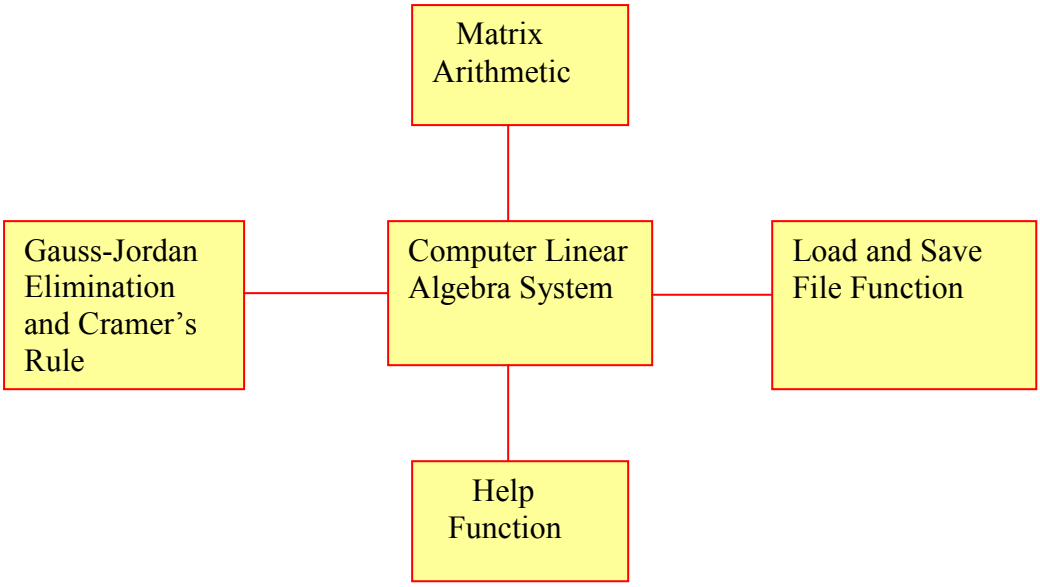


Figure 1 Computer Linear Algebra System Architecture

Figure 1 is an architectural model shows the principal sub-systems that make up a computer linear algebra system.

5.2.3 Data Flows

In order that the system can be designed at a lower level (with respect to the software being used for implementation) the movement of data needs to be identified. Each diagram is accompanied by some descriptive text to walk through the flows taking place in the relevant diagram.

5.2.3.1 Matrix arithmetic

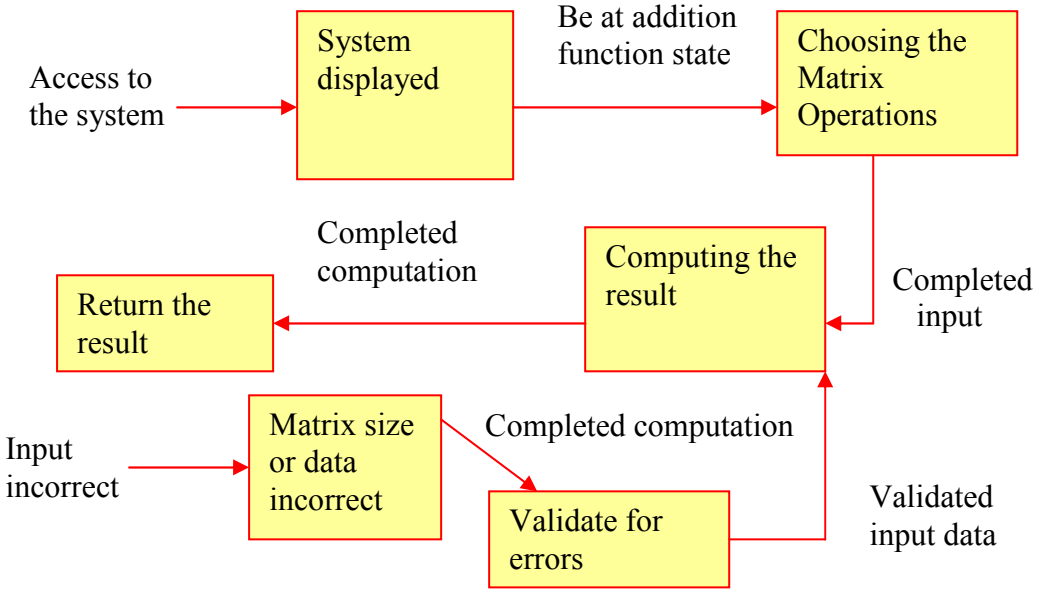


Figure 2 Data Flow Diagram for Matrix Arithmetic

This diagram of the system is to compute matrix arithmetic. Start by access into the system and it will be at addition function state as default. After it, the users could choose any functions they wish. Following it, matrices A and B should be input to corresponding location and compute them. The completed matrices are sent for the system computing. Finally, the result will be returned. If the user input incorrect data such as matrices size are not matched or data error, validation will take place and then the validated data will be sent to the system for computing.

Name	Description	Type
Access to the system	Run java program to access to the system.	Flow
Display the system	The linear algebra system is displayed.	Process
Be at addition function panel	It is at addition function panel as default.	Flow
Choosing the functions they would like	The users could choose any functions they would like to use.	Process
Completed input	Matrices have been input at corresponding location.	Flow
Computing the result	The system will be computing the result for matrices.	Process
Completed computation	Computation is finished.	Flow
Return the result	The final result of matrices will be returned.	Process
Input incorrect data	The users input data are incorrect.	Flow
Matrix size or data incorrect	Error message will be sent to the users that matrix size or flawed data have been input.	Process
Validate for errors	Validate the input to ensure they are correct.	Process
Validated input data	The data have been validated and send to the system.	Flow

Data dictionary for Matrix Arithmetic

5.2.3.2 Systems of linear equations

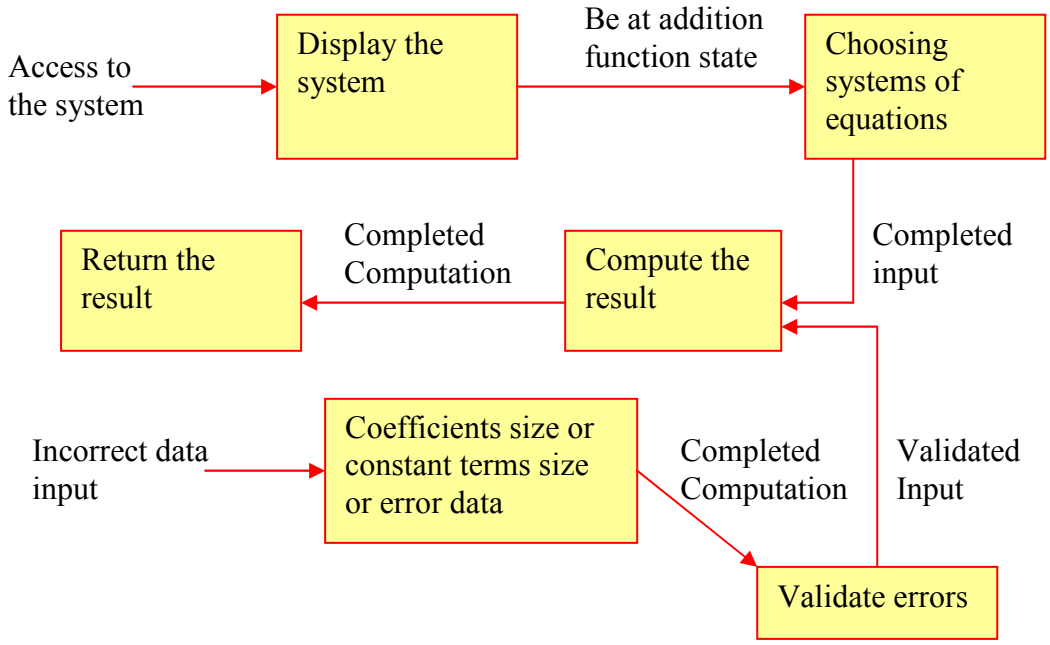


Figure 3 Data Flow Diagram for Systems of linear equations

This data flow diagram shows how the Cramer's rule or Gauss-Jordan Elimination works. As well as before, the very first step is to access to the system and it will be at addition state as default. Then by choosing systems of equations is to access to Cramer's rule state or Gauss-Jordan Elimination state. After that, the user should input coefficients and constant terms of equations to corresponding location. By clicking "Compute" is to obtain the final result. If the user input incorrect data such as coefficients or constant terms size are incorrect or data error, error message will be sent to notify the users to validate the data, then the validated data will be sent to the system for computing to obtain the final result.

Name	Description	Type
Access to the system	Run java program to access to the system.	Flow
Display the system	The linear algebra system is displayed.	Process
Be at addition function state	It is at addition function state as default.	Flow
Choosing System of linear equations	The users could choose system of linear equations for computing equations.	Process
Completed input	Coefficients and constant terms of equations have been input at corresponding location.	Flow

Compute the result	The system will be computing the result for each variable.	Process
Completed computation	Computation is finished.	Flow
Return the result	The final result of each variable will be returned.	Process
Incorrect data input	The users input data are incorrect.	Flow
Coefficients size or constant terms size or error data	Error message will be sent to the users that matrix size or flawed data have been input.	Process
Validate errors	Validate the input to ensure they are correct	Process
Validated input	The data have been validated and send to the system for computing the final result.	Flow

Data dictionary for System of linear equations

5.2.3.3 Load function

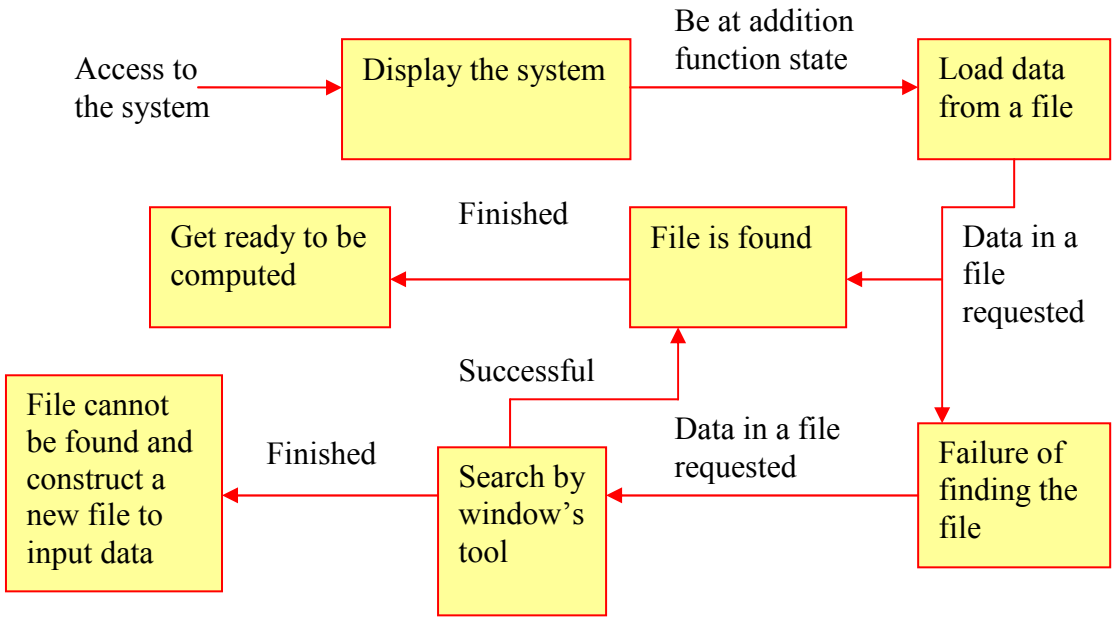


Figure 4 Data Flow Diagram for Load Function

This data flow diagram shows how to load data from a file. Firstly, access to the system, it will be at addition function state as a default. Pressing load button, it will lead you to find the file you wish. In the event of a failure of finding a file, it would require the user to search the file by windows tool. It will either display success or failure. If file cannot be found, then the user should construct a new file to save the

data. Ones file is found by window's tool, and then the data in that file will be displayed at corresponding location to get ready to be computed.

Name	Description	Type
Access to the system	Run java program to access to the system	Flow
Display the system	The linear algebra system is displayed	Process
Be at addition function panel	It will be at addition function panel as a default	Flow
Load data from a file	The user may wish to load his/her data from file	Process
Data in a file requested	The required data in a file is requested	Flow
File is found	The requested file is found	Process
Failure of finding the file	The requested file is not found	Process
Search by window's tool	It could search the specific file by using search function provided by windows XP	Process
Finished	The data containing a terminate request.	Flow
File cannot be found and construct a new file to save data	The specific file is required cannot be found and then the user should construct a new file to input data	Process
Successful	File is found by window's tool	Flow
Get ready to be computed	Data in the specific file will be displayed at corresponding location to get ready to be computed	process

Data Dictionary for Load Function

5.2.3.4 Save function

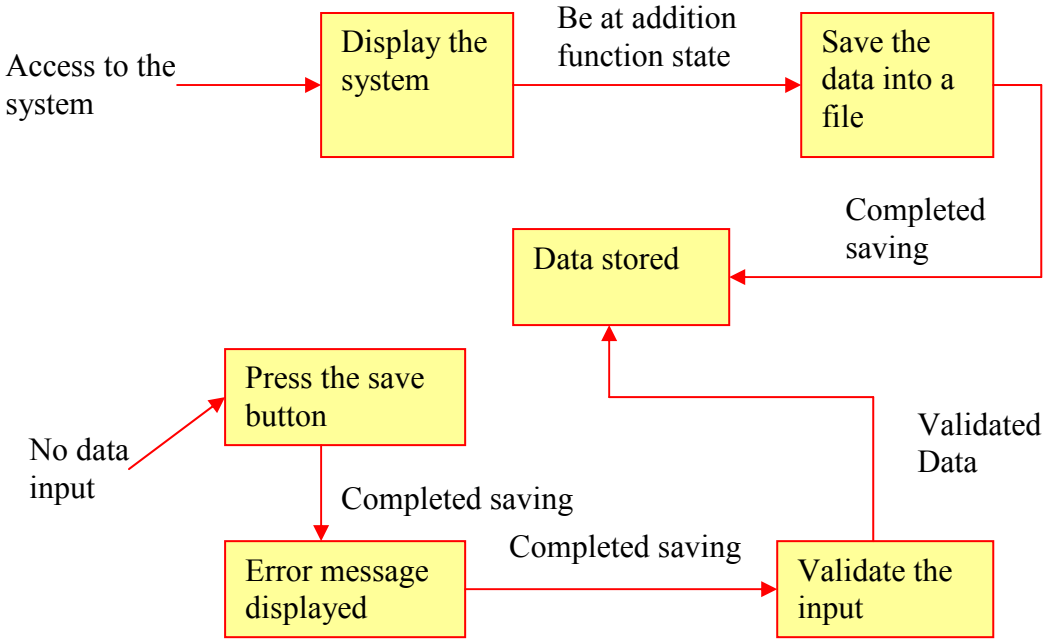


Figure 5 Data Flow Diagram for Save Function

This diagram covers save function. It starts with accessing to the system, and it should be at addition function state as a default. Pressing the "Save" button displayed on the screen, it will require user saving the data into a file. In the event of no data input, error message will display on the screen to require data input. Following it, data can be saved the data into a specific file.

Name	Description	Type
Access to the system	Run java program to access to the system	Flow
Display the system	The linear algebra system is displayed	Process
Be at addition function panel	It will be at addition function panel as a default	Flow
Save the data into a file	Pressing the save button to save the data they wish	Process
Completed saving	Saving a file is completed	Flow
Data stored	The data have been stored into a file by the system	Process
No data input	No data have been input for saving	Flow
Press the save button	Save button have been pressing for saving purpose	Process
Error message displayed	Error message will return when the user has not input data to save	Process
Validate the input	User has input data for saving	Process

Data Dictionary for Save Function

5.2.3.5 Help function

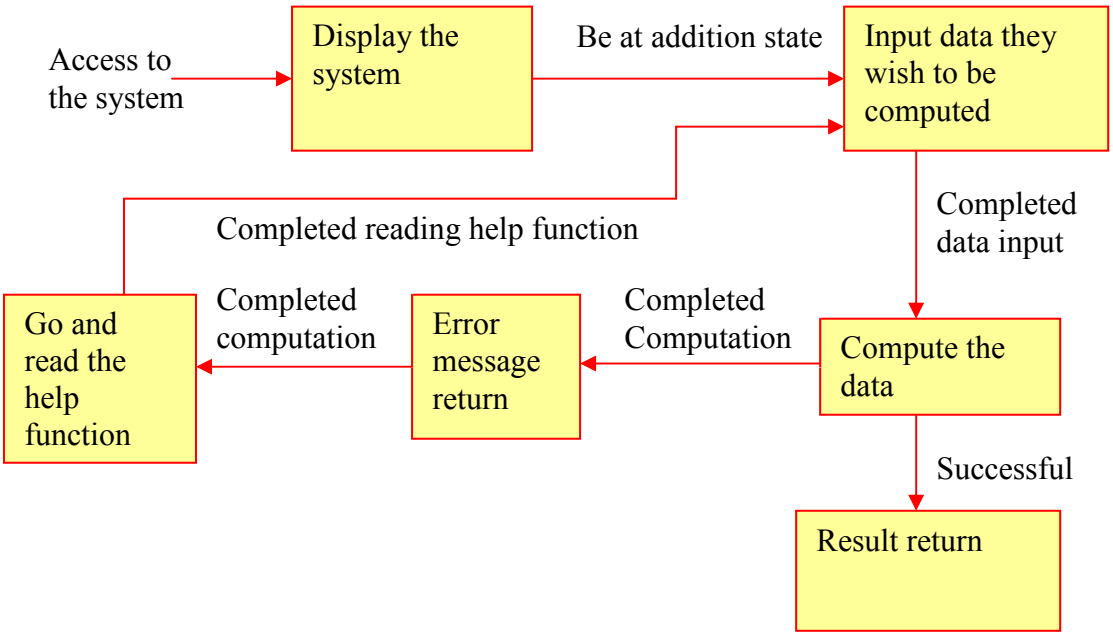


Figure 6 Data Flow Diagram for Help Function

This diagram shows help function being used. Firstly, access to the system and as the same as before it should be at addition state. After it, the user could input data to be computed by the system, in the event of error message being returned, the user needs to go and read the help function to gain an understanding how the system works, then the user is able to manipulate the system, and finally, the result will be returned when the user inputs data correctly.

Name	Description	Type
Access to the system	Run java program to access to the system	Flow
Display the system	The linear algebra system is displayed	Process
Be at addition function panel	It will be at addition function panel as a default	Flow
Input data they wish to be computed	The user maybe directly input data without fully understanding how it works	Process
Completed data input	The data have been input into the system	Flow
Compute the data	Pressing button to compute the data	Process
Successful	The data have been computed successfully	Flow
Result return	Result will display on the screen when correct data input	Process

Error message return	Error message will be notified to the user it he/she inputs data are incorrect	Process
Go and read help function	Users do not fully understand how to operate the system, they could go and read help function for some tips	Process

Data Dictionary for Help Function

5.3 Software Model

5.3.1 Introduction

This section is part of design describe in detail the software design considerations used to implement the system.

The basis for the design of the system is from the initial CRC modelling techniques, it should involve the features required of the system, and also form functional and non-functional requirements accordingly.
The CRC modelling could from clear class relationships and show the important areas of the system. This transformation from the low-level modelling to high-level class organisation will be discussed in section 4.3.2 CRC Modelling.

5.3.2 CRC Modelling

CRC (Class-Responsibility-Collaborator) Modelling has been the key to a well designed system. From the initial blocks which were used to define the basic system requirements, to the high-level class relationships and design choices, this technique has been thoroughly successful.

CRC cards are very simple in the design. CRC cards are drawn out as follows:

CLASS	
RESPOSIBILITY	**COLLABORATOR**

CRC card is split into three sections:

1). Class represents a collection of similar object. Objects are things of interest in the system. They can be a person, place, thing or and other concept important to the system. The class name appears across the top the card.

2). Responsibility is anything that the class knows or does. These responsibilities are things the class has knowledge about itself, or things the class can do with the knowledge it has.

3). Collaborator is another class that is used to get information for, or perform actions for the class at hand. It often works with a particular class to complete a step (or steps) in a scenario. The collaborators of a class appear along the right size of the CRC card.

5.3.3 Components and Classes

In this section the classes will be identified by using CRC modelling which form the system. Individual entities will be grouped into classes which can form the basis of the system architecture.

5.3.3.1 Components

The following parts will form the system:

1. Matrix Arithmetic
   1.1) Matrix Addition
   1.2) Matrix Subtraction
   1.3) Matrix Scalar
   1.4) Matrix Multiplication
   1.5) Transpose
   1.6) Trace

1.7) Determinant 3 by 3
1.8) Determinant
1.9) Inverse
1.10) Adjoint

2. Systems of linear equations
   2.1) Gauss-Jordan Elimination
   2.2) Cramer's Rule
3. Save Function
4. Load Function
5. Help Function

Each class should be detailed, with their responsibilities and collaborators listed:

Matrix Arithmetic	
Responsibility	**Collaborator**
Compute Matrix Addition	Save Function
Compute Matrix Subtraction	Load Function
Compute Matrix Scalar	Help Function
Compute Matrix Multiplication	
Compute Matrix Transpose	
Compute Matrix Trace	
Compute Matrix Determinant 3 by 3	
Compute Matrix Determinant	
Compute Matrix Inverse	
Compute Matrix Adjoint	

System of linear equations	
Responsibility	**Collaborator**
Compute linear equation by using Gauss-Jordan elimination	Help Function
Compute linear equation by using Cramer's rule	

Save Function	
Responsibility	**Collaborator**
The user could save the matrix for further computation	Matrix Arithmetic

Load Function	
Responsibility	**Collaborator**
The user could load a file for computing	Matrix Arithmetic

Help Function	
Responsibility	**Collaborator**
List help topics	All other classes
Detail a help topic	

5.3.3.2 Class Structure

From these classes have been concerned above, an initial class relationship diagram can be drawn up. It will show the relationship of the classes in a high-level design view. This diagram could be easily transferred to actual implementation of the linear algebra system.

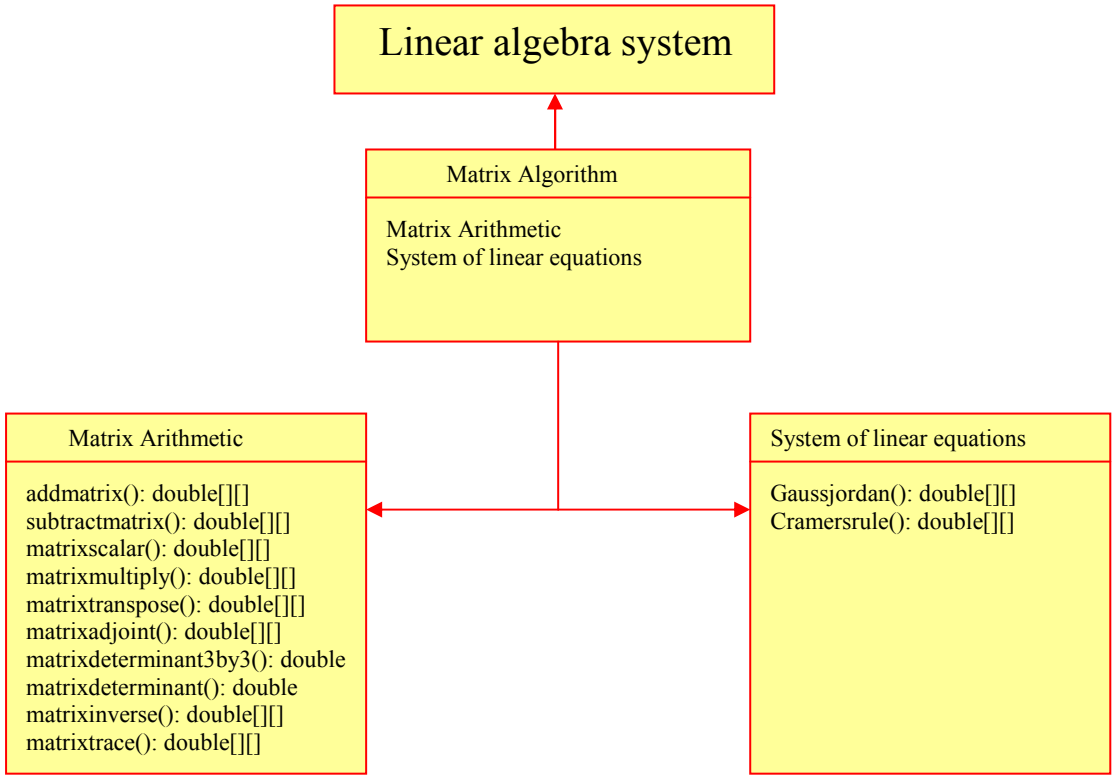


Figure 7 UML class diagram

5.3 User Interface Design

5.4.1 Introduction

The way information is presented is absolutely vital to the system. How to design an effective and user-friendly interface should be a challenge to the developer. The description of the principals is related to UI design and how they will be involved in the system.

5.4.2 User Interface Design Principles

User interface should take consideration of the physical and mental capabilities of the people who will be involved using the software. Six design principles presented in the table 15.3 of software engineering book (Sommerville, I. 2001). The interface design should meet each of guidelines in the software engineering book.

5.4.2.1 User familiarity

The designs draw, as much as possible, from the terms and concepts used and from the environment of the user. The design of linear algebra system is almost like a calculator for scientific use. This makes the interface of the system easy.

5.4.2.2 Consistency

Comparable operations are activated in the same way. As an example: matrix addition operation and matrix subtraction operation are carried in the same way;

5.4.2.3 Minimal surprise

"Users should never be surprised by the behaviour of a system." (Sommerville, I. 2001). Most of features of the linear algebra system are performed in the standard way.

5.4.2.4 Recoverability

There is a mechanism to allow users to recover from errors.

5.4.2.5 User guidance

"The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities." (Sommerville, I. 2001). Error messages will be sent when user inputs incorrect data and error messages are context sensitive and helpful. Also, the system will provide a help function for user, which is a novice's guide to all the operations that can be carried in the linear algebra system.

5.4.2.6 User diversity

"The interface should provide appropriate interaction facilities for different types of system user." (Sommerville, I. 2001). This principle could not be applied in this design, with the limitation of the time; this point could be considered for future developments.

With these principles taken into account, the next phase of the User interface design process is to develop the layout of the system.

5.4.3 Interface Designs

This is an initial stage that has been undertaken of the basic system interface design to sketch out the ideas for the interface layout.

5.4.3.1 Design of overall system

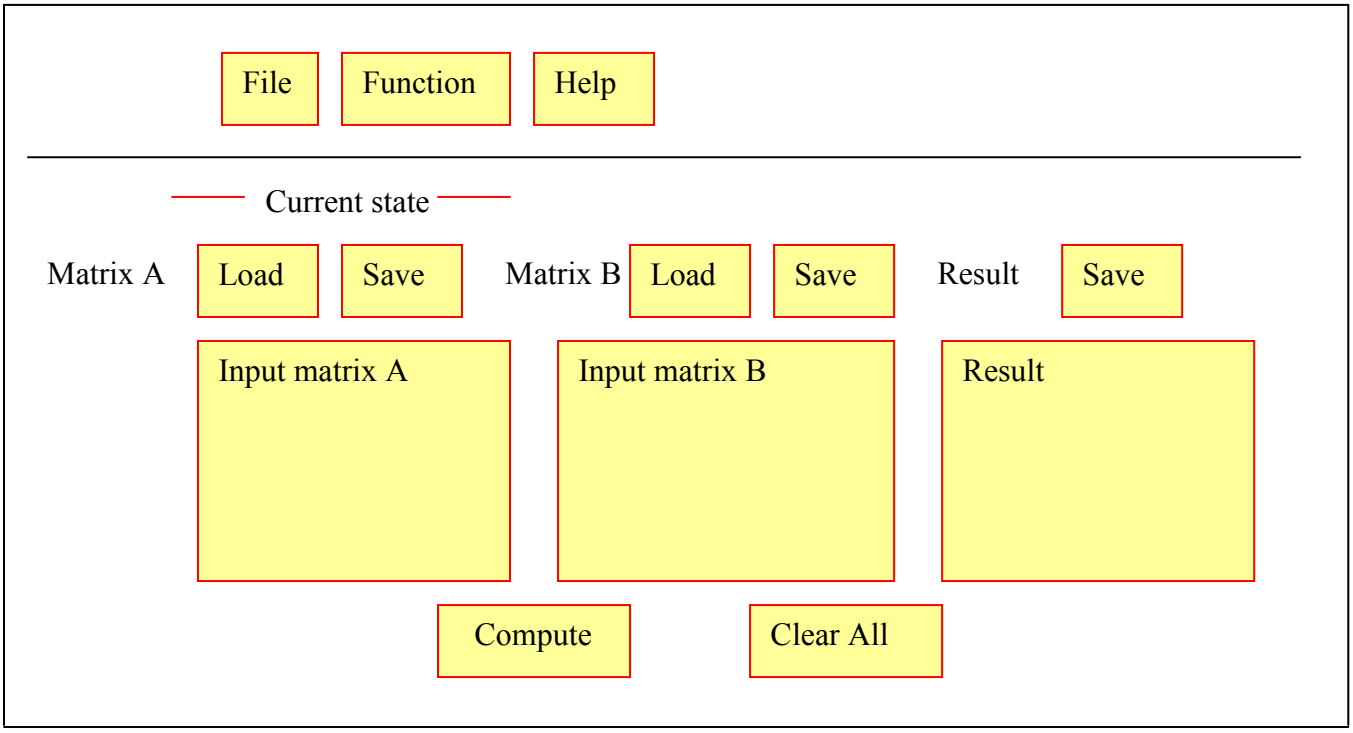


Figure 8 Overall view of Computer Linear Algebra System

As it can be seen, this is an overall view of the system. It is to achieve the design principles as being described above. For user familiarity, the terms provided of the system is really easy and concise. On the top, three are three menu bars; users could choose any functions they like. The system will notify the users which state it is. Users may be straightforward to follow the information provided on the system to operate the matrix arithmetic. On the bottom of the system, “Get Result” button and “Clear All” could let users get the final result of matrices and clear the screen without any input. The navigation is provided how to manipulate the system easily.

5.4.3.2 Design of functions in the system

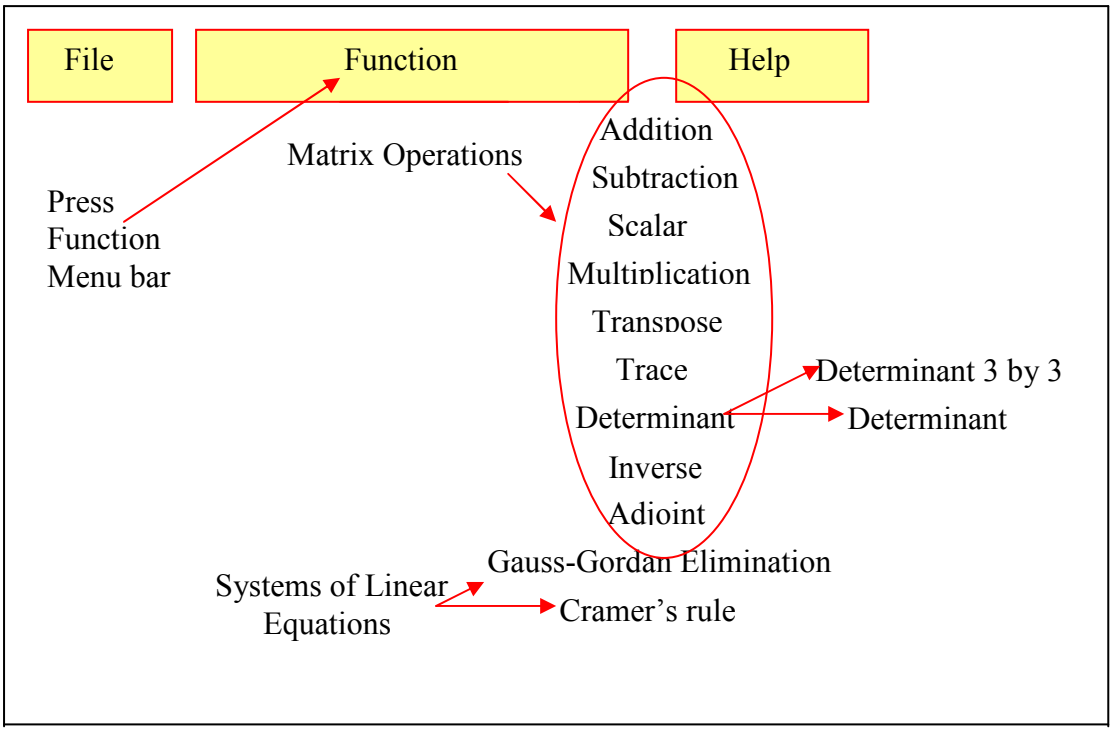


Figure 9 Interface Design of functions

This is the interface for function of the system. When the user is pressing function mean bar, all the operations involved will be scrolling down. It is straightforward for users to find the operation they wish. Also, it is easy to understand what is going on in the system.

5.4.3.3 Design of Help function in the system

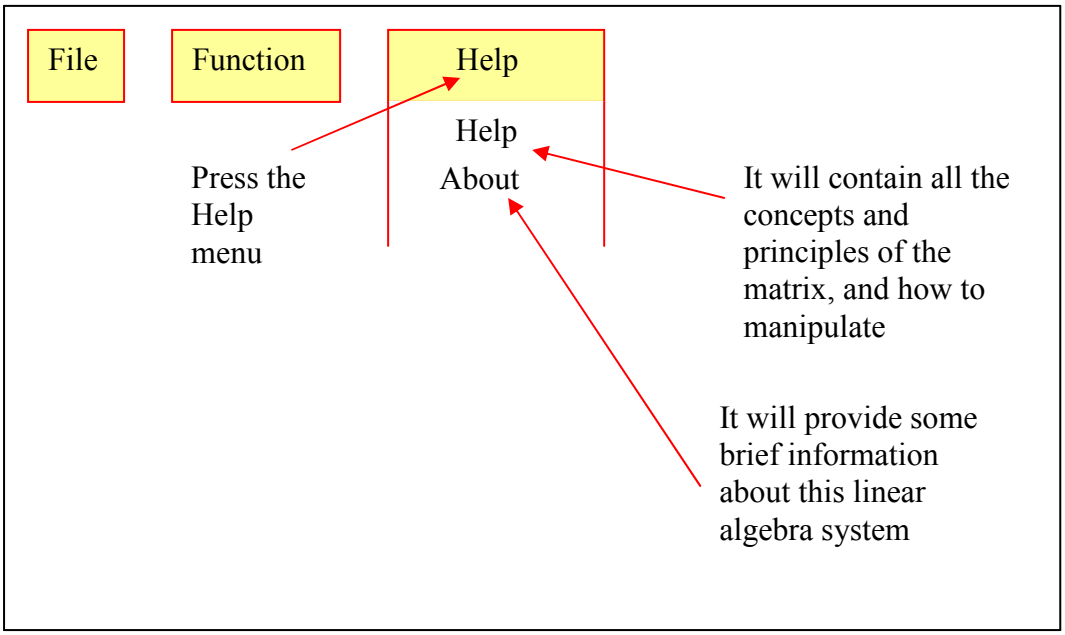


Figure 10 Interface Design of Help Function

This is help function which could provide brief information how to manipulate the system. Information in the help function will be concise and easy to apprehend.

5.4.3.4 Another major layout of the interface for different operations on matrix

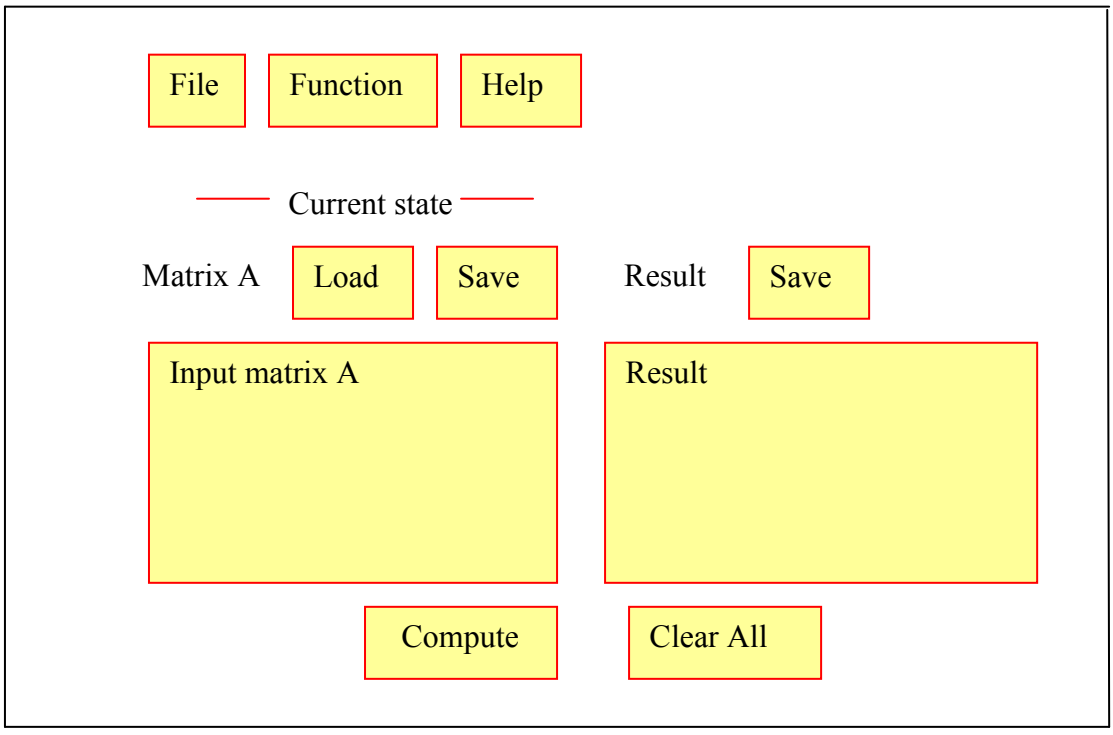


Figure 11 Interface layouts for other matrix operations

This is another major interface layout for the design. This is for a single matrix operation such as determinant, transpose, inverse, and ad-joint because there is only matrix to be involved for computing.

5.4.3.5 Brief description of each function provided by the system

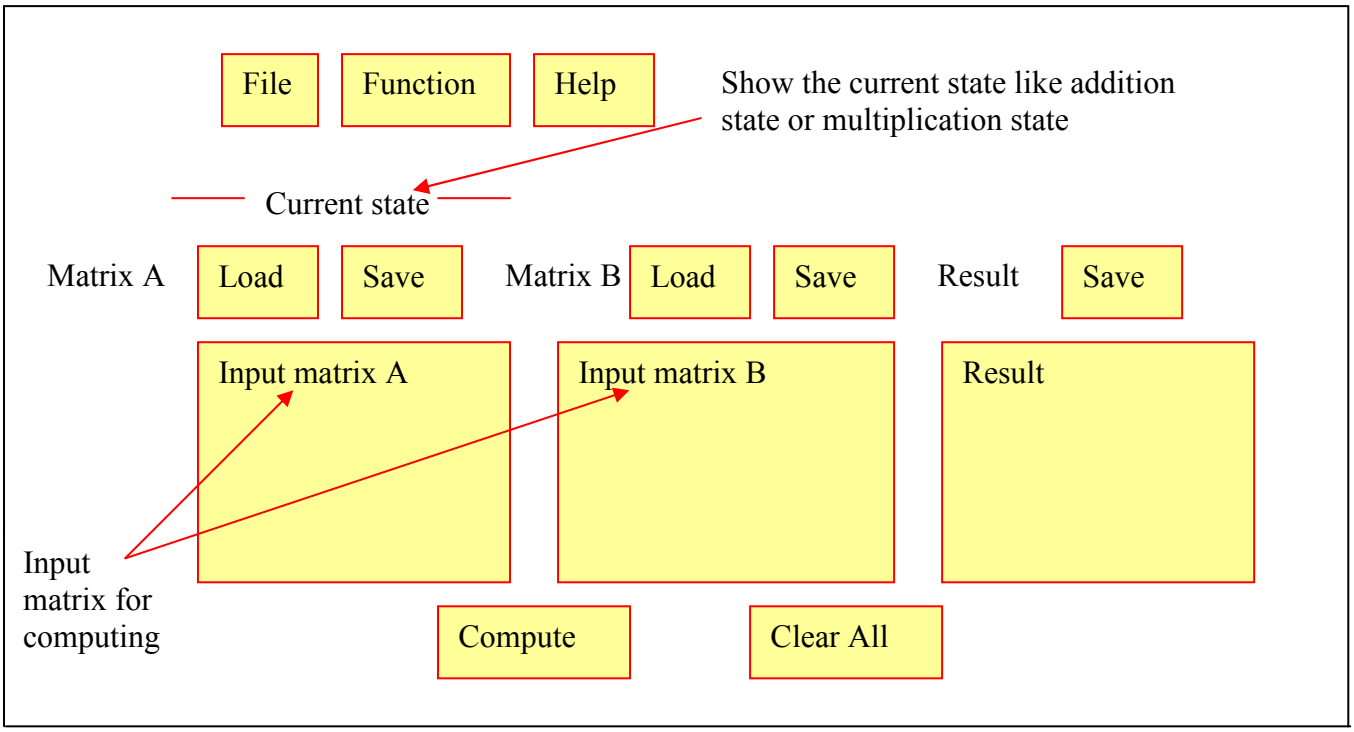


Figure 12 Brief description of system interface

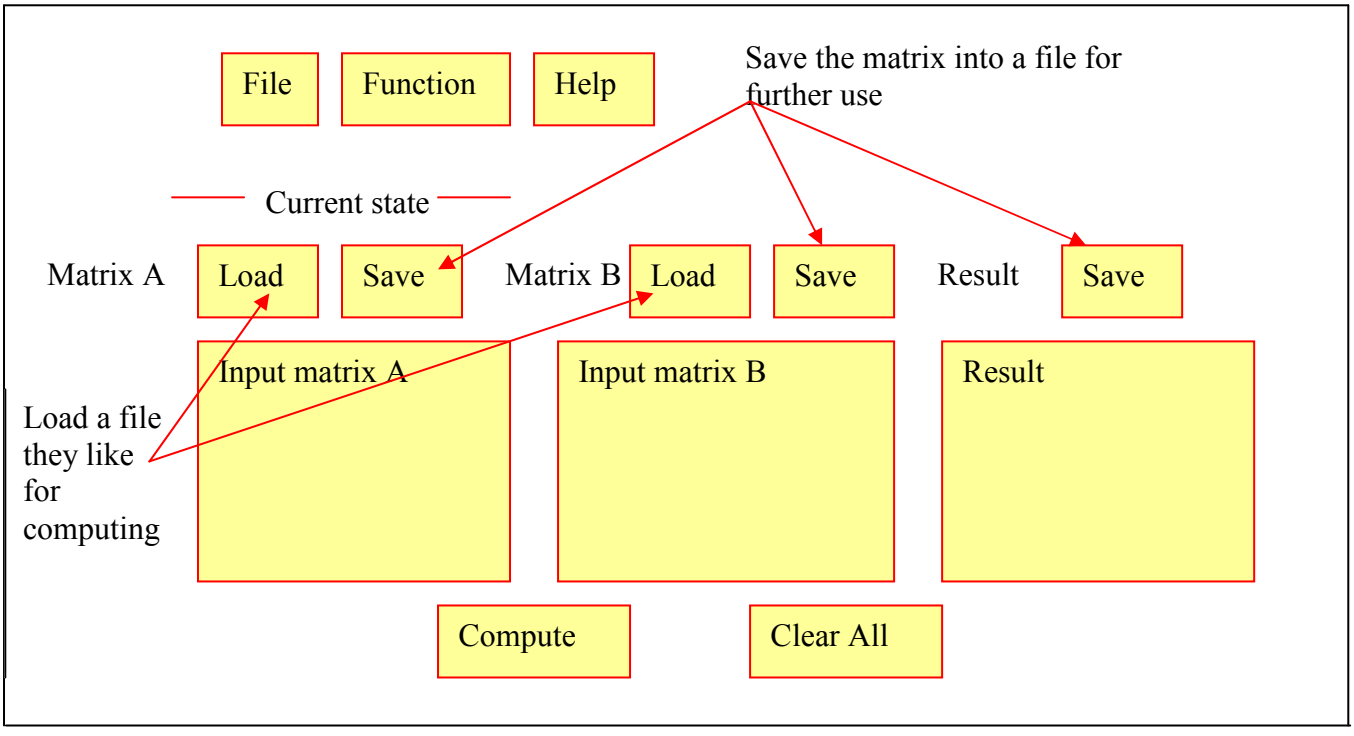


Figure 13 Brief description of system interface

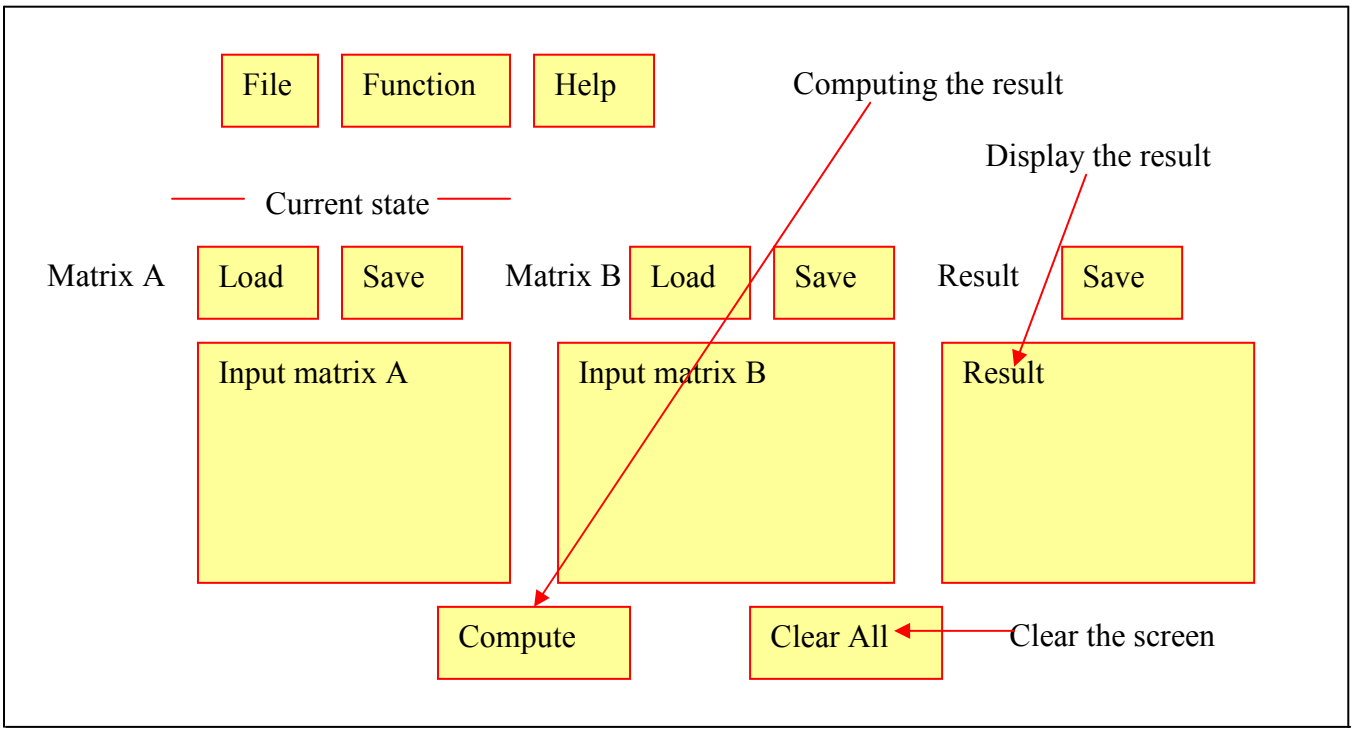


Figure 14 Brief description of system interface

6. Detailed Design and Implementation

6.1 Introduction

This part includes with a low level description of the classes that the algorithm part will be used in 'pseudo' code. As the algorithm parts play as a very important role in this project, all the applications displayed on the interface are based on it. It can be seen as following diagram:

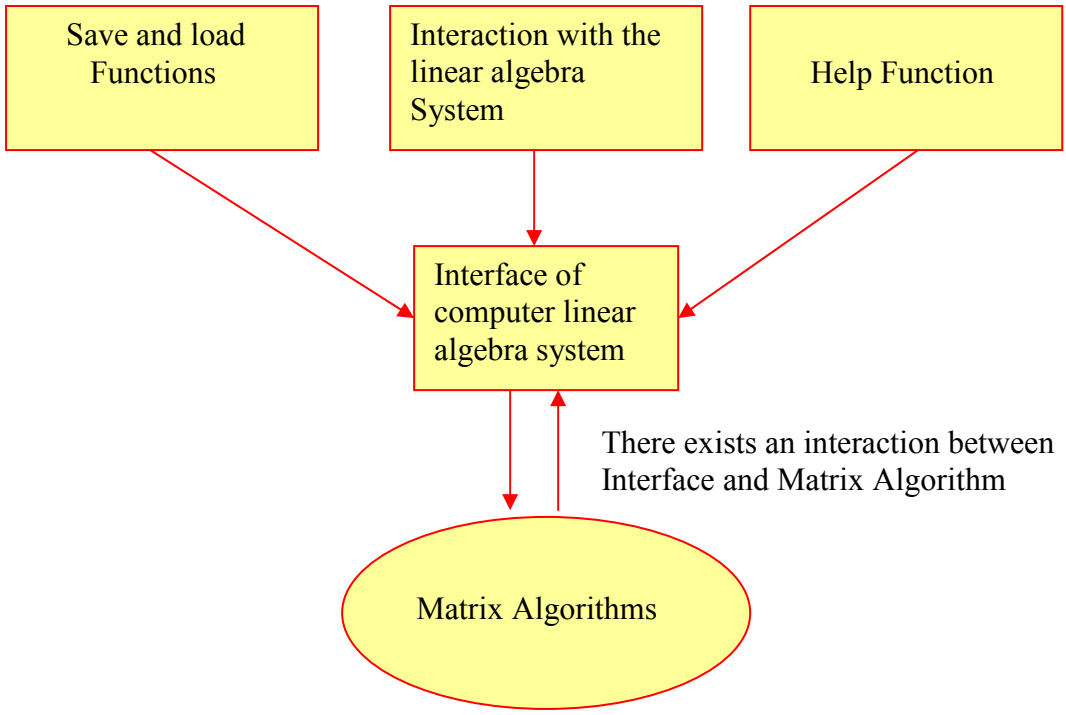


Figure 15 Relationships between the User Interface and Matrix Algorithms

When user is interactive with the system is actually interactive with the algorithms of the system. As all the matrices input will be transformed to algorithms part, after computation, the corresponding result will be sent back to the interface of being displayed to the user. The aforementioned procedure is how the system works. It is fully based algorithms of matrix. In this case, the core part which is algorithm part should be fully analyzed and how to implement each operation such as matrix addition, matrix transpose, matrix determinant and so on. Java 'pseudo' code will be used to explain the source code more straightforward to be apprehended. Also, how to read users' input from the interface is another major point should be mentioned.

6.2 Technologies

There are a number of different languages could be implemented for this project, all the languages have their own advantages and disadvantages. For C and C++, These two languages are able to deal with memory allocation for dynamic data input to make more efficient use of memory, for Java language, the interface could be easily implemented by Java swing. Also, the designer of the project has a better knowledge on Java rather than C and C++ programming languages.

6.3 Formal specification of Matrix operations

This part will provide the formal specification of Matrix Arithmetic and System of linear equations.

The operations:

1. Matrix addition
2. Matrix subtraction
3. Matrix scalar
4. Matrix multiplication
5. Matrix trace
6. Matrix transpose
7. Matrix determinant 3 by 3
8. Matrix determinant
9. Matrix inverse
10. Matrix ad-joint
11. Cramer's rule
12. Gauss-Jordan Elimination

Formal Specification:

Operations:

Matrix addition: $\rightarrow$ double
Matrix subtraction: $\rightarrow$ double
Matrix scalar: $\rightarrow$ double
Matrix multiplication: $\rightarrow$ double
Matrix trace: $\rightarrow$ double
Matrix transpose: $\rightarrow$ double
Matrix determinant3by3: $\rightarrow$ double
Matrix determinant: $\rightarrow$ double
Matrix inverse: $\rightarrow$ double
Matrix ad-joint: $\rightarrow$ double
Cramer's rule: $\rightarrow$ double
Gauss-Jordan elimination: $\rightarrow$ double

Semantics:

Matrix addition: $\rightarrow$ double
Pre-condition: the size of two given matrices should be matched
Post-condition: return the result of two input matrices

Matrix subtraction: $\rightarrow$ double
Pre-condition: the size of two given matrices should be matched

Post-condition: return the result of two input matrices

Matrix scalar: → double
Pre-condition: None
Post-condition: return the input multiplied by given scalar

Matrix multiplication: → double
Pre-condition: the number of columns of the first given matrix should be equal to number of rows of the second given matrix.
Post-condition: return the product of two matrices

Matrix trace: → double
Pre-condition: the given matrix should be a square matrix (n×n matrix)
Post-condition: return a single number trace of given matrix

Matrix transpose: → double
Pre-condition: None
Post-condition: return the transpose of given matrix

Matrix determinant 3 by 3: → double
Pre-condition: the given matrix should be a 3 by 3 square matrix
Post-condition: return a single number determinant of the matrix

Matrix determinant: → double
Pre-condition: the given matrix should be a square matrix (n×n matrix)
Post-condition: return a single number determinant of given matrix

Matrix inverse: → double
Pre-condition: the given matrix should be a square matrix (n×n matrix) and it is not a singular matrix which means the determinant of given matrix is not zero.
Post-condition: return the inverse of given matrix

Matrix ad-joint: → double
Pre-condition: the given matrix should be a square matrix (n×n matrix).
Post-condition: return the ad-joint of the given matrix

Cramer's rule: → double
Pre-condition: coefficients for each variable and constant term from the equations to be solved.
Post-condition: return the result for each variable, if no solutions or many solutions return no unique solution

Gauss-Jordan elimination: → double
Pre-condition: coefficients for each variable and constant term from the equations to be solved.
Post-condition: return the reduced row echelon form.

6.4 Algorithm of Matrix arithmetic

Once the formal specification of the arithmetic operations has been defined, operations could be implemented in Java.

6.4.1 Matrix Addition

From the formal specification above, precondition for matrix addition is that matrices should have the same size. Otherwise, the operation cannot be done. In this case, the algorithm should detect whether the size of matrices are matched. Also, error handling should be included in it. Java 'pseudo' code provided below:

```
double[][] add (double[][] matrixA, double[][] matrixB)
{
boolean geterror = false;

 // first  we need to check if the users' input of matrix A are not matrices.
     if((length of columns in any row in matrix A) != the length of columns in first row
in matrix A)
     {
          geterror = true;
     }

// second we need to check if the users' input of matrix B are not matrices
      if((length of columns in any row in matrix B) != (the length of columns in first
row in matrix B))
      {
          geterror = true;
       }

// Check the size of matrix A and matrix B are matched or not
     if (matrixA.length != matrixB.length){
                    return null;
       }

       for(int i = 0; i < matrixA.length; i++){
              if(matrixA[i].length != column || matrixB[i].length != column){
                                  return null;
              }
        }

   // the sizes of matrices are matched, then do the addition operation
       for (int i=0; i<matrixA.length;i++){
              for (int j=0; j < matrixA[0].length; j++){
                          result[i][j] = matrixA[i][j] + matrixB[i][j];
              }
      }
      return result;
 }
```

6.4.2 Matrix Subtraction

Also, from formal specification above, precondition for matrix subtraction is that matrices should have the same size. Otherwise, the operation cannot be done. In this case, the algorithm should detect whether the size of matrices are matched. Again, error handling should be included in it. Java 'pseudo' code provided below:

```
double[][] subtraction(double[][] matrixA, double[][] matrixB)
{
    boolean geterror = false;

// first checking if the users' input of matrix A are not matrices.
     if((length of columns in any row in matrix A) != the length of columns in first row
in matrix A)
     {
          geterror = true;
     }

// second checking if the users' input of matrix B are not matrices
      if((length of columns in any row in matrix B) != (the length of columns in first
row in matrix B))
      {
           geterror = true;
       }

// next step checking the size of matrices are matched or not
  if (matrixA.length != matrixB.length){
              return null;
  }

      for(int i=0; i<matrixA.length; i++){
            if (matrixA[i].length ! = column || matrixB[i].length != column){
                                    return null;
            }
      }

// the size of matrices is matched, then do the subtraction operation
       for (int i = 0; i < matrixA.length; i++){
              for( int j =0; j < matrixA[0].length; j++){
                     result[i][j] = matrixA[i][j]- matrixB[i][j];
               }
       }
       return result;
}
```

6.4.3 Matrix Multiplication

The same as before, from formal specification above, precondition for matrix multiplication is that the number of columns of the first given matrix should be equal to number of rows of the second given matrix. In this case, the algorithm should detect whether the size of matrices satisfied precondition of matrix multiplication. As well as before, error handling is included in this method. Java 'pseudo' code provided below:

```
double[][] multiplication( double[][] matrixA, double[][] matrixB)
{
    boolean geterror = false;

// checking the user's input is a correct matrix for matrix A
if(the length of columns in any row in matrix A != the length of columns in the first
row in matrix A)
{
            geterror = true;
}

// checking the user's input is a correct matrix for matrix B
if(the length of any row in matrix B !=  the length of first row in matrix B)
{
            geterror = true;
}


// the number of columns of the first given matrix should be equal to number of rows
// of the second given matrix
if (matrixA[0].length != matrixB.length){
            return null
}


// checking any row in matrix A is not matched the column of matrix B
for( every column in matrix A using int i control it)
{
      for(every row in matrix B using int j control it)
      {
          if(any column in matrix A is not matched the row in matrix B)
          {
                return null;
           }
       }
}


// the number of columns of the first given matrix is equal to number of rows of the
//second given matrix, then do multiplication operation
```

```
    for (int i = 0; I < matrixA.length; i++){
          for( int j = 0; j < matrixB[0].length; j++){
                result[i][j] = 0;
                for ( int k=0; k < matrixA[0].length-1; k++){
                      result[i][j] =  result[i][j] + matrixA[i][k] * matrixB[k][j];
                 }
           }
      }
      return result;
}
```

6.4.4 Matrix Scalar

From formal specification above, precondition for matrix scalar is none. It will involve error handling to deal with the incorrect input of matrices. Java 'pseudo' code provided below:

```
double[][] scalar( double scalar, double[][] matrixA)
{
boolean geterror = false;

// checking the input matrix is correct
if(the length of columns in any row in matrix A != the length of columns in the first
row of matrix)
{
    return null;
   geterror = true;
}

// precondition here is none
     for ( int i = 0; i < matrix.length; i++){
          for ( int j = 0; j < matrix[0].length; j++){
                result[i][j] = scalar * matrix[i][j];
           }
     }
     return result;
}
```

6.4.5 Matrix Transpose

From formal specification above, precondition for matrix Transpose is none. It also will include error handling to deal with the error input. Java 'pseudo' code provided below:

```
double[][] transpose (double[][] matrixA)
{
```

```
// checking the input matrix is correct
if(length of any row in matrix A != length of first row in matrix A)
{
      return null;
}

 // precondition here is none
double matrixtranspose[][] = new double[matrixA[0].length][matrix.length];
    for (int i = 0; i < matrixA[0].length; i++){
            for (int j = 0; j < matrixA.length; j++){
                    matrixtranspose[i][j] = matrixA[j][i];
             }
     }
return matrixtranspose;

}
```

6.4.6 Matrix Trace

From the specification of the matrix trace, a square matrix will be required for precondition. This method is able to deal with errors of input data as well. The java 'pseudo' code is below:

```
double trace(double[][] matrixA)
{
boolean geterror = false;
// checking the input matrix is correct
if(length of columns in any row in matrix A != length of columns in first row in
matrix A)
{
      geterror = true;
}

// here square matrix should be checked, ensure the length of row should be equal to
//the length of column

if(the length of therow in matrix A != the length of the column in matrix A)
{
      geterror = true;
}

// the input matrix is a square matrix
if(geterror == false)
{
     For(int i = 0; i < matrixA.length; i++)
    {
         result = result + matrixA[i][i];
     }
}
return result;
}
```

6.4.7 Matrix Determinant 3 by 3 Matrices only

The aim for 3 by 3 determinant is to use a different algorithm to compute it, and from the formal specification, precondition is that the given matrix should be 3 by 3 matrices. In this algorithm, error handling should be included to detect non 3 by 3 square matrices or any incorrect non-square matrices. Java 'pseudo' code is below:

```
double[][] determinant3by3 (double[][] mat)
{
        double det = 0;
         boolean geterror = false;

//checking the square matrices
         if(length of columns in any row of mat != length of columns in first row of mat)
         {
                  return true;
         }

         If(length of rows in mat != length of columns in mat)
         {
                   return true;
          }

// checking the square matrices are 3 by 3 square matrices
          if(length of rows in mat != 3 && length of columns in mat != 3)
          {
                   return true;
          }

// the square matrices are 3 by 3 matrices
           if(geterror == false)
           {
                 if(length of rows in mat == 3 && length of columns ==3)
                 {
                  det=
mat[0][0]*mat[1][1]*mat[2][2]+mat[0][1]*mat[1][2]*mat[2][0]+mat[0][2]*mat[1][0]
*mat[2][1]-mat[0][2]*mat[1][1]*mat[2][0]-mat[0][0]*mat[1][2]*mat[2][1]-
mat[0][1]*mat[1][0]*mat[2][2];
                 }
            }
     return det;
}
```

6.4.8 Matrix Determinant

Matrix determinant is a crucial part in the algorithm sections, because it has been related to other operations such as matrix inverse and Cramer's rule. From formal specification above, precondition for matrix determinant is that the given matrix should be a square matrix (n×n matrix). Otherwise, the operation cannot be done. 1 by 1 matrix, 2 by 2 matrix will be considered in our algorithm. Also, the algorithm should detect whether the size of matrix is square and error handling should be provided in this method. Java 'pseudo' code provided below:

```
double[][] determinant ( double[][] matrix)
{
     Boolean geterror = false;
    double det = 0;

// checking the users' input matrix is correct
     if(length of columns in any row in matrix A != length of columns in first row in
matrix A)
     {
      geterror = true;
      }


// checking square matrix
       if ( matrix.length != matrix[0].length)
        {
             geterror = true;
        }

// the matrix is square matrix but size is 1 by 1
        if ( matrix.length == 1 && matrix[0].length ==1)
        {
                    det = matrix[0][0];
         }


// size of square matrix of A is 2 by 2
        if ( matrix.length ==2 && matrix[0].length ==2)
          {
                     det = matrix[0][0]*matrix[1][1] – matrix[0][1]*matrix[1][0];
           }

// size of square matrix is n by n, some more concepts will be incorporated into it, like
// minor and cofactor
// for minor from mathematical background, the minor of the element a_ij is denoted
// by M_ij, and the determinant of the matrix is the remaining by deleting the row i and
// column j of A
// and for cofactor from mathematical background, the cofactor of a_ij is denoted C_ij,
// and given by C_ij = (–1)^(i+j) M_ij. The minor and cofactor differ in at most sign.
```

// $C_{ij} = \pm M_{ij}$.
// find the minor of the element at the position of the given row and column
// calculate the determinant of the minor
// multiply the determinant of minor by cofactors, the result is -1 or 1 depends on
// sum of given row and column

// determinant is the sum of the products of the elements of any rows and any
// columns with the cofactors For example |A| = $a_{11}\ c_{11}$+……$a_{nn}\ c_{nn}$.

```
        if ( matrix.length == n && matrix[0].length ==n){

            for( int i = 0; i < matrix.length;  i++){

                    for (int j=0; j < matrix.length; j++){

                        if (int i and int j in cofactor, sum == even then multiply 1){

det = matrix[0][0]*cofactorofmatrix[0][0] + matrix[1][1]*cofactorofmatrix[1][1]

+……+

matrix[matrix.length][matrix.length]* cofactormatrix[matrix.length][matrixlength];

                        }
                    }
            }
         return det;
        }
```

```
        if ( matrix.length == n && matrix[0].length ==n)
{
```

// determinant is the sum of the products of the elements of any rows and any
// columns with cofactors. For example |A| = $a_{11}\ c_{11}$+……$a_{nn}\ c_{nn}$.

```
            for( int i = 0; i < matrix.length;  i++){

                    for (int j=0; j < matrix.length; j++){

                        if (int i and int j in cofactor, sum == odd then multiply -1){

det = matrix[0][0]*cofactorofmatrix[0][0] +    matrix[1][1]*cofactorofmatrix[1][1]
+ ……. +
matrix[matrix.length][matrix.length]*cofactormatrix[matrix.length][matrixlength];
                        }
                    }
```

```
            }
        return det;
         }
}
```

6.4.9 Matrix Adjoint

Matrix adjoint is another important part in algorithm section, because it has been related to matrix inverse. From formal specification above, precondition for matrix adjoint is that the given matrix should be a square matrix ($n \times n$ matrix), otherwise, the operation cannot be done. In this case, the algorithm should detect whether the size of matrix is square and error handling will be involved in this method. Java 'pseudo' code provided below:

```
double[][] adjoint( double[][] matrix)
{

// checking the input matrix is correct

if(length of columns in any row in matrix A != length of columns in first row in
matrix A)
{
      return null;
}


// check matrix size whether it is a square matrix

      if (matrix.length != matrix[0].length)
      {
                  return null;
       }


// the size of matrix is square which means n by n matrix
// from the definition of matrix adjoint, first we calculate the cofactor for each element
// in the matrix, then transpose cofactor matrix into adjoint matrix.
// find the minor of the element at the position of the given row and column
// calculate the determinant of the minor
// multiply the determinant of minor by cofactors, the result is -1 or 1 depends on
// sum of given row and column

        if ( matrix.length == n && matrix[0].length ==n){
            for( int i = 0; i < matrix.length;  i++){
                  for (int j=0; j < matrix.length; j++){
                      if (int i and int j in cofactor, sum == even then multiply 1)
                     Cofactor[i][j] = 1 * minor[i][j]
                      }
                 }
```

```
            return transpose(resultofcofactor);
        }


    if ( matrix.length == n && matrix[0].length ==n){
        for( int i = 0; i < matrix.length;  i++){
            for (int j=0; j < matrix.length; j++){
                if (int i and int j in cofactor, sum == odd then multiply -1)
                Cofactor[i][j] = (-1) * minor[i][j]
                }
            }
        }
      return transpose(resultofcofactor);
    }
}
```

6.4.10 Matrix Inverse

Matrix inverse will have been related to matrix determinant and matrix adjoint, and also the precondition from formal specification requires square matrix (n by n matrix) and aforementioned it is related to matrix determinant, and so it should be a non-singular matrix which means determinant of matrix is not zero. Otherwise, the operation cannot be done. In the algorithm design, it could be able to detect whether it is a non-singular square matrix or not and error handling could be included in this method to check error input. Java 'pseudo' code provided below:

```
double[][] inverse(double[][] matrix)
{
boolean geterror = false;

// check the size of matrix is square matrix or not (n by n matrix)

        if ( matrix.length != matrix[0].length)
        {
            return null;
        }

// check the determinant of the matrix is zero or not

        else if (determinant(mat) == 0)
        {
            geterror = true;
        }
// from the definition of linear algebra, matrix inverse could be calculating as
// inverse(matrix) = (1/|determinant|) * adjoint (matrix);

        double result[][] = (1/determinant(matrix)) * adjoint(matrix);
        return result;
  }
```

6.4.11Cramer's Rule

Cramer's rule is to solve linear system of equations; the precondition from formal specification is to solve the variables of equations. With the constant terms together, it will form augmented matrix to be solved. In the design, it should detect errors by error handling.

```
double[][] cramersrule(double[][] coefficient, double[][] remain)
{
    double result[][] = new double[coefficient.length][1];
    double temp[][] = new double[coefficient.length][coefficient.length];
    boolean coef = false;
    boolean vectorcheck =false;
    boolean rowcheck = false;
    boolean errorchecking = false;

    // checking the error in coefficient part
    for(int i = 0; i < length of coefficient; i++)
    {
        if(coefficient[i].length != coefficient[0].length)
        {
            coef = true;
            return null;
        }
     }

   // checking the error in constant terms part
   for( int i = 0; i < remain.length; i++)
   {
        if(remain[i].length != 1)
        {
            vectorcheck = true;
            return null;
        }
  }

  // checking the length of row in coefficient A is equal to length of row of constant
  //terms B
  if(coefficient.length != remain.length)
  {
        errorchecking = true;
        retirm null;
  }

  // checking the number of variables is more than constant terms
  for( int i = 0; i < coefficient.length; i++)
  {
       if( coefficient[i].length != remain.length)
       {
              rowcheck = true;
              return null;
```

```
        }
    }

    // checking the determinant of coefficient
    if(determinant(coefficient) == 0)
    {
        geterror = true;
    }



    // if coefficients part and constant terms part are both correct
    for (int i = 0; i < coefficient[0].length; i++)
    {
         for(int j = 0; j < coefficient[0].length; j++)
        {
            for(int k = 0; k <coefficient[0].length;k++)
            {
                temp[j][k] = coefficient[j][k];
            }
        }
        result[i][0] = determinant (temp)/determinant(coefficient);
     }
    return result;
    }
```

6.4.12 Gauss-Jordan Elimination

Gauss-Jordan Elimination is another method to solve linear system of equations; the precondition from formal specification is to solve the variables of equations. With the constant terms together, it will form augmented matrix to be solved into the reduced row echelon form. In the design, it should detect errors by error handling. 'pseudo' code provided below:

```
double[][] gaussjordanelimination(double[][] coefficient, double[][] constant)
{       boolean coef = false;
        vectorcheck = false;
        errorchecking = false;
        double[][] augmented = new double[coefficient.length][coefficient[0].length+1];
        double[][] result = new double[coefficient.length][coefficient[0].length+1];

         // check the error in the coefficient part
        // if the number of columns of any row is different from number of the first row
        // then return null.
        for(int i=0; i < coefficient.length; i++)
        {
                if( coefficient[i].length != coefficient[0].length)
                {
                    coef = true;
                    return null;
```

```
            }
        }
    // it is going to check the error in the constant part
    // if the number of column in each row is not equal to one
    // then return null;

    for(int i=0; i<constant.length;i++)
    {
            if(the number of columns in each != 1)
            {
                    vectorcheck = true;
                    return null;
             }
     }

     // if the number of rows in coefficient is not equal to
     // the number of rows in constant term.
     // the result should return null.

     if(number of rows of coefficient != number of rows of constant term)
     {
                     errorchecking = true;
                     return null;
     }

     // define the row and column
     int row = augmented.length;
     int column = augmented[0].length;

     // assign the user input of coefficient and constant
     for(the number of rows in coefficients)
     {
            for( the number of columns in coefficients part)
            {
                    augmented[i][j] = coefficient[i][j];
             }
     }

      for( the number of rows in constant)
      {
             Augmented[i][column-1] =  constant[i][0];
      }


// checking the leading 1 of the first row
        for(int i =0; i < column.length; i++)
        {
              if(the element of first row is not 1)
              {
                the first row / the number is at [0][0] position;
```

```
                }
            }

        // delete the element below the leading one of the first row
          for(int i=0; i<row.length;i++)
       {
          Set the number below the leading equals to zero.
       }

   // for next row, find the leading 1, if it is not, divide by itself
        for(int i=0;i<column.length;i++)
        {
          if (the leading number of second is not 1)
          {
               The second row will be divided the leading number;
          }
        }
    for(int i=0;  i<row.length; i++)
    {
       For(int j=0; j<column.length; j++)
       {
       if( the leading number of second row is at right of leading number of third row)
       {
            swap the second row with third row;
       }
       }
    }

   // carry on sort the leading 1
   for(int i = 0; i < row.length;i++)
   {
        for(int j = 0; j < column.length; j++)
        {
             sort leading one;
        }
   }

   result = reduced echelon form;
   }
}
```

6.5 Implementation of Interface

User interface implementation is another major point should be mentioned. The core part in interface is how to read users' input from the interface.

6.5.1 How to read users' input from the interface

One Java method should be incorporated into the implementation, which is called StringTokenizer method. It could break a string into tokens.

```
// users input could be read from the interface by the token forms
for(int i=0; i<matrixArownumber; i++)
{
    StringTokenizer st = new StringTokenizer(matrixAtextline[i]);

    int linecolumn = st.countTokens();
    matrixAarray[i] = new double[linecolumn];

        while(st.hasMoreTokens())
        {
                for(int j=0; j<linecolumn; j++)
                {
                 matrixAarray[i][j] = Double.parseDouble(st.nextToken());
                }
        }
}
```

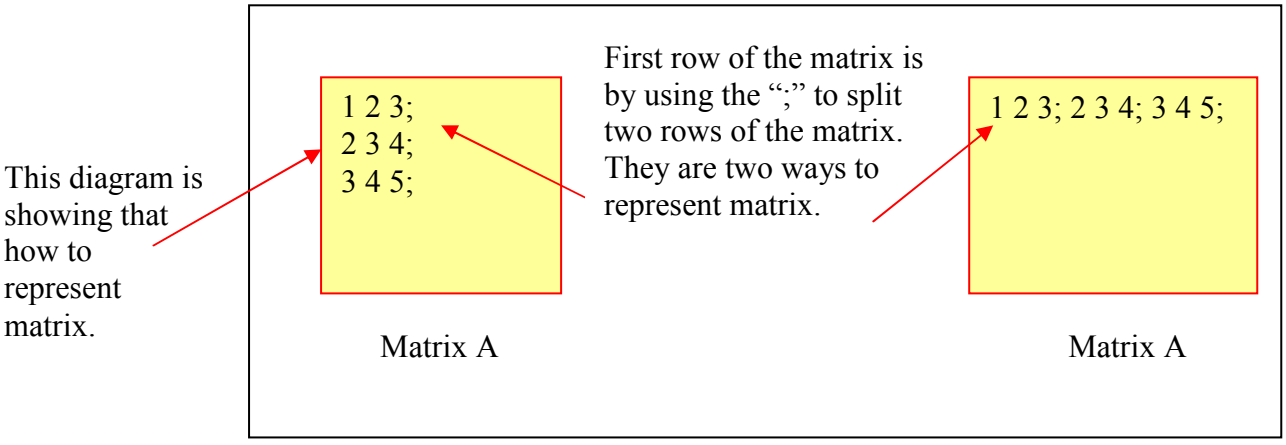


Figure 16 How to represent a Matrix

From figure15, the matrix can be read as Matrix A = $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$

7. System Testing

7.1 Introduction

This chapter will outline the testing and test plans that will be used to evaluate the computer linear algebra system. Testing is defined as executing the system with the purpose of finding the errors. For the testing is to be effective it will need to be planned ahead. The testing is usually divided into three sections: unit testing, integration testing and system testing.

7.2 Testing Process

7.2.1 Unit Testing

The unit testing isolates each individual unit within the system. Each unit will be tested. This method consists of two main stages:

1) Black box testing: It is also called functional testing. Normally it is used in computer programming to check that the outputs of a program, a specific set of inputs, conform to the functional specification of the program. "The black box indicates that the internal implementation of the program being executed is not examined by the tester. For this reason black box testing is not carried out by the designer or programmer of the system." (WIKIPEDIA. The Free Encyclopedia. 2005).
2) White box testing: It is also called structural testing and normally used in computer programming to check that the outputs of the system. Given certain inputs, conform to the internal design and implementation of the program. "The term white box indicates that the tester closely examines the internal implementation of the program being tested." (WIKIPEDIA. The Free Encyclopedia. 2005).

7.2.2 Integration Testing

By the definition, the integration testing is the phase of software testing in which individual software modules are combined and tested as a group. The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items (WIKIPEDIA. The Free Encyclopedia. 2005).

7.2.3 System Testing

System testing is designed to ensure that the software system meets its original requirements. Also, by the definition, "system testing is the phase of software testing is which the complete system is tested "(WIKIPEDIA. The Free Encyclopedia. 2005).

7.3 Validation of Requirements

This part will show all the functional requirements will be tested. It is included so that the system can validated against the requirements produced. Each requirement is detailed with two possible tests.

1) Positive Test, Test to check that the unit does what it is expected to do.
2) Negative Test, Test to check that the unit does not do anything that it is not specified to do.

With each requirement having these tests that can be run the system can be tested against individual requirements. Moreover, this makes it easy to trace a test failure back and find out which requirement is not successfully being met. This will check whether the systems valid to the requirements.

7.3.1 Functional Requirements Testing

The following tables display the functional requirement tests for each unit.

Requirement:	(1) Matrix Addition
Positive test:	Input two matrices of being the same size, the result should return.
Negative test:	Input two matrices of being not the same size. If the system crashes or something else happen, then the system is defective.

Requirement:	(2) Matrix Subtraction
Positive test:	Input two matrices of being the same size, the result should return.
Negative test:	Input two matrices of being not the same size. If the system crashes or something else happen, then the system is defective.

Requirement:	(3) Matrix Scalar
Positive test:	Input a matrix and scalar, the result should return.
Negative test:	Input an incorrect matrix with a scalar, if the system has no response or crashes. Then it should be defective of the system.

Requirement:	(4) Matrix Multiplication
Positive test:	Input two matrices, the columns of first matrix are equal to the rows of second matrix. Then the result should return.
Negative test:	Input two matrices and the size of them are not matched. If there is no error message return or the system crashes, then it should be defective of the system.

Requirement:	(5) Matrix Transpose
Positive test:	Input a single matrix, then the result of it being transposed return.
Negative test:	Input an incorrect matrix, if the system has no response or crashes. It should be defective of the system.

Requirement:	(6) Matrix Trace
Positive test:	Input a square matrix, the trace of it should return.
Negative test:	Input a non-square matrix, if the system has no response or crashes or it returns the result for the matrix, then bugs should be fixed, or it is a defective of the system.

Requirement:	(7) Matrix Determinant 3 by 3
Positive test:	Input a 3 by 3 square matrix, the determinant of the matrix double return if it has.
Negative test:	Input a non 3 by 3 square matrix, if the system does not return the error messages or it crashes or it has no response on this request. Bugs are there should be fixed.

Requirement:	(8) Matrix Determinant
Positive test:	Input a square matrix, the determinant of the matrix should return if it has.
Negative test:	Input a non-square matrix, if the system does not return the error messages or it crashes or it has no response on the request. Bugs exist in the system, they should be fixed.

Requirement:	(9) Matrix Inverse
Positive test:	Input a non-singular and square matrix, the inverse for the matrix should return.
Negative test:	Firstly, input a singular but square matrix, if the system can return the result. It should be defective of the system. Secondly, input a non-square but non-singular matrix. If the system does not return the error messages or crashes. Bugs should be fixed in this case. Thirdly, input a non-square and singular matrix, if the system returns the result or does not return error messages or no response on the request. Then the system has bugs needs to be fixed.

Requirement:	(10) Matrix Ad-joint
Positive test:	Input a square matrix, the ad-joint of the square matrix should return.
Negative test:	Input a non-square matrix, or flawed matrix or characters, if the system doses not return the error messages or it crashes or it has no response. Bugs should be fixed in the system.

Requirement:	(11) Cramer's Rule
Positive test:	Input a proper augmented matrix, the result should return if the equations have unique solution for each variable.
Negative test:	Input flawed data into the system, if the error message does not return, or it crashes or it has no response, then bugs are in the system needed to be fixed.

Requirement:	(12) Gauss-Jordan Elimination
Positive test:	Input a proper augmented matrix, the result should return.
Negative test:	Input incorrect data into the system, if the error message does not return, or it crashes or it has no response, then bugs are in the system needed to be fixed.

Requirement:	(13) Save function
Positive test:	Input data to save, the system is able to save the matrix for purpose.
Negative test:	Leave the interface empty, then pressing save button, if the error message does not return, or it could save empty file, then bugs are in the system should be dealt with.

Requirement:	(14) Load function
Positive test:	Load the file from a directory, then the interface of the system could display the content of that file.
Negative test:	If it cannot load the content of the file into the system, or it has no response, then bugs are in the system.

7.4 Unit Testing

7.4.1 Black Box Testing

Black-box test design treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements. This section is going to test the interface by using black box testing; it will check the request of the users will be working correctly against the requirements specification. Giving each one a status of "pass" or "fail", depending on whether it has been met the requirement or not.

Action	Expected Result	Actual Result	Status
Choosing matrix addition operation	It will display matrix addition state	Yes, it displays the matrix addition state	Pass
Choosing matrix subtraction	It will display matrix subtraction state	Yes, it displays the matrix subtraction state	Pass
Choosing matrix multiplication operation and scalar	It will display matrix multiplication state.	Yes, it displays the matrix multiplication state	Pass
Choosing matrix trace operation, matrix inverse operation, matrix determinant operation, matrix ad-joint operation, Cramer's rule operation	The system will display each operation state as a response.	Yes, it could display each operation state as a response	Pass
Users would like to save the matrix. They input matrices into corresponding locations, then pressing save	The system is able to save the matrices for purpose.	Yes, it works as it is supposed to do.	Pass
Users would like to load the content of a file. They press the load button to enter the specific directory to load the content of file.	The system could be able to load the contents of a file	Yes, it works as it is supposed to do.	Pass
Entering help function	The system will display the help contents for users	Yes, it works.	Pass
Two Matrices addition whose sizes are matched	The system will return the result.	Yes, result returns.	Pass
Two matrices subtraction whose sizes are matched.	The system will return the result.	Yes, result returns.	Pass
Input two matrices for multiplication, the columns of matrix A are matched the rows of matrix B.	It will return the result.	Yes, it returns the result.	Pass

Compute the determinant of square matrix	It could return the result	Yes, it could return the result.	Pass
Compute the trace of square matrix	It could return the result.	Yes, it could return the result of the trace of the square matrix	Pass
Compute the inverse of square matrix	It could return the result.	Yes, it could return the result as expected.	Pass
Compute the ad-joint of square matrix	The system could return the result	Yes, it could return it.	Pass
Cramer's rule computation	It could return the result for each variable if the determinant of coefficients is not zero	Yes, it returns unique result for each variable.	Pass

The interface has been passed all the black box tests, it has met the requirements has been specified in the functional requirements in chapter 3.

7.5 White Box Testing

White box testing is also called structural testing, which include designing tests such that every line of code is executed at least once, or requiring every function to be individually tested. Again, a formal table of testing format will be constructed.

Code being tested	Working	If not working Reason for it	Error handling for matrix size or two matrices sizes are not appropriate	Status
Codes for matrix addition	Yes.	None	Yes	Pass
Codes for matrix subtraction	Yes	None	Yes	Pass
Codes for matrix multiplication and scalar	Yes	None	Yes	Pass
Codes for matrix determinant	Yes	None	Yes	Pass

Codes for matrix inverse	Yes	None	Yes	Pass
Codes for matrix trace	Yes	None	Yes	Pass
Codes for matrix ad-joint	Yes	None	Yes	Pass
Codes for Cramer's rule	Yes	None	Yes	Pass

Each operation is fully tested when coding was produced, and each matrix operation has included error handling for checking matrix size or two matrices are not appropriate.

7.6 Integration testing

Integration testing is a logical extension of unit testing. It could be explained in a simple way, test two combined units. A component should refer to an integrated aggregate of more than one unit. Integration testing identifies problems that occur when units are combined. In the case of this project, the integration testing has been fully implemented when designer was coding the program.

7.7 System Testing

The system testing is aim to test the system is met the original requirements which have been stated in the validation of requirements. Also, all the other features involved in each function should be fully tested to ensure the system run robustness. Here each requirement individually, also giving each one a status of "pass" or "fail", depending on whether it has been met the requirement or not. Furthermore, testing plan will fully follow 1) positive testing and 2) negative testing to see what response of the system.

7.7.1 Matrix Addition Testing

Matrix A	Matrix B	Expected Result	Actual Result	Status
1 4 7; 0 -2 3;	2 5 -6; -3 1 8;	3.0 9.0 1.0; -3.0 -1.0 11.0;	3.0 9.0 1.0; -3.0 -1.0 11.0;	Pass
1 4 7; 0 -2 3;	-5 4; 2 7;	Incorrect, Check the matrices size.	Incorrect Data Input! Please Check Matrix Size!	Pass
1 4 7; 43 3;	None	Incorrect matrix input for matrix A, and input Matrix B.	Incorrect Data Input! Please Check Matrix Size! Please Input Matrix B!	Pass

“fsafds”	“fsajfkdsa”	Incorrect data.	Incorrect Data Input! Please Check!	Pass
None	None	Input matrix A and matrix B.	Please Input Matrices!	Pass
1 4 7; 0 -2 3;	None	Input matrix B	Please Input Matrix B!	Pass
None	1 4 7; 43 3;	Incorrect matrix input for matrix B, and input matrix A	Incorrect Data Input! Please Check Matrix Size! Please Input Matrix A!	Pass
“fdsaf”	None	Incorrect data input in matrix A and input matrix B	Incorrect Data Input In Matrix A! Please Check! Please Input Matrix B!	Pass
None	“fdsajlf”	Incorrect data input in matrix B and input matrix A	Incorrect Data Input In Matrix B! Please Check! Please Input Matrix A!	Pass
1 4 7; 43 3 ff;	43 54 43; 54 5 3;	Incorrect data input.	Incorrect Data Input! Please Check!	Pass
43 54 43; 54 5 3;	1 4 7; 43 3 ff;	Incorrect data input.	Incorrect Data Input! Please Check!	Pass

7.7.2 Matrix Subtraction Testing

Matrix A	Matrix B	Expected Result	Actual Result	Status
1 4 7; 0 -2 3;	2 5 -6; -3 1 8;	-1.0 -1.0 13.0; 3.0 -3.0 -5.0;	-1.0 -1.0 13.0; 3.0 -3.0 -5.0;	Pass
1 4 7; 0 -2 3;	-5 4; 2 7;	Incorrect, Check the matrices size.	Incorrect Data Input! Please Check Matrix Size!	Pass
1 4 7; 43 3;	None	Incorrect matrix input for matrix A, and input Matrix B.	Incorrect Data Input! Please Check Matrix Size! Please Input Matrix B!	Pass

“fsafds”	“fsajfkdsa”	Incorrect data.	Incorrect Data Input! Please Check!	Pass
None	None	Input matrix A and matrix B	Please Input Matrices!	Pass
“fdlsafj”	None	Incorrect data input in matrix, and input matrix B.	Incorrect Data Input In Matrix A! Please Check! Please Input Matrix B!	Pass
None	“fdlsaj”	Incorrect data input in matrix B and input matrix A.	Incorrect Data Input In Matrix B! Please Check! Please Input Matrix A!	Pass
1 4 7; 43 3 fd;	43 3 5; 54 3 2;	Incorrect data input.	Incorrect Data Input! Please Check!	Pass
43 3 5; 54 3 2;	1 4 7; 43 3 fd;	Incorrect data input	Incorrect Data Input! Please Check!	Pass

7.7.3 Matrix Scalar Testing

Matrix A	Scalar	Expected Result	Actual Result	Status
1 2 3; 4 5 6;	2	2.0 4.0 6.0; 8.0 10.0 12.0	2.0 4.0 6.0; 8.0 10.0 12.0	Pass
43 43 54; 34 54;	None	Incorrect data input!	Incorrect Data Input! Please Check!	Pass
32 4 23; 43 54 4;	None	Input a scalar.	Please Input a Scalar!	Pass
None	3	Input matrix A	Please input Matrix A	Pass
None	54 43 54;	Incorrect data input.	Incorrect data input! Please Check!	Pass
None	None	Input a scalar and a matrix	Please Input Matrix A and a Scalar	Pass
“fsafasd”	None	Incorrect data input.	Incorrect data input! Please Check!	Pass

None	“fdsalfha”	Incorrect data input.	Incorrect data input! Please Check!	Pass
“fdsalfjdasl”	“fhkafdsa”	Incorrect data input.	Incorrect data input! Please Check!	Pass
43 437 54; 34 54f 43;	4	Incorrect data input	Incorrect data input! Please Check!	Pass
43 544 3; 54 43 5;	43 fd;	Incorrect data input.	Incorrect data input! Please Check!	Pass
43 54 43; 43 54 fd;	54 f;	Incorrect data input.	Incorrect data input! Please Check!	Pass

7.7.4 Matrix Multiplication Testing

Matrix A	Matrix B	Expected Result	Actual Result	Status
2 1; 7 0; -3 -2;	-1 0; 3 5;	1.0 5.0; -7.0 0.0; -3.0 -10.0;	1.0 5.0; -7.0 0.0; -3.0 -10.0;	Pass
2 1 -3; 4 5 8;	2 1; -3 4;	Incorrect input .Matrices sizes are not matched.	Incorrect Data Input! Please Check Matrix Size!	Pass
2 1; 4 5 8;	2 1; -3 4;	Incorrect, check the size of matrix.	Incorrect Data Input! Please Check Matrix Size!	Pass
“fsadfdsa”	None	Incorrect data input in matrix A, input matrix B.	Incorrect Data Input In Matrix A! Please Check! Please Input Matrix B!	Pass
None	None	Input matrix A and matrix B.	Please Input Matrices!	Pass
None	“fasfdsa”	Incorrect data input in matrix B, and input matrix A.	Incorrect Data Input In Matrix B! Please Check! Please Input Matrix A!	Pass
“fsdafdfds”	“fdsalfjas”	Incorrect data input.	Incorrect Data Input! Please Check!	Pass

43 54 43; 43 65 43; 4 54 3fd;	54; 43; 65;	Incorrect data input.	Incorrect Data Input! Please Check!	Pass
54 43 5; 43 5 34; 43 54 43;	43; 54; fds;	Incorrect data input.	Incorrect Data Input! Please Check!	Pass

7.7.5 Matrix Transpose Testing

Matrix A	Expected Result	Actual Result	Status
43 54 4; 45 5 6;	43.0 45.0; 54.0 5.0; 4.0 6.0;	43.0 45.0; 54.0 5.0; 4.0 6.0;	Pass
43 54 43; 5 43 54; 4 64;	The size of matrix is incorrect.	Incorrect data input! Please Check Matrix Size!	Pass
“fsdafsa”	Incorrect data input.	Incorrect data input! Please Check!	Pass
None	Input matrix A.	Please input Matrix A!	Pass
43 5; 4 fds;	Incorrect data input.	Incorrect data input! Please Check!	Pass

7.7.6 Matrix Trace Testing

Matrix A	Expected Result	Actual Result	Status
43 54 4; 43 54 43; 54 65 34;	131.0	131.0	Pass
43 54 4; 43 54 43;	Square Matrix only.	Incorrect data input! Square Matrix Only!	Pass
None	Input matrix A.	Please input Matrix A!	Pass
“fdjsalfj”	Incorrect data input.	Incorrect data input! Please Check!	Pass
43 54 43; 43 54;	Incorrect matrix size.	Incorrect data input! Square Matrix Only!	Pass
54 43 54; 54 56f 43;	Incorrect data Input.	Incorrect data input! Please Check!	Pass

7.7.7 Matrix Determinant Testing

Matrix A	Expected Result	Actual Result	Status
5 3 5; 2 3 5; 2 6 78;	612.0;	612.0;	Pass
43 54 4; 43 54 43;	Square matrix only.	Incorrect data input! Square Matrix Only!	Pass
1 1; 1 1;	0.0;	0.0;	Pass
None	Input matrix A.	Please input Matrix A!	Pass
"fdsadf"	Incorrect data input	Incorrect data input!	Pass
43 43f 54; 54 43 54;	Incorrect data input.	Incorrect data input! Please Check!	Pass
432 43 43; 54 65;	Incorrect data input.	Incorrect data input! Square Matrix Only!	Pass

7.7.8 Matrix Determinant 3 by 3 Testing

Matrix A	Expected Result	Actual Result	Status
5 3 5; 2 3 5; 2 6 78;	612.0;	612.0;	Pass
5 3 5; 2 3 5;	3 by 3 square matrix only.	Incorrect data input! 3 by 3 square matrix only!	Pass
1 1 1; 1 1 1; 1 1 1;	0.0	0.0	Pass
None	Input matrix A	Please input matrix A	Pass
"fdsaf"	Incorrect data input.	Incorrect data input! Please Check!	Pass
343 543 4; 43 54;	Incorrect data input.	Incorrect data input! 3 by 3 Square Matrix Only!	Pass
34 54 4ffsa; 54 43 5;	Incorrect data input.	Incorrect data input! Please Check!	Pass

7.7.9 Matrix Inverse Testing

Matrix A	Expected Result	Actual Result	Status
-2 1; 1.5 -0.5;	1.0 2.0; 3.0 4.0;	1.0 2.0; 3.0 4.0;	Pass
43 32 4; 32 53;	Not a square matrix	Incorrect data input! Square Matrix Only!	Pass
1 1; 1 1;	No inverse	Error, Singular Matrix! No Inverse Exists!	Pass
None	Input a matrix.	Please input Matrix A!	Pass
43 32 4; 43 54 fd;	Incorrect data input.	Incorrect data input! Please Check!	Pass
"fdslafd"	Incorrect data input.	Incorrect data input! Please Check!	Pass

7.7.10 Matrix Ad-joint Testing

Matrix A	Expected Result	Actual Result	Status
2 0 3; -1 4 -2; 1 -3 5;	14.0 -9.0 -12.0; 3.0 7.0 1.0; -1.0 6.0 8.0;	14.0 -9.0 -12.0; 3.0 7.0 1.0; -1.0 6.0 8.0;	Pass
None	Input a matrix	Please input Matrix A!	Pass
"fdsaf"	Incorrect data input!	Incorrect data input! Please Check!	Pass
32 43 32; 43 54 43;	Not square matrix.	Incorrect data input! Square Matrix Only!	Pass
32 43 32; 43 54 43; 54 43 fds;	Incorrect data input.	Incorrect data input! Please Check!	Pass

7.7.11 Cramer's Rule

Coefficients A	Constant Terms B	Expected Result	Actual Result	Status
1 3 1; 2 5 1; 1 2 3;	-2; -5; 6;	$x_1 = 1.0$ $x_2 = -2.0$ $x_3 = 3.0$	1.0; -2.0; 3.0;	Pass
1 2 3; 2 1 3; 1 1 2;	3; 3; 0;	No solutions	No Unique Solution!	Pass

1 -2 3; 3 -4 5; 2 -3 4;	1; 3; 2;	Many solutions	No Unique Solution!	Pass
32 43 32; 43 32;	None	Incorrect data input.	Error Input for Coefficients A!	Pass
None	432 3; 43 54;	Incorrect data input.	Error Input for Constant Terms B!	Pass
None	None	Input coefficients A and Constant Terms B.	Please Input Coefficients A and Constant Terms B of Equations!	Pass
“fdsafd”	None	Incorrect data input.	Incorrect Data Input in Coefficients A! Please Check! Please Input Constant Terms B!	Pass
None	“fdsaf”	Incorrect data input.	Incorrect data input in Constant Terms B! Please Check! Please Input Coefficient A!	Pass
“fsafsd”	“fds”	Incorrect data input.	Incorrect data input! Please Check!	Pass
43 32 32; 43 354 34;	“fdsfasd”	Incorrect data input.	Incorrect Data input! Please Check!	Pass
43 32 43; 5 43 54f;	3; 54;	Incorrect data input.	Incorrect Data input! Please Check!	Pass
43 54 4; 54 43 4;	43; 4f;	Incorrect data input.	Incorrect Data input! Please Check!	Pass
32 43 32; 43 54 3;	43;	Incorrect data input.	Incorrect Data Input! Please Check the Size!	Pass
32 43 32; 43 54 3;	43; 54; 65; 76; 65;	Incorrect data input.	Incorrect Data Input! Please Check the Size!	Pass

7.7.12 Gauss-Jordan Elimination

Coefficients A	Constant Terms B	Expected Result	Actual Result	Status
0 0 2 -2; 3 3 -3 9; 4 4 -2 11;	2; 12; 12;	1.0 1.0 0.0 0.0 17.0; 0.0 0.0 1.0 0.0 -5.0; 0.0 0.0 0.0 1.0 -6.0;	1.0 1.0 0.0 0.0 17.0; 0.0 0.0 1.0 0.0 -5.0; 0.0 0.0 0.0 1.0 -6.0;	Pass
3 -3 3; 2 -1 4; 3 -5 -1;	9; 7; 7;	1.0 0.0 3.0 4.0; 0.0 1.0 2.0 1.0; 0.0 0.0 0.0 0.0;	1.0 0.0 3.0 4.0; 0.0 1.0 2.0 1.0; 0.0 0.0 0.0 0.0;	Pass
3 -3 3; 2 -1 4; 3 -5 -1;	None	input constant terms	Please Input Constant Terms B!	Pass
1 2 3;	4;	1 2 3 4;	1 2 3 4;	Pass
43 32 54;	43; 54;	Incorrect data input	Incorrect Data Input! Please Check the Size!	Pass
None	None	Input augmented matrix.	Please Input Coefficients A and Constant Terms B of Equations!	Pass
“fafdsa”	None	Incorrect data input, input constant terms B.	Incorrect Data Input In Coefficient A! Please Check! Please Input Constant Terms B!	Pass
None	“fsfsa”	Incorrect data input, and input coefficients A.	Incorrect Data Input In Constant Terms B! Please Check! Please Input Coefficient A !	Pass
“fasfdsa”	“fafdas”	Incorrect data input, you should check.	Incorrect Data Input! Please Check!	Pass
32 43 32 45; 32 5 3;	None	Incorrect input in coefficients A.	Error Input for Coefficients A!	Pass
453 54 32 5; 324 54 32 fds;	12; 14;	Incorrect data input, you should check.	Incorrect Data Input! Please Check!	Pass

7.7.13 Save Function

User Input	Expected Result	Actual Result	Status
32 43 3; 32 43 2; 43 54 6;	It could be saved.	Yes, it could be saved.	Pass
43 54 54; 43 54 4; 3 5 4; The comments of this matrix: (This is the matrix for matrix addition!)	It could be saved.	Yes, it could be saved.	Pass
None! Empty Input!	It could not be saved.	Yes, the error message will return if the interface is empty.	Pass

7.7.14 Load Function

Load User Input	Expected Result	Actual Result	Status
Load the file of user input from a specific directory	It could load the content of that specific file.	Yes, it could load the content of that specific file.	Pass

7.8 Conclusion of Testing

From the results of the testing, a good coverage testing has implemented. All the functionalities of the system have been tested against the functional requirements.
Not only these functional requirements have been fully achieved, but other features within the system also have been fully achieved. Other features include 1) what response if an invalid number is input? 2) What response if undesired data are input? 3) What response if the matrix size is not appropriate or two matrices sizes are not matched? 4) What response for a non-square matrix input when the system requires a square matrix? All of these features within the system have been totally fulfilled.

8. Critical Evaluation

8.1 Introduction

In this chapter, a general evaluation on the linear algebra system should be made. The system has met all the original functional requirements and requirements gained from existing systems. In chapter literature survey, there were three existing systems had been mentioned, comparing to these systems, what improvement did for this new design will be a main points to be evaluated.

8.2 Evaluation

The system has fully fulfilled the requirements that gained from literature survey and existing systems. It covers more functions than existing system but except Maple system. Also, the system has no constraints on the size on the matrix input.

8.2.1 What has the system fulfilled?

The goal in the new design was trying to improve the constraints within the existing systems. This project has fully overcome the constraints from above Java Matrix Calculator and previous master student's work. The system is able to support both non-square and square matrix. Also, the current state has been provided for the users in order to reduce errors being made. In terms of the error messages sending, it could provide a specific error that the user made. Functions provided by the system is another important part, this project has covered more functions which are required for matrix computation. Furthermore, users could directly save and load the content of a file from the interface. As broad functionalities provided by Maple system, it requires training time for people who are not familiar with the system. However, the project supplies easy software to apprehend by users that no training time requirements.

8.2.2 Improvement of the system

With respect to improvement of the system, more techniques for each operation could be included. For example, there are two different methods to compute matrix inverse. Doing different techniques for each operation is to find the most efficient algorithm which could reduce response time of the system. Not only this improvement should be considered, but there are also some more algorithms could be involved to improve the functions, such as rank, vector.

8.3 Conclusion of evaluation

In conclusion of this chapter, the system has improved the constraints from the existing systems. Furthermore, the system is straightforward to apprehend and use and it runs robust. Also, it provides two different ways for users input. However, with time constraint, more algorithms could not be developed. Future work in next chapter will detail the algorithms could be developed as further work.

9. Conclusion and Future Work

This section is going to give a conclusion of this dissertation as a whole and the further work related to the project. Further work could be the core part in the section as it is going to provide more aspects and features that can be involved or improved within this project.

9.1 Conclusion

This dissertation is fully based on the formal structure. This dissertation has detailed the process of how to design the software. The algorithm part can be seen as the core of the whole system. Also, functional requirements are fully around the algorithm design. Furthermore, in chapter detailed design and implementation, it shows how to algorithms are implemented by coding. Refer to the system testing is in order to test all the aspect of the system. By the different testing methodologies of the algorithms have been used on the each aspect of the system, the functional requirements on algorithms part are fully achieved. As well as algorithm functional requirements, other functional requirements like help function, save and load functions provided by the system are fully achieved. Furthermore, the interface of the system followed the user interface design principles, and the interface is user-friendly and looks like a normal calculator.

9.2 Future Work on the system

This section is the major part in this chapter, and the future work on the system will be detailed here. With the time constraints, although the system has met to its requirements, a numbers of additions and improvements could be made. These considerations could form the future work upon the system. As mentioned in chapter 7 critical evaluation, more techniques for each operation could be included.

9.2.1 Algorithms improvement

9.2.1.1 Strassen's algorithm is another algorithm for matrix multiplication

For matrix multiplication, there is another method called Strassen's algorithm. Here the detail of Strassen's algorithm will discuss.

For Strassen's algorithm if $\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$,

then

$P_1 = ag - ah = a(g - h)$
$P_2 = ah + bh = (a + b)h$
$P_3 = ce + de = (c + d)e$
$P_4 = df - de = d(f - e)$
$P_5 = ae + ah + de + dh = (a + d)(e + h)$
$P_6 = bf + bh - df - dh = (b - d)(f + h)$
$P_7 = ae + ag - ce - cg = (a - c)(e + g)$
$r = P_5 + P_4 - P_2 + P_6$
$s = P_1 + P_2$
$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7$

As Strassen's formulas use only seven multiplications (18 additions), the divide-and-conquer approach using these formulas immediately leads to the following functional equation for the complexity:

$$T(n) = 7T(n/2) + 18(n^2/4).$$

According to Master Theorem, the solutions is

$$T(n) = \theta(n^{\log 7}) = \theta(n^{2.81\ldots}).$$

Strassen's algorithm could design effective algorithms for matrix multiplication.

9.2.1.2 Gauss-Jordan Elimination for finding the inverse of a Matrix

Let A be an n by n matrix that is square matrix.

1) Ad-joint the identity n by n matrix I_n to A to form the matrix [A: I_n].
2) Compute the reduced echelon form of [A: I_n].

   If the reduced echelon form is of the type [I_n : B], then B is the inverse of A.

   If the reduced echelon form is not of the type [I_n : B], in that the first n by n sub-matrix is not I_n, then A has no inverse.

(Gareth Williams 2001)

9.2.1.3 Another algorithm for matrix determinant

A square matrix is called an upper triangular matrix if all the elements below the main diagonal are zero.

$$A = \begin{pmatrix} 2 & -1 & 9 \\ 0 & 3 & -4 \\ 0 & 0 & -5 \end{pmatrix}$$

This is an upper triangular matrix. The determinant is the product of the diagonal elements.

$|A| = 2 \times 3 \times (-5) = -30$

(Gareth Williams 2001)

Sections 9.2.1, 9.2.2 and 9.2.3 are other methods for matrix multiplication; matrix inverse and matrix determinant respectively which could be developed within the linear algebra system to improve the current algorithms.

9.2.2 Complexity of Algorithms

With respect to complexity of algorithms, it could improve the efficiency of the system. With a lower complexity of the algorithm, response time of the system will be reduced.

9.2.3 Improvement of Other Functions

Refer to other functions; vector and rank of a matrix are other two large parts can be developed within the system to improve the current functions.

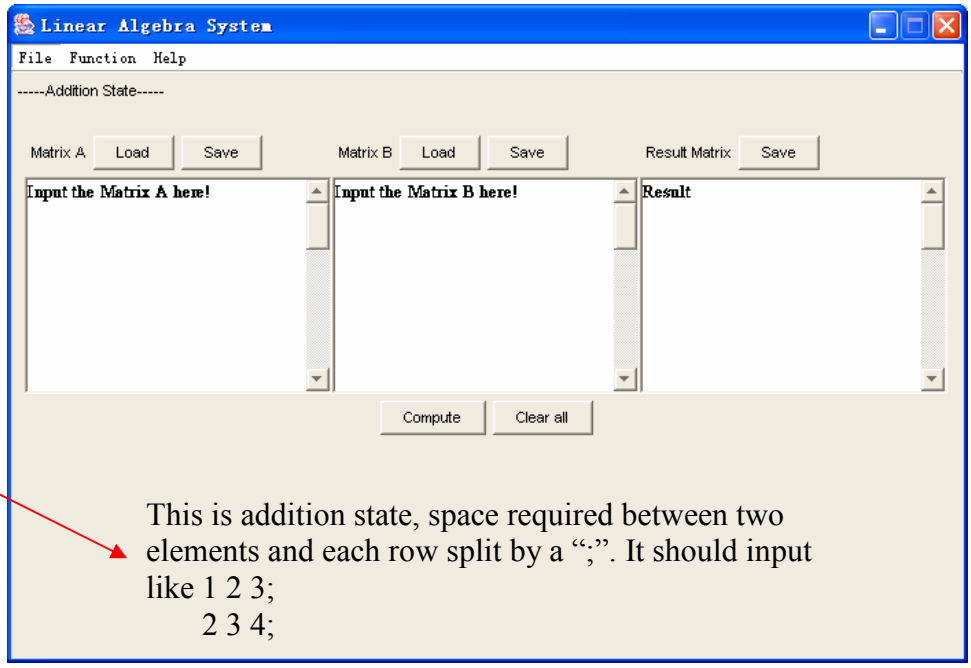
9.2.4 Function improvement

In terms of "Save" function, random characters could be saved rather than a matrix. In future development, the "Save" function should be only able to save the specific matrix for further computing.

9.2.5 Improvement of System Interface

With respect to improvement of system interface, more information should be provided for users without the help function, users should easy to manipulate the system, and it should look like below:

More information could be provided here in order to explain how the current state works



It could make the interface more usable and more understandable. Also, it make the system more efficient.

10. Bibliography

Char, Bruce W., 1992. *First Leaves: a tutorial introduction to Maple V*. Springer-Verlag.

Eric, W, Weisstein., *Determinant*. A Wolfram Web Resource. Available from: http://mathworld.wolfram.com/Determinant.html

Gauss – Jordan Elimination. Available from: http://www.aspire.cs.uah.edu/textbook/gauss.html

Heck, A., 1996. *Introduction to Maple*. 2nd ed. Springer.

Harper, D., 1990. 4th ed. *A Guide to Computer Algebra Systems*. Available from: http://www.inf.ed.ac.uk/teaching/courses/ca/caguide.pdf [Accessed March 1990].

Linear algebra with maple. 2004. Available from: http://www.indiana.edu/~statmath/math/maple/linalg.html [Accessed 21st July 2004].

IX. Maple: Computer Algebra System. Available from: http://www.math.toronto.edu/help/maple99.pdf

M. A. Khamsi., *Systems of Linear Equations: Gaussian Elimination*. Available from: http://www.sosmath.com/matrix/system1/system1.html

Marcus Kazmierczak, 2002. *Java Matrix Calculator*. Available from: http://www.mkaz.com/math/matrix.html [Accessed 13th June 2002].

Pilbeam, B., 2002. *Computer linear algebra system*. Thesis (MSC). University of Bath.

Santamaria-Ortega, Laia., 2002. *Computer linear algebra system*. Thesis (MSC). University of Bath.

Stanley I. Grossman., 1991. 4th ed. *Elementary Linear Algebra*. Sauders College.

Seymour Lipschutz.,1974. Schaum's outline of theory and problems of linear algebra. McGraw-Hill. Chapter 3.

Sommerville, I., 2001. 6th ed. Software Engineering. Addison Wesley.

Towers, David, A., 1988. Guide to linear algebra. Macmillan.

Williams, G., 2001. 4th ed. Linear algebra with applications. Jones and Bartlett.

WIKIPEDIA. The Free Encyclopedia. 2005. *Black box testing.*
Available from: http://en.wikipedia.org/wiki/Black_box_testing
[Accessed: 25th April 2005].

WIKIPEDIA. The Free Encyclopedia. 2005. *White box testing.*
Available from: http://en.wikipedia.org/wiki/White_box_testing
[Accessed: 17th April 2005].

WIKIPEDIA. The Free Encyclopedia. 2005. *Integration Testing.*
Available form: http://en.wikipedia.org/wiki/Integration_testing
[Accessed: 17th April 2005].

WIKIPEDIA. The Free Encyclopedia. 2005. *System testing*.
Available form: http://en.wikipedia.org/wiki/System_testing
[Accessed: 17th April 2005].

11. Appendices

11.1 A Simple User Guide

This section is a user guide for the computer linear algebra system. The first part covers how to install the system into a PC. The next sections describe step-by step methods to perform certain tasks. These are written in a very easy to understand format, and should be read by anyone having difficulty navigating to their desired function.

11.1.1 Functional Description

The computer linear algebra system is a mathematical based system. The system is spilt into several key areas, or 'modules'. A summary of each module's function is illustrated below:

Matrix Operations:

1) Matrix Addition
2) Matrix Subtraction
3) Matrix Scalar
4) Matrix Multiplication
5) Matrix Transpose
6) Matrix Trace
7) Matrix Determinant 3 by 3
8) Matrix Determinant for general size square matrix
9) Matrix Inverse
10) Matrix Ad-joint

System of Linear Equations

1) Gauss-Jordan Elimination
2) Cramer's Rule

11.1.2 Installation Guide

This system is written in Java programming language. PCs within Bath University provided Unix System and Terms which could be both using for the system. The system was fully compiled; it could be run by writing "java drive". Before run java program, ensure your system has got Java JDK Platform for supporting Java Languages. It could be found from: (http://java.sun.com/j2se/1.5.0/download.jsp)

11.1.3 User Manual

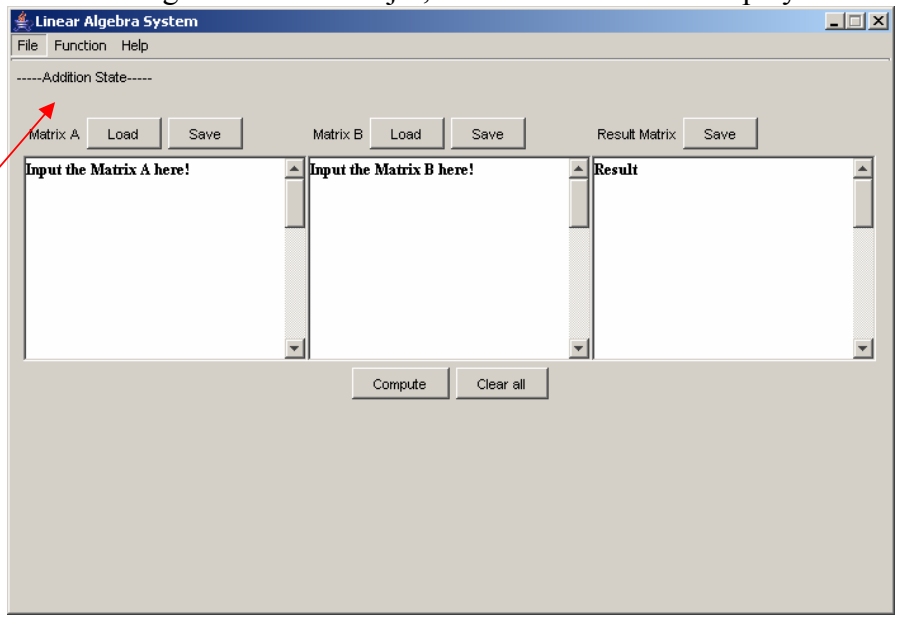
This section of the user guide deals with the tasks you wish to perform. You could use this as a "how-to" guide if you having problems with how to perform a task.

11.1.3.1 Display the Screen

Step-by-Step Guide: This is the pre-step before to access any functions, users have to do these steps:

1) Turn on the computer
2) Log in to windows
3) Run the program
4) By double clicking the XiaoweiXu.jar, the main screen will display as below:

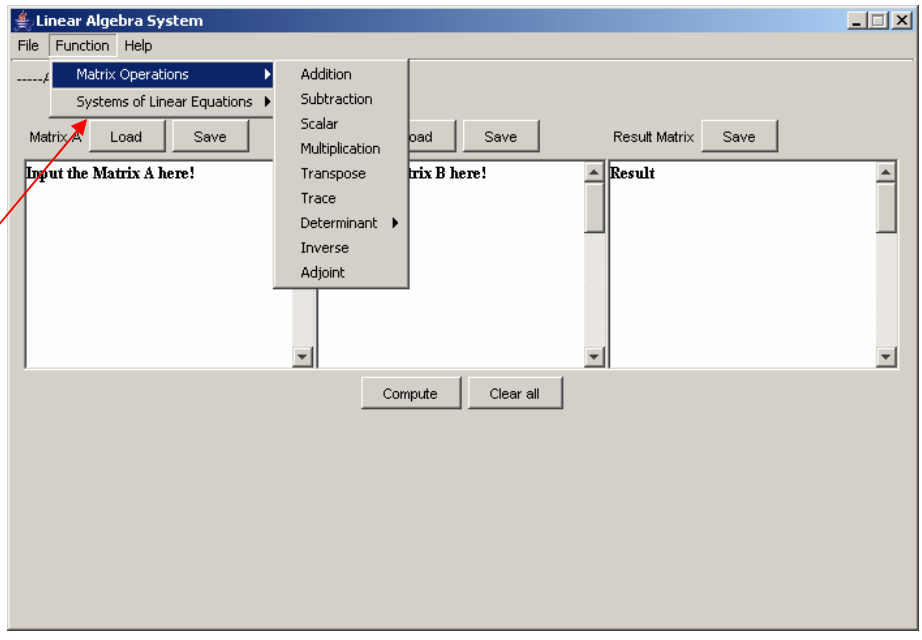
Current State

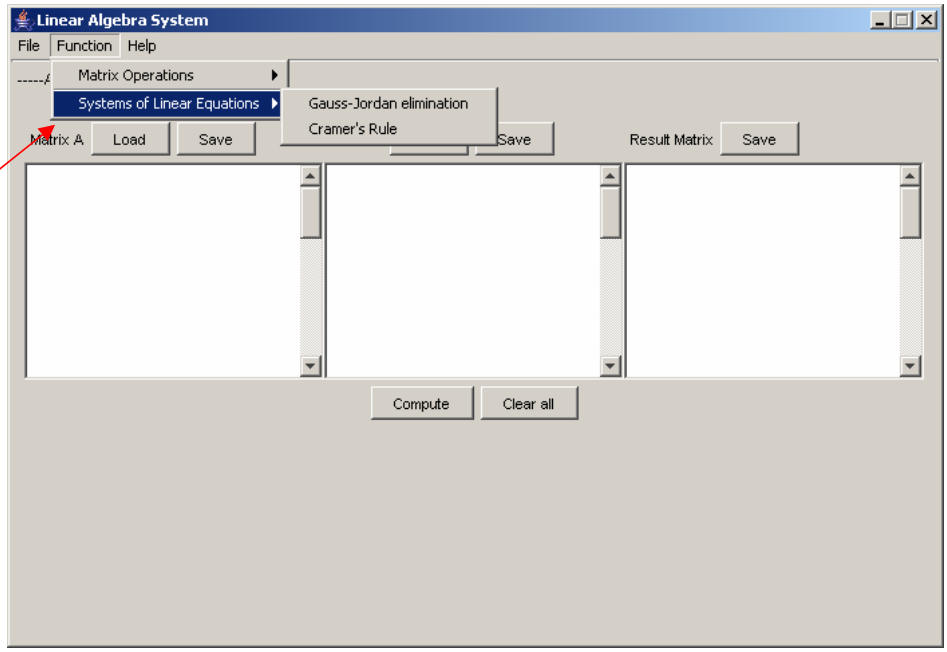


Here is the main screen of computer linear algebra system; and it is in addition state as default.

11.1.3.2 Applications could be chosen

Matrix Operations

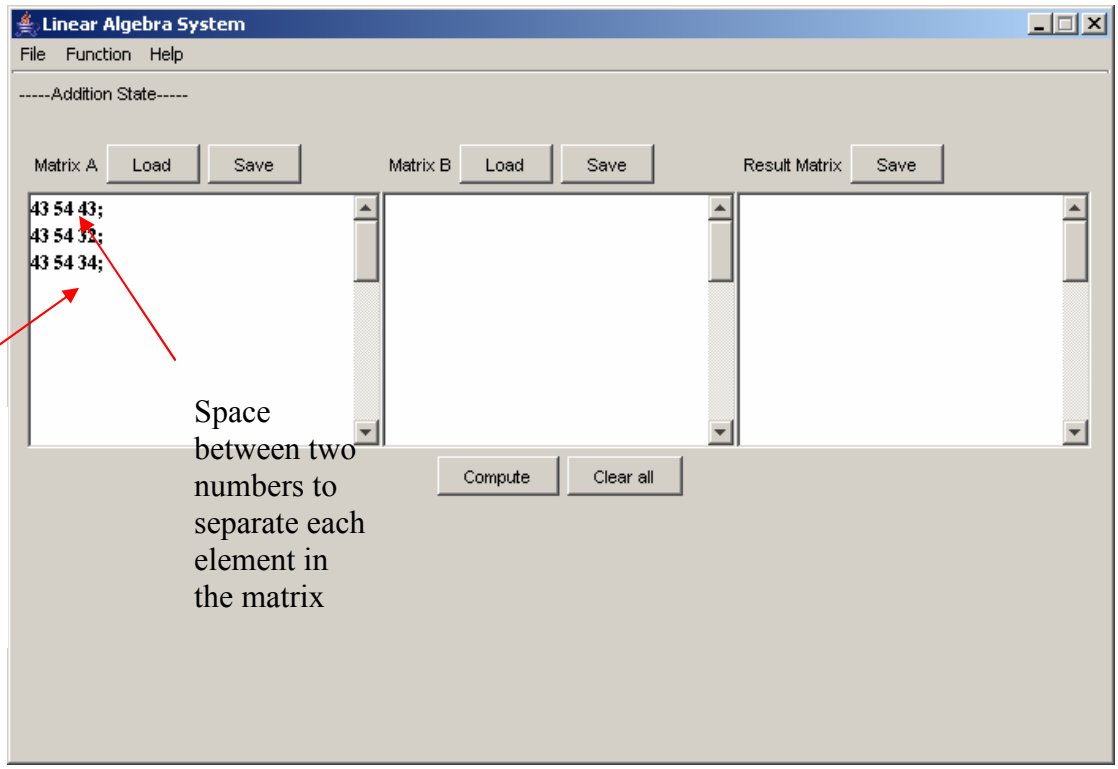




System of Linear Equations

11.1.3.3 How to input the matrix

Users should input matrix like "43 32 43;" using semi-colon to separate two different rows and between each number a space required. See below screen to gain an idea.



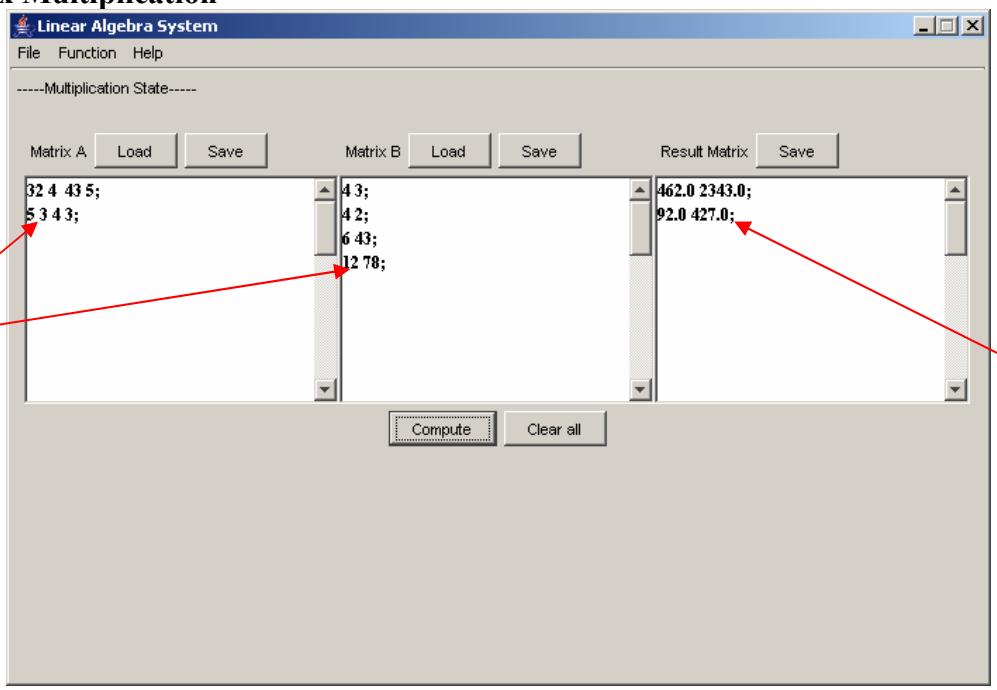
Semi-colon ";" required for separating different rows of matrix

Space between two numbers to separate each element in the matrix

11.1.3.4 Matrix Application

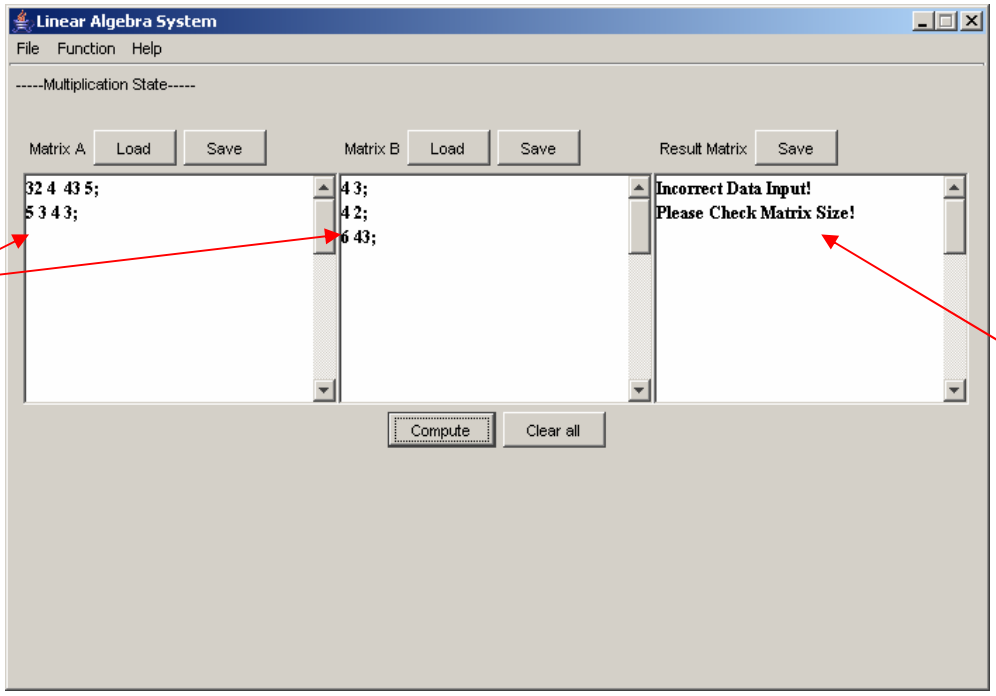
For each matrix application, how each works should be considered here, and how the system deals with error handling for the applications. Here we could take matrix multiplication and matrix determinant for example to gain an overview of how the system works.

Matrix Multiplication



Columns of matrix A and rows of matrix B are matched.

Result returns

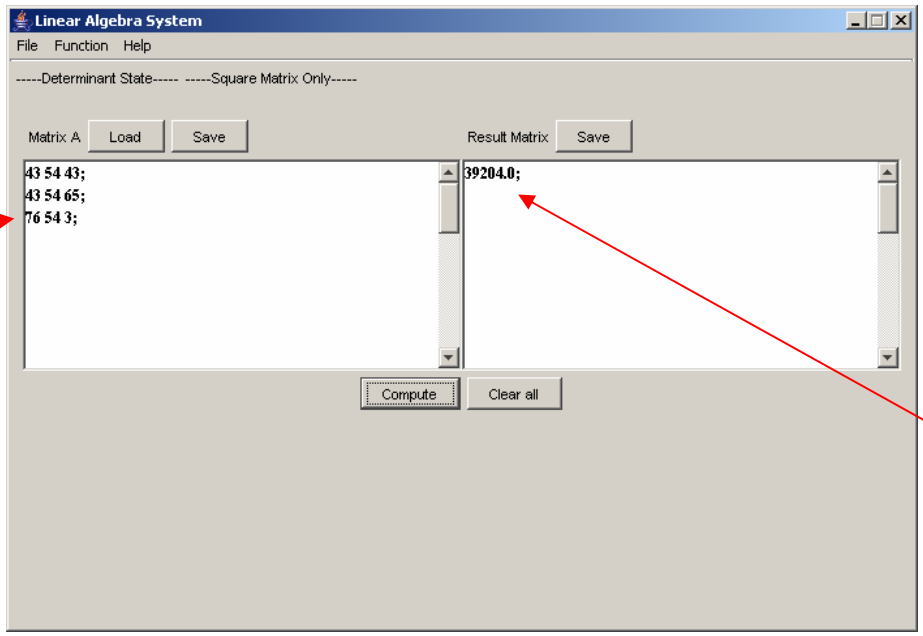


Columns of matrix A and rows of matrix B are not matched.

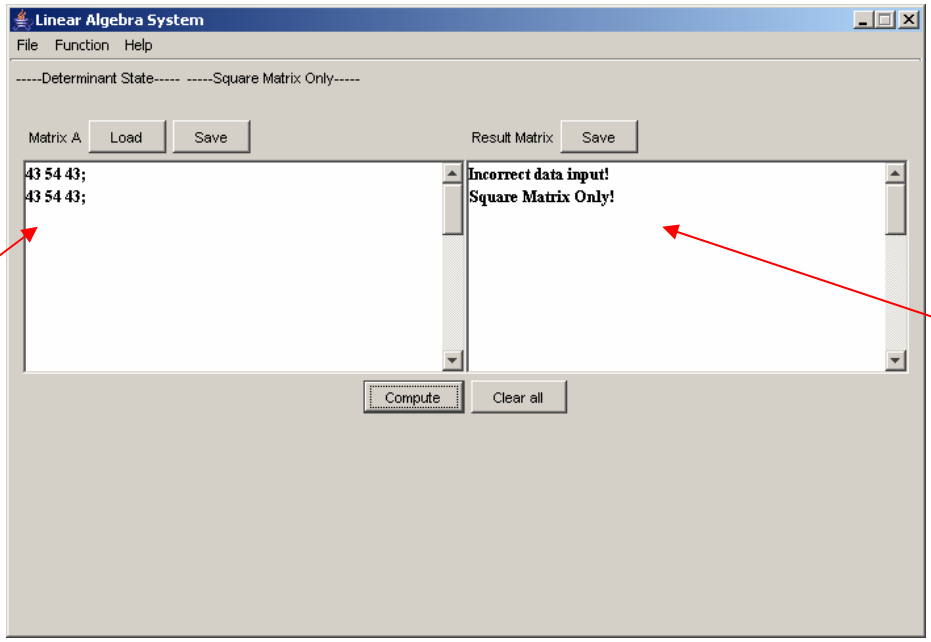
Error message returns

Matrix Determinant

Correct square matrix.



Result returns for the determinant



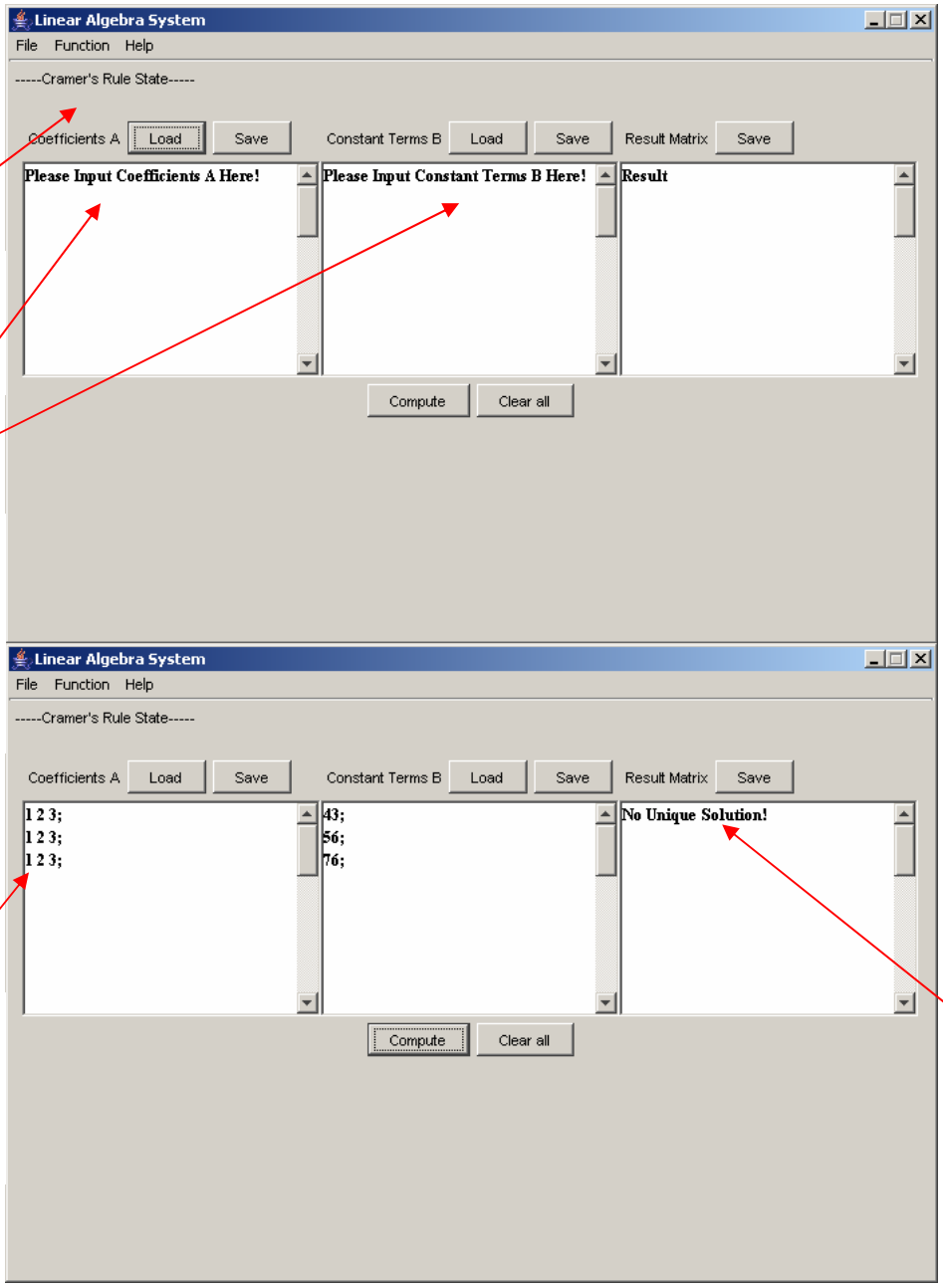
It is not a square matrix.

Error message returns for non-square matrix input

11.1.3.5 System of linear equations

There are two methods to solve system of linear equations, Cramer's Rule and Gauss-Jordan Elimination. It is shown what the system returns with respect to error message and how each method works in a proper manner.

Cramer's Rule

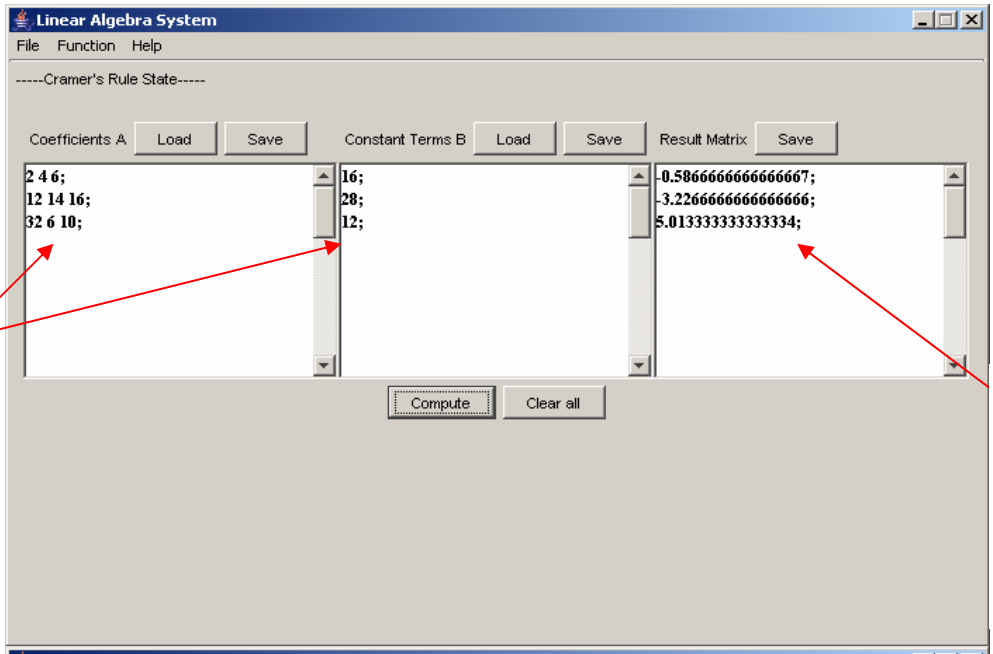


It is in Cramer's Rule state

Input the coefficients and constant terms of equations.

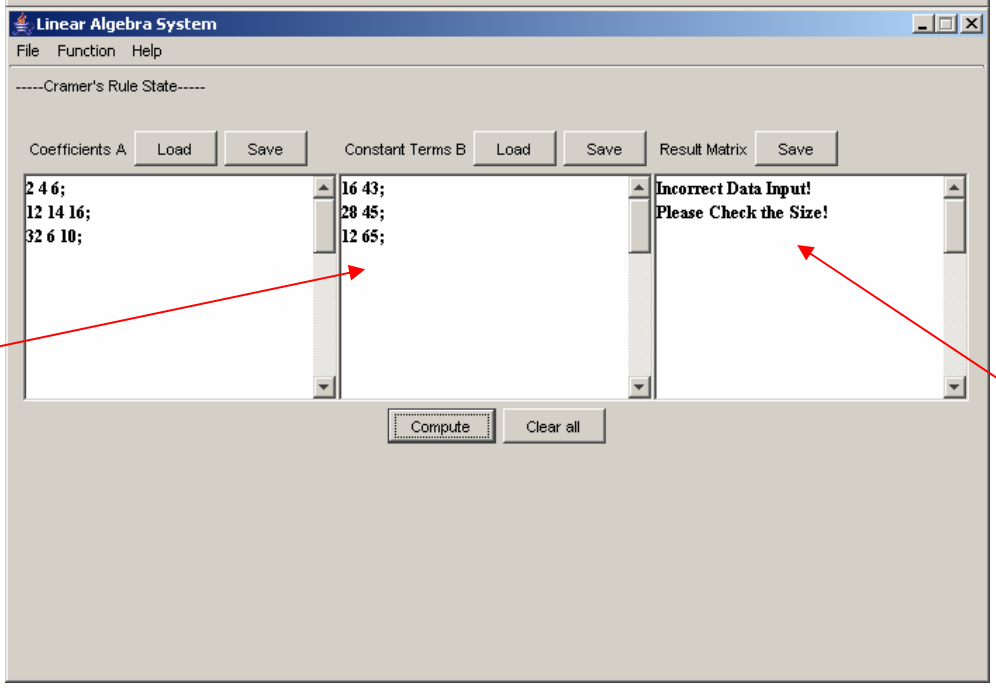
The determinant of input is zero.

No unique solution will return to users.



Valid coefficients input and valid constant terms input

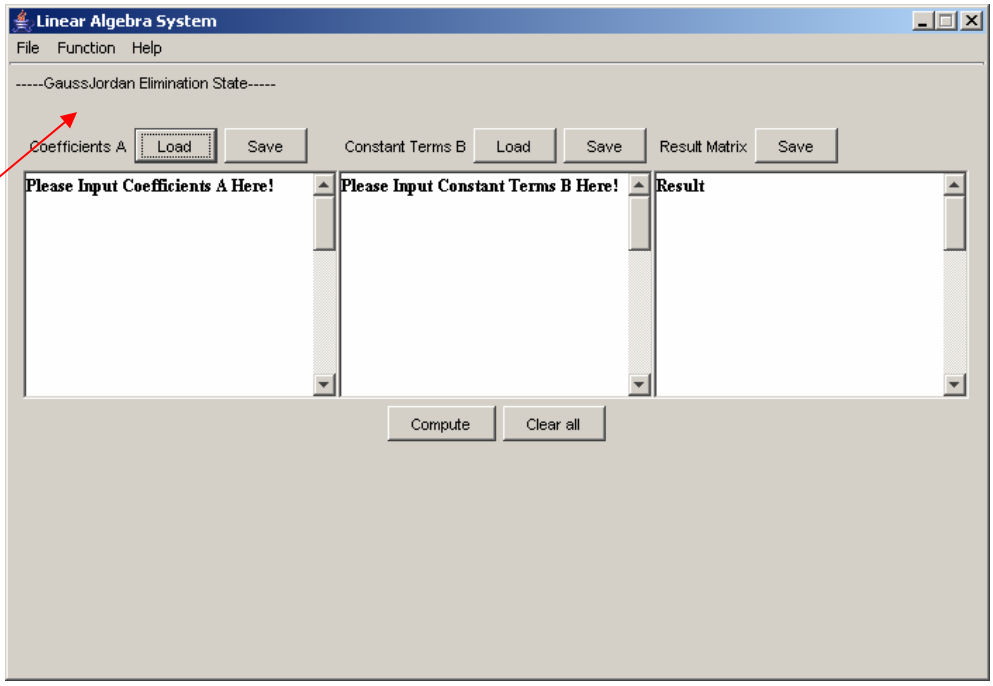
Result returns.



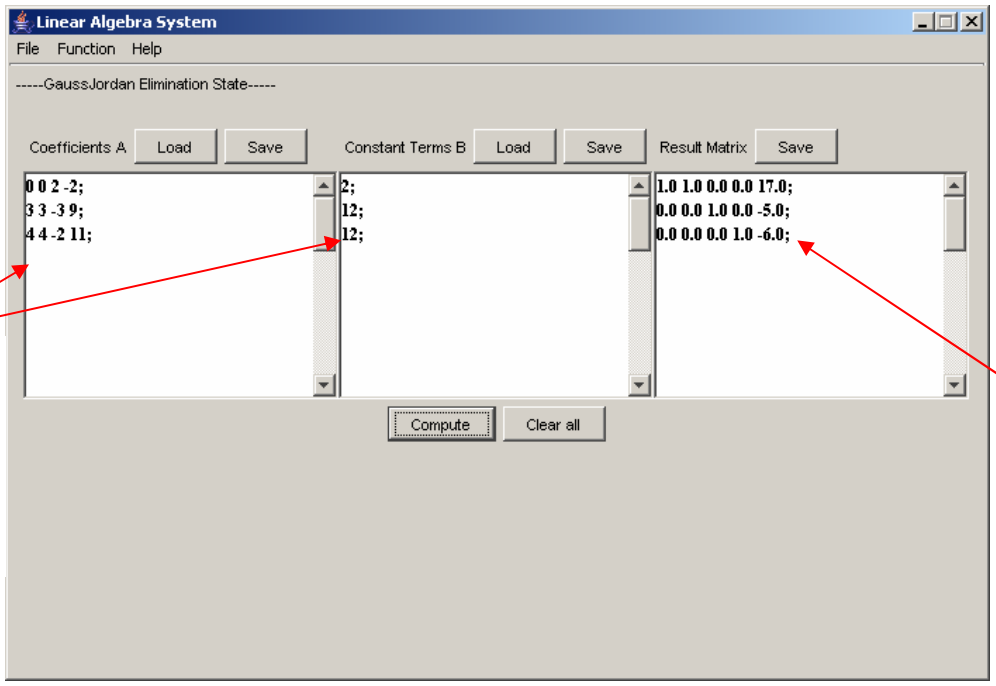
Invalid input for constant terms

Error message returns to users

Gauss-Jordan Elimination

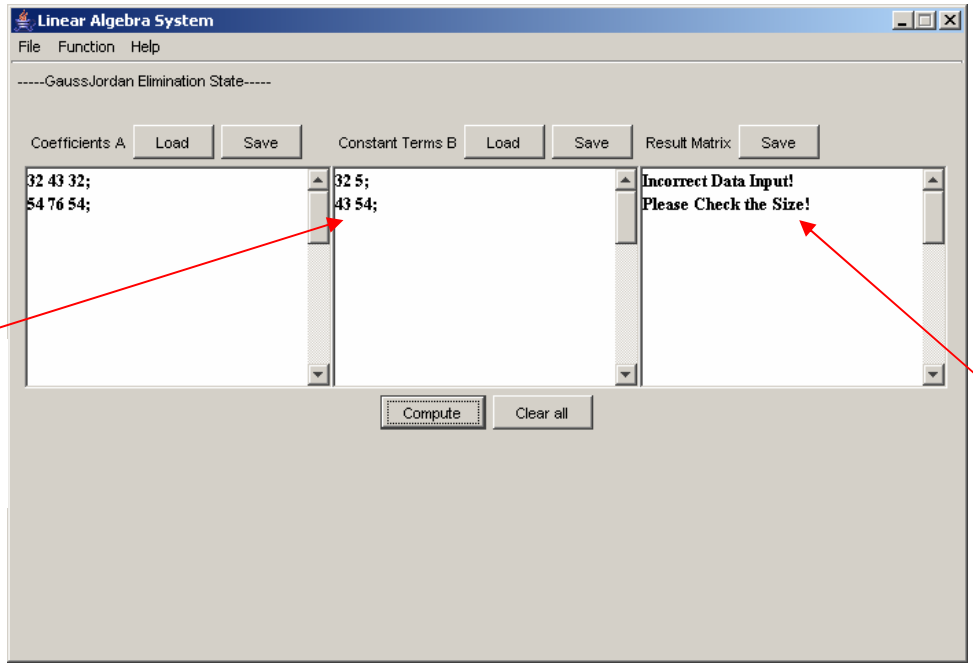


Gauss-Jordan Elimination State



Valid input for coefficient and constant terms of equations

Result returns

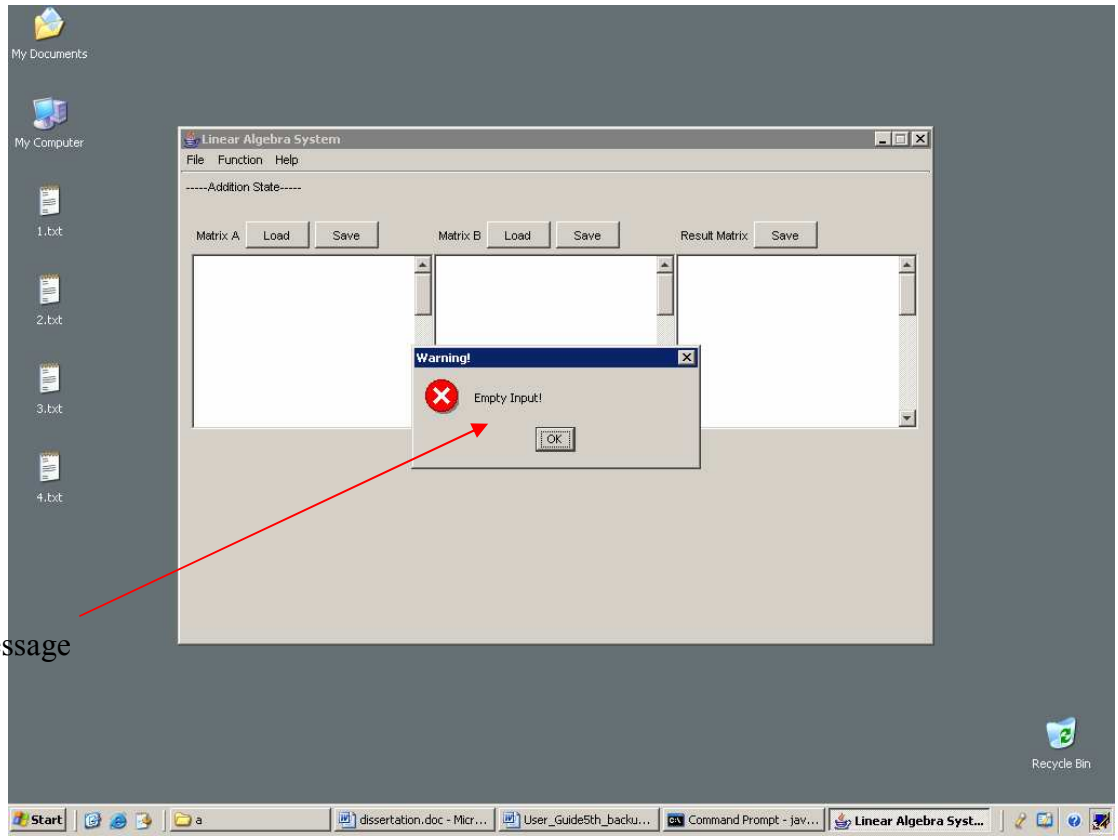


Invalid input for constant terms of equations

Error message returns

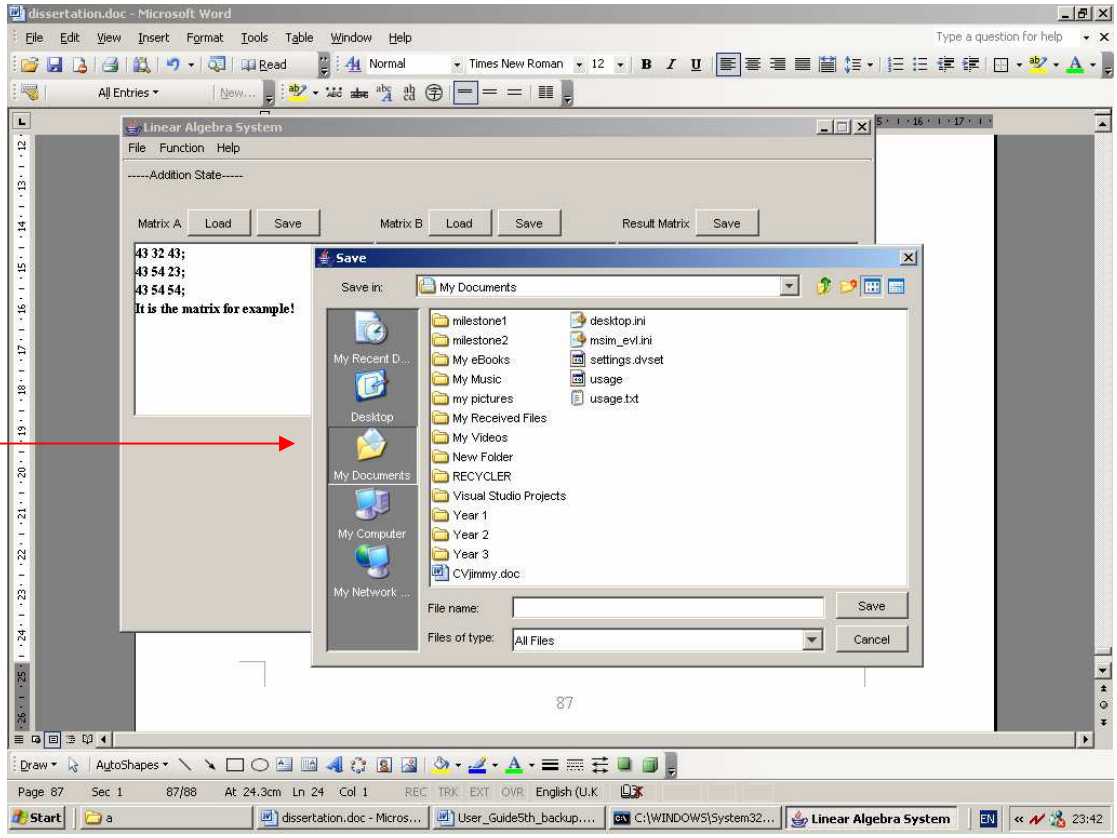
11.1.3.6 How to save the matrix

The specific location for matrix is empty on the interface. Error message will display, if it is not empty, it could be saved. The "save function" will work as below:



Error Message

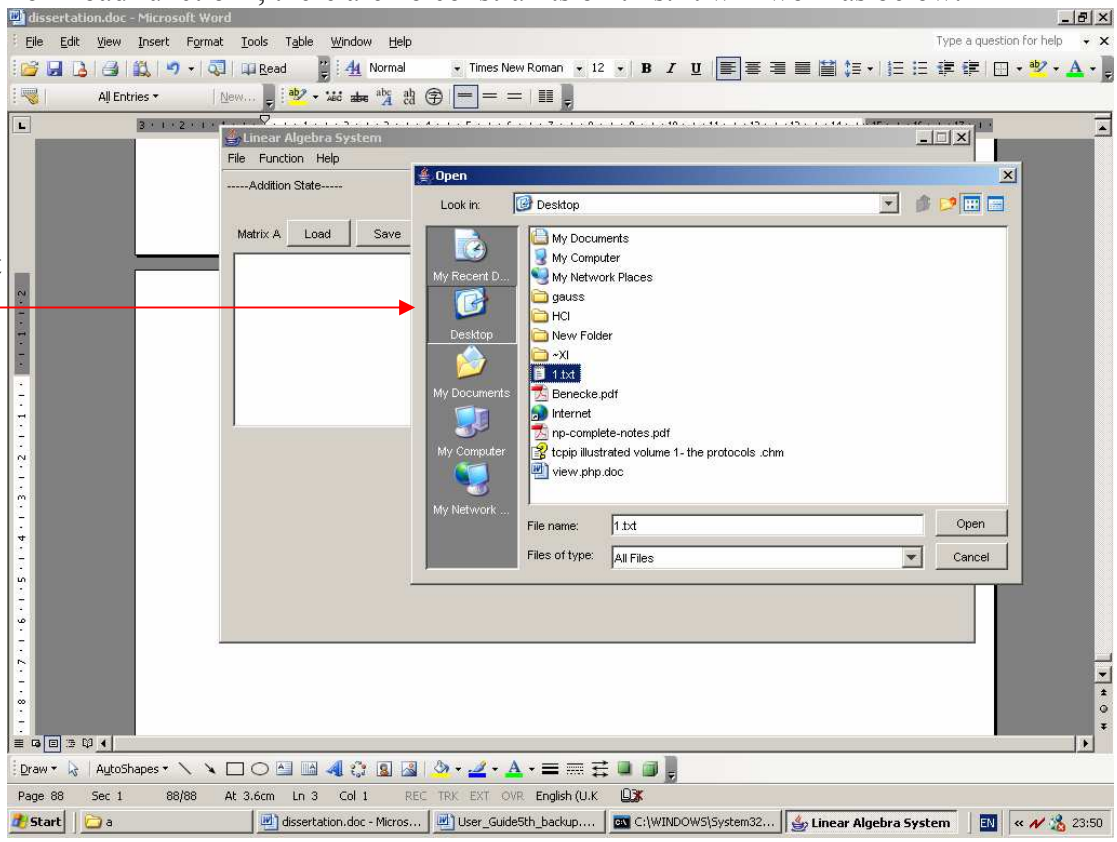
Save the matrix into a file of any formats. Normally txt preferred.



11.1.3.7 How to Load the content of a matrix

For "load function", there are no constraints on this. It will work as below:

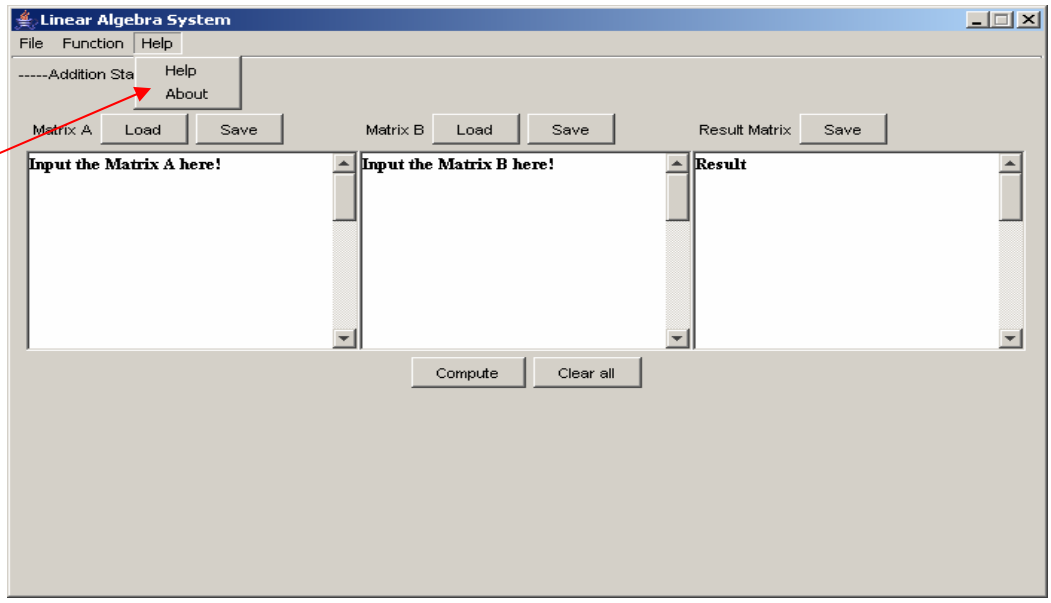
Load the content of matrix Onto interface from a file



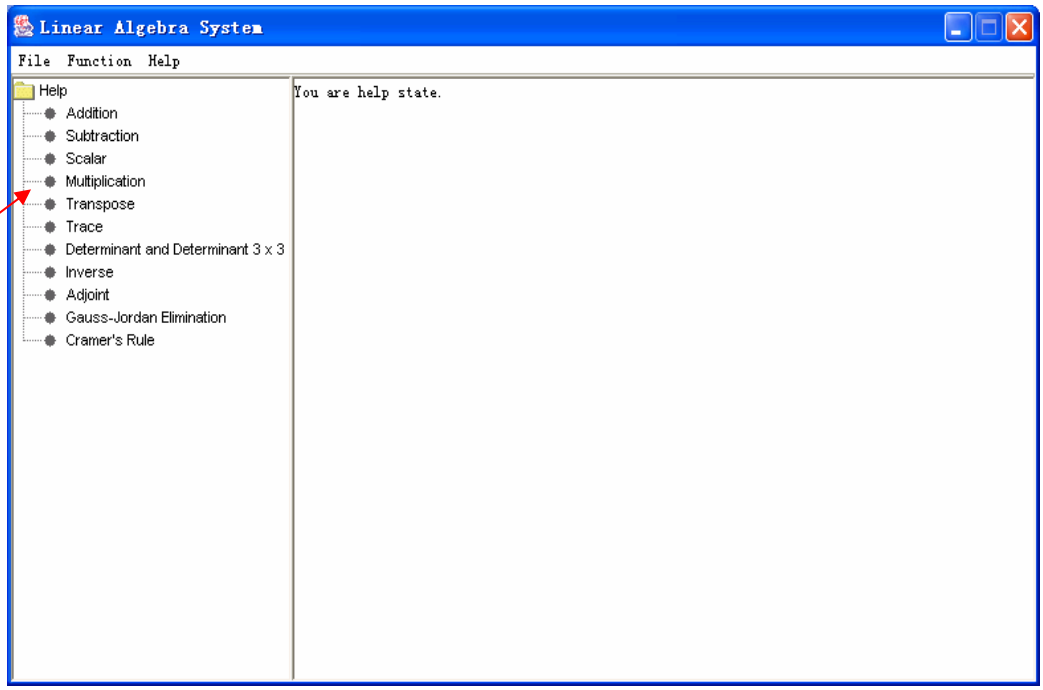
11.1.3.8 Help Function

Help function is provided by the system for people who are not familiar with the system. Help function will provide the details to each matrix application.

Help function provided here



It is the help state



11.2 Code