COMPUTER SICIENCE, UNIVERSITY OF BATH

Computer linear algebra system

RUOSHI HE
Bachelor of Science in Computer Science with Honours
The University of Bath
04/2009

Computer linear algebra

Submitted by: Ruoshi He

Abstract

The main aim of this project is designing and implementing a Computer Linear Algebra System. The Computer Linear Algebra System is a Java based program. The system is able to do the basic calculations of Linear Algebra , such as the cross product of vectors, the multiplication of matrices, solve the system of equations and so on. The main advantage of the program is simple to use, and the storage of the program is small.

CONTNET

1. Introduction

Linear algebra is an important part of modern mathematics. It is widely used in the different part of sciences, such as economics, computer graphics. The general use of linear algebra is solving the problems by the matrix calculations. For instance, the linear algebra is using to solve the unknown values of linear functions. In this case if there are only few equations, we could apply the algorithm by writing on the paper. But imagine there are hundreds of linear equations, we could not calculate them by hands. That is why we are using computer to sort out this problem. Computer is a tool which uses to help people with the complex works, it could solve the problems more accurately and efficiently. The computer linear algebra system is the method to help people with these painful problems.

Nowadays there is lots of mathematics software, for example the Maple, the Matlab and so on. These software are powerful, not only they can solve the linear algebra problems, but also they could sort out more mathematic problems. The disadvantages of these systems are:

- The software is too large, it always uses almost 1 gigabits from the hard disk.
- When it is running, it usually spends amount memories.
- The software is not easy to use, the interface of it is not easy to follow. The users who are not computer scientists will find some difficulty of use.
- If the user only wishes to use part of the software, such as only doing the simple calculations, it is wasteful to use this software.
- The problem of the compatibility of the software is not perfect.

The linear algebra system is still necessary in this case. This project is going to narrow down the scope to only focus on the linear algebra system.

1.1 The aim

The main aim of this project is designing and implementing a computer linear algebra system. The system should be able to solve the basic operations of linear algebra, for instance solving the linear equations, inversing the matrix, calculating the determinant of matrix… The system should have a better compatibility than the existing software.

1.2 Objectives

Main Objectives

- The system should be able to process different types of data format, such as integer, float and rational number.
- The system should contain the definition algorithms of linear algebra.
- The system might have advanced algorithms for the matrices multiplications, such as Strassen’s.
- The system should get the correct arithmetic solutions.

Other objectives

- The easy and nice user interface.
- The input data could be a format of file.
- User should be able to use the system after read the user manual.

2. Literature Review

2.1. Introduction

In the first place of doing the project, the background knowledge of linear algebra is the most significant object. The theories of linear algebra are the foundation of the linear algebra system. By researching the theories, I need to understand them and apply them in the computer linear algebra system. The research is covered concerning with the current system of linear algebra or large mathematical software.

In the theory section, it is talking about the basic operations of linear algebra, the vectors and the matrix. From the literature review, both the basic operation algorithms and the advanced algorithms will be discussed.

There are already some mathematical software has released, and there are some software for linear algebra has released as well. It is possible to research them and get a general understanding of how they work. It will help with the idea of designing this project. I do not need to waste time to think about the knowledge which has already established. And these systems will bring some hints for the future implementing the system.

In this review, as mentioned before there will be two parts: background knowledge of linear algebra system and current system of solving the linear algebra system. The background knowledge will use the most of the space of the Review

2.2 Linear algebra

2.2.1 Matrices and System of Linear equations

2.2.1.1 Linear equations

A **linear equation** is an equation like x+y=2. If there are two linear equations in the system:

X+y=2, 3x+5y=9

The pair of values of x and y satisfy the equations called the **solution**.

The linear equation in n variables x_1, x_2,..., x_n has the form

$$a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n = b$$

$a_1, a_2, \ldots, a_n$ and b are called **coefficient**, and they are numbers.

2.2.1.2 Matrix

Matrix and Elements

A **matrix** is a rectangular array of numbers, the numbers in the matrix are called the **elements** of the matrix. Matrix is used to describe system of linear equations.

Matrix is denoted by capital letter, for example:

$$A = \begin{bmatrix} 2 & 5 & 1 \\ -8 & 3 & 11 \end{bmatrix}$$

Rows and Columns

Rows are labelled from the top of the matrix; **columns** are from the left of the matrix. In the example above, row1 is$\begin{bmatrix} 2 & 5 & 1 \end{bmatrix}$, column1 is$\begin{bmatrix} 2 \\ -8 \end{bmatrix}$.

Submatrix

Submatrix is an array obtained by deleting certain rows and columns of the original matrix. For example the submatrix of A could be:

$P = \begin{bmatrix} 2 & 1 \\ -8 & 11 \end{bmatrix}$, $Q = \begin{bmatrix} -8 & 3 & 11 \end{bmatrix}$

Size of matrix

The **size** of a matrix is described as a x b, and a is the number of rows, b is the number of column. And the matrix A is a 2x3 matrix.

Types of matrices

There are different **types** of matrix, when a=b the matrix is called **square matrix**; a matrix consisting one row is called **row matrix**, consisting one column is called **column matrix**.

Location of element in the matrix

The **location** of an element is given by the row number and column number which the element lies in. For example, in matrix A, number 3's location is (2, 2), which means it lies in row 2 and column 2.

Identity matrix

The **identity matrix** is a square matrix with 1s in the diagonal, and the rest elements are all zeros. And for n x n matrix, it is denoted asI_n. For example: $I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Triangular matrix

The **upper triangular matrix** is a square matrix whose all elements under the main diagonal are zero; the **lower triangular matrix** is a square matrix whose all elements above the main diagonal are zero. For example:

$$U = \begin{bmatrix} 6 & 20 & 10 \\ 0 & 1 & 9 \\ 0 & 0 & 15 \end{bmatrix} \qquad L = \begin{bmatrix} 6 & 0 & 0 \\ 12 & 9 & 0 \\ 7 & 3 & 4 \end{bmatrix}$$

$$upper\ triangular\ matrix \qquad lower\ triangular\ matrix$$

2.2.2 Solving linear equations by matrix

2.2.2.1 Elementary transformation and elementary row operations.

The linear equations could be described by matrices. The coefficients of the variables from a matrix called the **matrix of coefficients**; the coefficients with the constant term from a matrix called the **augmented matrix** of the system. For example:

$$\begin{aligned} x + y - 5z &= 3 \\ x + 3y - z &= 0 \\ x - y + z &= 2 \end{aligned} \qquad \underset{matrix\ of\ coefficients}{\begin{bmatrix} 1 & 1 & -5 \\ 1 & 3 & -1 \\ 1 & 1 & 1 \end{bmatrix}} \qquad \underset{augmented\ matrix}{\begin{bmatrix} 1 & 1 & -5 & 3 \\ 1 & 3 & -1 & 0 \\ 1 & -1 & 1 & 2 \end{bmatrix}}$$

Elementary transformation can change a system of linear equations into another system with the same solution. In the matrix the similar operation is called **elementary row operations**.

Elementary transformation:

1. Interchange two equations, $e_1 \leftrightarrow e_2$
2. Multiply both side of an equation by a non-zero number, $k * e_1 \rightarrow e_1^{'}, k \neq 0$
3. Add a multiple of one equation to another equation, $k * e_2 + e_1 \rightarrow e_1^{'}, k \neq 0$

Elementary row operation:

1. Interchange two rows of the matrix, $R_i \leftrightarrow R_j$
2. Multiply the elements of a row by a nonzero constant, $k * R_i \rightarrow R_i^{'}, k \neq 0$
3. Add a multiple of the elements of one row to the corresponding elements of another row. $k * R_j + R_I \rightarrow R_i^{'}, k \neq 0$

2.2.2.2 Gauss-Jordan Elimination

The Gauss-Jordan elimination is used to solve systems of n equations in n variables. The solution of the system could be a unique solution, many solutions or no solutions. We started from the augmented matrix of the system, and perform a sequence of elementary row operation, it will make the matrix become much simpler (the reduced echelon form), the simpler matrix directly leads to the solutions to the system.

The reduced echelon form:

- If all the elements of the row are 0, the row should be at the bottom of the matrix.
- The first nonzero element of each row is 1, and the element called **Leading 1**.
- The leading 1 of each row after the first is positioned to the right of the leading 1 of the previous row.
- All other elements in a column that contains a leading 1 are zero.

The Gauss-Jordan elimination:

- Write down the augmented matrix of the system of linear equations.
- By applying elementary row operation, reduce the augmented matrix into the reduced echelon form. This is done by creating the leading 1s of each row, then creating the elements above and below the leading 1 into 0, move the rows with all zero into the bottom. And make sure the leading 1 of the row is positioned right of the previous row.
- Write down the system of equations corresponding to the reduced echelon form. This system gives the solution.

2.2.3 Addition, Scalar Multiplication, and Multiplication of Matrices

The location of an element in the matrix could be described asa_{ij} , i is the row number and j is the column number the element lies on.

When the two matrices have the same size and the corresponding elements are equal, the two matrices are equal.

Addition

The addition of matrices can only apply to the same size matrices, if the sizes are different, it is called **the sum does not exist**. Matrices A and B are the same size matrices, C is the sum and C will be the same size as A and B. For the elements in C it will be: $c_{ij} = a_{ij} + b_{ij}$.

For example:

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 4 & 3 \\ 8 & 4 \end{bmatrix}$$

$$C = A + B = \begin{bmatrix} 1+4 & 3+2 \\ 4+8 & 3+4 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 12 & 7 \end{bmatrix}$$

Scalar Multiplication

Let A be the Matrix and c be the scalar. The scalar multiplication cA is multiplying every element in A by c. If the result matrix is B, thus B=cA, $b_{ij} = ca_{ij}$. And B is the same size as A.

For example:

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix} \qquad c = 4$$

$$B = cA = \begin{bmatrix} 4\times 4 & 1\times 4 \\ 2\times 4 & 1\times 4 \end{bmatrix} = \begin{bmatrix} 16 & 4 \\ 8 & 4 \end{bmatrix}$$

Multiplication of Matrix

The column number of matrix A is the same as the row number of matrix B, A and B could be multiplied. C is the product, and the size of C is row number of A times column number of B. The element c_{ij} in the matrix C is the sum of multiplying the corresponding elements of row i of A and column j of B.

Let A be a $m \times n$ matrix, B be a $n \times m$ matrix, the i_{th} row of A is $[a_{i1} \quad a_{i2} \quad \cdots \quad a_{in}]$ and the j_{th} column B is $\begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{bmatrix}$. Thus if C=AB, then $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$.

And size of C is $m \times m$.

2.2.4 Algebraic Properties of Matrix Operation

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

Properties of matrix addition and scalar multiplication:

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

Properties of matrix multiplication:

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

In general, multiplication of matrices is not commutative.

2.2.5 Symmetric Matrices and Seriation in Archaeology

Transpose

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

Properties of transpose

Let A and B be matrices and c be a scalar. Assume that the sizes of the matrices are such that the operation can be performed.

1.	$(A + B)^t = A^t + B^t$	$\textit{Transpose of a sum}$
2.	$(cA)^t = cA^t$	$\textit{Transpose of a scalar multiple}$
3.	$(AB)^t = B^tA^t$	$\textit{Transpose of a product}$
4.	$(A^t)^t = A$	

Symmetric matrix

A **symmetric matrix** is a matrix that is equal to its transpose.

Trace

The **trace** of a square matrix A, denied tr(A) is the sum of the diagonal elements of A.

Properties of trace

Let A and B be matrices and c be a scalar. Assume that the sizes of the matrices are such that the operation can be performed.

1. $tr(A+B) = tr(A) + tr(B)$
2. $tr(AB) = tr(BA)$
3. $tr(cA) = ctr(A)$
4. $tr(A^t) = tr(A)$

2.2.6 Inverse of a Matrix

Let A be an n x n matrix. If a matrix B can be found such that $AB = BA = I_n$, then A is said to be **invertible** and B is called the inverse of A. If B does not exist, then A doesn't have inverse. And the inverse of an invariable matrix is unique.

Finding the inverse of a matrix

Let A be an n x n matrix

1. Adjoin A with the identity matrix I_n, the matrix becomes $[A{:}I_n]$.
2. Compute the reduced echelon form of $[A{:}I_n]$.
   If the reduced echelon form is in the form of $[I_n{:}B]$, then B is the inverse of A, if $[I_n{:}B]$ not exist, A has no inverse.

Systems of Linear Equations

The inverse of matrices could be used to solve the solutions for the system of linear equations. Let AX=Y be a system of n linear equations in n variables. If A^{-1} exists, the solution is unique and is given by $X = A^{-1}Y$.

For example:

The system of linear equation is

$$3x - 2y = 2$$

$$x + 2y - 3z = 5$$

$$-x - y + z = 6$$

The system can be written in the flowing matrix form:

$$\begin{bmatrix} 3 & -2 & 0 \\ 1 & 2 & -3 \\ -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 6 \end{bmatrix}$$

If the matrix of coefficients is invertible, the unique solution is

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 6 \end{bmatrix}\begin{bmatrix} 3 & -2 & 0 \\ 1 & 2 & -3 \\ -1 & -1 & 1 \end{bmatrix}^{-1}$$

x, y and z is the result of the multiplication.

2.2.7 Determinants

Determinant of 2x2 matrix

Let A be a 2x2 matrix$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, and its determinant is denoted as $|A|$ and is given by

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

The notation det(A) is another way of determinant.

Minor and cofactor

Let A be a square matrix.

The **minor** of the element a_{ij} is denoted M_{ij}, and M_{ij} is the determinant of the matrix that remains after deleting row i and column j of A.

The **cofactor** of a_{ij} is denoted C_{ij} and defined as $C_{ij} = (-1)^{i+j} M_{ij}$.

The only differ of minor and cofactor is the sign.

Determinant of square matrix

The determinant of square matrix is the sum of the products of the elements of row or column from it and their cofactors.

Let A be an n x n matrix, the cofactor expansions of $|A|$ are:

$$i_{th}\ row\ expansion: |A| = a_{i1}C_{i1} + a_{i2}C_{i2} + a_{i3}C_{i3} + \cdots + a_{in}C_{in}$$

$$j_{th}\ column\ expansion: |A| = a_{1j}C_{1j} + a_{2j}C_{2j} + a_{3j}C_{3j} + \cdots + a_{nj}C_{nj}$$

The determinant of a triangular matrix is the product of its diagonal elements.

Let A be a square matrix, if $|A| = 0$, A is **singular**, otherwise A is **non-singular**.

To determine a square matrix is singular, there are three methods:

1. All the elements of a row (column) are 0.
2. Two rows (columns) are equal.
3. Two rows (columns) are proportional.

Properties of determinants

Let A and B be n x n matrices and c be a nonzero scalar.

1. If a matrix B is obtained from A by multiplying the elements of a row (column) by c then $|B| = c|A|$.
2. If a matrix B is obtained from A by interchanging two rows (columns) then $|B| = -|A|$.
3. If a matrix B is obtained from A by 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. $|A^{-1}| = {}^{1}/_{|A|}\,, assume\ the\ inverse\ is\ exists.$
8.

Matrix of cofactor

Let A be an n x n matrix and C_{ij} be the cofactor of a_{ij}. The matrix whose (i,j)th elements is C_{ij} is called the **matrix of cofactors** of A. The transpose of this matrix is called the **adjoint** of A and is denoted adj(A).

Determinants and Matrix Inverses

Let A be a non-singular square matrix. A is invertible with

$$A^{-1} = \frac{1}{|A|} adj(A)$$

A square matrix A is invertible if and only if $|A| \neq 0$.

Determinants and systems of linear equations

Let AX=B be a system of n linear equations in n variables. If $|A| \neq 0$, there is a unique solution, if $|A| = 0$, there may be no solution or many solutions.

Cramer's Rule

Let AX=B be a system of n linear equations in n variables, and $|A| \neq 0$. The system's unique solution is given by

$$x_1 = \frac{|A_1|}{|A|}, \quad x_2 = \frac{|A_2|}{|A|}, \dots, \ x_n = \frac{|A_n|}{|A|}$$

Where A_i is the matrix obtained by replacing column i of A with B.

2.2.8 Vector Spaces

The **n-space** is a sequence of n real numbers, and it is denoted $\boldsymbol{R^n}$**.** For example (2, 3, 4, 5) is elements in R^4.

The two elements of $\boldsymbol{R^n}$ are equal if their corresponding components are equal. For example let $u = (u_1, \dots, u_n)$ and $v = (v_1, \dots v_n)$ be two elements of R^n , when $u_1 = v_1, \dots, u_{n=}v_n$,u and v are equal.

Let $u = (u_1, \dots, u_n)$ and$v = (v_1, \dots v_n)$ be two elements of R^n, and let c be a scalar

Addition: $u + v = (u_1 + v_1, \dots, u_n + v_n)$

Scalar multiplication:$cu = (cu_1, \dots, cu_n)$

Property

Let u, v and w be vectors in R^n, and let c and d be scalar

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. $1u = u$

Dot product

Let $u = (u_1, \dots, u_n)$ and $v = (v_1, \dots v_n)$ be two elements of R^n,the dot product of u and v is denoted u•v, and it is given by

$$u \cdot v = u_1 v_1 + \cdots + u_n v_n$$

Norm

Let $u = (u_1, \dots, u_n)$ be a element of R^n, the norm denoted as $\|u\|$ is defined by

$$\|u\| = \sqrt{(u_1)^2 + \cdots + (u_n)^2} = \sqrt{u \cdot u}$$

Cross product

Let $u = (u_1, \dots, u_3)$ and $v = (v_1, \dots v_3)$ be two elements of R^3, the cross product of u and v is denoted uXv, and it is given by

$$uXv = (u_2 v_3 - u_3 v_2, \qquad u_3 v_1 - u_1 v_3, \qquad u_1 v_2 - u_2 v_1)$$

Unit vector

A unit vector is a vector whose norm is 1.

Angle between vectors

Let u and v be two nonzero vectors in R^n. The cosine of the angle θ between these vectors is

$$\cos\theta = \frac{u \cdot v}{\|u\|\|v\|} \qquad 0 \leq \theta \leq \pi$$

Two nonzero vectors u and v are orthogonal if and only if u•v=0.

Distance between points

Let $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots y_n)$ be two points in R^n, the distance between x and y is denoted as d(x,y) and is given by

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} = \|x - y\|$$

Linearly combination and span

Let $v_1, v_2, \dots, v_m$ be vectors in a vector space V. If there exist scalars$c_1, c_2, \dots, c_m$such that v can be written

$v = c_1 v_1 + c_2 v_2 + \cdots + c_m v_m$

This is called linear combination.

Every vector in the vector $v_1, v_2, \dots, v_m$ can be expressed as a linear combination of these vectors, v is said to span a vector space.

Linearly Independence and Dependence

The set of vectors $\{v_1, \dots, v_m\}$ is in a vector space V is said to be linearly dependent if there exist scalars $c_1, \dots, c_m$, not all zero, such that

$$c_1 v_1 + \cdots + c_m v_m = 0$$

The set of vectors $\{v_1, \dots, v_m\}$ is in a vector space V is said to be linearly independent if $c_1 v_1 + \cdots + c_m v_m = 0 \; can \; only \; be \; satisfied \; when \; c_1 = 0, \dots, c_m = 0$

Basis and Dimension

A finite set of vectors $\{v_1, \dots, v_m\}$ is called a basis for a vector space V if the set spans V and is linearly independent.

In the vector space V, there is a basis consisting of n vectors, then the dimension of V is said to be n, and is denoted dim(V).

2.2.9 Eigenvalues and Eigenvectors

Let A be an n x n matrix, A scalar λ is called an eigenvalue of A if there exists a nonzero vector x in R^n such that

Ax = λx

The vector is called an eigenvector corresponding to λ. The λ is an eigenvalue of A if and only if $p(\lambda) = |A - \lambda I| = 0$

2.2.10 Advanced Algorithm

There is some better algorithm for the multiplication of matrix, which has less time complexity than the definition algorithm. The time complexity is a way to compare the efficiency of algorithm. The less time complexity, the algorithm is better.

2.2.10.1 Strassen Algorithm

This is an algorithm published by Volker Strassen in 1969. This algorithm is slight faster than the definition algorithm. In the definition algorithm, it takes $O(n^3) = O(n^{\log_2 8})$ multiplications and Strassen reduce the multiplication into $(n^{\log_2 7}) \approx O(n^{2.807})$. and the algorithm is showing as

Let A and B be n x n matrices and matrix C=AB. If the matrices *A*, *B* are not of type 2^n x 2^n we fill the missing rows and columns with zeros. Partition the A, B and C into equal size blocked matrix, which is n/2 x n/2 sub-matrices. In the last we will get the matrices like

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

And in the definition algorithm

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$

$$C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$

$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$

$$C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

There are 8 multiplications; in Strassen we defined 7 new matrices

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1}+B_{2,2})$$

$$M_2 \coloneqq (A_{2,1} + A_{2,2})B_{1,1}$$

$$M_3 \coloneqq A_{1,1}(B_{1,2} - B_{2,2})$$

$$M_4 \coloneqq A_{2,2}(B_{2,1} - B_{1,1})$$

$$M_5 \coloneqq (A_{1,1} + A_{1,2})B_{2,2}$$

$$M_6 \coloneqq (A_{2,1} - A_{1,1})(B_{1,1}+B_{1,2})$$

$$M_7 \coloneqq (A_{1,2} - A_{2,2})(B_{2,1}+B_{2,2})$$

By adding the seven new matrices, the 2x2 sub-matrix C will be

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$C_{1,2} = M_3 + M_5$$

$$C_{2,1} = M_2 + M_4$$

$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

Applying the algorithm to the rest 2x2 sub-matrices to get the final C.

2.2.10.2 Coppersmith-Winograd algorithm

This is the fastest method to do the multiplication of matrices at this moment. It can multiply two $n \times n$matrices in $O(n^{2.376})$ times. The Coppersmith–Winograd algorithm is not used in practice, the reason is it only provides an advantage for matrices so large that they cannot be processed by modern hardware.(wikipedia).

2.2.10.3 LU decomposition

LU decomposition is a way to decompose the matrix into the product of a lower and upper triangular matrix. Let A be a square matrix and LU decomposition is in the form A=LU, L and U are the lower and upper triangular matrix. The best application of the LU decomposition is:

Inverse matrix

Let A be a square matrix, L and U are the lower and upper triangular matrices

The inverse of A can be given by

$$A^{-1} = U^{-1}L^{-1}$$

This is a way to calculate the inverse of matrix in computer implementation.

2.3 The current system

In this section, the two most powerful soft ware Maple and Matlab will be general described and given out some opinion in this project.

Maple

Maple is a powerful mathematic system in computer, and it is a dynamically typed imperative-style programming language. It could solving many mathematical problems, such as integrals, statistics, system of equations, linear algebra and so on. Maple is GUI, however, it is not easy for the user to use. The interface is a worksheet and for the advanced user they could allow to write their own functions in a separate file, and it could be input in the maple. It has a good support in the linear algebra, it is able to solve the equations by solve ({eq1,eq2,...,eqn}). The error message in Maple is not real helpful.

Matlab

Matlab is a more powerful numerical environment and a programming language. MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Same as Maple, it is working in the interface as work sheet, and allows users to write their own functions. It has numbers same features of Maple. Matlab is able to output the result both on the screen and in a file, this is better than Maple.

From both Maple and Matlab, their user interface is not perfect; in this project the interface should be better than that. The interface should be simple, the command should be easy to use and understand. Both of them supported the different data types, and able to input file. This is the project should be inherited from. Unlike Maple and Matlab, this project is only focus on the linear algebra, some of their features are not important in this system.

3. Requirements Analysis and Specification

3.1 Introduction

This section will give out more detail of analysis of the whole computer linear algebra system, such as the theoretical mathematics, the whole system outline, the data handling and other requirements.

The whole system is split into three main parts, vector operation, matrix operations and solving the system of equations. This project is unlike other projects, it is not only a mathematical program, but also it is a program to handling and storing the data.

3.2 Primary objectives

3.2.1 Vector Operation

As mentioned in the literature review, for the vectors there are several basic operations. The theoretical mathematical linear algebra arithmetic has been given in the literature review, project need to provide all the operation in that way. And these operations are the primary objectives in this project. There is the basic list of vector operations:

- Addition
- Subtraction
- Scale Multiplication
- Dot Product
- Norm
- Cross Product
- Distance
- Cosine(angle between two vectors)

3.2.2 Matrix Operations

Matrix operations are other main objectives in the project. The matrix operations are the fundamentals in linear algebra. For every basic operation, the project should apply them and make them work. And the list of matrix operations is given below:

- Addition
- Subtraction
- Scalar Multiplication
- Multiplication
- Transpose

- Inverse
- Elementary Row Operations
    - Row Swap
    - Multiply row by a scalar
    - Add multiples of one row to another
- Minor
- Cofactor
- Determinant

3.2.3 Solve System of Equations

A main use in the linear algebra for mathematicians is solving system of equations. In the large amount of data, to get the result correct and rapid is another primary objective in the project. In the literature review, there are some arithmetical operations to get the solution of unknown variables in the equations. The mathematical method in the literature review has given as:

- Gaussian elimination
- $X = A^{-1}Y$
- Cramer's Rule

Each method has its only limited; in the design section the best algorithm will be chosen.

3.3 Algorithm selection

In the literature review, for one calculation there might be more than one to algorithm to solve. Some algorithms are the standard methods which are the definition ones, and there still are some more advanced algorithms. The advanced algorithms have less time complexity than the normal algorithms. On the other hand, the way of advanced algorithm calculation is more complex than the standard algorithm. In this case, the advanced algorithm is not always the best solution for the requested question. For example, in the matrix multiplication, the time complexity of definition algorithm is $O(n^3)$, and the Strassen algorithm is $O(n^{2.8})$. For small number, the time complexity is almost the same, only for the large number the time difference will appear. In this kind of situation, user should be able to choose the operation method when they want to.

Another requirement is the system should be able to select optimal method by itself. Review the same example above, the definition algorithm and Strassen algorithm of matrix multiplication. When the input matrices are square matrix, the system could use both methods. When the input is in a small dimension, which is called pivot and it is below a reasonable integer, the system should use definition algorithm; if the dimension is greater than the pivot, the system should apply the Strassen algorithm.

There are two arithmetic operations to calculate the determinant of matrix. When the matrix is a triangular matrix, the determinant is just the product of the elements in the diagonal. The system should detect the input matrix is triangular or not, then select best method to operate.

3.4 DATA

The data type in the linear algebra system is integer number and decimal number. The rational number in the system is replaced by decimal number. In the program the number are all designed in double type.

The storage of the data is vector or matrix. In the system, vector is one dimension array. And matrix is two dimension arrays. Vector is defined as an object in the system. Operations in Vector class are setValue, and getValue. System is able to modify every element in the vector by the index number.

Matrix is another object, and in Matrix Class there are more operations. At first the matrix is defined by its dimensions. And then apply basic matrix operation setValue to get the matrix initialize. The operations list is shown below:

- Get the number of rows in the matrix
- Get the number of columns in the matrix
- Set a value in location i,j.
- Get the value from location i,j.
- Set the values in a row.
- Get the values from a row.
- Set the values in a column.
- Get the values from a column.
- Set the whole matrix with array.
- Get the array from the matrix.
- Triangular matrix check.
- Symmetric matrix check.
- Identity matrix check.
- Square matrix check.
- Matrix equality.

toString method is use to represent the matrix in a easy readable way. It will be used in the general output.

3.5 Input and Output

User must be able to input data they want to calculate. In this requirement, there might be several methods:

- User interface
- Command line
- File input/output

The user interface is not the main objective in this project, but it is the best way to present information to the normal users. A good user interface can make user easily understand how the system work. It is a very important way to narrow down the input error by users, and tell users where the errors occur. As a result the error message should be clear and straightforward to the problem.

Command line is easier to implement than user interface. User can input the data in the command line, and use command to calculate the linear operation. In this case, the user manual needs to be provided. Unlike the user interface, it is not a very straightforward way to use. Same as user interface, the error massage should be helpful.

File input and output is the way to handle a large number of inputs. System read a file, and then generates the data into the vector or matrix class, after that it applies the linear operation, at last system output the data in an output file or directly on the screen.

3.6 Testing and Error Handling Requirement

In linear algebra system the data are numbers, and different length of arrays. The errors and exceptions are easy to take place. When the error occurs, the reasonable suggestion should give to users. If only tell user the input is wrong, it doesn't helpful at all.

There are four stages in the testing requirement, the acceptance testing, accuracy testing, component test, and system testing.

In the acceptance testing, the user input will be tested. In this stage, every possible input should be tested, and for the invalid inputs, the error message should be appeared when needed, and the suggestion should be given as well.

In the accuracy testing, the output will compare to the maple. Check the accuracy of the results. As mentioned before this is a mathematic system. To insure the accuracy of the results is the most significant objective.

The system is made up by numbers of component. Each component need to be tested separately. The system is designed in a hierarchy from; a component may have its own components. The testing is from the low level to high level.

The system testing is testing the integration of the whole system. This testing is after component testing. And this testing is more focus on the output, and unification of the whole system.

3.7 Compatibility Requirement

This system should have high compatibility. It should be able to work in different operating systems, such as Windows XP/Vista, UNIX and Mac. The system should be able to work in recent computers.

Java is the language to implement the system. The main reason is, Java is working on its own virtual machine. Java has the same virtual machine in different operating system; it can meet the compatibility much easier than C or C++.

3.8 User Manual Requirement

The user of this system is not only the programmer, but also other users. Normally the users use this system to do a small calculations, such as student doing the course work, or the user don’t want to use large software to do small calculations. In this situation, the user manual document needs to provide.

4. Requirement specification

4.1 Introduction

Requirement specification is the conclusion of requirement analysis. This section will focus on the software requirements, which are functional requirement and non-functional requirement. The table will be present in this section.

4.2 Functional requirement

The functional requirements are services that the system should provide. It tell the user and designer how the system should react to the particular input, and how the system should handle in that particular situations. As mentioned in the section 3, the functions are all the linear algebra operations. In this case the functional requirement should focus on how the system behaves in applying the operations.

Index	**1**
Name	Vector Addition
Description	User input two vectors; System gets the right result after calculation.

Index	**2**
Name	Vector Subtraction
Description	User input two vectors; System gets the difference between two vectors after calculation.

Index	**3**
Name	Vector and Scale Multiplication
Description	User input a vectors and a scale System gets a new vector which is the right result after calculation.

Index	**4**
Name	Dot Product
Description	User input two vectors; System gets the right dot product after calculation.

Index	**5**
Name	Norm
Description	User input one vector; System gets the norm of input vector after calculation.

Index	**6**
Name	Distance
Description	User input two vectors; System gets the right distance between after calculation.

Index	**7**
Name	Cross Product
Description	User input two vectors; System gets the cross product after calculation.

Index	**8**
Name	Cosine between two vectors
Description	User input two vectors; System gets the cosine value after calculation.

Index	**9**
Name	Matrix Addition
Description	User input two matrices; System gets the sum matrix of the inputs after calculation.

Index	**10**
Name	Matrix Subtraction
Description	User input two matrices; System gets the cosine value after calculation.

Index	**11**
Name	Matrix Multiplication
Description	User input two Matrices; System gets the product matrix after calculation.

Index	**13**
Name	Matrix and Scale Multiplication
Description	User input one scale and one matrix; System gets the product matrix after calculation.

Index	**14**
Name	Matrix Transpose
Description	User input one matrix; System gets the transpose of the matrix after calculation.

Index	**15**
Name	Matrix Determinant
Description	User input one matrix; System gets the determinant after calculation.

Index	**16**
Name	Matrix Minor
Description	User input one matrix; User input the element location; System gets the minor after calculation.

Index	**17**
Name	Matrix Cofactor
Description	User input one matrix; User input the element location; System gets the cofactor after calculation.

Index	**18**
Name	Matrix Inverse
Description	User input one matrix; System gets the inverse matrix after calculation.

Index	**19**
Name	Matrix Row Swap
Description	User input one matrix; User input two row number; System gets the new matrix with the swapped rows after calculation.

Index	**20**
Name	Matrix Row Multiplication
Description	User input one matrix; User input one row number and a non zero scale; System gets the new matrix with the modified row after calculation.

Index	21
Name	Matrix Row Addition
Description	User input one matrix; User input first row number; User input second row number and a scale; System gets the new matrix with the first row replaced by 1st row + 2nd row * scale after calculation.

Index	22
Name	Matrix Row Subtraction
Description	User input one matrix; User input first row number; User input second row number and a scale; System gets the new matrix with the first row replaced by 1st row - 2nd row * scale after calculation.

Index	23
Name	Gaussian Elimination
Description	User input one matrix; System gets the row-echelon form matrix after calculation.

Index	24
Name	Solve System of Linear Equations
Description	User input the augmented matrix of the linear equations; System gets the row-echelon form matrix after calculation; System prints out the values of unknown variables.

4.3 Non-functional Requirement

The non-functional requirements are not directly concerned with the specific functions; it always focuses on the system in the whole.

Index	25
Name	Usability
Description	The user should be able to use the program after reading the user manual.

Index	26
Name	Compatibility
Description	The system should be able to work in the different OS, and in different machine.

Index	27
Name	Reliability
Description	The output result should be accuracy. This is the main objective in the project.

Index	28
Name	Maintainability
Description	The souse code should be underhanded by other program. They can make any improvement from the source code.

5. Design

5.1 Introduction

Design is a very important section in the whole system process. To make sure the system meets all the recruitment in section 4. Design should be clear and particular in each area.

5.2 Architectural Design

In linear algebra system, the components should be developed as separate units. The system has three main components: vectors, matrices and equations. The static structural model is shown in fig 5.1.

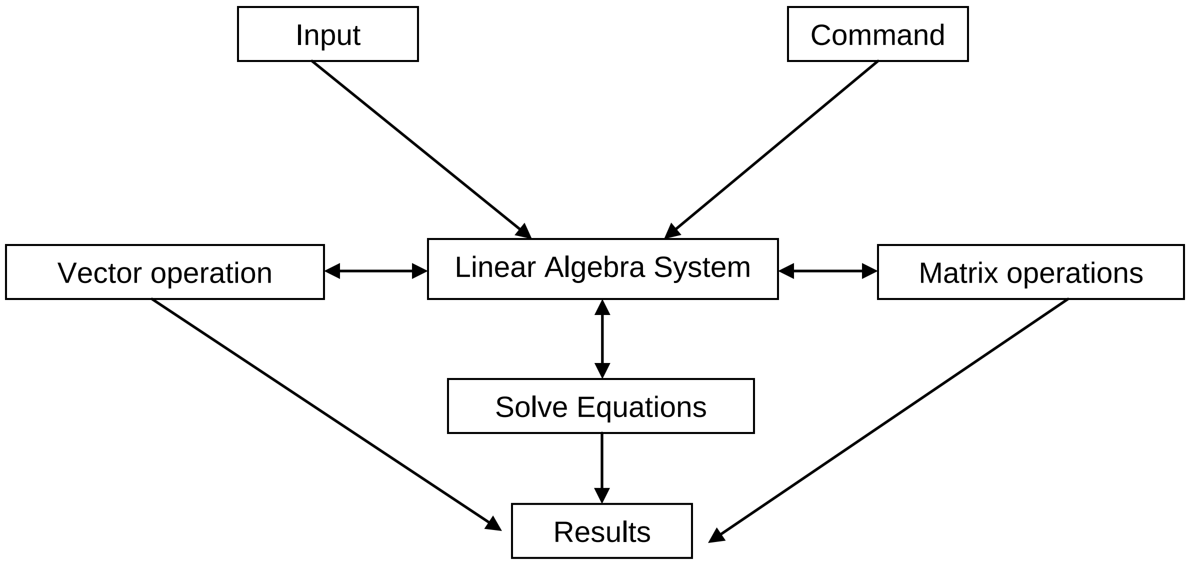
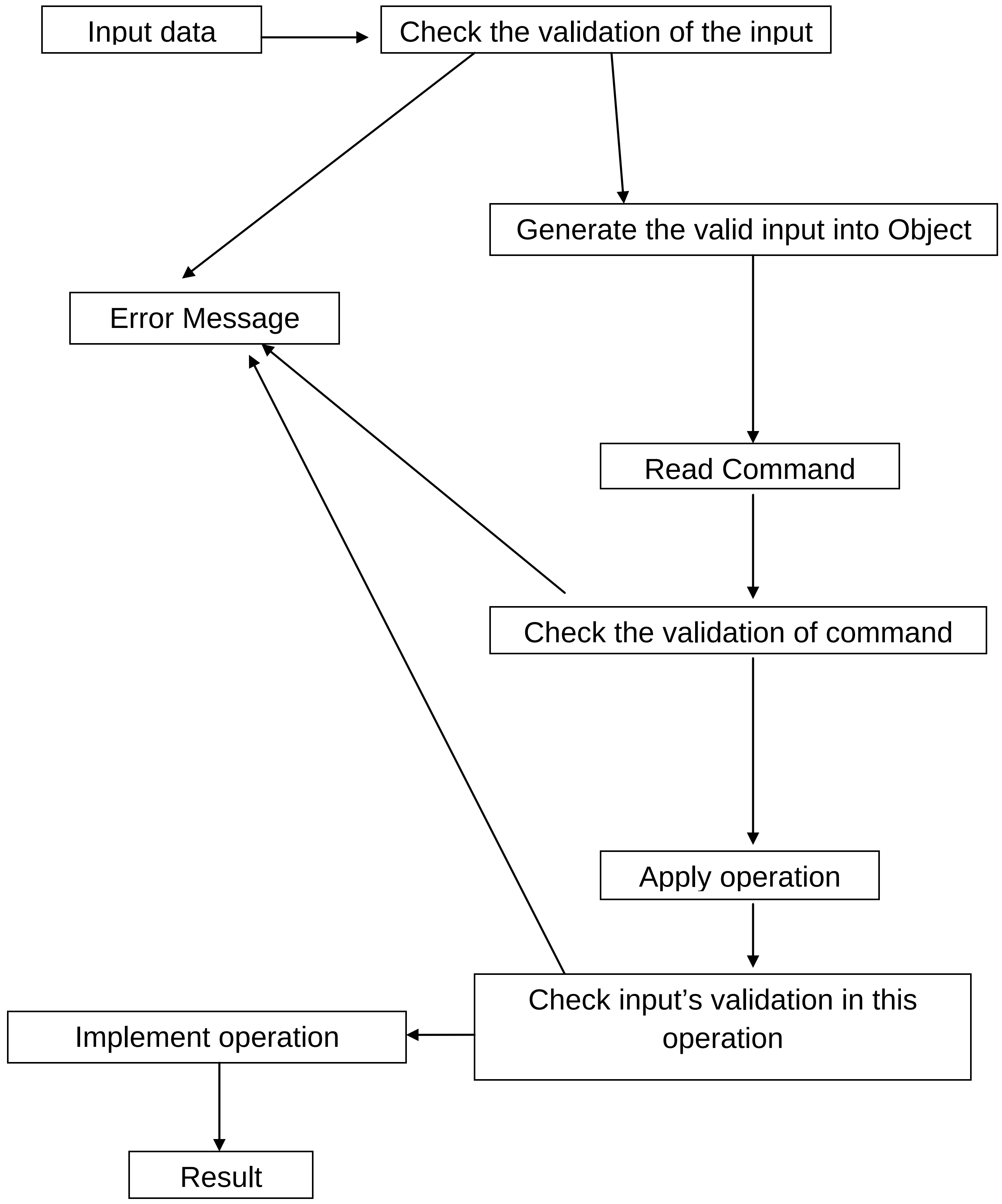


fig 5.1

5.3 Data Flow Model

In the linear algebra system, the basic operation is handling with data. In this kind of situation, the system behaviour of operating data should be designed after the whole system architectural design.


Input data
Check the validation of the input
Generate the valid input into Object
Error Message
Read Command
Check the validation of command
Apply operation
Check input’s validation in this operation
Implement operation
Result

When the system gets the input, at first the input data should be checked, when the input is in valid type such as array and string with numbers, the system generate the input into the right object. And system read the command, also the command need to be checked. If the command is not defined, system must return error message and terminate. After error checking, then apply the operation generated from command. Every operation has its own exceptions; the error check is needed here as well. When error checked in every stage, the result will be calculated accurately.

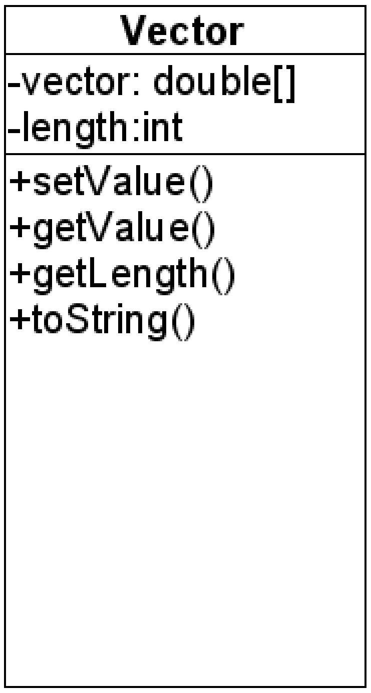
5.4 Object oriented design

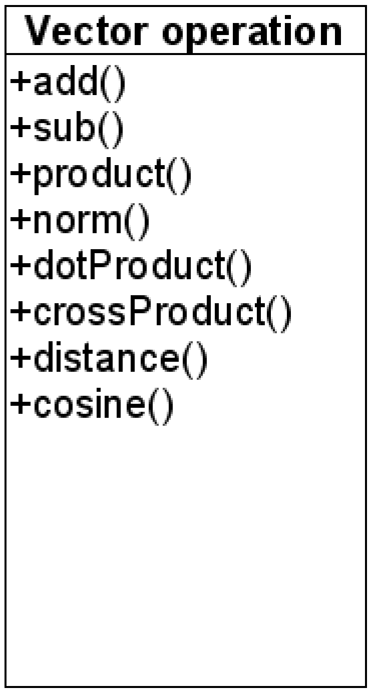
In the program the objects are defined as classes. In this system, there are four basic classes defined:

- Vector Class
- Vector operation class
- Matrix class
- Matrix operation class

The object class is defined in a named rectangle with two sections; the top section is the attributes, and bottom section is the operations.

5.4.1 Vector Objects

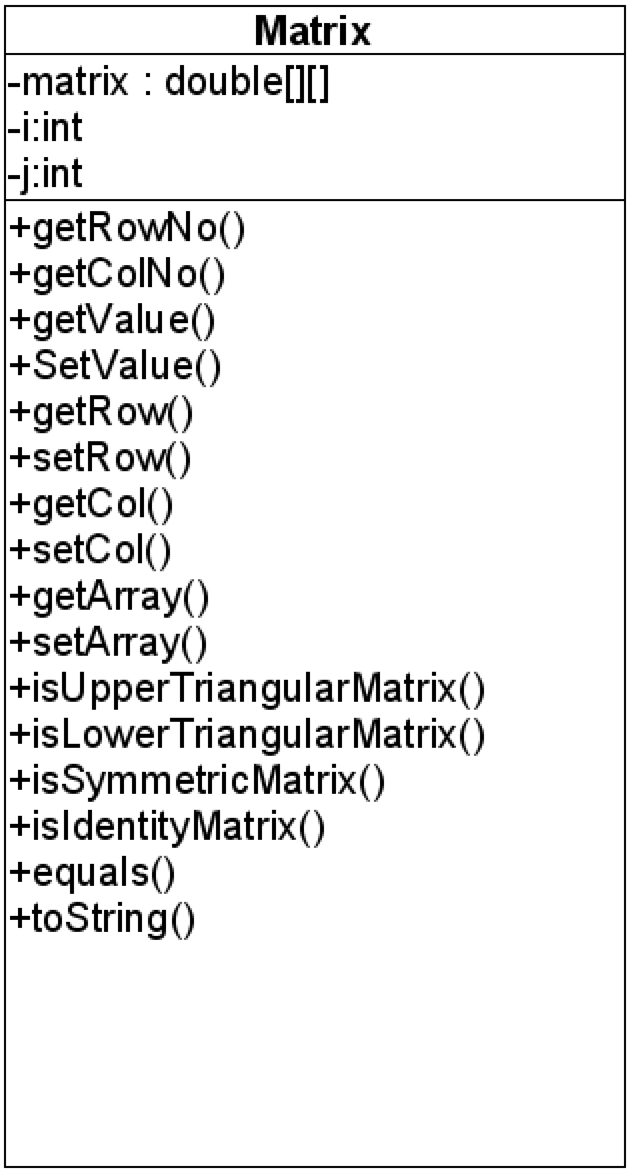




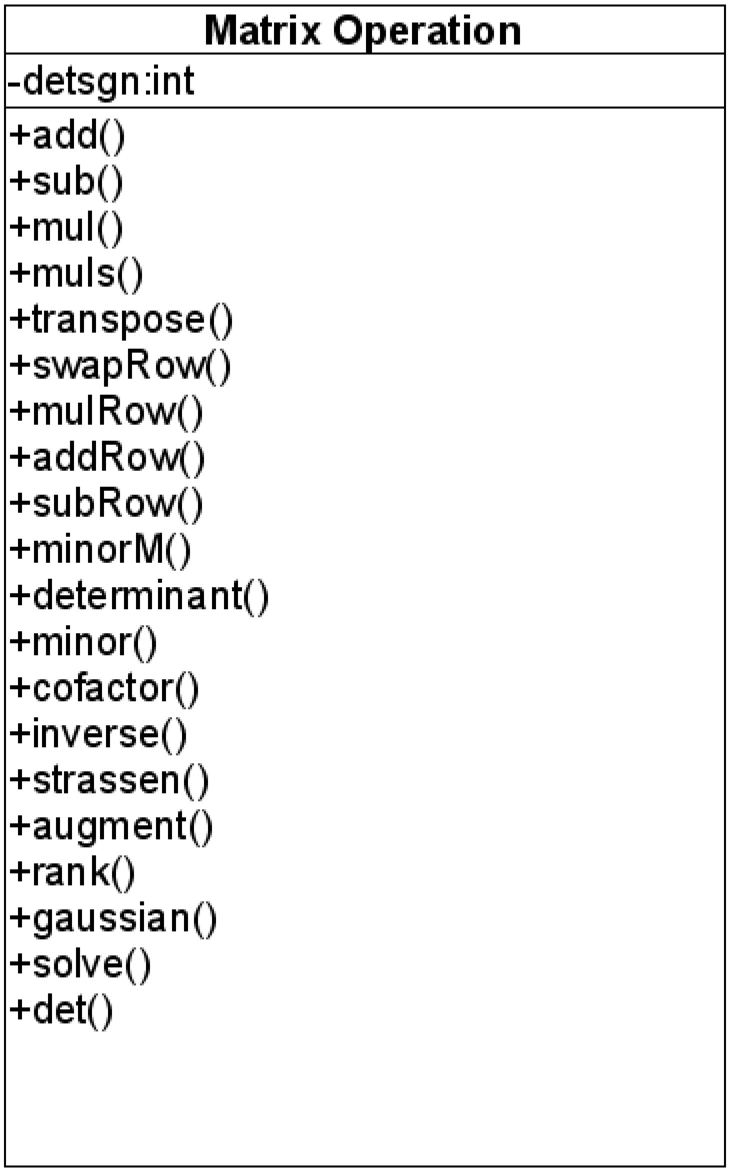
The basic vector class is defined by an integer length and it is a double value one dimension array. As mentioned in requirement, the system should be able to modify and assign the value to every element in the vector. The length is used to handling with error detection; the setValue and getValue are the operations for each element. And toString operation is a way to output the vector in a readable way.

As the named called vector operation. In this class the arithmetic operations are designed. Error detect are applying in each operation.

5.4.2 Matrix Objects



Matrix has more property than vector. In the object class of matrix, the operation is a lot. These operations make the future matrix calculation become much simple. This is the low level of the system. And for matrix operation is the main class in whole project.



The matrix operation has been fully defined in this class object. This class is able to do almost of the matrix operations. In this class there are some fundamental operations, such as elementary row operations. The operation minorM is a helper function, which is deleting the row and column of the select location in the matrix. Using this, the system is easy to get the minor of the matrix. In this class, the hierarchy is very obvious. Each component should be tested separately, and also test them in the whole. Det is another function to get the determinant, which is mentioned in the requirement. In these operations Strassen is one of the advanced matrix multiplication for n x n matrices.

5.4.3 Other classes

There are also the input class and test classes, for these classes they are the applications of these four basic classes. In this section, these methods will not be discussed; they will be described in details at next section.

EquationSolve class is another application of matrix operation, the method is Gaussain Elimination. Rank is used to check the numbers of solutions.

5.5 User Interface Design

The user interface is designed very simple; it is shown in fig 5.2.

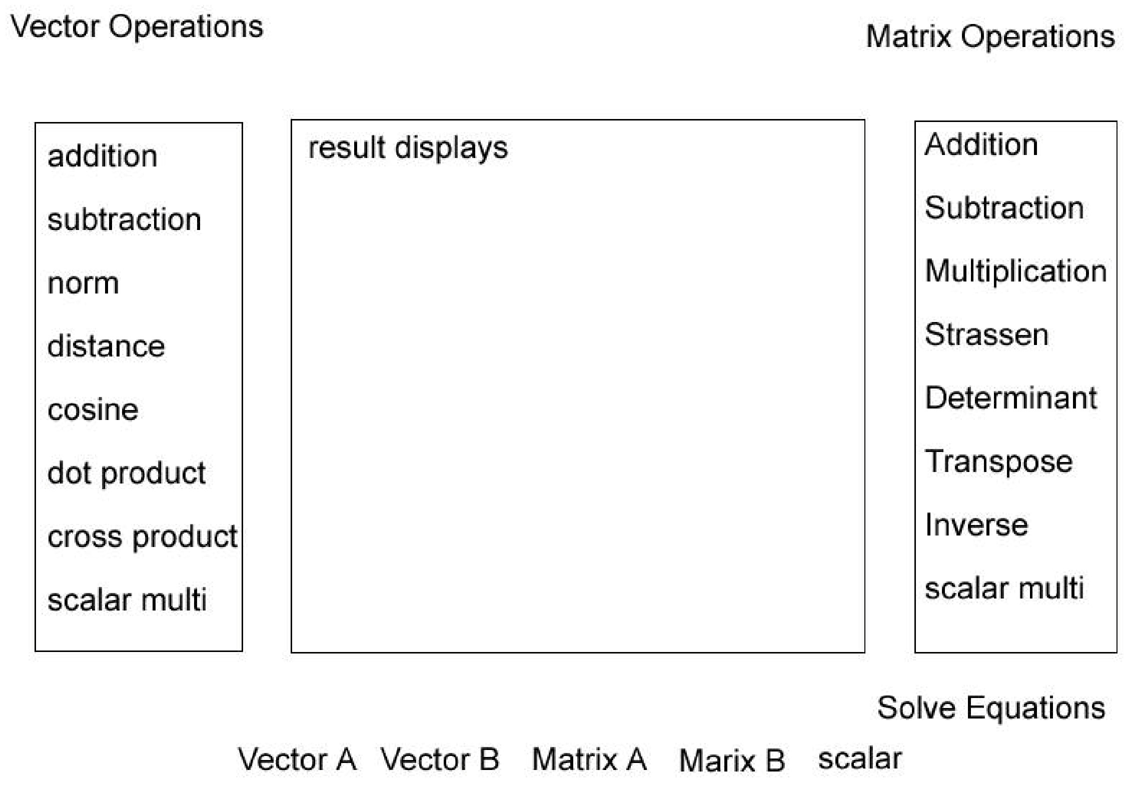


fig 5.2

The bottom buttons are used for input data into the system. After input the data, users could choose the operation they want. And the results will display in the middle of the interface.

If use wants to solve equations, user should type in the augment matrix. Then the system will give out the results.

5.6 Error and Exception Handling

The error handling in the project is based on the logic testing. The exception class will not be written. It might sound not professional. But in this way, every possible exception and error will be checked before doing the calculation. Another reason is checking error in this way is simple to implement as well.

The first kind of errors is the validation of input. The user may give the invalid inputs, such as two matrices are not the same dimension, but the user applies addition, there will be an error.

Another type of error is the dependence. As the system is designed in components, the relation between components may cause errors.

5.7 User manual design

The user manual is a written document. It should give out more examples than the definitions. When the users using the system, they prefer to see how the examples like, they can find adapting for the examples is quicker than reading the instructions.

6. Implementation

6.1 Introduction

This section is stated the work done in the project. The implementations are based on the design, but there might be some differences between them. In this section not every method will be discussed. Only some examples will be described.

6.2 Vector class

The vector is base on the one dimension array, in the class we first define the vector as a double type array. And the length of array is defined as an integer:

```
private double vector[];
private int x;
```

Both of them are private, in the design section the vector has been designed as array with length, the object array should be:

```
public Vector(int x) {
      vector = new double[x];
      this.x = x;
}
```

The length of the vector we still need to take, it will be use in the exception handling and calculations.

```
public int getLength() {
      return x;
```

The operation in this class is setVlaue and getValue, the operation is direct access to the array address of vector:

```
public void setValue(int i, double d) {
      vector[i-1] = d;
}

public double getValue(int i) {
      return vector[i-1];
}
```

In Java the index of length is from 0 to n-1, but in this class, normally in the use of vector, we always say the vector index is form 1 to n. When we access the vector (i), the location in the vector array is i-1.

In the design stage the get array and set array operation have not been designed. But when implementing the program, in vector initialization, if only use loop to get the vector set, it is a waste of the memory. This is the optimal way to set a vector:

```
	public void setArr(double[] array) {
			vector=array;
	}

	public double[] getArrays(){
		return vector;
	}
```

This is almost the vector structure, and operations. As mentioned in design the system output is kind of string:

```
	public String toString(){
		String str="";
		for(int i=1;i<=getLength();i++){
			str+=" "+getValue(i);
		}
		return str;
```

The output will become to string with all the values in the vector. Print like this could be clear for user to read the values and for designer is easy to check the values correct or not.

6.3 Vector Operation class

There are numbers of operations in this section, the operation method is directly adapted from the arithmetic formulas. The full code will be given out in the appendix. There will be given some examples of the operations.

6.3.1 Norm

The arithmetic formula:

$$\|u\| = \sqrt{(u_1)^2 + \cdots + (u_n)^2} = \sqrt{u \cdot u}$$

In this operation, there is only one variable. In the program, we take one vector as the input and set 3 local double values:

```
		double a = 0;
		double b = 0;
		double c = 0;
```

They are all initialized by 0; from the formula firstly, it squared every element in the vector, in the program for all element is a loop; and we set a = element, then we square it and add them together. Variable b is the sum of all the squares. Variable c is square root of b, so c is the return value which is the result.

```
for (int i = 1; i <= v1.getLength(); ++i) {
            a = v1.getValue(i);
            b += a * a;
      }
      c = Math.sqrt(b);
```

From this formula we also can apply the square roots of dot product of the vector self, in this case it could be written as:

```
double r = dotProduct(v1,v1);

c = Math.sqrt(b);
```

In the program I decided to use the original loop method, to do this may avoid the error in operation independence.

6.3.2 Cross product

The arithmetic function is:

$$uXv = (u_2 v_3 - u_3 v_2, \qquad u_3 v_1 - u_1 v_3, \qquad u_1 v_2 - u_2 v_1)$$

In the program we need to check the length of vector in the first place, and we need to access to each element in both u and v. After the multiplication and subtraction, we need to assign the results into a new result vector, and the vector length is 3.

```
v3.setValue(1, v1.getValue(2) * v2.getValue(3) - v1.getValue(3) *
v2.getValue(2));
v3.setValue(2, v1.getValue(3) * v2.getValue(1) - v1.getValue(1) *
v2.getValue(3));
v3.setValue(3, v1.getValue(1) * v2.getValue(2) - v1.getValue(2) *
v2.getValue(1));
```

6.4 Matrix Class

Same as the Vector class, this is the fundamental class in the project. The Matrix is defined as two-dimension array. We can assume the Vector is just a row or a column in matrix. But in the implementation, if we want to use vector operations, we need to convert the input into vector; this may cause more errors in the dependence. So we directly transfer the input into matrix to do the calculation.

Some of the operations in the matrix class are same as in the vector class; the difference is the dimension of arrays.

```
      public Matrix(int i,int j) {
            matrix = new double[i][j];
            this.i = i;
```

```
		this.j = j;
	}
```

The basic get length, set and get value to each element is like:

```
	public int getRowNo() {
		return i;
	}

	public int getColNo() {
		return j;
	}

	public void setValue(int i, int j, double a) {
		matrix[i-1][j-1] = a;
	}

	public double[] getRow(int i){
		return matrix[i-1];
	}
```

And in some of the matrix operations, we need to access to each row or each column. For example, the row 1 is a one- dimension array, the code is like:

```
	public double[] getRow(int i){
		double row[] = new double[j];
		for (int k=0; k<j; ++k){
			row[k]=matrix[i-1][k];
		}
		return row;

	}

	public void setRow(int i, double[] row) {
		for (int k=0; k<j; ++k){
		      matrix[i-1][k]= row[k];
		}
	}
	public double[] getCol(int j){
		double col[] = new double[i];
		for (int k=0; k<i; ++k){
			col[k]=matrix[k][j-1];
		}
		return col;
	}

	public void setCol(int j, double[] col) {
		for (int k=0; k<i; ++k){
		      matrix[k][j-1]= col[k];
		}

	}
```

Setting and getting arrays is same as vector class, and toString is same as vector class.

There are some logic test of the matrix type, they use loop to check elements are match to the property of the special matrix. These Boolean operations are all given in design section. These is an example of upper triangular matrix:

```
public boolean isUpperTriangularMatrix(){
            boolean b=true;
            for(int i=1;i<=getRowNo();i++){
                  for(int j=1;j<i;j++){
                        if(getValue(i,j)!=0){
                              b=false;
                              break;
                        }
                  }
            }
            return b;
      }
```

If any of the elements below the diagonal is not a zero, it is not an upper triangular. And this is the definition of upper triangular matrix. Basically, the Boolean operation is checking the definition.

6.5 Matrix Operation Class

The full code is given in the appendix. The code appears in this section is used to explain the implementation ideas.

6.5.1 Addition

Implanting by loop, calculation is same as the definition. The input has been checked before operation.

6.5.2 Subtraction

All the operation is same as addition, only the operation sign change to minus sign.

6.5.3 Multiplication

The multiplication is to apply dot product of the row in the first matrix to the column of second matrix, and then assign the value to the new matrix. As mention before here is direct access to the two input matrix, it is too complicated to convert into vector and get it back into matrix. The dimension is check before the calculation, and then system creates a new matrix with first matrix’s row length and second matrix’s column length, after doing the dot product, the result are sending into the new matrix. The full code is given in appendix.

6.5.4 Scale multiplication

This multiplication is applying every element in the matrix to multiply by the given scale. This operation is straightforward.

6.5.5 Transpose

This operation is changing the matrix rows into columns. A new Matrix is created: its row number is the original matrix's column number and column number is the original row number, then directly copy form the original matrix.

6.5.6 Elementary row operations

There are three kind of row operations, the definition are given in literature review. In these operations, the original matrix has been modified after the calculation. If we want to keep the original matrix, we need copy it before calculation. Only the multiplication of a nonzero number to a row could change the determinant of the matrix. So this is not used in getting the echelon form, it is only called in the solve operation which is get the reduce echelon form

6.5.7 Determinant

There are two methods to get the determinant of a square matrix, definition method and diagonal method.

In definition method, we need to define the minor and cofactor. In this case I defined an operation to create a minor matrix which is deleting the selected location's row and column, called it minorM.

The new matrix m3 is defined as:

```
Matrix m3 = new Matrix(m1.getRowNo()-1, m1.getColNo()-1);
```

It is smaller than the original. The matrix dimension has not been checked in this stage, because user can not directly access here. The matrix copy is divided in four parts, left top, left bottom, right top and right bottom.

Left top copy, m,n is the input location, m is row index and n is column index.

```
for (int i = 1; i < m; ++i) {
      for (int j =1; j < n; ++j){
      m3.setValue(i, j, m1.getValue(i,j));
      }
}
```

Left bottom copy, from row m+1 to the last row of original:

```
            for (int i = m + 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j < n; ++j){
                  m3.setValue(i-1, j, m1.getValue(i,j));
                  }
            }
```

Right top, row from 1 to m-1, n from n+1 to the last column:

```
            for (int i = 1; i < m; ++i) {
                  for (int j = n + 1; j <= m1.getColNo(); ++j){
                  m3.setValue(i, j-1, m1.getValue(i,j));
                  }
            }
```

Right bottom row form m+1, column form n+1, copy from the original:

```
            for (int i = m + 1; i <= m1.getRowNo(); ++i) {
                  for (int j =n + 1; j <= m1.getColNo(); ++j){
                  m3.setValue(i-1, j-1, m1.getValue(i,j));
                  }
            }
```

The determinant of minorM is the minor of the element (m.n). Minor, cofactor and determinant should define at the same time. The reason is:

$$C_{ij} = (-1)^{i+j} M_{ij}$$
$$|A| = a_{i1}C_{i1} + a_{i2}C_{i2} + a_{i3}C_{i3} + \cdots + a_{in}C_{in}$$

And Mij is the determinant after deleting row i and column j. There is a strong dependence between them:

```
            if (m1.getRowNo() == 1){
                  mi = 0;
            }
            else {
                  mi = determinant((minorM(m1, i, j)));
            }
```

When the input m1 is a 1 x 1 matrix, it doesn’t have minor.

Cofactor has only different with minor, which is the sign, it is defined the system as:

```
co = Math.pow(-1, i + j) * minor(m1, i, j);
```

And the determinant id defined as:

```
            if (m1m == 1){
                  det = m1.getValue(1, 1);
            }

            for (int j = 1; j <= m1n; ++j ){
                  det = det + m1.getValue(1, j) * cofactor(m1, 1, j);
            }
```

When the matrix is 1 x 1, this is the base case, and the value is the only element. Then we apply the definition of determinant.

There is another method of calculate the determinant, that method is transfer the matrix into the triangular matrix. And then get the product for all elements in the diagonal. Then multiply the sign, because every time the row swap, the sign changes. This code is given in appendix.

6.5.8 Inverse

The inverse function is using matrix multiplication, transpose, determinant and cofactor to achieve the formula:

$$A^{-1} = \frac{1}{|A|} adj(A)$$

The determinant can't be zero, and the adj(A) is transpose of cofactor matrix of A. In the program we first create the matrix of cofactors:

```
for (int i = 1; i <= m1.getRowNo(); ++i) {
      for (int j =1; j <= m1.getColNo(); ++j){
      coe.setValue(i, j, cofactor(m1, i, j));
      }
}
```

Coe is defined as the same size of input matrix, then calculate each element's cofactor, then assign to coe.

```
adj = transpose(coe);
```

adj is the transpose of cofactor matrix:

```
inv = muls(adj, 1/det);
```

Inverse is calculated by the equation.

6.5.9 Strassen Algorithm

This function is used for multiplication as well. But it only works on the n X n matrices. When implementing the algorithm. In the first place we need to get the base case of the strassen, when dimension is 1, we just multiple two elements together; when the dimension is two, we able to apply the Strassen:

```
if (m1i ==2){
double v1, v2, v3, v4, v5, v6, v7, c11, c12, c21, c22;

v1= (m1.getValue(1, 1)+ m1.getValue(2, 2))*(m2.getValue(1, 1)+
m2.getValue(2, 2));
v2= (m1.getValue(2, 1)+ m1.getValue(2, 2))*m2.getValue(1, 1);
v3= m1.getValue(1, 1)*(m2.getValue(1, 2)- m2.getValue(2, 2));
v4= m1.getValue(2, 2)*(m2.getValue(2, 1)- m2.getValue(1, 1));
v5= (m1.getValue(1, 1)+ m1.getValue(1, 2))*m2.getValue(2, 2);
v6= (m1.getValue(2, 1)- m1.getValue(1, 1))*(m2.getValue(1, 1)+
m2.getValue(1, 2));
v7=(m1.getValue(1, 2)- m1.getValue(2, 2))*(m2.getValue(2, 1)+
m2.getValue(2, 2));

                  c11=v1+v4-v5+v7;
                  c12=v3+v5;
                  c21=v2+v4;
                  c22=v1-v2+v3+v6;

                  mat.setValue(1, 1, c11);
                  mat.setValue(1, 2, c12);
                  mat.setValue(2, 1, c21);
                  mat.setValue(2, 2, c22);

                  return mat;
            }
```

The second is creating the sub matrices; the idea is same as minorM, copy from the top to bottom, left to right. But in Strassen the matrix dimension should be 2^n. So we need to compare the dimension with 2^n, we get log2 (dimension), and make it into integer, for example, log2 (dimension) =2.3, I make it into 3. If the log2 (dimension) =3, we don't need to make it in integer. In this case we could check the dimension. We take the exact log2(dimension) value, and get the integer type of log2(dimension), the function using in Java is "`Math.ceil`". If the numbers are not same, we need to get the difference between them. The code is like:

```
            int diff = 0;
            if (Math.ceil(log2(m1i))!= log2(m1i)){
                  double p = Math.ceil(log2(m1i));
                  diff =((int)Math.pow(2, p))-m1i;
            }
```

The code might be more clearly than the explanation. Now we get the difference, if there is no difference, that means the matrix is in 2^n dimension. The new dimension now become to original dimension plus difference. Then we create sub matrices with the dimension which is half of new dimension. When the matrix is not in 2^n dimension, the new dimension is larger than original, the added rows and columns are filled in 0s

Eight sub-matrices have been created, and then we apply the Strassen as the definition given.

```
            Matrix t1 = strassen(add(a11,a22),add(b11,b22));
            Matrix t2 = strassen(add(a21,a22),b11);
            Matrix t3 = strassen(a11,sub(b12,b22));
            Matrix t4 = strassen(a22,sub(b21,b11));
            Matrix t5 = strassen(add(a11,a12),b22);
            Matrix t6 = strassen(sub(a21,a11),add(b11,b12));
            Matrix t7 = strassen(sub(a12,a22),add(b21,b22));

            Matrix c11=add(t1,add(t7,sub(t4,t5)));
            Matrix c12=add(t3,t5);
            Matrix c21=add(t2,t4);
            Matrix c22=add(t1,add(t6,sub(t3,t2)));
```

This will call Strassen function itselt, until it gets into the base case. At last we fill c11 c12 c21 c22 into the result matrix.

6.5.10 Gaussian and Reduce echelon Form

This function is used for solving the equations and getting the determinant.

Firstly we sort the matrix by non-zero elements. We create a new matrix with the same dimension, and then we check from the diagonal location in each row. For example, if the location (i, i) is zero, and location (i+10, i) is not zero, we swap the row i with i+10. And the detsgn(determinant sign) become to –detsgn. If every element between the row i to the last row in column i are zero, we check the location from (i,i+1), and this is how the loop working. If the location (i, i) is not zero, straight assign the row into the new matrix.

```
      Matrix mat = new Matrix(m,n);

            if (m==1){
                  mat.setRow(1, m1.getRow(1));
            }

                        int kmax = Math.min(m, n);
            for (int i = 1; i<=m; ++i){
                  int k=i;
                  if(i>=kmax){
                        k=kmax;
                  }
                  if (m1.getValue(i, k)!=0){
                        mat.setRow(i,m1.getRow(i));
                  }
                  else if (m1.getValue(i, k)==0 && k!=m){
                        for (int pi=1; pi<=m-i; ++pi){
                              if (m1.getValue(i+pi, k)!= 0){
                              swapRow(m1, i, i+pi);
                              mat.setRow(i, m1.getRow(i));
                              detsgn = -detsgn;
                              break;
                              }
                              else{
                                    mat.setRow(i, m1.getRow(i));
                              }
                        }
                  }
                  else{
                        mat.setRow(i,m1.getRow(i));
                  }

            }
```

After sorted the matrix, we apply row operation to make the elements under diagonal into 0. This may cause the elements on the diagonal into 0 as well, so we need to sort again. The loop is start from left to right, top to bottom.

```
            for(int j=1; j<=n;++j){
                  int i;
                  i=j;
                  if(i>=kmax){
                        i=kmax;
                  }
                  if (mat.getValue(i, j)!=0){
                        for (int k=1; k<=m-i; ++k){
                              if (mat.getValue(i+k, j)!=0){
                                    addRow(mat, i+k, i, -
1*mat.getValue(i+k, j)/mat.getValue(i, j));
                              }
                        }
                  }
                  else if(mat.getValue(i, j)==0){
                        for (int pi2=j; pi2<=n;++pi2){
                        for (int pi=1; pi<=m-i; ++pi){
                              if (mat.getValue(i+pi, pi2)!= 0){
                              swapRow(mat, i, i+pi);
                              mat.setRow(i, mat.getRow(i));
                              detsgn = -detsgn;
                              break;
                              }
                              else{
                                    mat.setRow(i, mat.getRow(i));
                              }
                        }
                        }
                  }
            }
```

Now the matrix becomes to upper triangular. Then we transfer the elements above the diagonal into 0 by row operations as well. This time we don't need to sort the matrix. Then the loop is from right to left, bottom to top.

```
                  for(int j=n; j>=1;j--){
                  int i =j;
                  if(i>=kmax){
                        i=kmax;
                  }
                  if (mat.getValue(i, j)!=0){
                        for (int k=1; k<i; ++k){
                              if (mat.getValue(i-k, j)!=0){
                                    addRow(mat, i-k, i, -
1*mat.getValue(i-k, j)/mat.getValue(i, j));
                              }
                        }
                  }
```

Because the determinant method 2 is using the triangular matrix, so at this stage we not going to get the reduce echelon form.

Applying the reduce echelon form is using row operation as well. For every row divide the first non-zero element. This is using in the solve equations.

6.6 SolveEquation class

This is an application of MatrixOpreation class, in this class. The input of the class is coefficient matrix, the number of variable, and the result matrix. Doing like this, we could compare the rank of coefficient matrix with the rank of augmented matrix. The system is only work on the unique solutions. When the number of solutions has been checked, we carry on solve the equations. It is get the reduced echelon form of augmented matrix, then print out the last column. The last column is the solution of the system of equations.

6.7 Input Class and User Interface

The input class is in the command line, it read values from the keyboard. To finish an input, simply by press return, then #, then return. The first 2 inputs are the dimensions of matrices. The next two inputs are matrix or vectors. They input should be a one dimension array. The matrix class should transfer the one dimension array into matrix class. If the product of two input dimension is not equal to input array length, the matrix is not defined. The last token is the command; if the command is not defined the system will print out the error messages.

The user interface has not been implemented in this project. It could be work out in the future improvement.

The file input and output is not been implemented, the reason is it is very similar to command input. All the token generators are the same, only the input buffer is different. It will be developed in the future user interface.

7. Testing

7.1 The accuracy testing

As mentioned time and times the project is based on the mathematics. If the arithmetic is wrong, even you have a wonderful user interface.

Vector addition				
Vector 1	Vector 2	Expected result	Actual result	Status
1, 2, 3	2,2,22	3,4,25	3,4,25	ok
null	1,2,1	Error	1,2,1	ok
1,2,3	2,3,1,1,3,1	Error message	Error message	ok
2.4, 3, 1.6	1.6, 1, 1	4,4,2.6	4,4,2.6	ok

Vector subtraction				
Vector 1	Vector 2	Expected result	Actual result	Status
1, 2, 3	2,3,0	-1, -1, 3	-1, -1, 3	ok
null	1,2,1	Error	-1,-2,-1	ok
9,1,1	7,4,21,1	Error message	Error message	ok

Dot product				
Vector 1	Vector 2	Expected result	Actual result	Status
5,2,3	6,1,0	32	32.0	ok
null	9,9,9	Error	0	ok
7,8,3,69	2,3,1,1,3,1	Error message	Error message	ok

Scale Multiplication				
Vector 1	Scale	Expected result	Actual result	Status
5,2,3	6	30, 12,18	30,12,18	ok
null	9	Error	0	ok
2	null	Error message	0	ok

Vector distance				
Vector 1	Vector 2	Expected result	Actual result	Status
5,2,1	3,1,0	2.45	2.449489742783178	ok
1,2,3	2,3,1,1,3,1	Error message	Error message	ok
7,8,3	2, 1, 1	8.832	8.831760866327848	ok

Norm				
Vector 1	Vector 2	Expected result	Actual result	Status
9,1,1		9.11	9.11.043	ok
S,12,a		System Error	Can’t compile	ok

Vector cosine				
Vector 1	Vector 2	Expected result	Actual result	Status
9,1,1	8,3,5	86.9	86.9	ok
13,13,12	7,7,5,8,5	Error message	Error message	ok

Vector cross product				
Vector 1	Vector 2	Expected result	Actual result	Status
9,1,1	8,3,5	2, -37,19	2,-37,19	ok
2,31,12,1	6,3,3,3	Error message	Error message	ok
13,13,12	7,7,5,8,5	Error message	Error message	ok

There is a vector testing class to check all the operations, all the logic check are working as expected. For the Java program, if you don’t give enough input of the method required. You can’t compile, the default initialization of any array is 0s. In the test table, this is the reason, the expected results and actual result are different, and the status is still ok.

Basic vector are working in the manner as I expected.

In the matrix operation, the testing is taking place step by step. First the basic operation is matrix has been tested.

Matrix Basic Operations

Name	Matrix	Function	Expected result	Actual result	status
getValue(i,j)	[1,2,3; 2,2,1]	getValue(1,2)	2	2	ok
	[1,2,3; 2,2,1]	getValue(9,1)	System Error	System Error	ok
setValue(i,j,a)	[0,0,0,0; 1,2,3,4]	setValue(2,1,14)	[0,0,0,0; 14,2,3,4]	[0,0,0,0; 14,2,3,4]	ok
	[0,0,0,0; 1,2,3,4]	setValue(2,5,14)	System Error	System Error	ok
getRow(i)	[0,0,0,0; 1,2,3,4]	getRow(1);	[0,0,0,0]	[0,0,0,0]	ok
	[0,0,0,0; 1,2,3,4]	getRow(0);	System Error	System Error	ok

setRow(i, array[] a)	[0,0,0,0; 1,2,3,4]	setRow(1, [1,2,3,3])	[1,2,3,3; 1,2,3,4]	[1,2,3,3; 1,2,3,4]	ok
	[0,0,0,0; 1,2,3,4]	setRow(1, [1])	[1,0,0,0; 1,2,3,4]	[1,0,0,0; 1,2,3,4]	ok
	[0,0,0,0; 1,2,3,4]	setRow(3, [1])	System error	System error	ok
	[0,0,0,0; 1,2,3,4]	setRow(1, [1,2,3,3,5])	[1,2,3,3; 1,2,3,4]	[1,2,3,3; 1,2,3,4]	ok
getCol(i)	[0,0,0,0; 1,2,3,4]	getCol(1);	[0,1]	[0,1]	ok
	[0,0,0,0; 1,2,3,4]	getCol(12);	System Error	System Error	ok
setCol(i, array[] a)	[0,0,0,0; 1,2,3,4]	setCol(1, [9,9])	[9,0,0,0; 9,2,3,4]	[9,0,0,0; 9,2,3,4]]	ok
	[0,0,0,0; 1,2,3,4]	setCol(1, [1])	[1,0,0,0; 1,2,3,4]	[1,0,0,0; 1,2,3,4]	ok
	[0,0,0,0; 1,2,3,4]	setCol(3, [1])	System error	System error	ok
	[0,0,0,0; 1,2,3,4]	setCol(1, [1,2,3,3,5])	[1,0,0,0; 2,2,3,4]	[1,0,0,0; 2,2,3,4]	ok
getArrays()	[0,0,0,0; 1,2,3,4]	getArrays();	[0,0,0,0; 1,2,3,4]	[0,0,0,0; 1,2,3,4]	ok
setArr(array[][] a)	[0,0,0,0; 1,2,3,4]	setArr([9,9; 8,8])	[9,9,0,0; 8,8,3,4]	[9,9,0,0; 8,8,3,4]	ok
	[0,0,0,0; 1,2,3,4]	setArr([6,0,0,0; 2,2,3,4])	([6,0,0,0; 2,2,3,4]	([6,0,0,0; 2,2,3,4]	ok

The basic operation of a matrix, the user won't directly access and use these operations, this is for the programmer to check and use. This test is for the programmer use, they are never use the invalid array and data as the input.

Matrix Operations

Matrix is difficult to use a table to represent, because the matrices are large, all the matrix operation testing is given in appendix, and it has been given with test codes as well. For all the operations, the arithmetic results are correct.

In Java there is an error when round number, so in this case Gaussian always runs twice to get the reduced echelon form. Some time the element is very tiny, after round it is not a zero, just closed to zero. This is Java's number handling problem, is not a real error.

Solve equations

This class is written separately. A main method is written inside of this class, it could test independently. In the testing, the variable values are correctly in arithmetic.

7.2 Input Testing

The input class is fully tested. For the valid input, the system will give out the errors. For example, if the command is not defined, system will tell user the command cannot find. For the input, when the input is not in the type required, the system will crash. This is one of the most important things. And for the particular calculation by the command, as each component is fully tested in 7.1, the accuracy is reliable.

7.3 User Test Feedback

After the system has done, I asked some of my friend try to use it. For most of them thought the system is helpful for the small calculation. Half of them thought the system should have a user interface, which is easier to use. After training them the use of system, everyone could use the system to do the basic calculations.

7.4 Compatibility Test

The system has been tested in Windows Vista and Windows Xp on my own machines. The Vmware workstation is a kind of virtual machine, in this virtual machine, I test in the Chinese Version Linux6, and Windows 2003. The system is working all the same in the different OS. But I haven’t get a chance to test on the Mac Os, the reason is I don’t have the copyright of the Os.

8. Conclusion

8.1 Achievement

Recall to the project objectives in the introductions, almost all objectives have been achieved.

- The main object is achieved which is the arithmetic of the linear algebra should calculated correctly;
- The system should be able to calculate the main type of numbers;
- The system should give out some advanced algorithm like Strassen;
- The system could be able to do almost all the linear algebra operations;
- The user could use the system after read the manual.

There also some objective I have not achieved:

- A user interface
- Input output file

In conclude, the system should take the main type of number, and in the Linear algebra system, it is able to do almost 20 kind of calculation. Two multiplication methods have been implemented, two determinant methods has been designed. For the system self, the hierarchy is clear and the dependency between components are quite strong. Although there is no user interface, the user such as student is still able to use it.

From the project, I gained the skill of software engineering, from getting the background knowledge, than analyse the whole project and making plan, design the system, then doing the implementation. Although I am not a good programmer, I tried hard to finish the whole system, it is improved my programming skill as well. From this project, I have leant a lot.

8.2 Problem and Future Improvement

8.2.1 Problems in the project

- The user interface is not designed.
- There should be more advanced algorithm.
- The layout of the code is not professional.
- There might be an advanced data structure to store large amount of data.
- There is still a lot possible test need to run.
- There are too many duplicated code in the input class.
- The exception class need to be added.

8.2.2 Future improvement

The number class may defined in the future, in this project the number are all in double, for the rational number, it is still in decimal format. In the future, this kind of data type should be written.

The user interface, the user interface is still need in the project. Try to sort out the problem with Java swing, and added on the interface.

Get more knowledge about the data structure; there might be a link list or a tree to store every array, not only variable one and variable two.

To practise more program writing, make it look more professional, and deleted the duplicated code.

The algorithm and more linear algebra function should be implemented. Such as LU decomposition, QR decomposition and more functions on vector Calculations.

Bibliography

1. Williams, G. 2005. Linear Algebra with Application. 5thed. London: Jones and Bartlett;
2. Lay, David C. 1997. Linear algebra and its applications. 2nd ed. Harlow: Addison-wsley;
3. Davenport, J, H. Siret, Y. Tournier, E. 1988. Computer algebra: system and algorithms for algebra. London: Academic Press;
4. Strang, Gilbert. 1986. Linear algebra and its applications. 3rd ed. San diego: Harcourt Brace Jovanovich;
5. Grossman, Stanley I. 1994. Elementary Linear Algebra. 5th en. London: Sauders College;
6. Sommerville, I. 2007. Software Engineering. 8th ed. London: Addison-Wesley;
7. Deitel, H.M. 2005. Java How to program. 6th ed. London: Pearson Education;
8. Kolman, Bernard. Hill, David R. 2005. Introductory Linear Algebra: an applied first course. 8th ed. N.J.: Pearson/Prentice Hall

Appendix A

User Manual

The system is required a basic java working environment. The Eclipse is strongly recommended.

For the programmer, the test code is enough to get understand the whole system.

In the first place, the user could modify the test class to get the answer you want: the whole test example is in Appendix B.

For the command user, run the input.class, then compile, the message will show as below:

```
the first two inputs are dimension of matrix.
If you would like to use Vector operations.
please press any number then press return, then #, then return to skip
the token.
please type in the 1 matrix dimention, it must be array of numbers, and
it divied by ',' to finish input, press return, then #, then return:
```

In the input system, first two array are token as the matrix dimension. The dimension could not have length 0, the Java will cause problem.

To skip the two dimension array taken, is press anything, then press return, press #, press return, it might look like:

```
please type in the 1 matrix dimention, it must be array of numbers, and
it divied by ',' to finish input, press return, then #, then return:
3
#
please type in the 2 matrix dimention, it must be array of numbers, and
it divied by ',' to finish input, press return, then #, then return:
2
#
```

If you want to use matrix operation, then two dimensions must be in length 2. It like:

```
please type in the 1 matrix dimention, it must be array of numbers, and
it divied by ',' to finish input, press return, then #, then return:
3,2
#
please type in the 2 matrix dimention, it must be array of numbers, and
it divied by ',' to finish input, press return, then #, then return:
2,3
#
```

The first matrix dimension is 3 x 2, and second matrix dimension is 2 x 3;

Next step is getting the input of array; the array will be generated in to vector or matrix:

The example input is like:

```
please type in the 1 input, it must be array of numbers, and it divide
by ',' to finish input, press return, then #, then return:
2,3,4,5,6,1
#
```

If you want to do vector calculation , the input will be generated as v1 ={`2,3,4,5,6,1`}.
If you want to do matrix calculation, the input becomes to m1 ={`2,3;4,5;6,1`} , in the first input we defined the dimension is 3x2.
For the input 2, is doing the same.

Then is the command reader, when you decide which calculation to use, the two inputs will generate to the required type.

The vector operations are:
addV(v1,v2)
#
subV(v1,v2)
#
dotProduct(v1,v2)
#
distance(v1,v2)
#
norm (v1)
#
cosin(v1,v2)
#
crossProduct(v1,v2)
#
The matrix operations are:
add(m1,m2)
#
sub(m1,m2)
#
mul(m1,m2)
#
transpose(m1)
#
det(m1)
#
Det2(m1)
#
inverse(m1)
#

The mul is an auto algorithm method, it will detect the n x nmatrix and apply the Strassen multiplication.

A full example is given below:

```
the first two inputs are dimension of matrix.
If you would like to use Vector operations.
please press any number then press return, then #, then return to skip
the token.
please type in the 1 matrix dimention, it must be array of numbers, and
it divide by ',' to finish input, press return, then #, then return:
2,3
#
please type in the 2 matrix dimention, it must be array of numbers, and
it divide by ',' to finish input, press return, then #, then return:
3,2
#
please type in the 1 input, it must be array of numbers, and it divide
by ',' to finish input, press return, then #, then return:
1,2,2,2,2,3
#
please type in the 2 input, it must be array of numbers, and it divide
by ',' to finish input, press return, then #, then return:
2,2,2,1,4,1
#
lease type in the 1 command, it must be array of numbers, and it divide
by ',' to finish input, press return, then #, then return:
mul(m1, m2)
#
The row number of second matrix must equal to the column number of
first matrix.
The product of the two input matrix is:
 10.0 10.0 10.0
 7.0 28.0 7.0
```

Appendix B

Raw Test code and results

```
public class VectorTest {

	public static void main(String[] args) {

		double aa[]={9,1,1};
		double b[]={8,3,5};
		Vector v1 = new Vector(3);
		Vector v2 = new Vector(3);
		Vector v3 = new Vector(3);

		VectorOperation a = new VectorOperation();
		v1.setArr(aa);
		v2.setArr(b);

		v3=a.addV(v1, v2);
		System.out.println("add: "+v3.toString());

		v3 = a.product(v1, 6);
		System.out.println("scale multi: " +v3.toString());

		v3 = a.subV(v1, v2);
		System.out.println("sub: "+v3.toString());

		System.out.println("distance: "+a.distance(v1, v2));

		System.out.println("dot product: "+a.dotProduct(v1, v2));

		System.out.println("norm: "+a.norm(v1));

		v3 = a.crossProduct(v1, v2);
		System.out.println("cross product: "+v3.toString());

		System.out.println("cosine: "+a.cosine(v1, v2));
	}
}
```

The results are:

```
add: 17.0 4.0 6.0
scale multi: 54.0 6.0 6.0
sub: 1.0 -2.0 -4.0
distance: 4.58257569495584
dot product: 80.0
norm: 9.1104335791443
cross product: 2.0 -37.0 19.0
cosine: 86.92885887910928
```

```java
public class MatTest {
      public static void main(String args[]){
            Matrix m1 = new Matrix(5,5);
            Matrix m2 = new Matrix(5,5);
            Matrix m3 = new Matrix(5,5);
            Matrix m4 = new Matrix(5,5);
            double a[] = {0,0,2,6,4};
            double b[] = {0,0,-1,0,-5};
            double c[] = {1,0,0,8,11};
            double d[] = {2,4,0,1,32};
            double e[] = {0,0,2,2,5};

            double aa[][]= {{ 0, 0,  2, -2, 2},
                        {3,  3, -3,  9,  12},
                        { 4, 4, -2,  1, 12},
                        {6,  3, 3,  9,  12},
                        {9,  3, -3,  19,  12}};

            double bb[][]= {{ 0, 0,  2, 0, 2},
                  {5,  3, -3,  0,  12},
                  {1, 4, 10,  1, 12},
                  {0,  3, 3,  9,  12},
                  {0,  0, 0,  0,  12}};

            double scale =3.0;
            MatrixOperation op = new MatrixOperation();
            m3.setRow(1, a);
            m3.setRow(2, b);
            m3.setRow(3, c);
            m3.setRow(4, d);
            m3.setRow(5, e);

            m1.setArr(aa);
            m2.setArr(bb);

            //copy of m1 use for compare the row operation
            Matrix mat = new Matrix(m3.getRowNo(), m3.getColNo());
            for (int i = 1; i <= m3.getRowNo(); ++i) {
                  for (int j =1; j <= m3.getColNo(); ++j){
                  mat.setValue(i, j, m3.getValue(i,j));
                  }

            }


            Matrix sum=op.add(m1, m2);
            Matrix sub=op.sub(m1, m2);
            Matrix mul=op.mul(m1, m2);
            Matrix muls=op.muls(m1, scale);
            Matrix tran=op.transpose(m1);
            Matrix inv =op.inverse(m1);
            double det1=op.determinant(m1);
            double det2=op.det(m1);
            Matrix mk =op.gaussian(m1);
            Matrix mkk=op.solve(mk);
            Matrix u =op.addRow(m3, 1, 2,-1*m3.getValue(1,
1)/m3.getValue(2, 1));

            System.out.println(m1.equals(m2));
            System.out.println("m1:");
            System.out.println(m1.toString());
            System.out.println("m2:");
            System.out.println(m2.toString());
            System.out.println("m1 + m2:");
            System.out.println(sum.toString());
            System.out.println("m1 - m2:");
            System.out.println(sub.toString());
            System.out.println("m1 * m2");
            System.out.println(mul.toString());
            System.out.println("m1 * scale:");
            System.out.println(muls.toString());
            System.out.println("m1 transpose:");
            System.out.println(tran.toString());
            System.out.println("m1 inverse:");
            System.out.println(inv.toString());
            System.out.println("m1's det(definition):");
            System.out.println(det1);
            System.out.println("m1's det(diagonal):");
            System.out.println(det2);
            System.out.println("m1's reduced echelon form:");
            System.out.println(mkk.toString());

      }
}
```

Resluts:

```
false
m1:
 3.0 3.0 -3.0 9.0 12.0
 4.0 4.0 -2.0 1.0 12.0
 0.0 0.0 2.0 -2.0 2.0
 6.0 3.0 3.0 9.0 12.0
 9.0 3.0 -3.0 19.0 12.0

m2:
 0.0 0.0 2.0 0.0 2.0
 5.0 3.0 -3.0 0.0 12.0
 1.0 4.0 10.0 1.0 12.0
 0.0 3.0 3.0 9.0 12.0
 0.0 0.0 0.0 0.0 12.0

m1 + m2:
 0.0 0.0 4.0 -2.0 4.0
 8.0 6.0 -6.0 9.0 24.0
 5.0 8.0 8.0 2.0 24.0
 6.0 6.0 6.0 18.0 24.0
 9.0 3.0 -3.0 19.0 24.0
```

```
m1 - m2:
 0.0 0.0 0.0 -2.0 0.0
 -2.0 0.0 0.0 9.0 0.0
 3.0 0.0 -12.0 0.0 0.0
 6.0 0.0 0.0 0.0 0.0
 9.0 3.0 -3.0 19.0 0.0

m1 * m2
 0.0 0.0 4.0 0.0 4.0
 120.0 72.0 -72.0 0.0 288.0
 19.0 76.0 190.0 19.0 228.0
 0.0 99.0 99.0 297.0 396.0
 0.0 0.0 0.0 0.0 480.0

m1 * scale:
 0.0 0.0 6.0 -6.0 6.0
 9.0 9.0 -9.0 27.0 36.0
 12.0 12.0 -6.0 3.0 36.0
 18.0 9.0 9.0 27.0 36.0
 27.0 9.0 -9.0 57.0 36.0

m1 transpose:
 0.0 3.0 4.0 6.0 9.0
 0.0 3.0 4.0 3.0 3.0
 2.0 -3.0 -2.0 3.0 -3.0
 -2.0 9.0 1.0 9.0 19.0
 2.0 12.0 12.0 12.0 12.0

m1 inverse:
 0.33333333333333337 -0.3055555555555556 0.16666666666666669 -
0.16666666666666669 0.25
 -1.7666666666666668 -0.013888888888888889 0.11666666666666667
0.7166666666666667 -0.525
 -0.16666666666666669 -0.013888888888888889 -0.08333333333333334 0.25 -
0.125
 -0.2 0.08333333333333334 -0.1 0.1 -0.05
 0.4666666666666667 0.09722222222222222 -0.016666666666666666 -0.15
0.075

m1's det(definition):
-2160.0
m1's det(diagonal):
-2159.9999999999995
m1's reduced echelon form:
 1.0 0.0 0.0 0.0 0.0
 -0.0 1.0 -0.0 -0.0 -0.0
 0.0 1.2335811384723961E-17 1.0 0.0 0.0
 -0.0 -0.0 -0.0 1.0 -0.0
 -0.0 -0.0 -0.0 -0.0 1.0
```

```
public class SolveEqution {

      //coefficient matrix A, x is the numbers of variables, B is the
reslut Matrix
      public static double[] solveE(Matrix A, int x, Matrix B){
            MatrixOperation mo = new MatrixOperation();
            double[] variable = new double[x];
            Matrix ag = mo.augment(A, B);
            Matrix Gag = mo.gaussian(ag);
            Matrix GA = mo.gaussian(A);

            int agRank = mo.rank(Gag);
            int ARank = mo.rank(GA);

            if (agRank != ARank){
                  System.out.println("There is no solutions");
            }
            else if(agRank == ARank && ARank == x ){
                  Matrix xx =mo.solve(Gag);
                  variable = xx.getCol(xx.getColNo());
            }
            else if(agRank == ARank && ARank < x ){
                  System.out.println("Sorry, there are so many
solutions. The system could only work on the unique equations");
            }
            return variable;
      }

        public static void main(String argv[]) {

                int n = 3;
                double x[] = new double[n];
                double b[] = {200,0,0};
                double a[][]= {{ 100,  100,  100},
                               {-100,  300, -100},
                               {-100, -100,  300}};

                Matrix A = new Matrix(3,3);
                A.setArr(a);
                Matrix B = new Matrix(3,1);
                B.setCol(1, b);


               x = solveE(A, n, B);

                for (int i=0; i<n; i++)
                  System.out.println("X" + (i+1) + " = " + x[i]);
              }


}
```

Resluts:

```
X1 = 1.0 X2 = 0.5 X3 = 0.5
```

Appendix C

CODE

Class Vector:

```
public class Vector {

	private double vector[];
	private int x;

	public Vector(int x) {
		vector = new double[x];
		this.x = x;
	}
	/*public Vector(int v[]){
		this(v.length);
		for(int i=0;i<v.length;i++){
			this.vector[i]=v[i];
			}
		}*/
	public int getLength() {
		return x;
	}

	public void setValue(int i, double d) {
		vector[i-1] = d;
	}

	public double getValue(int i) {
		return vector[i-1];
	}

	public void setArr(double[] array) {
			vector=array;

	}

	public double[] getArrays(){
		return vector;
	}

	public String toString(){
		String str="";
		for(int i=1;i<=getLength();i++){
			str+=" "+getValue(i);
		}
		return str;
	}
}
```

Class VectorOperation:

```
public class VectorOperation {

	//addition
	//addition
	public Vector addV(Vector v1, Vector v2) {
		if(v1.getLength()!=v2.getLength()){
			System.out.println("Only the same length vector could
be added together, please try to input the correct vectors.");
		}
		Vector v3 = new Vector(v1.getLength());

		for (int i = 1; i <= v1.getLength(); ++i) {
			v3.setValue(i, v1.getValue(i) + v2.getValue(i));
		}
		return v3;
	}

	//subtraction
	public Vector subV(Vector v1, Vector v2) {
		if(v1.getLength()!=v2.getLength()){
			System.out.println("Only the same length vector could
be subed together, please try to input the correct vectors.");
		}
		Vector v3 = new Vector(v1.getLength());

		for (int i = 1; i <= v1.getLength(); ++i) {
			v3.setValue(i, v1.getValue(i) - v2.getValue(i));
		}
		return v3;
	}

	//product
	public Vector product(Vector v1, int a) {
		Vector v2 = new Vector(v1.getLength());

		for (int i = 1; i <= v1.getLength(); ++i) {
			v2.setValue(i, v1.getValue(i) * a);
		}
		return v2;
	}

	//dot product
	public double dotProduct(Vector v1, Vector v2) {
		double a = 0;
		if(v1.getLength()!=v2.getLength()){
			System.out.println("Only the same length vector could
get Dot Product, please try to input the correct vectors.");
		}
		for (int i = 1; i <= v1.getLength(); ++i) {
			a += (v1.getValue(i) * v2.getValue(i));
		}
		return a;
	}
	//distance
	public double distance(Vector v1, Vector v2) {
		if(v1.getLength()!=v2.getLength()){
			System.out.println("Only the same length vector could 
be get distance, please try to input the correct vectors.");
		}
		double a = 0;
		double b = 0.00;
		for (int i = 1; i <= v1.getLength(); ++i) {
			double j = (v1.getValue(i) - v2.getValue(i)) * 
					(v1.getValue(i) - v2.getValue(i));
			a += j;
		}
		b = Math.sqrt(a);
		return b;
	}

	public double norm(Vector v1) {
		double a = 0;
		double b = 0;
		double c = 0;
	//	double r = dotProduct(v1,v1);
	//	c = Math.sqrt(r);
		for (int i = 1; i <= v1.getLength(); ++i) {
			a = v1.getValue(i);
			b += a * a;
		}
		c = Math.sqrt(b);
		return c;
	}

	public double cosine(Vector v1, Vector v2) {
		if(v1.getLength()!=v2.getLength()){
			System.out.println("Only the same length vector could 
get cosine, please try to input the correct vectors.");
		}
		int a = 0;
		int b = 0;
		int c = 0;
		double temp = 0;

		for (int i = 1; i <= v1.getLength(); ++i) {
			a += (v1.getValue(i) * v2.getValue(i));
		}

		for (int i = 1; i <= v1.getLength(); ++i) {
			temp = v1.getValue(i);
			b += temp * temp;
		}

		for (int i = 1; i <= v2.getLength(); ++i) {
			temp = v2.getValue(i);
			c += temp * temp;
		}

		return a / Math.sqrt(b) * Math.sqrt(c);
	}
	public Vector crossProduct(Vector v1, Vector v2) {

		Vector v3 = new Vector(v1.getLength());

		if (v1.getLength() == 3 || v2.getLength() == 3){
			v3.setValue(1, v1.getValue(2) * v2.getValue(3) -
v1.getValue(3) * v2.getValue(2));
			v3.setValue(2, v1.getValue(3) * v2.getValue(1) -
v1.getValue(1) * v2.getValue(3));
			v3.setValue(3, v1.getValue(1) * v2.getValue(2) -
v1.getValue(2) * v2.getValue(1));
		}
		else{
			System.out.println("Only the length 3 vectors could
be get cross product, please try to input the correct vectors.");

		}

		return v3;
	}
}
```

Class Matrix:

```
public class Matrix {
		private double matrix[][];
		private int i,j;

		public Matrix(int i,int j) {
			matrix = new double[i][j];
			this.i = i;
			this.j = j;
		}

		/*public Matrix(int mat[][]){
			this(mat.length,mat[0].length);
			for(int i=0;i<mat.length;i++){
				for(int j=0;j<mat[i].length;j++){
					this.matrix[i][j]=mat[i][j];
				}
			}
		}*/

		public int getRowNo() {
			return i;
		}

		public int getColNo() {
			return j;
		}

		public void setValue(int i, int j, double a) {
			matrix[i-1][j-1] = a;
		}

		public double[] getRow(int i){
			double row[] = new double[j];
			for (int k=0; k<j; ++k){
				row[k]=matrix[i-1][k];
			}
			return row;

		}

		public void setRow(int i, double[] row) {
			for (int k=0; k<j; ++k){
				matrix[i-1][k]= row[k];
			}
		}

		public double[] getCol(int j){
			double col[] = new double[i];
			for (int k=0; k<i; ++k){
				col[k]=matrix[k][j-1];
			}
			return col;
		}

		public void setCol(int j, double[] col) {
			for (int k=0; k<i; ++k){
				matrix[k][j-1]= col[k];
			}

		}

		public void setArr(double[][] array) {
			for(int i=0;i<array.length;i++){
				for(int j=0;j<array[0].length;j++){
					matrix[i][j]=array[i][j];
				}
			}


		}

		public double[][] getArrays(){
			return matrix;
		}

		public double getValue(int i, int j) {
			return matrix [i-1][j-1];
		}

//check the matrix is UpperTriangularMatrix or not
		public boolean isUpperTriangularMatrix(){
			boolean b=true;
			for(int i=1;i<=getRowNo();i++){
				for(int j=1;j<i;j++){
					if(getValue(i,j)!=0){
```

```
                                b=false;
                                break;
                          }
                    }
              }
              if(getRowNo()!=getColNo()){
                    b=false;
              }
              return b;
        }

//check the matrix is LowerTriangularMatrix or not
        public boolean isLowerTriangularMatrix(){
              boolean b=true;
              for(int i=1;i<=getRowNo();i++){
                    for(int j=getColNo();j>i;j--){
                          if(getValue(i,j)!=0){
                                b=false;
                                break;
                          }
                    }
              }
              if(getRowNo()!=getColNo()){
                    b=false;
              }
              return b;
        }


//Check Symmetric
        public boolean isSymmetricMatrix(){
              boolean b=true;
              for(int i=1;i<getRowNo();i++){
                    for(int j=1;j<i;j++){
                          if(getValue(i,j)!=getValue(j,i)){
                                b=false;
                                break;
                          }
                    }
              }
              if(getRowNo()!=getColNo()){
                    b=false;
              }
              return b;
        }

//Check identity
        public boolean isIdentityMatrix(){
              boolean b=true;
              for(int i=1;i<getRowNo();i++){
                    if(getValue(i,i)!=1){
                          b=false;
                          break;
                    }

              }
              if(getRowNo()!=getColNo()){
```

```
                        b=false;
                }
                return b;
        }

//check matrix a = matrix b
        public boolean equals(Matrix m1){
                boolean b=true;

        if(getRowNo()==m1.getRowNo()&&getColNo()==m1.getColNo()){
                        for(int i=1;i<getRowNo();i++){
                                for(int j=1;j<getColNo();j++){
                                        if(getValue(i, j)!=m1.getValue(i,
j)){
                                                b=false;
                                                break;
                                        }
                                }
                        }
                }
                else{
                        b=false;
                }
                return b;
        }



        public String toString(){
                String str="";
                for(int i=1;i<=getRowNo();i++){
                        for(int j=1;j<=getColNo();j++){
                                str+=" "+getValue(i,j);
                        }
                        str+="\n";
                }
                return str;
        }
}
```

Class Matrix Operation:

```
public class MatrixOperation {

        int detsgn = 1;

        public Matrix add(Matrix m1, Matrix m2) {

                //check the dimensions of m1 and m2
                if(m1.getRowNo() != m2.getRowNo() || m1.getColNo() !=
m2.getColNo()){
                        System.out.println("The SIZE of matrices should be
the same.");
                }
```

```
            Matrix m3 = new Matrix(m1.getRowNo(), m1.getColNo());

            for (int i = 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j <= m1.getColNo(); ++j){
                  m3.setValue(i, j, m1.getValue(i,j) +
m2.getValue(i,j));
                  }
            }
            return m3;
      }


      public Matrix sub(Matrix m1, Matrix m2) {

            //check the dimensions of m1 and m2
            if(m1.getRowNo() != m2.getRowNo() || m1.getColNo() !=
m2.getColNo()){
                  System.out.println("The SIZE of matrices should be
the same.");
                  }

            Matrix m3 = new Matrix(m1.getRowNo(), m1.getColNo());

            for (int i = 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j <= m1.getColNo(); ++j){
                  m3.setValue(i, j, m1.getValue(i,j) -
m2.getValue(i,j));
                  }
            }
            return m3;
      }

      public Matrix mul(Matrix m1, Matrix m2) {
            //get dimensions
            int m1i = m1.getRowNo();
            int m1j = m1.getColNo();
            int m2i = m2.getRowNo();
            int m2j = m2.getColNo();

            //check the dimensions of m1 and m2
            if(m1j != m2i){
                  System.out.println("The row number of second matrix
must equal to the column number of first matrix.");
                  }

            Matrix m3 = new Matrix(m1i, m2j);

            for (int i = 1; i <= m1i; ++i) {
                  for (int j =1; j <= m2j; ++j){
                        double sum = 0;
                        for(int k=1; k <= m1j; ++k){
                              sum +=      m1.getValue(i,k) *
m2.getValue(i,j);
                        }
                        m3.setValue(i, j, sum);
                  }
            }
            return m3;
      }

      public Matrix muls(Matrix m1, double m){
            Matrix m2 = new Matrix(m1.getRowNo(), m1.getColNo());
            double x = m;
            for(int i=1;i <= m1.getRowNo();i++){
                  for(int j=1;j <= m1.getColNo();j++){
                        m2.setValue(i,j, x * m1.getValue(i,j));
                  }
            }

            return m2;
      }
      public Matrix transpose(Matrix m1){
            Matrix mat=new Matrix(m1.getColNo(), m1.getRowNo());
            for(int i=1;i <= m1.getRowNo();i++){
                  for(int j=1;j <= m1.getColNo();j++){
                        mat.setValue(j,i, m1.getValue(i,j));
                  }
            }
            return mat;
      }

      public Matrix swapRow(Matrix m1, int m, int n){

            Matrix temRow = new Matrix(1,m1.getColNo());

            for (int k=1; k <= m1.getColNo(); ++k){
                  temRow.setValue(1, k, m1.getValue(m,k));
            }
            for (int k=1; k <= m1.getColNo(); ++k){
                  m1.setValue(m, k, m1.getValue(n,k));
            }
            for (int k=1; k <= m1.getColNo(); ++k){
                  m1.setValue(n, k, temRow.getValue(1,k));
            }
//          System.out.println(temRow.toString());
            return m1;

      }

      public Matrix mulRow(Matrix m1, int m, double a){

            /*Matrix mat = new Matrix(m1.getRowNo(), m1.getColNo());
            for (int i = 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j <= m1.getColNo(); ++j){
                  mat.setValue(i, j, m1.getValue(i,j));
                  }
            }*/
            for (int j = 1; j <= m1.getColNo(); ++j) {
                  m1.setValue(m, j, m1.getValue(m,j) * a);
            }

            return m1;
      }

//RowM + a * RowN replace rowM
      public Matrix addRow(Matrix m1, int m, int n, double a){

      /*    Matrix mat = new Matrix(m1.getRowNo(), m1.getColNo());
            for (int i = 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j <= m1.getColNo(); ++j){
                  mat.setValue(i, j, m1.getValue(i,j));
                  }
            }*/


            for (int j = 1; j <= m1.getColNo(); ++j) {
                  m1.setValue(m, j, m1.getValue(m,j) + m1.getValue(n,j)
* a);
            }

            return m1;
      }

      //RowM - a * RowN replace rowM      change the value of the mat
address
      public Matrix subRow(Matrix mat, int m, int n, double a){

      /*    Matrix mat = new Matrix(m1.getRowNo(), m1.getColNo());
            for (int i = 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j <= m1.getColNo(); ++j){
                  mat.setValue(i, j, m1.getValue(i,j));
                  }
            }*/


            for (int j = 1; j <= mat.getColNo(); ++j) {
                  mat.setValue(m, j, mat.getValue(m,j) -
mat.getValue(n,j) * a);
            }

            return mat;
      }

      public Matrix minorM(Matrix m1, int m, int n) {
            Matrix m3 = new Matrix(m1.getRowNo()-1, m1.getColNo()-1);

            for (int i = 1; i < m; ++i) {
                  for (int j =1; j < n; ++j){
                  m3.setValue(i, j, m1.getValue(i,j));
                  }
            }

            for (int i = m + 1; i <= m1.getRowNo(); ++i) {
                  for (int j =1; j < n; ++j){
                  m3.setValue(i-1, j, m1.getValue(i,j));
                  }
            }

            for (int i = 1; i < m; ++i) {
```

```
                        for (int j = n + 1; j <= m1.getColNo(); ++j){
                        m3.setValue(i, j-1, m1.getValue(i,j));
                        }
                }

                for (int i = m + 1; i <= m1.getRowNo(); ++i) {
                        for (int j =n + 1; j <= m1.getColNo(); ++j){
                        m3.setValue(i-1, j-1, m1.getValue(i,j));
                        }
                }

                return m3;

        }



        public double determinant(Matrix m1) {
                int m1m = m1.getRowNo();
                int m1n = m1.getColNo();
            double det = 0.0;
                if (m1m != m1n){
                        System.out.println("invalid input, only square matrix
has the determinant");
                }

                if (m1m == 1){
                        det = m1.getValue(1, 1);
                }

                for (int j = 1; j <= m1n; ++j ){
                        det = det + m1.getValue(1, j) * cofactor(m1, 1, j);
                }

                return det;
        }


        public double minor(Matrix m1, int m, int n){
                double mi = 0.0;
                int i=m;
                int j=n;
                if (m1.getRowNo() == 1){
                        mi = 0;
                }
                else {
                        mi = determinant((minorM(m1, i, j)));
                }
                return mi;
        }

        public double cofactor(Matrix m1, int m, int n){
                double co = 0.0;
                int i=m;
                int j=n;
                co = Math.pow(-1, i + j) * minor(m1, i, j);
                return co;
```

```
	}

	public Matrix inverse(Matrix m1){

		if (m1.getRowNo() != m1.getColNo()){
			System.out.println("invalid input, only square matrix 
has the inverse");
		}

		Matrix inv = new Matrix(m1.getRowNo(), m1.getColNo());
		Matrix coe = new Matrix(m1.getRowNo(), m1.getColNo());
		Matrix adj = new Matrix(m1.getRowNo(), m1.getColNo());
		double det = determinant(m1);

		if (det == 0){
			System.out.println("Only Non-singular Matrix has 
inverse");
		}

		for (int i = 1; i <= m1.getRowNo(); ++i) {
			for (int j =1; j <= m1.getColNo(); ++j){
			coe.setValue(i, j, cofactor(m1, i, j));
			}
		}

		adj = transpose(coe);

		inv = muls(adj, 1/det);

		return inv;
	}

	public Matrix strassen(Matrix m1, Matrix m2){
		int m1i = m1.getRowNo();
		int m1j = m1.getColNo();
		int m2i = m2.getRowNo();
		int m2j = m2.getColNo();
		Matrix mat = new Matrix(m1i,m1j);
		if (m1i != m1j || m2i != m2j || m2i != m1i){
			System.out.println("Only two same dimension square 
Matrices could use Strassen");
		}
		if (m1i ==1){
				mat.setValue(1, 1, m1.getValue(1, 1)*
m2.getValue(1, 1));
			return mat;
		}
		if (m1i ==2){
		     double v1, v2, v3, v4, v5, v6, v7, c11, c12, c21, c22;

			v1= (m1.getValue(1, 1)+ m1.getValue(2, 
2))*(m2.getValue(1, 1)+ m2.getValue(2, 2));
			v2= (m1.getValue(2, 1)+ m1.getValue(2, 
2))*m2.getValue(1, 1);
			v3= m1.getValue(1, 1)*(m2.getValue(1, 2)-
m2.getValue(2, 2));
```

```
                  v4= m1.getValue(2, 2)*(m2.getValue(2, 1)-
m2.getValue(1, 1));
                  v5= (m1.getValue(1, 1)+ m1.getValue(1,
2))*m2.getValue(2, 2);
                  v6= (m1.getValue(2, 1)- m1.getValue(1,
1))*(m2.getValue(1, 1)+ m2.getValue(1, 2));
                  v7=(m1.getValue(1, 2)- m1.getValue(2,
2))*(m2.getValue(2, 1)+ m2.getValue(2, 2));

                  c11=v1+v4-v5+v7;
                  c12=v3+v5;
                  c21=v2+v4;
                  c22=v1-v2+v3+v6;

                  mat.setValue(1, 1, c11);
                  mat.setValue(1, 2, c12);
                  mat.setValue(2, 1, c21);
                  mat.setValue(2, 2, c22);

                  return mat;
            }

            //check the matrix is 2^n, if not, fill the 0s

            int diff = 0;
            if (Math.ceil(log2(m1i))!= log2(m1i)){
                  double p = Math.ceil(log2(m1i));
                  diff =((int)Math.pow(2, p))-m1i;
            }

            int newd = m1i+diff; //new matrix dimension
            int subi = newd/2;

            Matrix a11 = new Matrix(subi,subi);
            Matrix a12 = new Matrix(subi,subi);
            Matrix a21 = new Matrix(subi,subi);
            Matrix a22 = new Matrix(subi,subi);
            Matrix b11 = new Matrix(subi,subi);
            Matrix b12 = new Matrix(subi,subi);
            Matrix b21 = new Matrix(subi,subi);
            Matrix b22 = new Matrix(subi,subi);

            // set a11 values
            for (int i = 1; i <= subi; ++i) {
                  for (int j = 1; j <= subi; ++j){
                  a11.setValue(i, j, m1.getValue(i,j));
                  }
            }

            //set a12 value
            for (int i = 1; i <= subi; ++i) {
                  for (int j = subi+1; j <= m1j; ++j){
                  a12.setValue(i, j-subi, m1.getValue(i,j));
                  }
            }
            //fill 0s if needed
            if (newd > m1i){
```

```
        for (int i = 1; i <= subi; ++i) {
              for (int j = m1j+1; j <= newd; ++j){
              a12.setValue(i, j-subi, 0);
              }
        }
  }

  //set a21
  for (int i = subi+1; i <= m1i; ++i) {
        for (int j = 1; j <= subi; ++j){
        a21.setValue(i-subi, j, m1.getValue(i,j));
        }
  }
  //fill 0s if needed
  if (newd > m1i){
        for (int i = m1i+1; i <= newd; ++i) {
              for (int j = 1; j <= subi; ++j){
              a21.setValue(i-subi, j, 0);
              }
        }
  }

  //set a22
  for (int i = subi + 1; i <= m1i; ++i) {
        for (int j = subi+1; j <= m1j; ++j){
        a22.setValue(i-subi, j-subi, m1.getValue(i,j));
        }
  }
  //fill 0s if needed
  if (newd > m1i){
        for (int i = m1i+1; i <= newd; ++i) {
              for (int j = subi+1; j <= newd; ++j){
              a22.setValue(i-subi, j-subi, 0);
              }
        }
        for (int i = subi+1; i <= m1i; ++i) {
              for (int j = m1j+1; j <= newd; ++j){
              a22.setValue(i-subi, j-subi, 0);
              }
        }
  }

  // set b11 values
  for (int i = 1; i <= subi; ++i) {
        for (int j = 1; j <= subi; ++j){
        b11.setValue(i, j, m2.getValue(i,j));
        }
  }

  //set b12 value
  for (int i = 1; i <= subi; ++i) {
        for (int j = subi+1; j <= m1j; ++j){
        b12.setValue(i, j-subi, m2.getValue(i,j));
        }
  }
  //fill 0s if needed
  if (newd > m1i){
```

```
		for (int i = 1; i <= subi; ++i) {
			for (int j = m1j+1; j <= newd; ++j){
			b12.setValue(i, j-subi, 0);
			}
		}
	}

	//set b21
	for (int i = subi+1; i <= m1i; ++i) {
		for (int j = 1; j <= subi; ++j){
		b21.setValue(i-subi, j, m2.getValue(i,j));
		}
	}
	//fill 0s if needed
	if (newd > m1i){
		for (int i = m1i+1; i <= newd; ++i) {
			for (int j = 1; j <= subi; ++j){
			a21.setValue(i-subi, j, 0);
			}
		}
	}

	//set b22
	for (int i = subi + 1; i <= m1i; ++i) {
		for (int j = subi+1; j <= m1j; ++j){
		b22.setValue(i-subi, j-subi, m2.getValue(i,j));
		}
	}
	//fill 0s if needed
	if (newd > m1i){
		for (int i = m1i+1; i <= newd; ++i) {
			for (int j = subi+1; j <= newd; ++j){
			b22.setValue(i-subi, j-subi, 0);
			}
		}
		for (int i = subi+1; i <= m1i; ++i) {
			for (int j = m1j+1; j <= newd; ++j){
			a22.setValue(i-subi, j-subi, 0);
			}
		}
	}

	//calculate the 7 temp matrix

	Matrix t1 = strassen(add(a11,a22),add(b11,b22));
	Matrix t2 = strassen(add(a21,a22),b11);
	Matrix t3 = strassen(a11,sub(b12,b22));
	Matrix t4 = strassen(a22,sub(b21,b11));
	Matrix t5 = strassen(add(a11,a12),b22);
	Matrix t6 = strassen(sub(a21,a11),add(b11,b12));
	Matrix t7 = strassen(sub(a12,a22),add(b21,b22));

	Matrix c11=add(t1,add(t7,sub(t4,t5)));
	Matrix c12=add(t3,t5);
	Matrix c21=add(t2,t4);
	Matrix c22=add(t1,add(t6,sub(t3,t2)));
```

```
            for(int i=1; i<=subi; ++i){
                  for(int j=1; j<=subi; ++j){
                        mat.setValue(i, j, c11.getValue(i, j));
                  }
            }

            for(int i =1; i<=subi; ++i){
                  for(int j = subi+1; j<= m1i; ++j){
                        mat.setValue(i, j, c12.getValue(i, j-subi));
                  }
            }

            for(int i = subi+1; i<=m1i; ++i){
                  for(int j = 1; j<= subi; ++j){
                        mat.setValue(i, j, c21.getValue(i-subi, j));
                  }
            }

            for(int i =subi+1; i<=m1i; ++i){
                  for(int j = subi+1; j<= m1i; ++j){
                        mat.setValue(i, j, c22.getValue(i-subi, j-
subi));
                  }
            }
            return mat;
      }

      public double log2(double a){
            double b=0;
            b= Math.log(a)/Math.log(2);
            return b;
      }

      //solving equations

      public Matrix augment(Matrix m1, Matrix m2){
            if(m1.getRowNo() != m2.getRowNo()){
                  System.out.println("Can not get the augment matrix");
            }
            Matrix m = new Matrix(m1.getRowNo(),m1.getColNo()+
m2.getColNo());
            for (int j=1; j<=m1.getColNo();++j){
                  m.setCol(j, m1.getCol(j));
            }

            for (int j=m1.getColNo()+1; j<=m1.getColNo()+m2.getColNo();
++j){
                  m.setCol(j, m2.getCol(j-m1.getColNo()));
            }
            return m;
    }

      public Matrix gaussian(Matrix m1){
            int m = m1.getRowNo();
            int n = m1.getColNo();
            Matrix mat = new Matrix(m,n);
            if (m==1){
                  mat.setRow(1, m1.getRow(1));
            }

            int kmax = Math.min(m, n);
            for (int i = 1; i<=m; ++i){
                  int k=i;
                  if(i>=kmax){
                        k=kmax;
                  }
                  if (m1.getValue(i, k)!=0){
                        mat.setRow(i,m1.getRow(i));
                  }
                  else if (m1.getValue(i, k)==0 && k!=m){
                        for (int pi=1; pi<=m-i; ++pi){
                              if (m1.getValue(i+pi, k)!= 0){
                              swapRow(m1, i, i+pi);
                              mat.setRow(i, m1.getRow(i));
                              detsgn = -detsgn;
                              break;
                              }
                              else{
                                    mat.setRow(i, m1.getRow(i));
                              }
                        }
                  }
                  else{
                        mat.setRow(i,m1.getRow(i));
                  }

            }

            for(int j=1; j<=n;++j){
                  int i;
                  i=j;
                  if(i>=kmax){
                        i=kmax;
                  }
                  if (mat.getValue(i, j)!=0){
                        for (int k=1; k<=m-i; ++k){
                              if (mat.getValue(i+k, j)!=0){
                                    addRow(mat, i+k, i, -
      1*mat.getValue(i+k, j)/mat.getValue(i, j));
                              }
                        }
                  }
                  else if(mat.getValue(i, j)==0){
                        for (int pi2=j; pi2<=n;++pi2){
                        for (int pi=1; pi<=m-i; ++pi){
                              if (mat.getValue(i+pi, pi2)!= 0){
                              swapRow(mat, i, i+pi);
                              mat.setRow(i, mat.getRow(i));
                              detsgn = -detsgn;
                              break;
                              }
                              else{
                                    mat.setRow(i, mat.getRow(i));
                              }
                        }
                        }
                  }
            }

            for(int j=n; j>=1;j--){
                  int i =j;
                  if(i>=kmax){
                        i=kmax;
                  }
                  if (mat.getValue(i, j)!=0){
                        for (int k=1; k<i; ++k){
                              if (mat.getValue(i-k, j)!=0){
                                    addRow(mat, i-k, i, -
1*mat.getValue(i-k, j)/mat.getValue(i, j));
                              }
                        }
                  }
                  /*else if(mat.getValue(i, j)==0){
                        for (int pi=1; pi<i; ++pi){
                              if (mat.getValue(i-pi, j)!= 0){
                              swapRow(mat, i, i-pi);
                              mat.setRow(i, mat.getRow(i));
                              detsgn = -detsgn;
                              break;
                              }
                              else{
                                    mat.setRow(i, mat.getRow(i));
                              }
                        }

                  }*/
            }
            return mat;
      }

      public int rank(Matrix A){
            int r=0;
            for(int i=1; i <=A.getRowNo();++i){
                  for(int j=1; j <=A.getColNo(); ++j){
                        if (A.getValue(i, j)!=0){
                              r++;
                              break;
                        }
                  }
            }
            return r;
      }

      public Matrix solve(Matrix A){

            int m = A.getRowNo();
            int n = A.getColNo();
            Matrix M = new Matrix(m,n);

            M = gaussian(A);
            int kmax = Math.min(m, n);
            for (int i=1; i<=kmax; ++i){
                  if (M.getValue(i, i)!=0){
                        mulRow(M, i, 1/M.getValue(i, i));
                        }
                  else{
                        for(int k=i; k<=n; ++k){
                              if (M.getValue(i, k)!=0){
                                    mulRow(M, i, 1/M.getValue(i, k));
                                    break;
                                    }
                        }

                  }

            }


            return M;

      }

      public double det(Matrix m1){
            double det =1;
            Matrix M = gaussian(m1);
            if (M.isUpperTriangularMatrix()||
M.isLowerTriangularMatrix()){
                  for (int i=1; i<=M.getRowNo(); ++i){
                        det *= M.getValue(i, i);

                  }
            }
            else{
                  System.out.println("invalid input, only square matrix
has the determinant");
            }
            det*= detsgn;
            return det;
      }
}
```

Class Input:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;


public class input {


      public String[] getDimension(int i){
              String s="";
              System.out.print("please type in the "+ i +" matrix
dimention, it must be array of numbers, and it divide by ',' to finish
input, press return, then #, then return:\n");
              String[] array =null;
              while(true){
               try{
                BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
                s=in.readLine();
                if(!s.equals("#"))
                {
                  array = s.split(",");
                }
                else
                {
                  break;
                }
                }catch(IOException e){}
              }
            return array;
      }

      public String[] getArray(int i){
              String s="";
              System.out.print("please type in the "+ i +" input, it
must be array of numbers, and it divide by ',' to finish input, press
return, then #, then return:\n");
              String[] array =null;
              while(true){
               try{
                BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
                s=in.readLine();
                if(!s.equals("#"))
                {
                  array = s.split(",");
                }
                else
                {
                  break;
                }
                }catch(IOException e){}
              }
            return array;
      }

      public String[] getComm(int i){
              String s="";
              System.out.print("lease type in the "+ i +" command, it
must be array of numbers, and it divide by ',' to finish input, press
return, then #, then return:\n");
              String[] array =null;
              while(true){
               try{
                BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
                s=in.readLine();
                if(!s.equals("#"))
                {
                  array = s.split(" ");
                }
                else
                {
                  break;
                }
                }catch(IOException e){}
              }
            return array;

      }


      public double[] getValue(String[] array1){
            double[] array = new double[array1.length];
            for(int i=0;i<array1.length;i++)
            {
                  array[i] = Double.parseDouble(array1[i]);
            }
            return array;
      }

      public static  void main(String[] args) {
            //welcome message
            System.out.print("the first two inputs are dimension of
matrix.\n");
            System.out.print("If you would like to use Vector
operations.\n");
           System.out.print("please press any number then press return,
then #, then return to skip the token.\n");
            // m1 demention
            String[] m1Demention =null;
            //m2 demention
            String[] m2Demention =null;
            //m1 array / v1
            String[] array1 =null;
            //m2 array /v2
            String[] array2 =null;

            //
            double[] array3 = null;
            //comand array
            String[] Carray = null;
            //
            input in = new input();
            //获取值
            m1Demention =in.getDimension(1);
            m2Demention =in.getDimension(2);
            array1 = in.getArray(1);
            array2 = in.getArray(2);
            Carray = in.getComm(1);
            //read commond
            String s ="";
            for (int i=0; i<Carray.length; ++i){
            s = s + Carray[i];
            }
            double[] d1=in.getValue(m1Demention);
            double[] d2=in.getValue(m1Demention);
            int m1i=0;
            int m1j=0;
            int m2i=0;
            int m2j=0;


            if(d1.length != 2 || d1.length != 2){
                  System.out.println("Wrong matrix dimension input, you
can only use vector operations this time");
            }
            else{
                  m1i=(int)d1[0];
                  m1j=(int)d1[1];
                  m2i=(int)d1[0];
                  m2j=(int)d1[1];
            }

            //vector operations

            double[] in1=in.getValue(array1);
            double[] in2=in.getValue(array2);
            double[] result = null;
            //addVector
            if (s.startsWith("addV(") &&
s.endsWith(")")&&s.length()>6){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  Vector res = vop.addV(v1, v2);
                  System.out.println("The sum of the two input vectors
is:");
                  System.out.println( res.toString());
            }
            //subVector
            else if (s.startsWith("subV(") &&
s.endsWith(")")&&s.length()>6){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  Vector res = vop.subV(v1, v2);
                  System.out.println("The difference of the two input
vectors is:");
                  System.out.println( res.toString());
            }
            //dot product
            else if (s.startsWith("dotProduct(") &&
s.endsWith(")")&&s.length()>12){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  double res = vop.dotProduct(v1, v2);
                  System.out.println("The dotProduct of the two input
vectors is:");
                  System.out.println(res);
            }
            //distance
            else if (s.startsWith("distance(") &&
s.endsWith(")")&&s.length()>10){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  double res = vop.distance(v1, v2);
                  System.out.println("The distance of the two input
vectors is:");
                  System.out.println(res);
            }
            //norm
            else if (s.startsWith("norm(") &&
s.endsWith(")")&&s.length()>6){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  double res = vop.norm(v1);
                  System.out.println("The norm of the first input
vectors is:");
                  System.out.println(res);
            }
            //cosine
            else if (s.startsWith("cosine(") &&
s.endsWith(")")&&s.length()>8){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  double res = vop.cosine(v1, v2);
                  System.out.println("The cosine of the two input
vectors is:");
                  System.out.println(res);
            }
            //Cross product
            else if (s.startsWith("crossProduct(") &&
s.endsWith(")")&&s.length()>14){
                  VectorOperation vop = new VectorOperation();
                  Vector v1 = new Vector(in1.length);
                  v1.setArr(in1);
                  Vector v2 = new Vector(in2.length);
                  v2.setArr(in2);

                  Vector res = vop.subV(v1, v2);
                  System.out.println("The Cross Product of the two
input vectors is:");
                  System.out.println( res.toString());
            }
            //matrix add
            else if (s.startsWith("add(") &&
s.endsWith(")")&&s.length()>6){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);
                        Matrix m2 = new Matrix(m2i,m2j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);
                              }
                        }
                        for (int i=1; i<=m2i; ++i){
                              for (int j=1; j<=m2j;++j ){
                                    m2.setValue(i, j, in2[(i-1)*m1j+j-
1]);
                              }
                        }
                        Matrix res = mop.add(m1, m2);

                        System.out.println("The sum of the two input
matrix is:");
                        System.out.println(res.toString());
                  }

            }
            //sub
            else if (s.startsWith("sub(") &&
s.endsWith(")")&&s.length()>6){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);
                        Matrix m2 = new Matrix(m2i,m2j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);
                              }
                        }
                        for (int i=1; i<=m2i; ++i){
                              for (int j=1; j<=m2j;++j ){
                                    m2.setValue(i, j, in2[(i-1)*m1j+j-
1]);
                              }
                        }
                        Matrix res = mop.sub(m1, m2);

                        System.out.println("The difference of the two
input matrix is:");
                        System.out.println(res.toString());
                  }

            }
      //mul
            else if (s.startsWith("mul(") &&
s.endsWith(")")&&s.length()>6){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);
                        Matrix m2 = new Matrix(m2i,m2j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);
                              }
                        }
                        for (int i=1; i<=m2i; ++i){
                              for (int j=1; j<=m2j;++j ){
                                    m2.setValue(i, j, in2[(i-1)*m1j+j-
1]);
                              }
                        }
                        Matrix res = new Matrix(m1i,m2j);

                        if (m1i==m1j&& m2i==m2j && m1i==m2i){
                              res = mop.strassen(m1, m2);
                        }
                        else{
                              res = mop.mul(m1, m2);
                        }
                        System.out.println("The product of the two
input matrix is:");
                        System.out.println(res.toString());
                  }

            }
            //transpose
            else if (s.startsWith("transpose(") &&
s.endsWith(")")&&s.length()>11){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);

                              }
                        }

                        Matrix res = mop.transpose(m1);

                        System.out.println("The transpose of the input
matrix is:");
                        System.out.println(res.toString());
                  }

            }
            //det2 //gasussian
            else if (s.startsWith("det(") &&
s.endsWith(")")&&s.length()>5){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);

                              }
                        }


                        double res = mop.det(m1);

                        System.out.println("The deterninant of the
input matrix is:");
```

```
                        System.out.println(res);
                  }

            }
            //definition det
            else if (s.startsWith("det2(") &&
s.endsWith(")")&&s.length()>6){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);

                              }
                        }


                        double res = mop.determinant(m1);

                        System.out.println("The deterninant of the
input matrix is:");
                        System.out.println(res);
                  }

            }
            //inverse
            else if (s.startsWith("inverse(") &&
s.endsWith(")")&&s.length()>9){
                  if (in1.length != m1i * m1j || in2.length != m2i *
m2j){
                        System.out.println("The dimension and the array
lenth is not match.");
                  }
                  else {
                        MatrixOperation mop = new MatrixOperation();
                        Matrix m1 = new Matrix(m1i,m1j);

                        for (int i=1; i<=m1i; ++i){
                              for (int j=1; j<=m1j;++j ){
                                    m1.setValue(i, j, in1[(i-1)*m1j+j-
1]);

                              }
                        }


                        Matrix res = mop.inverse(m1);

                        System.out.println("The inverse of the input
matrix is:");
                        System.out.println(res.toString());
                        }

                }
                else {
                        System.out.println("Sorry, no such command");
                }

                //

        }
}
```