Linear Algebra System

Kun Wang

Batchelor of Science in Computer Information System with Honours
The University of Bath
May 2007

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

Signed:

Linear Algebra System

Submitted by: Kun Wang

COPYRIGHT

Declaration

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

Signed:

Abstract

This Linear Algebra System is a Java Code written application, which is equipped with functions of solving Linear Algebra related computation by implementing mathematical algorithms. It is concerned with the manipulation and arithmetic of arbitrary sized vectors and matrices. It can work with decimal number, rational number and complex number. Besides using the normal linear algebra algorithms, it also delivers functions employing some advanced algorithms with low complexity. A basic neat Artificial Intelligence is programmed in order to improve the performance.

1 Introduction

Linear Algebra is a central course in undergraduate mathematics, which is placed as the beginning of abstract mathematics. It also plays an important role in other areas, such as natural sciences and social sciences. When we use the theories of Linear Algebra to solve problem, in most time we deal with matrix calculations. It's simple when we do 2 by 2 or 3 by 3 matrix operation using pen and paper. However, it takes hours or days to do computations on tens or hundreds of dimension matrices. So a Linear Algebra based mathematical system is quit necessary.

In fact, currently there has been some mathematic software created for solving matrices calculation, such as Maple, Matlab, Mathematica and so on. However, these softwares also cover other aspects of mathematics, which makes the software complicated to use and learn. It's quite useful for a mathematician doing cross-fields computations, but it is very wasteful for the users who only want to do some matrix operations on it. Moreover, these softwares normally take hundreds of megabits from hard disk to install and a while to start up. In addition, the most of the existing systems has the problem of compatibility. For example, the Matlab running in the Windows XP is different from the version running in the Mac OS.

Considering all above, it is very necessary to narrow down scope of mathematics to create a purely Linear Algebra System with the following primary objectives and secondary objectives.

Primary Objectives:

- Capable of process different types of number, such as decimal number, rational number or complex number
- Capable of doing some general linear algebra arithmetic, which is actually about vector and matrix operations
- Capable of implementing some advanced math algorithms with low complexity to do matrix calculations

Secondary Objectives:

- Artificial Intelligence
- Friendly user interface
- Robust system with error handling functions
- Independent code structure working on different fields, which can make the system easy to improve and develop later

2 Literature Review

2.1 Introduction

Before getting further to the system designing and coding, there are many linear algebra theories I should understand and apply to the project. This literature review is going to prove what knowledge and ideas have been established on this project.

Concerning with the aim of my project, 'create a system, which applies the algorithms of Linear Algebra to solve matrices and vectors based problem', the scope of the project is only focused on the matrices and vectors based computation. So in the later work I need find out what kind of operations and algorithms apply to matrices and vectors there are and how they work.

What's more, since there have been already many mathematical systems invented, my project could be considered as a scale-reduced version of other systems. Then it is necessary to research and analyse the current mathematical systems. The reason it is necessary is because I need to prove the knowledge I gain from Linear Algebra is applicable and the previous work will give me some ideas and directions in the later work of designing the system.

There are two sections in this review, which are Linear Algebra explaining the academic understanding on Linear Algebra, and current system discussing two current systems, Maple and Matlab.

2.2 Numbers

A number is a mathematical concept and an abstract entity used to representing a quantity and express how many things are being referred to, or how much there is of some thing or property. It can be a count tool for individual objects, and also to measure quantities such as length, area, weight and time (1).

2.2.1 Rational Number

Rational number is the subset of real number, which also includes irrational number and can be described informally as numbers that can be given by an infinite decimal representation. A rational number is a number which can be expressed as a ratio of two integers. Non-integer rational numbers (commonly called fractions) are usually written as the vulgar fraction a / b, where b is not zero and a is called numerator and b is called dominator.

A rational number can be represented in an infinite form, like 1/2=2/4=3/6… So normally the rational number is simplified into the form which the numerator and dominator have no common

divisor exception 1. The biggest common divisor between the numerator and dominator is called the greatest common division, which is used to factorize the rational number into the simplest form.

The decimal expression with finite decimal digits is a special form a rational number, such as 1.44393, 42894.420098. They can change back to the normal form of rational number. The normal method is to multiply it by 10^n and set it as a numerator and set 10^n as a dominator, where n is the number of its decimal digits. Then we factorise it into the simplest form. However, change the normal form rational number to a decimal number may change the number into other type of decimal number with infinite decimal digits. For example, 1/3 is a rational number, if change it into a decimal expression, it will become 1.33333...with repeating 3.

Arithmetic:

- Two rational number a/b and c/d are equal if and only if

$$ad = bc$$

- Two fractions are added as follow

$$-\left(\frac{a}{b}\right) = \frac{-a}{b} = \frac{a}{-b}$$

- Two fractions are multiplies as follow

$$\frac{a}{b} \bullet \frac{c}{d} = \frac{ac}{bd}$$

Additive and multiplicative inverse exist in the rational numbers

$$-\left(\frac{a}{b}\right) = \frac{-a}{b} = \frac{a}{-b} \quad \textbf{and} \quad \left(\frac{a}{b}\right)^{-1} = \frac{b}{a} \quad \textbf{if} \quad b \neq 0$$

It follows that the quotient of two fractions is given by

$$\frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc}$$

2.2.2 Complex Number

The process which first requires the use of complex number is that of solving quadratic equations.

$$x^2 = -1$$

The equation has no real solution, since the square of any real number is never negative. So we have to extend the concept of number by introducing numbers that will make the equation solvable. Then people introduced the symbol i by defining i^2=-1 and complex number.

In mathematics, a complex number is a number of the form

$$a + bi$$

where a and b are real numbers, and i is the imaginary unit with the property i^2=-1. the real number

a is called real part and b is called imaginary part.

Arithmetic:

- Addition : $(a+bi)+(c+di)=(a+c)+(b+d)i$
- Subtraction: $(a+bi)-(c+di)=(a-c)+(b-d)i$
- Multiplication: $(a+bi)(c+di)=ac+bci+adi+bdi^2=(ac-bd)+(bc+ad)i$
- Division: $\frac{a+bi}{c+di}=\frac{(a+bi)(c-di)}{(c+di)(c-di)}=\frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$

2.3 Linear Algebra

In this section, I will establish the knowledge of Linear Algebra based on the objectives of the project.

2.3.1 Matrices and Linear Equations

Linear Equations are equations with variables in them, such as x-2y=7.

A matrix is *a* rectangular array of numbers arranged by rows and columns. We call each number in the array the element of such matrix. However, in this dissertation, the entry has also been used to describe the element of a matrix.

In Linear Algebra, we focus on the large system of n linear equations with n variables represented by matrices. So before illustrating how matrices work on the system of linear equation, there are some terminologies about a matrix that should be aware of.

2.3.1.1 Matrices Properties

Rows and Columns

A matrix can be decomposed into Rows or Columns. For this matrix $A=\begin{bmatrix}2 & 5 & 8\\ 6 & 1 & 7\end{bmatrix}$, there are two rows and three columns in it.

The first row is $\begin{bmatrix}2 & 5 & 8\end{bmatrix}$ and the second row is $\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 3 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$.

The first column is $\begin{bmatrix}2\\6\end{bmatrix}$, the second is $\begin{bmatrix}5\\1\end{bmatrix}$ and the third is $\begin{bmatrix}8\\7\end{bmatrix}$.

Location

The location of an element in a matrix is decided by giving row and column of that element. In the previous example, matrix **A,** the number in Row 1 and Column 2 is 5 represented in location(1, 2).

Submatrix

A submatrix is a part of a given matrix, which is generated by removing certain rows and columns of certain location of the matrix.

For example, in the matrix $A=\begin{bmatrix}2 & 5 & 8\\6 & 1 & 7\end{bmatrix}$, its submatrices can be:

B=$\begin{bmatrix}2 & 5\\6 & 1\end{bmatrix}$, C=$\begin{bmatrix}8\\7\end{bmatrix}$, D= $\begin{bmatrix}6 & 1 & 7\end{bmatrix}$.

Size and Type

The size of a matrix is described in form of **a**×**b**, which **a** is the number of rows and **b** is the number of column. For matrix **A,** it is 2×3 matrix.

There are three types of matrix, **square matrix**, in which the number of rows and column are same, **row matrix**, in which there is only one row and **column matrix**, in which there is only one column matrix.

$\begin{bmatrix}2 & 5 & 8\\6 & 1 & 7\\4 & 3 & 0\end{bmatrix}$	$\begin{bmatrix}6 & 1 & 7\end{bmatrix}$	$\begin{bmatrix}3\\5\\2\end{bmatrix}$
Square Matrix	Row Matrix	Column Matrix

Identity Matrices

An identity matrix is a square matrix with 1s in the diagonal locations and zeros elsewhere.

Example: $I_2=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$ $I_3=\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$

Triangular Matrices

A triangular matrix is a special kind of square matrix where the entries below or above the main diagonal are zero.

A matrix

$$\mathbf{L} = \begin{bmatrix} l_{1,1} & & & & 0 \\ l_{2,1} & l_{2,2} & & & \\ l_{3,1} & l_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & l_{n,n} \end{bmatrix}$$

is called lower triangular matrix or left triangular matrix, and analogously a matrix of

$$\mathbf{U} = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \dots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

is called upper triangular matrix or right triangular matrix.

Row-Echelon Form

The matrix in row-echelon form is actually upper triangular matrix. It must is satisfies the following requirements:

- All nonzero rows are above any rows of all zeroes.
- The leading coefficient of a row is always strictly to the right of the leading coefficient of the row above it.

For example:

$$\begin{bmatrix} 9 & 3 & 5 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 9 & 3 & 5 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

*The leading coefficient of a row in a matrix is the first nonzero entry in that row.

Every matrix can be transformed into a Row-Echelon form by using **Gaussian elimination**. It employs elementary operation, which Given a matrix with rows $R_1,R_2,...,R_n$, the following are the three elementary operations:

- Interchange two rows
- Replace a row with a nonzero multiple of itself
- Replace a row with the sum of itself and a multiple of another row.

2.3.1.2 Matrices and Linear Equations

In this session, I will introduce the way of using matrices to describe and solve linear equations.

For the system of linear equation:

$$\begin{aligned} x_1 + x_2 + x_3 &= 3 \\ 3x_1 + 7x2 + 2x_3 &= 6 \\ 9x_1 + 4x_2 + 6x_3 &= 1 \end{aligned}$$

There are two kinds of matrix to represent this. One is the **matrix of coefficients** formed by the coefficients of variables. The other one is the **augmented matrix** consisting coefficients and constant terms together.

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 7 & 2 \\ 9 & 4 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 3 \\ 3 & 7 & 2 & 6 \\ 9 & 4 & 6 & 1 \end{bmatrix}$$

matrix of coefficients augmented matrix

Reduced echelon form

When solving a system of linear equations, the augmented matrix is often used. In order to get the result, we need to transform the augmented matrix into reduced echelon form.

A matrix is in reduced echelon form if:

1. Any rows consisting entirely of zeros are grouped at the bottom of the matrix;
2. The first nonzero element of each other row is 1. This element is called a leading 1;
3. The leading 1 of each row after the first is positioned to the right of the leading 1 of the previous row.
4. All other elements in a column that contains a leading 1 are zero.

The follower matrices are all in reduced echelon form.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 8 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following are not.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Gauss-Jordan Elimination

We are using Gauss-Jordan Elimination to complete transforming a matrix into the reduced echelon form, which is a version of Gaussian elimination.

1, Write down the augmented matrix of the system of linear equations

2. Change the matrix to reduced echelon form by using elementary row operations. Starts from find the first leading 1 column by column from left to right, then make the element above or before leading 1 zero.
3. Apply the reduced Echelon form to the system of equation to get the result.

Example:

$$\begin{aligned} x_1 + 4x_2 + 3x_3 &= 1 \\ 2x_1 + 8x_2 + 11x_3 &= 7 \\ x_1 + 6x_2 + 7x_3 &= 3 \end{aligned}$$

so, the augmented matrix is

$$\begin{bmatrix} 1 & 4 & 3 & 1 \\ 2 & 8 & 11 & 7 \\ 1 & 6 & 7 & 3 \end{bmatrix}$$

Using the Gauss-Jordan Elimination

$$\begin{bmatrix} 1 & 4 & 3 & 1 \\ 2 & 8 & 11 & 7 \\ 1 & 6 & 7 & 3 \end{bmatrix} \overset{\substack{R2-R1*2 \\ R3-R1*2}}{\approx} \begin{bmatrix} 1 & 4 & 3 & 1 \\ 0 & 0 & 5 & 5 \\ 1 & 6 & 7 & 3 \end{bmatrix} \overset{R2 \leftrightarrow R3}{\approx} \begin{bmatrix} 1 & 4 & 3 & 1 \\ 1 & 6 & 7 & 3 \\ 0 & 0 & 5 & 5 \end{bmatrix} \overset{R2-R1}{\approx}$$

$$\begin{bmatrix} 1 & 4 & 3 & 1 \\ 0 & 2 & 4 & 2 \\ 0 & 0 & 5 & 5 \end{bmatrix} \overset{\substack{R2*(1/2) \\ R3*(1/5)}}{\approx} \begin{bmatrix} 1 & 4 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \overset{R1-R2*4}{\approx} \begin{bmatrix} 1 & 0 & -5 & 3 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\overset{\substack{R1+5R3 \\ R2-2R1}}{\approx} \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} x_1 &= 2 \\ x_2 &= 3 \\ x_3 &= 1 \end{aligned}$$

So, the solution for this system of equation is $x_1=2$, $x_2=3$, $x_3=1$.

Solution

We can only use reduced echelon form to solve the system of equation, of which the number of variables is same as the number given equations. However, even under this condition, the solution has three different situations, unique solution, many solutions and no solution. The discussion about how to judge the solution situations will be carried on in the following sections.

2.3.2 Matrix Operations

2.3.2.1 Addition, Scalar Multiplication, and Multiplication of Matrices

Addition of Matrices

Definition: Matrices addition can only happen between two matrices with same dimension. The sum of matrix A + B is obtains by adding the elements in the same location. The result matrix of A+B also has the same size as of A and B

Example

$$A=\begin{bmatrix}1 & 3 & 4\\ 9 & 2 & 6\end{bmatrix} \qquad B=\begin{bmatrix}3 & 6 & 4\\ 7 & 4 & 9\end{bmatrix}$$

$$A+B=\begin{bmatrix}1+3 & 3+6 & 4+4\\ 9+7 & 2+4 & 6+9\end{bmatrix}=\begin{bmatrix}4 & 9 & 8\\ 16 & 6 & 15\end{bmatrix}$$

Scalar Multiplication

*Definition: If A is a matrix, C is a scalar ,then **B=cA**, is obtained by multiplying each elements in A by c. The B will be the same size as A.*

Example:

$$A=\begin{bmatrix}1 & 3 & 4\\ 9 & 2 & 6\end{bmatrix} \quad , \quad c=3$$

$$B=cA=\begin{bmatrix}1\times 3 & 3\times 3 & 4\times 3\\ 9\times 3 & 2\times 3 & 6\times 3\end{bmatrix}=\begin{bmatrix}3 & 9 & 12\\ 27 & 6 & 18\end{bmatrix}$$

Multiplication of Matrices

Definition: The matrix A and B can be multiplied only if the number of columns in A is same as the number of rows in B. The element in the location of (i,j) in the product of AB is obtained from the sum of multiplying product of corresponding elements of row i in A and column in B.

If $A=\begin{bmatrix}a_{i1} \dots a_{in}\end{bmatrix}$, $B=\begin{bmatrix}b_{1j}\\ \dots\\ b_{nj}\end{bmatrix}$, C=AB, then $c_{ij}=a_{i1}b_{1j}+\dots+a_{in}b_{nj}$

Example:

$$\begin{bmatrix}2\\ 4\\ 1\end{bmatrix}*\begin{bmatrix}3 & 5 & 1\end{bmatrix}=\begin{bmatrix}6 & 10 & 2\\ 12 & 20 & 4\\ 3 & 5 & 1\end{bmatrix}$$

So, *"if A is an m*r matrix and B is an r*n matrix, then AB will be an m*n matrix"*.

2.3.3 Algebraic Properties of Matrix Operations

Theorem: Let A, B and C be matrices, and a, b and c be scalars. Assume that the sizes of the matrices are such that the operations can be performed.

Properties of Matrix Addition and Scalar Multiplication **(2)**

1. $A+B = B + A$ — *Commutative property of addition*
2. $A+(B+C)=(A+B)+C$ — *Associative property of addition*
3. $A+O=O+A=A$ — *(where O is the appropriate zero matrix)*
4. $c(A+B)=cA+cB$ — *Distributive property of addition*
5. $(a+b)C=aC+bC$ — *Distributive property of addition*
6. $(ab)C=a(bC)$

Properties of Matrix Multiplication

1. $A(BC)=(AB)C$ — *Associative property of multiplication*
2. $A(B+C)=AB+AC$ — *Distributive property of multiplication*
3. $(A+B)C=AC+BC$ — *Distributive property of multiplication*
4. $AI_n=I_nA=A$ — *(where I_n is the appropriate identity matrix)*
5. $c(AB)=(cA)B=A(cB)$

Note: $AB \neq BA$

2.3.3.1 The transpose, trace, inverse of matrices

Transpose

Definition: The transpose of a matrix A, denoted A', is the matrix whose columns are the rows of the given matrix

Example:

$$A=\begin{bmatrix}1 & 3 & 4\\ 2 & 6 & 9\end{bmatrix} \qquad A'=\begin{bmatrix}1 & 2\\ 3 & 6\\ 4 & 9\end{bmatrix}$$

Trace

*Definition: Let A be a square matrix. The trace of A, denoted tr(A) is the sum of the diagonal elements of A. Thus if A is an n*n matrix.*

$$tr(A) = a_{11}+a_{22}+\ldots+a_{nn}$$

Example:

$$A=\begin{bmatrix}1 & 3 & 4\\ 2 & 6 & 9\\ 3 & 5 & 2\end{bmatrix} \qquad tr(A)= 1+6+2 = 9$$

"Properties of Transpose and Trace

Let A and B be matrices and c be a scalar. Assume that the sizes of the matrices are such that the operations can be performed. **(2)**

1. *(A+B)' = A'+ B'*
2. *(cA)' = cA'*
3. *(AB)' = B'A'*
4. *(A')'= A*
5. *tr(A+B) = tr(A) + tr(B)*
6. *tr(AB) = tr(BA)*
7. *tr(cA) = ctr(A)*
8. *tr(A') = tr(A)"*

Inverse

Definition: if there is a matrix B can make A*B=B*A= I_n, we say matrix A is invertible and B is A's inverse. Moreover, there is only one inverse for an invertible matrix.

Example:

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

$$A*B = \begin{bmatrix} 2\times 3+5\times(-1) & 2\times(-5)+5\times 2 \\ 1\times 3+(-1)\times 3 & 1\times(-5)+2\times 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\text{And } B*A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

A is an n*n matrix, the process of finding a inverse of a matrix is:

1. Adjoin A with identity matrix I_n . The new matrix is $[A:I_n]$.
2. Compute the reduced echelon form of $[A:I_n]$.
3. If the reduced echelon form is in the form of $[I_n: B]$, then B is the inverse; If not, A does not have inverse.

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \qquad [A:I_n] = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & 3 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & 2 & -1 \end{bmatrix}$$

$$\text{So } A^{-1} = \begin{bmatrix} -3 & 2 \\ 2 & -1 \end{bmatrix}$$

Theorem: In the system of equation AX=Y, if matrix A is invertible, then the solution is unique.

If A is an invertible matrix, then A^T is and invertible matrix. **(3)**

2.3.4 Determinants of Square Matrices

2.3.4.1 What are determinants?

Definition: for the matrix $A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, *its determinant* $|A|=a_{11}*a_{22} - a_{12}*a_{21}$.

Definition: for the square matrix A, its ***Minor*** *of element* a_{ij} *denoted as* M_{ij} *is the determinant of the matrix A after deleting row i and column j. The* ***Cofactor*** *of* a_{ij} *denoted as* C_{ij} *is defined as,* $C_{ij} = (-1)^{i+j} * M_{ij}$

Definition: The determinant of a square matrix is the sum of the product of one of its rows or columns and their cofactors.
ith row expansion: $|A|=a_{i1}*C_{i1}+a_{i2}*C_{i2}+a_{i3}*C_{i3}+\ldots+a_{in}C_{in}$
ith column expansion: $|A|=a_{1i}*C_{1i}+a_{2i}*C_{2i}+a_{3i}*C_{3i}+\ldots+a_{ni}C_{ni}$

Example:

$$A=\begin{bmatrix} 1 & 2 & -1 \\ 3 & 0 & 1 \\ 4 & 2 & 1 \end{bmatrix} \qquad |A|= a_{21}*C_{21}+a_{22}*C_{22}+a_{23}*C_{23}$$

$$= -3 * \begin{vmatrix} 2 & -1 \\ 2 & 1 \end{vmatrix} + 0 * \begin{vmatrix} 1 & -1 \\ 4 & 1 \end{vmatrix} -1 * \begin{vmatrix} 1 & 2 \\ 4 & 2 \end{vmatrix}$$

$$= -6$$

2.3.4.2 Properties of Determinants

Definition: if $|A|=0$, *then the matrix A is* ***singular*** *or it's* ***non-singular.***

Let A be a n*n matrix and c be a nonzero scalar

1. If B= cA , then |B|= c|A|;
2. If B is the matrix after A interchanging two rows, |B|= -|A|;
3. If B is the matrix after A adding a multiple of one row(column) to another row(column), then |B| = |A|
4. $|cA| = c^n|A|$
5. |AB| = |A||B|
6. $|A^t| = |A|$
7. $|A^{-1}| = 1/|A|$
8. If elements in one row or column of square matrix A are all zeros, det A = 0;
9. If ith row or the jth column of A is multiplied by a scalar c, the det A is multiplied by c. That is, If we call this new matrix B, then det B= c* det A.

10. If A has two equal rows of columns, then det A = 0
11. If one row (column) of A is a scalar multiple of another row (column), the det A =0
12. the determinant of a triangular matrix is the product of its diagonals**(4)**.

2.3.4.3 Determinants

Definition: The matrix formed by the cofactors C_{ij} of given matrix A is called the **matrix of cofactor** of A. The transpose of this matrix is called the **adjoint** of A, adj(A).
Example:

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

$$C11=\begin{bmatrix} 4 & -2 \\ -3 & 5 \end{bmatrix} = 14, \quad C12= \begin{bmatrix} -1 & -2 \\ 1 & 5 \end{bmatrix} =3, \quad C13=\begin{bmatrix} -1 & 4 \\ 1 & -3 \end{bmatrix} = -1$$

$$C21=\begin{bmatrix} 0 & 3 \\ -3 & 5 \end{bmatrix} = -9, \quad C23=\begin{bmatrix} 2 & 3 \\ 1 & 5 \end{bmatrix} =7, \quad C24= \begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix} = 6$$

$$C31= \begin{bmatrix} 0 & 3 \\ 4 & -2 \end{bmatrix} =-12, \quad C32=\begin{bmatrix} 2 & 3 \\ -1 & -2 \end{bmatrix} =-1, \quad C33=\begin{bmatrix} 2 & 0 \\ -1 & 4 \end{bmatrix} = 4$$

Then the matrix of cofactors of A is $\begin{bmatrix} 14 & 3 & -1 \\ -9 & 7 & 6 \\ 12 & -1 & 4 \end{bmatrix}$

And the adjoint is, $\text{Adj(A)}= \begin{bmatrix} 14 & -9 & -12 \\ 3 & 7 & 1 \\ -1 & 6 & 8 \end{bmatrix}$

2.3.4.4 Determinants relating to Inverse and Linear Equation

Determinants and Inverse

It was mentioned in the last section about how to calculate the inverse of an invertible matrix.

However, it is not an algebraic formula and it not appropriate to programme. In here, an algebraic formula to calculate the inverse will be presented.

If a square matrix A is non-singular. A is invertible and $A^{-1} = (1/|A|) * adj(A)$.

Determinants and Systems of Linear Equations

Theorem: AX=Y is a system of n linear equations with n variables, which has unique solution only if $|A| \neq 0$*, or has many or no solutions.*

*Cramer's Rule***(5)***: If AX=Y is a system of n linear equations with n variables and* $|A|$ *0 Then the solution is:*

$$x_1=|A_1|/|A| \text{ , } x_2=|A_2|/|A| \text{ ,...,} x_n=|A_n|/|A|$$

A_i *is the matrix obtained by replace the i column of A with Y.*

2.3.5 Vector Spaces

2.3.5.1 Properties Vector Space R^n

Definition: R^n *consists of the set of a sequence of n real numbers. Fox example, (1,4,3,5) is one of elements in* R^4.

Definition: $u=(u_1...u_n)$ *and* $v=(v_1...v_n)$ *are two elements of* R^n *. They are equal when* $u_1=v_1$, ..., $u_n=v_n$.

Definition: $u=(u_1,...,u_n)$ *and* $v=(v_1,...,v_n)$ *are two elements of* R^n *and c is a scalar*

$$u+v=(u_1+v_1,...,u_n+v_n)$$

$$cu=(c\,u_1, \ldots, c\,u_n)$$

Properties

u, v and w are vectors in R^n, c and d are scalars.

1. u + v = v + u
2. u + (v +w)= (u + v) + w
3. u + 0 = 0 + u = u
4. u + (-u) = 0
5. c(u + v) = cu + cv
6. (c + d)u= cu + du
7. c(du)=(cd)u
8. 1u=u

2.3.5.2 Dot Product, Cross Product, Norm, Angle, and Distance

Dot Prodect.

Definition: $u=(u_1,...,u_n)$ *and* $v=(v_1,...,v_n)$ *are two vectors in* R^n. *. The dot product of u and v,* $u \cdot v=$

$u_1 v_1 + ,...,+ u_n v_n$.

Example:

u = (1, −2, 4) and v = (3, 0, 2)

u • v = $(1\times3)+(-2\times0)+(4\times2)=11$

Cross Product

There is another way to combine two vectors apart from dot product.

Definition: a=$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$, b=$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$, a×b=$\begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$. Being aware of this, this result is a vector.

More important, the cross product can only apply to 3-vectors.

Exmaple:

$$a = \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 4 \\ 6 \\ 1 \end{bmatrix}, a\times b = \begin{bmatrix} 4\times1-2\times6 \\ 2\times4-1\times1 \\ 1\times6-4\times4 \end{bmatrix} = \begin{bmatrix} -8 \\ 7 \\ -10 \end{bmatrix}$$

Norm

Definition: the norm of $u=(u_1,...,u_n)$ in R^n is defined as $\|u\| = \sqrt{u_1^2 + ... + u_n^2}$

Definition: the unit vector for a nonzero vector v is ***u=(1/||v||)*v***

Example:

a = (2, 4, 1) $|a| = \sqrt{2^2 + 4^2 + 1^2} = \sqrt{21}$

Angle

Definition: The angle between two vector u and v is defined as cosθ=u • v/(||u||||v||) 0≤ θ ≤Π*

Theorem: if the product of nonzero vector is 0, then the vector u and v are orthogonal.

Example:

u = (1,0,0) and v=(1,0,1)

u • v=1, ||u||= 1, $\|v\| = \sqrt{2}$

$$\cos\theta = \frac{u \bullet v}{\|u\|\|v\|} = \frac{1}{\sqrt{2}}$$

The angle between u and v is 45°

Distance

Definition: $x=(x_1,...x_n)$ and $y=(y_1,...,y_n)$ are two points in R^n. The distance between x and y, d(x,y)=||x-y||

2.3.5.3 Rank, Eigenvalues and Eigenvectors

Rank

Definition: “the dimension of the row space and the column space of a matrix A is called the rank of A. The rank of A is denoted rank(A)”

To calculate the rank of nonzero row vectors of a matrix, we need change the matrix to reduced echelon form first. Then, the number of nonzero row vectors is the rank.

Eigenvalues and Eigenvectors

*Definition: A is an n*n Matrix. If there exist a scalar* λ *and a nonzero vector x in* R^n *such that Ax=* λx*. Then we call x eigenvector and* λ *eigenvalue.*

Theorem: Let A be and $n \times n$ *matrix. Then* λ *is an eigenvalue of A is and only if*

$$p(\lambda)=|A-\lambda I|=0$$

Example:

$$A=\begin{bmatrix} 1 & -1 & 4 \\ 3 & 2 & -1 \\ 2 & 1 & -1 \end{bmatrix},$$

Then $|A-\lambda I|=\begin{vmatrix} 1-\lambda & -1 & 4 \\ 3 & 2-\lambda & -1 \\ 2 & 1 & -1-\lambda \end{vmatrix}=-(\lambda-1)(\lambda+2)(\lambda-3)=0$

Thus the eigenvalue of A are $\lambda_1=1, \lambda_2=-2, \lambda_3=-3$

For $\lambda_1=1$, we have $(A-I)V=\begin{bmatrix} 0 & -1 & 4 \\ 3 & 1 & -1 \\ 2 & 1 & -2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Then

$$x_1=-x_3,\ x_2=4x_3,\ V_1=\begin{bmatrix} -1 \\ 4 \\ 1 \end{bmatrix}$$

For $\lambda_2=-2$, we have $V_2=\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$

For $\lambda_3=3$, we have $V_3=\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

2.4 Advanced Algorithms

In this part, some algorithms will be introduced, which do the same computations as the previous basic algorithms but with lower complexities.

2.4.1 Complexity

The concept of complexity should be explained in order to understand more clearly about the necessity of introducing the advanced algorithms.

The complexity in here is about the computational complexity, which describes the scalability of algorithms and the inherent difficulty in providing scalable algorithms for specific computational problems. In other word, it's about how the running time and memory requirements of algorithm change as the size of input to an algorithm increases. So if an algorithm fails to scale well, no matter how far the computing technology improved, this algorithm is not applicable in industry.

Time Complexity
Complexity is a measure of how effective an algorithm is. There are many different ways to calculate the effectiveness of an algorithm. However, one of the most important ways is to compute its time complexity.

The time complexity of an algorithm is the number of steps it takes to get the output. For different length of input, it may go through different number of steps to get the output. It is not realistic to analyse every time complexity for different instances. So we are only interest into rough calculations to get the time complexity, eg. n^2 and n^2+n would be the same. This concept is called Asymptotic notations. Big-O, the notations Θ and Ω are most usual notations to describe the asymptotic behavior of functions.

2.4.2 Strassen Algorithm

In mid-sixties, Volker Strassen developed the standard matrix multiplication algorithm. In his algorithm, named Strassen Algorithm, there are three steps to multiply matrices A and B with both n*n dimension to get matrix C with n*n dimension.
Divide the input matrices A and B into n/2 * n/2 sub-matrices.
Compute another seven matrices by using additions, subtractions and multiplications on sub-matrices.
Calculate the n/2*n/2 sub-matrices of C from addition the previous seven matrices.

Lemma if

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Then

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$

$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$

$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$

$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$

$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$

$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$

$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

Finally,

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$

$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$

$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

In Strassen algorithms, the time complexity is only $\Theta(n^{lg7})$,which is O $(n^{2.807})$**(6)**, reduced from the standard matrix multiplication with time complexity of $\Theta(n^3)$

2.4.3 Coppersmith–Winograd algorithm

In the mathematical discipline of linear algebra, the Coppersmith–Winograd algorithm is the fastest currently known algorithm for square matrix multiplication. It can multiply two n×n matrices in O $(n^{2.376})$ time (see Big O notation) **(7)**. This is an improvement over the trivial O (n^3)time algorithm and the O $(n^{2.807})$time Strassen Algorithm.

Lemma if

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$S_1 = A_{21} + A_{22}$$
$$S_2 = S_1 - A_{11}$$
$$S_3 = A_{11} - A_{21}$$
$$S_4 = A_{12} - S_2$$

Then

$$T_1 = B_{12} - B_{11}$$
$$T_2 = B_{22} - T_1$$
$$T_3 = B_{22} - B_{12}$$
$$T_4 = B_{21} - T_2$$

after that

$$P_1 = A_{11} * B_{11}$$
$$P_2 = A_{12} * B_{21}$$
$$P_3 = S_1 * T_1$$
$$P_4 = S_2 * T_2$$
$$P_5 = S_3 * T_3$$
$$P_6 = S_4 * T_{22}$$
$$P_7 = A_{22} * T_4$$

$$
\begin{aligned}
U_1 &= P_1 + P_2 \\
U_2 &= P_1 + P_4 \\
U_3 &= U_2 + P_5 \\
U_4 &= U_3 + P_7 \\
U_5 &= U_3 + P_3 \\
U_6 &= U_2 + P_3 \\
U_7 &= U_6 + P_6
\end{aligned}
\qquad \textbf{finally,} \qquad
\begin{aligned}
C_{11} &= U_1 \\
C_{12} &= U_7 \\
C_{21} &= U_4 \\
C_{22} &= U_5
\end{aligned}
$$

2.4.4 Blockwise inversion(Frobenius Lemma)

This algorithm is used to solve matrix inversion.

Lemma.
Let matrix A be divided into blocks:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

With the matrix A11 square and non-singular, and the matrix:

$$\Delta = A_{22} - A_{21}{A_{11}}^{-1}A_{12}$$

is non-singular. Then

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}\Delta^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}\Delta^{-1} \\ -\Delta^{-1}A_{21}A_{11}^{-1} & -\Delta^{-1} \end{bmatrix}$$

The complexity of this algorithm is about O ($n^{2.807}$).

This algorithm works with some pre-conditions, which is that the matrix should be either upper triangular matrix or symmetric positive-definite matrix. So it not work on every non-singular matrix. However, for a arbitrary real non-singular matrix A, the product of its transpose and itself A^TA is symmetric positive-definite.

∵ $A^{-1} = (A^T A)^{-1} A^T$ and $A^T A$ is symmetric positive-definite

∴ The problem of calculating the inverse of A becomes calculating the inverse of $A^T A$. The calculating the inverse of $A^T A$ can use this algorithm.

2.4.5 Artificial Intelligence

Artificial Intelligence is referred to the intelligence performed in an artificial (man-made, non-natural, manufactured) entity. Generally speaking AI systems are built around automated inference engines. By judging certain conditions ("if"), the system infers certain consequences ("then"). There are two types of AI applications, according to consequences: classifiers ("if shiny then diamond") and controllers ("if shiny then pick up").

In Linear Algebra, there are usually more than more one ways to do calculations on a matrix, so for different form of matrices, there should be a better way to compute rather than others. Concerning with the project, it is possible to equip AI in this system to make the system ‘think’ and then solve the problems like a mathematician.

2.5 Current System

In this section, I will discuss two of current mathematics system, Maple and Matlab. The reason of choosing these two is their Mathematical functionality, flexibility and the specialty on Linear Algebra.

By researching and analysing the current system and recalling to the aim of my system, some techniques of the current systems can be inherited and improved in my project.

2.5.1 Maple

Maple is a Symbolic Computation System or Computer Algebra System providing the user interface to allow users to input mathematics in traditional mathematical notation. It is equipped with tools of solving many mathematical problem, including integrals, systems of equations, differential equation, and problems in linear algebra. Moreover, graphics support is Maple’s another specialty. The user can use its graphics routines for visualizing mathematical problem.

Maple is worksheet-based graphical interface. The user can choose worksheet or command-line to access calling functions. So the user can edit the worksheet file in common editors without running Maple system.

As a successful and well-known mathematics system in industry and academia, there are some features that may fit to this project and also some may be not necessary but need to pay attention to improve or change.

- **User interface**: as mentioned before, Maple use work-sheet based interface, which is flexible to invoke functions and modify input. However, for beginners, they may find the

language and interface daunting **(8)**. This need to pay attention to avoid in the feature work since this project should have a user-friendly interface.

- **Syntax and structure**: for advanced user, Maple provides them a powerful platform to create own functions. However, it is not easy to use without being familiar and doing some exercise is necessary . Because of the scale of my project, only focusing on Linear Algebra, the simplicity of using the system should be enhanced.
- **Error message**: Maple is not forgiving of error, but subsequent message is not very helpful . The flexibility and usability of error message should be considered in my project.
- **Number input**: Maple allows many different of number type input and they can also work together in the system. This feature should be highly considered.

2.5.2 Matlab

Matlab is another mathematical system with interactive environment with hundreds of built-in functions for mathematical computations, graphics and external interface. It also provides users a platform to write their own functions in the Matlab language, which just will behaved just like the built-in functions.

The working environment in Matlab is similar as in Maple. It also use worksheet based interface. It has separate Edit and Graphic windows to let user edit and save program and plot graphics. The basic building block in Matlab is matrix and the fundamental data-type is array. By using the highly efficient algorithm from the LINPACK libraries, Matlab is optimized for matrix and vector operations**(9)**.

There are many similar features between Matlab and Maple. So the features overlapping with Maple can fit and be improved in my project will not be bothered.

Fitting:

- **Input-output:** Matlab support interactive computation, the user can input data in command window and the system show the result in the screen and the user can choose to input the data from file and output the result into a file

- **Data Type:** as mentioned before, the fundamental data-type is array. There are several distinct data object can be created, integers, real numbers, matrices, strings, structures and cells. There is no need to declare the data type when creating a data object. The system will categorise it automatically **(10)**.

- **Command history:** it records commands executed in the command window. The user can use up-arrow to choose the previous commands.

Improving:

- **Strict working directory:** the file cannot be found and executed unless it's in the current directory. It is not convenient for the user because the user have to copy the file to the current directory. Especially for beginners, without being aware of this, they will think they did not install the system appropriately or there is something wrong with the configuration.

3 Requirements Analysis and Specification

3.1 Introduction

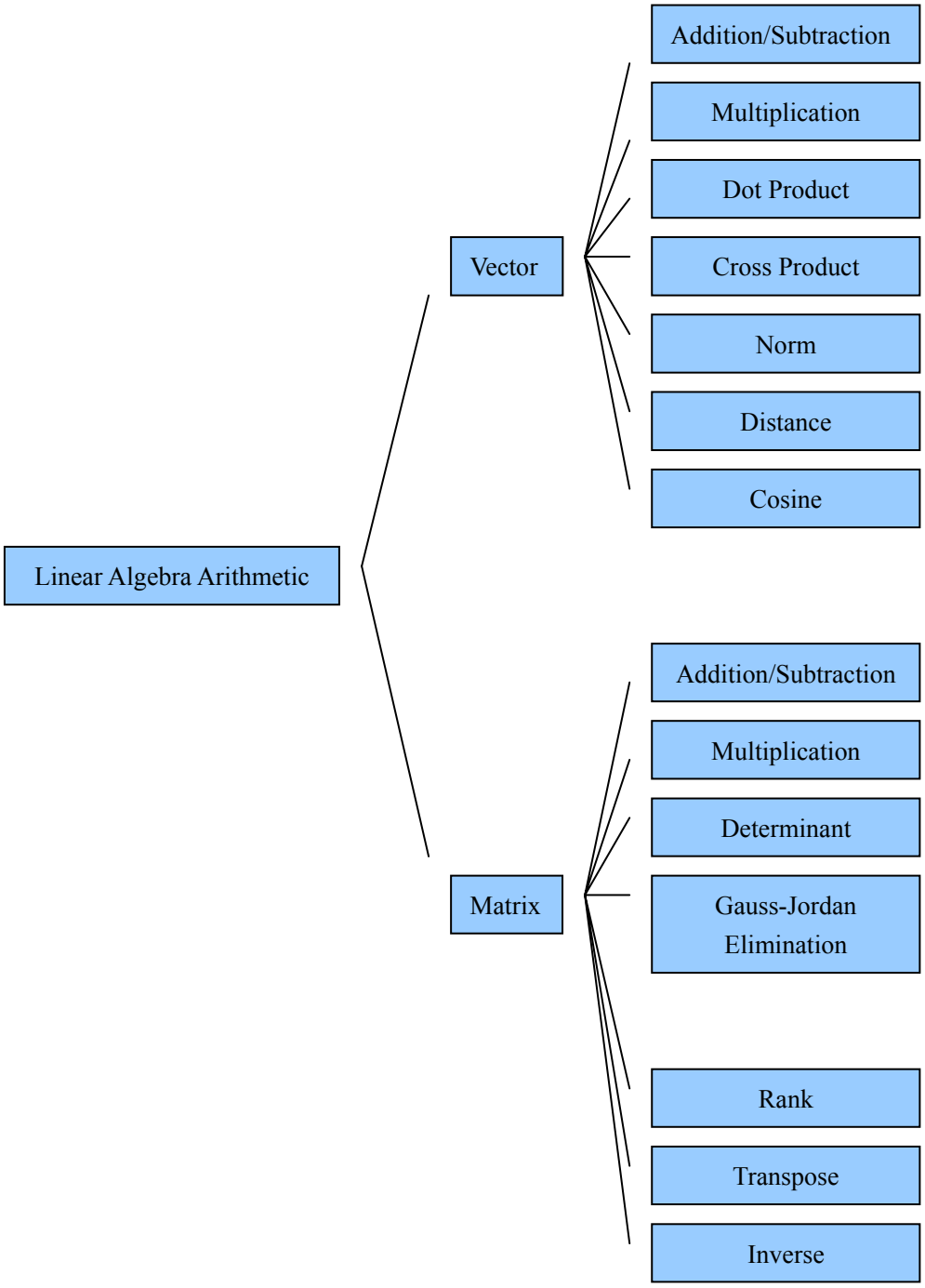
In the literature review part, literature evidence about the project is provided. However, they are still theoretical. In this part, it is the time to analyse these theories to find out what are applicable in the project. In addition, it is necessary to re-consider the required project objectives mentioned in the first part Introduction and discuss them in details. However, it will not be concerned about how to implement these requirements in this chapter, which will be detailed in the system Design part.

3.2 Use Cases

Use cases is an important part to get a better understanding on the requirement of the system. It provides the guideline of analysing literature review and the functional requirement in the next chapter.

3.2.1 Scope

The scope of the use cases is the all fundamental linear algebra arithmetic. It can be divided into two streams, vector operations and matrix operations.



3.2.2 Vector Use Cases

Use case 1. Vectors Addition

Task:	Adding two vectors
Primary Actor:	Users
Preconditions:	Users choose the matrix addition function from the system
Process:	Users input two vectors

	Users press ‘compute’ button
Result:	Users get the result

Use case 2. Vectors Subtraction

Task:	Subtracting two vectors
Primary Actor:	Users
Preconditions:	Users choose the vector subtraction function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 3. Vectors multiplication

Task:	Multiplying two vectors
Primary Actor:	Users
Preconditions:	Users choose the vector multiplication function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 4. Vectors dot product

Task:	Calculating the dot product of two vectors
Primary Actor:	Users
Preconditions:	Users choose the dot product function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 5. Vectors cross product

Task:	Calculating the cross product of two vectors
Primary Actor:	Users
Preconditions:	Users choose the cross product function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 6. Vectors Cosine

Task:	Calculating the cosine of two vectors
Primary Actor:	Users
Preconditions:	Users choose the cosine function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 7. Vectors distance

Task:	Calculating the distance between two vectors
Primary Actor:	Users
Preconditions:	Users choose the distance function from the system
Process:	Users input two vectors
	Users press ‘compute’ button
Result:	Users get the result

Use case 8. Vector Norm

Task:	Calculating the norm of a vector
Primary Actor:	Users
Preconditions:	Users choose the norm function from the system
Process:	Users input one vector
	Users press ‘compute’ button
Result:	Users get the result

3.2.3 Matrix Use Cases

Use case 9. Matrices Addition

Task:	Adding two matrices
Primary Actor:	Users
Preconditions:	Users choose the addition function from the system
Process:	Users input two matrices
	Users press ‘compute’ button
Result:	Users get the result

Use case 10. Matrices subtraction

Task:	Subtracting two matrices
Primary Actor:	Users
Preconditions:	Users choose the subtraction function from the system
Process:	Users input two matrices
	Users press ‘compute’ button
Result:	Users get the result

Use case 11. Matrices Multiplication

Task:	Multiplying two matrices
Primary Actor:	Users
Preconditions:	Users choose the multiplication function from the system
Process:	Users input two matrices
	Users press ‘compute’ button
Result:	Users get the result

Use case 12. Matrix Determinant

Task:	Calculate the determinant of a matrix
Primary Actor:	Users

Preconditions:	Users choose the determinant function from the system
Process:	Users input one matrix
	Users press ‘compute’ button
Result:	Users get the result

Use case 13. Gauss-Jordan Elimination

Task:	Calculate the Reduced row-echelon form of a matrix
Primary Actor:	Users
Preconditions:	Users choose the Gauss-Jordan Elimination function from the system
Process:	Users input one matrix
	Users press ‘compute’ button
Result:	Users get the result

Use case 14. Matrix Rank

Task:	Calculate the Rank of a matrix
Primary Actor:	Users
Preconditions:	Users choose the Rank function from the system
Process:	Users input one matrix
	Users press ‘compute’ button
Result:	Users get the result

Use case 15. Matrix Transpose

Task:	Calculate the Transpose of a matrix
Primary Actor:	Users
Preconditions:	Users choose the Transpose function from the system
Process:	Users input one matrix
	Users press ‘compute’ button
Result:	Users get the result

Use case 16. Matrix Inverse

Task:	Calculate the Inverse of a matrix
Primary Actor:	Users
Preconditions:	Users choose the Inverse function from the system
Process:	Users input one matrix
	Users press ‘compute’ button
Result:	Users get the result

3.3 Project Functions

3.3.1 Linear Algebra Arithmetic

Since this project is about deal with linear algebra field, solving the fundamental linear algebra computations should be the primary requirements, which are:

In Vector Area:

- Vectors Addition/Subtraction
- Vectors Multiplication
- Vectors Dot Product
- Vectors Cross Product
- Vectors Distance
- Vectors Cosine
- Vector Norm

In Matrix Area:

- Matrices Addition/Subtraction
- Matrices Multiplication
- Matrix Determinant
- Matrix Row-Echelon Form
- Matrix Reduced Row-Echelon Form
- Matrix Determinant
- Matrix Rank
- Matrix Transpose
- Matrix Inverse

3.3.2 Advanced Algorithms

As what is discussed in the Literature Review, there are three different algorithms calculating the matrices multiplication and inverse. Recalling the third objective of this project, developing these three algorithms will be the core in the advanced stage.

Although these algorithms have the lower time complexity than the standard ones, they still have some constraints. It is because the Asymptotic Notation is based the big scale of input, which means that it only works properly in large dimension of matrices. For a small dimension matrix, like a 3*3 matrix, the standard algorithms are more applicable. So the system should have options for the users to choose between standard algorithms and these algorithms.

3.3.3 Artificial Intelligence

The reason considering AI in this project is that there is always more than one method when doing calculations on matrices and for matrices with different characteristic one method could take less time then others.

For example, the matrix $A = \begin{bmatrix} 2 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 8 & 7 \end{bmatrix}$, which the determinant is 14.

Either using the standard algorithm or Strassen Algorithm will take more than ten steps to get the result. However, for mathematicians, they would recognise that this matrix is actually a lower triangular matrix, which the determinant is just the product of diagonal entries, 2*1*7=14. This only takes several steps.

In this project, the AI technology will used to calculate the determinant of the matrix. It can be developed in two ways.

The first one is to recognise the matrix form. As what is mentioned in the literature review, if the matrix is in triangular form, then the determinant for this matrix is just the product of its diagonal entries. So there are two steps to implement AI:

1. Analysing the matrix form
2. Calculate the product of its diagonal entries if it is the triangular matrix.

The Second one is to analyse the characteristic of the matrix. If a matrix has a row or column with several zeros, it will be easier to use that row or column entries to multiply with cofactors because the product of multiplying zero and a cofactor is zero. So the steps to implement this AI are:

1. Analysing the matrix and find a row or column with more zeros
2. use that row or column to calculate the determinant

3.3.4 User Interface

The user interface in this project is not the key objective. It does not need a fully functional and professional interface. The most important features are simplicity and user-friendliness, which can make the users start using quickly and even without reading user manual. Considering the scale of the project, the errors or mistakes made by the users are limited. So the system should provide a helpful error message function, which tells users how to fix the error or mistake.

3.4 Number Structure, Input and Output

3.4.1 Number Structure

Either vectors or matrices need numbers for their entries. So the system should be capable of reading user input in different forms. In this stage, it is defined that the number entries should be in three different forms, decimal number, rational number and complex number.

These three different number types should independent exist in the system. However, in maths theory, these three forms can be actually transformed from each one. So although they need to be entity-independent, they also need to be function-dependent, which means, it works that doing a computation between two different types of number.

3.4.2 Data Input and Output

Before using functions in the system to calculate vectors or matrix, the users must input data in the system to create a vector or matrix. These input is actually refer to two aspect, entries and fields. The entries are the number in a certain position of a vector or matrix. The fields are vector and matrix. So the system must have the function to recognise both entries and field from users' input and then create an object in the system. For different field calculation, the system should output the result in different form. Then the users can easily tell if the result is a vector or matrix. Moreover, the entries should also be presented in different form according to their number types.

3.5 Error and Exception Handling

The number type and data input and output are quite complicated, it is easy to have errors and exception when the system running, so in order to make the system robust, it should has a strong error and exception handling functions. Once the error or exception happen, the system can catch it and pop up an error window.

Besides catching errors and exceptions, it is more important to design the system in an error and exception preventing way to make the error and exception can not easily happen. For example, in case of the system using command line input when the users implement a calculation, sometimes they will input the wrong command, which may causes an error. To prevent this, we can set some calculation buttons in the user interface to let users to choose.

3.6 Testing Requirement

System Testing is to verify how well the system actually meets the original requirements. Since this

is a mathematical project, the accuracy of computational outputs is the most important. This requirement must be achieved in all three testing stages, which are component testing, system testing and acceptance testing.

In addition, the logical inherence in linear algebra arithmetic is quite strong, for example, the matrices multiplication requires the calculation of matrices addition. So it is not appropriate to start next component until the current one is completed and tested.

In the system testing stage, the system integration shall be tested. Since each linear algebra arithmetic component had been already tested in the previous stage the integration for this part would not be a problem. The data input and output testing is the key in this level, because they may belong to different types.

The acceptance testing needs to consider every circumstance that errors and mistake made by user lead to cause a problem.

3.7 General Constraints

3.7.1 Operating System and Hardware Requirement

The system is required to have a high compatibility (Objective 4), which means it can run in different main Operating Systems, like Windows XP, Mac OS, Unix and Linux. Moreover, the system should work on the recent five year computers.

3.7.2 Programming Language

Regarding to the scope of this project, any main stream programming language can work well, like C, C++ and Java. C and C++ all have memory allocation function to mange memory using. However, the developer would prefer using Java to write the program. It is because firstly considering the scale of the project, it will only occupy a big amount of memory. Secondly, java has decent and easy-to-use graphic package to fulfil the requirements on the system interface. Thirdly, the Java Virtual Machine will ensure the compatibility of the system since there are few operating systems cannot run JVM.

3.7.3 Time Constraints

This is one term project. But actually, most of the core works started from the second semester and in the meanwhile, there are other courses and project to work on. So there is actually not too much time. So it could happen that some requirements will not be achieved finally.

3.7.4 Users of the system

The users for this project should be students or others who just want to implement some quick calculation on vectors or matrices rather than some professionals who want to do some complicated operations.

4 Requirement Specification

4.1 Introduction

The requirement specification focuses on both user and system requirement. It can be seen as an advanced stage developed from requirement analysis, which summarise the requirements and then represent them in a more design like form. In other word, it is the bridge between gathering user's need and telling the developer what exactly need to be designed and implemented. So it may repeat the same content which has been mentioned before, but in a more practice relating way. But for some content, like the requirement operating system, hardware and interface, they will not be mentioned in here because they have already discussed enough in the previous chapter and the on

4.2 Functional Requirements

They are about what functions and service the system should provide. It may involve the calculations, technical details, data manipulation and processing, and other specific functionality that show how the use cases are to be satisfied. In this project the functional requirement is relating to the linear algebra arithmetic, which has two streams, vector calculation and matrix calculation.

4.2.1 Vector Functions

Index	1
Name	Vector Addition(Scalar)
Summary	1. User input the vector and scalar 2. Then the system outputs the result of adding a vector to a scalar
Rationale	This is the basic vector arithmetic

Index	2
Name	Vectors Addition(Vectors)
Summary	1. User input two vector 2. Then the system outputs the result of adding a vector to another vector
Rationale	This is the basic vector arithmetic

Index	3
Name	Vector Subtraction(Scalar)
Summary	1. User input the vector and scalar 2. Then the system outputs the result of subtracting a vector to a scalar
Rationale	This is the basic vector arithmetic

Index	4
Name	Vectors Subtraction(Vectors)
Summary	1. User input two vector 2. Then the system outputs the result of subtracting a vector to another vector
Rationale	This is the basic vector arithmetic

Index	5
Name	Vector Multiplication(Scalar)
Summary	1. User input the vector and scalar 2. Then the system outputs the result of multiplying a vector by a scalar
Rationale	This is the basic vector arithmetic

Index	6
Name	Vector Multiplication(Vector)
Summary	1. User input two vectors 2. Then the system outputs the result of multiplying two vectors
Rationale	This is the basic vector arithmetic

Index	7
Name	Vector Multiplication(Matrix)
Summary	3. User input a vector and matrix 4. Then the system outputs the result of multiplying the vector and matrix
Rationale	This is the basic vector arithmetic

Index	8
Name	Dot Product
Summary	1. User input two vectors 2. Then the system outputs the dot product of two vectors

Rationale	This is the basic vector arithmetic

Index	9
Name	Cross Product
Summary	1. User input two vectors 2. Then the system outputs the cross product of two vectors
Rationale	This is the basic vector arithmetic

Index	10
Name	Vectors Distance
Summary	1. User input two vectors 2. Then the system outputs the distance between two vectors
Rationale	This is the basic vector arithmetic

Index	11
Name	Vector Cosine
Summary	1. User input two vectors 2. Then the system outputs the cosine between two vectors
Rationale	This is the basic vector arithmetic

Index	12
Name	Norm
Summary	1. User input the vector 2. Then the system outputs the norm of the vector
Rationale	This is the basic vector arithmetic

4.2.2 Matrix Functions

Index	13
Name	Matrices Addition
Summary	1. User input two matrices 2. Then the system outputs sum of two matrices
Rationale	This is the basic matrix arithmetic

Index	14.
Name	Matrices Subtraction
Summary	1. User input two matrices

	2. Then the system outputs result of subtracting two matrices
Rationale	This is the basic matrix arithmetic

Index	15.
Name	Matrices Multiplication(Scalar)
Summary	1. User input one matrix and a scalar 2. Then the system outputs product of two matrices
Rationale	This is the basic matrix arithmetic

Index	16.
Name	Matrices Multiplication(Matrix)
Summary	3. User input two matrices 4. Then the system outputs product of two matrices
Rationale	This is the basic matrix arithmetic

Index	17.
Name	Matrices Multiplication(Strassen Algorithm)
Summary	1. User input two matrices 2. Then the system outputs product of two matrices by using Straseen Algorithm.
Rationale	It is required from the objective of the project

Index	18.
Name	Matrices Multiplication(Coppersmith–Winograd algorithm)
Summary	1. User input two matrices 2. Then the system outputs product of two matrices by using Coppersmith–Winograd algorithm
Rationale	It is required from the objective of the project

Index	19.
Name	Gaussian Elimination
Summary	1. User input the matrix 2. Then the system outputs the Row-Echelon Form of this matrix
Rationale	This is the basic matrix arithmetic and also be used for other functions

Index	20.

Name	Gauss-Jordan Reduction
Summary	1. User input the matrix 2. Then the system outputs the Reduced Row-Echelon Form of this matrix
Rationale	This is the basic matrix arithmetic and also be used for other functions

Index	21.
Name	Determinant
Summary	1. User input the matrix 2. Then the system outputs the determinant of this matrix
Rationale	This is the basic matrix arithmetic

Index	22.
Name	Rank
Summary	1. User input the matrix 2. Then the system outputs the Rank of this matrix
Rationale	This is the basic matrix arithmetic

Index	23.
Name	Matrix Inverse
Summary	1. User input the matrix 2. Then the system outputs the inverse of this matrix
Rationale	This is the basic matrix arithmetic

Index	24.
Name	Matrix Inverse (Blockwise)
Summary	1. User input the matrix 2. Then the system outputs the inverse of this matrix
Rationale	This is required

Index	25.
Name	Transpose
Summary	1. User input the matrix 2. Then the system outputs the Transpose of this matrix
Rationale	It is required from the objective of the project

4.2.3 Other functions

Index	26.
Name	Error Message
Summary	When user input is wrong or make some unexpected operations, the system pop up a error message to tell user how to fix errors.
Rationale	It will make the user interface more friendly.

4.3 Non-Functional Requirements

Non-Functional Requirements refer to how well the system functions are delivered, which include timing constraints, constraints on the development process and standards(11).

Index	27.
Name	Reliability
Summary	The reliability is about how reliably the system work. For this project, it focuses on the accuracy of doing calculation. It is not tolerable to fail getting the right result of calculation.
Rationale	This is what the system must achieve. No matter what functions the system provide or how fast the system run, if the system can not work probably or fail in a high rate, the system is worthless.

Index	28.
Name	Compatibility
Summary	The system should work on recent computers and usual operation system
Rationale	There is no need to writhe different version for this scale of system for running on different OS. So the compatibility must be high in this system.

Index	29.
Name	Maintainability
Summary	The system should allow for new developers to carry out maintain in a efficient manner.
Rationale	The scope for this project is still small comparing to other maths tools and the functions concerning in this project is narrow. So the system might be updated in the future. This is also from the secondary objectives.

Index	30.

Name	Ease of use
Summary	It should have a friendly user interface. User do not even need to read the guideline to use it.
Rationale	Because this project is designed for the users who want to do some simple linear algebra calculations but do not spend hours to figure out how to use matlab or maple. So the friendly user interface is very necessary.

5 System Design

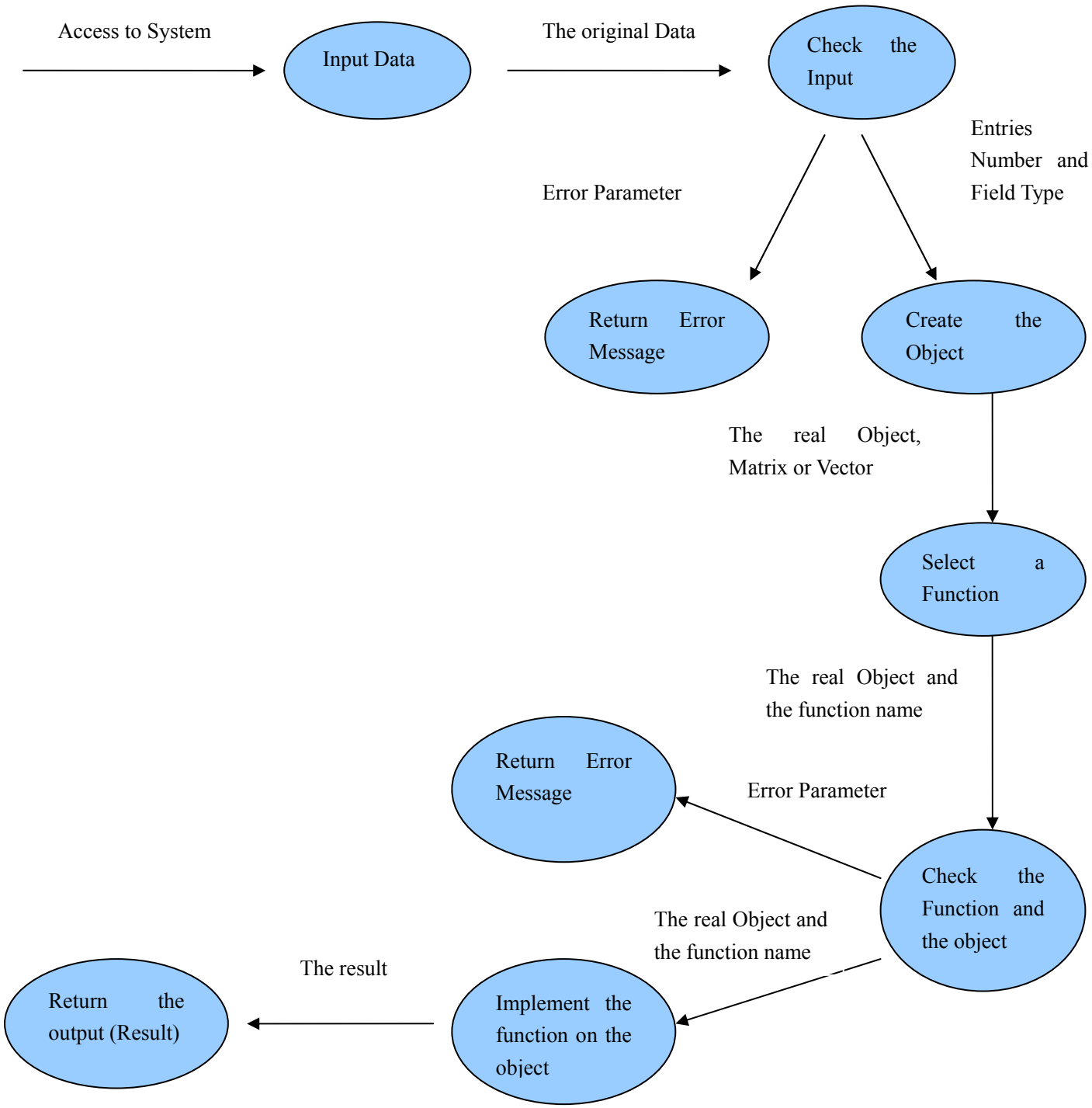
5.1 Introduction

The system design is the most important part of a software development process, which decides the logical organisation of the software. The easiest way to illustrate it is to use models in UML language. The models using in this part are System Models and Software Models.

5.2 System Models

The system models are about representing users requirement in a technical way. For different models, they focus on the different perspectives of the system design. Considering the system itself, the most important models are Data Flow Model, which shows how data is processed.

5.2.1 Data Flow Model



There are two main stages in this model. The one is from the beginning to 'Create an Object'. In this stage, the system read the users input and transform them into an object form, a matrix or a vector. However, if the input is not acceptable in the system, the system will show an error message and terminate the process by sending an error parameter to the error function.

Then the system passes the object into the next stage. In this stage, after user choosing the function to implement on the object, like calculating the determinant of the matrix, the system will check if the function is valid on this object. If it is not a valid function, such as calculating the determinant of

a 3*2 matrix, the system shows an error message and terminate the process. Or the system implements the function on the object and then returns the result.

5.3 Object-Oriented Design

Object-Oriented Design is about developing an object-oriented model of a software system to reflecting the identified requirements. It involves designing object classes and relationships between these classes. Then these classes define objects in the system and the interaction between objects.

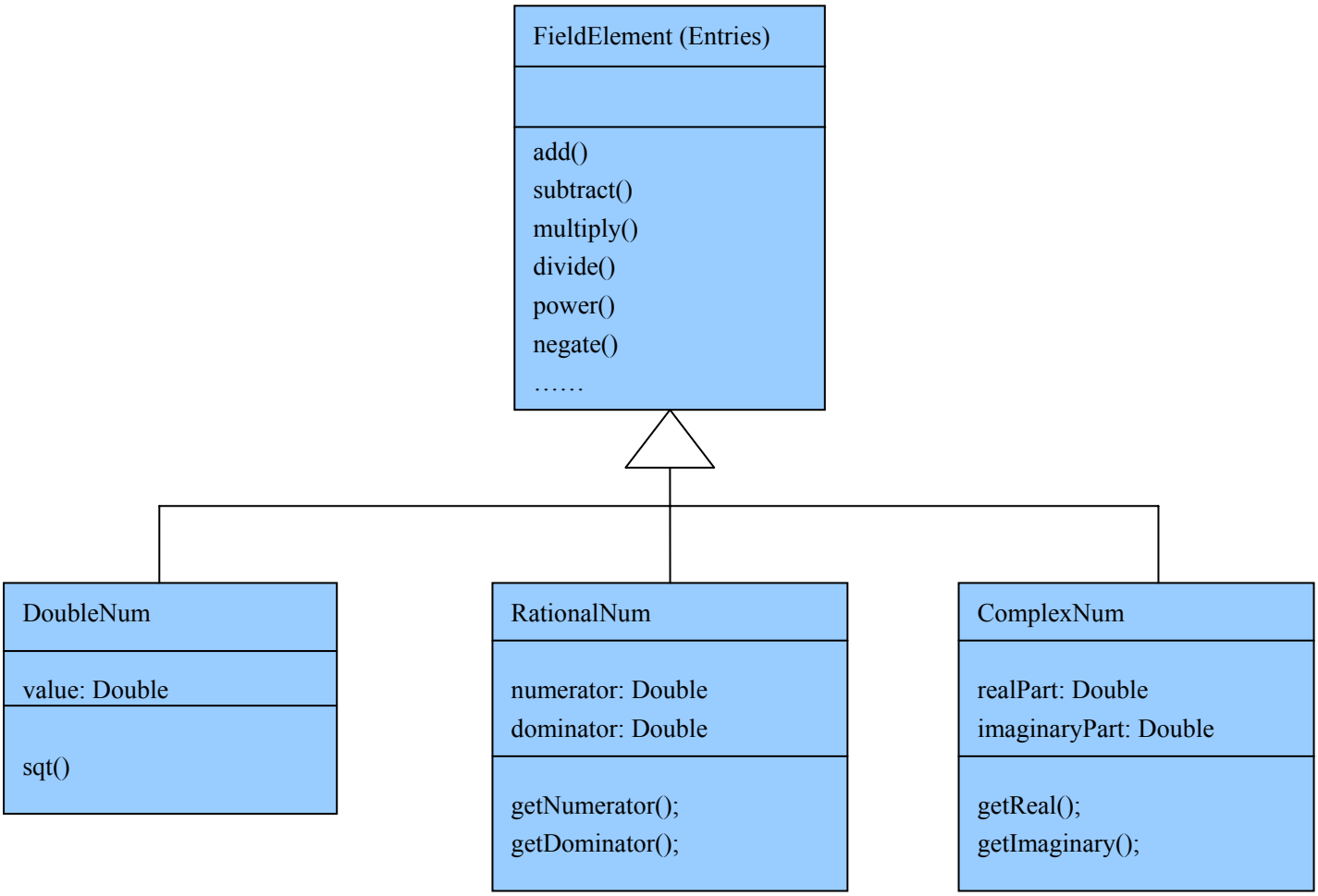
5.3.1 Object Class Model

The object class model is to encapsulate information in objects. For every object, it has attributes and functions. In programming language, the objects are represented in the forms of classes. There are four different groups of classes in this project.

- FieldElement is the entries in vectors and matrices
- Field is referring to vectors and matrices
- Operators is about this operations on FieldElements
- Advanced algorithms reflect the advanced algorithms mentioned in the requirement

The object class is represented as a named rectangle with two sections, which are the attributes and methods from top to bottom. It is using inheritance hierarchy to show the relationship between classes in the same group. The child class inherits attributes and methods from the parent class.

FieldElement

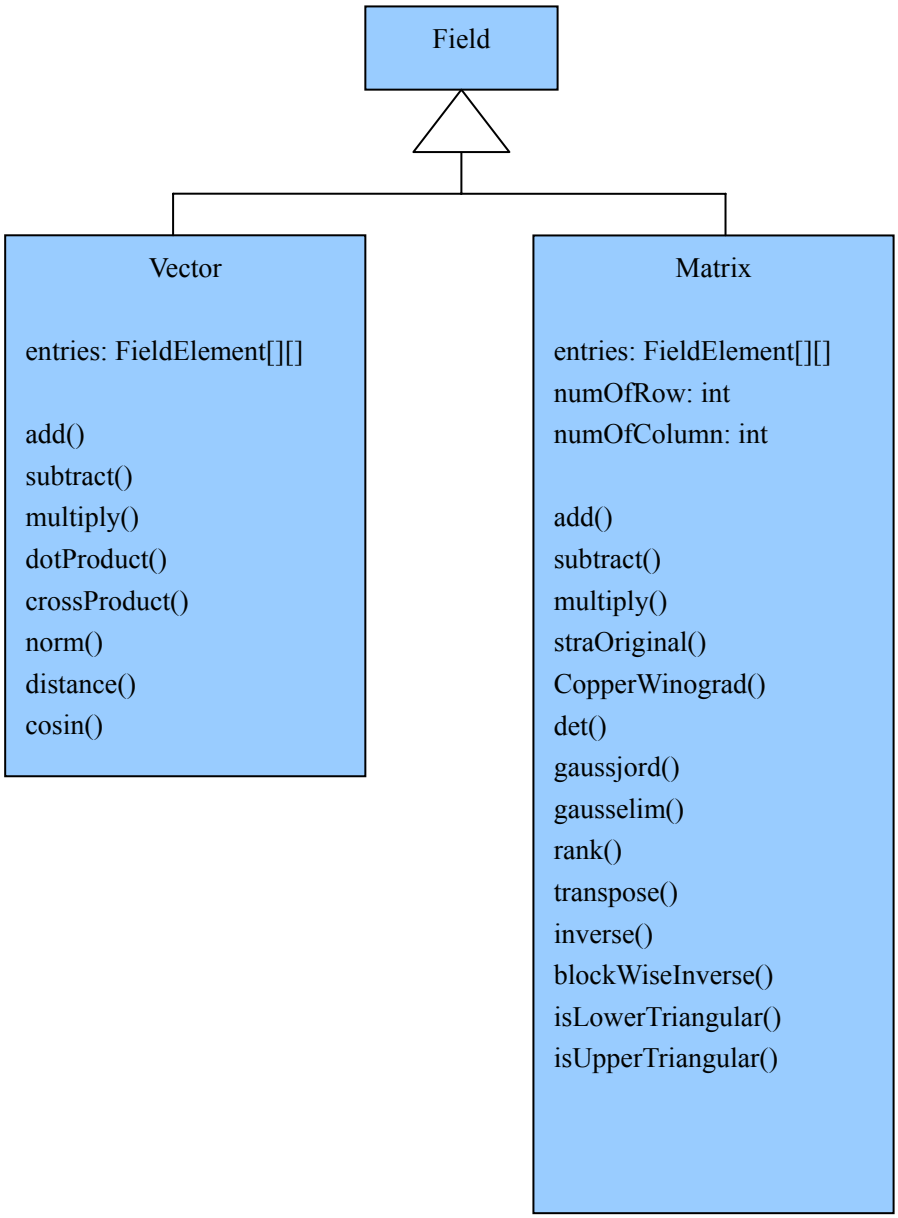


As it is required in the Data Structure in Requirement Analysis section, the project should work on three different types of number as input and output. So the system uses DoubleNum, RationaNum and ComplexNum to represent three types of number, which are decimal number, rational number and complex number respectively.

Since rational number is in the form of ‘a/b’, which a and b are any integers, the numerator as ‘a’ and dominator as ‘b’ are defined in RationalNum class. The complex number is in the form of a+b*i so realPart as ‘a’ and imaginaryPart as ‘b’ are defined in the complexNum.

The methods defined in this model are all some general mathematic arithmetic, like addition, multiplication.

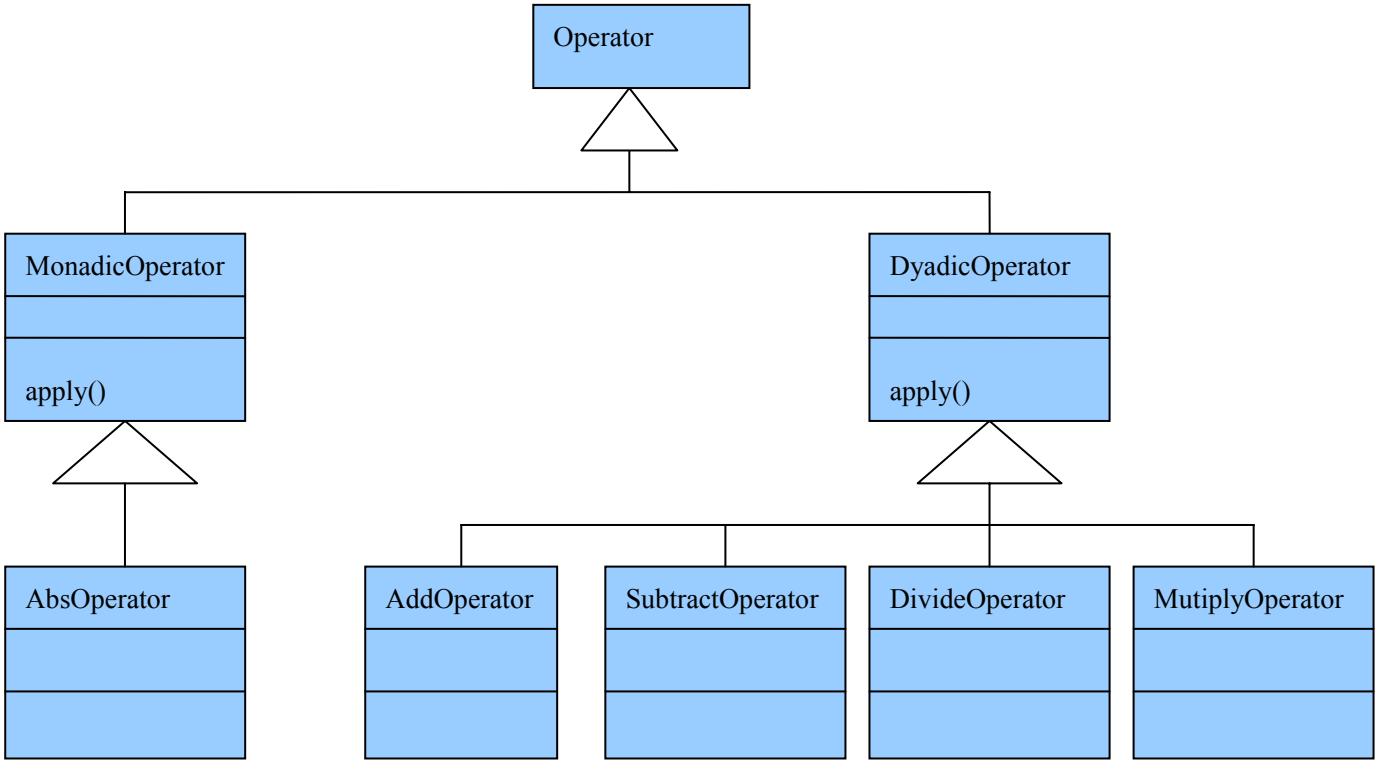
Fields



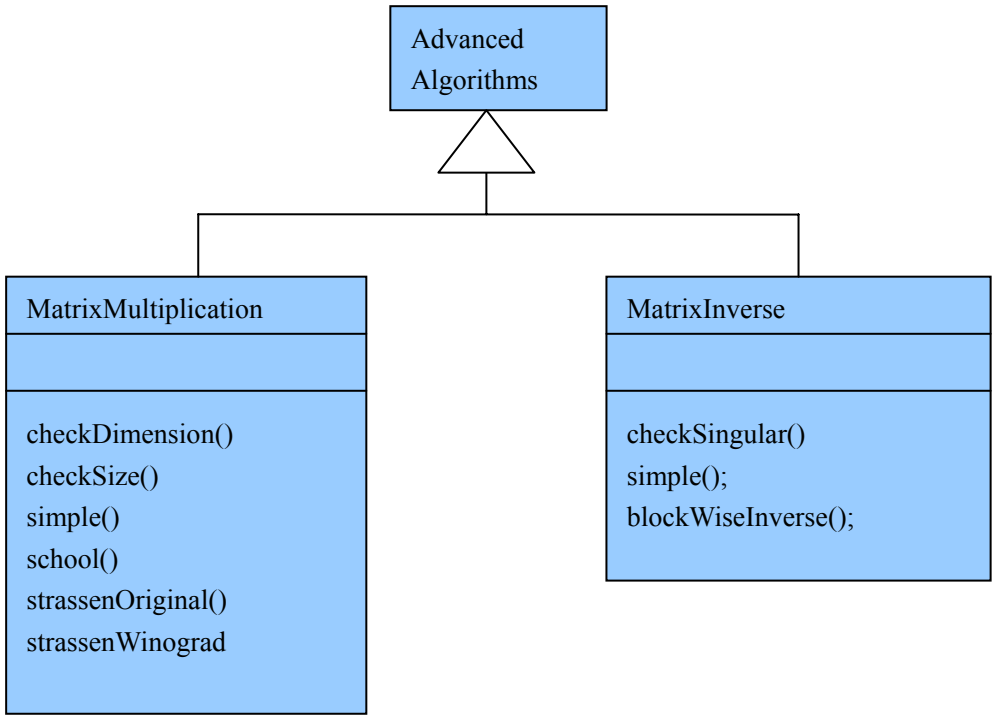
There are two kinds of field types in this system, vector and matrix. They are using FieldElement as entries. In matrix, the attributes numOfRow and numOfColumn refer to the current row or column index. They will be used in the afterward functions.

Each method corresponds to one function in Functional Requirement in the last chapter. The methods ‘straOriginal()’ and ‘CopperWinograd()’ will call the responding method in the advanced object. ‘det()’ is to calculate the determinant of a matrix. ‘eig()’ is to calculate the eigenvalue of a matrix. ‘gaussjord’ is to change the matrix into row-echelon form and gausselim is to change the matrix into reduced row-echelon form.

‘isLowerTriangular’ and ‘isUpperTriangular’ are used to check the matrix form in order to apply AI to calculate the determinant. This will be discussed in the AI design.



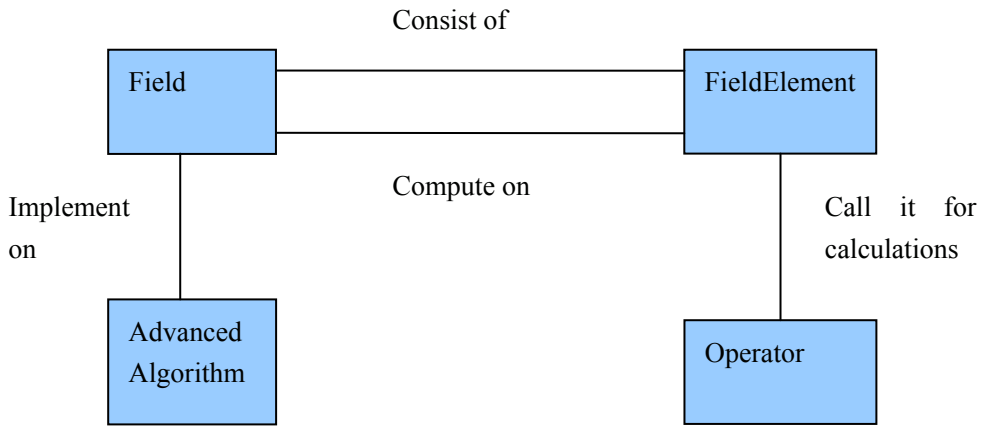
These operators are used to do some general calculations defined in 'FieldElement'. The 'MonadicOperator' is used to implement arbitrary monadic (one-argument) functions to the Fieldelement. The 'DyadicOperator' is about doing arbitrary dyadic (two-argument) functions.



These classes are using Strassen Algorithm, Coppersmith – Winograd Algorithms and Block-wise as theory to calculate the multiplication and inverse of matrices. They also have the 'simple' methods to use the standard ways to do calculation.

The 'checkDimension()' in 'MatrixMultiplication' is to check if two matrix have right dimension to multiply with each other. The 'checkSize()' is to check if the dimension of two matrix meet the pre-condition of using Strassen Algorithm or Coppersmith – Winograd. The checkSingular() is to check if the matrix is invertible. If it is invertible, the system will do the inverse or the system terminates the process.

5.3.2 Class Association Model



The relation between classes is that the Field (vector or matrix) is firstly constructed with FieldElements as entries. When the Field wants to implement some function or use the advanced algorithms, it actually do the calculation on FieldElements. The FieldElement will call Operator to finish some calculations.

5.4 AI Design

The AI requirement in the 3.3 mentioned about two methods of implement AI.

The first method can be achieved by checking the matrix form in the early stage of calculating the determinant of a matrix. The second methods after the careful consideration will be implemented in this way.

Although after analysing the matrix feature and find the row or column has the most zeros, it still uses the standard algorithm to calculate the determinant of matrix, which has the complexity of n! **(12)**. It means in a n*n matrix, even there are n-1 zeros in a row, the complexity of calculating the

determinant is still big. So the system will only check if there is a row or column of 0s. If there is, the system will return 0. Or the system uses Gaussian elimination to reduce the matrix into upper triangular form then multiply the main diagonal to get the determinant. The operation of doing Gaussian elimination is about $n^3/3$, which the complexity is $O(n^3)$.

5.5 User Interface Design

Since the simplicity and user familiarity are the main goals of the interface, it should be designed in a user friendly way, which means the users can easily find the function they want to use and be clearly aware of what they doing. This is the draft user interface.

<table>
<tr><td colspan="5">Menu Bar</td></tr>
<tr><td>Vector A</td><td>Vector B</td><td>Matrix A</td><td>Matrix B</td><td>Result</td></tr>
<tr><td colspan="5">Vector or Matrix Creating Tool Bar
Vector or Matrix size input NumberType Selcetion combo-box Create Button</td></tr>
<tr><td colspan="5">Vector or Matrix Entries</td></tr>
<tr><td colspan="5">Function Tool Bar
Function Selection combo-box Scalar input vector or matrix selection combo-box implement button</td></tr>
</table>

There are five components in this interface design. The first one is the menu bar which has menu items in it. Since the user interface is the secondary objective in this project, there will only be some simple functions, like clear the result panel, quit the system some system information functions.

The second part is the tabbed panel, which has Vector A, B, Matrix A, B and Result panels. The user can easily to go to Vectors or Matrices created by clicking on their names.

The Vector or Matrices Creating Toolbar is used to create vectors or matrices. The user define the size of a vector or matrix and number type in here.

The biggest part is used to let the users to input numbers into vectors and matrices.

The last one is the function toolbar, which let the users to choose the functions to work on vectors or matrices.

5.6 Non-functional Requirement Design

5.6.1 Minimise the scope of local variables

By minimise the scope of local variables, the readability and maintainability will increased(13). Fox example, if one variable is only used in the for loop of one method in the middle of the whole class code but it is declared in the constructor then the scope for this variable is too big. When people read the code, before they really read the method using this variable, they have already the variable's type and initial value. Moreover, if the program evolves and the variable is no long used, it is easy to forget to remove the declaration of the variable. So the best way for minimising the scope of local variable, it is to declare it where it is first used.

5.6.2 Error and Exception Handling

Using Error and Exception Handling can increase the reliability and robustness of a program. It can be achieved in two stages.

Stage one: finding all possible errors and exceptions

1. We can go through each use cases to find all possible circumstance of users make a mistake to case the errors and exceptions
2. Analysing the relation between objects and their functions to check if they conflicts with each other
3. Since the objects in this system are all related with each other. The attribute of one object is same for itself but maybe dangerous for others. So does the functions. We need to find these potential problem

Stage two: handling these errors and exceptions

Besides using exception package in Java Api Library, we also need to create some new exception objects. Then we catch these exceptions from where the system calls the functions, which are possible to cause the errors and exceptions. This will be discussed in detail in Implementation.

6 Implementation

6.1 Introduction

This chapter is about how the design is actually implemented. In other words, it illustrates how the developer writes the program to reflect the requirements. It also includes some specific problems and solutions happened in the process of coding. The structure of this chapter will basically follow the object-class model to discuss how these objects implemented both in high level discussion and some code explained in detail.

6.2 Number Structure and General Functions

The number here does not only mean the decimal, rational and complex number. It also involves the vector and matrix, which is the basic ‘number’ of Linear Algebra. For different types of number, there are some different functions.

This section will talk about three main number types, which are FieldElement, Vector and Matrix and their functions.

6.2.1 FieldElement

FielddElement is the actual value of entries in vectors or matrices. It is the super class of other types of number, which are DoubleNum, RationalNum, ComplexNum. Some basic methods need to be defined in this class. However, the arithmetic is different from these three numbers. So most methods will be empty, which only give a basic structure for subclasses.

Because different types of number have different attributes and use different arithmetic, there will be a problem when they work together. Basically, there is a hierarchy in these three number types, firstly ComplexNum, then RationalNum, and DoubleNum comes last. So when computing on two different types of number, the number with lower hierarchy will change the form to accommodate the higher hierarchy..

6.2.1.1 **DoubleNum**

This class inherits from FieldElement. It is used to store the decimal number. When the system create a DoubleNum instance, it pass the decimal number into its field, ‘value’

```
protected double value;
```

When the system operates on the DoubleNum object, it actually calculates the ‘value’.

6.2.1.2 Rational Number

The rational number should be in the form of a/b, which a and b are integers. So in rational number, there are two variables declared in the field, numerator and dominator to represent a and b. They are all ‘BigInteger’ type, which can store long integers then ‘int’ type.

Constructor Problems

There are two problems need to be solved when initialising the Rational Number object. The first one is the rational number input by the user is not factorised. To solve this problem, the system called the cancel() to normalise the rational number. The cancel() is a method in the RationalNum class, which uses the gcd() method from BigInteger Class to find the greatest common divisor and

then divide the both numerator and dominator to get the factorised rational number form.

The second problem is to change the decimal number or integers into rational number form. This is has been mention in the Literature Review Rational Number section.

6.2.1.3 **ComplexNum**

The complex number has the form of a+bi, where a and b are real numbers, i is the imaginary unit. In the system, it uses the realPart variable as 'a' and imaginaryPart variable as 'b'. The type of realPart and imaginaryPart is both RationalNum. The reason to do this is to make it easier to change the DoubleNum object into a ComplexNum object.

6.2.2 Vector

It uses a single FieldElement type array to store entries. The Vector functions iterate on every FieldElement objects stored in it and call the objects' operations. For different Vector operations, the system will call different FieldElement's methods.

Fox exmaple

Norm

```
public FieldElement norm()
  {
      FieldElement temp=new DoubleNum(0);
      for(int i=0;i<entries.length;i++){
             temp=temp.add(entries[i].power(2.0));
         }

      }
          return temp.power(0.5);

  }
```

This method is to calculate the norm of a vector. The 'for' loop goes through every entry of the vector. The statement in the 'for' loop is to apply the norm algorithm to calculate the norm. `'temp=temp.add(entries[i].power(2.0));',which` adds all entries after square power.

Operators

In vectors operations, the add, subtract, multiply and divide functions use the 'operator' concept. This is designed for increasing the maintainability of the system.

```
public Vector add(Vector anotherVector) throws
InvalidOperationException {
```

```
        return operate(this, anotherVector, new AddOperator(), "add");
    }
```

Step 1: when using add() method to add one vector with another, the system calls operate(`this, anotherVector, new AddOperator(), "add"`) method.
Step 2: Checks the length of two vectors first. Only if the check_length() is successful the system will continue to next process.
Step 3: The system goes through each FieldElement objects in the Vector array and invokes the 'apply()' method in AddOperator class.
Step 4: then 'apply()' method calls the add() method in FieldElement class to adds two FieldElement object and return the result of the addition.
Step 5: Storing each FieldElement objects addition result into the new vector.
Step 6: The operate(…..) method returns the new vector which is the result of adding two vectors.

All other Operators work in a similar way. There are two difference operator interface. One is DyadicOperator interface, which supports application of arbitrary dyadic (two-argument) functions to the elements of two Matrix or Vector objects, via the Matrix or Vector's method. The other one is the MonadicOperator interface, which supports application of arbitrary monadic (one-argument) functions to the elements of a Matrix or Vector, via the Matrix or Vector's apply methods.

The AddOperator, SubtractOperator, MultiplyOperator and DividOperator implement the Dyadic Operator interface and AbsOperator implements the MonadicOperator..

6.2.3 Matrix

The Matrix class uses double FieldElement type array to store entries. Some of matrix functions also use the operator classes. The matrix functions can be divided into three levels. The high level functions cannot work until the lower level functions are work properly and the algorithm become more complicated as the level getting further.

6.2.3.1 First Level

The main functions in the first level are addition, subtraction, transpose, the functions of checking if the matrix in lower-triangle or upper-triangle forms, getting rows or columns and swapping rows or columns. The addition and subtraction in Matrix are quite similar to those in Vector. The only difference is the matrix is double array, which means getting entries need an extra pair of indices.

For example:

Transpose

```
    public Matrix transpose() {
        Matrix tmp = new Matrix(this.getCols(), this.getRows());
        for (int row = 1; row <= this.getRows(); row++) {
```

```
                for (int col = 1; col <= this.getCols(); col++) {
                    tmp.set(col, row, this.get(row, col));
                }
            }
            return tmp;
        }
```

There is a temporary matrix tmp defined in this method. It is using two 'for' loops to get every entry and set the entry in the ith row and jth column in the original matrix to the jth row and ith column by invoking this line of code '`tmp.set(col, row, this.get(row, col));`'.

`isLowerTriangular()`

This function was a little bit tricky to code. Firstly, if the certain entry is not zero as expected, then the whole process can stop no matter loop is still on the way. Secondly, there is no need to check every entry but only the entries above the diagonals. The first problem is solved by a 'if' 'else' statement. If the certain entry is not zero, the method go to 'else' statement and returns false to shut down the loop.

```
for(int i=1;i<this.numOfRows;i++){
        for(int j=i+1;j<=this.numOfCols;j++){
            if (this.get(i, j).isZero()){

            }
            else{
                return false;
            }
```

The second one is solved by set the value of j in the second 'for' loop equal to i+1. Then for every time the outer 'for' loop recurs and i increments. j counts from i+1. It makes the method only check the entries above the diagonals.

`isUpperTriangular()`

The method to check upper-triangle is similar to check for lower-triangle. The only difference is the code in the second 'for' loop is like this

```
for(int j=1;j<=this.numOfCols-i-1;j++){
            if (this.get(i, j).isZero()){

            }
            ... ...
```

So the postcondition for j is decrement every time the loop repeats. Then the method will only check the entries below the diagonals.

getRow(), getColumn()

The method getRow(int) or getColumn(int) returns a new Vector object with the same entries as the ith row or column of a Matrix.

Swapping rows or columns(swapRows(), swapColumns()**)**
The function of swapping rows or columns in a matrix swaps two rows or columns indices. So this is done by calling getRow of getColumn to get ith row or column and store it in a temporary vector variable. Then it gets jth row or column and set this row or column as ith row or column and set the temporary vector as the jth row or column.

6.2.3.2 Second Level

The functions in this level are multiplication, Gaussian elimination and Gauss-Jordan Reduction.

Multiplication
Coding the standard multiplication function is actually simple. It is just about using 'for' loop to find the corresponding entries and multiply the FieldElement objects on those entries and the return new FieldElement objects for the entries in the new matrix. This method is from MatrixMultiplication Class.

```
public static Matrix simple(Matrix m1, Matrix m2)
     throws InvalidOperationException {
     checkDimensions(m1, m2);

     int resultRows = m1.getRows();
     int resultCols = m2.getCols();

     Matrix resultMatrix = new Matrix(resultRows, resultCols);

     for (int i = 1; i <= resultRows; i++) {
         for (int j = 1; j <= resultCols; j++) {
             resultMatrix.set(i, j,
             m1.getRow(i).multiply(m2.getCol(j)));
         }
     }
     return resultMatrix;
  }
```

The checkDimensions is to check the dimensions of two matrix to see if they can be multiplied. The 'm1.getRow(i)' and 'm2.getCol(j)' in the second 'for'loop return the ith row of matrix 'm1' and the jth column of matrix 'm2' as two vectors. So instead of getting each entry of two vectors and multiply them, the system uses vector multiplication function, which is resulting dot product in fact. And this dot product is just the entry in the new matrix.

There are the other two method in this MatrixMultiplicaion class. But those two will be discussed in the Advanced Algorithms session.

Gaussian Elimination(gausselim()**)**
This function is to change the matrix into the row-echelon form by using row operations. It uses the

getRow() and swapRows() function from the first level and operations in Vector class.

There are two steps in this method.

1. Find the non-zero diagonal entry.

Firstly, it check the entry in the first diagonal. If the diagonal entry in the current row is 0, then it swaps with another row with non-zero entry in the first column if the whole column entries are all zeros, it goes to the entry next to diagonal and do checking again.

```
if (diagonalEntry.isZero()) {
            // search for non zero entry
            boolean found = false;
            for (int candidate = row + 1; candidate <= tmp.getRows();
candidate++) {
                if (!tmp.get(candidate, colCounter).isZero()) {
                    tmp.swapRows(row, candidate);
                    found = true;
                    break;
                }
            }
            if (!found) {
                if (colCounter == tmp.getCols()) {
                    return tmp;
                }

                row--;
                continue;
            } else {
                    diagonalEntry = tmp.get(row, colCounter);
                }
            }
```

The ‘if’ is to check the diagonal entry and the ‘for’ loop is to find a row with non-zero entry can be swapped. If the row is found, the ‘for’ loop breaks. The second ‘if’ is about checking if the non-zero entry can be found or not. So if it is not found and the process is in the last column, it returns the ‘tmp’, which is the matrix to store the row-echelon form of the original matrix. Or ‘row--‘ and ‘continue;’ to go to the entry next to the diagonal. However, if it is found, it set the current diagonal entry to diagonalEntry to do the later process.

2. After finding the non-zero diagonal, it implements the row operations to make the entries (denoted as non-diagonal entry in the following) with same column index as the diagonal and row index below the diagonal zero. To do this, it needs to factorise the row (denoted as non-diagonal row in the following) in which non-diagonal entry in. It can be done by multiply non-diagonal rows by factors, which is resulted from diagonal entry by non-diagonal entries. To make the non-diagonal entries zero, it use diagonal row to subtract the non-diagonal rows and set the returned result as the new entries in the corresponding index in non-diagonal rows.

```
for (int j = row; j <= tmp.getRows(); j++) {
```

```
            FieldElement factor = tmp.get(j, colCounter).divide(
                    diagonalEntry);
            if (row == j || factor.isZero()) {
                continue;
            }

    for (int k = colCounter; k <= tmp.getCols(); k++) {
                FieldElement oldEntry = tmp.get(j, k);
            tmp.set(j, k, oldEntry.subtract(factor.multiply(tmp.get(
                        row, k))));
            }
        }
```

The first 'for' is factoring the diagonal row. The second 'for' is factoring non-diagonal rows and then operate on the non-diagonal rows to make the non-diagonal entries zero.

3. After finishing the first diagonal, go to the next diagonal and repeat 1 and 2.

Gauss-Jordan Reduction (gaussjord()**)**

It is quite similar as Gaussian Elimination, but get the reduced row-echelon form. The only difference is the step 2.

Step 2 for this

After finding the non-zero diagonal, it implements the row operations to make the diagonal one and the entries (denoted as non-diagonal entry in the following) with same column index zero. To make the diagonal one, in other word of factorising the row, it can be done by divide the entries in this row (denoted as diagonal row in the following) by the value of this diagonal. To make the non-diagonal entries zero, it is done by multiplying the diagonal row by the non-diagonal entires and then using diagonal row to subtract the other non-diagonal rows.

```
    for (int j = colCounter; j <= tmp.getCols(); j++) {
                FieldElement oldEntry = tmp.get(row, j);
                tmp.set(row, j, oldEntry.divide(diagonalEntry));
            }

            for (int j = 1; j <= tmp.getRows(); j++) {

                FieldElement factor = tmp.get(j, colCounter);
                if (row == j || factor.isZero()) {
                    continue;
                }

                for (int k = colCounter; k <= tmp.getCols(); k++) {
                    FieldElement oldEntry = tmp.get(j, k);
                    tmp.set(j, k, oldEntry.subtract(tmp.get(row,
```

```
k).multiply(
                        factor)));
            }
        }
```

There are two differences in code comparing with gausselim(). One is that it factorise the diagonal row in the first ‘for’ loop. The second difference is in the second ‘for’ loop, ‘j’ counts from 1, and the ‘j’ counts from ‘row’, which is the current row number. So the So gausselim() only changes the entries below the diagonal with the same column index to zero, gaussjord() changes all entries with the same column index as the diagonal.

6.2.3.3 Third Level

There are six functions in this level, which are det(), inverse(), rank() from Matrix class and straOriginal(), CopperWinograd() from MatrixMultiplication class and blockWiseInverse() from MatrixInverse class. The det(), straOriginal(), CopperWinograd() and blockWiseInverse() will be discussed in the next session.

inverse()

This method is to calculate the inverse of a matrix. This method calls the simple() method from MatrixInverse class. The algorithm using in simple() is the standard algorithm to calculating the inverse of the matrix, which is adding a identity matrix at the back of the matrix first and find the reduced-echelon form of that matrix, then the last a few columns of the matrix is the inverse of the original matrix.

However, the actual implementation is a little bit different from the algorithm. It uses two temporary matrices ‘tmp’ and ‘tmp2’ representing the original matrix and the inverse. Then it calculated the reduced row-echelon form for tmp. For every row operations during this process, it also works on tmp2 to makes tmp and tmp2 like one matrix. After finishing this, the system checks tmp to see if it is an identity matrix. If it is tmp2 is the inverse of the original matrix. If not, the method returns null. (code can be seen in appendix A)

Although this method does not call the gaussjord() method to calculating the reduced row-echelon form because of concerning doing the row operations in two matrices, the procedure is same. The way to test if the tmp is a identity matrix is to check if the diagonal entries are all equal to one.

rank()

This method is to calculate the rank of the matrix. It firstly calls gausslim() to find the row-echelon form of the matrix. Then the number of not full rows of zeros is the rank of this matrix.

6.3 Advanced Algorithms and AI

The section is about how the advanced algorithms and AI design is implemented in this project and the problems happened when implementing them.

6.3.1 straOriginal()

This method from MatrixMultiplication class reflects the Strassen Algorithm to multiply two matrices. There is one pre-condition to use this algorithm, which is that the dimension of two matrices must in for of 2^n*2^n. The method first evenly divides the matrix into a 2*2 matrix. The entries are the sub-matrix of the original matrix. So if the method is used recursively, the matrix can be divided into n+1 levels of 2*2 submatrices, then by applying the multiplication and addition between submatrices, the result can be calculated. In the smallest submatrices, there will be just a single number. In that case, matrix multiplication becomes number addition and multiplication.

However, in the real work, 2^n*2^n is limited. So the algorithm in this method does not exactly follow the original algorithm. The pre-condition for this method is that the two matrices must have the form of 2n*2n, where n can be any integers. The method will keep on dividing the matrix and submatrices into equal dimension submatrices until it is not dividable and then use matrix additions and standard multiplications to calculate the final result.

In addition, according to Strassen Algorithm explained in the literature review, there have to be some temporary matrices to store the submatrices and their operations.

So the first step of applying this method is to check if two matrices dimension are multipliable and meet the pre-condition of using this algorithm.

Then it goes to division step. The original two matrices will be divided into four equal dimension submatrices. As it is required, we should use the recursive division in here. So the method calls itself in the content of the method.

```
        int endIndex = m1.getRows();
        int splitIndex = endIndex / 2;

        Matrix a11 = m1.getMatrix(1, splitIndex, 1, splitIndex);
        Matrix a12 = m1.getMatrix(1, splitIndex, splitIndex + 1, endIndex);
        Matrix a21 = m1.getMatrix(splitIndex + 1, endIndex, 1, splitIndex);
        Matrix a22 =
            m1.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);

        Matrix b11 = m2.getMatrix(1, splitIndex, 1, splitIndex);
```

```
        Matrix b12 = m2.getMatrix(1, splitIndex, splitIndex + 1, endIndex);
        Matrix b21 = m2.getMatrix(splitIndex + 1, endIndex, 1, splitIndex);
        Matrix b22 =
            m2.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);
```

The next step is to calculate from M_1 to M_7 (details can be seen in the Literature Review, Strassen Algorithm section). This is where the method calls itself. So when submatrices doing multiplication, they also use the straOriginal() method. Then it calculates from C_{11} to C_{22}.

```
        Matrix p1 = strassenOriginal(a11.add(a22), b11.add(b22));
        Matrix p2 = strassenOriginal(a21.add(a22), b11);
        Matrix p3 = strassenOriginal(a11, b12.subtract(b22));
        Matrix p4 = strassenOriginal(a22, b21.subtract(b11));
        Matrix p5 = strassenOriginal(a11.add(a12), b22);
        Matrix p6 = strassenOriginal(a21.subtract(a11), b11.add(b12));
        Matrix p7 = strassenOriginal(a12.subtract(a22), b21.add(b22));

        Matrix c11 = p1.add(p4).subtract(p5).add(p7);
        Matrix c12 = p3.add(p5);
        Matrix c21 = p2.add(p4);
               Matrix c22 = p1.add(p3).subtract(p2).add(p6);
```

The final step is reconstructing the matrix from C_{11} to C_{22} to get the final result.(Detail can be seen from Appendix A MatrixMultiplication class)

6.3.2 CopperWinograd()

It uses Coppersmith–Winograd algorithm to calculate two matrices multiplication. It requires same pre-conditions and goes through same steps of process. The differences are temporary objects it creates and the computations between these objects. Regarding with the code, the differences are different name and number of objects and objects operations. The structure and the basic idea is as same as that in straOriginal() . (Detail can be seen from Appendix A MatrixMultiplication class).

6.3.3 blockWiseInverse()

It uses Blockwise algorithm to calculate the inverse of the matrix. The pre-condition of this algorithm is that A_1 and delta is invertible (see Literature Review Blockwise Algorithm). This limits the algorithm. So in order to apply this algorithm to an invertible arbitrary real matrix, it requires another lemma, which is for A, which is invertible, $A^{-1} = (A^T A)^{-1} A^T$, and $A^T A$ is symmetric and positive definite which can use this algorithm. Moreover, in Blockwise algorithm, there are two

matrices, which need to calculate the inverse. So the system could call this method recursively until reach 1*1 matrix.

So the first step is to check if the matrix is invertible and calculate the $A^T A$.

Then the method decomposes the matrix $A^T A$ into four submatrices, A_{11}, A_{12}, A_{21}, A_{22}.

After that, it calculates the delta and four entries of $(A^T A)^{-1}$ by using the Blockwise lemma and reconstructing the matrix.

Finally, by multiplying the inverse of $(A^T A)^{-1}$ and A^T, the final result is found. (Details in Appendix A, MatrixInvers clasee)

6.3.4 det()

det() is the only method concerning with AI in this project. The implementation of the AI in calculating the determinant is exactly followed AI design in 5.4.

It invokes isUpperTriangle() and isLowerTriangle() first to implement the first method of AI. If the matrix is, the system calculates the product of the diagonal entries. If the matrix is not, it runs two 'for' loops to check if there is a row or column of 0s by calling isZeroRow() and isZeroCol(). If there is, return 0. Or it change the matrix into a low triangular by calling gausselim() method. (Detailed code can be seen in the Appendix A)

6.4 User Interface

The layout of each component is like the draft layout in the Design part. When the users want to create a vector or matrix, they need to input the dimension of matrix first and then choose the number type they want to enter. After clicking the 'create' button, there will be a certain number of empty TextField created in the middle panel. Those are the entries of the matrix.

Entries Layout

For different number type, the layout of the entries will be different, because DoubleNum only needs one value to initialise but the other two need two values, which are the numerator and the dominator for rational number and imaginary and real part for complex number.

When creating rationalNum entries, in order to make the users clear about which entry refers to the numerator of a rational number and which entry refers to the dominator of a rational number, the system will put a '\' between the numerator and dominator and put every single number in a bracket.

Code: ……

```
if(tp.equals("Rational Number")){
    JTextField numeratorEntries = new JTextField(3);
    JTextField denominatorEntries = new JTextField(3);
    JLabel slash=new JLabel("/");
    JLabel leftBra=new JLabel("(");
    JLabel rightBra=new JLabel(")");

    jc.add(leftBra);
    jc.add(numeratorEntries);
    jc.add(slash);
    jc.add(denominatorEntries);
    jc.add(rightBra);
}
```

'tp' is the text get from the selection of the number type comobo-box and 'jc' is the panel to show the entries.

When creating complexNum entries, in order to make the users clear about which entry refers to the real part of a complex number and which entry refers to the imaginary part of a rational number. They system will put a '+' between the real part and imaginary part, and a 'i' after the imaginary part and a bracket around each single number.

Code:

```
if(tp.equals("Complex Number")){
    JTextField realEntries = new JTextField(3);
    JTextField imaginaryEntries = new JTextField(3);
    JLabel add=new JLabel("+");
    JLabel symbol=new JLabel("*i");
    JLabel leftBra=new JLabel("(");
    JLabel rightBra=new JLabel(")");

    jc.add(leftBra);
    jc.add(realEntries);
    jc.add(add);
    jc.add(imaginaryEntries);
    jc.add(symbol);
    jc.add(rightBra);
}
```

Read entries

When the users click 'confirm' button, the system will read the input from TextField to create a vector or matrix.

Reading entries was a pain during coding the interface. There were two problems.

1. Because there are not only TextField in the entry panel, so when the system traces the components in the panel and reads the text in it, it is easy to crush.
2. Although the system can skip the non-TextField Component, it also need to be aware of which

what part of number it is reading when processing rational and complex number, which index should put it in the vector or matrix.

The approach to solve the first problem is to use a 'if' statement to check the type of the component. If the type is right, the syterm read or ignore it.

```
……
if(compType.equals("javax.swing.JTextField")){
  entry = ((JTextField)jc.getComponent(k)).getText();
……
```

The way to solve the second problem is analysing the number of component firstly, then when the system read the first part of the rational or complex number, force the system go to the second part entry. In order to put the number into the right index, the system employ index variable.

```
if(compType.equals("javax.swing.JTextField")){
                String numeratorEntry =
                ((JTextField)jc.getComponent(k)).getText();
                k=k+2;
                 String dominatorEntry =
                ((JTextField)jc.getComponent(k++)).getText();
                BigInteger numeratortmp=null;
                BigInteger dominatortmp=null;

                  try {

                   numeratortmp = new BigInteger(numeratorEntry);
                   dominatortmp = new BigInteger(dominatorEntry);
                  m.set(i+1,currentCol,new
                 RationalNum(numeratortmp, dominatortmp));
                    currentCol++;
```

When the condition of if is true, the system reads this first part of the rational number. Then it makes k=k+2 to reading the component in the 'k+2' position, where the second part of the rational number is. 'm.set' sets the rational number just read in the matrix. Then it calls currentCol++ to point to the next entry of the matrix.

6.5 Error and Exception Handling

The structure of Error and Exception Handling is quite simple.

- Define some new exception class
- Create an exception instance with an exception message in the functions, which may cause errors and throw this error.
- Catch the exception instance in the methods, which call the functions with exception instance and get the message from the instance
- Pop up an error message

Example:

The method in Matrix Class

```
public FieldElement get(int rowIndex, int colIndex)
            throws InvalidOperationException {
            ……
            throw new InvalidOperationException("Tried row index "
                        + rowIndex + ". Only row indices from 1 to "
                                + this.numOfRows + " valid. Please input the
                    matrix " +    "with right dimension");
            ……
```

The method in LAS class

```
private void matrixoperations()
            ……
            catch (InvalidOperationException e){
                ErrorMess.createErr(e.getMessage());
            }
```

In the matrixoperations() method, it will call the methods to do calculations on matrix and these method will call get() method from Matrix Class. If the get() method fails to get the matrix entry, it **throw** InvalidOperationException with the test message. Then in the matrixopertions() will **catch** this exception and call '`ErrorMess.createErr(e.getMessage())`' to create a err message.

7 Testing

7.1 Testing Objectives and Principles

The software testing is to find out how correctly, completely, securely and efficiently is developed. It also test if software achieves the requirements and how well it behaviour.

Objectives

This first feature of this project is that it is a user interactive software. It allows the users input and after operations delivers output to the users. So the first part of testing should be input testing. It involves checking if the provide the users a interface to communicate, if the system can read the user's input correctly.

The second feature is that this is a mathematical software project, which means the correctness is the most important. So the testing will focus on the correctness testing, which is test if the system delivers a right output. This part of is about the functional testing. It also checks the completeness of the functional requirement in chapter 4.

The user interface is the only way to communicate with the users for the system. So it is also necessary to have an interface testing.

Principles

This system uses three number types. So when testing functions, it should be concerned if these functions works on all three types of number. However, if in some circumstance it has successfully pass the test on one type of number, then it may not have to test for other types of number. For example, the system returns error message when adding different dimension of matrices. If it works on DoubleNum then it should also work on other two types because checkDimension() actually do not process on these numbers.

The numbers, data and functions using in this system are quite closely connect to each other. It has a quite strong function hierarchy, like the three levels of matrix functions mentioned in the Implementation. The pre-conditions of the functions in the higher level working probably is that the functions in the lower class must work well. Since there are 20 classes and tens of methods in this project, it is quite hard to test all classes and methods. So the testing shown in here will focus on testing the methods and function in high hierarchy.

7.2 Function Testing

In this part, the system will be tested by input certain numbers and check if the output is what expected. The input will be both in right or wrong form and different types of number.

7.2.1 Vector Functions

Vector Addition (Scalar)

Vector	Scalar	Expected Result	Actual Result	Status
[3 2 11]	5	[8 7 16]	[8 7 16]	Pass
Null	5	Error Message	Error Message	Pass
[1/2 2/3 4/7]	2	[3/2 4/3 21/7]	[3/2 4/3 21/7]	Pass
[2+6i 3+3i 9+7i]	3	[5+6i 6+3i 12+7i]	[5+6i 6+3i 12+7i]	Pass

Vector Addition (Vector)

Vector A	Vector B	Expected Result	Actual Result	Status
[1 2 6]	[2 4 1]	[3 6 7]	[3 6 7]	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/14]	[7/12 11/6 11/14]	[7/12 11/6 11/14]	Pass
[1 2 4]	[1/2 1/4 2/3]	[3/2 9/4 14/3]	[3/2 9/4 14/3]	Pass
[1/2 1/4 2/3]	[1 2 4]	[3/2 9/4 14/3]	[3/2 9/4 14/3]	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	[6+6i 10+3i 11+7i]	[6+6i 10+3i 11+7i]	Pass

[2+6i 3+3i 9+7i]	[4 7 2]	[6+6i 10+3i 11+7i]	[6+6i 10+3i 11+7i]	Pass
[1/2 1/4 2/3]	[1+i 2+i 4+i]	[3/2+i 9/4+i 14/3+i]	[3/2+i 9/4+i 14/3+i]	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	[3/2+i 9/4+i 14/3+i]	[3/2+i 9/4+i 14/3+i]	Pass

Vector Subtraction (Scalar)

Vector	Scalar	Expected Result	Actual Result	Status
[3 2 11]	5	[-2 –3 6]	[-2 –3 6]	Pass
Null	5	Error Message	Error Message	Pass
[5/2 8/3 20/7]	2	[1/2 2/3 6/7]	[1/2 2/3 6/7]	Pass
[2+6i 3+3i 9+7i]	3	[-1+6i 0+3i 6+7i]	[-1+6i 0+3i 6+7i]	Pass

Vector Subtraction (Vector)

Vector A	Vector B	Expected Result	Actual Result	Status
[1 2 6]	[2 4 1]	[-1 –2 5]	[-1 –2 5]	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/14]	[1/12 7/6 5/14]	[1/12 7/6 5/14]	Pass
[1 2 4]	[1/2 1/4 2/3]	[1/2 7/4 10/3]	[1/2 7/4 10/3]	Pass
[1/2 1/4 2/3]	[1 2 4]	[-1/2 -7/4 -10/3]	[-1/2 -7/4 -10/3]	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	[2+6i 4+3i -7+7i]	[2+6i 4+3i –7+7i]	Pass
[2+6i 3+3i 9+7i]	[4 7 2]	[-2+6i -4+3i 7+7i]	[-2+6i -4+3i 7+7i]	Pass
[1/2 1/4 2/3]	[1+i 2+i 4+i]	[-1/2+i -7/4+i -10/3+i]	[-1/2+i -7/4+i -10/3+i]	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	[1/2+i 7/4+i 10/3+i]	[1/2+i 7/4+i 10/3+i]	Pass

Vector Multiplication (Scalar)

Vector	Scalar	Expected Result	Actual Result	Status
[3 2 11]	5	[15 10 55]	[15 10 55]	Pass
Null	5	Error Message	Error Message	Pass
[5/2 8/3 20/7]	2	[5/1 16/3 40/7]	[5/1 16/3 40/7]	Pass
[2+6i 3+3i 9+7i]	3	[6+6i 9+3i 27+7i]	[6+6i 9+3i 27+7i]	Pass

Vector Multiplication (Dot Product)

Vector A	Vector B	Expected Result	Actual Result	Status
[1 2 6]	[2 4 1]	[2 8 6]	[2 8 6]	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass

[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/2]	[1/12 3/6 12/14]	[1/12 3/6 12/14]	Pass
[1 2 4]	[1/2 1/4 2/3]	[1/2 1/2 8/3]	[1/2 1/2 8/3]	Pass
[1/2 1/4 2/3]	[1 2 4]	[-1/2 -7/4 -10/3]	[-1/2 -7/4 -10/3]	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	[8+24i 21+21i 18+14i]	[8+24i 21+21i 18+14i]	Pass
[2+6i 3+3i 9+7i]	[4 7 2]	[8+24i 21+21i 18+14i]	[8+24i 21+21i 18+14i]	Pass
[1/2 1/4 2/3]	[1+i 2+i 4+i]	[1/2+1/2i 1/2+1/4i 8/3+2/3i]	[1/2+1/2i 1/2+1/4i 8/3+2/3i]	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	[1/2+1/2i 1/2+1/4i 8/3+2/3i]	[1/2+1/2i 1/2+1/4i 8/3+2/3i]	Pass

Vector Multiplication (Matrix)

Vector A	Matrix A	Expected Result	Actual Result	Status
[1 2 6]	[2; 4; 1]	[16]	[16]	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/2]	[43/42]	[43/42]	Pass
[1 2 4]	[1/2 1/4 2/3]	[11/3]	[11/3]	Pass
[1/2 1/4 2/3]	[1 2 4]	[11/3]	[11/3]	Pass
[4 7 2]	[1+i 2+i 4+i]	[26+13i]	[8+24i 21+21i 18+14i]	Pass
[1+i 2+i 4+i]	[4 7 2]	[26+13i]	[8+24i 21+21i 18+14i]	Pass
[1/2 1/4 2/3]	[1+i 2+i 4+i]	[11/3+17/12i]	[11/3+17/12i]	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	[11/3+17/12i]	[11/3+17/12i]	Pass

Vector Division(Scalar)

Vector	Scalar	Expected Result	Actual Result	Status
[3 2 11]	5	[0.6 0..4 2.2]	[0.6 0..4 2.2]	Pass
Null	5	Error Message	Error Message	Pass
[5/2 8/3 20/7]	2	[5/4 4/3 10/7]	[5/1 16/3 40/7]	Pass
[1+i 2+i 4+i]	2	[1/2+1/2i 1+1/2i 2+1/2i]	[1/2+1/2i 1+1/2i 2+1/2i]	Pass

Cross Product

Vector A	Vector B	Expected Result	Actual Result	Status
[3 6 1]	[2 5 1]	[1 –1 3]	[1 –1 3]	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass

[1/3 3/2 4/7]	[1/4 1/3 3/2]	[173/84 -5/14 –19/72]	[173/84 -5/14 –19/72]	Pass
[1 2 4]	[1/4 1/3 3/2]	[5/3 –1/2 –1/6]	[5/3 –1/2 –1/6]	Pass
[1/2 1/4 2/3]	[1 2 4]	[-5/3 1/2 1/6]	[-5/3 1/2 1/6]	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	[57+43i -32-16i –2-30i]	[57+43i -32-16i –2-30i]	Pass
[2+6i 3+3i 9+7i]	[4 7 2]	[-57-43i 32+16i 2+30i]	[57+43i -32-16i –2-30i]	Pass
[1/2 1/4 2/3]	[2+6i 3+3i 9+7i]	[1/4-1/4i, -19/6+1/2i, 1]	[1/4-1/4i, -19/6+1/2i, 1]	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	[-1/4+1/4i, 19/6-1/2i, 1]	[-1/4+1/4i, 19/6-1/2i, 1]	Pass

Vector Distance

Vector A	Vector B	Expected Result	Actual Result	Status
[3 6 1]	[2 5 1]	1.41421356	1.41421356	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/2]	1.49341904	1.49341904	Pass
[1 2 4]	[1/4 1/3 3/2]	3.09681736	3.09681736	Pass
[1/2 1/4 2/3]	[1 2 4]	3.09681736	3.09681736	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	Error Message	Error Message	Pass
[2+6i 3+3i 9+7i]	[4 7 2]	Error Message	Error Message	Pass
[1/2 1/4 2/3]	[2+6i 3+3i 9+7i]	Error Message	Error Message	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	Error Message	Error Message	Pass

Vector Cosine

Vector A	Vector B	Expected Result	Actual Result	Status
[3 6 1]	[2 5 1]	0.99600651	0.99600651	Pass
[3 1 7]	[2 4]	Error Message	Error Message	Pass
[4 7 2]	Null	Error Message	Error Message	Pass
[1/3 3/2 4/7]	[1/4 1/3 3/2]	$1.5*10^{18}/2.8*10^{18}$	$1.5*10^{18}/2.8*10^{18}$	Pass
[1 2 4]	[1/4 1/3 3/2]	$4.8*10^{16}/5*10^{16}$	$4.8*10^{16}/5*10^{16}$	Pass
[1/2 1/4 2/3]	[1 2 4]	$4.8*10^{16}/5*10^{16}$	$4.8*10^{16}/5*10^{16}$	Pass
[4 7 2]	[2+6i 3+3i 9+7i]	Error Message	Error Message	Pass
[2+6i 3+3i 9+7i]	[4 7 2]	Error Message	Error Message	Pass
[1/2 1/4 2/3]	[2+6i 3+3i 9+7i]	Error Message	Error Message	Pass
[1+i 2+i 4+i]	[1/2 1/4 2/3]	Error Message	Error Message	Pass

Vector Norm

Vector A	Expected Result	Actual Result	Status
[3 6 1]	6.78232998	6.78232998	Pass
[3/5 5/8 1/3]	0.92829742	0.92829742	Pass

[1+i 2+i 4+i]	Error Message	Error Message	Pass

7.2.2 Matrix Functions

In the last part of testing, when testing each function the system is also testing the number's function. Actually, testing the number's function is the real target of testing the functions working between different types of numbers. Since this function testing has passed, there is no need to test calculations between different numbers afterward.

Moreover, as it is mentioned in the Testing Principle, functions with high hierarchy have the priority to be tested. So the matrix addition, matrix multiplication (matrix) and Gaussian Elimination will not be tested in here.

Subtraction(Scalar)

Matrix A	Scalar	Expected Result	Actual Result	Status
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	3	$\begin{bmatrix} 0 & 2 & 2 \\ 6 & 5 & 0 \\ 5 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 2 \\ 6 & 5 & 0 \\ 5 & -1 & 1 \end{bmatrix}$	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	null	Error Message	Error Message	Pass

Subtraction(Matrix)

Matrix A	Matrix B	Expected Result	Actual Result	Status
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	$\begin{bmatrix} 4 & 8 & 9 \\ 2 & 1 & 8 \\ 6 & 3 & 11 \end{bmatrix}$	$\begin{bmatrix} -1 & -3 & -4 \\ 7 & 7 & -5 \\ 2 & -1 & -7 \end{bmatrix}$	$\begin{bmatrix} -1 & -3 & -4 \\ 7 & 7 & -5 \\ 2 & -1 & -7 \end{bmatrix}$	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 2 \end{bmatrix}$	Error Message	Error Message	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	Null	Error Message	Error Message	Pass

Multiplication (Scalar)

Matrix A	Scalar	Expected Result	Actual Result	Status
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	3	$\begin{bmatrix} 9 & 15 & 15 \\ 27 & 24 & 9 \\ 24 & 6 & 12 \end{bmatrix}$	$\begin{bmatrix} 9 & 15 & 15 \\ 27 & 24 & 9 \\ 24 & 6 & 12 \end{bmatrix}$	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	null	Error Message	Error Message	Pass

Matrix Multiplication (Strassen Algorithm)

Matrix A	Matrix B	Expected Result	Actual Result	Status
$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 97 & 468 & 95 & 233 \\ 255 & 658 & 241 & 367 \\ 342 & 785 & 311 & 520 \\ 102 & 319 & 188 & 236 \end{bmatrix}$	$\begin{bmatrix} 97 & 468 & 95 & 233 \\ 255 & 658 & 241 & 367 \\ 342 & 785 & 311 & 520 \\ 102 & 319 & 188 & 236 \end{bmatrix}$	Pass
$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 2 \end{bmatrix}$	Error Message	Error Message	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	Null	Error Message	Error Message	Pass

Matrix Multiplication (Coppersmith Winograd Algorithm)

Matrix A	Matrix B	Expected Result	Actual Result	Status
$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 97 & 468 & 95 & 233 \\ 255 & 658 & 241 & 367 \\ 342 & 785 & 311 & 520 \\ 102 & 319 & 188 & 236 \end{bmatrix}$	$\begin{bmatrix} 97 & 468 & 95 & 233 \\ 255 & 658 & 241 & 367 \\ 342 & 785 & 311 & 520 \\ 102 & 319 & 188 & 236 \end{bmatrix}$	Pass

$\begin{bmatrix} 2 & 4 & 11 & 4 \\ 6 & 16 & 7 & 14 \\ 3 & 32 & 3 & 15 \\ 9 & 11 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 2 \end{bmatrix}$	Error Message	Error Message	Pass
$\begin{bmatrix} 3 & 5 & 5 \\ 9 & 8 & 3 \\ 8 & 2 & 4 \end{bmatrix}$	Null	Error Message	Error Message	Pass

Gaussian-Jordan Reduction

Matrix A	Expected Result	Actual Result	Status
$\begin{bmatrix} 2 & 0 & 11 & 4 \\ 7 & 0 & 7 & 14 \\ 3 & 0 & 3 & 15 \\ 9 & 0 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	Pass
null	Error Message	Error Message	Pass

Determinant

Matrix A	Expected Result	Actual Result	Status
$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 1 & 4 \\ 0 & -2 & 5 \end{bmatrix}$	16	16	Pass
$\begin{bmatrix} 1 & -1 & 2 \end{bmatrix}$	Null	Null	
null	Error Message	Error Message	Pass

Rank

Matrix A	Expected Result	Actual Result	Status

$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 1 & 4 \\ 0 & -2 & 5 \end{bmatrix}$	3	3	Pass
null	Error Message	Error Message	Pass

Matrix Inverse

Matrix A	Expected Result	Actual Result	Status
$\begin{bmatrix} 2 & -3 \\ -4 & 5 \end{bmatrix}$	$\begin{bmatrix} -2.5 & -1.5 \\ -2 & -1 \end{bmatrix}$	$\begin{bmatrix} -2.5 & -1.5 \\ -2 & -1 \end{bmatrix}$	Pass
[2 3 4]	Null	Null	Pass
$\begin{bmatrix} 1 & 2 \\ -2 & -4 \end{bmatrix}$	Null	Null	Pass
Null	Null	Null	Pass

Matrix Inverse (Blockwise)

Matrix A	Expected Result	Actual Result	Status
$\begin{bmatrix} 2 & -3 \\ -4 & 5 \end{bmatrix}$	$\begin{bmatrix} -2.5 & -1.5 \\ -2 & -1 \end{bmatrix}$	$\begin{bmatrix} 7.2721906E9 & -1.462152112E10 \\ -1.406588E7 & 2.798348E7 \end{bmatrix}$	Fail
[2 3 4]	Null	Null	Pass
$\begin{bmatrix} 1 & 2 \\ -2 & -4 \end{bmatrix}$	Null	Null	Pass
Null	Null	Null	Pass

Transpose

Matrix A	Expected Result	Actual Result	Status
$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 1 & 4 \\ 0 & -2 & 5 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 1 & 4 \\ 0 & -2 & 5 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & 0 \\ -1 & 1 & -2 \\ 2 & 4 & 5 \end{bmatrix}$	Fail
Null	Null	Null	Pass

7.3 Input Testing and User Interface Test

Because the users have to use the interface to input numbers, so these two testing could combining together. In a addition, all function testing actually used user interface to implement the function. So this interface test will just ignore testing the mathematical functions in the interface.

Interface Testing

Action	Expected Result	Actual Result	Status
Input the size of a vector or matrix in the text area	The test area shows the input	The test area shows the input	Pass
Choose the data type from the combo-box	The combo-box shows the selected number type	The combo-box shows the selected number type	Pass
Click on 'create' button	It create some text area entries and labels in the middle part of the application window according to the size and number type input	It create some text area and labels in the middle part of the application window according to the size and number type input	Pass
Input the number in the entries	It shows input	It shows input	Pass
Click confirm button	The vector or matrix created is shown in the result panel	The vector or matrix created is shown in the result panel	Pass
Select function from the function combo-box at the bottom	The combo-box shows the selected function name	The combo-box shows the selected function name	Pass
Input number in scalar area	It show input	It shows input	Pass
Select the vector or matrix to compute the current vector or matrix with from the right bottom como-box	It shows the selected vector or matrix name	It shows the selected vector or matrix name	Pass
Click on the get result button	It doing the calculation on the vector or matrix according to the function chosen to read the scalar or selected vector or matrix from the	It doing the calculation on the vector or matrix according to the function chosen to read the scalar or selected vector or matrix from the	Pass

	combo-box and shows the result in the result panel	combo-box and shows the result in the result panel	
Click the tabbed panel	It shows the selected panel	It shows the selected panel	Pass
Click 'File' from menu bar	It shows menu item in under File	It shows menu item in under File	Pass
Click 'clear it' menu item from File	It clear the text in result panel	It clear the text in result panel	Pass
Click 'Quit	The application window closes	The application window closes	Pass

Input Testing

Action	Expected Result	Actual Result	Status
Input right format of number in the area of defining vectors and matrices size and click create button	It creates the certain number of entries.	It creates the certain number of entries.	Pass
Input wrong format of number or nothing in the area of defining vectors and matrices size and click create button	It popup a Error Message	It popup a Error Message	Pass
Input right format of number in the Scalar area and implement the function	It reads that scalar and use it in the function.	It reads that scalar and use it in the function.	Pass
Input wrong format of number or nothing in the Scalar area and implement the function	It popup a Error Message	It popup a Error Message	Pass
Input right format of number in entries of vectors of matrices and click confirm button	It reads entries and shows vector of matrices in the result panel.	It reads entries and shows vector of matrices in the result panel.	Pass
Input wrong format of number or nothing in entries of vectors of matrices and click confirm button	it pops up a error window	it pops up a error window	Pass

8 Conclusion

In this part, a general evaluation and critique about this project should be made. It will also talk about how this project can be developed in the next stage.

8.1 Evaluation and Critiques

8.1.1 Achievement

Recalling the objectives of this project in the chapter one, the work so far has finished most of them, especially the primary objectives.

- Capable of process different types of number, such as decimal number, rational number or complex number
- Capable of doing some general linear algebra arithmetic, which is actually about vector and matrix operations
- Capable of implementing some advanced math algorithms with low complexity to do matrix calculations

Firstly, the system can work on three types of number, which are Decimal Number, Rational Number in form of a/b and Complex Number. Each type of numbers has its own functions and they can also work together.

Secondly, the system can do 30 different calculations. It covers most normal linear algebra arithmetic in vector and matrix area. Functions required in Requirement all have been completed.

There are three advanced algorithm implemented. The two for multiplication passed the testing. The one for matrix inverse does not work properly. The code implementation exactly follows the theory in the Literature Review but the output is wrong. So the problem could be the misunderstanding of that algorithm in the early stage or the algorithm can only work on some certain situations.

Secondary Objectives:

- Artificial Intelligence
- Friendly user interface
- Robust system with error handling functions
- Independent code structure working on different fields, which can make the system easy to improve and develop later

Artificial Intelligence has applied in calculating the determinant of the matrix.

The user interface of the system is simple to follow. The users do not need to read the user manual to start using the system.

The error handling functions in this system can handle most common errors and exceptions caused by the users' mistake.

Code independency is quite strong in this system. For so classes, they can be used for other system without any change. They should be quite easy to maintain and update.

8.1.2 Critiques

There are still some problems in this project.

1. The Blockwise algorithm does not work properly in this project.
2. Artificial Intelligence in this project is only designed on calculating the matrix determinant. There are a lot of aspects of AI in linear algebra have not been concerned.
3. Although the system achieved the interface simplicity, it limits the function of the system. Moreover, interface layout still has problem. When input a large matrix, the entries are not structurally sorted out in the application window.
4. It should have a array to store vectors or matrices. Then the system work on more than two vectors or matrices.
5. Error handling is too simple. There are only two classes created. For most errors, they also use InvalidOperationException(). This categorising method is too general.
6. The code coupling is a problem. To implement one function, it calls several methods. This will badly affect maintainability.
7. There are too many duplicated codes in the interface LAS class and there is only one class to support the user interface, which is not ideal.
8. System testing is not refined enough. There are still a lot of aspects need to be tested.

8.2 Further Development

8.2.1 Numbers

- Developing the system to work on more types of numbers. Fox example, we can create a class to store the root numbers.
- Adding More number arithmetic. We can add more arithmetic functions for each type of number, especially the complex number.

8.2.2 Linear Algebra Functions

- Adding more linear algebra functions, such as calculating the Eigenvalues and Eigenvectors

8.2.3 User Interface

- Improving the user interface layout
- Adding command line into the interface to give user more freedom
- Adding more non-mathematical functions, like loading external file with matrix of vector, recording command history, saving the result and so on.

8.2.4 Advanced Algorithm

- Fixing the problem of Blockwise Inverse
- Developing more algorithms. For example, LU decomposition which decompose the matrix into upper and lower triangle, which can be used to solve the linear equations and matrix inverse. The QR decomposition, which decompose a matrix into an orthogonal and a triangular matrix. The QR decomposition is also the basis for the QR algorithm. QR algorithm can be used to calculate the eigenvalues.

8.2.5 Artificial Intelligence

- Developing AI on calculating the determinant of a matrix by using the properties of determinant. For example, If A has two equal rows of columns, then det A = 0
- Developing AI on other functions. Fox example, when doing calculation on more then two matrix, using the algebraic properties may make the calculation easier.

9 Bibliography

(1) Courant, R. and Robbins, H. , 1996. What is Mathematics?. Oxford: Oxford University Press;

(2) Williams, G. 2005. Linear Algebra with Application . 5th ed. London : Jones and Bartlett;

(3) Lay, David C. 1997. Linear Algebra and its Applications .2nd ed Harlow: Addison-Wsley

(4) Grossman, Stanley I. 1994. Elementary Linear Algebra. 5th en. London: Sauders College

(5) Kolman, Bernard. Hill, David R. 2005. Introductory Linear Algebra: an applied first course. 8th ed. N.J.: Pearson/Prentice Hall

(6) Corman, Thomas H. 2001. Introduction to algorithms. 2nd ed. London: MIT

(7) WIKIPEDIA. Coppersmith–Winograd algorithm. In: Wikipedia: the free encyclopedia[online].St Petersburg, Florida: Wikimedia Foundation. Available from: http://en.wikipedia.org/wiki/Coppersmith-Winograd_algorithm [Accessed 25.04.07]

(8) Richard, D. 2002. Advanced Mathematical Methods with Maple. 1st ed. Cambridge: Cambridge Univsersity Press

(9) Wilson, Howard B. 2002. Advanced Mathematics and Mechanics applications Using matlab. 3rd ed. London: CRC

(10) Pratap, R. 2006. Getting Started with Matlab. Oxford: Oxford University Press

(11) Sommerville, I. 2004. Software Engineering. 7th ed. London: Addison-Wesley.

(12) WIKIPEDIA. Determinant. In: Wikipedia: the free encyclopedia[online].St Petersburg, Florida: Wikimedia Foundation. Available from: http://en.wikipedia.org/wiki/Determinant [Accessed 27.04.07]

(13) Bloch, J. 2001. Effective Java : programming language guide. London: Addison-Wesley.

10 Appendix A

Matrix Class

```
package Backend;


public class Matrix{

public FieldElement det(){

        if (!this.isSquare()){
            return null;
        }

        if (this.isLowerTriangular() || this.isUpperTriangular()){
            FieldElement det = new DoubleNum(1);
            for (int row = 1; row <= this.getRows(); row++) {
                det = det.multiply(this.get(row, row));
            }
            return det;
        }
        for (int i=1; i<=this.getCols();i++){
            if(this.isZeroCol(i)){
                return new DoubleNum(0);
            }

        }
        for (int i=1; i<=this.getRows();i++){
            if(this.isZeroRow(i)){
                return new DoubleNum(0);
            }

        }
        Matrix tmp = this.gausselim();
        FieldElement determinant = (tmp.get(1, 1).one());

        for (int row = 1; row <= tmp.getRows(); row++) {
            determinant = determinant.multiply(tmp.get(row, row));
        }

        return determinant;
```

```
    }
/**
     * Returns a matrix that is this Matrix with the Gauss-Jordan algorithm
     * executed on. In other words: It returns the reduced row echelon form
of
     * this Matrix.
     *
     * @return matrix in reduced row-echelon form
     */

    public Matrix gaussjord() {
        Matrix tmp = this.copy();

        int minOfRowsCols = Math.min(tmp.getRows(), tmp.getCols());
        int colCounter = 0;

        int row = 0;
        while (row < minOfRowsCols && colCounter < tmp.getCols()) {
            row++;
            colCounter++;
            FieldElement diagonalEntry = tmp.get(row, colCounter);

            if (diagonalEntry.isZero()) {
                // search for non zero entry
                boolean found = false;
                for (int candidate = row + 1; candidate <= tmp.getRows();
candidate++) {
                    if (!tmp.get(candidate, colCounter).isZero()) {
                        tmp.swapRows(row, candidate);
                        found = true;
                        break;
                    }
                }

                if (!found) {
                    if (colCounter == tmp.getCols()) {
                        return tmp;
                    }

                    row--;
                    continue;
                } else {
                    diagonalEntry = tmp.get(row, colCounter);
```

```
                }
            }

            for (int j = colCounter; j <= tmp.getCols(); j++) {
                FieldElement oldEntry = tmp.get(row, j);
                tmp.set(row, j, oldEntry.divide(diagonalEntry));
            }

            for (int j = 1; j <= tmp.getRows(); j++) {

                FieldElement factor = tmp.get(j, colCounter);
                if (row == j || factor.isZero()) {
                    continue;
                }

                for (int k = colCounter; k <= tmp.getCols(); k++) {
                    FieldElement oldEntry = tmp.get(j, k);
                    tmp.set(j, k, oldEntry.subtract(tmp.get(row,
k).multiply(
                            factor)));
                }
            }
        }
        return tmp;
    }

    /**
     * Returns a matrix that is this Matrix with Gauss-elimination executed
on.
     * In other words: It returns a row echelon form of this Matrix.
     *
     * @return matrix in row-echelon form
     */

    public Matrix gausselim() {
        Matrix tmp = this.copy();

        int minOfRowsCols = Math.min(tmp.getRows(), tmp.getCols());
        int colCounter = 0;

        int row = 0;
        while (row < minOfRowsCols && colCounter < tmp.getCols()) {
            row++;
            colCounter++;
```

```
            FieldElement diagonalEntry = tmp.get(row, colCounter);

            if (diagonalEntry.isZero()) {
                // search for non zero entry
                boolean found = false;
                for (int candidate = row + 1; candidate <= tmp.getRows();
candidate++) {
                    if (!tmp.get(candidate, colCounter).isZero()) {
                        tmp.swapRows(row, candidate);
                        rowswap.negate();
                        found = true;
                        break;
                    }
                }

                if (!found) {
                    if (colCounter == tmp.getCols()) {
                        return tmp;
                    }

                    row--;
                    continue;
                } else {
                    diagonalEntry = tmp.get(row, colCounter);
                }
            }

            for (int j = row; j <= tmp.getRows(); j++) {

                FieldElement factor = tmp.get(j, colCounter).divide(
                        diagonalEntry);


                if (row == j || factor.isZero()) {
                    continue;
                }

                for (int k = colCounter; k <= tmp.getCols(); k++) {
                    FieldElement oldEntry = tmp.get(j, k);
                    tmp.set(j, k,
oldEntry.subtract(factor.multiply(tmp.get(
                            row, k))));

                }
```

```
                //rowMul=rowMul.multiply(11);
            }
        }
        return tmp;
    }


}
```

MatrixInverse() Class

```
package Backend;


public class MatrixInverse{
    //private static int FROBENIUS_POINT = 48;

    /**
     *Check if the matrix is a square matrix
     */
    public static boolean checkSingular(Matrix m)
    {
        boolean check=true;
        if (m.getRows() != m.getCols()) {
            check=false;
        }
        else{
            Matrix tmp = m.copy();

            int minOfRowsCols = Math.min(tmp.getRows(), tmp.getCols());
            int colCounter = 0;
            int row = 0;

            while (row <= minOfRowsCols){

                row++;

                while (colCounter <= tmp.getCols()){
                    colCounter++;
                    if (colCounter==tmp.getCols()&&tmp.get(row,
colCounter).isZero()){
                        System.out.println("this is not a invertible
matrix");
```

```
                    check = false;
                    return false;
                }
                if (!tmp.get(row, colCounter).isZero()){
                    break;
                }
            }
        }
    }
    return check;
}
public static Matrix simple(Matrix m)
{
    if (m.getRows() != m.getCols()) {
        return null;
    }

    Matrix tmp = m.copy();

    FieldElement zero = tmp.get(1, 1).zero();
    FieldElement one = zero.one();

    FieldElement[][] entries2 = new FieldElement[tmp.getRows()][tmp
            .getCols()];

    for (int i = 0; i < tmp.getRows(); i++) {
        for (int j = 0; j < tmp.getCols(); j++) {
            if (i == j) {
                entries2[i][j] = one;
            } else {
                entries2[i][j] = zero;
            }
        }
    }

    Matrix tmp2 = new Matrix(entries2);

    int minOfRowsCols = Math.min(tmp.getRows(), tmp.getCols());
    int colCounter = 0;

    int row = 0;
    while (row < minOfRowsCols && colCounter < tmp.getCols()) {
        row++;
        colCounter++;
```

```
            FieldElement diagonalEntry = tmp.get(row, colCounter);

            if (diagonalEntry.isZero()) {
                // search for non zero entry
                boolean found = false;
                for (int candidate = row + 1; candidate <= tmp.getRows();
candidate++) {
                    if (!tmp.get(candidate, colCounter).isZero()) {
                        tmp.swapRows(row, candidate);
                        tmp2.swapRows(row, candidate);
                        found = true;
                        break;
                    }
                }

                if (!found) {
                    if (colCounter == tmp.getCols()) {
                        return null; // Because there is no inverse of m.
                    }

                    row--;
                    continue;
                } else {
                    diagonalEntry = tmp.get(row, colCounter);

                }
            }

            for (int j = 1; j <= tmp.getCols(); j++) {
                FieldElement oldEntry = tmp.get(row, j);
                FieldElement oldEntry2 = tmp2.get(row, j);
                tmp.set(row, j, oldEntry.divide(diagonalEntry));
                tmp2.set(row, j, oldEntry2.divide(diagonalEntry));
            }

            for (int j = 1; j <= tmp.getRows(); j++) {

                FieldElement factor = tmp.get(j, colCounter);
                if (row == j || factor.isZero()) {
                    continue;
                }

                for (int k = 1; k <= tmp.getCols(); k++) {
                    FieldElement oldEntry = tmp.get(j, k);
```

```
                FieldElement oldEntry2 = tmp2.get(j, k);
                tmp.set(j, k, oldEntry.subtract(tmp.get(row,
k).multiply(
                        factor)));
                tmp2.set(j, k, oldEntry2.subtract(tmp2.get(row, k)
                        .multiply(factor)));
            }
        }
    }

    boolean is=false;
    for (int i=1,j=1;i<=tmp.getRows()&&j<=tmp.getCols();i++,j++){
        if(tmp.get(i, j).isOne()){
            is=true;
        }
        else{
            break;
        }
    }
    if(is){
        return tmp2;
    }
    return null;


}
public static Matrix blockWiseInverse(Matrix m){
//     if (!checkSingular(m)){
//         return null;
//     }
    if (m.getRows()==1){
        return m;
    }
//
    if (m.getRows() != m.getCols()) {
        return null;
    }

    Matrix mTranspose = m.transpose();
    //tmp1 is used to calculate the product of the inverse of A.tran
multiplied by A
    Matrix tmp1 = mTranspose.multiply(m);
    //Matrix tmp1=m.gausselim();
    int endIndex = tmp1.getRows();
```

```
        int splitIndex = endIndex / 2;

        Matrix a11 = tmp1.getMatrix(1, splitIndex, 1, splitIndex);
        Matrix a12 = tmp1.getMatrix(1, splitIndex, splitIndex + 1,
endIndex);
        Matrix a21 = tmp1.getMatrix(splitIndex + 1, endIndex, 1,
splitIndex);
        Matrix a22 = tmp1.getMatrix(splitIndex + 1, endIndex, splitIndex
+ 1, endIndex);

        //tmp2 is the temporary matrix used to calculate the delta

        System.out.println(a11.toString());
        System.out.println(a12.toString());
        System.out.println(a21.toString());
        System.out.println(a22.toString());

        Matrix a11Inverse = blockWiseInverse(a11);
        Matrix delta = a21.multiply(a11Inverse);
        delta = delta.multiply(a12);
        delta = a22.subtract(delta);
        Matrix deltaInverse = blockWiseInverse(delta);

        System.out.println("delta "+delta+deltaInverse+a11Inverse+"\n");
        //the code below is to calculate the inverse of tmp1
        Matrix c11 = a11Inverse.multiply(a12);
        System.out.println(c11.toString());
        c11=c11.multiply(deltaInverse);
        System.out.println(c11.toString());
        c11 = c11.multiply(a21);
        System.out.println(c11.toString());
        c11 = c11.multiply(a11Inverse);
        System.out.println(c11.toString());
        c11=a11Inverse.add(c11);
        System.out.println(c11.toString());
        Matrix c12 =
a11Inverse.multiply(a12).multiply(deltaInverse).multiply(new
DoubleNum(-1));

        Matrix c21 =
deltaInverse.multiply(a21).multiply(a11Inverse).multiply(new
DoubleNum(-1));
        Matrix c22 = deltaInverse;
```

```
        FieldElement[][] c11Entries = c11.getEntries();
        FieldElement[][] c12Entries = c12.getEntries();
        FieldElement[][] c21Entries = c21.getEntries();
        FieldElement[][] c22Entries = c22.getEntries();

        FieldElement[][] cEntries = new
FieldElement[m.getRows()][m.getCols()];

        for (int i = 0; i < c11.getRows(); i++) {
            for (int j = 0; j < c11.getCols(); j++) {
                cEntries[i][j] = c11Entries[i][j];
            }
        }

        for (int i = 0; i < c12.getRows(); i++) {
            for (int j = 0; j < c12.getCols(); j++) {
                cEntries[i][j + splitIndex] = c12Entries[i][j];
            }
        }

        for (int i = 0; i < c21.getRows(); i++) {
            for (int j = 0; j < c21.getCols(); j++) {
                cEntries[i + splitIndex][j] = c21Entries[i][j];
            }
        }

        for (int i = 0; i < c22.getRows(); i++) {
            for (int j = 0; j < c22.getCols(); j++) {
                cEntries[i + splitIndex][j + splitIndex] =
c22Entries[i][j];
            }
        }

        Matrix mInverse = new Matrix(cEntries);
        System.out.println(mInverse.toString());
        System.out.println(mTranspose.toString());
        mInverse = mInverse.multiply(mTranspose);

        return mInverse;

    }

}
```

MatrixMultiplication Class

```
package Backend;



/**
 * This includes some different methods of multiplying two matrices.
 * The standard method of the Matrix class is the school-method.
 * All the other stuff is to be considered experimental stuff.
 *
 */

public class MatrixMultiplication {

    //private static int STRASSEN_ORIGINAL_TRUNCATION_POINT = 48;
    //private static int STRASSEN_WINOGRAD_TRUNCATION_POINT = 48;

    private static void checkDimensions(Matrix m1, Matrix m2)
        throws InvalidOperationException {
        if (m1.getCols() != m2.getRows()) {
            throw new InvalidOperationException(
                "Tried to multiply a matrix with "
                    + m1.getCols()
                    + " columns and a matrix with "
                    + m2.getRows()
                    + " rows");
        }
    }
    private static boolean checkSize(Matrix m1, Matrix m2) {

        if (m1.getCols()%2!=0||m1.getRows()!=m1.getCols()){
            return false;
        }
        return true;

    }

    /**
     * Uses the standard method for multiplication of Matrix-objects.
     * Asymptotic runtime: 0(n^3)
     * @param m1
     * @param m2
     * @return m1 multiplied by m2
```

```
     * @throws InvalidOperationException
     */
    public static Matrix simple(Matrix m1, Matrix m2)
        throws InvalidOperationException {
        checkDimensions(m1, m2);

        int resultRows = m1.getRows();
        int resultCols = m2.getCols();

        Matrix resultMatrix = new Matrix(resultRows, resultCols);

        for (int i = 1; i <= resultRows; i++) {
            for (int j = 1; j <= resultCols; j++) {
                resultMatrix.set(i, j,
m1.getRow(i).multiply(m2.getCol(j)));
            }
        }
        return resultMatrix;
    }


    /**
     * The original Strassen-Algorithm for matrix-multiplication.
     * @param m1
     * @param m2
     * @return m1 multiplied by m2
     * @throws InvalidOperationException
     */
    public static Matrix strassenOriginal(Matrix m1, Matrix m2) {

        checkDimensions(m1, m2);
        if(checkSize(m1,m2)){
            int endIndex = m1.getRows();
            int splitIndex = endIndex / 2;
            System.out.println(m1);

            Matrix a11 = m1.getMatrix(1, splitIndex, 1, splitIndex);
            Matrix a12 = m1.getMatrix(1, splitIndex, splitIndex + 1,
endIndex);
            Matrix a21 = m1.getMatrix(splitIndex + 1, endIndex, 1,
splitIndex);
            Matrix a22 =
                m1.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);
```

```
            Matrix b11 = m2.getMatrix(1, splitIndex, 1, splitIndex);
            Matrix b12 = m2.getMatrix(1, splitIndex, splitIndex + 1,
endIndex);
            Matrix b21 = m2.getMatrix(splitIndex + 1, endIndex, 1,
splitIndex);
            Matrix b22 =
                m2.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);
            System.out.println(a11.toString());

            Matrix p1 = strassenOriginal(a11.add(a22), b11.add(b22));
            Matrix p2 = strassenOriginal(a21.add(a22), b11);
            Matrix p3 = strassenOriginal(a11, b12.subtract(b22));
            Matrix p4 = strassenOriginal(a22, b21.subtract(b11));
            Matrix p5 = strassenOriginal(a11.add(a12), b22);
            Matrix p6 = strassenOriginal(a21.subtract(a11), b11.add(b12));
            Matrix p7 = strassenOriginal(a12.subtract(a22), b21.add(b22));
            System.out.println(p1+"\n");

            Matrix c11 = p1.add(p4).subtract(p5).add(p7);
            Matrix c12 = p3.add(p5);
            Matrix c21 = p2.add(p4);
            Matrix c22 = p1.add(p3).subtract(p2).add(p6);

            FieldElement[][] c11Entries = c11.getEntries();
            FieldElement[][] c12Entries = c12.getEntries();
            FieldElement[][] c21Entries = c21.getEntries();
            FieldElement[][] c22Entries = c22.getEntries();

            FieldElement[][] cEntries =
                new FieldElement[m1.getRows()][m2.getCols()];

            for (int i = 0; i < c11.getRows(); i++) {
                for (int j = 0; j < c11.getCols(); j++) {
                    cEntries[i][j] = c11Entries[i][j];
                }
                System.out.println(cEntries+"cEntries\n");
            }

            for (int i = 0; i < c12.getRows(); i++) {
                //int offset = splitIndex;
                for (int j = 0; j < c12.getCols(); j++) {
                    cEntries[i][j + splitIndex] = c12Entries[i][j];
```

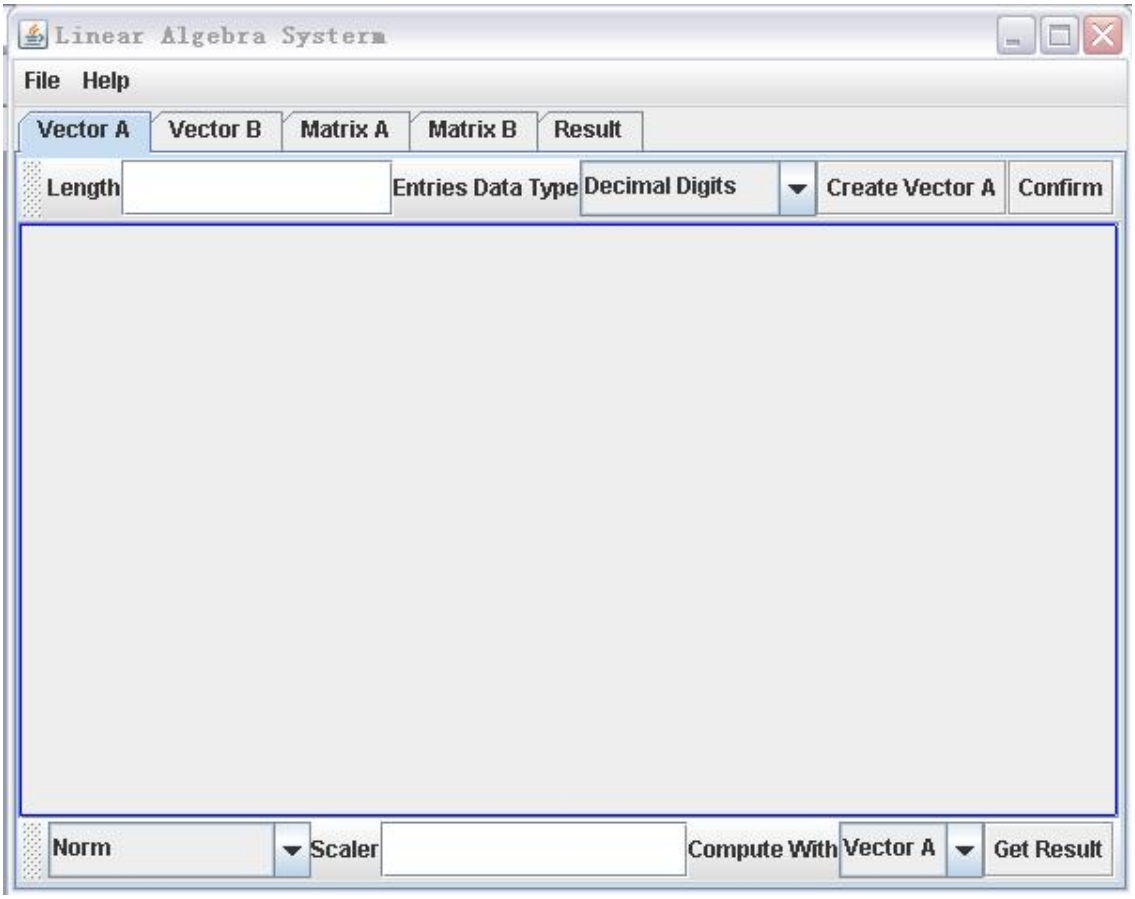
```
            }
        }

        for (int i = 0; i < c21.getRows(); i++) {
            //int offset = splitIndex;
            for (int j = 0; j < c21.getCols(); j++) {
                cEntries[i + splitIndex][j] = c21Entries[i][j];
            }
        }

        for (int i = 0; i < c22.getRows(); i++) {
            //int offset = splitIndex;
            for (int j = 0; j < c22.getCols(); j++) {
                cEntries[i + splitIndex][j + splitIndex] =
c22Entries[i][j];
            }
        }

        return new Matrix(cEntries);
    }
    return simple(m1, m2);


}

/**
 * The Algorithm of Coppersmith–Winograd for matrix-multiplication.
 * @param m1
 * @param m2
 * @return m1 multiplied by m2
 * @throws InvalidOperationException
 */

public static Matrix coppersmithWinograd(Matrix m1, Matrix m2) {
    checkDimensions(m1, m2);
    if(checkSize(m1,m2)){
        int endIndex = m1.getRows();
        int splitIndex = endIndex / 2;

        Matrix a11 = m1.getMatrix(1, splitIndex, 1, splitIndex);
        Matrix a12 = m1.getMatrix(1, splitIndex, splitIndex + 1,
endIndex);
        Matrix a21 = m1.getMatrix(splitIndex + 1, endIndex, 1,
splitIndex);
```

```
            Matrix a22 =
                m1.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);

            Matrix b11 = m2.getMatrix(1, splitIndex, 1, splitIndex);
            Matrix b12 = m2.getMatrix(1, splitIndex, splitIndex + 1,
endIndex);
            Matrix b21 = m2.getMatrix(splitIndex + 1, endIndex, 1,
splitIndex);
            Matrix b22 =
                m2.getMatrix(splitIndex + 1, endIndex, splitIndex + 1,
endIndex);

            Matrix s1 = a21.add(a22);
            Matrix s2 = s1.subtract(a11);
            Matrix s3 = a11.subtract(a21);
            Matrix s4 = a12.subtract(s2);

            Matrix t1 = b12.subtract(b11);
            Matrix t2 = b22.subtract(t1);
            Matrix t3 = b22.subtract(b12);
            Matrix t4 = b21.subtract(t2);

            Matrix p1 = coppersmithWinograd(a11, b11);
            Matrix p2 = coppersmithWinograd(a12, b21);
            Matrix p3 = coppersmithWinograd(s1, t1);
            Matrix p4 = coppersmithWinograd(s2, t2);
            Matrix p5 = coppersmithWinograd(s3, t3);
            Matrix p6 = coppersmithWinograd(s4, b22);
            Matrix p7 = coppersmithWinograd(a22, t4);

            Matrix u1 = p1.add(p2);
            Matrix u2 = p1.add(p4);
            Matrix u3 = u2.add(p5);
            Matrix u4 = u3.add(p7);
            Matrix u5 = u3.add(p3);
            Matrix u6 = u2.add(p3);
            Matrix u7 = u6.add(p6);

            FieldElement[][] c11Entries = u1.getEntries();
            FieldElement[][] c12Entries = u7.getEntries();
            FieldElement[][] c21Entries = u4.getEntries();
            FieldElement[][] c22Entries = u5.getEntries();
```

```
            FieldElement[][] cEntries =
                new FieldElement[m1.getRows()][m2.getCols()];

            for (int i = 0; i < u1.getRows(); i++) {
                for (int j = 0; j < u1.getCols(); j++) {
                    cEntries[i][j] = c11Entries[i][j];
                }
            }

            for (int i = 0; i < u7.getRows(); i++) {
                //int offset = splitIndex;
                for (int j = 0; j < u7.getCols(); j++) {
                    cEntries[i][j + splitIndex] = c12Entries[i][j];
                }
            }

            for (int i = 0; i < u4.getRows(); i++) {
                //int offset = splitIndex;
                for (int j = 0; j < u4.getCols(); j++) {
                    cEntries[i + splitIndex][j] = c21Entries[i][j];
                }
            }

            for (int i = 0; i < u5.getRows(); i++) {
                //int offset = splitIndex;
                for (int j = 0; j < u5.getCols(); j++) {
                    cEntries[i + splitIndex][j + splitIndex] =
c22Entries[i][j];
                }
            }

            return new Matrix(cEntries);
        }
        return simple(m1,m2);



    }

}
```

11 Appendix B User Manual

This software is quite easy to use. There are five tabbed panel in the application window, which are Vector A, Vector B, Matrix A and Matrix B and Result. Vector A, Vector B, Matrix A and Matrix B panel are used to create vectors and matrices. The Result panel is for showing the calculation result.



User cases

Case 1 Creating a Vector

1. Choose the vector panel from Vector A and Vector B.
2. Input the length of the vector you want to create in the text area on the right of 'Length'. The system only accept the positive integer
3. Choose the type of number you want to create from the 'Entries Data Type' combo-box
4. Click 'Create...' button
5. Input the number in the entries in the middle of the application window shown right after clicking the 'Create…' button. The system only accept decimal number for entries.
6. After finishing the input, click the 'Confirm' button. The system will create this Vector. Please make sure every entry has input number. Then the result panel will show the vector just been created.

Case 2 Creating a Matrix

1. Choose the Matrix panel from Matrix A and Matrix B.
2. Input the length of row and column you want to create in the ‘row’ and ‘column’ text area. The system only accept the positive integer
3. Choose the type of number you want to create from the ‘Entries Data Type’ combo-box
4. Click ‘Create...’ button
5. Input the number in the entries in the middle of the application window shown right after clicking the ‘Create…’ button. The system only accept decimal number for entries.
6. After finishing the input, click the ‘Confirm’ button. The system will create this Matrix. Please make sure every entry has input number.

Case 3 Implement Calculation on a VectorA(B)

1. Choose Vector A(B) from the tabbed panel
2. Make sure the Vector A(B) has been created. If not go to the case 1 to create it.
3. Choose the function from the ‘function’ combo-box in the bottom left of application window
4. If the function needs to work with a scalar, then input the scalar in the scalar text area. this text area only accept the decimal number
5. If the function needs to work with another vector or matrix, choose the vector name or matrix name from the combo-box on the right of ‘Compute With’ label. Before doing that, please make if the vector or matrix chosen is not empty.
6. Click ‘Get Result’ button.
7. then the result panel will show the result

Case 4 Implement Calculation on a Matrix A(B)

1. Choose Matrix A(B) from the tabbed panel
2. Make sure the Matrix A(B) has been created. If not go to the case 1 to create it.
3. Choose the function from the ‘function’ combo-box in the bottom left of application window
4. If the function needs to work with a scalar, then input the scalar in the scalar text area. this text area only accept the decimal number
5. If the function needs to work with another vector or matrix, choose the vector or matrix name from the combo-box on the right of ‘Compute With’ label. Before doing that, please make if the vector or matrix chosen is not empty.
6. Click ‘Get Result’ button.
7. then the result panel will show the result

Case 5 Clear the Result Panel

1. Click the ‘File’ from the menubar
2. Click ‘Clear Result Panel’

Case 6 Close the Application

1. Click the ‘File’ from the menubar
2. Click ‘Close’