ÇANKAYA UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MATHEMATICS AND COMPUTER SCIENCE

MASTER THESIS

ITERATIVE SOLUTION OF SPARSE LINEAR SYSTEMS

LAILA ABOSHARB

JANUARY 2013

Title of the Thesis: **Iterative Solution of Sparse Linear Systems**

Submitted by **Laila Aboshарb**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

[illegible]

Prof. Dr. Taner Altunok
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

Billûr K

Prof. Dr. Billur Kaymakçalan
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Emre Sermutlu
Supervisor

Examination Date: 23-01.2013

Examining Committee Members

Assist. Prof.Dr. Tolga Pusatlı	(Çankaya Univ.)	[illegible]
Assist. Prof. Dr. Emre Sermutlu	(Çankaya Univ.)	[illegible]
Assoc. Prof. Dr. Burak Aksoylu	(TOBB Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	: Laila Abosharb
Signature	: [illegible]
Date	: 23.01.2013

ABSTRACT

ITERATIVE SOLUTION OF SPARSE LINEAR SYSTEMS

ABOSHARB, Laila

M.Sc., Department of Mathematics and Computer Science

Supervisor: Assist. Prof. Dr. Emre Sermutlu

January 2013, 46 pages

Linear systems of equations are encountered frequently in many problems in science and engineering. In large systems representing complicated problems, it is vital to make use of the sparsity of the problem. In such systems, using iterative methods rather than direct methods may decrease the time necessary for solutions. This theses is a survey of techniques used to obtain the solution of large sparse linear systems with emphasis on preconditioning. Mainly, we compare the number of arithmetic operations necessary to solve sparse linear systems using Gaussian elimination before and after reordering the coefficient matrix by Cuthill-McKee algorithm to reduce bandwidth.

Keywords: Linear Systems of Equations, Gaussian Elimination, LU Factorization, Cuthill-McKee Ordering, Iterative Methods, Sparse Matrices, Preconditioning

ÖZ

SEYREK DOĞRUSAL SİSTEMLERİN TEKRARLAMALI ÇÖZÜMLERİ

ABOSHARB, Laila

M.Sc., Matematik–Bilgisayar Bölümü

Tez Yöneticisi: Assist. Prof. Dr. Emre Sermutlu

Ocak 2013, 46 pages

Fen ve mühendislikteki pek çok problemde doğrusal denklem sistemleriyle sıkça karşılaşılmaktadır. Karmaşık problemleri temsil eden büyük sistemlerde, problemdeki seyrekliği kullanmak hayati önem taşır. Bu tür sistemlerde, doğrudan çözümler yerine tekrarlamalı çözüm metodları kullanmak çözüm süresini azaltabilir. Bu tez büyük seyrek lineer sistemlerin çözümleri için kullanılan teknikleri ve özellikle önhazırlama metodlarını incelemektedir. Başlıca sonucu, seyrek doğrusal sistemler Gauss eleme metodu ile çözümlerinde Cuthill-McKee algoritması ile önhazırlama yapmadan önce ve yaptıktan sonra, gerekli aritmetik işlem sayılarının karşılaştırılmasıdır.

Anahtar Kelimeler: Doğrusal Denklem Sistemleri, Gauss Eleme Metodu, LU Çarpanlara Ayırma metodu, Cuthill-McKee Sıralaması, Tekrarlamalı metodlar, Seyrek Matrisler, Önhazırlama.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my deep sense of gratitude to the person who has taught me what dedication and respect are, and gave me the power to continue my thesis supervisor Assist. Prof. Dr. Emre Sermutlu. Thank you for every thing.

I would like to thank My mother and my father, and how much I wished to see your faces in this day and I hope God to be proud of me.

I would like to thank my husband Salem, who was with me in every step and gave me all his love and support me to lead to this day.

I want to say to my small family my love children (Yakeen, Omar and my baby Yumna) every thing was because of you in my life and all what I did and will do for you.

I would like to express my deep gratitude to my family (my sisters and my brothers) for their endless and continuous encourage and support throughout the years.

Also I would not forget thank my teachers Prof. Dr. Fahd Jarad and Prof. Dr. Thabet Abdaljawad who gave to me their help and support as a friend not as a student.

Finally, I would like to thank everyone supported me, even if this support was a word.

I would like to dedicate this work to the MARTYRS of LIBYA who sacrificed their lives to breathe freedom and I would like to say to them every speck of dust of Libya is proud of you, and we will continue the journey you starting to be proud of us to live up to our country to the highest levels.

TABLE OF CONTENTS

LIST OF TABLES

INTRODUCTION

Solving large linear systems of equations is an important step in many problems of applied science and engineering. In theory, we can solve any such system provided it is consistent, but in practice, direct solutions may take an excessive amount of time.

The matrices representing such systems are usually sparse, that is, most of the entries are zero. In this theses, we implement on MATLAB and analyze numerical techniques for the solution of such linear systems. We start with direct methods and then consider iterative methods of Gauss-Seidel, Jacobi and SOR.

Preconditioning a matrix can reduce fill-ins and therefore the number of steps of a solution. Among many different techniques, we investigate Cuthill-McKee algorithm in depth.

CHAPTER I

DIRECT METHODS FOR SOLVING SYSTEM OF LINEAR EQUATIONS

1.1 SYSTEM OF LINEAR EQUATIONS

A system of linear equation is simply a finite set of linear equations.

For example,

$$\begin{array}{ccccccc} x_1 & + & 4x_2 & + & x_3 & = & 1 \\ 2x_1 & + & 4x_2 & + & x_3 & = & 9 \\ 3x_1 & + & 5x_2 & - & 2x_3 & = & 11 \end{array} \tag{1.1}$$

is system of three equations in three variables $x1$, $x2$ and $x3$.

In order to write a general system of m linear equations in the n variables $x_1, x_2, ..., x_n$, we have:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & a_{2n}x_n & = & b_2 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & a_{mn}x_n & = & b_m \end{array} \tag{1.2}$$

or, in compact form system (1.1) can be written

$$\sum_{j=1}^{n} a_{ij}x_j i = 1, 2, \cdots, m. \tag{1.3}$$

1.2 SOLUTION OF LINEAR SYSTEM

Every system of linear equation has either no solution, exactly one solution, or infinitely many solution.

A system of equations with no solution is said to be *inconsistent* and if it has at least one solution, it is said to be *consistent.*

1.3 LINEAR SYSTEM IN MATRIX NOTATION

The general simultaneous system of n linear equations in the n unknown variables in the (1.1) can be written as:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{2n} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{1.4}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{2n} & \cdots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{1.5}$$

A the coefficient matrix, x the column matrix of unknown, and b the column matrix of constants in the system (1.3) can be written compactly as:

$$Ax = b \tag{1.6}$$

1.4 DIRECT METHOD FOR LINEAR SYSTEM

To solve the system of linear equations using numerical methods, there are two types of methods available. First type are called *direct methods* or *elimination method*. Like *(Cramer's rule, Gaussian elimination method, Gauss-Jordon, and LU decomposition.)* This type of method finds the solution in a finite number of steps. These methods are guaranteed to succeed and are recommended for general purposes. Second type are called *indirect or iterative methods*. Iterative methods start with an arbitrary first approximation to the unknown solution x of linear system $AX = b$.

Jacobi method, Gauss-Seidel method, successive over-relaxation (SOR), and conjugate method are widely used for solving large systems of equations. In this chapter we consider just two methods for first type and in next chapters we will consider the other methods.[2]

1.5 GAUSSIAN ELIMINATION METHOD

This method is one of the most popular and widely used direct methods for solving linear system of algebraic equations.

The Gaussian elimination procedure start with forward elimination, in which the first equation in the linear system is used to eliminate the first variable from the rest of the $(n-1)$ equations. Then the new second equation is used to eliminate the second variable from the rest of the $(n-2)$ equations, and so on. If $(n-1)$ such elimination is performed,then the resulting system will be the triangle form. Once this forward elimination is completed, we can determine whether the system is overdetermined or undetermined or has a unique solution. If it has a unique solution, then *backward substitution* is used to solve triangle system easily and one can find the unknown variables involved in the system.[2]

Example 1.5.1.

Solve the following system using Gaussian elimination method

$$\begin{array}{rcrcrcrcr} 4x_1 & + & 12x_2 & + & 8x_3 & + & 4x_4 & = & -4 \\ x_1 & + & 7x_2 & + & 18x_3 & + & 9x_4 & = & -5 \\ 2x_1 & + & 9x_2 & + & 20x_3 & + & 20x_4 & = & -25 \\ 3x_1 & + & 11x_2 & + & 15x_3 & + & 14x_4 & = & -18 \end{array} \tag{1.7}$$

Solution: Write the given system in the augmented form

$$\begin{bmatrix} 4 & 12 & 8 & 4 & \vdots & -4 \\ 1 & 7 & 18 & 9 & \vdots & -5 \\ 2 & 9 & 20 & 20 & \vdots & -25 \\ 3 & 11 & 15 & 14 & \vdots & -18 \end{bmatrix} \tag{1.8}$$

Firstly we have

$$\begin{array}{rcrcr} m_{21} & = & -a_{21}/a_{11} & = & -1/4 \\ m_{31} & = & -a_{31}/a_{11} & = & -1/4 \\ m_{41} & = & -a_{41}/a_{11} & = & -3/4 \end{array} \tag{1.9}$$

Now multiply the first row by m_{21} and add the result to the second row ,multiply the first row by m_{31} and add the result to the third row and multiply the first row by m_{41} and add the result to the fourth row
$R_2 = R_1 * m_{21} + R_2$
$R_3 = R_1 * m_{31} + R_3$
$R_4 = R_1 * m_{41} + R_4$
we will get

$$\begin{array}{ccccccc} a_{21} = 0 & ; & a_{22} = 4 & ; & a_{23} = 16 & ; & a_{24} = 8 \\ a_{31} = 0 & ; & a_{32} = 3 & ; & a_{33} = 16 & ; & a_{34} = 18 \\ a_{41} = 0 & ; & a_{42} = 2 & ; & a_{43} = 9 & ; & a_{44} = 11 \\ b_2 = -4 & ; & b_3 = -23 & ; & b_4 = -23 & & \end{array} \tag{1.10}$$

Now we will get the first elimination as

$$\begin{bmatrix} 4 & 12 & 8 & 4 & \vdots & -4 \\ 0 & 4 & 16 & 8 & \vdots & -4 \\ 0 & 3 & 16 & 18 & \vdots & -23 \\ 0 & 2 & 9 & 11 & \vdots & -15 \end{bmatrix} \tag{1.11}$$

Secondly we find:

$$\begin{aligned} m_{32} &= -a_{32}/a_{22} &= -3/4 \\ m_{42} &= -a_{42}/a_{22} &= -1/4 \end{aligned} \tag{1.12}$$

We do the same thing for the third and forth rows

$R_3 = R_2 * m_{32} + R_3$

$R_4 = R_2 * m_{42} + R_4$

We will get

$$\begin{aligned} &a_{31} = 0 \quad ; \quad a_{32} = 0 \quad ; \quad a_{33} = 4 \quad ; \quad a_{34} = 12 \\ &a_{41} = 0 \quad ; \quad a_{42} = 0 \quad ; \quad a_{43} = 1 \quad ; \quad a_{44} = 7 \\ &b_3 = -20 \quad ; \quad b_4 = -13 \end{aligned} \tag{1.13}$$

The second elimination is

$$\begin{bmatrix} 4 & 12 & 8 & 4 & \vdots & -4 \\ 0 & 4 & 16 & 8 & \vdots & -4 \\ 0 & 0 & 4 & 12 & \vdots & -20 \\ 0 & 0 & 1 & 7 & \vdots & -13 \end{bmatrix} \tag{1.14}$$

Finally

$$\begin{aligned} m_{43} &= -a_{43}/a_{33} &= -1/4 \\ R_4 &= R_3 * m_{43} &+ \quad R_4 \end{aligned} \tag{1.15}$$

Now we will get

$$\begin{aligned} &a_{41} = 0 \quad ; \quad a_{42} = 0 \quad ; \quad a_{43} = 0 \quad ; \quad a_{44} = 4 \\ &b_4 = -8 \end{aligned} \tag{1.16}$$

And the last elimination is

$$\begin{bmatrix} 4 & 12 & 8 & 4 & \vdots & -4 \\ 0 & 4 & 16 & 8 & \vdots & -4 \\ 0 & 0 & 4 & 12 & \vdots & -20 \\ 0 & 0 & 0 & 4 & \vdots & -8 \end{bmatrix} \tag{1.17}$$

Back substitution

$$\begin{aligned} 4x_4 = -8 &\Rightarrow x_4 = -2 \\ 4x_3 + 12x_4 = -20 &\Rightarrow x_3 = 1 \\ 4x_2 + 16x_3 + 8x_4 = -4 &\Rightarrow x_2 = -1 \\ 4x_1 + 12x_2 + 8x_3 + 4x_4 = -4 &\Rightarrow x_1 = 2 \end{aligned} \tag{1.18}$$

The vector of solution is $X = [-2, 1, -1, 2]^T$

Algorithm Form for Gaussian Elimination

```
Integer $i, j, k$
Input $A$ coefficients matrix $n \times m$
$b$ right-hand side matrix $n1 \times m1$
$n, m$ number of rows and column of $A$
$opc = 0$ number of the operations
if $n \neq m$
  ERROR
  The coefficient matrix must be square
end if
if $n1 \neq n$ And $m1 \neq 1$
  ERROR
  The column vector b must have the same number of rows as A
end if
$A \leftarrow [A\ b]$
for $i = 1$ to $n - 1$ do
    if $A(i, i) = 0$
        $[mxm\ \ indis] = max(|(A(i : n, i)|))$
        $A = interchange(A, i, indis + i - 1)$
    end if
    for $j = i + 1 : min(i + 3, n)$ do
        $C = A(j, i)/A(i, i)$
        for $k = i$ to $n + 1$ do
            $A(j, k) = A(j, k) - C * A(i, k)$
            $opc = opc + 1$
        end for
    end for
end for
$d = min(|(diag(A)|))$
if $d < 10^{-15}$
```

```
 ERROR
     The system does not have unique solution
end if
$sol = zeros(n, 1)$
$sol(n) = A(n, n+1)/A(n, n)$
for $i = n - 1$ to $-1$ to 1
   $T = 0$
   for $j = i + 1$ to $n$ do
      $T = T + A(i, j) * sol(j)$
   end for
   $sol(i) = (A(i, n+1) - T)/A(i, i)$
end for
Return
   $opc$   counter of operations   ,   $sol$   matrix of solution
End Algorithm
```

The file for the Matlab program for Gaussian Elimination is given in Appendix A.

1.6 GAUSSIAN ELIMINATION WITH ROW PIVOTING

This is another direct method to find the solution of a system of linear equations. Basic Gaussian elimination, as presented in the previous section, fails if the pivot element at any stage of the elimination process is zero, because division by zero not possible. In addition, difficulties that are not as easy to detect arise if the pivot element is significantly smaller than the coefficients it is being used to elimination that prevents or alleviates some of these shortcomings of the basic procedure.[2]

Here we develop an implementation of Gaussian elimination, which try to solve the problem which discussed above. In using Gaussian elimination by partial pivoting, the basic approach is to use the largest (in absolute value) element on or below the diagonal in the column of current interest as the pivotal element for elimination in the rest of that column.

Example 1.6.1.

Consider the following linear system

$$\begin{array}{rcrcrcr} 2x_1 & + & 2x_2 & - & 2x_3 & = & 0 \\ -4x_1 & - & 2x_2 & + & 2x_3 & = & -14 \\ -2x_1 & + & 3x_2 & + & 9x_3 & = & 9 \end{array} \tag{1.19}$$

The augmented matrix is

$$\left[\begin{array}{ccccc} 2 & 2 & -2 & \vdots & 8 \\ -4 & -2 & 2 & \vdots & -14 \\ -2 & 3 & 9 & \vdots & 9 \end{array}\right] \tag{1.20}$$

For the first elimination step, since -4 the largest absolute coefficient of first variable x_1, the first row and the second row are interchanged, giving us

$$\left[\begin{array}{ccccc} -4 & -2 & 2 & \vdots & -14 \\ 2 & 2 & -2 & \vdots & 8 \\ -2 & 3 & 9 & \vdots & 9 \end{array}\right] \tag{1.21}$$

By Gaussian elimination we obtain of the first elimination as

$$\left[\begin{array}{ccccc} -4 & -2 & 2 & \vdots & -14 \\ 0 & 1 & -1 & \vdots & 1 \\ 0 & 4 & 8 & \vdots & 16 \end{array}\right] \tag{1.22}$$

For the second elimination step, 4 is the largest absolute coefficient of the second variable x_2, the second and the third row are interchanged and after the elimination we have the second elimination as

$$\left[\begin{array}{ccccc} -4 & -2 & 2 & \vdots & -14 \\ 0 & 4 & 8 & \vdots & 16 \\ 0 & 0 & -3 & \vdots & -3 \end{array}\right] \tag{1.23}$$

Now use backward substitution to get the vector of solution as

$$X = [3, 2, 1]^T \tag{1.24}$$

```
Integer $i, j, k$;
Input $A$ coefficients matrix $n \times n$
$b$ right-hand side of equation $Ax = b$
$n, m$ number of rows and column of $A$
Initialize $M = A * b$
        for $k = 1$ to $n - 1$ do
            $pivot \leftarrow |M(k.k)|$
            $p = k$
            for $i = k + 1$ to $n$ do
                if $(|M(i,k)|) > pivot$
                    $pivot \leftarrow |M(i,k)|$
                    $p = i$
                end if
            end for
            if $(p > k)$
                $T \leftarrow M(k,:)$
                $M(k,:) \leftarrow M(p,:)$
            end if
            for $i = k + 1$ to $n$ do
                $C \leftarrow -M(i,k) \div M(k,k)$
                $M(i,:) \leftarrow M(i,:) + C * M(k,:)$
            end for
        end for
        for $j = 1$ to $m$ do
            $X(k,j) \leftarrow M(n, n+j)/M(n,n)$
            for $k = n - 1$ to $1$ do
                $X(k,j) \leftarrow (M(k, n+j) - M(k, k+1:n) * X(k+1:n, j))/M(k,k)$
            end for
        end for
Return $X$ matrix of solution
End Algorithm
```

Table 1.1: Algorithm for Gaussian Elimination With Pivoting

CHAPTER II

LU FACTORIZATION

An $n \times n$ system of linear equations can be factorized to $LU = A$ where L is lower triangular matrix and U is upper triangular matrix, i,e.,

$$\begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix} = A \tag{2.1}$$

An LU factorization of A can be used to solve linear system of equations i,e $Ax = b$.
There are many methods of finding an LU factorization in this chapter we will cover most of them. LU methods for sparse linear systems are studied in [3, 5].

2.1 DERIVATION OF LU FACTORIZATION

LU factorization of A seeks to find L and U such that $A = LU$. We will consider in next example and explain the method

Example 2.1.1.

Consider the LU factorization of three-by-three matrix

$$A = \begin{bmatrix} 2 & 4 & 0 \\ 2 & 8 & 2 \\ 0 & 4 & 4 \end{bmatrix} \tag{2.2}$$

First of all we have:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, U = \begin{bmatrix} 2 & 4 & 4 \\ 2 & 8 & 2 \\ 0 & 4 & 4 \end{bmatrix} \tag{2.3}$$

$$\begin{aligned} m_{21} = -u_{21}/u_{11} = -2/2 = -1 \\ m_{31} = -u_{31}/u_{11} = 0 \end{aligned} \tag{2.4}$$

Now after the first step from Gaussian elimination we have

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} U = \begin{bmatrix} 2 & 4 & 0 \\ 0 & 4 & 2 \\ 0 & 4 & 4 \end{bmatrix} \tag{2.5}$$

finally

$$m_{32} = -u_{32}/u_{22} = -4/4 = -1 \tag{2.6}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} U = \begin{bmatrix} 2 & 4 & 0 \\ 0 & 4 & 2 \\ 0 & 0 & 2 \end{bmatrix} \tag{2.7}$$

Now multiply L by U we have A

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 4 & 0 \\ 0 & 4 & 2 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 0 \\ 2 & 8 & 2 \\ 0 & 4 & 4 \end{bmatrix} \tag{2.8}$$

```
integer $i, j, k, n$
input $A$ n by n matrix
$L = I$ where $I$ is n by n identity marrix; $U = A$
for $k = 1$ to $n - 1$ do
    for $i = k + 1$ to $n$ do
        $m(i, k) \leftarrow -U(i, k)/U(i, k)$
        for $j = k$ to $n$ do
            $U(i, j) \leftarrow U(i, j) + m(i, k) * U(k, j)$
        end for
            $L(i, k) \leftarrow -m(i, k)$
    end for
end for
return $L; U$
```

Table 2.1: Algorithm for LU factorization using Gaussian Elimination

2.2 LU FACTORIZATION WITH PIVOTING

For problems in which row pivoting must be performed on the coefficient matrix A during Gaussian elimination, the LU factorization with pivoting of A can be represented in matrix $PA = LU$, where P is the permutation matrix that represents the row interchanges that occurred during the pivoting. We will explain this method in next example:

Example 2.2.1.

Consider the following matrix

$$A = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 4 & 1 \\ 3 & 5 & 2 \end{bmatrix} \tag{2.9}$$

initialize we have

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 4 & 1 \\ 3 & 5 & 2 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

For the first elimination step, since 3 large than 1 interchange rows 1 and 3 in matrices U and P.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & 5 & 2 \\ 2 & 4 & 1 \\ 1 & 4 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{2.11}$$

The first stage of elimination gives

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & 5 & 2 \\ 0 & -2/3 & -1/3 \\ 0 & 7/3 & 1/3 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{2.12}$$

The second step of elimination gives

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 5/6 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & 5 & 2 \\ 0 & 2/3 & -1/3 \\ 0 & 0 & -5/6 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{2.13}$$

Now $L * U = A$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 5/6 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 5 & 2 \\ 0 & 2/3 & -1/3 \\ 0 & 0 & 9/6 \end{bmatrix} = \begin{bmatrix} 3 & 5 & 2 \\ 2 & 4 & 1 \\ 1 & 4 & 1 \end{bmatrix} \tag{2.14}$$

Now multiply P by A we will get A in (4.3)

2.3 DOOLITTLE FACTORIZATION

The Doolittle form of LU factorization assume that the diagonal elements of matrix L are ones. Thus for three-by-three matrix A, the problem is to find matrices L and U so that $LU = A$:

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{32} & \ell_{32} & 1 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{1} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{2.15}$$

Example 2.3.1.

To find LU factorization for

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.16}$$

by Doolittle's method,we write the matrix A as following form:

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{32} & \ell_{32} & 1 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.17}$$

$$\begin{aligned} &(1)u_{11} = a_{11} = 1 \Rightarrow u_{11} = 1 \\ &(1)u_{12} = a_{12} = 2 \Rightarrow u_{12} = 2 \\ &(1)u_{13} = a_{13} = 3 \Rightarrow u_{13} = 3 \\ &\ell_{21}u_{11} = a_{31} = 2 \Rightarrow \ell_{21} = 2 \\ &\ell_{31}u_{11} = a_{21} = 3 \Rightarrow \ell_{31} = 5 \end{aligned} \tag{2.18}$$

By using these values we can find the second row of U and the second row of U:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & \ell_{32} & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.19}$$

$$\begin{aligned}(2)(2) + u_{22} = a_{22} = 20 &\Rightarrow u_{22} = 20 - 4 = 16 \\ (2)(3) + u_{23} = a_{23} = 26 &\Rightarrow u_{23} = 26 - 6 = 20 \\ (3)(2) + \ell_{32}u_{22} = a_{32} = 26 &\Rightarrow \ell_{32} = (26 - 6)/16 = 5/4\end{aligned} \tag{2.20}$$

finally,

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 5/4 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 0 & 16 & 20 \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.21}$$

$$(3)(3) + (5/4)(20) + u_{33} = 70 \Rightarrow u_{33} = 36 \tag{2.22}$$

The factorization is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 5/4 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 0 & 16 & 20 \\ 0 & 0 & 36 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.23}$$

2.4 CHOLESKY LU FACTORIZATION

For the symmetric positive definite matrix the general form, for three by three linear system, the problem is to find matrices L and U so $LU = A$; as following:

$$\begin{bmatrix} x_{11} & 0 & 0 \\ \ell_{21} & x_{22} & 0 \\ \ell_{31} & \ell_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} x_{11} & u_{12} & u_{13} \\ 0 & x_{22} & u_{23} \\ 0 & 0 & x_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{31} & a_{33} \end{bmatrix} \tag{2.24}$$

Example 2.4.1.

To solve last example by Cholesky method we write the matrix A as (4.5)

$$\begin{bmatrix} x_{11} & 0 & 0 \\ \ell_{21} & x_{22} & 0 \\ \ell_{31} & \ell_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} x_{11} & u_{12} & u_{13} \\ 0 & x_{22} & u_{23} \\ 0 & 0 & x_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.25}$$

The first stage of calculation is:

$$\begin{aligned} x_{11}x_{11} &= a_{11} = 1 \Rightarrow x_{11} = 1 \\ x_{11}u_{12} &= a_{12} = 2 \Rightarrow u_{12} = 2 \\ x_{11}u_{13} &= a_{13} = 3 \Rightarrow u_{13} = 3 \\ \ell_{21}x_{11} &= a_{21} = 2 \Rightarrow \ell_{21} = 2 \\ \ell_{31}x_{11} &= a_{31} = 3 \Rightarrow \ell_{31} = 3 \end{aligned} \tag{2.26}$$

Next, from the values computed in the first stage, the product is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & x_{22} & 0 \\ 3 & \ell_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 0 & x_{22} & u_{23} \\ 0 & 0 & x_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.27}$$

We thus compute:

$$\begin{aligned} (2)(2) + (x_{22})(x_{22}) = 20 &\Rightarrow x_{22} = (20 - 4)^{1/2} = 4 \\ (2)(3) + (x_{22})(u_{23}) = 26 &\Rightarrow u_{23} = (26 - 6)/4 = 5 \\ (3)(2) + (\ell_{32})(x_{22}) = 26 &\Rightarrow \ell_{32} = (26 - 6)/4 = 5 \end{aligned} \tag{2.28}$$

Now we find the unknown:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & x_{33} \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & x_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 20 & 26 \\ 3 & 26 & 70 \end{bmatrix} \tag{2.29}$$

$$\begin{aligned}(3)(3)+(5)(5)+(x_{33})(x_{33}) &= 70\\ x_{33}^2=(70-9-25) &= 36\\ \Rightarrow x_{33} &= 9\end{aligned} \tag{2.30}$$

Finally LU factorization is:

$$L\begin{bmatrix}1 & 0 & 0\\ 2 & 4 & 0\\ 3 & 5 & 6\end{bmatrix} * U\begin{bmatrix}1 & 2 & 3\\ 0 & 4 & 5\\ 0 & 0 & 6\end{bmatrix} = A\begin{bmatrix}1 & 2 & 3\\ 2 & 20 & 26\\ 3 & 26 & 70\end{bmatrix} \tag{2.31}$$

```
integer $i, j, k, n$
input $A$ n by n matrix
$L = I$ where $I$ is n by n identity matrix $U = A$
$P = I$
for $k = 1$ to $n - 1$ do
    $piv = abs(U(k, k))$
    $p = k$
          for$i = k + 1$ to $n$ do
             if$abs(U(i, k))$ do
              $piv = abs(U(i, k))$
              $p = i$
             end if
          end for
    if $(p > k)$
      $rch1 = U(k, :)$ ; $U(k, :) = U(p, :)$
      $(p, :) = rch1$ ; $rch2 = P(k, :)$
      $P(k, :) = P(p, :)$ ; $P(k, :) = rch2$
      for $j = 1$ to $k - 1$
          $rch3 = L(k, j)$
          $L(k, j) = L(p, j)$
          $L(p, j) = rch3$
      end for
    end if
    for $i =$ to $n$
        $s = -U(i, k)/U(k, k)$
        $U(i, :) = U(i, k) + s * U(k, :)$
        $L = (i, k) = -s$
    end for
end for
return $L; U$
```

Table 2.2: Algorithm Form for LU factorization with pivoting

```
integer $i, j, k$
$n$ is size of $A$
input $A$ n by n symmetric matrix
$L = I(n)$ where $I$ is n by n identity matrix
$U = 0(n)$
for $k = 1$ to $n$ do
    for $i = k + 1$ to $n$ do
        $U(k, k) \leftarrow A(k, k) - L(k, 1 : k - 1) * U(1 : k - 1, k)$
        for $j = k + 1$to $n$ do
                $U(k, j) \leftarrow A(k, j) - L(k, 1 : k - 1) * U(1 : k - 1, j)$
                $L(j, k) \leftarrow A(j, k) - L(j, 1 : k - 1) * U(1 : k - 1, k)/U(k, k)$
        end for
    end for
end for
return $L; U$
```

Table 2.3: Algorithm for LU factorization using Doolittle method

```
integer $i, j, k$
$n$ is size of $A$
input $A$ n by n symmetric matrix
    for $k = 1$ to $n$ do
        $a(k,k) = \sqrt{a(k,k)}$
        for $i = k+1$ to $n$ do
            $a(i,k) = a(i,k)/a(k,k)$
        end for
        for $j = k+1$ to $n$ do
            for $i = j$ to $n$ do
                $a(i,j) = a(i,j) - a(i,k) * a(j,k)$
            end for
        end for
    end for
Output $A$ is an n by n matrix
```

Table 2.4: Algorithm for Cholesky LU factorization Method

CHAPTER III

ITERATIVE METHODS

The *iterative methods* strategy produces a sequence approximate solution vectors $x^{(0)}, x^{(1)}, \ldots$ for system $Ax = b$ [7]. The numerical procedure is designed so that, in principle, the sequence of vectors converges to the actual solution. The process can be stopped when sufficient precision has been attained. The iterative methods to solve system of linear equation start with an approximation $x^{(0)} \in \mathbb{R}$ to the solution x of $Ax = b$, and general a sequence of vectors $\left\{x^{(k)}\right\}_{k=0}^{\infty}$ that converge to x. Reliable and efficient iterative methods are studied further in [8]. In a typical iterative method, we start with the system $Ax = b$ and convert it into the equivalent system $x = Cx + d$. After that, we generate successive approximations $x^{(0)}, x^{(1)}, \ldots$, where

$$x^{(k)} = Cx^{(k-1)} + d \tag{3.1}$$

In this chapter we consider three common iterative techniques for solving linear system. The Jacobi, Gauss-Seidel, and SOR methods. The basic idea is to solve the i^{th} equation in the system for i^{th} variables, in order to convert the given system (using a three by three system for illustration):

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & = & b_2 \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & = & b_3 \end{array} \tag{3.2}$$

into the system

$$\begin{array}{cccccc} x_1^{(k+1)} & = & & -a_{12}/a_{11}x_2^{(k)} & -a_{13}/a_{11}x_3^{(k)} & + & b_1/a_{11} \\ x_2^{(k+1)} & = & -a_{21}/a_{22}x_1^{(k)} & & -a_{23}/a_{22}x_3^{(k)} & + & b_2/a_{22} \\ x_3^{(k+1)} & = & -a_{31}/a_{33}x_1^{(k)} & -a_{32}/a_{33}x_2^{(k)} & & + & b_3/a_{33} \end{array} \tag{3.3}$$

3.1 JACOBI METHOD

In Jacobi method the linear system $Ax = b$ is converted into the system $x = Cx + d$. Here, the matrix C has zeros on the diagonal. Then, at each step, the vector x is updated by using all the components of x of the previous step.

Example 3.1.1.

Consider the system of equations

$$\begin{array}{ccccccc} 9x_1 & + & x_2 & + & x_3 & = & 10 \\ 2x_1 & + & 10x_2 & + & 3x_3 & = & 19 \\ 3x_1 & + & 4x_2 & + & 11x_3 & = & 0 \end{array} \tag{3.4}$$

which are converted to

$$\begin{aligned} x_1^{(1)} &= \frac{1}{9}(-x_2^{(0)} - x_3^{(0)} + 10) \\ x_2^{(1)} &= \frac{1}{10}(-2x_1^{(0)} - x_3^{(0)} + 19) \\ x_3^{(1)} &= \frac{1}{11}(-3x_1^{(0)} - 4x_2^{(0)}) \end{aligned} \tag{3.5}$$

In the matrix notation, the original system, $Ax = b$, i.e.,

$$\begin{bmatrix} 9 & 1 & 1 \\ 2 & 10 & 3 \\ 3 & 4 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 9 \\ 0 \end{bmatrix} \tag{3.6}$$

has been transformed to

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \begin{bmatrix} 0 & -1/9 & -1/9 \\ -1/5 & 0 & 1/10 \\ -3/11 & -4/11 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ x_3^{(k-1)} \end{bmatrix} + \begin{bmatrix} 10/9 \\ 19/10 \\ 0 \end{bmatrix} \tag{3.7}$$

where the iteration counter is indicated by a superscript starting with $x^{(0)} = (0, 0, 0)$ we find

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & -1/9 & -1/9 \\ -1/5 & 0 & 1/10 \\ -3/11 & -4/11 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 10/9 \\ 19/10 \\ 0 \end{bmatrix} = \begin{bmatrix} 10/9 \\ 19/10 \\ 0 \end{bmatrix} \tag{3.8}$$

for the second iteration, we have

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} 0 & -1/9 & -1/9 \\ -1/5 & 0 & 1/10 \\ -3/11 & -4/11 & 0 \end{bmatrix} \begin{bmatrix} 10/9 \\ 19/10 \\ 0 \end{bmatrix} + \begin{bmatrix} 10/9 \\ 19/10 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 1.6778 \\ -0.9939 \end{bmatrix} \tag{3.9}$$

The Jacobi method converges in 30 iterations to the $x^{(k)} = (1, 2, -1)$. In the next table number of steps for Jacobi method is given:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.0000	0.0000	0.0000
1	1.1111	1.9000	0.0000
2	0.9000	1.6778	-0.9939
$\vdots$	$\vdots$	$\vdots$	$\vdots$
30	1.0000	2.0000	-1.000

3.2 GAUSS-SEIDEL METHOD

This iterative method is a modification of the Jacobi iterative method and gives us recently calculated values. In this method each component of the vector x on the right hand side of the transformed equation is updated immediately as each iteration proses.

Example 3.2.1.

Consider the last example in Jacobi method we found the solution after 30 iteration and here we found it after just 14 iteration.In the next table number of steps for Gauss-Seidel method. We listed the solution below.

k	$x_1^{(k)}$	$x_1^{(k)}$	$x_3^{(k)}$
0	0.0000	0.0000	0.0000
1	1.0000	0.7143	01.0317
2	1.0397	1.014	-0.9895
$\vdots$	$\vdots$	$\vdots$	$\vdots$
14	1.0000	2.0000	-1.000

In the previous example it easy to see that Gauss-Seidel method converges faster than the Jacobi method.

3.3 SUCCESSIVE OVER-RELAXATION METHOD (SOR)

It is possible to modify the Gauss-Seidel method by introducing an additional parameter ω(omega),that may accelerate the convergence of the iteration. The idea is to take a combination of the previous value of $x$and the current update (from the last method). The parameter ω controls the proportion of the update that comes from the previous solution and proportion that comes from the current calculation. For $0 < \omega < 1$, the method is called successive under relaxation; For $1 < \omega < 2$, the method is called successive over relaxation (SOR). In order convert the given system (3.1) into the system

$$\begin{aligned} x_1^{(k+1)} &= (1-\omega)x_1^{(k)} + \omega/a_{11}\left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}\right) \\ x_2^{(k+1)} &= (1-\omega)x_2^{(k)} + \omega/a_{22}\left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)}\right) \\ x_3^{(k+1)} &= (1-\omega)x_3^{(k)} + \omega/a_{33}\left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)}\right) \end{aligned} \tag{3.10}$$

Example 3.3.1.

We will consider the same example in Jacobi and Gauss-Seidel methods

SOR. We start with initial $X^{(0)} = [0,0,0]^{(T)}$ and with the best $\omega = 0.9$

$$\begin{aligned} x_1^{(1)} &= (1-\omega)x_1^{(0)} + \omega/9\left(10 - x_2^{(0)} - x_3^{(0)}\right) &= 1.1111 \\ x_2^{(1)} &= (1-\omega)x_2^{(0)} + \omega/10\left(19 - 2x_1^{(0)} - 3x_3^{(0)}\right) &= 1.6778 \\ x_3^{(1)} &= (1-\omega)x_3^{(0)} + \omega/0\left(0 - 3x_1^{(0)} - 4x_2^{(0)}\right) &= -0.9131 \end{aligned} \quad (3.11)$$

The first and subsequent iterations are listed below.

k	$x_1^{(k)}$	$x_1^{(k)}$	$x_3^{(k)}$
0	0.0000	0.0000	0.0000
1	1.1111	1.6778	-0.9131
2	1.0262	1.9687	-0.9958
$\vdots$	$\vdots$	$\vdots$	$\vdots$
7	1.0000	2.0000	-1.0000

```
Integer $i, j, k$
Input $A$ is an $n$ by $m$ nonsingular matrix
$b$ is an $n$ by 1 matrix
$xold$ is an $n$ by 1 matrix the initial guess
$maxits$ is the maximum number of iterations
$\epsilon$ is the tolerance for $xold$ Initialize $n$ = length(b)
for $k = 1$ to $maxits$ do
    for $i = 1$ to $n$ do
        $sum = 0$
        for $j = 1$ to $n$ do
            $sum = sum + a[i, j] \times xold[j]$
        end for
        $xnew[i] = xold[i] + (b[i] - sum)/a[i, i]$
    end for
    if $\|xnew - xold\| \leq \epsilon$
      then return $xnew$
    end if
    for $i = 1$ to $n$ do
        $xold[i] = xnew[i]$
    end for
end for
Output $xold$ is an n x 1 matrix the Jacobi approximation to the solution of $Ax = b$
```

Table 3.1: Algorithm for Jacobi Iteration Method

```
Integer $i, j, k$
Input $A$ is an $n$ by $m$ nonsingular matrix
$b$ is an $n$ by 1 matrix
$x0$ is an $n$ by 1 matrix the initial guess
$maxits$ is the maximum number of iterations
$\epsilon$ is the tolerance for $xold$ Initialize $n = length(b)$ ; $x = zeros(n, 1)$
$Y = zeros(n, 1)$ ;$Y = x0$
for $k = 1$ to $maxit + 1$ do
    for $i = 1$ to $n$ do
        $S \leftarrow 0$
        for $j = 1$ to $i - 1$ do
            $S \leftarrow S + A(i, j) * x(j)$
        end for
        for $j = i + 1$ to $n$ do
            $S = S + A(i, j) * x0(j)$
        end for
        if $A(i, i) == 0$
          break
        end if
        $x(i) \leftarrow (-S + b(i))/A(i, i)$
        end for
    $error \leftarrow \|(norm(x - x0))\|$
    $rerr = error/(norm(x) + eps)$
    $x0 \leftarrow x$
    $Y \leftarrow [Yx]$
    if $rerr < \epsilon$
      break
    end if
end for
output $x$
end algorithm
```

Table 3.2: Algorithm for Gauss-Seidel Iteration Method.

```
Integer $i, j, k$
Input $A$ is an $n$ by $m$ nonsingular matrix
$b$ is an $n$ by 1 matrix
$x0$ is an $n$ by 1 matrix the initial guess
$maxits$ is the maximum number of iterations
$\epsilon$ is the tolerance for $xold$
$\omega$ is the relaxation scalar, between 0 and 2
for $i = 1$ to $n$ do
    $C(i, i) \leftarrow 0$
end for
for $i = 1$ to $n$ do
    $C(i, 1 : n) \leftarrow C(i, 1 : n)/A(i, i)$
    $r(i, 1) \leftarrow b(i)/A(i, i)$
end for
while $(i <= max_i tr)$ do
        $xold \leftarrow x$
        for $k = 1$ to $n$ do
            $x(k) \leftarrow (1 - w) * xold(k) + w * (C(k, :) * x + r(k))$
        end for
        if $norm(xold - x) <= \epsilon$
        output SOR method is converged
                return $x$ the vector of solution
        end if
        $i \leftarrow i + 1$
end while
Output SOR method did not converged
$x$ the vector of solution
End algorithm
```

Table 3.3: Algorithm for SOR Iteration Method.

CHAPTER IV

SPARSE MATRICES

A sparse matrix is a matrix handled by special techniques making use of the large number of zero entries. This definition will helps to define how many zeros a matrix needs in order to be sparse. The answer is that it depends on the structure of the matrix, and how we can use it. In many applications it is common to deal with very large matrices where only a few coefficients are not zero. In such cases, memory requirements can be decreased and performance increased by using a specialized representation storing only those coefficients. In next equation there is (matrix A) which shown a small example for sparse matrix.

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 & a_{17} & 0 \\ 0 & a_{22} & 0 & 0 & a_{24} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 & 0 & 0 & a_{38} \\ 0 & 0 & 0 & a_{44} & 0 & 0 & 0 & 0 \\ 0 & a_{52} & 0 & 0 & a_{55} & 0 & 0 & 0 \\ a_{61} & 0 & 0 & 0 & 0 & a_{66} & 0 & 0 \\ 0 & 0 & a_{73} & 0 & 0 & 0 & a_{77} & 0 \\ 0 & 0 & 0 & a_{84} & 0 & a_{86} & 0 & a_{88} \end{bmatrix} \tag{4.1}$$

4.1 STORAGE SCHEME FOR SPARSE MATRIX

There are different storage schemes for sparse matrices. The main goal from these storage is to represent only the nonzero elements, and do the matrix multiplications with fewer arithmetic operations.

4.2 COORDINATE FORMAT STORAGE

This method of storage is the simplest storage scheme method. It is also flexible. It is often the most basic format in sparse matrix applications. Here we have three arrays:

1. AA is a real array values of the nonzero elements of A in any order.
2. SR is an integer array containing there row indices.
3. SC is the second integer array containing their column indices.

The length of each array is equal to the number of nonzero elements.

Example 4.2.1.

The matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 4 & 0 & 0 & 13 & 0 & 0 & 0 \\ 22 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 18 & 2 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 12 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 14 & 8 & 0 & 0 \\ 20 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 & 19 \end{bmatrix} \tag{4.2}$$

by coordinate format method all three arrays are of length Nz and will be storage as

$$\begin{aligned} AA &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 19 & 6 & 20 & 14 & 8 & 18 & 12 & 7 & 3 & 2 & 5 & 11 & 13 & 1 & 15 & 3 & 4 & 2 \\ \hline \end{array} \\ SR &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 8 & 8 & 7 & 6 & 6 & 4 & 5 & 7 & 3 & 4 & 4 & 1 & 2 & 1 & 6 & 3 & 2 & 4 \\ \hline \end{array} \\ SC &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 8 & 2 & 1 & 5 & 6 & 4 & 5 & 7 & 3 & 5 & 1 & 8 & 5 & 1 & 3 & 3 & 2 & 5 \\ \hline \end{array} \end{aligned} \tag{4.3}$$

4.3 COMPRESSED SPARSE ROW (CSR) FORMAT

This format is used very frequently for storing general sparse matrices.
For typical computations, it is better than the coordinate format. We again construct three arrays: [7]

1. A real array AA contains the real values a_{ij} stored row by row, from row 1 to n.
2. An integer array JA contains the column indices of the elements a_{ij} as stored in array AA.
3. An integer array IA contains the pointers to the beginning of the each row in the arrays AA and JA.

The length of AA and JA are Nz while the length of IA is $n+1$ with $IA((n+1)$. Thus, the above matrix may be stored as follows:

$$
\begin{array}{lcl}
AA & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 11 & 4 & 13 & 22 & 3 & 5 & 18 & 2 & 9 & 12 & 15 & 14 & 8 & 20 & 7 & 6 & 19 \\ \hline\end{array} \\
JA & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 8 & 2 & 5 & 1 & 3 & 1 & 4 & 5 & 2 & 5 & 3 & 5 & 6 & 1 & 7 & 2 & 8 \\ \hline\end{array} \\
IA & = & \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline 1 & 3 & 5 & 7 & 10 & 12 & 15 & 17 & 19 \\ \hline\end{array}
\end{array}
\tag{4.4}
$$

There are a number of variations for the (CSR) format. The most obvious variation is storage the columns instead of the rows.

4.4 MODIFIED SPARSE ROW (MSR) FORMAT

Many sparse matrices have nonzero diagonals. Also, these elements are accessed more frequently. Therefore it may be advantageous to store them separately. The (MSR) format has just two arrays:

1. AA is a real array.

2. JA an integer array.

The first n positions in AA contain the diagonal elements of the matrix in order. The unused last position of the array AA may hold information about the matrix.

Starting at position $n+2$, the nonzero entries of AA, except the diagonal elements, are stored by row. For each element $AA(k)$, the integer $JA(k)$ represents its column index on the matrix. The $n+1$ first positions of JA contain the pointer to the beginning of each row in AA and JA. The arrays for the above above example will be:[7]

$$\begin{aligned} AA &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 4 & 3 & 18 & 12 & 8 & 7 & 18 & * & 11 & 13 & 22 & 5 & 2 & 9 & 15 & 8 & 20 & 6 \\ \hline \end{array} \\ JA &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 10 & 14 & 18 & 25 & 28 & 33 & 41 & 44 & 44 & 8 & 5 & 1 & 1 & 5 & 2 & 3 & 6 & 2 \\ \hline \end{array} \end{aligned} \tag{4.5}$$

4.5 SPARSE MATRICES MULTIPLICATION

In this section we will show how can the storage sparse matrix formats helps in sparse matrices multiplication.

When we multiply two sparse matrices normally, we perform many unnecessary operations. There are a lot of multiplications and additions by zero which do not contribute to the result but nevertheless take time and occupy space in memory. But if the matrices are kept in sparse format, these operations are eliminated.

The disadvantage of sparse format is increased time for indexing and searching.

```
integer $i, j$
input $A$ n by m sparse matrix
Initialize empty vectors $AA$, $JA$
counter=1
    for $i = 1$ to $n$ do
        for $j = 1$ to $m$ and if $i, j \neq 0$ do
            compute $AA \leftarrow [AA \quad a_{ij}]$
            compute $JA \leftarrow [JA \quad j]$
            counter=counter+1
        end for
        IA(i+1)=counter
    end for
```

Table 4.1: Algorithm Form for CSR format

```
Integer $i, j, k$
Input
$AA$ the real values $a_{ij}$ stored row by row, from row 1 to $n$
$JA$ the column indices of the elements $a_{ij}$ as stored in array $AA$
$IA$ contains the pointers to the beginning of the each row in the arrays $AA$ and $JA$
$AB$ the real values $b_{ij}$ stored row by row, from row 1 to $n$
$JB$ the column indices of the elements $b_{ij}$
$IB$ contains the pointers to the beginning of the each row in the arrays $AB$ and $JB$
Let $n \leftarrow size(IA, 2) - 1$
Initialize $C \leftarrow zeros(n)$
        for $i = 1$ to $n$ do
            $k \leftarrow IA(i)$ to $IA(i+1) - 1$ ; $m \leftarrow size(k, 2)$
            $r \leftarrow AA(k)$ ; $c \leftarrow JA(k)$
            for $j = 1$ to $n$ do
                $z \leftarrow zeros(1, m)$
                for $p = 1$ to $m$ do
                    for $q = IB(c(p))$ to $IB(c(p)+1) - 1$ do
                        if $JB(q) == j$
                            $z(p) \leftarrow AB(q)$
                        end if
                    end for
                end for
                $C(i, j) \leftarrow sum(r. * z)$
            end for
        end for
output $C$ sparse matrix result
```

Table 4.2: Algorithm Form for MSR format

CHAPTER V

SPARSE MATRIX REORDERING

In this section, we will describe some important concepts in matrix ordering. Then, we will present and discuss the implementation of one of the most commonly used ordering algorithms known as Cuthill-McKee Ordering. Bandwidth reduction is presented for a variety of practical problems in [6].

5.1 FILL-IN

Consider the matrix A where

$$A = \begin{bmatrix} 8 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 20 & 3 & 8 & 0 & 0 & 0 & 0 \\ 0 & 3 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 15 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 17 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 11 \end{bmatrix} \tag{5.1}$$

LU factorization gives:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3750 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1589 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.4238 & -0.1949 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4444 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0000 \end{bmatrix} \tag{5.2}$$

$$U = \begin{bmatrix} 8 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18.8750 & 3 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.5232 & -1.2715 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 11.3614 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 15.2222 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8.5 \end{bmatrix} \tag{5.3}$$

After reordering the matrix A and doing factorization we will obtain L and U as:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.375 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1589 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.4238 & -0.1949 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4444 & 1 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1 \end{bmatrix} \tag{5.4}$$

$$U = \begin{bmatrix} 8 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18.8750 & 3 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.5232 & -1.2715 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 11.3614 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 4 & 0 & 0 \\ 0 & 0 & 0 & & 0 & 15.2222 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8.5 \end{bmatrix} \tag{5.5}$$

5.2 SYMMETRIC REORDERING

If we reorder the variables and equations of a linear system, it is still the same system. But this type of reordering may make the system easier to solve in the sense of doing fewer operations for solution.

For example

$$\begin{aligned} 3x_1 + 2x_2 - x_3 &= 1 \\ x_1 + 5x_2 + 4x_3 &= -4 \\ 6x_1 - 4x_2 + 7x_3 &= 10 \end{aligned} \tag{5.6}$$

can be rewritten as

$$\begin{aligned} 7y_1 + 6y_2 - 4y_3 &= 10 \\ -y_1 + 3y_2 + 2y_3 &= 1 \\ 4y_1 + y_2 + 5y_3 &= -4 \end{aligned} \tag{5.7}$$

where

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_1 \\ x_2 \end{bmatrix} \tag{5.8}$$

In general, we can express this symmetric reordering as follows: Let P be a permutation matrix of appropriate order and P^T its transpose. Then for an $n \times n$ linear system

$$Ax = c \tag{5.9}$$

$$PAP^T\,Px = Pc \tag{5.10}$$

$$By = d \tag{5.11}$$

where $B = PAP^T$, $y = Px$ and $d = Pc$.

5.3 CUTHILL-MCKEE ORDERING

Cuthill-McKee Ordering is one of the most common reordering algorithms. [1] The algorithm works on sparse symmetric matrices. We consider the matrix as the adjacency matrix of a graph and relabel the vertices. The result is a matrix with reduced bandwidth.

We start with the linear system

$$Ax = b \tag{5.12}$$

where A is an N by N sparse symmetric, positive definite matrix

Example 5.3.1.

$$A = \begin{bmatrix} x & 0 & 0 & 0 & x & 0 & 0 & x \\ 0 & x & 0 & 0 & 0 & 0 & 0 & x \\ 0 & 0 & x & 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & x & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & x & 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & 0 \\ x & x & 0 & 0 & 0 & 0 & 0 & x \end{bmatrix} \tag{5.13}$$

The bandwidth in this example is large we will try to decrease it.

Now we will find the relation between the rows, we will lock to the nodes which lies between every row with the other rows. In the next table there is the relation between $N(R)$, row number and and $N(C)$, the column numbers where it has a nonzero entry.

N(R)	N(C)
1	5,8
2	8
3	6
4	0
5	1
6	7,3
7	6
8	1,2

Table 5.1: The relation nodes

N(R)	1	2	3	4	5	6	7	8
C(R)	4	5	1	8	2	3	6	7

Table 5.2: Rows with changing rows

After that we generate the permutation matrix P as follows: Take the row with minimum number of connections and make it the first row. Then, among its neighbors, again choose the row with minimum number and make it the second. Then we repeat this step until getting of the P matrix. The next table shows the rows from 1 to 8 and the changing rows

Now the P matrix will be as following:

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.14}$$

Now we make the operation PAP^T we get the matrix

$$PAP^T = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x \end{bmatrix} \tag{5.15}$$

If we see the bandwidth in (5,13) and the bandwidth in (5,15) after this operation we will see clearly it is decreased.

The Cuthill-Mckee method take advantage from the zeros in large sparse matrices and make them easy to solve in the other systems.

We count the multiplication operations to reduce the sparse symmetric matrix A to diagonal from using Gaussian Elimination.

The ratio of the number of operations after Cuthill-Mckee reordering and before that are given below.

The numbers less than 1 indicate the after Cuthill-Mckee we did fever operations.

Each case is tested with random matrices 1000 times and average values are obtained.

Test matrices were produced by our special program using MATLAB's random number generator.

The results clearly show that Cuthill-McKee reordering is advantageous only in certain cases.

In the table, N is the size of matrix and r denotes the ratio of nonzero elements

```
Integer $i, j$
input $A$ n by m sparse matrix
Initialize the matrix $B$ =zeros(m)
          for $i = 1$ to $n$ do
                    $connected \leftarrow 0$
                    for $j = 1$ to $m$
                              if $i \neq j$
                                        if $A(i, j) \neq 0$
                                                  $connected \leftarrow connected + 1$
                                                  $B(i, connected) \leftarrow j$
                                        end if
                              end if
                    end for
                    $B(i, m) \leftarrow connected$
          end for
Let $level = -1$ ; $cnter = 0$ ; $memo = []$
$nodelevel \leftarrow info \times ones(1, m)$
$neighbors \leftarrow B(:, m)$
          while $cnter < m$
                    $[nodesindex] \leftarrow min(neighbors)$
                    $nodelevel(index) \leftarrow level + 1$
                    $memo \leftarrow [memoindex]$
                              while $size(memo, 2) > 0$
                                        $index \leftarrow memo(1)$
                                        for $i = 1$ to $B(index, m)$
                                                  $level \leftarrow nodelevel(index) + 1$
                                                  $num \leftarrow B(index, i)$
                                                  if $nodelevel(num) > level$
                                                            $nodelevel(num) \leftarrow level$
                                                            $memo \leftarrow [memonum]$
                                                  end if
                                        end for
```

Table 5.3: Algorithm Form for Cuthill-Mckee Reordering - Part I

```
                $memo(1) \leftarrow [\ ]$
                $neighbors(index) \leftarrow$
                $cntr \leftarrow cntr + 1$
            end while
    end while
```

Table 5.4: Algorithm Form for Cuthill-Mckee Reordering - Part II

$N \setminus r$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$	$\frac{1}{2048}$
16	1.00	1.14	0.89	0.67	0.58	0.78	0.90	0.96	0.99	0.99
32	1.02	1.13	1.40	1.31	0.63	0.46	0.57	0.73	0.84	0.97
64	1.01	1.08	1.24	1.66	1.94	0.61	0.37	0.34	0.50	0.72
128	1.00	1.04	1.12	1.33	1.92	2.54	0.78	0.33	0.24	0.28
256	1.00	1.02	1.06	1.17	1.39	2.08	3.67	0.92	0.31	0.20

Table 5.5: Improvement After Cuthill-Mckee Reordering

```
Integer $i$
Input $N$ the Matrix size
$r$ is the ratio of zero elements of the symmetric sparse matrix
$m$ is the number of times operation is repeated
Initialize$v = zeros(m, 1)$
        for $i = 1$ to $m$ do
            $A1 = TestSparseMatrix(N, r)$
            $A2 = (A1 + A1_T)/2$
            $c1 = GaussElim3(A2)$
            $A3 = cuthill(A2)$
            $c2 = GaussElim3(A3)$
            $v(i) = c2/c1$
        end for
return $ratio = sum(v)/m$
```

Table 5.6: Algorithm Form for ratio of operations

```
Integer $i, j$
Input $A = zeros(N)$
      for $i = 1$ to $N$ do
         for $j = 1$ to $N$ do
            $r = rand(1)$
            if $r < cut$
               $A(i, j) = r$
            end if
         end for
      end for
      for $i = 1$ to $N$ do
         $A(i, i) = 1$
      end for
Return $A$ random sparse matrix
```

Table 5.7: Algorithm Form for finding random sparse matrix

CHAPTER VI

PRECONDITIONING

Preconditioning is a technique consisting in preconditioning the system that we want to solve, and in our case is $Ax = b$, so that the new system has better properties (e.g. faster convergence to the solution of numerical methods, less cost of computation,...). The solution to the original problem can be easily constructed from the solution of the preconditioned problem. Usually the main aim of preconditioning is to reduce the condition number of the problem.

$$M^{-1}Ax = M^{-1}b \tag{6.1}$$

The preconditioner M is chosen in such a way that the iterative method we are using converges rapidly. Incomplete matrix factorizations can be used if there are no natural preconditioners[7].

6.1 ITERATIVE METHODS PRECONDITIONER

Starting by the linear system

$$Ax = b \tag{6.2}$$

The main idea is to modify components of an approximate vector solution to improve the accuracy of iterations. One example is to annihilate some component(s) of the residual vector $b - Ax$ [7]. We begin with the decomposition
$A = D - E - F$
where D is diagonal of A, $-E$ its strict lower part, and $-F$ its strict upper part
we can find $-E$ and $-F$ by the following algorithm:

```
integer $i, j$
input $A$ real matrix $n$ by $m$ size
initialize the matrix $E = A$
    for $i = 1$ to $n$
      for $j = i$ to $n$
        $E(i, j) \leftarrow 0$
      end for
    end for
    for $i = 1$ to $m$ do
      for $j = 1$ to $i$ do
        $F(i, j) \leftarrow 0$
        end for
      end for
return $F \leftarrow -F$
return $E \leftarrow -E$
```

Table 6.1: Algorithm to find the matrices E and F

Here we will consider the most common preconditioner used for solving a large sparse matrices and compares their performance.

For solving a linear system $Ax = b$ a fixed point iteration takes the form

$$x_{k+1} = M^{-1}N_{xk} + M^{-1}b \tag{6.3}$$

where M and N are splitting of A into

$$A = M - N \tag{6.4}$$

6.2 PRECONDITION AND ITERATION MATRICES

The local form of Jacobi and Gauss-Seidel iterations will be as:

$$x_{k+1} = Gx_k + f \tag{6.5}$$

which

$$G_{JA}(A) = I - D^{-1}A \tag{6.6}$$

for the Jacobi and Gauss-Seidel iterations, respectively. Moreover, given the matrix splitting in (6,4) which can be defined by the relation:

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b \tag{6.7}$$

and it has the form (6,4) with

$$G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A, f = M^{-1}b. \tag{6.8}$$

This iteration, $G_{JA}(A) = I - D^{-1}A$ can be shown as at a technique for solving the system

$$(I - G)x = f \tag{6.9}$$

Since G has the form $G = I - M^{-1}A$, this system can be rewritten as

$$M^{-1}Ax = M^{-1}b \tag{6.10}$$

The system $M^{-1}Ax = M^{-1}b$ has the same solution as the original system and it is called a preconditioned system and M is the preconditioning matrix or preconditioner. In other words, a relaxation scheme and a fixed-point iteration on a preconditioned system are equivalent. Comparison of various preconditioning schemes can be found in [4, 9].

6.3 JACOBI PRECONDITIONER

The Jacobi preconditioner is simply the diagonal of A

$$M_{JA} = D \tag{6.11}$$

The next Algorithm will show the Jacobi precondition for solving vector x as:

$$x_{k+1} = M^{-1}N_{xk} + M^{-1}b \tag{6.12}$$

```
integer $i$ ; $max$ maximum number of iterations ;$tol$ Real error tolerance
input $A$ is the coefficient sparse matrix
input $b$ is the right-hand side column vector
input $x0$ initial vector with zeros
$M = diag(A) = D$
$N = M - A$
$invm = inverse(M)$
$G = invm * N$
$f = invm * b$
for $i = 1$ to $max$ do
    $xnew \leftarrow G * x0 + f$
    if $norm(xnew - xold) < tol$
       break
       end if
    $x0 \leftarrow xnew$
end for
return x
```

Table 6.2: Algorithm Form for Jacobi precondition method

6.4 GAUSS-SEIDEL PRECONDITION

The Gauss-Seidel preconditioner is the lower triangular part of A

$$M_{GS} = D - E \tag{6.13}$$

```
integer $i$ ; $max$ maximum number of iterations ;$tol$ Real error tolerance
input $A$ is the coefficient sparse matrix
input $b$ is the right-hand side column vector
input $x0$ initial vector with zeros
input $E$ the strict Lower of (A)
input $F$ the strict upper of (A)
compute $D = A + E + F$ ; $M = D - E$ ; $N = M - A$
let invm=inverse of (M)
compute $G \leftarrow invm * N$
compute $f = invm * b$
        for $i = 1$ to $max$
            $xnew \leftarrow G * x0 + f$
            if $norm(xnew - x0) < tol$
              break
        end if
            $x0 \leftarrow xnew$
        end for
        return x0
```

Table 6.3: Algorithm for Jacobi Precondition Method

CONCLUSION

In this thesis we have made an analysis of iterative methods of solution for linear systems. We have implemented the well-known algorithms using MATLAB. These implementations are given in the appendix.

In particular, sparse linear systems deserve special attention. We have considered how Cuthill-McKee algorithm reduces the bandwidth of a sparse matrix by obtaining a suitable permutation matrix. Preconditioning a sparse system reduces memory and time requirements of the solution process.

The number of operations required by the Method of Gaussian Elimination before and after using reordering by Cuthill-McKee are given in table 5.5. This comparison shows that the advantage of reordering is dependent on matrix size and also the ratio of nonzero elements.

We wish to continue further work in the direction of solution of large, sparse systems on parallel multiprocessors.

REFERENCES

[1] **E. Cuthill and J. Mckee** reducing bandwidth of sparse symmetric matrices, *proceedings of ACM 69* (1969) 157 - 172.

[2] **L.V. Fausett**, Applied Numerical Analysis Using MATLAB, Prentice Hall, New Jersey, 1999.

[3] **A. Al-Kurdi, D.R. Kincaid**, *LU-decomposition with iterative refinement for solving sparse linear systems*, Journal of Computational and Applied Mathematics 185 (2006) 391 - 403.

[4] **W. Li**, Comparison results for solving preconditioned linear systems, *Journal of Computational and Applied Mathematics* 176 (2005) 319 - 329.

[5] **R.C. Mittal and A. Al-Kurdi**, LU-Decomposition and Numerical Structure for Solving Large Sparse Nonsymmetric Linear Systems, *Computers and mathematics with Applications*, 43 (2002) 131 - 155.

[6] **J.K. Reid and J.A Scott**, Reducing the total bandwidth of a sparse unsymmetric matrix, *Council for the Central Laboratory of the Research Councils* (2005) RAL-TR-2005-001.

[7] **Y. Saad**, Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[8] **H.A. van der Vorst**, Efficient and reliable iterative methods for linear systems, *Journal of Computational and Applied Mathematics*, 149 (2002) 251 - 265.

[9] **X.Z. Wang, T.Z. Huang, Y.D. Fu**, Comparison results on preconditioned SOR-type iterative method for Z-matrices linear systems *Journal of Computational and Applied Mathematics* 206 (2007) 726 - 732.

APPENDIX A

MATLAB CODES

Gaussian Elimination

```
function cozum = GaussElim(A,b)
%This program solves a linear system of equations using
%the method of Gaussian elimination with back substitution
%Usage: GaussElim(A,b) where A is an nxn coefficient matrix.
[n n2]=size(A);
if n~=n2
error('The coefficient matrix must be square')
end
[n3 n4]=size(b);
if n3~=n || n4~=1
error('The column vector b must have the same number of rows as A')
end
A=[A b];
operationcount=0;
for i=1:n-1
    if A(i,i)==0
        [mxm indis]=max(abs(A(i:n,i)));
        A=interchange(A,i,indis+i-1);
    end
    %for j=i+1:n
    for j=i+1:min(i+3,n)
        carpan=A(j,i)/A(i,i);
        for k=i:n+1
            A(j,k)=A(j,k)-carpan*A(i,k);
            operationcount=operationcount+1;
        end
    end
end
operationcount
d=min(abs(diag(A)));
if d<10^-15
error('The system does not have unique solution')
end
cozum=zeros(n,1);
cozum(n)=A(n,n+1)/A(n,n);
for i=n-1:-1:1
topla=0;
for j=i+1:n
topla=topla+A(i,j)*cozum(j);
end
cozum(i)=(A(i,n+1)-topla)/A(i,i);
end
```

LU Factorization Using Gaussian Elimination

```
function [L,U]=lu_factor(A)
[row,col]=size(A);
U=A;
L=eye(row);
for j=1:row
    for i=j+1:row
        L(i,j)=U(i,j)/U(j,j);
        U(i,:)=U(i,:)-L(i,j)*U(j,:);
    end
end
```

Cuthill-McKee

```
function P=cuthill(A)

[m,n]=size(A);
if m~=n
    error('Input must be a square matrix')
end

B=zeros(m);
for i=1:m
    connected=0;
    for j=1:m
        if i~=j
            if A(i,j)~=0
                connected=connected+1;
                B(i,connected)=j;
            end
        end
    end
    B(i,m)=connected;
end
level=-1;
cntr=0;
nodelevel=inf*ones(1,m);
memo=[];
neighbors=B(:,m);
while cntr<m
[nodes index]=min(neighbors);
nodelevel(index)=level+1;
memo=[memo index];
    while size(memo,2)>0
        index=memo(1);
        for i=1:B(index,m);
```

```
            level=nodelevel(index)+1;
            num=B(index,i);
            if nodelevel(num)>level;
                nodelevel(num)=level;
                memo=[memo num];
            end
        end
        memo(1)=[];
        neighbors(index)=inf;
        cntr=cntr+1;
    end
end

P=zeros(m);
for i=1:m
    [a k]=min(nodelevel);
    nodelevel(k)=inf;
    P(i,k)=1;
end
end
```

CSR Format

```
function CSR(A)
[n,n]=size(A);
IA=(zeros);
for i=1:n
    k1=IA(i);
    k2=IA(i+1);
    y(i)=dot(A(k1:k2),x(jA(k1:k2)));
end
```

Jacobi Precondition Method

```
function sol=Amn(A,b,xold,max,tol)
[row,col]=size(A);
M=diag(diag(A));
N=M-A;
invm=inv(M);
G=invm*N;
f=invm*b;

for i=1:max
    xnew=G*xold+f;
    i
    if norm(xnew-xold)<tol
        break
    end
    xold=xnew;
end
sol=xnew;
end
```

APPENDIX B

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: ABOSHARB, Laila
Nationality: Libya
Date and Place of Birth: 6 February 1979, Trablus, Libya
Marital Status: Married
Phone: 0(507) 998 30 16
e-mail: lailaabusharb@yahoo.com

EDUCATION

Degree	Institution	Year of Graduation
BS	Aljabal Algharbi University Mathematics and Computer Science Yefren, Libya	2000
High School	Khadije Al Kubra, Yefren, Libya	1996

FOREIGN LANGUAGES

Advanced English.

Advanced Turkish.

HOBBIES

Traveling, Writing.