Direct Methods and Iterative Methods for solving Linear Systems

Name: Mengqi Qiu

Course: Mathematics

Department of Mathematics,
Brunel University London

Supervisor: Dr Mike Weaby

Year: 2017/8

April 4, 2018

Contents

Abstract

In this brief thesis, a detailed discussion is made of Linear systems of equations and their matrix representations. In order to explore its **Implementation** and **Performance** to help us know which method to implement for large n matrices mainly on computer. In particular, properties of matrices like, norms, factorizations are given sufficient treatment and explanation for revision purposes. We also discuss the solution of linear systems in general, leading us to discussion of the least squares method, the normal equations, the concept of pseudo-inverses. This eventually leads us to consider the system of square matrices in further detail, whence we discuss about the Gaussian elimination and LU Factorization, Gaussian elimination with pivoting and the Cholesky method. We also discuss the computational aspects associated with solving such systems, typically stability and accuracy of solving and computational cost. Discussion is also made of solution by direct methods and iterative methods, where we discuss suitability and stability of Gauss-Siedel, Jacobi, Steepest Descent, Method of Conjugate directions, Method of Conjugate Gradients and finally the Method of Preconditioned Conjugate Gradients. Several algorithms are discussed, and further, several illustrations are also given. Possible directions of further work is pointed out. Proofs of some theorems and propositions are given and several others are referred to their original sources. This is to give you, the reader, an effective introduction into the topic regarding the implementation and the performance of direct and iterative methods. Hence equipping you with information to tackle them.

Acknowledgement

I wish to thank the supervisor, Dr. Mike Warby for guidance throughout the preparation of the thesis, particularly, in pointing out several errors and mistakes and also suggesting ways to improve the work. His guidance was crucial right from the beginning in the selection of references to the end of finally making the manuscript ready.

Chapter 1

Introduction

[1](pp.1-32)

A system of m linear equations with n unknowns, where m, n need not be necessarily equal, can be represented using the matrices in the form of a matrix equation as

$$Ax = b \tag{1.1}$$

Where A represents the entries of all the coefficients of the equations in the form of an $m \times n$ matrix, x represents the variables in the form of an $n \times 1$ column vector and b represents the $m \times 1$ column vector for the constants on the RHS of the system of linear equations. Many practical problems could be reduced to solving a linear system of equations formulated as matrix form.

Though the above said equation seems to be simple to state and understand, but practical experience shows that there are several complications associated with the problem. We are considering the case here when n is large and all computations are done on a computer.

The field of linear algebra, and, in particular, numerical linear algebra, is a vast field of ongoing research. In this sequel, we concentrate on mainly the numerical aspects of solving systems. The problem mainly comes from the fact of using computers, which work on an arithmetic quite different from the one that we are acquainted with. Methods of solving systems can be stable or unstable depending on the procedure involved. In addition, more stable methods may be, at times, more time consuming(high computational cost).

In this thesis, though we discuss general system of equations, our focus will be square nonsingular systems. This is because, as will be seen, in a general system of equations, when there are more equations than unknowns, i.e. $m > n$ there is usually no vector x satisfying $Ax = b$ but in this case it is common to consider what is called the "least squares solution of $Ax = b$" which involves finding the vector x which minimises the 2-norm of $Ax - b$. We will see that the general

system of equations, can be reduced by the principle of least squares to a system of square matrices.

We will focus on both direct and iterative method for solving linear systems. Though direct methods are accurate and stable, for large sparse(many zero entries in matrices), the computational cost is considerably reduced by iterative methods. Thus, the discussion will include the direct methods like Gaussian elimination and Cholesky factorization and also the theory of iterative methods like Gauss-Siedel, Jacobi, Steepest Descent, Conjugate Directions, Conjugate Gradients, preconditioned Conjugate Gradients and knowing which one of the iterative methods are faster. We explore the link between computers and the concept of stability of an algorithm also. We hope this study makes enables you to be able to decide on whether to use direct methods or iterative methods and also give you the ability to be able to possibly know a bit about the conditions and stability pertaining to its computations before deciding the method to solve a given set of questions. Effort was made in understanding the material after reading from different sources and writing in my own words in an attempt to aid you, the reader,in understanding this topic. Overall you will notice that the main theme of the thesis is about implementation and performance of both direct and iterative methods on computers. Mainly the case when n is too large. This is the main reason why all these topics such as stability, conditioning etc have been chosen. The main reference throughout the thesis is the book by Trefethen and Bau(Numerical Linear Algebra) [1]

Chapter 2

Matrices-Preliminaries

This chapter is important because the definitions here will appear many times in this thesis. We briefly therefore give here a revision of the definition of these terms together with some of the basic properties:

Definition 1. The *determinant* of an $n \times n$ matrix A, denoted by $det(A)$ is defined as

$$det(A) := \sum_{\sigma} sgn(\sigma) a_{1\sigma(1)} a_{1\sigma(2)} ... a_{n\sigma(n)}$$

where the summation is over all the permutations $\sigma(1)$,..., $\sigma(n)$ of $1, 2, ..., n$ and $sgn(\sigma)$ is the signature of the permutation that is 1 or -1 as σ is even or odd respectively. [2]

Definition 2. The minor of a matrix is the determinant of submatrix formed by removing one or more of its rows and columns.

Definition 3. The transpose of the co-factor matrix of a matrix or the adjugate of the matrix, $adj(A)$ is the matrix of its first minors corresponding to each element multiplied by $(-1)^{i+j}$.

Definition 4. The inverse of a matrix, if it exists is unique and determined by the formula

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

Thus, it is clear that the inverse of a matrix exists iff $det(A) \neq 0$

Definition 5. A vector x is said to be an *eigenvector* corresponding to a scalar λ iff $Ax = \lambda x$

Note that product and sum of eigenvalues are the determinant and trace(sum of diagonals) respectively.

Definition 6. The *rank* of a matrix A is defined as the size of the largest submatrix of A such that the corresponding minor is non zero of A. [2]

Definition 7. The *rank* of a matrix A is also defined to be the number of independent rows or columns of the matrix, i.e., the largest n such that $c_1C_1 + c_2C_2 + \ldots + c_nC_n = 0 \implies c_1, \ldots, c_n = 0$, where $c_1, c_2 \ldots, c_n$ are scalars and $C_1, C_2, \ldots, C_n$ are column vectors.

For getting the unique solutions to a system of linear equations, the determinant of the matrix A needs to be non-zero or, equivalently, the rank of the matrix A need to be full, i.e.,n.

Definition 8. An $n \times n$ matrix A is said to be *symmetric* matrix, whenever $A = A^T$ that is, whenever, $a_{ij} = a_{ji}$.

Definition 9. An $n \times n$ matrix A over the field of complex numbers, $\mathbb{C}$ is said to be hermitian if $A^* = A$ where * denotes conjugate transpose or adjoint, i.e., $A^* = \overline{a_{ji}}$

Note that by the spectral theorem, all eigenvalues of a hermitian matrix are real.

Definition 10. A matrix is positive definite if it is symmetric(hermitian) and all its eigenvalues are positive and negative definite if all its eigenvalues are negative. Similarly, a hermitian matrix is positive semidefinite if all its eigenvalues are non-negative and negative semidefinite if all its eigenvalues are non-positive.

As its a symmetric(hemitian) matrix all the eigenvalues are real, so it makes sense to talk about them being positive or negative.

Definition 11. The inner product of two n-dimensional vectors or columns of a matrix x, y, denoted by $\langle x, y \rangle$ or x^*y is

$$x^*y = \sum_{i=1}^{n} \bar{x_i} y_i$$

where summation is done over the components of product of vectors.

Definition 12. Two vectors x, y are orthogonal iff $x * y = 0$

A set of mutually orthogonal sets are linearly independent and thus form a basis of the vector space in which they lie.

Definition 13. A complex $n \times n$ matrix is unitary iff $A^*A = I$. If the matrices be real, they are said to be orthogonal

Definition 14. A norm of a vector or a matrix, $\|x\|$ or $\|A\|$, is a real valued function from the space of n-dimensional vectors or matrices satisfying

i)(positivity)

$$\|x\| \geq 0 \ \|x\| = 0 \ iff \ x = 0$$

ii)(triangle inequality)

$$\|x + y\| \leq \|x\| + \|y\|$$

iii)(scalability)

$$\|kx\| = |k|\|x\|$$

The notions of convergence and approximations are studied using norms.

The various vector norms are [1](pp.3-11):

i) l_1 or (1-norm)-

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

ii)l_2 or (2-norm)-

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} = \sqrt{x^* x}$$

iii)l_p or (p-norm)-

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} = \sqrt{x^* x}$$

iv)l_∞ norm-

$$\|x\|_\infty = max_{1 \leq n} |x_i|$$

Note that the vectors x are of dimension n in the above definitions

The matrix norms induced by vector norms are: $\|A\|_p$ given by:

$$\|A\|_p = \sup_{x \in \mathbb{C}^n \, x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \sup_{x \in \mathbb{C}^n \, \|x\|_p = 1} \|Ax\|_p$$

Note that the n norm on a diagonal matrix D induced by a vector is equal to

$$\|D\|_n = max_{1 \leq n} |d_i|$$

Similarly, the 1-norm induced by a vector on a matrix A is equal to the maximum column sum of A and the ∞ norm of a matrix is equal to maximum row sum of the matrix [3] .

The vector norms and the matrix norm induced by the vector norms satisfy

the Holder inequality: For any two vectors x, y

$$|x^*y| \leq \|x\|_p \|y\|_q$$

where

$$\frac{1}{p} + \frac{1}{q} = 1, \quad 1 \leq p, q \leq \infty$$

The Cauchy-Schwarz inequality is the special case of Holder inequality for $p = 2$.

The Cauchy-Schwarz inequality is used in several proofs requiring bounding the norms, especially in stability and conditioning and related areas.

A matrix norm not induced by a vector norm is the *Hilbert-Scmidt* or the Frobenius norm, defined by

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}} = \sqrt{tr(A^*A)} = \sqrt{tr(AA^*)}$$

The Frobenius norm is involved in the results consisting of conditioning and stability pertaining to matrix multiplication, which we shall, in this work, not delve further.

Before we embark on the discussion of solving systems of linear equations, it is essential to understand the various factorization and decompositions of matrices.

Chapter 3

Decomposition and Factorization of matrices

[1](pp.39-48)

3.1 The Singular Value Decomposition

The information in this chapter is important because, as stated at the end of chapter 2, it is essential to discuss factorization and decomposition's of matrices, as the least squares problem, which was briefly touched upon in the introduction, is related by this chapter. The information here will appear many times in chapter 3, as we discuss the implementation and performance, which very briefly discusses methods for solving least square problems, A chapter we need to show how the principle of least squares is linked a system of square matrices. Its will also appear in some subsequent chapters. You will notice that we stay true to our theme, by dicussing the implementation and performance of the topics of this chapter too. We begin:

Definition 15. 'A number $\sigma \geq 0$ is said to be a singular value for a matrix A corresponding to two unit vectors $\vec{u}, \vec{v}$ iff

$$A\vec{u} = \sigma\vec{v} \text{ and } A^*\vec{v} = \sigma\vec{u}$$

The vectors $\vec{u}, \vec{v}$ are said to be left and right singular vectors for σ. Singular values are non-negative real numbers in all cases

Why do use start using this notation for vectors here when you mostly do not else where. A consistent notation would help throughout the document.

Theorem 1. *The fundamental theorem on Singular value decomposition states that any $m \times n$ complex matrix A can be factored uniquely as*

$$A = U\Sigma V^*$$

where U, V are unitary matrices consisting of left and right singular vectors of A and Σ is a diagonal matrix consisting of singular values σ of A. It is also assumed that the singular values are non-negative and non-increasing in order.

The concept of Singular value decomposition(SVD) can be geometrically looked as transformation of a sphere in $\mathbb{C}^n$ described by vectors to a hyperellipse by A in $\mathbb{C}^m$, in which V^* preserves the sphere, the diagonal matrix Σ stretches the sphere into a hyperellise and the unitary matrices U rotates or reflects the hyperellipse.

The various properties of a matrix can be viewed directly related to its singular valued decomposition:

The rank of a matrix A is r where r is the number of nonzero singular values of A

The range(the vector space $\{y \in \mathbb{C}^m : Ax = y\}$) of A is the span of the r left singular vectors and the null space(the vector space $\{x \in \mathbb{C}^n : Ax = 0\}$) is the span of r right singular vectors.

Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. The 2-norm of a matrix $A, \|A\|_2$ is the first nonzero singular value of the matrix and the frobenius norm of the matrix $A, \|A\|_F$ is the square root of the sum of squares of the singular values of A

$$\|A\|_F = \sqrt{\sigma_1^2 + \ldots + \sigma_r^2}$$

The nonzero singular values of A are the square roots of the nonzero eigenvalues of A^*A or AA^*

For a unitary matrix A, i.e, if $A = A^*$, the singular values of A are the absolute values of the eigenvalues of A

For an n-dimensional square matrix A, its absolute value of determinant is equal to $\prod_{i=1}^{n} \sigma_i$, where σ_i are the singular values.

Another convenient way of writing the singular valued decomposition is as the following sum of r rank-one matrices given by

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^*$$

where σ_i, u_i, v_i are the singular values, left and right singular vectors respectively.

The computation of SVD of any complex matrix is similar in algorithmic time to computing the eigendecomposition of the matrix [1].

Example 1. [4] As an example, let us consider the SVD of the matrix

$$A = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix}$$

Here, $A^* = \begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix}$. Thus,

$$A^*A = \begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}$$

We first find the eigenvalues of A^*A. In the above case, it is 8,2. Corresponding to the eigen values, the eigenvectors are $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$. In order to have unitary matrix for V^*, we normalize the above eigenvectors to $v_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and $v_2 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$. Now, since the singular values are square roots of the eigenvalues of A^*A, we have $\sigma_1 = \sqrt{8} = 2\sqrt{2}$ and $\sigma_2 = \sqrt{2}$. In order to find the columns of U, we proceed as

$$Av_1 = \sigma_1 u_1 \implies u_1 = \frac{Av_1}{\sigma_1} \implies$$

$$Av_2 = \sigma_1 u_2 \implies u_2 = \frac{Av_2}{\sigma_2}$$

Performing the calculations as above, we obtain:

$$u_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Thus, the matrices U and V are

$$U = \begin{pmatrix} u_1 & u_2 \end{pmatrix} \quad V = \begin{pmatrix} v_1 & v_2 \end{pmatrix}$$

Hence, the SVD of A is

$$A = U\Sigma V^* = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

3.2 QR Factorization

[1, Theorem 7.1, p.51](pp.48-56)

Theorem 2. *The fundamental theorem in this instance is that any complex $m \times n$ matrix with linearly independent columns($m > n$) can be decomposed, or factored as*

$$A = QR$$

where $Q = Q^*$ is unitary and R is upper triangular given by $R = Q^*A$

It is clear that the columns of Q being orthonormal (mutually orthogonal with individual norm of vectors equal to 1) individual span the columns of A.

The columns of Q are calculated using methods such as Gram-Schmidt orthogonalization, Householder transformations, or Givens rotations.

The Gram-Schmidt projections compute the columns q_n of the matrix Q as follows:

$$q_1 = \frac{P_1 a_1}{\|P_1 a_1\|}, \quad q_2 = \frac{P_2 a_2}{\|P_2 a_2\|}, \quad \ldots, \quad q_n = \frac{P_n a_n}{\|P_n a_n\|}$$

where P_i denotes an orthogonal projector, i.e., a linear transformation P such that $P^2 = P$ and $P = P^*$ and the norm is the familiar 2-norm. Note that the P_i's are mutually orthogonal and $P_1 = I$. They are given by

$$q_1 = a_1 \quad q_2 = a_2 - \frac{P_1^* a_2}{P_1^* P_1} q_1 \quad q_3 = a_3 - \frac{P_1^* a_3}{P_1^* P_1} q_1 - \frac{P_2^* a_3}{P_2^* P_2} q_2 \ldots$$

The entries of the matrix R are given by

$$r_{ij} = q_i^* a_j$$

Note that the columns of Q are normalized, i.e., have their 2-norm equal to 1, and hence, the process can sometimes said to be Gram-Schmidt orthonormalization.

An example involving Gram-Schmidt process would be: [4]

Example 2. Let us consider the matrix

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

Here,

$$a_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad a_2 = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 4 \end{pmatrix} \quad a_3 = \begin{pmatrix} 1 \\ 2 \\ -4 \\ -3 \end{pmatrix}$$

$$P_1 = a_1 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The first step of Gram-Schmidt would be:

$$P_2 = a_2 - \frac{P_1^* a_2}{P_1^* P_1} P_1 = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 4 \end{pmatrix} - \frac{8}{4} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 0 \\ 2 \end{pmatrix}$$

The next step of Gram-Schmidt would be:

$$P_3 = a_3 - \frac{P_1^* a_3}{P_1^* P_1} P_1 - \frac{P_2^* a_3}{P_2^* P_2} P_2 = \begin{pmatrix} 1 \\ 2 \\ -4 \\ -3 \end{pmatrix} - \frac{-4}{4} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \frac{-9}{6} \begin{pmatrix} -1 \\ -1 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{3}{2}1 \\ -3 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 3 \\ -6 \\ 2 \end{pmatrix}$$

Thus,

$$Q = \frac{1}{2} \begin{pmatrix} 1 & -1 & 1 \\ 1 & -1 & 3 \\ 1 & 0 & -6 \\ 1 & 2 & 2 \end{pmatrix}$$

and $R = Q^* A$.

This process requires about $2mn^2$ floating point operations to compute the QR factorization [1].

On the other hand, the Householder triangularization computes the QR factorization by computing the columns of R from the matrix A unlike in Gram-Schmidt where we used A to calculate the columns of Q. Thus, this process can be said to be orthogonal triangularization.

The transformation used to make the columns of A to that of R are the matrices $H = I - 2uu^*$ where $u^*u = 1$ and the rows(elements) u_i of u are given by :

$$u_1 = \left(\frac{\hat{a}_1 - a_1}{2\hat{a}_1} \right)^{\frac{1}{2}}$$

$$\hat{a}_1 = -sign(a_1)(a^*a)^{\frac{1}{2}}$$

$$u_i = \frac{x_i}{-2\hat{x}_1 u_1}$$

where a_i denote the columns of A [1]. where a_1 is the first column of the matrix.

Example 3. As an example, let us factorize the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}$$

First, we calculate u. By using the above given formulae, $\hat{x}_1 = -\sqrt{3} = -1.72, u_1 = 0.888, u_2 = u_3 = 0.325$

The columns of R in this case are

$$r_1 = H_1(a_1) = I - uu^*(a_1) = \begin{pmatrix} -1.72 \\ 0 \\ 0 \end{pmatrix}$$

To find r_2, we have to first transform the second column of A using H_1 and H_2 where H_2 is defined in the same way for H_1 except that the transformation is now applied on the new column of the transformed A, i.e. with the second column a_2' given by:

$$a_2' = H_1(a_2) = I - uu^*(a_2) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} - 2 \begin{pmatrix} 0.888 \\ 0.325 \\ 0.325 \end{pmatrix} \begin{pmatrix} 0.888 & 0.325 & 0.325 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} -3.46 \\ 0.366 \\ 1.366 \end{pmatrix}$$

which gives us the first transformed matrix as:

$$H_1 A = \begin{pmatrix} -1.72 & -3.464 \\ 0 & 0.366 \\ 0 & 1.366 \end{pmatrix}$$

The H_2 is found out bu using $x = \begin{pmatrix} 0.366 \\ 1.366 \end{pmatrix}$ so that

$$\hat{x_1} = -1.414 \; u = \begin{pmatrix} 0.792 \\ 0.609 \end{pmatrix}$$

and

$$H_2 H_1 A = \begin{pmatrix} -1.732 & -3.464 \\ 0 & -1.414 \\ 0 & 0 \end{pmatrix}$$

which gives

$$(H_2H_1)^* = Q = \begin{pmatrix} -0.577 & 0.707 & -0.408 \\ -0.577 & 0 & 0.816 \\ -0.577 & -0.707 & -0.4082 \end{pmatrix}$$

Yet another method of transformation/factorizing matrices to QR form is the Givens rotations, in which the transformation matrix is a rotation matrix with innermost block given by;

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

with

$$\sin\theta = \frac{-b}{\sqrt{a^2+b^2}} \quad \cos\theta = \frac{a}{\sqrt{a^2+b^2}} \quad \hat{a} = \sqrt{a^2+b^2}$$

where $\begin{pmatrix} a \\ b \end{pmatrix}$ is the vector below the first row.

The remaining rows and columns are consisting of permutations of the unit vectors and their transposes.

Each individual Givens transformation matrix yields one zero to the transformed matrix of A, thus the number of multiplications required is equal to the number of zeros required to convert the A to triangular form.

Example 4. In the previous example, the Givens rotation matrices for transforming the matrix A to triangular form are:

$$G_{12}^* = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$G_{13}^* = \begin{pmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{3}} & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{pmatrix}$$

$$G_{23}^* = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix}$$

Note that each G_{ij} produces a zero at the ij place of A. And finally the matrix $Q = G_{23}^* G_{13}^* G_{12}^*$.

The ultimate use of Householder reflectors and Givens rotations is the reduction in the computational time of factorization. Note that the householder transformations require $2mn^2 - \frac{2}{3}n^3$, (where m, n denote the number of rows and columns of the matrix to be factored) flops or floating point operations to perform

the final factorization to QR form, which is clearly less than that required for the Gram-Schmidt process, hence it is widely used in the factorization of matrices. [1](pp.74-75) [5] However, Givens rotations have even lesser computational cost, but are used for sparse matrices, i.e. matrices which have large number of 0s as their entries. Their computational cost is estimated to be around $2n^2(m - \frac{n}{3})$ flops or floating point operations, where m, n denote the number of rows and columns of the matrix to be factored. [6]

Now, we will discuss various examples of the different methods that are available in the literature for solving the system of linear equations.

Chapter 4

General System of Equations

[1](pp.77-87)

The equation that is the main concern and was given in the beginnning is the equation

$$Ax = b$$

for arbitrary matrices $m \times n$, with $m > n$ A and x, b being corresponding n and m dimensional vectors respectively. We note that the equation has a unique solution only in the case of non-singular square(determinant not zero) matrices A.

However in cases where A is non-square, the solution is usually obtained in an approximate form, known as least squares approximation. In the case of square matrices which are non-singular, the solution is usually calculated directly and exactly, or by iterative methods. Note that the direct methods are apt to be used for matrices with sparsity very less, or, many non-zero entries. In the case of sparse matrices, the computational cost involved forces one to use iterative methods.

Here, we briefly discuss the least squares method for arbitrary matrices, direct methods for square matrices and iterative methods for square matrices, with typical emphasis for symmetric or unitary matrices in the case of iterative methods.

[7].

4.1 Least squares principle

[1, Theorem 11.1,p.80](pp.77-87)

As briefly said in the introduction, when there are more equations than unknowns, i.e. $m > n$ there is usually no vector x satisfying $Ax = b$ but in this case it is common to consider what is called the "least squares solution of $Ax = b$" which involves finding the vector x which minimises the 2-norm of $Ax - b$

With regards to this problem, we have a fundamental theorem, enunciated as

follows:

Theorem 3. *[1, Theorem 11.1,p.80] If A be an $m \times n$ complex matrix and b be an m dimensional vector. Then, an n- dimensional vector x minimizes the residual norm $\|r\|_2 = \|b - Ax\|_2$, thus solving the least squares problem*

$$Ax = b$$

iff r in orthogonal to the range of A, that is

$$A^*r = 0$$

$$\implies A^*Ax = A^*b$$

$$Pb = Ax$$

where P is the orthogonal projector onto the range of A. The equation

$$A^*Ax = A^*b$$

is an $n \times n$ system of equations and is known as normal equations. The solution x is unique iff A has rank n

Another formulation of the least squares problem frequently in use in computation is the use of the concept of pseudoinverse. The least squares problem

$$Ax = b$$

as seen from the previous problem has a unique solution iff A has full rank, or a rank of n. If that be the case, then as seen in the previous solution, the solution is given by

$$x = (A^*A)^{-1}A^*b$$

The matrix

$$(A^*A)^{-1}A^*$$

is said to be the pseudoinverse of A denoted by A^+. It is an $n \times m$ matrix.

The two formulations above lead to several methods of computing the least squares solution. The principle ones among them are:The Normal Equations method, The QR Factorization method and The Singular Value Decomposition method(SVD) method.

4.2 Solution to Normal Equations by Cholesky Factorization

[1](pp.77-87)

The normal equations described in the previous section

$$A^*Ax = A^*b$$

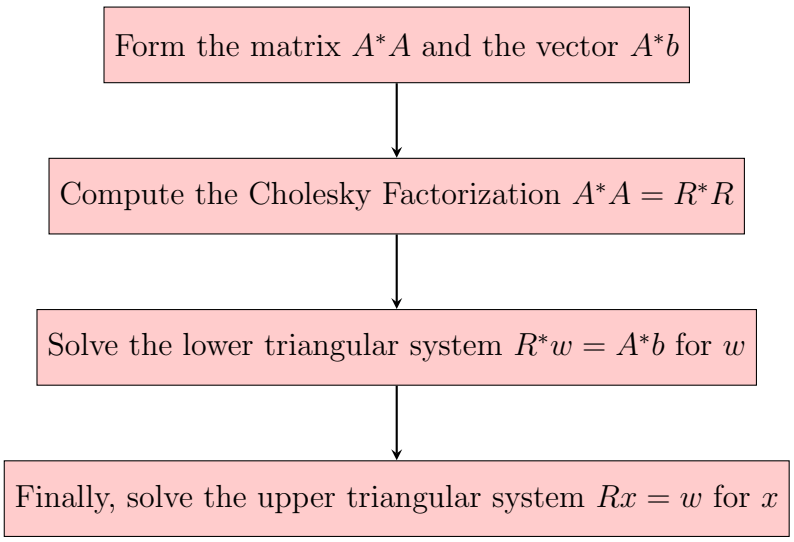
for A being a full rank(of rank n) yields us a hermitian and positive definite matrix A^*A, whence, the system of normal equations is usually solved by factorizing the matrix A^*A to a form

$$A^*A = R^*R$$

where R be upper-triangular, reducing the normal equations to

$$R^*Rx = A^*b$$

The factorization is said to be Cholesky factorization of A^*A. The standard algorithm for doing so is:



It is seen that the above algorithm requires about $mn^2 + \frac{1}{3}n^3$ flops or floating point operations in computation for a matrix A with dimensions $m \times n$ [1].

The Cholesky factorization works only for a positive definite hermitian matrix and is discussed after the LU decomposition.

4.3 Solution to least squares using QR Factorization

[1](pp.77-87)

The QR Factorization discussed in previous Chapter is also applied to solve the least squares problem, especially in cases where the normal equations are not positive definite.

In this case, the matrix A is factored by virtue of Gram-Schmidt, or more popularly, by householder triangularization to the QR form and the projection operator is taken to be $P = QQ^*$. This gives

$$y = Pb = QQ^*b$$

Since $y \in \text{range}(A)$, therefore, the solution to the normal equations would be unique. Combining the QR factorization with the above formulation yields us

$$QRx = QQ^*b$$

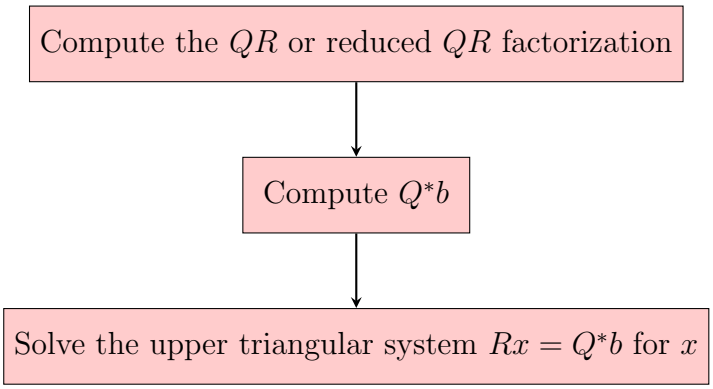
$$\implies Q^*QRx = Q^*QQ^*b$$

$$\implies Rx = Q^*b$$

The above system is an upper triangular system, which would be solved by back-substitution usual in Gaussian elimination, to be discussed later.

We also note that multiplying the last equation by R^{-1} gives us the formula $A^+ = R^{-1}Q^*$, the formula for pseudoinverse.

Hence, the algorithm for solving the least squares problem using QR factorization is:



[1]

you need to mention what the reduced QR means if you refer to it as above.

It is seen that for an $m \times n$ matrix A, the cost for the above algorithm, if Householder reflections are used, is about $\sim 2mn^2 - \frac{2}{3}n^3$ floating point operations,

clearly more than that of the previous algorithm of solving by normal equations [1].

4.4 Solution to least squares using Singular Value Decomposition(SVD)

[1](pp.77-87)

As a preliminary note, in this thesis, we are considering A to be $m \times n$, $m > n$ matrix. Analogous theorems and procedures apply for $n \times m$ matrices. Yet another method for solving the least squares problem is by the use of SVD of $A = U\Sigma V^*$. In this case, the projection operator $P = UU^*$ whence

$$y = Pb = UU^*b$$

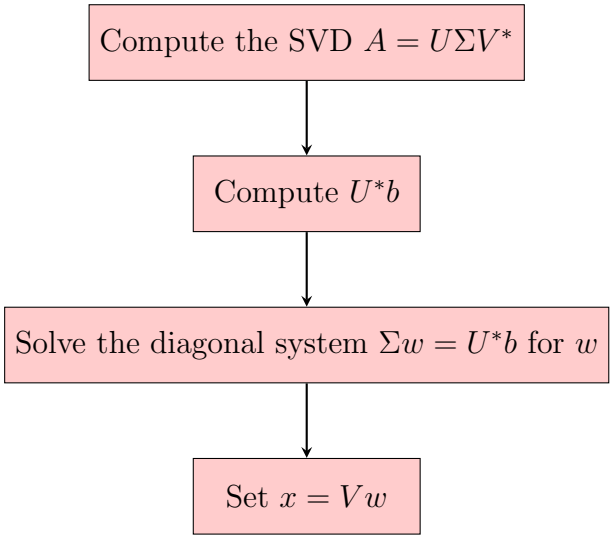
$$\implies U\Sigma V^*x = UU^*b$$

$$\implies U^*U\Sigma V^*x = U^*UU^*b$$

$$\implies \Sigma V^*x = U^*b$$

by virtue of the unitariness of U. Again, we note that, in this case, multiplying both sides by $V\Sigma^{-1}$ gives us $A^+b = V\Sigma^{-1}U^*b$, which gives the solution by pseudo inverse.

Thus, the algorithm for solving through SVD is [1] :



We note that in the third and fourth steps above, the system is reduced to a diagonal system of equations, unlike in the QR factorization case, where the system was reduced to a triangular system.

It is seen that the above algorithm for solving the least squares problem has a cost of about $\sim 2mn^2 + 11n^3$ floating point operations, which is approximately same as that for the QR factorization [1].

We see that we are faced with the problem of choosing between the algorithms. When speed is the main issue, it is best to solve using the first algorithm, based on the normal equations. However, due to accuracy concerns, the classical method prescribed is that by the QR factorization method. But, due to rank deficiency of A, sometimes SVD is preferred over QR subject to certain stability conditions.

The stability conditions stated above is also thus a study to be made, crucial for the decision of choosing between algorithms, which is undertaken next.

Chapter 5

Conditioning and Stability of numerical problems

[1](pp.87-97)

In numerical analysis, it is very important to analyze two very important terms-Conditioning and Stability of problems.

Conditioning refers to the perturbation behaviour, or in other words, the change in the properties of a particular problem when its parameters are changed [1].

Stability pertains to the perturbation behaviour of an algorithm used to solve the problem [1].

A typical problem in numerical analysis can be viewed as a continuous function in its variable. A well conditioned problem is a problem in which all small perturbations of the variable lead to a small change in the value of function. An ill conditioned problem is a problem in which small perturbations in the variable lead to large change in the value of the function.

The smallness and largeness in the above descriptions pertain with respect to the norm in which the variable is perturbed.

In this regard, a very useful concept is that of condition number. The absolute condition number, $\hat{k}$ of any problem, say f at x, where x is the input or the variable of f is defined as:

$$\hat{k} = \lim_{h \to 0} \sup_{\|dx\| \leq h} \frac{\|df\|}{\|dx\|}$$

where

$$df = f(x + dx) - f(x)$$

and dx denotes a small perturbation of x [1].

Since, due to continuity of problem, the limit of supremum can be viewed as

supremum over all infinitesimal perturbations of dx, we can write the condition number as

$$\hat{k} = \sup_{dx} \frac{\|df\|}{\|dx\|}$$

Passing to limit $dx \to 0$, we can get, if the problem be differentiable, the condition number k as a derivative:

$$\hat{k} = \|J(x)\|$$

where $J(x)$ denotes the Jacobian of f. Where the Jacobian is used to describe a matrix and its determinant when the matrix is a square matrix.

The relative condition number, k pertains to relative changes in the problem. It is defined as

$$k = \sup_{dx} \left(\frac{\frac{\|df\|}{\|f\|}}{\frac{\|dx\|}{\|x\|}} \right)$$

Again, assuming differentiability of f, we can obtain the above term in terms of the Jacobian J

$$k = \frac{\|J(x)\|}{\frac{\|f(x)\|}{\|x\|}}$$

The more useful of the condition numbers is the relative condition numbers in numerical analysis.

Now, the above concept of relative condition number is applied to that of the linear algebraic problem of perturbation in Ax from the input x, where A is an $m \times n$ complex matrix. The perturbations relative to x would then be defined as

$$k = \sup_{dx} \left(\frac{\frac{\|A(x+dx) - Ax\|}{\|Ax\|}}{\frac{\|dx\|}{\|x\|}} \right) = \sup_{dx} \frac{\frac{\|Adx\|}{\|\ dx\|}}{\frac{\|Ax\|}{\|x\|}}$$

$$\implies k = \|A\| \frac{\|x\|}{\|Ax\|}$$

by the linearity of the operation on matrices. where $\|\cdot\|$ denotes the vector norm for x, Ax and the vector induced norm for A respectively. The definition of a norm was referred to in chapter 2.

If, in the above calculation, A is a square matrix with non-zero determinant. Then, the fact that $\frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|$ can be used in the above equation to obtain [1]:

$$k \leq \|A\| \|A^{-1}\|$$

$$\implies k = a\|A\| \|A^{-1}\|$$

with

$$a = \frac{\frac{\|x\|}{\|Ax\|}}{\|A^{-1}\|}$$

If the norm be the 2-norm on vectors, then $a = 1$, whence

$$k = \|A\|\|A^{-1}\|$$

Even if A is not a square matrix, the above argument can be generalized to arbitrary complex $m \times n$ matrix with the use of pseudo inverse [1].

The above analysis lead to the following theorem:

Theorem 4. *[1, Theorem 12.1,p.94]*

Let A be a complex $m \times m$ nonsingular matrix. Then, the problem of computing b given x in the system has the condition number:

$$k = \|A\| \frac{\|x\|}{\|b\|} \leq \|A\|\|A^{-1}\|$$

with respect to changes/perturbations of x. Correspondingly, the inverse problem of computing x given b has a condition number:

$$k = \|A^{-1}\} \frac{\|b\|}{\|x\|} \leq \|A\|\|A^{-1}\|$$

with respect to changes/perturbations of b. Equality holds in the above equations if the norm is chosen to be the 2-norm and if x is a multiple of a right singular vector of A corresponding to the first singular value of A or b is a multiple of a left singular vector of A corresponding to the mth singular value of A.

.

We note that the term $k = \|A\|\|A^{-1}\|$ is often called the condition number of the matrix A and is denoted by $k(A)$.

we must note in what follows that the norm is the 2-norm. By using the singular values of A the condition number can also be written as

$$k(A) = \frac{\sigma_1}{\sigma_m}$$

the singular values are $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_m$ and $\|A\| = \sigma_1$ and $\|A^{-1}\| = 1/\sigma_m$. If $k(A)$ is small, A is said to be well conditioned; else if it is large, A is said to be ill conditioned. When the matrix is square singular, $k(A) = \infty$.

We note that for a rectangular $m \times n$ complex matrix A the definition of condition number of A is

$$k(A) = \|A\|\|A^{+}\|$$

For 2-norm in consideration, the condition number can be written as

$$k(A) = \frac{\sigma_1}{\sigma_n}$$

where σ_1, σ_n are the first and n th singular values respectively

5.1 Condition number of a system of equations

[1](pp.87-97)

In the previous section, the perturbation was with respect to x or b for fixed A in the equation $Ax = b$. But, when A is perturbed fixing b or x, we see that a similar conditioning behaviour is observed. Specifically, when A is perturbed by an infinitesimal dA, fixing b, x changes by an infinitesimal dx, where

$$(A + dA)(x + dx) = b$$

Using $Ax = b$ and neglecting $dAdx$, we obtain$(dA)x + A(dx) = 0$, that is, $dx = -A^{-1}(dA)x$, which implies [1]:

$$\|dx\| \leq \|A^{-1}\|\|dA\|\|x\|$$

$$\implies \frac{\frac{\|dx\|}{\|x\|}}{\frac{\|dA\|}{\|A\|}} \leq \|A^{-1}\|\|A\| = k(A)$$

Equality holds whenever dA is such that

$$\|A^{-1}(dA)x\| = \|A^{-1}\|\|dA\|\|x\|$$

The above analysis can be summed as a theorem:

Theorem 5. *[1, Theorem 12.2] If b be fixed and A is a complex non-singular matrix, then the condition number of this problem with respect to perturbations of A is*

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

5.2 Floating point arithmetic and Stability

[1](pp.97-108)

The whole point of the previous sections considering conditioning is related to the crucial concept of stability of algorithm with respect to computers which in turn is related to floating point arithmetic. Computers perform arithmetic in floating point format which involves rounding and even store numbers in such a format, which is unlike the continuous manner in which our human intuition works. Therefore, there are bound to be errors in any computation and working

of any algorithm successfully depends on minimizing the errors associated with such calculations.

Example 5. Let us see an example to see how computers can go wrong in floating point computations. Consider

$$(1 + 1e20) - 1e20 = 0.00000, \quad \text{and} \quad 1 + (1e20 - 1e20) = 1.00000$$

Rounding on a computer would have lead to: $1 + 1e20$ being stored as $1e20$ in the first case.

We see that though both parts are actually the same value in reality, but the computer calculates them significantly differently because of its use of floating point numbers, which rely on the significant digits.

Now follow very carefully the following definitions and examples. They relate to stability, conditioning and accuracy, which is important for in this thesis.

Definition 16. The floating point number system, F is a special system of numbers which is a discrete subset of the real numbers $\mathbb{R}$ determined by the base b, which is typically 2 and an integer $t \geq 1$ known as the precision. It is the set of numbers 0 and numbers of the form

$$x = \pm(\frac{m}{b^t})b^e$$

where m is an integer such that $1 \leq m \leq b^t$ and e is arbitrary integer.

The quantity $\pm\frac{m}{b^t}$ is known as the fraction or mantissa of x and e is the exponent. Thus, the floating point system F is a countably infinite set [1].

Definition 17. The resolution of F is summarized by a number known as the machine epsilon defined as

$$\epsilon_{machine} = \frac{1}{2}b^{1-t}$$

The machine epsilon is half the distance between 1 and the next larger floating point number [1].

Proposition 1. *The fundamental property of $\epsilon_{machine}$ is:*

$$\forall x \in \mathbb{R}, \exists x' \in \mathrm{F} : |x - x'| \leq \epsilon_{machine}|x|$$

In terms of fl, we have:

Proposition 2.

$$\forall x \in \mathbb{R}, \exists |\epsilon| \leq \epsilon_{machine} : fl(x) = x(1 + \epsilon)$$

where fl is a function from $\mathbb{R}$ to F *giving the closest floating point approximation to a real number.*

[1]

Using the above propositions and definitions, we have the fundamental axiom of floating point arithmetic as:

Definition 18.

$$\forall x, y \in F, \exists |\epsilon| \leq \epsilon_{machine} : x \circ y = (x * y)(1 + \epsilon)$$

where $\circ$ is the floating point analogue of any usual arithmetic operation $*$ [1].

In other words, floating point arithmetic is exact up to a relative error of size at most $\epsilon_{machine}$

We also note that complex floating point numbers are represented by pairs of floating point real numbers.

Definition 19. An algorithm $\tilde{f}(x)$ is a map between two vector spaces:$X \to Y$ of data to solutions, similar to a problem defined in the section on conditioning. The data fed is the floating point analogue of the real data and the resulting function is thus a floating point analogue of the real problem [1].

Definition 20. [1] The relative error of computation of a mathematical problem, f is defined as

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)}$$

Now, the main definition concerning accuracy of an algorithm:

Definition 21. [1] An algorithm $\tilde{f}$ is a good algorithm, or accurate, if for each $x \in X$,

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)} = O(\epsilon_{machine})$$

where $O(\epsilon_{machine}$ means on the order of machine, or there exists a constant C such that $\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)} \leq C\epsilon_{machine}$ in the limit $x \to \infty$

The most important concept of stability is defined as:

Definition 22. [1] An algorithm $\tilde{f}$ for a problem f is said to be stable if for each $x \in X$,

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)} = O(\epsilon_{machine})$$

for some $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{machine})$$

where ˜ represents the floating point analogue of a real operation.

In other words, a stable algorithm gives nearly the right answer to nearly the right question.

Another important definition is that of backward stability:

Definition 23. [1] An algorithm $\tilde{f}$ corresponding to a problem f is said to be backward stable if for each $x \in X$,

$$\tilde{f}(x) = f(\tilde{x})$$

for some $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{machine})$$

In other words, a backward stable algorithm gives exactly the right answer to nearly the right question.

A very useful theorem in the areas of accuracy, stability and backward stability is the independence of norm stated as:

Theorem 6. *[1, Theorem 14.1] If problems f and $\tilde{f}$ are defined on finite dimensional spaces X, Y, then the properties of accuracy, stability and backward stability hold independent of the choice of norms in X, Y.*

A proof can be given as follows:

It is known that any two norms on a finite dimensional vector space are equivalent, in the sense that both are of the same order, or there exist constants c_1, c_2 such that the norm of a vector $\|x\|$ is bounded between $c_1\|x\|$ and $c_2\|x\|$. Thus, changing the norm may affect the size of constant bounding $O(\epsilon_{machine})$ but not the existence of such a constant, whence the theorem stands proved [1].

Example 6. The inner product of two vectors $x, y \in \mathbb{C}^m, x^*y$ can be shown to be backward stable for the algorithm being to compute pairwise products of the components of the two vectors and adding them.

Example 7. However the outer product of the same two products, xy^* for two vectors $x \in \mathbb{C}^m, y \in \mathbb{C}^n$ with the algorithm of forming pairwise products and collecting them into a matrix is only stable and not backward stable for the reason that the computational analogue of the collection of the products would not have rank exactly one which is required in the real case.

Example 8. The calculation of eigenvalues of a given complex matrix by using the algorithm of first finding the coefficients of the characteristic polynomial and then finding its roots is unstable, as the problem of finding the roots of a polynomial is ill conditioned. Thus, the algorithm is unsuitable for use.

We also note that the accuracy of a backward stable algorithm depends on the condition number of the problem to be solved:if the problem is well conditioned, the accuracy will be good, else the accuracy is slacked.

The Householder triangularization was preferred in the section on QR factorization of matrices. This is because the algorithm for householder triangularization is backward stable for all matrices. In addition there is the following theorem:

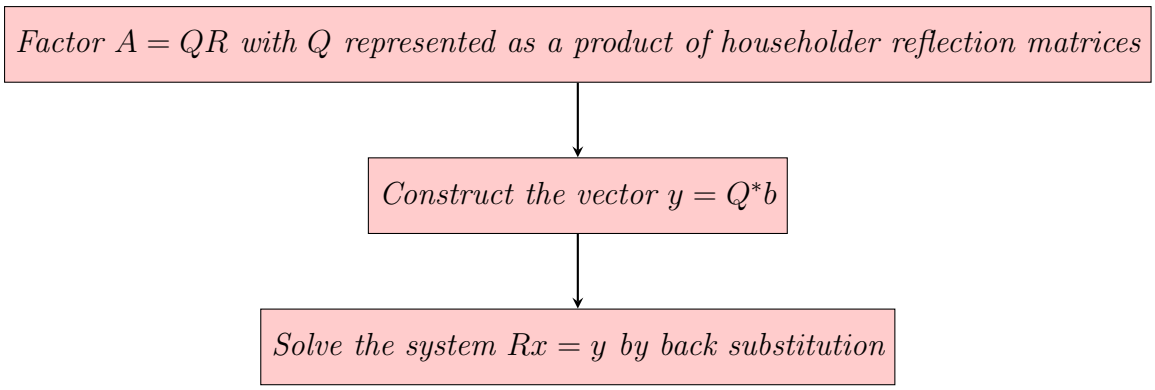
Theorem 7. *[1, Theorem 16.1,p.116] If the QR factorization $A = QR$ of a complex $m \times n$ matrix is computed by means of Householder triangularization on a computer with computed factors $\tilde{Q}, \tilde{R}$, then*

$$\tilde{Q}\tilde{R} = A + dA, \quad \frac{\|dA\|}{\|A\|} = O(\epsilon_{machine})$$

for some $dA \in \mathbb{C}^{m\times n}$

The main problem that we are concerned with in this thesis is the solution of system of equations. Since it is seen that any system of rectangular matrices can be reduced to that of a system of square matrices using the least squares method, therefore our primary concern would be in the solution of such systems, that is solving systems of non-singular square matrices. It was seen that such a system can be reduced to solving a triangular system by the use of QR factorization. The triangular system is in turn solved by the method of back substitution, which is seen to be backward stable by the following theorem:

Theorem 8. *[1, Theorem 16.2,p.118] If we consider the following algorithm to solve a linear nonsingular system of equations $Ax = b$:*



Then, the algorithm is backward stable, satisfying:

$$(A + dA)\tilde{x} = b, \quad \frac{\|dA\|}{\|A\|} = O(\epsilon_{machine})$$

for some $dA \in \mathbb{C}^{m\times m}$ *where˜denotes the computational analogue of the real operation.*

A proof to the above can be given as follows: In addition, it can be shown, by combining the backward stability of QR factorization, householder triangularization and the condition number of A, that:

Theorem 9. *[1, Theorem 16.3, p.119] The solution of* $Ax = b$, $\tilde{x}$ *computed by the above algorithm satisfies*

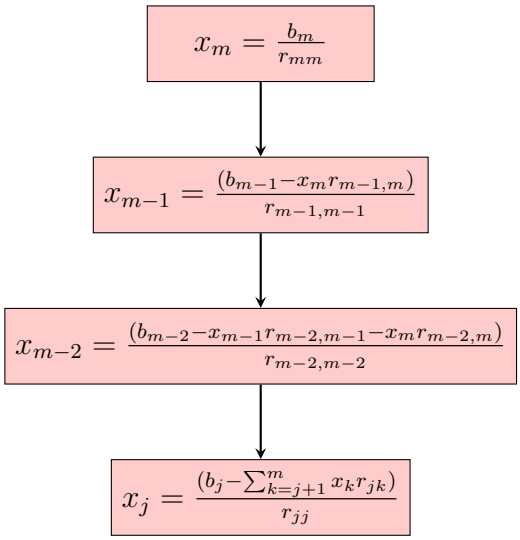
$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(k(A)\epsilon_{machine})$$

where $k(A)$ *denotes the condition number of the matrix* A.

The method of back substitution is also seen to be backward stable. The method of back substitution is used in solving triangular system of equations. It will be observed that any given nonsingular system of equations can be reduced to solving a triangular system of equations in the previous sections.

It would also be observed in the coming section that direct methods to solve linear system of nonsingular equations such as Gaussian elimination, LU Decomposition, Cholesky Factorization etc. are essentially triangular systems. Therefore the stability of algorithm to solve such systems is vitally to be studied.

In regards to this, the algorithm of solving lower triangular system of equations, that is , systems of the form $RX = b$ where R is a lower triangular complex square matrix is said to be forward substitution whereas that of solving upper triangular systems, where R is upper triangular square complex matrix is back substitution. The algorithm of back substitution can be described as:



where $Rx = b$ is the upper triangular system to be solved with x_i, b_i, r_{ij} denoting components of x, b, R respectively [1].

The above algorithm requires about $\sim m^2$ floating point operations [1].

With regards to the above algorithm, we have the following theorem:

Theorem 10. *[1, Theorem 17.1,p.122] The back substitution algorithm to solve the upper triangular system is backward stable with the computed solution $\tilde{x} \in \mathbb{C}^m$ satisfying*

$$(R + dR)\tilde{x} = b$$

for some upper triangular $dR \in \mathbb{C}^{m \times m}$ with

$$\frac{\|dR\|}{\|R\|} = O(\epsilon_{machine})$$

specifically, for each component of the matrix dR, we have:

$$\frac{|dr_{ij}|}{|r_{ij}|} \leq m\epsilon_{machine} + O(\epsilon^2_{machine})$$

There exist theorems and definitions pertaining to conditioning and stability least squares problems and their solution algorithms like SVD, Householder triangularization and Normal Equations. However, in this thesis, we will be mainly concerned with solving $Ax = b$ for square matrices A, hence the discussion of least squares in detail would not be made.

With the above observations on the general nature of solving a typical equation $Ax = b$ for arbitrary complex matrix A, we embark on the solution to non singular system of equations $Ax = b$, since it was observed that the use of least squares problem gives $A^*Ax = A^*b$. Meaning it actually reduces the original system to solving $Ax = b$ for square matrices. And this is the square system that you solve. We naturally begin with Direct methods, which are mathematically straightforward.

Chapter 6

Direct Methods of Solving nonsingular systems

[1](pp.145-155)

6.1 Gaussian Elimination

Now that you are well equipped in the background of conditioning, computational costs and stability, we can now effectively analyze the performance of the direct methods implemented for square matrices. As I said in the background document, the classical solution method for systems of linear equations, familiar from linear algebra, is **Gaussian elimination**, the natural generalization of solving two equations with two unknowns. The method is quite similar to factorization of matrices by QR factorization, that is Gaussian elimination is essentially triangularization of the system $Ax = b$, the main difference is that the transformations applied to A to triangularize are not unitary. The algorithm can be basically described as follows:

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

- Solve one of the n equations (e.g. the first one) for one of the unknowns (e.g. x_1)
- Replace x_1 by the resulting term (depending on $x_2, \ldots, x_n$) in the other $n-1$ equations
- Hence, x_1 is eliminated from those
- Solve the resulting system of $n-1$ equations with $n-1$ unknowns analogously and continue until an equation only contains x_n, which can therefore

be explicitly calculated

- Now, x_n is inserted into the elimination equation of x_{n-1}, so x_{n-1} can be given explicitly
- Continue until at last the elimination equation of x_1 provides the value for x_1 by inserting the values for $x_2, \ldots, x_n$ (known by now)
- Simply speaking, the elimination means that A and B are modified such that there are only zeros below $a_{1,1}$ in the first column. Note that the new system (consisting of the first equation and the remaining x_1-free equations), of course, is solved by the same vector x as the old one

Note: When you put A/b in Matlab, a proprietary programming language developed by Mathworks that allows matrix manipulations, the software attempts to classify the type of matrix involved and then selects the best technique to use to reliably solve the problem in an efficient way [3]. Seeing as this thesis is based on solving these methods when large n. It's of interest to us.

6.2 LU Decomposition

[1](pp.147-155)

We will see in the coming algorithm that together with the matrix A, the matrices L and U appear in the algorithm of Gaussian elimination. We have,

- In U, only the upper triangular part (inclusive the diagonal) is populated.
- In L, only the strict lower triangular part is populated (without the diagonal).
- If filling the diagonal in L with ones, we get the fundamental relation

$$A = L \cdot U$$

.

At the start you do not have a previous algorithm. You can check my MA2715 notes. You are not explaining things here and L is unit lower triangular.

Such a decomposition or factorization of a given matrix A in factors with certain properties (here: triangular form) is a very basic technique in numerical linear algebra.

[8]

It is important to note that the matrix must have all the principal minors non-zero so LU decomposition only works in limited conditions. For more details, we refer to [9]

Here is an illustration of LU decomposition in one case,

1. Our initial augmented system will look like

$$\left(\begin{array}{ccccc} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & -4 \end{array}\right) \left|\begin{array}{c} -8 \\ -34 \\ 43 \\ 19 \\ 57 \end{array}\right. \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

2. After that, the first column will be eliminated and the system will transform as

$$\left(\begin{array}{ccccc} 1 & 2 & -2 & -1 & 1 \\ 0 & -1 & 1 & 4 & 1 \\ 0 & 0 & 7 & 4 & -3 \\ 0 & -9 & 8 & 4 & -5 \\ 0 & 0 & 5 & 0 & 3 \end{array}\right) \left|\begin{array}{c} -8 \\ -18 \\ 51 \\ 43 \\ 65 \end{array}\right. \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

3. Now, the second column will be eliminated using elementary row transformations, which are nothing but linear combinations of the rows of the matrix

$$\left(\begin{array}{ccccc} 1 & 2 & -2 & -1 & 1 \\ 0 & -1 & 1 & 4 & 1 \\ 0 & 0 & 7 & 4 & -3 \\ 0 & 0 & -1 & -32 & -14 \\ 0 & 0 & 5 & 0 & 3 \end{array}\right) \left|\begin{array}{c} -8 \\ -18 \\ 51 \\ 205 \\ 65 \end{array}\right. \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 3 & 9 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

4. After the third column elimination, we have

$$\left(\begin{array}{ccccc} 1 & 2 & -2 & -1 & 1 \\ 0 & -1 & 1 & 4 & 1 \\ 0 & 0 & 7 & 4 & -3 \\ 0 & 0 & 0 & -\frac{220}{7} & -\frac{101}{7} \\ 0 & 0 & 0 & -\frac{20}{7} & \frac{36}{7} \end{array}\right) \left|\begin{array}{c} -8 \\ -18 \\ 51 \\ \frac{1486}{7} \\ \frac{200}{7} \end{array}\right. \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 3 & 9 & -\frac{1}{7} & 1 & 0 \\ 1 & 0 & \frac{5}{7} & 0 & 1 \end{array}\right)$$

5. Finally, the last column will be eliminated and we get our U and L as

$$U := \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 0 & -1 & 1 & 4 & 1 \\ 0 & 0 & 7 & 4 & -3 \\ 0 & 0 & 0 & -\frac{220}{7} & -\frac{101}{7} \\ 0 & 0 & 0 & 0 & \frac{497}{77} \end{pmatrix} \Bigg| \begin{pmatrix} -8 \\ -18 \\ 51 \\ \frac{1486}{7} \\ \frac{714}{77} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 3 & 9 & -\frac{1}{7} & 1 & 0 \\ 1 & 0 & \frac{5}{7} & \frac{1}{4} & 1 \end{pmatrix} := L$$

6. Finally, we have the factors L and U and

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 3 & 9 & -\frac{1}{7} & 1 & 0 \\ 1 & 0 & \frac{5}{7} & \frac{1}{4} & 1 \end{pmatrix} . \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 0 & -1 & 1 & 4 & 1 \\ 0 & 0 & 7 & 4 & -3 \\ 0 & 0 & 0 & -\frac{220}{7} & -\frac{101}{7} \\ 0 & 0 & 0 & 0 & \frac{497}{77} \end{pmatrix} = \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 2 & 3 & -3 & 2 & 3 \\ 1 & 2 & 5 & 3 & -2 \\ 3 & -3 & 2 & 1 & -2 \\ 1 & 2 & 3 & -1 & -4 \end{pmatrix}$$

7. thus, we have $L \cdot U = A$.

The insight above means for us: Instead of using the classical Gaussian elimination, we can solve $Ax = b$ with the triangular decomposition $A = LU$, namely with the algorithm resulting from

$$Ax = LUx = L(Ux) = Ly = b$$

. We note that this process would essentially involve solving two triangular systems:

- First, solve $Ly = b$ for unknown y by forward substitution
- Then, solve $Ux = y$ by back substitution.

It is observed that the above processes require $\sim \frac{2}{3}m^3 + m^2 + m^2$ floating point operations, which amounts to about $\sim \frac{2}{3}m^3$ floating point operations, which is half of $\sim \frac{4}{3}m^3$ floating point operations required for a solution by Householder triangularization [1].

We must also note that the above algorithm presented is unstable in that it is not backward stable. The instability is related to the fact that the Gaussian elimination, as presented above, attempts a division by zero for some matrices.

Example 9. [1](pp.152-153)

To illustrate the instability of Gaussian elimination in the usual algorithm as

described above, we see that for the matrix

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

The matrix is well conditioned with a condition number of

$$k(A) = \frac{(3+\sqrt{5})}{2} \approx 2.618$$

However, the above algorithm of Gaussian elimination fails at the first step because of the attempt at division by zero.

When the matrix is perturbed slightly, say the Gaussian elimination is applied to the matrix

$$\begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}$$

the procedure does not fail.

Supposing that the Gaussian elimination is performed on a machine with $\epsilon_{machine} \approx 10^{-16}$, the number $1 - 10^{20}$ will be represented rounded off to say, $-10^{-}20$. The matrices $\tilde{L}\tilde{U}$ produced would be:

$$\begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \quad \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{-20} \end{pmatrix}$$

respectively.

But, the product $\tilde{L}\tilde{U}$ would be:

$$\begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}$$

which is not at all close to A.

The problem with the last example and, in general, with instability of Gaussian elimination is that the LU factorization, though stable, is not backward stable. In addition, the triangular matrices generated have condition numbers which may be arbitrarily large.

To encounter the problem with Gaussian elimination, as seen in the previous example, the concept of pivoting is introduced to the process of Gaussian elimination, which is discussed next.

6.3 Gaussian Elimination with Pivoting

[1](pp.155-172)

We observed that in the algorithm of the previous section, we assumed that divisions of the kind $a_{i,j}/\ u_{j,j}$ or $x_i/u_{i,i}$ do not cause any problems, i.e. particularly that there occur no zeros in the diagonal of U.

But, we also saw in an example the instability that this can give rise to. To encounter these issues, the concept of pivoting is introduced.

- As everything centers on those values in the diagonal, they are called Pivots (French word).
- If the basic Gauss elimination process is applied to a positive definite matrix then it can be shown that the diagonal entries at all stages can be shown to be positive. So in the positive definite case, all eigenvalues are positive, which is why zeros are impossible in the diagonal – for example, in the Cholesky method, to be discussed in the next section, everything is alright [1].
- However, in the general case, the requirement $u_{j,j} \neq 0 \forall j$ is not granted. If a zero emerges, the algorithm has to be modified, and a feasible situation, i.e. a non-zero in the diagonal, has to be forced by permutations of rows or columns(which is possible, of course, when A is not singular!) [1].
- This 'column pivot search' leads to what is known as row pivoting, i.e. by swapping rows based on this search. The 'total pivot search' which leads to complete pivoting which involves swapping both rows and columns.

Consider for example the matrix

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 1 & 1 \end{pmatrix}$$

A possible partner for exchange of a zero $u_{i,i}$ in the diagonal can be found either in the column i below the diagonal (column pivot search) or in the entire remaining matrix (everything from the row and the column $i+1$ onward, total pivot search).

However, we see that the above method of total pivoting or total pivot search is computationally quite expensive, as close to m^3 floating point operations or flops are required for a square complex matrix A of order m. Thus, in order to reduce the computational cost, the concept of partial pivoting is also in use.

The method of partial pivoting consists in swapping only rows. The pivot is chosen to be largest of the sub-diagonal entries of a given column. This reduces the computational cost by an order of one, that is, only about m^2 floating point operations are required for a square complex matrix of order m.

The algorithm for partial pivoting is similar to the usual elimination, except that the successive elimination is pre-multiplied by a permutation matrix, which is a matrix with zero everywhere except for a 1 at each row and column, or, the matrix obtained by permuting the rows or columns of the identity matrix. After $m-1$ steps, the original matrix A becomes an upper triangular matrix U where

$$L_{m-1}P_{m-1}\dots L_2P_2L_1P_1A = U$$

with L_i being the lower triangular matrices and P_j being the permutation matrices.

Example 10. [1, p.157-158] An illustration is quite useful in this case. We consider factorizing

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

the first step in partial pivoting is to interchange the first and third rows, which amounts to left multiplication by P_1

$$\begin{pmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

The first elimination step would be:

$$\begin{pmatrix} 1 & & & \\ -\frac{1}{2} & 1 & & \\ -\frac{1}{4} & & 1 & \\ -\frac{3}{4} & & & 1 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix}$$

Now, we interchange the second and fourth rows, which is equivalent to left multiplication by P_2:

$$\begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix}$$

The next elimination step would be multiplication by L_1:

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{3}{7} & 1 & \\ & \frac{2}{7} & & 1 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix}$$

Again, the third and fourth rows are interchanged, which amounts to multiplication by P_3:

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -\frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix}$$

The final elimination step is pre multiplication by L_3:

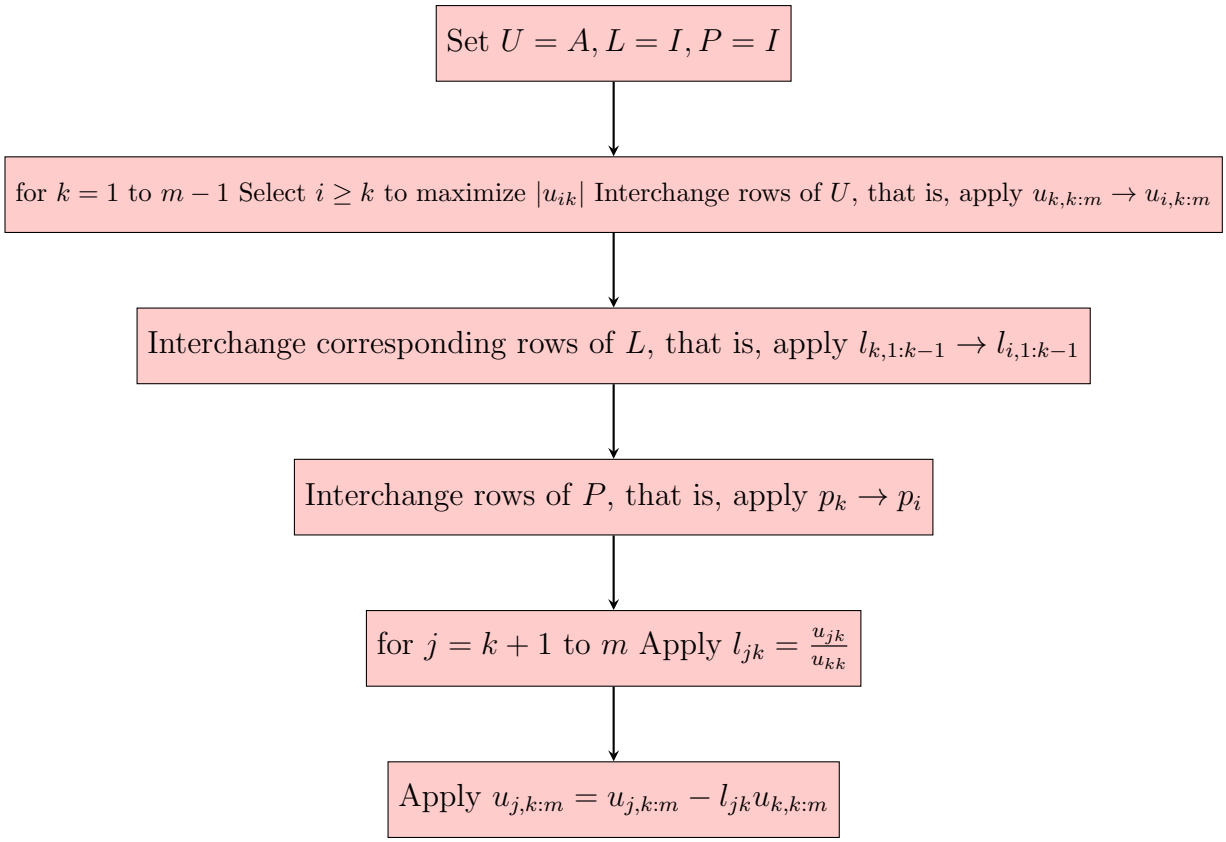
$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -\frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{pmatrix}$$

The example above illustrated the computation of the LU factorization of the matrix PA, where $P = P_{m-1} \dots P_2 P_1$ and $L = (L'_{m-1} \dots L'_2 L'_1)$ with $L'_k = P_{m-1} \dots P_{k+1} L_k P_{k+1}^{-1} \dots P_{m-1}^{-1}$ which is in general known as the LU factorization of A.

The Gaussian elimination with partial pivoting can be thus written as equivalent to:

- Permutation of the rows of A, equivalent to pre-multiplying A by P..
- Apply the usual Gaussian elimination without pivoting to PA.

The formal statement of the algorithm is [1]:



The above algorithm is for Gaussian elimination with partial pivoting. There exists, another, computationally expensive , method of Gaussian elimination with complete pivoting, in which both rows and columns are interchanged. The schematic factorization of A in that case could be written as:

$$PAQ = LU$$

where, in addition to the PA applied in partial pivoting, Q denotes the permutations of columns of A [1].

6.4 Stability of Gaussian elimination

[1](pp.163-172)

Gaussian elimination is very difficult to analyze in terms of stability. The general problem of instability of Gaussian elimination is not yet satisfactorily understood. However, Gaussian elimination with partial pivoting is backward stable for most of the problems that are computed in numerical linear algebra. To this effect, some theorems follow which lead us to a clearer understanding of the stability of Gaussian elimination algorithms:

Theorem 11. *[1, Theorem 21.1,p.166] If the factorization $A = LU$ of a nonsingular matrix $A \in \mathbb{C}^{m\times m}$ be calculated in a machine which satisfies the fundamental*

axiom of floating point arithmetic. If A has an LU factorization, then for sufficiently small $\epsilon_{machine}$, the computation of the factorization would be successful and the computed matrices $\tilde{L}, \tilde{U}$ satisfy:

$$\tilde{L}\tilde{U} = A + dA \quad \frac{\|dA\|}{\|L\|\|U\|} = O(\epsilon machine)$$

for some $dA \in \mathbb{C}^{m\times m}$

We note that, in general $\|L\|\|U\| \neq O(\|A\|)$, which implies that the algorithm is not backward stable, however we must note that it is not guaranteed that $\|L\|\|U\| \neq O(\|A\|)$.

Whereas, in the case of Gaussian elimination with partial pivoting, since pivoting involves maximization over a column, therefore, the algorithm produces a matrix L with entries of absolute value less than or equal to 1. Therefore, we have, $\|L\| = O(1)$, which in turn, from the previous theorem, implies that

$$\frac{\|dA\|}{\|U\|} = O(\epsilon_{machine})$$

which implies backward stability provided $\|U\| = O(\|A\|)$. This leads us to the following theorem on the stability of the algorithm:

Theorem 12. *[1, Theorem 22.2,p.165] The factorization $PA = LU$ of a complex square matrix A of order m computed by using Gaussian elimination with partial pivoting on a computer satisfying the fundamental axiom of floating point arithmetic produces $\tilde{P}, \tilde{L}, \tilde{U}$ that satisfy*

$$\tilde{L}\tilde{U} = \tilde{P}A + dA \quad \frac{\|dA\|}{\|A\|} = O(\rho\epsilon_{machine})$$

for some complex matrix dA of order $m \times m$ and $\rho = \frac{max_{i,j}|u_{ij}|}{max_{i,j}|a_{ij}|}$ is said to be growth factor for A. In addition, if $|l_{ij}| < 1$ for all $i > j$, then $\tilde{P} = P$ for all sufficiently small $\epsilon_{machine}$.

thus, the algorithm is backward stable if $\rho = O(1)$

Example 11. [1, p.165] Despite the backward stability promised in the last theorem, theoretically it is possible for the algorithm of Gaussian elimination

with partial pivoting to be unstable. An example is illustrated with the matrix

$$A = \begin{pmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

At the end of factorization, we obtain:

$$U = \begin{pmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{pmatrix}$$

giving the final $PA = LU$ factorization as:

$$\begin{pmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{pmatrix}$$

The growth factor, ρ for the above matrix is 16. Thus, for a complex square matrix of order m of the same form, the growth factor can get as large as 2^{m-1}, which indicates the unstable nature of the Gaussian elimination with partial pivoting.

Despite the above example, the Gaussian elimination with partial pivoting is widely used in practice because factors of the form in the last example never seem to appear in practical applications. Thus, from a statistical point of view, the gaussian elimination with partial pivoting is relatively backward stable.

A partial explanation for this could be given as follows: Since we have $PA = LU$, $U = L^{-1}PA$. Thus, if Gaussian elimination is unstable when applied to the matrix A, then it is implied that ρ is large, which thus implies that L^{-1} must also be large. However, there exist correlations among the signs of entries of L that render the matrices L^{-1} well conditioned when gaussian elimination with partial pivoting is applied to A. In other words, when A is random, its column space is randomly oriented and the same follows for $P^{-1}L$ which is incompatible with L^{-1} being large. If L^{-1} is large, then the column spaces of L or $P^{-1}L$ must be skewed in such a way that its column space is not random.

6.5 Cholesky Factorization

[1](pp.172-179)

When the matrix A in the familiar system of equations

$$Ax = b$$

is hermitian, the factorization $PA = LU$ can be carried out twice as fast the case for non-hermitian matrices. The technique used for doing so is the Cholesky Factorization. This method only applies for positive definite matrices as for the reduction the matrix needs to have the positive real eigen values. The method described above, with which the calculation of the $u_{i,k}$, $i \neq k$, in the LU factorization can be avoided and with that about half of the total computing time and required memory.

[10].

We recall that a hermitian matrix A satisfies $x^*Ay = \overline{y^*Ax} \forall x, y \in \mathbb{C}^m$ and positive definite hermitian matrices satisfy $x^*Ax > 0$ in addition.

The eigenvalues of a hermitian positive definite matrix are all positive because, if $Ax = \lambda x, x \neq 0 \implies x^*Ax = \lambda x^*x > 0 \implies \lambda > 0$.

We also have that the eigenvectors corresponding to distinct eigenvalues are orthogonal For [1] (pp.169) ,

$$Ax_1 = \lambda_1 x_1 \quad Ax_2 = \lambda_2 x_2 \quad \lambda_1 \neq \lambda_2$$

$$\implies \lambda_2 x_1^* x_2 = x_1^* A x_2 = \overline{x_2^* A x_1} = \overline{\lambda_1 x_2^* x_1} = \lambda_1 x_1^* x_2$$

$$\implies (\lambda_1 - \lambda_2) x_1^* x_2 = 0$$

Since $\lambda_1 \neq \lambda_2$ we get that $x_1^* x_2 = 0$

the problem of decomposing a hermitian positive definite matrix proceeds along similar lines to that of Gaussian elimination. Consider the example of triangularizing

$$A = \begin{pmatrix} 1 & w^* \\ w & v \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ w & I \end{pmatrix} \begin{pmatrix} 1 & w^* \\ 0 & v - ww^* \end{pmatrix}$$

The usual gaussian elimination would now continue the reduction to triangular form by introducing zeros in the second column. But, in the Cholesky factorization first introduces zeros in the first row to match the zeros introduced in the first column. This is done by a right upper- triangular operation that subtracts

multiples of first column from subsequent ones:

$$\begin{pmatrix} 1 & w^* \\ 0 & v - ww^* \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & v - ww^* \end{pmatrix} \begin{pmatrix} 1 & w^* \\ 0 & I \end{pmatrix}$$

combining the two operations above, the matrix A has been factored into three terms:

$$A = \begin{pmatrix} 1 & w^* \\ w & v \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ w & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & v - ww^* \end{pmatrix} \begin{pmatrix} 1 & w^* \\ 0 & I \end{pmatrix}$$

The above process is repeted in the general Cholesky factorization process.

In general, the matrix A is factored into first into the form:

$$A = R_1^* A_1 R_1$$

where R is upper triangular which finally results in

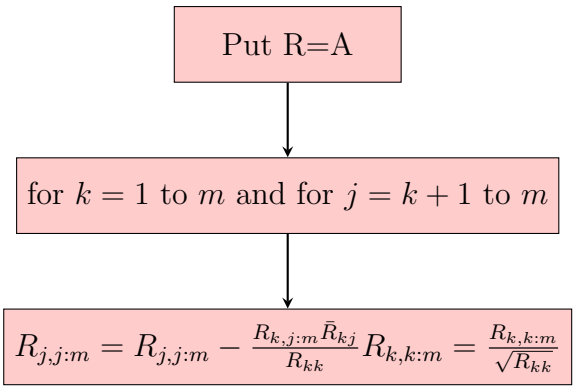
$$A = R^* R$$

with $R^* = R_1^* R_2^* \dots R_m^*$ and $R_m \dots R_2 R_1 = R$. [1](pp.173-174)

Hence, we have the following theorem:

Theorem 13. *[1, Theorem 23.1,p.174] Every hermitian positive definite matrix $A \in \mathbb{C}^m$ has a unique Cholesky factorization $A = R^* R$*

The algorithm for Cholesky factorization can be written as [1] :



It is seen that the Cholesky factorization is dominated by the operations performed in the inner loop of the above algorithm, which requires about $m-j+1$ multiplications, one division and $m - j + 1$ subtractions which is carried $2(m - j$ floating point operations, repeated for each j from $k + 1$ to m and the whole loop being repeated for each k from 1 to m. thus the algorithm requires $\sim \frac{1}{3}m^3$ floating point operations, which is half of that of Gaussian elimination $\sim \frac{2}{3}m^3$ for a complex hermitian matrix of order m

Example 12. We let

$$A = \begin{pmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{pmatrix} = \begin{pmatrix} R_{11} & 0 & 0 \\ R_{12} & R_{22} & 0 \\ -R_{13} & R_{23} & R_{33} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R13 \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{pmatrix}$$

The first row of R would be formed as follows:

$$A = \begin{pmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 3 & R_{22} & 0 \\ -1 & R_{23} & R_{33} \end{pmatrix} \begin{pmatrix} 5 & 3 & -1 \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{pmatrix}$$

The second row of R would be formed as follows:

$$A = \begin{pmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & R_{33} \end{pmatrix} \begin{pmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & R_{33} \end{pmatrix}$$

where the operation is

$$\begin{pmatrix} 18 & 0 \\ 0 & 11 \end{pmatrix} - \begin{pmatrix} 3 \\ -1 \end{pmatrix} \begin{pmatrix} 3 & -1 \end{pmatrix} = \begin{pmatrix} R_{22} & 0 \\ R_{23} & R_{33} \end{pmatrix} \begin{pmatrix} R_{22} & R_{23} \\ 0 & R_{33} \end{pmatrix}$$

$$\implies \begin{pmatrix} 9 & 3 \\ 3 & 10 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 1 & R_{33} \end{pmatrix} \begin{pmatrix} 3 & 1 \\ 0 & RR_{33} \end{pmatrix}$$

The third and final column of R would be calculated as:

$$10 - 1 = R_{33}^2 \implies R_{33} = 3$$

resulting in the final factorization as

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

The above method of Cholesky factorization is backward stable, intuitively because R can never grow too large by the Singular Value Decomposition, for example, in the 2-norm, $\|R\| = \|R^*\| = \|A\|^{\frac{1}{2}}$. Thus, the following theorem is with respect to this effect:

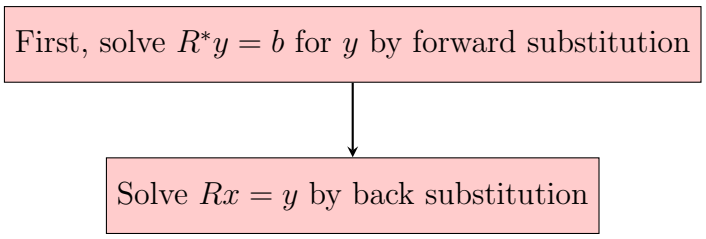
Theorem 14. *[1, Theorem 23.2,p.176] The Cholesky decomposition of a complex positive definite hermitian matrix A of order m, when performed on a machine satisfying the fundamental axiom of floating point arithmetic runs to completion*

with the computed factor $\tilde{R}$ *satisfying*

$$\tilde{R}^*\tilde{R} = A + dA \quad \frac{\|dA\|}{\|A\|} = O(\epsilon_{machine})$$

for some matrix $dA \in \mathbb{C}^{m\times m}$

Thus, due to the stability of Cholesky decomposition, it is frequently used to solve systems $Ax = b$ for positive definite hermitian $A = R^*R$ as follows:



It can be shown that the above procedure is backward stable:

Theorem 15. *[1, Theorem 23.3,p.177] The solution to hermitian positive definite system* $Ax = b$ *via the Cholesky decomposition algorithm is backward stable, with the computed solution* $\tilde{x}$ *satisfying*

$$(A + dA)\tilde{x} = b \quad \frac{\|dA\|}{\|A\|} = O(\epsilon_{machine})$$

for some $dA \in \mathbb{C}^{m\times m}$

Chapter 7

Iterative Methods

[1](pp.241-245) [11](pp.2-9)

The methods developed in the previous sections were of a direct nature, the number of computations being known in advance and even the accuracy also known in advance. But, we saw that for large systems, the direct methods were computationally very costly.

In practice, computation of solutions to systems of equations for large and sparse(with many elements of the matrix zero) is done by iterative methods, in which a solution is first guessed and, if the method is convergent the guessed solution is iterated to be closest to the exact solution by repeated iteration. This iterative method has the benefit that it saves a lot of computational steps especially for sparse systems, and, in additions the number of steps required varies with the required accuracy- thus giving a better control over the computations.

It should be noted, however, that the direct methods should be preferred over iterative methods in general and iterative methods must be used mainly for large matrices and mainly sparse systems.

7.1 Theory of iterative methods

[11](pp.2-9)

The basic idea involved in approximating the solution to a system in iterative methods is this:

Given a linear system $Ax = b$, with A invertible, suppose we can write A in the form

$$A = M - N$$

with M invertible, and "easy to invert," which means that M is close to being a diagonal or a triangular matrix (perhaps by blocks). Then, $Ax = b$ is equivalent to $Mx = Nx + b$,

Now, we assume a first approximation, x_0 and the iterate for better approximation using the above equation, that is,

$$x_{(k+1)} = M^{-1}Nx_{(k)} + M^{-1}b$$

where $k \in \mathbb{N}$

In the above procedure, the iteration matrix G is given by $I - M^{-1}A = M^{-1}N$

By the above procedure, the solution of the original system is approximated easily, and can be faster than the direct method of solving the system.

7.2 Jacobi Method

[11](pp.11-12) [12](pp.275-278)

Based on the general procedure for the iterative methods described above, the general algorithm for Jacobi iterative methods for square nonsingular systems is:

Given a linear system $Ax = b$ (with A a square invertible matrix), split it into a diagonal matrix , D with the entries being the diagonal entries of A and a remainder matrix, with diagonal entries zero and remaining entries equal to the corresponding entries of A. Then, we use the similar procedure as described in the last subsection.

More explicitly, if

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & ... & a_{2n-1} & a_{2n} \\ a_{31} & a_{32} & a_{33} & ... & a_{3n-1} & a_{3n} \\ ... & ... & ... & ... & ... & ... \\ a_{n-11} & a_{n-12} & a_{n-13} & ... & a_{n-1n-1} & a_{n-1n} \\ a_{n1} & a_{n2} & a_{n3} & ... & a_{nn-1} & a_{nn} \end{pmatrix}$$

then

$$D = \begin{pmatrix} a_{11} & 0 & 0 & ... & 0 & 0 \\ 0 & a_{22} & 0 & ... & 0 & 0 \\ 0 & 0 & a_{33} & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & a_{n-1n-1} & 0 \\ 0 & 0 & 0 & ... & 0 & a_{nn} \end{pmatrix}$$

thus, in Jacobi's method, we assume that all diagonal entries in A are nonzero,if

not, it is so rearranged such that this is the case, we pick

$$M := D$$

As a matter of notation, we let

$$J := I - D^{-1}A$$

which is called the Jacobi Matrix. [13] Here, the equivalent system's matrix as per our theory will be

$$G = M^{-1}N$$

that is to be more specific,

$$G = D^{-1}N$$

7.3 Gauss Seidel Method

[12](pp.275-278)

Here

$$A = (D - E) - F$$

and the splitting is $M = D - E$ and $N = F$

The corresponding method to Jacobi's method, the Gauss-Seidel, computes the sequence (x_k) using the recurrence

$$x_{k+1} = D^{-1}(E + F)x_k + D^{-1}b \quad k > 0$$

where E and F are the upper and lower triangular parts of A such that $A = E+F$ In practice we solve,

$$Dx_{k+1} = (E + F)x_k + Bk > 0$$

with the iteration matrix of Gauss-Seidel, denoted by L_1, with

$$L_1 = (D - E)^{-1}F$$

One of the advantages of the method of Gauss-Seidel is that it requires only half of the memory used by Jacobi's method [14].

The Jacobi and Gauss-Sidel methods converge, for any choice of the first approximation, if every equation of the system satisfies the condition that the sum of absolute values of the coefficients, $\frac{a_{ij}}{a_{ii}}$ is almost equal to, or in at least one

equation less than unity, that is,

$$\sum_{j=1,i\neq j}^{n} |\frac{a_{ij}}{a_{ii}}| \leq 1 \quad i = 1, 2, \ldots, n$$

[12]

Example 13. An illustration of the Jacobi and Gauss-seidel methods can be given as: Consider the system:

$$Ax = b$$

with

$$A = \begin{pmatrix} 10 & -2 & -1 & -1 \\ -2 & 10 & -1 & -1 \\ -1 & -1 & 10 & -2 \\ -1 & -1 & -2 & 10 \end{pmatrix}$$

and

$$b = \begin{pmatrix} = 3 \\ 15 \\ 27 \\ -9 \end{pmatrix}$$

In order to solve by iterative methods, the system can be written as:

$$x_1 = 0.3 + 0.2x_2 + 0.1x_3 + 0.1x_4$$

$$x_2 = 1.5 + 0.2x_1 + 0.1x_3 + 0.1x_4$$

$$x_3 = 2.7 + 0.1x_1 + 0.1x_2 + 0.2x_4$$

$$x_4 = -0.9 + 0.1x_1 + 0.1x_2 + 0.2x_3$$

It is easily verified that the sum of abolute value of ratio of coefficients in non diagonal entries to that of the diagonal entries is less than unity for at least one equation, whereby the methods of Jacobi and Gauss-Siedel would be convergent. The tables for Gauss-Siedel and jacobi iterations are given as:

Table for Gauss-Siedel iterations

n	x_1	x_2	x_3	x_4
1	0.3	1.56	2.886	-0.1368
2	0.8869	1.9523	2.9566	-.0248
3	0.9836	1.9899	2.9924	-0.0042
4	0.9968	1.9982	2.9987	-0.0008
5	0.9994	1.9997	2.9998	-0.0001
6	0.9999	1.9999	3.0	0.0
7	1.0	2,0	3.0	0.0

Table for Jacobi iterations

n	x_1	x_2	x_3	x_4
1	0.3	1.5	2.7	-0.9
2	0.78	1.74	2.7	-0.18
3	0.9	1.908	2.916	-0.108
4	0.9624	1.9608	2.9592	-0.036
5	0.9845	1.9848	2.9851	-0.0158
6	0.9939	1.9938	2.9938	-0.006
7	0.9975	1.9975	2.9976	-0.0025
8	0.9990	1.9990	2.9990	-0.0010
9	0.9996	1.9996	2.9996	-0.0004
10	0.9998	1.9998	2.9998	-0.0002
11	0.9999	1.9999	2.9999	-0.0001
12	1.0	2.0	3.0	0,0

It is clear from the above tables that the first method, or, the Gauss-Siedel method requires only seven iterations which is equal to twelve iterations as required by the second method, or Jacobi method.

[14]

Chapter 8

Conjugate Gradient Method

[11](pp.13-17)

Though the Gauss-Siedel and Jacobi methods of iteration are good for solving systems of equations, but much faster iterative methods exist and these rest on optimizing projections onto a vector space known as *Krylov subspace*. This method applies only to positive definite symmetric matices. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.

A quadratic form is simply a scalar, quadratic function of a vector with the form

$$f(x) = \frac{1}{2}x^*Ax - b^*x + c$$

with A a complex square matrix, x, b vectors, and c a scalar.

The gradient of a quadratic form is defined to be

$$f'(x) = \begin{pmatrix} \frac{\partial}{\partial x_1}f(x) \\ \frac{\partial}{\partial x_2} \\ \cdots \\ \cdots \\ \cdots \\ \frac{\partial}{\partial x_n}f(x) \end{pmatrix}$$

where $x_1, x_2 \ldots, x_n$ are the components of x. The gradient is a vector field that, for a given point x, points in the direction of greatest increase of $f(x)$. By the usual theorem in calculus, the minimum value of the quadratic form occurs when the gradient is equal to zero.

We have,

$$f'(x) = \frac{1}{2}A^*x + \frac{1}{2}Ax - b$$

for hermitian and positive definite matrix A, the above equation reduces to:

$$f'(x) = Ax - b$$

Setting the above equation equal to zero, we obtain;

$$Ax = b$$

which is the original system to solve. Thus, it is clear that for a positive definite hermitian matrix system, solving the system is equivalent to finding the critical point of the quadratic form associated with the matrix.

Thus, the problem of solving a system of equations in which the matrix is positive definite hermitian is equivalent to finding the critical points of the associated quadratic form, which can be done geometrically by methods such as steepest descent, conjugate directions or conjugate gradients. Note that geometrically, a quadratic form looks like a paraboloid, hence there exists a well defined critical point. This is the basis of our interest in iterative methods like conjugate gradients. We briefly discuss those.

Note that the above process of finding the critical point may get converted to finding a hyperplane in case the matrix is singular and non-symmetric, or , may have no solution in some cases, when the equating of the gradient of quadratic form to zero gives us a saddle point.

8.1 Steepest Descent Method

[11](pp.13-17)

In the method of steepest descent, geometrically speaking, we start with an arbitrary point x_0 and slide down the paraboloid(quadratic form) to the critical point. However in order to be accurate, we take series of steps x_1, x_2, x_3 until we reach the right critical point.

When a step is taken, the direction in which f decreases most quickly is taken, which is the direction opposite to the gradient of $f(x_i)$. According to the equation of gradient, this means the direction is

$$-f'(x_i) = b - Ax_i$$

Let x be the required solution. Then, $e_i = x_i - x$, the error determines how far from the real solution is the iteration. Similarly, the residual $r_i = b - Ax_i$ denotes how far from the correct value of b is the iteration. Thus, $r_i = -Ae_i = -f'(x_i)$, thus indicating that the residual is the direction of steepest descent.

Thus,

$$x_1 = x_0 + \alpha r_0$$

where x_1 is the first direction that tis taken to minimize f.

A line search is a procedure that chooses α to minimize along a line. Geometerically speaking, wee are restricted to choosing a point on the intersection of the vertical plane and the paraboloid, which is the parabola defined by the intersection of these surfaces.

We have, α minimizes f when the directional derivative $\frac{d}{d\alpha}f(x_1)$ is equal to zero. By chain rule,

$$\frac{d}{d\alpha}f(x_1) = f'(x_1)^* \frac{d}{d\alpha}x_1 = f'(x_1)^* r_0$$

Thus, setting the above expression zero, we observe that α should be chosen such that r_0 and $f'(x_1)$ are orthogonal.

There is an intuitive reason why we should expect these vectors to be orthogonal at the minimum.

The slope of the parabola, which is the surface along which we are minimizing at any point, is equal to the magnitude of the projection of the gradient onto the line. These projections represent the rate of increase of as one traverses the search line. is minimized where the projection is zero — where the gradient is orthogonal to the search line.

We have [11](pp.6):

$$\implies f'(x_1) = -r_1$$

$$\implies r_1^* r_0 = 0$$

$$\implies (b - Ax_1)^* r_0 = 0$$

$$\implies (b - A(x_0 + \alpha r_0))^* r_0 = 0$$

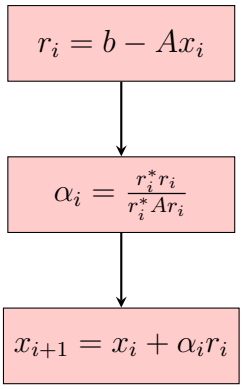
$$\implies (b - Ax_0)^* r_0 - \alpha (Ar_0)^* r_0 = 0$$

$$\implies (b - Ax_0)^* r_0 = \alpha (Ar_0)^* r_0$$

$$\implies r_0^* r_0 = \alpha r_0^* (Ar_0)$$

$$\implies \alpha = \frac{r_0^* r_0}{r_0^* A r_0}$$

Thus, the algorithm of steepest descent can be described as:



The above algorithm is run till it converges.

The computational cost in the above algorithm can be reduced by premultiplying the last equation above both sides by $-A$ and adding b

$$r_{i+1} = r_i - \alpha_i A r_i$$

We can see that the steepest descent algorithm converges, we use the analysis using eigenvalues and eigenvectors. Let us consider that e_i is an eigenvector with eigenvalue λ. Then, the residual is also an eigenvector, as $r_i = -Ae_i = -\lambda e_i$. Thus,

$$\begin{aligned} e_{i+1} &= e_i + \frac{r_i^* r_i}{r_i^* A r_i} r_i \\ &= e_i + \frac{r_{i^* r_i}}{\lambda r_{i^* r_i}} (\lambda e_i) \\ &= 0 \end{aligned}$$

To analyze convergence for a little more general error term e_i, we express e_i as a linear combination of orthogonal eigenvectors, whcih is guaranteed for the matrix A by the spectral theorem. Hence, we write [11](pp.15)

$$e_i = \sum_{j=1}^{n} \epsilon_j v_j$$

where ϵ_j is the length of each component of e_i and v_j are orthonormal eigenvectors. Thus, from the orthonormality of eigenvectors, we obtain

$$r_i = -Ae_i = -\sum_j \epsilon_j \lambda_j v_j$$

$$\implies \|e_i\|^2 = e_i * e_i = \sum_j \epsilon_j^2$$

$$\implies e_i * Ae_i = (\sum_j \epsilon_j v_j^*)(\sum_j \epsilon_j \lambda_j v_j)$$

$$\implies e_i * Ae_i = \sum_j \epsilon_j^2 \lambda_j$$

$$\implies \|r_i\|^2 = r_i * r_i = \sum_j \epsilon_j^2 \lambda_j^2$$

$$r_i * Ar_i = \sum_j \epsilon_j^2 \lambda_j^3$$

By the mechanics of the method of steepest descent, we obtain

$$e_{i+1} = e_i + \frac{r_i^* r_i}{r_I^* A r_i} r_i$$

$$= e_i + \frac{\sum_j \epsilon_j^2 \lambda_j^2}{\sum_j \epsilon_j^2 \lambda_j^3} r_i$$

If it is assumed that all eigenvectors v_j have a common eigenvalue λ, we obtain

$$e_{i+1} = e_i + \frac{\lambda^2 \sum_j \epsilon_j^2}{\lambda^3 \sum_j \epsilon_j^2} (\lambda e_i)$$

$$= 0$$

thus showing the convergence.

For an even more general analysis, when the eigenvalues corresponding to the orthogonal vectors are unequal, we use the enegy norm defined by

$$\|e\|_A = (e^* A e)^{\frac{1}{2}}$$

Then, we have

$$\|e_{i+1}\|_A^2 = e_{i+1}^* A e_{i+1}$$

From the mechanics of steepest descent procedure, we have:

$$= (e_i^* + \alpha_i r_i^*)(A(e_i + \alpha_i r_i)$$

Using symmetry of A:

$$= e_i^* A e_i + 2\alpha_i r_i^* A e_i + \alpha_i^2 r_i^* A r_i$$

$$= \|e_A\|^2 + 2\frac{r_i^* r_i}{r_i^* A r_i}(-r_i^* r_i) + \left(\frac{r_i^* r_i}{r_i^* A r_i}\right)^2 r_i^* A r_i$$

$$= \|e_i\|_A^2 - \frac{(r_i^* r_i)^2}{r_i^* A r_i}$$

$$= \|e_i\|_A^2 \left(1 - \frac{(r_i^* r_i)^2}{(r_i^* A r_i)(e_i^* A e_i)}\right)$$

$$= \|e_i\|_A^2 \left(1 - \frac{(\sum_j \epsilon_j^2 \lambda_j^2)^2}{(\sum_j \epsilon_j^2 \lambda_j^3)(\sum_j \epsilon_j^2 \lambda_j)}\right)$$

$$= \|e_i\|_A^2 \omega^2 \quad \omega^2 = 1 - \frac{(\sum_j \epsilon_j^2 \lambda_j^2)^2}{(\sum_j \epsilon_j^2 \lambda_j^3)(\sum_j \epsilon_j^2 \lambda_j)}$$

We define

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$$

where λ_{max} and λ_{min} are the maximum and minimum eigenvalues as the condition number of the matrix A. [11]

Then, we have:

$$\|e_i\|_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i \|e_o\|_A$$

and

$$\frac{f(x_i) - f(x)}{f(x_0) - f(x)} = \frac{\frac{1}{2} e_i^* A e_i}{\frac{1}{2} e_0^* A e_0}$$

$$\leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^{2i}$$

which thus guarantees us convergence provided the matrix is well conditioned.

8.2 Method of Conjugate directions

[11](pp.21-25)

Let d_i be the orthogonal search directions, that is vectors which approximate the solution from the initial vector, that is,

$$x_{i+1} = x_i + \alpha_i d_i$$

We set the directional derivative of the quadratic form associated to successive iteration to zero to obtain:

$$\frac{d}{d\alpha} f(x_{i+1}) = 0$$

$$\implies f'(x_{i+1})^* \frac{d}{d\alpha} x_{i+1} = 0$$

$$-r_{i+1}^* d_i = 0$$

$$d_i^* A e_{i+1} = 0$$

By orthogonality, we have:

$$d_i^* e_{i+1} = 0$$

$$\implies d_i^* (e_i + \alpha_i d_i) = 0$$

$$\alpha_i = -\frac{d_i^* e_i}{d_i^* d_i}$$

In the method of conjugate directions, we make the search directions d_i A-orthogonal instead orthogonal, where d_i, d_j are A-orthogonal if:

$$d_i^* A d_j = 0$$

Then, our α_i becomes:

$$\alpha_i = -\frac{d_i^* A e_i}{d_i^* A d_i}$$
$$= \frac{d_i^* r_i}{d_i^* A d_i}$$

to prove that the procedure computes the solution in n steps, we express the error term as a linear combination of search directions:

$$e_0 = \sum_{j=0}^{n-1} \delta_j d_j$$

where to calculate δ_j, the following steps are used:

$$d_k^* A e_0 = \sum_j \delta_j d_k^* A d_j$$

by the A- orthogonality of the search directions,we get:

$$\implies d_k^* A e_0 = \delta_k d_k^* A d_k$$

$$\implies \delta_k = \frac{d_k^* A e_0}{d_k^* A d_k}$$

$$= \frac{d_k^* A (e_0 + \sum_{i=0}^{k-1} \alpha_i d_i)}{d_k^* A d_k}$$

$$= \frac{d_k^* A e_k}{d_k^* A d_k}$$

Thus, we have:

$$e_i = e_0 + \sum_{j=0}^{i-1} \alpha_j d_j$$

$$= \sum_{j=0}^{n-1} \alpha_j d_j - \sum_{j=0}^{i-1} \alpha_j d_j$$

$$= \sum_{j=i}^{n-1} \alpha_j d_j$$

After n iterations, every component vanishes, thus showing the convergence.

8.3 Method of Gram-Schmidt conjugation

[11](pp.25-30)

The A-orthogonal search directions d_i in the method of conjugate directions is produced by a method known as Gram-Schmidt conjugation.

The method is to take n linearly independent vectors $u_0, u_1, \ldots, u_{n-1}$, take any u_i and subtract out any components that are not A- orthogonal to the previous d vectors. The formulaic description of the process is:

$$d_0 = u_0$$

and

$$d_i = u_i + \sum_{k=0}^{i-1} b_{ik} d_k$$

where b_{ik} are found as:

$$d_i^* A d_j = u_i^* A d_j + \sum_{k=0}^{i-1} b_{ik} d_k^* A d_j$$

$$\implies 0 = u_i^* A d_j + b_{ij} d_j^* A d_j \quad (i > j)$$

$$\implies b_{ij} = -\frac{u_i^* A d_j}{d_j^* A d_j}$$

It can be shown that the procedure of Gram-Schmidt conjugation requires $\sim n^3$ floating point steps, similar to the Gaussian elimination due to the process that the old search vectors must be kept in memeory. To circumvent this crucial problem, the method of conjugate gradients was develeoped, which is described in next section.

It is also to be noted that the method of conjugate directions, the error terms e_i are chosen such that the value $e_0 + D_i$ minimizes the energy norm $\|e_i\|_A$ where $D - i$ is the space spanned by the vectors $\{d_o, d_1, \ldots, d_{(i-1)}\}$. The energy norm can be expressed as a sum of linear sum as follows:

$$\|e_i\|_A = \sum_{j=1}^{n-1} \sum_{k=i}^{n-1} \delta_j \delta_k d_j^* A d_k$$

By the A-orthogonality of the d vectors, we obtain:

$$= \sum_{j=1}^{n-1} \delta_j^2 d_j^* A d_j$$

The above equation proves the optimality of conjugate directions, as Each term in this summation is associated with a search direction in its that has not yet been traversed. that Any other vector e chosen from $e_0 + D_i$ must have the same terms in its expansion, which shows that e_i must have the minimum energy norm. [11]

An important property of the method of conjugate directions is that the vectors r_i are orthogonal to D_i. This is because:

$$-d_i^* A e_j = -\sum_{j=i}^{n-1} \delta_j d_i^* A d_j$$

$$\implies d_i^* r_j = 0$$

by the A-orthogonality of the d_i vectors.

Again, because the search directions d_i are constructed from the u vectors, the subspace spanned by $u_0, u_1, \ldots, ui - 1$ is also $D - i$ and hence the residual vectors r_i are also orthogonal to u vectors as well. This can also be proven as follows:

$$d_i^* r_j = u_i^* r_j + \sum_{k=0}^{i-1} b_{ik} d_k^* r_j$$

$$\implies u_i^* r_j = 0$$

The above equations also give rise to the following identity, which would be used in the next section:

$$d_i^* r_i = u_i^* r_i$$

As with steepest descent, the number of iterations to find the residuals can be reduced by one as follows:

$$r_{i+1} = -A e_{i+1}$$

$$= -A(e_i + \alpha d_i)$$

$$= r_i - \alpha A d_i$$

8.4 The conjugate gradient method

[11](pp.30-35)

The method of conjugate gradients is the method of conjugate directions with the search vectors built by conjugation of residuals(by setting $r_i = u_i$). Thus the space spanned by $\{r_0, r_1, \ldots, r)i-1\}$ is the same as D_i of the last section. Thus, we have:

$$r_i^* r_j = 0$$

and

$$D_i = span\{d_0, Ad_0, \ldots, A^{i-1}d_0\}$$

$$= span\{r_0, Ar_0, \ldots, A^{i-1}r_0\}$$

This is key step in the conjugate gradient method but you do not explain why

$$D_i = \text{span}\{d_0, Ad_0, \ldots, A^{i-1}d_0\} = \text{span}\{r_0, r_1, \ldots, r_{i-1}\}.$$

The space D_i is known as Krylov subspace. It has the property that AD_i is included in D_{i+1}. From the last section on Gram-Schmidt conjugation, we have:

$$r_i^* r_{j+1} = r_i r_j - \alpha r_i^* A d_j$$

$$= \alpha_j r_i^* A d_j = r_i^* r_j - r_i^* r_{j+1}$$

$$\implies r_i^* A d_j = \begin{cases} \frac{1}{\alpha_i} r_i^* r_i & i = j \\ -\frac{1}{\alpha_{i-1}} r_i^* r_i & i = j+1 \\ 0 & otherwise \end{cases}$$

$$b_{ij} = \begin{cases} \frac{1}{\alpha_{i-1}} \frac{r_i^* r_i}{d_{i-1}^* A d_{i-1}} & i = j+1 \\ 0 & i > j+1 \end{cases}$$

Thus, most of the b_{ij} terms have disappeared which is what makes the conjugate gradient method better than other methods.

Thus, the summary of the method of conjugate gradients is :

$$d_0 = r_0 = b - Ax_0$$

$$\alpha_i = \frac{r_i^* r_i}{d_i^* A d}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

$$\beta_{i+1} = \frac{r_{i+1}^* r_{i+1}}{r_i^* r_i}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i$$

[11] In theory (with exact arithmetic) converges to solution in n steps. [15]

- The bad news: due to numerical round-off errors, can take more than n steps (or fail to converge).
- The good news: with luck (i.e., good spectrum of A), can get good approximate solution in $\ll n$ steps.
- Each step requires z $\rightarrow$ Az multiplication – can exploit a variety of structure in A.
- In many cases, never form or store the matrix A.
- Compared to direct (factor-solve) methods, CG is less reliable, data dependent; often requires good (problem-dependent) preconditioner
- But, when it works, can solve extremely large systems.

At each step of the Conjugate Gradient method, the value of error vector e_i is chosen from $e_0 + D_i$, where

$$D_i = span\{r_o, Ar_0, A^2r_0, \ldots, A^{i-1}r_0\}$$

$$= span\{Ae_0, A^2e_0, A^3e_0, ldots, A^ie_0\}$$

[11](pp.33)

In addition, such Krylov spaces have the additional property that for a fixed i , the error term has the form:

$$e_i = \left(I + \sum_{j=1}^{i} s_j A^j\right) e_0$$

where s_j are chosen that minimizes $\|e_i\|_A$

The expression in the paranthesis can be expressed as a polynomial. That is,

$$e_i = P_i(A)e_0$$

where P_i is a polynomial of degree i and $P_i(0) = 1$. The method of conjugate gradients chooses the polynomial when it chooses the λ_i coefficients. We have by decomposing the error as a sum of orthogonal unit vectors s_i:

$$e_0 = \sum_{j=1}^{n} s_{j_j} v_j$$

$$\implies e_i = \sum_j s_j P_i(\lambda_j) v_j$$

$$\implies Ae_i = \sum_j s_j P_i(\lambda_j) \lambda_j v_j$$

$$\implies \|e_i\|_A^2 = \sum_j s_j^2 [P_i(\lambda_j)]^2 \lambda_j$$

The conjugate gradient method chooses the polynomial that minimizes the above expression, but convergence is only as good as convergence of the worst eigenvector. If $L(A)$ denotes the spectrum, or the set of eigenvalues of A, we have

$$\|e_i(A)\|_A^2 \leq min_{P_i} max_{\lambda \in L(A)} [P_i(\lambda)]^2 \sum_j s_j^2 \lambda_j$$

$$= min_{P_i} max_{\lambda \in L(A)} [P_i(\lambda)]^2 \|e_0\|_A^2 e_i$$

In order to minimize the above expression, Chebyshev polynomials T_i are used in place of P_i. The Chebyshev polynomials are defined as

$$T_i(x) = \frac{1}{2}[(x + \sqrt{x^2 - 1})^i + (x - \sqrt{x^2 - 1})^i]$$

It can be shown that the expression for error term is minimized by taking

$$P_i(\lambda) = \frac{T_i(\frac{\lambda_{max} + \lambda_{min} - 2\lambda}{\lambda_{max} - \lambda_{min}})}{T_i(\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}})}$$

In addition, $P - i(0) = 1$. The value of maximum of the numerator of the above expression is 1 in the interval $[\lambda_{max}, \lambda_{min}$. Therefore, we have:

$$\|e_i\|_A \leq T_i(\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}})^{-1} \|e_0\|_A$$

$$= T_i(\frac{k+1}{k-1})^{-1} \|e_0\|_A$$

$$= 2 \left[\left(\frac{\sqrt{k}+1}{\sqrt{k}-1} \right)^i + \left(\frac{\sqrt{k}-1}{\sqrt{k}+1} \right)^i \right]^{-1} \|e_0\|_A$$

where k is the condition number of the matrix.

Since the second expression in the square brackets above tends to zero, therefore, the convergence of conjugate gradients can be expressed as

$$e_i\|_A \leq 2 \left(\frac{\sqrt{k}-1}{\sqrt{k}+1} \right)^i \|e_0\|_A$$

We note that the first step of conjugate gradients is identical to that of steepest descent.

8.5 Preconditioned Conjugate gradient method

[11](pp.39-41)

Preconditioning refers to the process of improving the condition number of a matrix. Suppose that a matrix B refers is hermitian and positive definite that approximates A and is easier to invert. Then, $Ax = b$ can be indirectly solved by solving:

$$B^{-1}Ax = B^{-1}b$$

If the condition number of the matrix $B^{-1}A$ is very less as compared to that of A, or, if the eigenvalues of $B^{-1}A$ are better clustered than those of A, then the computational cost associated to the problem can be reduced by solving $B^{-1}Ax$. But, the main difficulty with this method would be that positive definiteness of neither A nor B ensures the positive definiteness of $B^{-1}A$. To circumvent this difficulty, the help of Cholesky decomposition is taken, whereby, we have a matrix M for every positive definite matrix B, such that $MM^* = B$. No

for every eigenvector v of $B^{-1}A$ with eigenvalue λ, we have:

$$sec(M^{-1}AM^{-*})(M^*v) = (M^*M^{-*})M^{-1}Av = t^iB^{-1}Av = \lambda M^*v$$

whereby M^*v is the eigenvector of $M^{-1}AM^{-*}$ corresponding to λ. [11](pp.39)

Hence, the above method suggests solving the original positive definite system $Ax = b$ by solving

$$M^{-1}AM^{-*}\hat{x} = M^{-1}b \quad , \hat{x}M^*x$$

which is first solved for $\hat{x}$ and then for x. In this case, $M^{-1}AM^{-*}$ is hermitian and positive definite and $\hat{x}$ can be found by steepest descent or the method of conjugate gradients. The method of using conjugate gradient to solve this equation is known as the transformed preconditioned conjugate gradient method, described as:

$$\hat{d}_0 = \hat{r}_0 = M^{-1}b - M^{-1}AM{-}*\hat{x}_0$$

$$\alpha_i = \frac{\hat{r}_i^*\hat{r}_i}{\hat{d}_i^*M^{-1}AM^{-*}\hat{d}_i}$$

$$\hat{x}_{i+1} = \hat{x}_i + \alpha_i\hat{d}_i$$

$$\hat{r}_{i+1} = \hat{r}_i - \alpha_i\hat{d}_i$$

$$\hat{b}_{i+1} = \frac{\hat{r}^*_{i+1}\hat{r}_{i+1}}{\hat{r}^*_i\hat{r}_i}$$

$$\hat{d}_{i+1} = \hat{r}_{i+1} + b_{i+1}\hat{d}_i$$

The above method requires computation of M, where $MM^* = B$, which can be eliminated by making some substitutions: We set $\hat{r}_i = M^{-1}r_i$ and $\hat{d}_i = M^*D_i$ and using the formulae $\hat{x}_i = M^*x_i$ and $M^{-*}M^{-1} = B^{-1}$, we obtain the untransformed preconditioned conjugate gradient method as follows:

$$r_o = b - Ax_0$$

$$d_0 = B^{-1}r_0$$

$$\alpha_i = \frac{r^*_i B^{-1} r_i}{d^*_i A d_i}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

$$b_{i+1} = \frac{r^*_{i+1} B^{-1} r_{i+1}}{r^*_i B^{-1} r_i}$$

$$d_{i+1} = B^{-1}r_{i+1} + b_{i+1}d_i$$

We see that computations involving the matrix M was avoided in the untransformed* method. [11](pp.40)

The effectiveness of the preconditioned method rely on the effectiveness of the conditioner B which in turn is related to the condition number of $B^{-1}A$ and by the clustering of its eigenvalues.

Chapter 9

Conclusion

In our quest to discuss the implementation and performance of direct and iterative methods and possibly come to a conclusion about certain methods. We discovered a lot and acquired a lot of information. In Chapter 3 we discussed the SVD and the QR factorisation. We explored their concept, explored their computations in examples. We gave you insight into using methods such as Gram-Schmidt or-thogonalization, Householder transformations, or Givens rotations. We then explored their computational costs. Hence giving you information about the implementation and the performance of such methods. We explored the benefits and disadvantages of each method. Exploring this helped us gain insight into implementation and performance of such methods. In Chapter 4, the same was done by exploring the general system of equations. The case where an equation is non-square was explored. The methods to implement in such cases such as QR factorization and SVD was discussed also. We discussed their performances and their implementation. We found that we were faced with an issue of speed versus accuracy. Hence leading us to further discuss performance and implementation. An issue which naturally lead us to chapter 5. A chapter where we discussed stability and conditioning of numerical problems. This was essential, as we needed it to explore what we mainly wanted to focus on. This was Direct and iterative methods for a square matrix. It was to gain knowledge to help us discuss the performance of an implemented method in depth. In this chapter we simultaneously managed to also discuss floating point arithmetic and how it was linked or related to stability and conditioning. This therefore gave us an idea into the effects of implementation of problems on computer and its effects on the accuracy of the answer received. A major help with regards to our quest in finding out about the performance on computers with regards to accuracy. We also found out that, we needed to use well conditioned algorithms to be able to get a more accurate answer. Overall aiding in our understanding regarding performance of an algorithm used to solve a system of equation. In this

same chapter we were also able to embark naturally to chapter 6 because it was observed that the use of least squares problem gives $A * Ax = A * b$. Meaning it actually reduces the original system to solving $Ax = b$ for square matrices. And this is the square system that you solve. After this chapter, we were equipped to not only explore theory and implementation of direct and iterative methods. We were now equipped to analyze its accuracy, stability and its computational costs. Thus giving us insight into exploring more stable methods and insight into into solving almost any system of equation for an nxn matrix for large n. helping us find out just how much less computation was involved with iterative methods as opposed to direct methods etc. Thus, we have discussed the various intricacies involved in solving a general system of equations , how we reduce them to solving systems of square matrices which discussed topics such as least squares, normal equations, SVD, pseudoinverses, QR Decomposition , the complications involved in solving them on computers, including conditioning, accuracy and stability, further complications in solving equations on a large scale and methods to overcome them, which included various methods of direct solutions like LU Decomposition and associated Gaussian elimination as well as Cholesky decomposition, and also iterative methods like Jacobi, Gauss-Siedel for non-sparse matrices and conjugate directions, steepest descent and conjugate gradients for sparse matrices. We also discussed stability of the various algorithms involved. As you know, this thesis was based on the **implementation** and the **performance** of such methods. We have given you an insight into their performances, and which methods are appropriate to use depending on the nature of your problem. In the jist, it is difficult to say beforehand which method needs to be taken for solving a given set of equations. Rather, as you can see for your self, it requires deep analysis of the given conditions and stability pertaining to its computation and the nature of the method, whether sparse or not before deciding the method to solve a given set of equations. We hope this study has made a right introduction in this direction. And we hope you have gained insight in the performance and implementation of direct and iterative methods. In a proposed future work, we could plan to do some implementations and simulations on real examples so as to better understand the theory in practice. Preferbly on a programme such as matlab, where the accuracy and the time taken to solve, if given to you after calculation on matlab.

Bibliography

[1] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[2] Nicolas Bourbaki. *Algebra II: Chapters 4-7*. Springer Science & Business Media, 2013.

[3] Mike Warby. Advanced calculus and numerical methods lecture notes. March 2015/6.

[4] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

[5] Radu Trimbitas. Householder reflections and givens rotations, March 2009.

[6] Omer Egecioglu and Ashok Srinivasan.

[7] F Gerrish. Reduction of a square matrix to triangular form. *The Mathematical Gazette*, 64(430):266–270, 1980.

[8] Saeid Abbasbandy, Reza Ezzati, and Ahmad Jafarian. Lu decomposition method for solving fuzzy system of linear equations. *Applied Mathematics and Computation*, 172(1):633–643, 2006.

[9] Pavel Okunev and Charles R Johnson. Necessary and sufficient conditions for existence of the lu factorization of an arbitrary matrix. *arXiv preprint math/0506382*, 2005.

[10] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22, 2008.

[11] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[12] Shankar S Sastry. *Introductory methods of numerical analysis*. PHI Learning Pvt. Ltd., 2012.

[13] James Hardy Wilkinson. *The algebraic eigenvalue problem*, volume 87. Clarendon Press Oxford, 1965.

[14] Eric W Weisstein. Gauss-seidel method. 2002.

[15] David G Luenberger. Introduction to linear and nonlinear programming. 1973.