B.Sc. in Computer Science and Engineering Thesis

A New Algorithm for Solution of System of Linear Equations

Submitted by

Md. Iftekharul Hoque
200914016

Md. Nur-E-Arefin
200914040

Md Saiful Ahmad
200914060

Supervised by

Dr. M. Kaykobad
Professor, Department of CSE, BUET

Department of Computer Science and Engineering
Military Institute of Science and Technology

CERTIFICATION

This thesis paper titled **"A New Algorithm for Solution of System of Linear Equations"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering on December 2012.

Group Members:

Md. Iftekharul Hoque
Md. Nur-E-Arefin
Md Saiful Ahmad

Supervisor:

—————————————-

Dr. M. Kaykobad
Professor, Department of CSE
BUET

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis paper is the outcome of the investigation and research carried out by the following students under the supervision of Dr. M. Kaykobad, Professor, Department of CSE, BUET, Dhaka, Bangladesh.

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

—————————————-

Md. Iftekharul Hoque
200914016

—————————————-

Md. Nur-E-Arefin
200914040

—————————————-

Md Saiful Ahmad
200914060

ACKNOWLEDGEMENT

We are thankful to Almighty Allah for his blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisor Dr. M. Kaykobad, Professor, Department of CSE, BUET, Dhaka, Bangladesh, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing thesis.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

Dhaka
December 2012

Md. Iftekharul Hoque
Md. Nur-E-Arefin
Md Saiful Ahmad

ABSTRACT

Systems of linear equations are used in a variety of fields. The canonical problem of solving a system of linear equations arises in numerous contexts in information theory, communication theory, and related fields. This thesis is aimed at analyzing the available methods for solving a system of linear equations of the form n x n. Using a couple of iterative and/or direct methods, implement a program for these methods that could be run for different dimension size n of system of linear equation . At the end, a graph can be plotted with time taken for execution of a method considered V/s dimension size n. In this contribution, we develop a solution that does not involve direct matrix inversion. The iterative nature of our approach allows for a distributed message-passing implementation of the solution algorithm. We present test results which show that our solver achieves good results, both in terms of numerical accuracy as well as computing time. Furthermore, even very large systems (n $\leq$ 1000) can be solved given a cluster with sufficient resources. We also address some properties of the algorithm, including convergence, exactness, its complexity order and relation to classical solution methods.

Keywords: self-verifying methods, large linear interval systems, indirect methods.

TABLE OF CONTENT

CERTIFICATION **ii**

CANDIDATES' DECLARATION **iii**

ACKNOWLEDGEMENT **iv**

ABSTRACT **v**

List of Figures **viii**

List of Tables **ix**

List of Symbols **x**

1 Introduction **1**

1.1 LITERATURE REVIEW 1

1.2 SCOPE OF THE RESEARCH 2

1.3 OUTLINE OF THE RESEARCH 3

2 METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS **4**

2.1 INTRODUCTION 4

2.2 DIRECT METHODS FOR SOLVING SYSTEM OF LINEAR EQUATIONS 8

2.2.1 GAUSSIAN ELIMINATION METHOD 8

2.2.2 GAUSS-JORDAN ELIMINATION 13

2.2.3 LU DECOMPOSITION 24

2.3 ITERATIVE METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS 32

2.3.1 GAUSS-SEIDEL METHOD 32

2.3.2 THE JACOBI METHOD 36

2.3.3 CONJUGATE GRADIENT METHOD 39

2.3.4 SUCCESSIVE OVERRELAXATION METHOD 41

3 OUR CONTRIBUTION **43**

3.1 INTRODUCTION 43

3.2 A NEW ALGORITHM 43

3.3 FUNCTIONAL DEFINITION OF THE ALGORITHM 44

3.4 TOTAL COMPLEXITY ANALYSIS OF THE ALGORITHM 45

3.5 ANALYTICAL COMPARISON WITH EXISTING ALGORITHM 46

4 EXPERIMENTAL COMPARISON **47**

4.1 EXPERIMENTAL SETUP 47

4.2 EXPERIMENTAL ANALYSIS 47

4.3 DISCUSSION 49

5 RESULT AND RECOMMENDATION **50**

5.1 RESULT 50

5.2 RECOMMENDATION 50

References **52**

Appendix-A **53**

LIST OF FIGURES

4.1 Analysis Graph 48

LIST OF TABLES

4.1 Analysis of Data 48

LIST OF SYMBOLS

£	: Eigen function
λ	: Eigen Value
$\sum$	: Summation Notation
$\leq$	: Less than or equal Notation
[ ]	: Matrix Notation
$\prod$	: Product Notation
O	: Complexity Order Notation

CHAPTER 1
INTRODUCTION

1.1 LITERATURE REVIEW

A system of linear equations means two or more linear equations. (In plain speak: ’two or more lines’) If these two linear equations intersect, that point of intersection is called the solution to the system of linear equations. In mathematics, the theory of linear systems is the basis and a fundamental part of linear algebra, a subject which is used in most parts of modern mathematics. Computational algorithms for finding the solutions are an important part of numerical linear algebra, and play a prominent role in engineering, physics, chemistry, computer science, and economics. A system of non-linear equations can often be approximated by a linear system, a helpful technique when making a mathematical model or computer simulation of a relatively complex system.

There are existing algorithms for system of linear equations, but the main target is how to reduce the computational time as well as complexity. As solving a linear system is the main focus, there are many methods already discovered by the mathematicians. These techniques are Elimination of variables, Row reduction, Gaussian elimination, Cramer’s rule, Matrix solution and Other methods. But in our thesis our main focus was find a technique which consume less time in computing and obviously less complex.

we introduce a new algorithm to solve the system of linear equations where we first Generate solution X* having randomly n components and Generate nn matrix A. Then compute bi. generate initial point Po randomly from where computation starts and construct Pi ? Hi from Pi-1. Generate new initial point Pi’ and calculate distance between newly generated point Pi(t) and initial solution X* and try to minimize this distance.

Solving the system of linear equation by implementing algorithm and from the algorithm we implement code which runs through computer and gives us output so that we can measure the complexity and computation time. Its like simulation. As a computer science and engineering student its so much relevant with our field of study.

In conclusion to find a solution of system of linear equation is not the concern for us. Minimize the time and complexity is our main concern so that a correct solution can be found in less time. So that we can find a less complex solution which can be easily understood.

1.2 SCOPE OF THE RESEARCH

The scope of research is the areas covered in the research. This part of the research paper is telling exactly what was done and where the information that was used specifically came from. To get numerical answer out of any linear model, one must in the end to obtain the solution of a system of linear equations. It is not surprising that to carry out this task efficiently it has engaged the attention of some of the leading mathematicians. Two methods still in current use, Gaussian Elimination and Gauss-Seidel iteration, were devised by the prince of mathematicians. The size and scope of linear equation that could be solved efficiently has been enlarged enormously and the role of linear model correspondingly enhanced. The success of this effort has been due not only to the huge increase in computational speed and in the size of rapid access memory, but in equal measure to new, sophisticated, mathematical methods for solving linear equations.

In the 1940s it is instructive to recall that linear algebra was dead as a subject for research. Yet only a few years later, in response to the opportunities created by the availability of the high speed computers, very fast algorithms were found for the standard matrix operations.

In this thesis paper we introduce a new algorithm to solve the system of linear equations which is minimizes the use of resources both in terms of computing power and memory usage. In this algorithm we assume that system must have solution.

The aim of this thesis paper is to formulate linear system of equations in matrix form and find out the conditions under which solutions exist. Understand how to execute condition to be able to find solutions of system of linear equations with minimum cost.

1.3 OUTLINE OF THE RESEARCH

In this thesis paper chapter one introduce the scope of the thesis and literature review.

Chapter two provides background study of the existing algorithm of solving system of linear equation to understand , implement and test the system.

Chapter three introduce our new proposed algorithm and the functional definition of the algorithm. It also analyze the total complexity of the new algorithm and compares with existing algorithm.

Chapter four give brief description about experimental setup and shows the experimental result of the algorithms.

Chapter five discuss about the result and recommendation.

CHAPTER 2
METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

2.1 INTRODUCTION

Many real life situations are governed by a system of differential equations. Physical problems usually have more than one dependent variable to be considered. Models of such problems can give rise to system of differential equations in which there are two or more dependent variable and one independent variable. Because we are going to be working almost exclusively with systems of equations in which the number of unknowns equals the number of equations we will restrict our discussion to these kinds of systems. Linear equations are equations of the form $a_1x_1 + a_2x_2 + ... + a_nx_n = b$, where $a_1...a_n$ are constants called coefficients, $x_1...x_n$ are variables, and b is the constant term. This equations is a linear equation in n variables. The variables are also often seen as x, y, z, etc. Linear equations involve only variables of degree 1 (no exponents or roots of the variables), and they involve only sums and constant multiples of variables, without any products of variables.

The following equations are linear:

$$2x+3y=7$$

$$3x_1-x_2+4x_3=1$$

$$\Pi(x+y)=z$$

The following equations are not linear:

$$2x_2 = y$$

$$3xy=1$$

y = 1/x

A solution of a linear equation is a vector composed of values [s_1,...,s_n], where when we can substitute $x_1 = s_1$, ..., $x_n = s_n$ to obtain a correct equation. An equation in one variable (example, 2x = 6) has only one solution (a point on the real number line, in this case x = 6). A linear equation in two variables has solutions which geometrically form a line in the plane. A linear equation in three variables has solutions which form a plane. For linear equations of more variables, the geometric interpretations aren't as clear, but they still have an infinite set of solutions[1].

Systems of linear equations are groups of more than one linear equation. Solving the system refers to finding a solution common to all of the linear equations in the systems. There are only three possibilities:

1) The system has one unique solution (a consistent system)

2) The system has infinitely many solutions (also a consistent system)

3) The system has no solutions (an inconsistent system)

Geometrically, one solution can be interpreted geometrically as the point where the various lines, planes, etc. defined by the linear equations intersect. Infinitely many solutions occur when the equations define lines and/or planes that intersect in a line or plane, such as the intersection of two planes or two equal lines[2]. An inconsistent system can be geometrically interpreted as lines and/or planes which never intersect, such as parallel lines in the plane, or parallel lines or planes in 3 dimensions. In 3 dimensions, we can also have skew lines, which are not parallel, but never intersect.

Homogeneous Systems

Homogeneous systems of linear equations are systems where the constant term in each equations is zero. Therefore, they have the form [A/0], where A is the matrix of coefficients of the variables in the system of equations. Systems of this type always have a solution. There is always the trivial solution where [x_1, x_2, ..., x_n] = [0,0,...0]. This can be interpreted as a point at the origin of the space R^n. As well, the system may have infinitely many solutions. Let's look at homogeneous systems.

Solve the following Homogeneous system:

$$3x + 2y + z = 0$$

$$4x - y + z = 0$$

$$-2x + 3y + 2z = 0$$

Since this is homogeneous, we have the solution (x,y,z) =(0,0,0)

$$\begin{bmatrix} 3 & 2 & 1 & | & 0 \\ 4 & -1 & 1 & | & 0 \\ -2 & 3 & 2 & | & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 5 & 3 & | & 0 \\ 0 & 5 & 5 & | & 0 \\ -2 & 3 & 2 & | & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 5 & 3 & | & 0 \\ 0 & 1 & 1 & | & 0 \\ 0 & 13 & 8 & | & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 5 & 3 & | & 0 \\ 0 & 1 & 1 & | & 0 \\ 0 & 13 & 8 & | & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 5 & 3 & | & 0 \\ 0 & 1 & 1 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 5 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 5 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & | & 0 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 0 \end{bmatrix} \rightarrow \begin{array}{rcl} x & = & 0 \\ y & = & 0 \\ z & = & 0 \end{array}$$

Therefore the only solution to this system is (x, y, z) = (0, 0, 0)

In a homogeneous system, if the number of equations is less than the number of variables in the system, the system will always have infinitely many solutions. This is because a homogeneous system has to be consistent, and our formula for free variables guarantees at least one free variable.

NON HOMOGENEOUS SYSTEM OF LINEAR EQUATIONS

In general, the equation AX=B representing a system of equations is called homogeneous if B is the nx1 (column) vector of zeros. Otherwise, the equation is called non homogeneous. For square systems of equations (i.e. those with an equal number of equations and unknowns), the most powerful tool for determining the number of solutions the system has is the determinant. Suppose we have two equations and two unknowns: ax+by=c and dx+ey=f with b and e non-zero (i.e. the system is non homogeneous). These are two lines with slope -a/b and -d/e, respectively. Let's define the determinant of a 2x2 system of linear equations to be the determinant of the matrix of coefficients A of the system. For this system

$$A = \begin{pmatrix} a & b \\ d & e \end{pmatrix} \text{ and } |a| = \begin{vmatrix} a & b \\ d & e \end{vmatrix} = ae - bd$$

Suppose this determinant is zero. Then, this last equation implies a/b=d/e; in other words, the slopes are equal. From our discussion above, this means the lines are either identical (there is an infinite number of solutions) or parallel (there are no solutions). If the determinant is non-zero, then the slopes must be different and the lines must intersect in exactly one point. This leads us to the following result:

A nxn non homogeneous system of linear equations has a unique non-trivial solution if and only if its determinant is non-zero. If this determinant is zero, then the system has either no nontrivial solutions or an infinite number of solutions. **Example**
The non homogeneous system of equations 2x+3y=-8 and -x+5y=1 has determinant

$$\begin{bmatrix} 2 & 3 \\ -1 & 5 \end{bmatrix} = 2(5) - 3(-1) = 13$$

es the system has a unique solution (the point (-43/13,-28/75)).

Linear Independence

A set of vectors is said to be linearly dependent if each vector in the set can be expressed as a linear combination of the other ones. This is equivalent to saying that there is some nontrivial linear combination (not all scalars are zero) of this set of vectors that equals 0. If the vectors are not linearly dependent, they are otherwise called linearly independent. Solving questions involving linear dependence involves putting our vectors into an augmented system, and then solving that system. Some examples are below. We can see from above that if a system of linear equations is put in augmented form $[A|b]$, then it has a solution if b is a linear combination of columns of A. Also, if we create a matrix B, where the rows of B are vectors in row form, these vectors are linearly dependent if (if and only if) Rank(B)$\leq$m, where m is the number of vectors. Let's take our above examples and show how this works. Finally, any set of vectors in R_n is linearly dependent if the number of vectors (m) is greater than the dimension of the space (n), that is, m$\geq$n.

Spanning Sets

Closely linked to the idea of linear dependence is the idea of spanning sets. Say we have a set of vectors in R^n, where S = $\{v_1, v_2, \ldots, v_k\}$. The set of all possible linear combinations of the vectors in this set S is called the span of S. It is denoted span $(v_1, v_2, \ldots, v_k)$ or span(S). If span(S) = R^n, then S is called a spanning set for R^n. S can also span subsets of the whole space, such as a line, plane, etc.

2.2 DIRECT METHODS FOR SOLVING SYSTEM OF LINEAR EQUATIONS

2.2.1 GAUSSIAN ELIMINATION METHOD

Back substitution

Back substitution is the simplest method of finding the solution to the system, and is also the last step in the more powerful methods of solving systems that we see later. Back substitution involves isolating a single variable, and substituting it back into another equation to isolate another variable, and continuing until we have solved for all variables. To use back substitution, we need a system like above where the a variable is already isolated, or where we can express one variable in terms of another, as in the following example. Now if we solve the following system by substitution:

$$2x + 3y - z = 9$$

$$2y - z = 2$$

$$3z = 12$$

We first solve for z by dividing both sides of the third equation by the constant multiple 3:

$$3z = 12$$

$$z = 4$$

We can now substitute this into the second equation and solve for y:

$$2y - z = 2$$

$$2y - 4 = 2$$

$$2y = 6$$

$$y = 3$$

Finally we can substitute y and z into the first equation and solve for x:

$$2x + 3y - z = 9$$

$$2x + 3(3) - (4) = 9$$

$$2x = 4x = 2$$

so the solution to our system is the point (x , y, z) = (2, 3, 4)

It is not often that our system of equations is already in a state where we can proceed directly to back substitution. We need to get our system of equations to a point where we have isolated variables. We can do this using elementary row operations. Elementary row operations can be used to change the system to one that we can use back substitution on, without changing the solution. If we denote row 1 by R_1, row 2 by R_2, etc., and k is a scalar, the three elementary row operations are as follows:

1)Swap two rows (equations), denoted $R1 \leftrightarrow R2$ (this would swap rows 1 and 2)

2)Multiply a row by a nonzero scalar, denoted kR_1 (this would multiply row 1 by k)

3)Add a multiple of a row to another row, denoted $R_1 + kR_2$ (this would add k times row 2 to row 1, and replace the previous row 1)

Elementary row operations should be familiar from solving systems in high school math courses. By introducing the concept of an augmented matrix, we can simplify our work on the system, and combine this with elementary row operations and back substitution to define another method of solving systems.

An augmented matrix is a matrix which contains the coefficients and constant terms from our system of linear equations. By putting the systems in this matrix, we only need to concentrate on what is important, these constants, since the variables themselves are never operated on.

GAUSSIAN ELIMINATION

Gaussian elimination is a three step method for solving systems of linear equations:

1) We will write out the system as an augmented matrix (we will use zeroes for a particular variable of the system if it doesn't appear in one of the equations)
2) We will use elementary row operations to reduce the matrix to row echelon form (see below)
3) We will use back substitution to finally solve the system

Row echelon form occurs when a matrix is in a form as follows:

1) The first entry in each row (called the leading entry) has only zeroes in the column below it.
2) When a row has a leading entry, the leading entry in all rows above it are in columns to the left of the leading entry .
The following matrices are in row echelon form:

$$\begin{bmatrix} 2 & 1 & 3 & | & 11 \\ 0 & 3 & 1 & | & 8 \\ 0 & 0 & 4 & | & 8 \end{bmatrix}$$

Let’s take the system below and perform the steps involved in Gaussian elimination.

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

$$3x + 2y - z = 7$$

$$x + 4y - 5z = 3$$

After putting the system into an augmented matrix, we perform elementary row operations on this system until we get it into row echelon form as described above. This allows us to perform back substitution, beginning with the bottom row in the system, in order to find the solution. In this way, Gaussian elimination provides us with an algorithm of sorts to solve a system of linear equations. Below, we solve the above system using Gaussian elimination.

Solve the following system using Gaussian elimination :

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

$$3x + 2y - z = 7$$

$$x + 4y - 5z = 3$$

First, we put this system into an augmented matrix and label the rows.

$$\begin{array}{cc} \begin{bmatrix} 2 & -3 & 1 & | & -5 \\ 3 & 2 & -1 & | & 7 \\ 1 & 4 & -5 & | & 3 \end{bmatrix} & \begin{array}{c} R1 \\ R2 \\ R3 \end{array} \end{array}$$

Then we can use elementary row operations to reduce the matrix to row echelon form.

$$\begin{bmatrix} 2 & -3 & 1 & | & -5 \\ 3 & 2 & -1 & | & 7 \\ 1 & 4 & -5 & | & 3 \end{bmatrix} \to \begin{bmatrix} 2 & -3 & 1 & | & -5 \\ 0 & -10 & 14 & | & -2 \\ 0 & 11 & -11 & | & 11 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 2 & -3 & 1 & | & -5 \\ 0 & -10 & 14 & | & -2 \\ 0 & 0 & 4 & | & 8 \end{bmatrix}$$

We now put this row echelon form matrix back into equation form and use back substitution to solve the system:

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

$$-10y + 14z = -2$$

$$4z = 8$$

$$z=2$$

so

$$-10y + 28 = -2$$

$$-10y = -30$$

$$y = 3$$

$$2x - 9 + 2 = -5$$

$$2x = 2$$

$$x = 1$$

so the solution of our system is the point of intersection$(x, y, z) = (1, 2, 3)$

COMPLEXITY

We find that the number of operations required to perform Gaussian elimination is ON^3, where N is the number of equations. For large systems of equations Gaussian elimination is very inefficient even for the fastest super computer! On the other hand, the number of operations required for back-substitution scales with ON^2. Back substitution is much more efficient for large systems than Gaussian elimination.

2.2.2 GAUSS-JORDAN ELIMINATION

INTRODUCTION

In linear algebra, GaussJordan elimination is an algorithm for getting matrices in reduced row echelon form using elementary row operations. It is a variation of Gaussian elimination. Gaussian elimination places zeros below each pivot in the matrix, starting with the top row and working downwards. Matrices containing zeros below each pivot are said to be in row echelon form. GaussJordan elimination goes a step further by placing zeros above and below each pivot; such matrices are said to be in reduced row echelon form. Every matrix has a reduced row echelon form, and GaussJordan elimination is guaranteed to find it.

Consider the following system of linear equations:

$$x_1 - x_2 + 2x_3 = 3$$

$$2x_1 - 2x_2 + 5x_3 = 4$$

$$x_1 + 2x_2 - x_3 = -3$$

$$2x_2 + 2x_3 = 1$$

To use Gauss-Jordan Elimination, we start by representing the given system of linear equations as an augmented matrix.

Augmented Matrix Form

An augmented matrix is a matrix representation of a system of linear equations where each row of the matrix is the coefficients of the given equation and the equation's result.

Example

The following system of linear equations:

$$x_1 - x_2 + 2x_3 = 3$$

$$2x_1 - 2x_2 + 5x_3 = 4$$

$$x_1 + 2x_2 - x_3 = -3$$

$$2x_2 + 2x_3 = 1$$

would be represented as:

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

The goal of Gauss-Jordan elimination is to transform the matrix into reduced echelon form.

Reduced Echelon Form

A matrix is said to be in reduced echelon form if:

(1) Any rows consisting entirely of zeros are grouped at the bottom of the matrix.

(2) The first nonzero element of any row is 1. This element is called a leading 1.

(3) The leading 1 of each row after the first is positioned to the right of the leading 1 of the previous row.

(4) If a column contains a leading 1, then all other elements in that column are 0.
Here is a examples of matrices that are not in reduced echelon form.

The matrix below violates (1). The rows consisting entirely of 0's are not grouped at the bottom.

$$\begin{bmatrix} 1 & 2 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 8 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

The Matrix above is a examples of matrices that are in reduced echelon form.
Next, we need to use elementary row operations so the matrix results in reduced echelon form. The elementary row operations are operations that the result matrix represents the same system of linear equations. In other words, the two matrices are row equivalent.

Row Equivalence (Equivalent Systems)

Two matrices are row equivalent if they represent the same system of linear equations. They are also called equivalent systems.

The following three elementary operations preserve row equivalence. That is, the resulting matrix is row equivalent to the matrix before the operation[9].

(1) Interchanging two rows
(2) Multiplying the elements of a row by a nonzero constant
(3) Adding the elements of one row to the corresponding elements of another row.

(1) is changing the order of the linear equations does not change the values that solve them. (2) is equivalent to multiplying the same value to both sides of the equation. (3) is the

idea that two linear equations imply that their sum is also a valid linear equation. In actual practice, we will combine (2) and (3) to get: (4) Add a multiple of the elements of one row to the corresponding elements of another row.

So, the following elementary operations result in a matrix that is row equivalent to the previous matrix:

Elementary Operations

(1) Row Exchange: The exchange of two rows.
(2) Row Scaling: Multiply the elements of a row by a nonzero constant.
(3) Row Replacement: Add a multiple of the elements of one row to the corresponding elements of another row.

Once the matrix is in reduced echelon form, we can convert the matrix back into a linear system of equations to see the solution.

Here's the full algorithm: **Gauss-Jordan Elimination** (1) Represent the linear system of equations as a matrix in augmented matrix form

(2) Use elementary row operations to derive a matrix in reduced echelon form

(3) Write the system of linear equations corresponding to the reduced echelon form.

Gauss-Jordan Elimination Algorithm

(1) Let $A = m \times n$ matrix with m rows and n columns

(2) Let $k = 1, A_k = A, i = 1$

(3) If Ak is all zero's, then we are done and the A is in reduced echelon form

(4) Let C_v be the first nonzero column in Ak such that C_v is not all zero's and $v?i$

(5) Let $a_{u,v}$ be the first nonzero element in C_v such that $u?k$

(6) Let Ru be the row in A where au,v is found in C_v

(7) $R_u = R_u \times (1/a_{u,v})$ [After this, R_u has a leading 1, see proof below for explanation]

(8) If $u \neq k$, then exchange R_u and R_k

(9) For each row R_w in A, if $w?k$, then $R_w = R_w + (-a_{w,v}) \times R_k$[After this, the only nonzero element in C_v is found at $a_{k,v}$, see proof below for explanation if needed]

(10) If $k = m$, then we are done and A is in reduced echelon form

(11) Otherwise, we designate a portion of A_k such that:

$A_k = \frac{R_k}{A_{k+1}}$ [Note: In other words, A_{k+1} =all but the top row of A_k]

(12) Let $k = k+1; i = v$; and go to step 3

End

Here's an example:

Example 2: Using Gauss-Jordan Elimination

(1) We start with a system of linear equations:

$$
\begin{aligned}
x_1 - 2x_2 + 4x_3 &= 12 \\
2x_1 - x_2 + 5x_3 &= 18 \\
-x_1 + 3x_2 - 3x_3 &= -8
\end{aligned}
$$

(2) We represent them in augmented matrix form:

$$
\begin{bmatrix}
1 & -2 & 4 & 12 \\
2 & -1 & 5 & 18 \\
-1 & 3 & -3 & -8
\end{bmatrix}
$$

(3) We use elementary row operations to put this matrix into reduced echelon form (I will use R1, R2, R3 for Row 1, Row 2, and Row 3):

(a) Let $R_2 = R_2 + (-2)R_1$ [Operation 3]

$$
\begin{bmatrix}
1 & -2 & 4 & 12 \\
0 & 3 & -3 & -6 \\
-1 & 3 & -3 & -8
\end{bmatrix}
$$

(b) Let $R_3 = R_3 + R_1$ [Operation 3]

$$\begin{bmatrix} 1 & -2 & 4 & 12 \\ 0 & 3 & 3 & -6 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

(c) Let $R_2 = (1/3)R_2$ [Operation 2]

$$\begin{bmatrix} 1 & -2 & 4 & 12 \\ 0 & 1 & -1 & -2 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

(d) Let $R_1 = R_1 + 2 \times R_2$ [Operation 3]

$$\begin{bmatrix} 1 & 0 & 2 & 8 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 2 & 6 \end{bmatrix}$$

(e) Let $R_3 = R_3 + (-1) \times R_2$ [Operation 3]

$$\begin{bmatrix} 1 & 0 & 2 & 8 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 2 & 6 \end{bmatrix}$$

(f) Let $R_3 = (1/2)R_3$ [Operation 2]

$$\begin{bmatrix} 1 & 0 & 2 & 8 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 3 & 3 \end{bmatrix}$$

(g) Let $R_1 = R_1 + (-2) \times R_3$ [Operation 3]

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

(h) Let $R_2 = R_2 + R_3$ [Operation 3]

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

(4)We now write down the system of linear equations corresponding to the reduced echelon form:

$$x_1 = 2$$
$$x_2 = 1$$
$$x_3 = 3$$

REQUIREMENTS FOR A UNIQUE SOLUTION TO A SYSTEM OF LINEAR EQUATIONS USING THE GAUSS-JORDAN ELIMINATION METHOD

The requirements for a unique solution to a system of linear equations using the Gauss-Jordan Elimination Method are that the number of unknowns must equal the number of equations.

When the number of equations and the number of unknowns are the same,we will obtain an augmented matrix where the number of columns is equal to the number of rows plus 1.

For example, if we have a system of 4 linear equations in 4 unknowns, then the number of rows must be equal to 4 and the number of columns must be equal to 5 (4 columns for the coefficients and 1 column for the results).

The vertical line between the coefficients part of the matrix and the results part of the matrix (the constant terms are in the results part of the matrix) does not count as a column. It's there for display purposes only if you have the capability to display it.
When we create a matrix, just make sure that the number of columns is equal to the number of rows plus 1.

It is possible to get a unique solution to a system of linear equations where the number of equations is greater than the number of unknowns.

An example of this would be 5 lines in a plane all intersecting in the same point. There is a unique solution for the x and y variables that makes all the equations in the system true. This type of situation, however, is not conducive to solving using the Gauss-Jordan Elimination Method since that method requires the number of equations and the number of unknowns to be the same.

It is not possible to get a unique solution to a system of linear equations where the number of equations is less than the number of unknowns.

If we want to use the Gauss-Jordan Elimination Method, make sure that the number of equations is equal to the number of unknowns and the method will work just fine.

CAUSES OF NOT BEING ABLE TO FIND A UNIQUE SOLUTION TO A SYSTEM OF LINEAR EQUATIONS

With Linear Equations in 2 dimensions, the causes are clear. The lines represented by the equations are either parallel (no solution), or intersecting in a point (unique solution), or identical (infinite number of solutions). There is no in between. With Linear Equations in 3 dimensions, the causes get a little muddier. The possible causes are:
1. All 3 planes are parallel to each other. There are no points of intersection.This system would be classified as inconsistent.
2. All 3 planes are identical to each other.There are an infinite number of points of intersection.This system would be classified as dependent.
3. All 3 planes intersect with each other in a point. There is one unique point of intersection.This system would be classified as independent.
4. Two planes are parallel and the third plane intersects with each of them in a line.This system does not a unique point of intersection.Two of the equations are inconsistent with

each other.The third equation is independent of the other two because it does intersect with each of them only the intersection is in a line and not a point[12].

5. Two planes are identical to each other and the third plane intersects with each of them in the same line.Two of the equations are dependent on each other (they are identical).The third plane is independent of the other two because it does intersect with each of them only the intersection is in a line and not a point.

$$2x-3y-z=0$$
$$3x+2y+2z=2$$
$$x+5y+3z=2$$

The augmented matrix is

$$[2\ -3\ -1\ |\ 0]$$
$$[3\ \ 2\ \ 2\ \ |\ \ 2]$$
$$[1\ \ 5\ \ 3\ \ |\ \ 2]$$

We need to get 0's in the three lower left positions, that is, in the positions below the upper left to lower-right diagonal:

To get a 0 where the 3 in row 2 column 1 is,we multiply row 1 temporarily by -3 and add it to 2 times row 2. This is easy to do if we write -3 to the left of row 1 and 2 left of row 2:

$$-3[2\ -3\ -1\ |\ 0]$$
$$-2[3\ \ 2\ \ 2\ \ |\ \ 2]$$
$$[1\ \ 5\ \ 3\ \ |\ \ 2]$$
$$[2\ -3\ -1\ |\ 0]$$
$$[0\ \ 13\ \ 7\ \ |\ \ 4]$$
$$[1\ \ 5\ \ 3\ \ |\ \ 2]$$

To get a 0 where the 1 in row 3 column 1 is, we multiply row 1 temporarily by -1 and add it to 2 times row 3. This is easy to do if we write -1 to the left of row 1 and 2 left of row 3:

$$-1[2\ -3\ -1\ |\ 0]$$

$$[0 \quad 13 \quad 7 \quad | \quad 4]$$

$$2[1 \quad 5 \quad 3 \quad | \quad 2]$$

$$[2 \ -3 \ -1 \ | \ 0]$$

$$[0 \quad 13 \quad 7 \quad | \quad 4]$$

$$[0 \quad 13 \quad 7 \quad | \quad 4]$$

To get a 0 where the 13 in row 3 column 2 is, we multiply row 2 temporarily by -1 and add it to 1 times row 3. This is easy to do if we write -1 to the left of row 2 and 1 left of row 3:

$$-1[2 \ -3 \ -1 \ | \ 0]$$

$$[0 \quad 13 \quad 7 \quad | \quad 4]$$

$$2[0 \quad 13 \quad 7 \quad | \quad 4]$$

$$[2 \ -3 \ -1 \ | \ 0]$$

$$[0 \quad 13 \quad 7 \quad | \quad 4]$$

$$[0 \quad 0 \quad 0 \quad | \quad 0]$$

Now that we have the three zeros, the above is the abbreviation of this system:

$$2x - 3y - 1z = 0$$

$$0x + 13y + 7z = 4$$

$$0x + 0y + 0z = 0$$

or

$$2x - 3y - z = 0$$

$$13y + 7z = 4$$

$$0z = 0$$

The bottom equation,

$$0z = 0$$

has as a solution “can equal to any real number”

So we can select any letter to stand for any real number, say the letter k[13].

So

$$z = k$$

We substitute k for z into the middle equation

$$13y + 7z = 4$$
$$13y + 7k = 4$$
$$13y = 4 - 7k$$
$$y = \frac{4}{13} - \frac{7}{13}k$$

We substitute k for z and $\frac{4}{13} - \frac{7}{13}k$ for y into the top equation:

$$2x - 3y - z = 0$$
$$2x - 3(\frac{4}{13} - \frac{7}{13}k) - k = 0$$
$$2x - (\frac{12}{13} + \frac{21}{13}k) - k = 0$$

Clear of fractions by multiplying through by 13:

$$26x - 12 + 21k - 13k = 0$$
$$26x - 12 + 8k = 0$$
$$26x = 12 - 8k$$
$$x = \frac{12}{26} - \frac{8}{26}k$$
$$x = \frac{6}{13} - \frac{4}{13}k$$

Solution:

$$(x, y, z) = (\frac{16}{13} - \frac{4}{13}k, \frac{4}{13} - \frac{7}{13}k, k)$$

Complexity

Gauss-Jordan elimination, like Gaussian elimination, is used for inverting matrices and solving systems of linear equations. Both Gauss-Jordan and Gaussian elimination have time complexity of order $O(n^3)$ for an n by n full rank matrix (using Big O Notation), but the order of magnitude of the number of arithmetic operations (there are roughly the same number of additions and multiplications/divisions) used in solving a n by n matrix by Gauss-Jordan elimination is n^3, whereas that for Gaussian elimination is $\frac{2n^3}{3}$. Hence, Gauss-Jordan elimination requires approximately 50 percent more computation steps.However, the result of Gauss-Jordan elimination (reduced row echelon form) may be retrieved from the result of Gaussian elimination (row echelon form) in $O(n^2)$ arithmetic operations, by proceeding from the last pivot to the first one. Thus the needed number of operations has the same order of magnitude for both eliminations.

2.2.3 LU DECOMPOSITION

Introduction

An efficient procedure for solving $B = A.X$ is the LU-decomposition. While other methods such as Gaussian elimination method and Cholesky method can do the job well, this LU-decomposition method can help accelerate the computation. The LU-decomposition method first "decomposes" matrix A into $A = L.U$, where L and U are lower triangular and upper triangular matrices, respectively. More precisely, if A is a *nn* matrix, L and U are also *nn* matrices with forms like the following:

$$L=\begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & & 0 \\ \vdots & & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \dots & l_{n,n} \end{bmatrix} \qquad U=\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{l,n} \\ 0 & u_{2,2} & u_{2,3} & & u_{2,n} \\ 0 & 0 & u_{3,3} & & u_{3,n} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{n,n} \end{bmatrix}$$

The lower triangular matrix L has zeros in all entries above its diagonal and the upper triangular matrix U has zeros in all entries below its diagonal. If the LU-decomposition of $A = L.U$ is found, the original equation becomes $B = (L.U).X$. This equation can be rewritten as $B = L.(U.X)$. Since L and B are known, solving for $B = L.Y$ gives $Y = U.X$. Then, since U and Y are known, solving for X from $Y = U.X$ yields the desired result[15]. In this way, the original problem of solving for X from $B = A.X$ is decomposed into two steps:

1. Solving for Y from $B = L.Y$
2. Solving for X from $Y = U.X$

Forward Substitution

It turns out to be very easy. Consider the first step. Expanding $B = L.Y gives$

$$\begin{bmatrix} b_{1,1} & 0 & \cdots & 0 \\ b_{2,1} & b_{2,2} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & l_{n,n} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & & 0 \\ \vdots & & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \dots & l_{n,n} \end{bmatrix} \cdot \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,h} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,h} \\ \vdots & & \ddots & \vdots \\ y_{n,1} & y_{n,2} & \dots & l_{n,h} \end{bmatrix}$$

It is not difficult to verify that column j of matrix B is the product of matrix A and column j of matrix Y. Therefore, we can solve one column of Y at a time. This is shown below:

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & & 0 \\ \vdots & & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \dots & l_{n,n} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

This equation is equivalent to the following:

$$\begin{aligned} b_1 &= l_{1,1}y_1 \\ b_2 &= l_{2,1}y_1 + l_{2,2}y_2 \\ \vdots & \quad \vdots \qquad\qquad\qquad \ddots \\ b_n &= l_{n,1}y_1 + l_{n,2}y_2 + l_{n,3}y_3 + \cdots + l_{n,n}y_n \end{aligned}$$

From the above equations, we see that $y_1 = b_1/l_{11}$. Once we have y_1 available, the second equation yields $y_2 = (b_2 - l_{21}y_1)/l_{22}$. Now we have y_1 and y_2, from equation 3, we have $y_3 = (b_3 - (l_{31}y_1 + l_{32}y_2)/l_{33}$. Thus, we compute y_1 from the first equation and substitute it into the second to compute y_2. Once y_1 and y_2 are available, they are substituted into the third equation to solve for y_3. Repeating this process, when we reach equation i, we will have $y_1, y_2, ..., y_{i-1}$ available[16]. Then, they are substituted into equation i to solve for y_i using the formula:

$$y_i = 1/l_{ii}\left[b_i - \sum_{k=1}^{i} l_{i,k} y_k\right]$$

Because the values of the y_i's are substituted to solve for the next value of y, this process is referred to as forward substitution. We can repeat the forward substitution process for each column of Y and its corresponding column of B. The result is the solution Y. The following is an algorithm:

Input: Matrix B_{nXh} and a lower triangular matrix L_{nXh}

Output: Matrix Y_{nXh} such that B = L.Y holds.

Algorithm:

```
/* there are h columns */
for j := 1 to h do
/* do the following for each column */
begin
/* compute $y_1$ of the current column */
$y_{1,j} = b_{1,j}/l_{1,1}$;
for i := 2 to n do
/* process elements on that column */
begin
sum := 0;
/* solving for $y_i$ of the current column */
for k := 1 to i-1 do
sum := sum + $l_{i,k} * y_{k,j}$;
```

```
$y_{1,j} = (b_{1,j} - sum)/l_{i,i}$;
end
end
```

Backward Substitution

After Y becomes available, we can solve for X from Y = U.X. Expanding this equation and only considering a particular column of Y and the corresponding column of X yields the following:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & & u_{2,n} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & u_{n,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

This equation is equivalent to the following:

$$y_1 = u_{1,1}x_1 + u_{1,2}x_2 + u_{1,3}x_3 + \cdots + u_{1,n}x_n$$
$$y_2 = u_{2,2}x_2 + u_{2,3}x_3 + \cdots + u_{2,n}x_n$$
$$\vdots$$
$$y_n = u_{n,n}x_n$$

Now, x_n is immediately available from equation n, because $x_n = y_n/u_{n,n}$. Once x_n is available, plugging it into equation n-1

$$y_{n-1} = u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n$$

and solving for x_{n-1} yields $x_{n-1} = (y_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1}$. Now, we have x_n and x_{n-1}. Plugging them into equation n-2

$$y_{n-2} = u_{n-2,n-2}x_{n-2} + u_{n-2,n-1}x_{n-1} + u_{n-2}x_n$$

and solving for x_{n-2} yields $x_{n-2} = [y_{n-2} - (u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n)]/u_{n-2,n-2}$.

From x_n, x_{n-1} and x_{n-2}, we can solve for x_{n-3} from equation n-3. In general, after x_n, x_{n-1}, ..., x_{i+1} become available, we can solve for x_i from equation i using the following relation:

$$x_i = 1/u_{i,i}\left[y_i - \sum_{k=i+1}^{n} u_{i,k}x_k\right]$$

Repeat this process until x_1 is computed. Then, all unknown x's are available and the system of linear equations is solved. The following algorithm summarizes this process:

Input: Matrix Y_{nXh} and a upper triangular matrix U_{nXh}

Output: Matrix X_{nXh} such that Y = U.X holds.

Algorithm:

```
/* there are h columns */
for j := 1 to h do
/* do the following for each column */
begin
/* compute $x_n$ of the current column */
$x_{n,j} = y_{n,j}/u_{n,n}$;
for i := n-1 downto 1 do
/* process elements of that column */
begin
sum := 0;
/* solving for $x_i$ on the current column */
for k := i+1 to n do
sum := sum + $u_{i,k} * x_{k,j}$;
$x_{i,j} = (y_{i,j} - sum)/u_{i,i}$;
end
end
```

This time we work backward, from x_n backward to x_1, and, hence, this process is referred to as backward substitution.

Applications

Solving linear equations

Given a system of linear equations in matrix form

Ax = b ,

we want to solve the equation for x given A and b. Suppose we have already obtained the LUP decomposition of A such that PA = LU, (or the LU composition if one exists, in which case P = I) we can rewrite the equation equivalently as

LUx = Pb .

In this case the solution is done in two logical steps:

First, we solve the equation Ly = Pb for y; Second, we solve the equation Ux = y for x.

Note that in both cases we have dealing with triangular matrices (L and U) which can be solved directly by forward and backward substitution without using the Gaussian elimination process (however we do need this process or equivalent to compute the LU decomposition itself).

The above procedure can be repeatedly applied to solve the equation multiple times for different b. In this case it is faster (and more convenient) to do an LU decomposition of the matrix A once and then solve the triangular matrices for the different b, rather than using Gaussian elimination each time. The matrices L and U could be thought to have ”encoded” the Gaussian elimination process.

The cost of solving a system of linear equations is approximately $\frac{2}{3}n^3$ floating point operations if the matrix A has size n . This makes it twice as fast as algorithms based on the QR decomposition, which costs about $\frac{4}{3}n^3$ floating point operations when Householder reflections is used. For this reason, the LU decomposition is usually preferred.

Example

We factorize the following 2-by-2 matrix:

$$\begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}.$$

One way to find the LU decomposition of this simple matrix would be to simply solve the linear equations by inspection. Expanding the matrix multiplication gives

$$l_{11} \cdot u_{11} + 0 \cdot 0 = 4$$

$$l_{11} \cdot u_{12} + 0 \cdot u_{22} = 3$$

$$l_{21} \cdot u_{11} + l_{22} \cdot 0 = 6$$

$$l_{21} \cdot u_{12} + l_{22} \cdot u_{22} = 3.$$

This system of equations is underdetermined. In this case any two non-zero elements of L and U matrices are parameters of the solution and can be set arbitrarily to any non-zero value. Therefore to find the unique LU decomposition, it is necessary to put some restriction on L and U matrices. For example, we can conveniently require the lower triangular matrix L to be a unit one (i.e. set all the entries of its main diagonal to ones). Then the system of equations has the following solution:

$$l_{21} = 1.5$$

$$u_{11} = 4$$

$$u_{12} = 3$$

$$u_{22} = -1.5.$$

Substituting these values into the LU decomposition above yields

$$\begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & -1.5 \end{bmatrix}.$$

Inverting a matrix

When solving systems of equations, b is usually treated as a vector with a length equal to the height of matrix A. Instead of vector b, we have matrix B, where B is an n-by-p matrix, so that we are trying to find a matrix X (also a n-by-p matrix):

AX = LUX = B.

We can use the same algorithm presented earlier to solve for each column of matrix X. Now suppose that B is the identity matrix of size n. It would follow that the result X must be the inverse of A.[12]

Computing the determinant

Given the LU decomposition $A = P^{-1}LU$ of a square matrix A, the determinant of A can be computed straightforwardly as

$$\det(A) = \det(P^{-1})\det(L)\det(U) = (-1)^S \left(\prod_{i=1}^{n} l_{ii}\right)\left(\prod_{i=1}^{n} u_{ii}\right).$$

The second equation follows from the fact that the determinant of a triangular matrix is simply the product of its diagonal entries, and that the determinant of a permutation matrix is equal to (-1)S where S is the number of row exchanges in the decomposition.

The same method readily applies to LU decomposition by setting P to the identity matrix.

Existence and uniqueness

Square matrices

Any square matrix A admits an LUP factorization.[2] If A is invertible, then it admits an LU (or LDU) factorization if and only if all its leading principal minors are non-zero.[5] If A is a singular matrix of rank k , then it admits an LU factorization if the first k leading principal minors are non-zero, although the converse is not true.[6]

If a square, invertible matrix has an LDU factorization, then it is unique.[5] In that case, the LU factorization is also unique if we require that the diagonal of L(or U) consists of ones. Symmetric positive definite matrices

Symmetric positive definite matrices

If A is a symmetric (or Hermitian, if A is complex) positive definite matrix, we can arrange matters so that U is the conjugate transpose of L. That is, we can write A as

$$A = LL^*.$$

This decomposition is called the Cholesky decomposition. The Cholesky decomposition always exists and is unique. Furthermore, computing the Cholesky decomposition is more efficient and numerically more stable than computing some other LU decompositions. General matrices

General matrices

For a (not necessarily invertible) matrix over any field, the exact necessary and sufficient conditions under which it has an LU factorization are known. The conditions are expressed in terms of the ranks of certain submatrices. The Gaussian elimination algorithm for obtaining LU decomposition has also been extended to this most general case.

Sparse matrix decomposition

Special algorithms have been developed for factorizing large sparse matrices. These algorithms attempt to find sparse factors L and U. Ideally, the cost of computation is determined by the number of nonzero entries, rather than by the size of the matrix.

These algorithms use the freedom to exchange rows and columns to minimize fill-in (entries which change from an initial zero to a non-zero value during the execution of an algorithm).

General treatment of orderings that minimize fill-in can be addressed using graph theory

Complexity of LU Decomposition

to solve Ax=b:

decompose A into LU – cost 2n3/3 flops

solve Ly=b for y by forw. substitution – cost n2 flops

solve Ux=y for x by back substitution – cost n2 flops

slower alternative:

compute A-1 – cost 2n3 flops

multiply x=A-1b – cost 2n2 flops

this costs about 3 times as much as LU

2.3 ITERATIVE METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

2.3.1 GAUSS-SEIDEL METHOD

In certain cases, such as when a system of equations is large, iterative methods of solving equations are more advantageous. Elimination methods, such as Gaussian elimination, are prone to large round-off errors for a large set of equations. Iterative methods, such as the Gauss-Seidel method, give the user control of the round-off error.Also, if the physics of the problem are well known, initial guesses needed in iterative methods can be made more judiciously leading to faster convergence[20].Gauss-Seidel method is an iterative method used to solve linear systems of equations. Given a system of linear equations

AX = b

Where A is a square matrix, X is vector of unknowns and b is vector of right hand side values. Suppose we have a set of n equations and n unknowns in the following form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1,n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2,n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{n,n}x_n = b_n$$

We can rewrite each equation for solving corresponding unknowns. We can rewrite equation in generalized form. After that we have to choose initial guess to start Gauss Seidel method then substitute the solution in the above equation and use the most recent value. Iteration is continued until the relative approximate error is less than pre-specified tolerance. Convergence is only guaranteed in case of Gauss Seidel method if matrix A is diagonally dominant. Matrix is said to be diagonally dominant if the absolute value of diagonal element in each row has been greater than or equal to summation of absolute values of rest of elements of that particular row.

The iterative process is terminated when a convergence criterion is fulfilled. We can end up the computation when the difference between two successive iterations is less than the pre-specified tolerance.

Advantageously, Gauss Seidel method is very effective concerning computer storage and time requirements. It is automatically adjusting to if error is made. It possesses less memory when programmed. It is fast and simple to use when coefficient matrix is sparse. It starts with an approximate answer. In each iteration accuracy is improved. It has problem that it may not converge sometime even done correctly. Another drawback of Gauss Seidel method is that it is not applicable for non-square matrices. Non-square matrices are converted into square matrices by taking pseudo inverse of the matrix. It is necessary for the Gauss Seidel method to have non zero elements on the diagonals.

$$10x_1+2x_2+x_3=13$$

$$2x_1+10x_2+x_3=13$$

$$2x_1+x_2+10x_3=13$$

It is The first step is to solve the first equation for x_1, the second for x_2, and the third for x_3 when the system becomes:

$$x_1 = 1.3-.2x_2-.1x_3 \text{ (1)}$$

$$x_2 = 1.3-.2x_1-.1x_3 \text{ (2)}$$

$$x_3 = 1.3-.2x_1-.1x_2 \text{ (3).}$$

An initial solution is now assumed; we shall use $x_1 = 0, x_2 = 0$ and $x_3 = 0$. Inserting these values into the right-hand side of Equation (1) yields $x_1 = 1.3$. This value for x_1 is used immediately together with the remainder of the initial solution (i.e., $x_2 = 0$ and $x_3 = 0$) in the right-hand side of Equation (2) and yields $x_2 = 1.3-0.2x1.3-0 = 1.04$. Finally, the values $x_1 = 1.3$ and $x_2 = 1.04$ are inserted into Equation (3) to yield $x_3 = 0.936$. This second approximate solution (1.3, 1.04, 0.936) completes the first iteration. Beginning with this second approximation, we repeat the process to obtain a third approximation, etc. Under certain conditions relating to the coefficients of the system, this sequence will converge to the exact solution. We can set up recurrence relations which show clearly how the iterative process proceeds. Denoting the k-th and k+1-th approximations by (x_1^k, x_2^k, x_3^k) and $(x_1^k + 1, x_2^k + 1, x_3^k + 1)$, respectively, we find

$$x_1^k + 1 = 1.3-.2x_2^k - .1x_3^k(1)'$$

$$x_2^k + 1 = 1.3-.2x_1^k + 1 - .1x_3^k(2)' \; x_3^k + 1 = 1.3-.2x_1^k + 1 - .1x_2^k + 1(3)'$$

We begin with the starting vector all components of which are 0, and then apply these relations repeatedly in the order (1)', (2)' and (3)'. Note that, when we insert values for x_l, x_2 and x_3 into the right-hand sides, we always use the most recent estimates found for each unknown

Convergence

It is seen that the Gauss-Seidel solutions are rapidly approaching these values; in other words, the method is between the x^k+1 and x^k values are suitably small. 0ne stopping rule is to end the iteration when

$$S_k = \sum_{i=1}^{n} \left| x_i^k + 1 - x_i^k \right|$$

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1,n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2,n}x_n = b_2$$

$\vdots$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{n,n}x_n = b_n$$

Algorithm:

```
read n, $a_{11}$., $a_{nn}$, $b_1$,......, $b_n$
for k=1 to n-1 do
for i=k+1 to n do
$m = a_{ik}/a_{kk}$
for j=k+1 to n do
$a_{ij} = a_{ij} - m * a_{kj}$
endfor
$b_i = b_i - m * b_k$
endfor
endfor
$x_n = b_n/a_{nn}$
for i=n-1 down to i do
$x_i = b_i$
for j= i+1 to n do
$x_i = x_i - a_{ij} * x_j$
endfor
```

$x_i = x_i / a_{ii}$

endfor

2.3.2 THE JACOBI METHOD

Introduction:

Jacobi$'$s method is an easily understood algorithm for finding all eigenpairs for a symmetric matrix. It is a reliable method that produces uniformly accurate answers for the results. For matrices of order up to $10X10$, the algorithm is competitive with more sophisticated ones. If speed is not a major consideration, it is quite acceptable for matrices up to order $20X20$. A solution is guaranteed for all real symmetric matrices when Jacobi$'$s method is used. This limitation is not severe since many practical problems of applied mathematics and engineering involve symmetric matrices. From a theoretical viewpoint, the method embodies techniques that are found in more sophisticated algorithms. For instructive purposes, it is worthwhile to investigate the details of Jacobi$'$s method.

Jacobi Series of Transformations

Start with the real symmetric matrix $\bar{A}$. Then construct the sequence of orthogonal matrices R_1 ,R_2 ,..., R_n as follows:

$$D_0 = \bar{A} \text{ and}$$
$$D_j = R_j^T \ D_j \ R_j \text{ for j = 1, 2,...}$$

It is possible to construct the sequence R_j so that

$$\lim_{j\to\infty} D_j = D = \mathrm{diag}(\lambda_1 \ldots \lambda_n).$$

In practice we will stop when the off-diagonal elements are close to zero. Then we will have

$$D_m \approx = D.$$

The Jacobi method is easily derived by examining each of the n equations in the linear system Ax = b in isolation. If in the ith equation

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i.$$

We solve for the value of x_i while assuming the other entries of x_i remain fixed, we obtain

$$x_i = (b_i - \sum_{j \neq i} a_{i,j} x_j) \div a_{i,i}$$

This suggests an iterative method defined by

$$x_i^k = (b_i - \sum_{j \neq i} a_{i,j} x_j^{k-1}) \div a_{i,i}$$

which is the Jacobi method. Note that the order in which the equations are examined is irrelevant, since the Jacobi method treats them independently. For this reason, the Jacobi method is also known as the method of simultaneous displacements, since the updates could in principle be done simultaneously.Simultaneous displacements, method of: Jacobi method.
In matrix terms, the definition of the Jacobi method can be expressed as

$$x^{(k)} = D^{-1} (L + U)\, x^{k-1} + D^{-1} b$$

where the matrices D, L and U represent the diagonal, the strictly lower-triangular, and the strictly upper-triangular parts of A, respectively.

Algorithm

```
Choose an initial guess $x^0$ to the solution
k = 0
check if convergence is reached
while convergence not reached do
for i := 1 step until n do
$\sigma = 0$
for j := 1 step until n do
if $j \neq i$ then
$\sigma = \sigma + a_{ij} x_j^k$
end if
end (j-loop)
$x_i^{k+1} = (b_i - \sigma) \div a_{ii}$
end (i-loop)
check if convergence is reached
k = k+1
```

loop (while convergence condition not reached)

Convergence of the Jacobi method

Iterative methods are often used for solving discretized partial differential equations. In that context a rigorous analysis of the convergence of simple methods such as the Jacobi method can be given.

As an example, consider the boundary value problem

$$£u = -u_{xx} = f$$

$$\text{on } (0,1),$$

$$u(0) = u_0,\ u(1) = u_1$$

discretized by

$$£u(x_i) = 2u(x_i) - u(x_{i-1})\ u(x_{i+1}) = f(x_i) / N^2 \text{ for } x_i = i/N,\ i=1,\ldots,N-1.$$

The eigenfunctions of the £ and Loperator are the same: for n = 1,. . .,N-1 the function $u_n(x) = \sin n\pi x$ is an eigenfunction corresponding to $\lambda = 4\ \sin^2 n\pi/(2N)$. The eigenvalues of the Jacobi iteration matrix B are then

$$\lambda(B) = 1 - \lambda(L)/2 = 1 - \sin^2 n\pi/(2N)$$

From this it is easy to see that the high frequency modes (eigenfunction u_n with n large) are damped quickly, whereas the damping factor for modes with n small is close to1. The spectral radius of the Jacobi iteration matrix is $\approx 1 - 10/\ N^2$ and it is attained for the eigenfunction

$$u(x) = \sin\pi x$$

Spectral radius: The spectral radius of a matrix A is max $\{|\lambda(A)|\}$. Spectrum: The set of all eigenvalues of a matrix.

The type of analysis applied to this example can be generalized to higher dimensions and other stationary iterative methods. For both the Jacobi and Gauss-Seidel method (below) the spectral radius is found to be 1 - O(h^2) where h is the discretization mesh width, h = N^{-d}

where N is the number of variables and d is the number of space dimensions.

While numerical techniques abound to solve PDEs such as the Laplace equation, we will focus on the use of an iterative method as it will be shown to be readily parallelizable and lends itself to the opportunity to apply cartesian topology. The simplest of iterative techniques is the Jacobi scheme, which can be stated as follows:

Make initial guess for u$_{i,j}$ at all interior points (i,j) for all i=1:m and j=1:m.

1. Use Eq. 3 to compute u$_{i,j}^{n+1}$ at all interior points (i,j).
2. Stop if prescribed convergence threshold is reached, otherwise continue on next step.
3. u$_{i}^{n}$ = u$_{ij}^{n+1}$
4. Go to Step 2.

2.3.3 CONJUGATE GRADIENT METHOD

The Conjugate Gradient method is an effective method for symmetric positive definite systems. The method proceeds by generating vector sequences of iterates, residuals corresponding to the iterates, and search directions used in updating the iterates and residuals.

The unpreconditioned conjugate gradient method constructs the kth iterate x^k as an element of x^0 + span $\left\{r^0,\ldots,A^{k-1}r^0\right\}$ so that $\left(x^k-\bar{x}\right)$Ax^k - $\bar{x}$ is minimized, where $\bar{x}$ is the exact solution of Ax=b. This minimum is guaranteed to exist in general only if A is symmetric positive definite. The conjugate gradient iterates converge to the solution of Ax=b in no more than n steps, where n is the size of the matrix.

In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make the sequences satisfy certain orthogonal conditions. On a symmetric positive definite linear system these conditions imply that the distance to the true solution is minimized in some norm.

The iterates x^k are updated in each iteration by a^k multiple a of the search direction vector p^k

$$x^k = x^{k-1} + a^k\ p^k$$

Correspondingly the residuals r^k = b - Ax^k are updated as

$$r^k = r^{k-1} + aAp^k$$

The choice a= $r^{(k-1)T}r^{k-1}/p^{kT}$ minimizes $r^{(k)T}A^{-1}r^k$

The search directions are updated using the residuals $p^k = r^k + \beta^{k-1}\, p^{k-1}$ where the choice $\beta^k = r^{(k)T}r^k/r^{(k-1)T}r^{k-1}$ ensures that r^k and r^{k-1} are orthogonal.

Algorithm

The above algorithm gives the most straightforward explanation of the conjugate gradient method. However, it requires storage of all previous searching directions and residue vectors, as well as many matrix vector multiplications, and thus can be computationally expensive. In practice, one slightly modifies the condition obtaining the last residue vector, not to minimize the metric following the search direction, but instead to make it orthogonal to the previous residue. Minimization of the metric along the search direction will be obtained automatically in this case. One can then obtain an algorithm which only requires storage of the last two residue vectors and the last search direction, and only one matrix vector multiplication. Note that the algorithm described below is equivalent to the previously discussed straightforward procedure. The algorithm is detailed below for solving Ax = b where A is a real, symmetric, positive-definite matrix. The input vector x_0 can be an approximate initial solution or 0.

```
$r_0 := b - Ax_0$
$p_0 := r_0$
k : 0
repeat
$q_k := r_k^T r_k \,/\, p_k^T Ap_k$
$X_{k+1} := X_k + a_k p_k$
$r_{k+1} := r_k - a_k Ap_k$
```

if r_{k+1} is sufficiently small then exit loop end if

```
$\beta_k := r_{k+1}^T r_{k+1} / r_k^T r_k$
$P_{k+1} := r_{k+1} + \beta_k p_{k+1}$ k:=k+1 end repeat
The result is $x_{k+1}$
```

2.3.4 SUCCESSIVE OVERRELAXATION METHOD

The successive overrelaxation method (SOR) is a method of solving a linear system of equations Ax=b derived by extrapolating the Gauss-Seidel method. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component,

$$\mathrm{x}_i^{(k)} = \varpi x_i^{(k)} + (1-\varpi)\, x_i^{(k-1)}$$

where $\bar{x}$ denotes a Gauss-Seidel iterate and ω is the extrapolation factor. The idea is to choose a value for ω that will accelerate the rate of convergence of the iterates to the solution. In matrix terms, the SOR algorithm can be written as

$$\mathrm{x}^{(k)} = (D-\varpi L)^{-1}\,[\varpi U + (1-\varpi)D]\mathrm{x}^{(k-1)} + \varpi\,(D-\varpi L)^{-1}\mathrm{b}$$

where the matrices D, L, and U represent the diagonal, strictly lower-triangular, and strictly upper-triangular parts of A, respectively.
If ϖ = 1, the SOR method simplifies to the Gauss-Seidel method. A theorem due to Kahan (1958) shows that SOR fails to converge if ω is outside the interval (0,2)
In general, it is not possible to compute in advance the value of . that will maximize the rate of convergence of SOR. Frequently, some heuristic estimate is used, such as ϖ = 2- O(h) where h is the mesh spacing of the discretization of the underlying physical domain.
Suppose that iteration is used to solve the linear system AX=B, and that P_k is an approximate solution. We call R_k = B-AP_k the residual vector, and if P_k is a good approximation then R_k =0. A method based on reducing the norm of the residual will produce a sequence P_k that converges faster. The successive over relaxation - SOR method introduces a parameter ϖ which speeds up convergence. The SOR method can be used in the numerical solution of certain partial differential equations.

Algorithm

Since elements can be overwritten as they are computed in this algorithm, only one storage vector is needed, and vector indexing is omitted. The algorithm goes as follows:

```
Inputs: A, b, $\omega$
Output: $\varphi$
Choose an initial guess to $\varphi$ the solution
repeat until convergence
for i from 1 until n do
$\sigma \leftarrow 0$
for j from 1 until n do
if $j \neq i$ then
$\sigma \leftarrow \sigma + a^{ij}\varphi_j$
end if
end (j-loop)
$\varphi_j \leftarrow (1-w)\,\varphi_j + w/a_{ii}\,(b_i - \sigma)$
end (i-loop)
check if convergence is reached end (repeat)
```

CHAPTER 3
OUR CONTRIBUTION

3.1 INTRODUCTION

Verified solvers for dense linear (interval-) systems require a lot of resources, both in terms of computing power and memory usage. Computing a verified solution of large dense linear systems (dimension n $\leq$ 1000) on a single machine quickly approaches the limits of today$'$s hardware. Therefore, an efficient verified solver for distributed memory systems is needed. In this chapter we introduce a new algorithm to solve the system of linear equations which is minimizes the use of resources both in terms of computing power and memory usage. In this algorithm we assume that system must have solution.

3.2 A NEW ALGORITHM

The pseudo code of the algorithm are given below,

Step 1:

Generate random solution X^*.

Step 2:

Generate nXn matrix A, where $a^{'}_{ij} = a_{ij} \div \sum a^2_{ij}$.

Step 3:

Compute $b_i = \sum_{j=1}^{n} a_{ij} X^*$.

Step 4:

Generate initial point $P_\circ$.

Step 5:

Construct $P_i \perp H_i$ from P_{i-1}. Where $P_i = P_{i-1} + a_i\alpha$ and $\alpha = b_i - a_iP_{i-1}$.

Step 6:

Generate new initial point $P_i^{'}$.

Where $P_i(t) = P_i + (P_i^{'} - P_i)t$ and $t = \sum(a_iP_i - b_i)a_i(P_i^{'} - P_i) \div \sum a_i(P_i^{'} - P_i)$.

Step 7:

Minimize $\sum D_i^2(t)$. Find least value of t .

Where $\sum D_i^2(t) = |a_iP_i(t) - b_i|$.

Step 8:

Start from step 5 again until improvement is insufficient.

3.3 FUNCTIONAL DEFINITION OF THE ALGORITHM

Step 1:

Generate solution X^* having randomly n components. For example , if n=3 then $X^* = [X_1, X_2, X_3]$ where the value of X_1, X_2, X_3 are randomly generated.

Step 2:

Generate nXn matrix A, where coefficients a_{ij} of A are randomly generated. Then normalize matrix A using following rule $a_{ij}^{'} = a_{ij} \div \sum a_{ij}^2$.

Step 3:

Compute constant vector b_i using following rule, $b_i = \sum_{j=1}^{n} a_{ij}X^*$.

Step 4:

Generate initial point $P_\circ$ randomly from where computation starts.

Step 5:

Construct $P_i \perp H_i$ from P_{i-1}. Where $P_i = P_{i-1} + a_i\alpha$ and $\alpha = b_i - a_iP_{i-1}$. Draw perpendicular line from point P_{i-1} to point P_i of the hyperplane H_i where α is a direction vector. We can accomplish this task by using following pseudocode,

```
For i=1 to n do
    $\alpha = b_i - a_i P_{i-1}$
    $P_i = P_{i-1} + a_i \alpha$
Enddo
    $\alpha = b_n - a_n P_{n-1}$
    $P_1 = P_{n-1} + a_n \alpha$
```

Step 6:

Generate new initial point $P_i^{'}$.

Where $P_i(t) = P_i + (P_i^{'} - P_i)t$ and $t = \sum(a_i P_i - b_i) a_i (P_i^{'} - P_i) \div \sum a_i (P_i^{'} - P_i)$.

Each time a new point is generated from the two point $P_i^{'}$, P_i on the same hyperplane with direction vector t which can be calculated by using function f(t).
Here , f(t) can be defined by,

$f(t) = \sum_n^{i=1} |a_i P_i(t) - b_i|^2$

Now we take $f^{'}(t) = 0$ and get the value of t which is given below,

$t = \sum(a_i P_i - b_i) a_i (P_i^{'} - P_i) \div \sum a_i (P_i^{'} - P_i)$.

Step 7:

Calculate distance between newly generated point $P_i(t)$ and initial solution X^* where distance defined by $\sum D_i^2(t) = |a_i P_i(t) - b_i|$ and try to minimize this distance.

Step 8:

Repeat algorithm from step 5 untill improvement is insufficient.

3.4 TOTAL COMPLEXITY ANALYSIS OF THE ALGORITHM

1. For normalize matrix the required complexity order is $O(9n^2+9n+2)$.
2. To construct perpendicular line to each hyperplane and finding two points on this hyperplane the required complexity order is $O(18n^2+21n+6)$.
3. To determine new initial point the required complexity order is $O(9n^2+21n+7)$.

4.To minimize distance between solution vector and new point on the hyperplane the required complexity order is O(6n+7).
So that , the total time complexity of our proposed algorithm is O(36n^2+57n+22). So , approximately it can be determined as O(n^2).

3.5 ANALYTICAL COMPARISON WITH EXISTING ALGORITHM

Our proposed algorithm provides the time complexity O(n^2). At the other hand the existing algorithm Gauss Elimination method which is a direct method has time complexity O(n^3). So our proposed algorithm takes less time than Gauss Elimination Method. Moreover our proposed algorithm solves problem of system of linear equation using indirect method whereas Gauss Elimination Method solves problem of system of linear equation using direct method. So the solution of proposed algorithm has less roundoff or other errors than direct method. As direct methods lead to a very poor and useless results. This is because of the various types of errors involved in numerical approximations.

CHAPTER 4
EXPERIMENTAL COMPARISON

4.1 EXPERIMENTAL SETUP

In this chapter we want to establish effectiveness of solving system of linear equation. The following files are used for the experiments

- .txt file
- .cpp file

Our proposed algorithm technique is implemented by the help of Microsoft Visual Studio 2006 and the algorithm is implemented in the system which has following property are given below

Processor	: Intel(R) Core(TM) i5 CPU
Memory(RAM)	: 4.00 GB
System Type	: 32-bit Operating System

4.2 EXPERIMENTAL ANALYSIS

The existing algorithm Gauss Elimination Method is a direct method whereas our proposed algorithm is a indirect method. Now we compare our proposed algorithm with respect to Gauss Elimination method. Let us assume that n is the dimension size of the system of linear equation. Then for different size of dimension of the system of linear equation we apply both the algorithm and compare there code executing time in seconds.

The table 4.1 shows the analysis result of the two algorithm which are given below

Table 4.1 Analysis of Data

Step	Dimension Size,N	Gauss Elimination Method	Proposed Algorithm
1	N=15	0.001	0.001
2	N=20	0.001	0.001
3	N=25	0.002	0.002
4	N=30	0.004	0.003
4	N=50	0.013	0.011
4	N=80	0.027	0.O23
4	N=100	0.031	0.030
4	N=500	0.154	0.143
4	N=800	0.181	0.177
4	N=900	0.198	0.193
4	N=1000	0.201	0.197

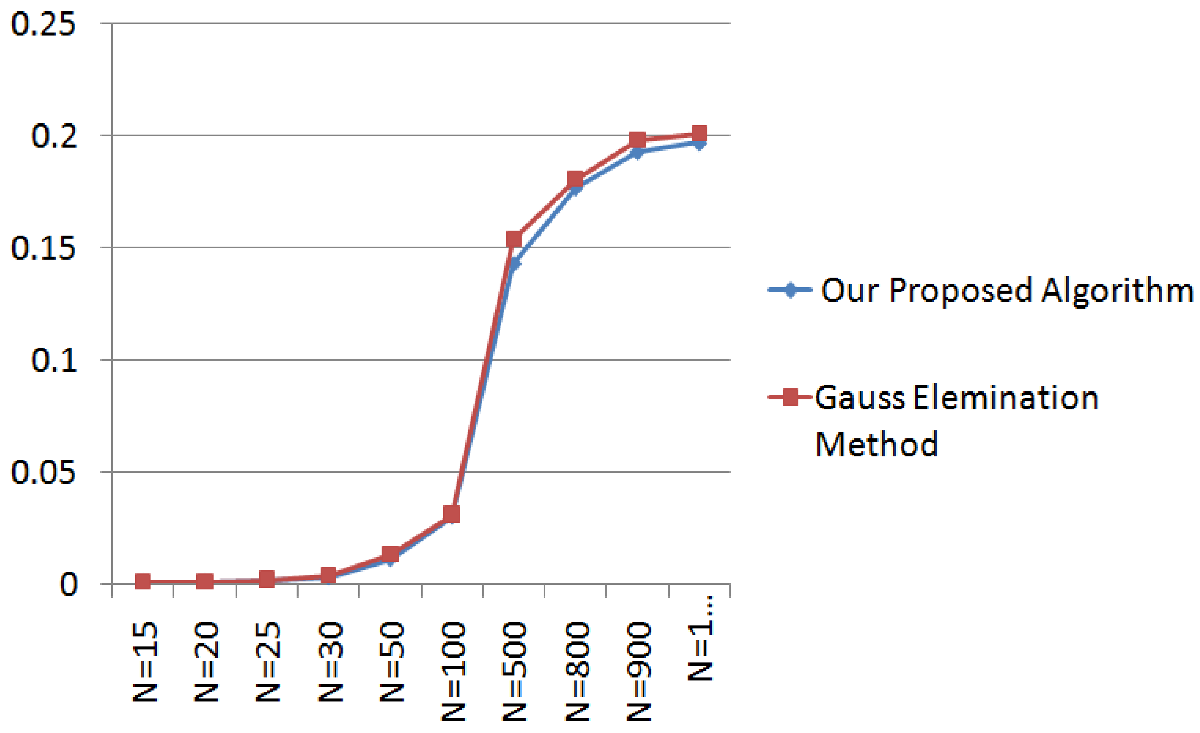

Figure 4.1: Analysis Graph

Form the result of experiment we can see that execution time of our proposed algorithm is little bit less than the execution time of gauss elimination algorithm. Though proposed algorithm.s execution time is little bit less compare to the Gauss Elemination Method.s execution time yet the effectiveness of solving system of linear equation is improved. We are successful in achieving the aim of minimize the computing power and memory usage.

4.3 DISCUSSION

Experimental results show that the proposed algorithm provide a cost efficient solution of a system of linear equation. Because of the efficient solution overall time complexity of the algorithm decreases. Hence we get a little bit better algorithm than existing algorithm.

CHAPTER 5
RESULT AND RECOMMENDATION

5.1 RESULT

Solving system of linear equations with minimum cost is well defined problem as studied from long years ago. Our goal was to solve the system of linear equation in a cost effective way which is necessary to analyze people and understand several real world situations. We are successfully reached our goal by minimizing the execution time of the proposed algorithm. So that complexity order of the proposed algorithm has become O(n^2) but existing algorithm(Gauss Elimination Method) has complexity order O(n^3).

5.2 RECOMMENDATION

In this thesis work some existing algorithm for solving system of linear equation which are direct in nature are studied , implemented and tested. For example Gauss Elimination Method. The proposed algorithm improves the efficiency of the existing algorithm little bit. Future works on solving system of linear equation algorithm can be carried on

- An efficient algorithm that minimizes running time more.
- Design an algorithm considering case of system of linear equation which have no solution.
- An efficient algorithm which improves complexity order more.
- Design an algorithm which provide better result than our proposed algorithm.

REFERENCES

[1] Robert J. Plemmons Abraham Berman. *Nonnegative Matrices in the Mathematical Sciences*. SIAM.

[2] McLaughlin Renate Althoen, Steven C. Gaussjordan reduction: a brief history. *The American Mathematical Monthly (Mathematical Association of America)*, 1987.

[3] Kendall A. Atkinson. *An Introduction to Numerical Analysis*. John Wiley Sons, ISBN 978-0-471-50023-0, 1989.

[4] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, ISBN 0-486-43227-0.

[5] Sheldon Jay Axler. *Linear Algebra Done Right*. Springer-Verlag, SBN 0-387-98259-0, 1997.

[6] Rajendra Bhatia. Matrix analysis, graduate texts in mathematics. 1996.

[7] John Bunch, James R.; Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation)*, 1974.

[8] Ronald. Calinger. *A Contextual History of Mathematics*. Prentice Hall, ISBN 978-0-02-318285-3, 1999.

[9] Leiserson Charles E. Rivest Ronald L. Stein Clifford Cormen, Thomas H. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[10] Jr David M. Young. *Iterative Solution of Large Linear Systems*. Academic Press.

[11] J. B. Fraleigh and R. A. Beauregard. *Linear Algebra*. Addison-Wesley Publishing Company, 1995.

[12] Felix R. Gantmacher. *Matrix Theory Vol. 2*. American Mathematical Society.

[13] Charles F. Golub, Gene H.; Van Loan. *Matrix Computations*. Johns Hopkins, ISBN 978-0-8018-5414-9, 1996.

[14] Charles F. Golub, Gene H.; Van Loan. *Schaum's outline of theory and problems of linear algebra*. New York: McGraw-Hill, ISBN 978-0-07-136200-9, 2001.

[15] Stiefel Eduard Hestenes, Magnus R. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards 49*, 1952.

[16] Johnson Charles R. . Horn, Roger A. *Topics in Matrix Analysis*. Cambridge University Press.

[17] Alston S. Householder. The theory of matrices in numerical analysis. 1975.

[18] David C. Lay. *Linear Algebra and Its Applications*. Addison Wesley, ISBN 978-0-321-28713-7, 2005.

[19] Steven J. Leon. Linear algebra with applications. 2006.

[20] Seymour Lipschutz and Mark Lipson. *Schaum's Outlines: Linear Algebra*. Tata McGrawHill edition, Delhi, 2001.

[21] David McMahon. *Linear Algebra Demystified*. McGrawHill Professional.

[22] P.J. Olver. *Equivalence, invariants, and symmetry*. Oxford University Press.

[23] David Poole. *Linear Algebra: A Modern Introduction*. Thomson Brooks/Cole, Canada, 2006.

[24] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, ISBN 978-0-89871-534-7.

[25] Thomas S. Shores. Applied linear algebra and matrix analysis. 2006.

[26] Gilbert Strang. Linear algebra and its applications. 2005.

[27] Richard S. Varga. *Matrix Iterative Analysis*. Prentice Hall edition,Springer-Verlag.

Appendix-A

```
#include<iostream>

#include<time.h>

using namespace std ;


int mat[1000][1000] ,b[1000],c[1000],flag=0;

double a[1000][1000],alpha,x[1000][1000],y[1000][1000],t[1000][1000],p[1000][1000];



void NormalizeMatrix(int n)

{

        int i,j,k,sum;

        for(i=1;i<=n;i++)

        {

                sum=0;

                for(j=1;j<=n;j++)

                {

                        sum=sum+(mat[i][j]*mat[i][j]);

                }

                for(k=1;k<=n;k++)

                {


                        a[i][k] = (double)mat[i][k]/sum;


                }

        }

}


void HyperplanePoint(int n)

{

        int i,j,k;


        for(i=1;i<=n;i++)
        {
                alpha=b[i];
                for(j=1;j<=n;j++)
                {
                        alpha = alpha - (a[i][j]*x[i-1][j]);
                }
                for(k=1;k<=n;k++)
                {
                        x[i][k]=x[i-1][k] + (alpha* a[i][k]);
                }

        }

        for(i=1;i<=n;i++)
        {
                y[0][i]=x[n][i];

        }

        for(i=1;i<=n;i++)
        {
                alpha=b[i];
                for(j=1;j<=n;j++)
                {
                        alpha= alpha - (a[i][j]*y[i-1][j]);
                }
                for(k=1;k<=n;k++)
                {
                        y[i][k]=y[i-1][j] + (alpha*a[i][k]);
                }
        }

}
```

```
void solutionVector(int n)
{
        int i,j;
        double z[200][200],s1[200][200],s2[200][200];
        s1[1][1]=0;
        s2[1][1]=0;
        for(i=1;i<=n;i++)
        {

                for(j=1;j<=n;j++)
                {
                        z[i][j] = (y[i][j]-x[i][j])*a[i][j];
                        s2[i][j] = s2[i-1][j]+(y[i][j]-x[i][j])*a[i][j];
                        s1[1][j] = s1[i-1][j]+(a[i][j]*x[i][j] - b[i])*z[i][j];

                }
        }

        for(i=1;i<=n;i++)
        t[0][i]= s1[n][i]/s2[n][i];

}

void solution(int n)
{
        int i,j;
        double dist[200][200];
        for(i=1;i<=n;i++)
        dist[0][i]=0;
        for(j=1;j<=n;j++)
        {
                p[0][j]=x[0][j] + (y[0][j]-x[0][j])* t[0][j];
        }
        for(i=1;i<=n;i++)
        {
                dist[0][i] = ((p[0][i]-x[0][i])*(p[0][i]-x[0][i]));
                if(dist[0][i]==0)
                        flag=1;
                else
                        flag=0;
        }
        printf("\n");
}
int main()
{
        clock_t start,end;
        double runTime;
        freopen("100n.txt","r",stdin);

        int n ,i ,j,q=0 ;
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                for(j=1 ; j<=n ; j++)
                {
                        scanf("%d",&mat[i][j]);
                }

        }

        for(i=1;i<=n;i++)
        {
                scanf("%d",&c[i]);
        }
        b[0]=0;
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        b[i]= b[i]+mat[i][j]*c[i];

                }
        }
        for(i=1;i<=n;i++)
                scanf("%lf",&x[0][i]);

        start=clock();

        NormalizeMatrix(n);

        while(flag==1)
        {

        for(i=1;i<=n;i++)
        {
                x[0][i]=p[0][i];
        }
        HyperplanePoint(n);

        solutionVector(n);
        solution(n);
        q++;
        }
        end = clock();
  runTime = ((end - start) / (double) CLOCKS_PER_SEC );
  printf ("Run time is \%g seconds", runTime);

return 0 ;
}
```