Computation of nodes and weights of Gaussian quadrature rule by using Jacobi's method

By

Raja Zafar Iqbal

A thesis submitted to
The University of Birmingham
for the Degree of
Master of Philosophy

School of Mathematics
The University of Birmingham
2008

Acknowledgements

I would like to thank my supervisor, *Professor Roy Mathias*, for his excellent guidance, patience and advice throughout this dissertation. Without his support it would not be possible in short span of time.

I wish to thank *Prof C.Parker , Dr Sharon Stephen, Dr Patricia Odber and Mrs J Lowe* for their magnificent support, professional advice and experience to overcome my problems.

I am grateful to my colleagues *Muhammad Ali*, *S.Ghufran* and *Nour Uddin* for their help and support, and *Habib ur Rehman* for his kindness.

I am also very grateful to my *parents* and my *wife* for their support and trust throughout these long hard times. Further I would like to thank my family and especially to my kids how missed me a lot.

Abstract

Numerical analysis has become important in solving large eigenvalue problems in science and engineering due to the increasing spread of quantitative analysis. These problems occur in a wide variety of applications. Eigenvalues are very useful in engineering for the dynamic analysis of large-scale structures such as aerospace. There is also a useful connection between nodes and weights of Gaussian quadrature and eigenvalues and eigenvectors. Thus the need for faster methods to solve these larger eigenvalue problems has become very important.

A standard textbook method for finding the eigenvalues of a matrix A is to solve for the roots λ of

$$| A - \lambda I |= 0$$

It is quite easy to solve analytically when matrix is small but it is more difficult for large matrices. For such problems numerical methods are used to find eigenvalues and corresponding eigenvectors. There are numerous numerical methods for the solution of the symmetric eigenvalue problems. Of these the QR algorithm, Cholesky iteration and Jacobi rotational methods are commonly used.

In this project we checked rate the of convergence and accuracy of the Cholesky-iterative method and the Jacobi's method for finding eigenvalues and eigen vectors and found that the Jacobi's method converges faster than the Cholesky method. Then by using "three-term recurrence relation" we calculated nodes and weights of Gaussian quadrature by eigenvalues and eigenvectors. The nodes and weights computed were found to be highly accurate, so this method allows one to perform Gaussian Quadrature without using standard tables of nodes and weights, saving time and avoiding the risk of errors in entering the nodes and weights from tables.

Dedicated to

My Wife , Sons and Daughters

They sacrificed a lot for my studies

Contents

Chapter 1

Introduction

Numerical methods for solving large eigenvalue problems in science and engineering have gained much importance. This is due to the increasing spread of quantitative analysis. The eigenvalue problems occur in a wide variety of applications like prediction of structural responses in solid and soil mechanics. These are very useful in structural-engineering as well.

There is a useful connection between nodes and weights of Gaussian quadrature and eigenvalues and eigenvectors, and this yields a fast and an accurate method to compute the nodes an weights for Gaussian Quadrature. Thus the need for faster methods to solve these larger eigenvalue problems has become very important.

The standard method for finding the eigenvalues of a matrix A is to solve for the roots λ of

$$| A - \lambda I | = 0$$

where λ is eigenvalue of matrix A. If size of matrix A is small then it is quite easy to solve it analytically. Whereas for large and sparse matrix it is more difficult to find eigenvalues using the direct methods. For such problems numerical iterative methods are used to find eigenvalues and corresponding eigenvectors.

Eigenvalues are often studies in the context of linear algebra and matrix analysis. These are being studied and are in use since $20th$ century. Hilbert studied the integral operators by viewing the operator as infinite matrices. In 1904, he was first to use the German word eigen to denote "characteristics". Where as the Power method by Van Mises, in 1929 was the first numerical algorithm for computing eigenvalues and eigenvectors.

There are numerous numerical methods for computation of eigenvalue problems in these days.. All these methods reduce matrix A to a simpler form. Of these the QR algorithm, Cholesky decomposition and Jacobi rotational methods are commonly used. But the Jacobi method is fast convergent and more accurate for finding eigenvalues of Hermitian matrices.

In this project we checked the rate of convergence and accuracy of the Cholesky-iterative method and the Jacobi method for finding eigenvalues and eigenvectors and found that the Jacobi method is fast convergent than the Cholesky method. Then by using "three-term recurrence relation" we calculated nodes and weights of Gaussian quadrature by eigenvalues and eigenvectors.

This thesis consists of five chapters.

Chapter 1 : Introduction

This chapter is about the introduction of eigenvalue problems and their computations. Various method are discussed which would be used in later work.
Brief introduction is presented about next four chapters.

Chapter 2 : Gaussian Quadrature and Eigenvalue Problems

The main aim of this chapter is to see which numerical integration method is better one. For this purpose a model problem of definite integral is selected.

Model Problem

Evaluate the integral

$$I = \int_0^1 sinxdx \tag{1.1}$$

Exact value (analytical solution) of problem calculated by simple integration rules is calculated equal to $4.59769769413186e-001$.
If we plot sine function for these limits, a curve is obtained. Now we apply numerical integration rules to measure the area under the curve. We shall apply Newton-Cotes rules and Gaussian quadrature formulae to find numerical integration for different n values. Then we compare errors due to these methods. Connection between nodes and weights of Gaussian quadrature formula and eigenvalues and eigenvectors turn our attention to Hermitian

eigenvalue problems.

Chapter 3 : The Hermitian Eigenvalue Problems

The main goal of this chapter is to give some introduction about matrices. Various decompositions of matrices are discussed and through Matlab codes results are being analyzed.

Chapter 4 : Accuracy of Methods

In this chapter we build Matlab codes for iterative Cholesky and Jacobi methods and check the convergence for various order of matrices. We see that Jacobi's method is fast convergent and more accurate than the Cholesky iterative method.

Chapter 5 : Applying Methods to compute weights and nodes of Gaussian quadrature

Finally we shall compute nodes and weights of Gaussian quadrature through eigenvalues and eigenvectors by using Matlab codes. Conclusion of work and future work be presented in this last chapter.

Chapter 2

Gaussian Quadrature and Eigenvalue Problems

2.1 Numerical Integration

Mathematicians and scientists are sometime confronted with definite integrals which are not easily evaluated analytically, even a function $f(x)$ is known completely. To overcome this difficulty numerical methods are used. Numerical integration involves replacing an integral by a sum. The term *quadrature* is used as a synonym for numerical integration in one dimension. Let $f(x)$ be a function which is defined on some interval [a,b] and on the set of distinct points $\{x_0, x_1, ..., x_n\}$. Then the numerical integration for approximation can be defined as

$$\int_a^b f(x)\ dx \cong \sum_{i=0}^{n} w_i f(x_i) \tag{2.1}$$

where w_i are the *quadrature weights* and x_i the *quadrature points*. There are a number of numerical integration methods for evaluation of definite integrals. The most commonly used methods are the Newton-Cotes formulas and Gaussian quadrature rules. Here we shall give a brief introduction and implementation for these methods.

2.1.1 Newton-Cotes Formulas

The numerical integration methods that are derived by integrating the Newton interpolation formulas are termed as *Newton-Cotes integration formulas*. The Trapezoidal Rule and Simpson's Rules are members of this family.

2.1.2 Trapezoidal Rule

The trapezoidal rule is a numerical integration method derived by integrating the linear polynomial interpolation. It is written as

$$I = \int_a^b f(x)dx = \frac{b-a}{2}[f(a) + f(b)] + E \tag{2.2}$$

where E represents the error of trapezoidal rule. This error is high when we approximate the area under a curve by using single trapezoid [5]. The interval $[a, b]$ can be divided into n intervals with equal width h. These points are $a = x_0, x_1, x_2, ..., x_n = b$ where $x_i = x_0 + ih,$ for all $i = 1, 2, ..., n$.
The value h is given as

$$h = \frac{b-a}{n}$$

Above relation for n-interval case can be written as

$$I = \int_a^b f(x)dx = \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)] + E \tag{2.3}$$

$$I = \int_a^b f(x)dx = \frac{h}{2}[f(a) + 2f(x_1) + 2f(x_2) + ... + 2f(x_{n-1}) + f(b)] + E \tag{2.4}$$

If we replace $f(a) = f_0, f_1 = f(a+h)$ and $f_i = f(a+ih)$ then above relation can be expressed as

$$I = \frac{h}{2}[f_0 + 2f_1 + 2f_2 + ... + 2f_{i-1} + f_i] + E \tag{2.5}$$

The error of the trapezoidal rule is given as:

$$E = -\frac{1}{12}(b-a)h^2 f''(\xi) \tag{2.6}$$

where $a \leqslant \xi \leqslant b$
It is clear that the error of the trapezoidal rule is proportional to f'' and decreases proportionally to h^2 when we increase the number of intervals. The error is large for the single segment trapezoidal rule. To reduce this error, we divide the interval into subintervals and then apply the trapezoidal rule over each segment.

2.1.3 Simpson's $\frac{1}{3}$ Rule

Simpson's $\frac{1}{3}$ rule is based on quadratic polynomial interpolation. In general the Simpson's rule is used for equally spaced data with width h. Results obtained by the trapezoidal rule lead us to think that we might be able to do better than the trapezoidal rule by using high-degree polynomial [1].

For three points $x_0 = a, \quad x_1 = a + h$ and $x_2 = b$, Simpson's rule can be written as :

$$I = \int_a^b f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] + E \tag{2.7}$$

where E denote error in Simpson's rule which is obtained as:

$$E = -(b-a)\frac{h^4}{180}f^{iv}(\xi) \tag{2.8}$$

where $a \leqslant \xi \leqslant b$.

This error would be zero if $f(x)$ is a polynomial of degree 3 or less. Simpson's rule is easy to apply and it is considered reasonable for many applications. Error of Simpson's rule is high for $n = 2$ and its accuracy can be enhanced by dividing interval $[a, b]$ into several subintervals. These intervals should be even for Simpson rule. In general we write:

$$I = \frac{h}{3}[f(a) + 4\sum_{i=1,odd\,i}^{n-1} f(a+ih) + 2\sum_{i=2,even\,i}^{n-2} f(a+ih) + f(b)] + E \tag{2.9}$$

Setting $f_i = f(a + ih)$ in above relation we get

$$I = \frac{h}{3}[f_0 + 4(f_1 + f_3 + ... + f_{n-1}) + 2(f_2 + f_4 + ... + f_{n-2}) + f_n] + E \tag{2.10}$$

2.2 Gaussian Quadrature Rule

The numerical integral methods described earlier have a simple choice of points to evaluate a function $f(x)$ in a known interval [a,b]. These methods are based on equally space points. When we have freedom of choice regarding evaluation points, then more accuracy can be achieved. Gaussian quadrature is a powerful tool for approximating integrals. The quadrature rules are all based on special values of *weights* and *abscissas*. Abscissas are commonly called evaluation points or "Gauss points", which are normally pre-computed

and available in most standard mathematics tables. Algorithms and computer codes are also available to compute them.
The two-point Gauss quadrature rule for a function $f(x)$ can be evaluated between fixed limits a and b as follow :

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) \tag{2.11}$$

There are four unknowns, c_1, c_2, x_1 and x_2 which can determined by integrating a general third order polynomial, $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$, which has 4 coefficients.

2.2.1 Higher point Gaussian Quadrature Formulas

To get more accurate results, the number of Gaussian Quadratures are increased. For this three points or higher point Gaussian Quadrature rule can be used. Three points Gaussian Quadrature rule is written as:

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) + c_3 f(x_3) \tag{2.12}$$

When n points are used, we call the method an "n-point" Gaussian method, which can be used to approximate a function $f(x)$ between fixed limits as:

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) + ... + c_n f(x_n) \tag{2.13}$$

Gaussian Integration is simply based on the use of a polynomials to approximate the integrand $f(t)$ over the interval $[-1, 1]$. The accuracy and optimality of results depend on the choice of polynomial. The coefficients of this polynomial are unknown variables which can be determined by using any suitable method. The simplest form of Gaussian quadrature uses a uniform weighting over the interval. The particular points at which we have to evaluate $f(t)$ are the roots of a particular class of polynomials, the Legendre polynomials, over the interval. Gaussian quadrature formulae are evaluating using abscissae and weight. The choice of n is not always clear, and experimentations are useful to see the influence of choosing a different number of points.

In most cases we shall evaluate the integral on a more general interval .We shall use the variable x on interval [a,b], and linearly map this interval for x onto the [-1,1] interval for t.

2.2.2 Change of interval

If the integral is not posed over the interval $[-1, 1]$ then we can apply a simple change of variable to rewrite any integral over $[a, b]$ as an integral over $[-1, 1]$.
Let $x = mt + c$.
If $x = a$, then $t = -1$ and when $x = b$ then $t = 1$.
So

$$x = \frac{b-a}{2}t + \frac{b+a}{2} \tag{2.14}$$

After simplification we have

$$I = \int_a^b f(x)dx = \int_{-1}^{1} (\frac{b-a}{2}t + \frac{b+a}{2})\frac{b-a}{2}dt \tag{2.15}$$

The simplest form of Gaussian quadrature uses a uniform weights over the interval [-1,1]. The particular points at which $f(t)$ is evaluated are the roots of Legendre polynomials as we will see in the next section.

If n number of points are evaluated for the function $f(t)$ the resulting integral value is of the same accuracy as a simplest polynomial method like the Simpson rule with degree $2n$.
The high accuracy of Gaussian quadrature comes from the fact that it integrates very-high-degree polynomials exactly. In trapezoidal method and Simpson's rule we used a fixed grid points which are predetermined. A fixed degree of polynomial is integrated exactly over each subinterval.

Let I_n be a quadrature rule, and assume it is exact for all polynomials of degree $\leqslant p$. Then we say that I_n has **degree of precision** p.
In the Gaussian quadrature both grid points and the weights are chosen to maximize the degree of precision. The degree of polynomial increases proportionally with number of points used in quadrature rule. The degree of polynomial is $\leqslant 2n-1$ (where n is number of grid points), and coefficients of this polynomial are $2n$. Thus the number of unknowns (n weights and n abscissas) is equal to the number of equations to get exact solution.

2.2.3 Error in Gaussian Quadrature

The error in the Gaussian quadrature rule [1] is

$$E = I(f) - G_n(f) = \frac{(b-a)^{2n+1}(n!)^4 f^{(2n)}(\xi_n)}{(2n+1)[(2n)!]^3} \tag{2.16}$$

where $a < \xi_n < b$ and $I(f)$ and $G_n(f)$ denote values of function calculated analytically and by using the Gaussian quadrature formula.

2.2.4 Orthogonal Polynomials and Gaussian Quadrature

Orthogonal polynomials are classes of polynomials $p_n(x)$ defined over a range $[a, b]$ that obey an orthogonality relation

$$\int_a^b w(x)p_m(x)p_n(x)dx = \delta_{mn}c_n \tag{2.17}$$

where $w(x)$ is a weighting function and δ_{mn} is the the Kronecker delta. Orthogonal polynomials are very useful in finding the solution of mathematical and physical problems. Such polynomials can be constructed by Gram-Schmidt orthonalization of the monomials $1, x, x^2, ...$. Our goal is to find nodes and weights. This is a tedious work . This difficulty is overcome by using the idea of "three-term recurrence". We will develop this theory a little in this section.

Theorem 1. *The abscissas of the N -point Gaussian quadrature formula are precisely the roots of the orthogonal polynomial [1] for the same interval and weighting function.*
Let p_n *be a nontrivial polynomial of degree* n *such that*

$$\int_a^b w(x)x^k p_n(x)dx = 0, \; for \; k = 0, 1, 2, ...n - 1. \tag{2.18}$$

If we pick the nodes to be the zeros of p_n , then there exist weights w_i which make the computed integral exact for all polynomials of degree $2n - 1$ or less. Furthermore, all these nodes will lie in the open interval (a, b).
For more understanding we present another theorem.

Theorem 2. ***Construction of Gaussian quadrature***
For $N = 2n - 1$*, there exists [1] a set of Gaussian points* $x_i^{(n)}$ *and weights* $w_i^{(n)}$*, such that*

$$\int_{-1}^{1} x^k dx = \sum_{i=1}^{n} w_i^{(n)} (x_i^{(n)})^k$$

holds for all $k = 0, 1, 2, ..., N$

Proof Let the Gaussian points $x_i^{(n)}$ be the roots of the Legendre family of orthogonal polynomials and $P(x)$ be an arbitrary polynomial of degree $\leqslant 2n-1$. We can write

$$P(x) = P_n(x)Q(x) + R(x) \tag{2.19}$$

where both Q and R are polynomials of degree $\leqslant n-1$ and P_n is defined as

$$P_n(x) = \prod_{i=1}^{n}(x - x_i^n) \tag{2.20}$$

For $P_n(x_i) = 0$ this implies that $P(x_i) = R(x_i)$.
Let $Q(x)$ be any polynomial of degree $\leqslant n-1$, so that the product $P_n(x)Q(x)$ has degree $\leqslant 2n-1$. Then

$$\int_{-1}^{1} P(x)dx = \int_{-1}^{1}(P_n(x)Q(x) + R(x))dx \tag{2.21}$$

$$= \int_{-1}^{1} P_n(x)Q(x)dx + \int_{-1}^{1} R(x))dx \tag{2.22}$$

The orthogonality property of P_n implies that first integral is zero. Hence

$$\int_{-1}^{1} P(x)dx = \int_{-1}^{1} \sum_{i=1}^{n} R(x_i)^{(n)} L_i(x))dx \tag{2.23}$$

$$= \sum_{i=1}^{n} R(x_i)^{(n)} \int_{-1}^{1} L_i(x)dx = \sum_{i=1}^{n} R(x_i)^{(n)} w_i^n \tag{2.24}$$

$$= \sum_{i=1}^{n} P(x_i^{(n)}) w_i^{(n)} \tag{2.25}$$

This proves that the quadrature rule is exact for an arbitrary polynomial of degree $\leqslant 2n-1$.

2.3 Comparison of Newton-Cotes rules with Gaussian quadrature rule

In this section we compare the accuracy of Newton-Cotes method and Gaussian quadrature for the model problem.

Example 1. *Evaluate the integral*

$$I = \int_0^1 sin(x)dx \tag{2.26}$$

by using the Trapezoidal rule , Simpson rule and Gaussian quadrature formula.

Exact value (analytical solution) is calculated by simple integration rules which is $4.5970e-001$.
We will present a table about the comparisons of errors due to the Trapezoidal rule, Simpson's rule and Gaussian quadrature rule.

Table 1.1: Comparison of Errors

N-Values	Trapezoidal rule	Simpson rule	Gaussian quadrature rule
2	9.6172e-003	-1.6450e-004	6.4180e-003
4	2.3968e-003	-1.0051e-005	1.5772e-005
6	1.0646e-003	-1.9771e-006	5.2273e-010
8	5.9872e-004	-6.2467e-007	4.8849e-015
10	3.8315e-004	-2.5569e-007	2.2204e-016
20	9.5774e-005	-1.5966e-008	4.4409e-016
30	4.2565e-005	-3.1534e-009	6.6613e-016
40	2.3943e-005	-3.1534e-009	1.3323e-015
50	1.5323e-005	-4.0864e-010	1.5754e-011
100	3.8308e-006	-2.5539e-011	1.3323e-015

From the table, we see that the error of 4 points Gaussian quadrature rule is almost equal to 50 points of the Trapezoidal rule. Similarly absolute error due to using 6 points Gaussian quadrature rule is almost equal to 50 points Simpson's rule. From these results it is clear that error of Gaussian quadrature rule is about 10 times less than Newton-Cotes formulas. Further when $n \geqslant 10$ the error due to Gaussian quadrature becomes negligibly small.

The main benefit of Gaussian quadrature is that of its very high-order accuracy with very few number of intervals, especially when we are using points less than 10. This accuracy is further enhanced by increasing number of points. Due to these reasons we prefer Gaussian quadrature rule over other methods.

2.4 Connection between Gaussian quadrature rule and Eigenvalues problem

We know that the zeros of orthogonal polynomials are the eigenvalues of particular tridiagonal symmetric matrices.
It is known that monic orthogonal polynomials satisfy a three- term recurrence relation [2] of the form

$$P_n(x) + (A_n - x)P_{n-1}(x) + B_n P_{n-2}(x) = 0 \qquad n = 2, 3, ... \tag{2.27}$$

where $B_n > 0$ and initial conditions are $P_0(x) = 1\,, P_1(x) = x - A_1$

The method generates a sequence of tridiagonal matrices T_n.
Let

$$T_n = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{2.28}$$

is a tridiagonal matrix, and let p_n be the characteristic polynomial of T_n.
Also let

$$P_n(x) = det(xI - T_n) \tag{2.29}$$

Expanding $det(xI - T_n)$ along the last row, we have

$$P_n(x) = det(xI - T_n) \tag{2.30}$$

$$\begin{aligned} P_n(x) &= (x - \alpha_n)\, det(xI - T_{n-1}) - B_{n-1}^2\, det(xI - T_{n-1}) \\ &= (x - \alpha_n)\, P_{n-1}(x) - B_n^2\, P_{n-2}(x) \\ P_n(x) &+ (\alpha_n - x)\, P_{n-1}(x) + B_n^2\, P_{n-2}(x) = 0 \end{aligned} \tag{2.31}$$

Thus p_n and P_n satisfy the same three-term recurrence .
Further, one can easily verify that $p_1(x) = P_1(x), \quad p_2(x) = P_2(x)$.
Thus

$$p_i(x) = P_i(x) \qquad i = 1, 2, ...$$

There is also a relation between the nodes and weights of Gauss quadrature formulas and the eigenvalues and eigenvectors of T_n. We further explain this relation by a theorem, but we omit the proof.

Theorem 3. *Let T_n be the $n \times n$ Jacobi matrix with entries $\beta_1, \beta_2, ..., \beta_{n-1}$. Let $T_n = VDV^T$ be an orthogonal diagonalization of T_n with $V = [v_1|...|v_n]$ and $D = diag(\lambda_1, ..., \lambda_n)$. Then the nodes and weights of the Gauss-Legendre quadrature formulas are given by*

$$x_j = \lambda_j\,, \quad w_j = 2(v_j)_1^2\,, \qquad j = 1, 2, ..., n.$$

where x_j and w_j are required nodes and weights and λ_j and v_j are eigenvalues and eigenvectors of T_n respectively.

The connection between the eigenvectors and the weights is considerably more complicated than the connection between the eigenvalues and nodes We have transform the problem of finding the nodes and weights for Gaussian quadrature to one of finding eigenvalues and eigenvectors of a symmetric tridiagonal matrix. There is considerable theory and many numerical methods for this problem.

Chapter 3

The Hermitian Eigenvalue Problems

Previous chapter ended up with conclusion that there is connection between Gaussian quadrature nodes and weights with eigenvalue problems. Hence we have a problem in Matrix Analysis. We start by giving some definitions and background theory about matrices their special forms which would be used later in this work .

3.1 Introduction to Matrix Analysis

Matrix analysis study can be traced centuries back. The work of the British mathematician Arthur Cayley (1821-1895) was pioneer of modern matrix analysis [4]. He singled out the matrix as a separate entity, distinct from the notion of a determinant. He defined various algebraic operations between matrices. In 1855, Cayley introduced his basic ideas through his papers. He laid the foundation for the modern theory and is generally credited for the birth of the subjects of matrix analysis and linear algebra. In 1850s, due to collaboration between Cayley and Sylvester matrix theory was born and nurtured. They shared many mathematical interests despite difference in their mathematical approaches. Morris Kline says in his book "Mathematical Thought from Ancient to Modern Times" that "the subject of matrix theory was well developed before it was created." It is fact that immediately after the publication of Cayley's " A Memoir on the theory of Matrices", the subject of matrix theory and linear algebra expanded and quickly evolved into discipline that now occupies a central position in applied mathematics.

Before we start the computation of eigenvalues and eigenvectors to establish relation with nodes and weights, here we present some basic understanding about matrices, their special forms and operations to convert them into diagonal form.

3.2 Some Special Matrices

A *matrix* is an $m \times n$ array of scalars from a field F. If $m = n$, the matrix is said to be square. The set of all m-by-n matrices over F is denoted by $M_{m,n}(F)$, and $M_{n,n}(F)$ is abbreviated to $M_n(F)$.
Let $A \in M_{m,n}(F)$, such that $A = (a_{ij})$

- Rank A is the largest number of columns of A which constitute a linearly independent set. A very useful result is that "rank A^T=rank A". Equivalently rank of a matrix can be expressed in terms of rows. It is also worth noting that "row rank = column rank".
- If $|A| = 0$ then A is singular. Otherwise it is non-singular.
- For a square non-singular matrix A, there exists a unique inverse matrix A^{-1} such that $AA^{-1} = A^{-1}A = I$
- $A^T \in M_{n,m}(F)$ is called transpose of matrix A and denoted by $A^T = (a_{ji})$.
- A is called *symmetric* if $A^T = A$,i.e., $a_{ij} = a_{ji}$ for all i, j.
  Symmetric matrices have the following properties:
  * Eigenvalues are real.
  * Eigenvectors are orthogonal.
  * They are always diagonalizable.
- A is called *skew-symmetric* if $A^T = -A$.
- $A \in M_n(C)$ is *Hermitian* if $A^* = A$, i.e., $a_{ij} = \bar{a}_{ji}$ for all i, j.
- $A \in M_n(C)$ is *skew-Hermitian* if $A^* = -A$, i.e., $a_{ij} = -\bar{a}_{ji}$ for all i, j
- Let $A \in M_n$ then A can be written as $A = H + iK$ with H and K hermitian in exactly one way:

$$H = (A + A^*)/2; \quad K = (A - A^*)/(2i).$$

- A matrix $U \in M_n$ is said to be *unitary* if $U^*U = UU^* = I$. Where as $U \in M_n(R)$ then U is said to be *real orthogonal*.
- A is diagonal if $a_{ij} = 0$ for $i \neq j$.
- The identity matrix is special case of diagonal matrix with all diagonal entries equal to 1.
- A *permutation matrix* denoted by P is obtained from the identity matrix by row or column permutation.
- Any square matrix A is upper triangular if $a_{ij} = 0$ for $i > j$.
- Any square matrix A is lower triangular if $a_{ij} = 0$ for $i < j$.
- If all diagonal elements of A are equal to 1 then A is called a unit upper or lower triangular.
- Two matrices A and B of order $n \times n$ are said to be *similar* whenever there exists a nonsingular matrix P such that $P^{-1}AP = B$. The product $P^{-1}AP$ is called a *similarity transformation* on A. Here are some important rules regarding similar matrices.
- A square matrix A is *diagonalizable* whenever A is similar to a diagonal matrix. Any $A_{n\times n}$ matrix is diagonalizable if and only if A possesses a complete set of eigenvectors. Moreover $P^{-1}AP = diag(\lambda_1, \lambda_2, ..., \lambda_n)$ if and only if the columns of P constitute a complete set of eigenvectors and the λ_j's are the associated eigenvalues.

Definition 1. ***Diagonally Dominant Matrices (DDM)*** *An $n \times n$ matrix A is termed diagonally dominant if*

$$|A_{ii}| \geqslant \sum_{j \neq i} |A_{ij}|, \ i = 1, , n.$$

Example 2.

Let

$$A = \begin{pmatrix} 3 & -2 & 1 \\ -3 & 6 & 2 \\ 4 & 1 & 3 \end{pmatrix}$$

is not diagonally dominant.
Where as

$$B = \begin{pmatrix} -5 & -2 & 1 \\ -3 & 6 & 1 \\ 4 & 1 & 7 \end{pmatrix}$$

is diagonally dominant matrix

3.3 Norms of Vectors and Matrices

Norms are very useful and important in error analysis of vectors and matrices. Norms are means of measuring the size of a vector or matrix. Here we shall present the formal definitions of various norms. In future work we will use norms to analyse the accuracy of the numerical methods we have used to compute eigenvalue and eigen vectors.

3.3.1 Vector Norms

Norms are a means of obtaining a scalar measure of a vector or matrix. Let V be a vector space over a field $F(R\,or\,C)$. A function $\|.\| : V \rightarrow R$ is a *vector norm*if for all $x, y \in V$,

- $\| x \| \geqslant 0$
- $\| x \| = 0$ if and only if x=0
- $\| cx \| = \mid c \mid \| x \|$ for all scalar $c \in F$
- $\| x + y \| \leqslant \| x \| + \| y \|$

The Holder p-norm is defined as

$$\|x\|_p = \left(\sum_{i=1}^{n} \mid x_i \mid^p \right)^{1/p}, \; p \geqslant 1.$$

By using this definition we get three vector norms :

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

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

(which is Euclidean Norm).

$$\|x\|_\infty = max_{1 \leqslant i \leqslant n} \mid x_i \mid$$

(which is the infinity norm).

Example 3. *For more explanation and to establish relation between various vector noms we present a example.*
Let

$$x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

$$\|x\|_1 = \sum_{i=1}^{3} \mid x_i \mid = 3$$

$$\|x\|_2 = \left(\sum_{i=1}^{3} \mid x_i \mid^2 \right)^{1/2} = (3)^{1/2} = \sqrt{3}$$

$$\|x\|_\infty = max_{1 \leqslant i \leqslant 3} \mid x_i \mid = 1$$

One can easily show that for any vector x,

$$\|x\|_1 \geqslant \|x\|_2 \geqslant \|x\|_\infty$$

3.3.2 Matrix Norms

We can define matrix norms as functions $\|.\| : C^{m \times n} \rightarrow R$ and satisfy the matrix norm axioms:

- $\| A \| \geqslant 0$ for all $A \in C^{n \times n}$, and $\| A \| = 0$ if and only if $A = 0$.
- $\| \kappa A \| = \mid \kappa \mid \| A \|$ for all scalar $\kappa \in C, A \in C^{n \times n}$.
- $\| A + B \| \leqslant \| A \| + \| B \|$ *for all* $A, B \in C^{n \times n}$.
- $\| AB \| \leqslant \| A \| \| B \|$ whenever AB is defined.

The matrix infinity norm,$\|A\|_\infty$,can be shown to be equivalent to the following computation:

$$\|A\|_\infty = max_{1 \leqslant i \leqslant n} \sum_{j=1}^{n} \mid a_{ij} \mid$$

Thus $\|A\|_\infty$ is defined as the maximum row sum.
Here is another useful subordinate matrix norms :

$$\|A\|_1 = max_{1 \leqslant j \leqslant n} \sum_{i=1}^{m} \mid a_{ij} \mid$$

(maximum column sum).

The Matrix 2-norm can be represented as follows :

$$\|A\|_2 = \sqrt{\rho(A^t A)} = \sigma_{max}(A)$$

where $\rho(A)$ is the largest absolute eigenvalue of the matrix A.
For a square matrix A, the number

$$\rho(A) = max_{\lambda \in \sigma(A)} |\lambda|$$

is called the *spectral radius* of A.

The *Frobenius norms* are very common matrix norms. Let $A \in C^{m \times n}$ then the Frobenius norms are defined as

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} \mid a_{ij} \mid^2 \right)^{1/2}$$

Example 4. *Let*

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

Find various matrix norms.

Solution: We can prove easily that

$$\|A\|_1 = 8$$

$$\|A\|_2 = 5.8107$$

$$\|A\|_\infty = 7$$

3.4 Matrix Factorizations

Representing a matrix as the product of two or more matrices with special properties, makes the solution of linear algebra problems straightforward.

3.4.1 LU factorization

If $A = LU$ where L is lower triangular and U is upper triangular then $A = LU$ is called a LU factorization of A. Not every matrix has an LU

factorization.For this let us take a example of symmetric and nonsingular matrix. Suppose

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = LU = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}$$

By comparison of corresponding entries, we have

$$l_{11}u_{11} = 0, \qquad l_{11}u_{12} = 1 \qquad l_{21}u_{11} = 1 \qquad l_{21}u_{12} + l_{22}u_{22} = 0$$

But $l_{11}u_{11} = 0$ implies that either $l_{11} = 0$ or $u_{11} = 0$. Which contradict $l_{11}u_{12} = 1$ and $l_{21}u_{11} = 1$.
Note that such a factorization is possible when A is symmetric and positive definite(we shall discuss it in detail under Choleski factorization)

Definition 2. *Let A be a nonsingular matrix, then a decomposition of A as a product of a unit lower triangular matrix L, a diagonal matrix D and a unit upper triangular matrix U such that :*

$$A = LDU$$

is called an LDU decomposition of A.

3.4.2 Schur form

Let $A \in M_n$ have eigenvalues $\lambda_1, \lambda_2 \ldots, \lambda_n$, (taken in any order) then there is a unitary matrix U and an upper triangular matrix T such that $A = U^*TU$ and $t_{ii} = \lambda_i$.
In other words every square matrix is unitarily similar to an upper triangular matrix. That is for each $A_{n\times n}$, there exist a unitary matrix U (not unique) and an upper-triangular matrix T, not unique, such that $U^*AU = T$ and the diagonal entries of T are the eigenvalues of A.
Real Schur form Let $A \in M_n(R)$. The Schur form above will not be real if A has complex eigenvalues. However, there is a real orthogonal matrix Q and a block upper triangular matrix T, with 1×1 and 2×2 diagonal blocks such that $A = Q^TTQ$.

3.4.3 Schur complement

Let us compute a block LU factorization of block matrix:

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

where A is $n \times n$ and A_{11} and A_{22} are square but A_{12} and A_{21} are not necessarily square:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

The matrix $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is called the Schur Complement of A_{11} in A.

3.4.4 QR factorization

Let $A \in M_n(R)$. If $A = QR$ where Q is orthogonal and R is upper triangular then $A = QR$ is called a QR factorization of A. We can prove that $A = QR$ factorization is much better than the $A = LU$ factorization when we use it to solve the system $AX = B$.
The QR decomposition rewrites the given matrix A into two special matrices. The system can then be solved in the form of

$$AX = B$$

$$QRX = B$$

$$(Q^TQ)RX = Q^TB$$

$$RX = Q^TB$$

Since Q is orthogonal, $\|Q\| = 1$ and $\|Q^{-1}\| = \|Q^T\| = 1$

How to find QR factorization?

The QR factorization algorithm is one of the most important and widely used for matrix computation. There are several forms of QR factorization which are used to compute the eigenvalues of real symmetric nonsymmetric matrices.

Many people have contributed to the development of various QR algorithms. The work of Wilkinson presented in his book, *The Algebraic Eigenvalue Problem* is remarkable.

Here we present the basic computational outline for QR.

1. Take A and find orthogonal matrix Q_1 such that $A_1 = Q_1 A$ has some zero entries in the right place.

2. Take A_1 and find orthogonal matrix Q_2 such that $A_2 = Q_2 A_1$, has some zero entries as A and some more.

Eventually we have

$$Q_k....Q_2 Q_1 A = R$$
$$A = Q_1^T....Q_k^T R$$
$$A = (\underbrace{Q_k, ..., Q_1})^T R$$
$$A = QR$$

Using different methods for putting zeros yields different QR factorization algorithms.

The most commonly used methods for $QR - factorization$ are:

1. Householder transformation

2. Givens rotation

3.4.5 Householder Transformation

Householder reflection is a matrix of form $H = I - 2uu^T$, where u is unitary vector,i.e., $\|u\| = 1$. It is the generalization of the reflection onto the plane with normal vector $u \in \mathtt{R}$. It is easy to check that H is symmetric,that is $H = H^T$. Now we prove that it is also symmetric orthogonal matrix,i.e., $HH^T = H^T H = I$.
Let

$$HH^T = (I - 2uu^T)(I - 2uu^T)$$
$$= I - 2uu^T - 2uu^T + 4uu^T uu^T$$
$$= I - 4uu^T + 4u(uu^T)u = I$$

Given a vector $x \neq 0$, it is easy to find a Householder reflection $H = I - 2uu^T$ to zero out all but the first entry of x.

$$Hx = x - 2(u^T x)u = \begin{pmatrix} c \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = ce_1 \tag{3.1}$$

Since H is orthogonal

$$\|Hx\| = |ce_1|$$
$$\Rightarrow |c| = \|x\|$$

From relation(3.1), we have

$$(x - ce_1) = 2(uu^T)$$

$$u = \frac{1}{2u^T u}(x - ce_1)$$

Finally we get

$$H(x) = I - 2uu^T$$
$$= I - \frac{2\tilde{u}\tilde{u^T}}{\|\tilde{u}\|^2}$$

where

$$\tilde{u} = \frac{u}{\|u\|}$$

QR-factorization of A by using Householder reflection

The working procedure of Householder reflection is similar to the Gauss elimination in the $LU - factorization$. Let $A \in M_n$. We convert A into upper triangular form R by multiplying it from the left by appropriately chosen Householder matrix. Assume that all columns of A are not fully zero. Select a_1, the first column of A. In the first step we eliminate all entries of column below $(1,1)$ position. We can do it by one single Householder matrix, namely by $H(a_1)$. the result is

$$A_1 = H(a_1)A = \begin{pmatrix} \oplus & \oplus & \oplus & \dots & \oplus \\ 0 & \underline{\oplus} & \oplus & \dots & \oplus \\ 0 & \underline{\oplus} & \oplus & \dots & \oplus \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \underline{\oplus} & \oplus & \dots & \oplus \end{pmatrix}$$

where $\oplus$ denotes a possibly non-zero entry. Also for simplicity we shall use $H_1 = H(a_1)$. Next, we look at the second column of the matrix $A_1 = H(a_1)A$. First row of matrix A_1 will remain unchanged so we cut off the first entry of the second column,i.e., we are left with vector $\tilde{a_2}$ of size $(n-1)$ which consists of the under lined elements. We shall convert it into upper triangular by reducing all entries to zero below $\tilde{a_2}$.

After this we use a Household reflection $H(\tilde{a_2})$ in the space of "cut off" vectors. Then the new

$$H(a_2) = H_2 = \left(\begin{array}{c|ccc} 1 & 0 & \dots & 0 \\ \hline 0 & & & \\ \vdots & & H(\tilde{a_2}) & \\ 0 & & & \end{array}\right)$$

In result

$$A_2 = H_2 A_1 = \begin{pmatrix} \oplus & \oplus & \oplus & \dots & \oplus \\ 0 & \oplus & \oplus & \dots & \oplus \\ 0 & 0 & \underline{\oplus} & \dots & \oplus \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \underline{\oplus} & \dots & \oplus \end{pmatrix}$$

Repeat all steps on similar lines for third row. Now consider rest columns of A_2 which has non-zero elements below the third row. Again by cutting off the top two entries of this vector $\tilde{a_3}$, we are left with $(n-2)$ elements which are under lined in A_2.
Now find Householder reflection $H(\tilde{a_3})$ and then calculate H_3 by multiplying with A_2 from the left

$$H(a_3) = H_3 = \left(\begin{array}{cc|ccc} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \hline 0 & 0 & & & \\ \vdots & \vdots & H(\tilde{a_3}) & & \\ 0 & 0 & & & \end{array}\right)$$

Hence in result

$$A_3 = H_3 A_2 = \begin{pmatrix} \oplus & \oplus & \oplus & \oplus & \dots & \oplus \\ 0 & \oplus & \oplus & \oplus & \dots & \oplus \\ 0 & 0 & \oplus & \oplus & \dots & \oplus \\ 0 & 0 & 0 & \underline{\oplus} & \dots & \oplus \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \underline{\oplus} & \dots & \oplus \end{pmatrix}$$

In $(m-1)$ steps we get R which is a upper triangular. In result we have

$$H_{m-1} H_{m-2} ... H_2 H_1 A = R$$

$\Rightarrow A = QR$
where $Q = H_1 H_2 ... H_{m-2} H_{m-1}$.

Example 5. *Let*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

find the QR factorization with Householder reflection.

Solution Select first column vector

$$a_1 = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

$$\|a_1\| = 2.4495$$

$$b_1 = \begin{pmatrix} 2.4495 \\ 0 \\ 0 \end{pmatrix}$$

Let $u_1 =$ The Householder vector $= (a_1 - b_1)$

$$u_1 = \begin{pmatrix} -1.4495 \\ 2 \\ 1 \end{pmatrix}$$

$\|u_1\| = 2.6648$

$$\tilde{u}_1 = \frac{u_1}{\|u_1\|} = \begin{pmatrix} -0.5439 \\ 0.7505 \\ 0.3753 \end{pmatrix}$$

$$H_1 = I - 2\tilde{u}_1\tilde{u}_1^T = \begin{pmatrix} 0.4082 & 0.8165 & 0.4082 \\ 0.8165 & -0.1266 & -0.5633 \\ 0.4082 & -0.5633 & 0.7184 \end{pmatrix}$$

In result

$$A_1 = H_1 A = \begin{pmatrix} 2.4495 & 3.2660 & 2.0412 \\ 0.0000 & 1.2532 & 1.3229 \\ 0.0000 & -0.8734 & 2.6614 \end{pmatrix}$$

The next cutoff vector is

$$a_2 = \begin{pmatrix} 1.2532 \\ -0.8734 \end{pmatrix}$$

$\|a_2\| = 1.5275$

$$b_2 = \begin{pmatrix} 1.5275 \\ 0 \end{pmatrix}$$

$$u_2 = (a_2 - b_2)$$

$$u_2 = \begin{pmatrix} -0.2743 \\ -0.8734 \end{pmatrix}$$

$$\tilde{u}_2 = \frac{u_2}{\|u_2\|} = \begin{pmatrix} -0.2997 \\ -0.9540 \end{pmatrix}$$

$$H(a_2) = I - 2\tilde{u}_2\tilde{u}_2^T = \begin{pmatrix} 0.8204 & -0.5718 \\ -0.5718 & -0.8204 \end{pmatrix}$$

$$H_2 = \begin{pmatrix} 1.0000 & 0 & 0 \\ 0 & 0.8204 & -0.5718 \\ 0 & -0.5718 & -0.8204 \end{pmatrix}$$

$$H_2H_1A = \begin{pmatrix} 2.4495 & 3.2660 & 2.0412 \\ 0.0000 & 1.5276 & -0.4365 \\ 0.0000 & -0.0000 & -2.9399 \end{pmatrix}$$

This is R factor of A.
And $Q = H_1 * H_2 * ....H_{m-1}$

3.4.6 Givens Rotations

Let A be an (m, n) matrix with $m \geqslant n$. An orthogonal QR decomposition consists in determining an (m, m) orthogonal matrix Q s.t

$$Q^T A = R$$

where R is upper triangular of order (n, n).In this way one has only to solve the triangular system $Rx = Py$, where P consists of the first n rows of Q . Householder transformations clear whole column except for first element of a vector. If one wants to clear out one element at a time then Givens rotations are the best choice. The basic idea is that by an appropriately chosen θ we can always rotate a given vector into a vector whose second entry is zero:

$$\begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2} \\ 0 \end{pmatrix}$$

where

$$c = cos\theta = \frac{x}{\sqrt{x^2 + y^2}}$$

and

$$s = sin\theta = \frac{-y}{\sqrt{x^2 + y^2}}$$

We can write these rotations as follow,

$$G(i,j,\theta) = \begin{pmatrix} 1 & & & & & \\ & c & & & -s & \\ & & 1 & & & \\ & & & 1 & & \\ & s & & & c & \\ & & & & & 1 \end{pmatrix}$$

where the trigonometric functions are in the i-th and j-th columns and all entries are zero.
Let $A \in M_n$. Select first column and take the first non-zero element in the first column below the diagonal. Chose θ appropriately for any $k-th$ row and can be eliminated by left multiplication by $R(1,k,\theta)$. All columns of A are converted into zero below the diagonal. We notice that the zeros in first column remain zero after these multiplications as $R(i,j,\theta)$ acts only on the i-th and j-th columns and rows respectively and does not touch the first column. After series of repeated N-steps we get an upper triangular matrix as a result of many multiplications by Givens matrices,i.e.

$$G_N G_{N-1} .... G_2 G_1 A = R$$

For any square matrix A of order $n \times n$, the maximum number of Givens rotations are

$$N = (n-1)(n-2)/2$$

The $QR-factorization$ of A is now obtained by inverting all Givens matrices to transposes,

$$Q = G_1^T G_2^T .... G_N^T$$
$$\Rightarrow A = QR$$

Working procedure of Givens rotations: Consider any matrix of order (4×3) .

$$A = \begin{pmatrix} \oplus & \oplus & \oplus \\ \underline{\oplus} & \oplus & \oplus \\ \oplus & \oplus & \oplus \\ \oplus & \oplus & \oplus \end{pmatrix}$$

Here the underlined position is to be eliminated. First we calculate values of $cos\theta$ and $sin\theta$ as follow

$$c = cos\theta = \frac{a_{11}}{\sqrt{a_{11}^2 + a_{21}^2}}$$

$$s = sin\theta = \frac{-a_{21}}{\sqrt{a_{11}^2 + a_{21}^2}}$$

For this purpose $R(1, 2, \theta)$ matrix is chosen like

$$G_1 = \begin{pmatrix} c & -s & & \\ s & c & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

This gives

$$A_1 = G_1 A = \begin{pmatrix} c & -s & & \\ s & c & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} \oplus & \oplus & \oplus \\ \underline{\oplus} & \oplus & \oplus \\ \oplus & \oplus & \oplus \\ \oplus & \oplus & \oplus \end{pmatrix} = \begin{pmatrix} \oplus & \oplus & \oplus \\ 0 & \oplus & \oplus \\ \underline{\oplus} & \oplus & \oplus \\ \oplus & \oplus & \oplus \end{pmatrix}$$

We repeat same procedure for matrices $G_2 = R(1, 3, \theta)$ and $G_3 = R(1, 4, \theta)$ to eliminate remaining entries of matrix A for first column. Similarly select second column and repeat step by step all procedure already done. Hence for third column. Ultimately we get a upper triangular matrix.

Example 6. *Find the QR factorization of*

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

by using Givens rotations.

Solution: Select first column vector

$$c = \frac{1}{\sqrt{5}} = 0.4472$$

$$s = \frac{-2}{\sqrt{5}} = -0.8944$$

$$G_1 = \begin{pmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$G_1 = \begin{pmatrix} 0.4472 & 0.8944 & 0 \\ -0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A_1 = G_1 A = \begin{pmatrix} 2.2361 & 1.3416 & 0.8944 \\ 0 & -0.4472 & 0.4472 \\ 0.8944 & 2.0000 & 1.0000 \end{pmatrix}$$

$$c_1 = \frac{-0.4472}{\sqrt{0.4472^2 + 4}} = -0.2182$$

$$s_1 = \frac{-2}{\sqrt{0.4472^2 + 4}} = -0.9759$$

$$G_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_1 & s_1 \\ 0 & -s_1 & c_1 \end{pmatrix}$$

$$G_2 = \begin{pmatrix} 1.0000 & 0 & 0 \\ 0 & -0.2182 & 0.9759 \\ 0 & -0.9759 & -0.2182 \end{pmatrix}$$

$$A_2 = G_2 A_1 = \begin{pmatrix} 2.2361 & 1.3416 & 0.8944 \\ 0 & 2.0494 & 0.8783 \\ 0.9844 & 0.0000 & -0.6546 \end{pmatrix}$$

$$Q = G_1 * G_2 \begin{pmatrix} 0.4472 & o.1952 & 0.8729 \\ 0.8944 & -0.0976 & -0.4364 \\ 0 & 0.9759 & -0.2182 \end{pmatrix}$$

This rotation gives one zero in $(3,1)$ position. By repeating the same procedure, an upper triangular matrix Q is obtained.

3.5 The Standard Eigenvalue Problem

If $A \in C, x \neq 0 \in C^n$ and $\lambda \in C$, then we call the problem

$$Ax = \lambda\, x \tag{3.2}$$

the standard eigenvalue problem. The scalar λ is an eigenvalue and x is a corresponding eigenvector. For a certain matrix, each one of its eigenvectors is associated with a particular (though not necessarily unique) eigenvalue. If we write (3.2) as

$$(A - \lambda I)x = 0 \tag{3.3}$$

Then we see that x is in the null space of the matrix $(A-\lambda I)$, which therefore has to be nonempty. Equivalently

$$det(A - \lambda I) = 0$$

which is a n degree polynomial called as the characteristic polynomial.

Definition 3. *The set of all $\lambda \in C$ that are eigenvalues of $A \in M_n$ is called the spectrum of A and is denoted by $\sigma(A)$. The spectrum radius of A is the nonnegative real number* $\rho(A) = max\{|\lambda| : \lambda \in \sigma(A)\}$.
This is just the radius of the smallest disc centered at the origin in the complex plane that includes all the eigenvalues of A.

3.5.1 Eigenvalue and polynomials

The problem of computing the eigenvalues of a $n \times n$ matrix is equivalent to that of finding the zeros of a complex polynomial of degree n. To see this note that the eigenvalues of $A \in M_n$ are the zeros of its characteristic polynomial: $p_A(t) = \det(tI - A)$. The zeros of a monic polynomial $p(t) = t^n + a_{n-1}t^{n-1} + \cdots + a_1 t + a_0$ are the eigenvalues of its companion matrix:

$$\begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ a_{n-1} & \cdots & \cdots & a_1 & a_0 \end{pmatrix}$$

It is a fact that there is no finite algorithm to compute the zeros of a polynomial of degree greater than 5, and consequently, there can be no finite algorithm to compute the eigenvalues of a general matrix of order 5 or greater. Consequently, any factorization that exposes the eigenvalues of a matrix cannot in general be computed in a finite number of steps. On the other hand, the zeros of a polynomial are continuous functions of the coefficients, and hence the eigenvalues of a matrix are continuous functions of the entries of the matrix.
To illustrate the concept of eigenvalues and eigenvectors let us see following **examples**. Let

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

Here $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is the eigenvector and 0 is its associated eigenvalue.

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

In the second example 3 is eigenvalue and $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is eigenvector.

Problem:*Diagonalize the matrix*

$$A = \begin{pmatrix} 2 & 5 \\ -1 & -4 \end{pmatrix}$$

SOLUTION: We have to find eigenvalues and eigenvectors. Steps follows as:
Step 1: Find the characteristic polynomial:

$$p(\lambda) = det \begin{pmatrix} 2-\lambda & 5 \\ -1 & -4-\lambda \end{pmatrix} = (2-\lambda)(-4-\lambda) - 5(-1) = \lambda^2 + 2\lambda - 3$$

Step 2: Find the roots of $p(\lambda)$.
$\lambda_1 = 1$, $\lambda_2 = -3$
Step 3: Find the eigenvectors. Must be done separately for each eigenvalue.
Eigenvector for $\lambda_1 = 1$ and $\lambda_2 = -3$ are

$$\begin{pmatrix} -5 \\ 1 \end{pmatrix} and \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

respectively.

3.6 Numerical Methods for finding Eigenvalue Problems

A number of numerical methods are commonly used for finding eigenvalues of symmetric matrices.

3.6.1 Power method for simple eigenvalues

This is very simple and the oldest method for finding eigenvalue problems. Take any square matrix A. Select any vector x_0 and start multiplying successively with A. In this way we can easily find eigenvalues of any square matrix.

The power method is the iterative method of the form

$$x_{k+1} = \frac{Ax_k}{s_{k+1}}$$

where s_k is a scaling factor, and x_0 is the chosen starting vector. If we take $s_k = 1$ for all k then we get

$$x_{k+1} = Ax_k = AAx_{k-1} = \cdots = A^{k+1}x_0.$$

From that we see why it is called the power method. In general, the iteration with $s_k \equiv 1$ will not converge. Either $\|x_k\|$ will converge to 0, or will go to infinity. To ensure convergence we must ensure that x_k neither goes to 0 nor blows up. A useful choice is to take s_k to be the largest entry of Ax_k in absolute value. This is what we will assume from now on. Then provided $|\lambda_1| > |\lambda_2|$ the scaling factors s_k converge to an eigenvalue of A and $|s_k| \to |\lambda_1|$, and the vectors x_k converge to a corresponding eigenvector. To see these results let us assume that A is diagonalizable, that is, there are n linearly independent vectors $v_1, \ldots, v_n$ such that $Av_i = \lambda_i v_i$. These vectors form a basis, so the starting vector x_0 can be written as a linear combination of the v_i's,

$$x_0 = \sum_{i=1}^{n} \alpha_i^{(0)} v_i.$$

Now multiplying by A we get

$$Ax_0 = \sum_{i=1}^{n} \alpha_i^{(0)} Av_i = \sum_{i=1}^{n} \alpha_i^{(0)} \lambda_i v_i,$$

and so

$$x_1 = \frac{Ax_0}{s_1} = \sum_{i=1}^{n} \frac{\alpha_i^{(0)} \lambda_i}{s_1} v_i = \sum_{i=1}^{n} \alpha_i^{(1)} v_i$$

where $\alpha_i^{(1)} = \alpha_i^{(0)} \lambda_i / s_1$.

In the same way

$$x_k = \sum_{i=1}^{n} \alpha_i^{(k)} v_k$$

where

$$\alpha_i^{(k)} = \alpha_i^{(0)} \lambda_i^k / (s_1 \cdots s_k).$$

Notice that the ratio

$$\frac{\alpha_i^{(k)}}{\alpha_1^{(k)}} = \left(\frac{\lambda_i}{\lambda_1} \right)^k$$

is independent of the choice of scaling, and converges to 0. If we choose the scaling to ensure that $\alpha_1^{(k)}$ converges then the $\alpha_i^{(k)}$ will go to zero for $i > 1$

and so, the sequence of vectors x_i will converge to a multiple of v_1. There are many scalings that achieve this, and the one we have chosen is one such. The rate of convergence is $\frac{|\lambda_1|}{|\lambda_2|}$– smaller the ratio faster the convergence. An important fact is that we don't need the matrix explicitly – we just need to know how to compute Ax. An important application is the estimation of $\|A^{-1}\|_2$. Suppose that we are solving $Ax = b$, using say an LU factorization. We then want to estimate $\|A^{-1}\|_2$ to get error bounds. Given L and U we could use them to compute A^{-1}, but this would require an additional computations. Instead, we can use the fact that

$$\|A^{-1}\|_2^2 = \rho(A^{-1}A^{-T}).$$

That is, we can compute the 2-norm by computing the spectral radius. To employ the power method for this we need only be able to compute products for the form

$$A^{-1}A^{-T}x = U^{-1}L^{-1}L^{-T}U^{-T}x = U\backslash(L\backslash(L^T\backslash(U^T\backslash x)))$$

Where $B\backslash y$ means solve the system $Bz = y$ for z. To compute $A^{-1}A^{-T}x$ for a given vector x we need only solve 4 triangular systems – $4n^2$ flops/. A few iterations of the power method will generally give an estimate of $\|A^{-1}\|$ that is correct up to a factor of 10.

Example 7. *Let*

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

Use the Power method to approximately compute the eigenvalue of A by largest modulus. Perform 3 *steps of the Power method. Use the entry of Ax_k that has the largest modulus as the scaling factor, and use*

$$x_0 = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Solution

$$Ax_0 = \begin{pmatrix} 6 \\ 11 \\ 14 \end{pmatrix}$$

$$\Rightarrow s_0 = 14$$

$$x_1 = \frac{Ax_0}{s_0} = \begin{pmatrix} 0.4286 \\ 0.7857 \\ 1.0000 \end{pmatrix}$$

$$Ax_1 = \begin{pmatrix} 5.0000 \\ 8.9994 \\ 11.2143 \end{pmatrix}$$

$$\Rightarrow s_1 = 11.2143$$

$$x_2 = \frac{Ax_1}{s_1} = \begin{pmatrix} 0.4459 \\ 0.8025 \\ 1.0000 \end{pmatrix}$$

$$Ax_2 = \begin{pmatrix} 5.0509 \\ 9.1018 \\ 11.3503 \end{pmatrix}$$

$$\Rightarrow s_2 = 11.3503$$

$$x_3 = \frac{Ax_2}{s_2} = \begin{pmatrix} 0.4450 \\ 0.8019 \\ 1.0000 \end{pmatrix}$$

$$Ax_3 = \begin{pmatrix} 5.0488 \\ 9.0976 \\ 11.3445 \end{pmatrix}$$

$$\Rightarrow s_3 = 11.3445$$

Best estimate for the eigenvalue of A is 11.3445.
True eigenvalues of A are:

$$eig(A) = \begin{pmatrix} -0.5157 \\ 0.1709 \\ 11.3448 \end{pmatrix}$$

Power method is converging to the eigen value with largest modulus.

What about the other eigenvalues? We have seen that we can compute the largest eigenvalue of A by the power method. If we apply the power method to A^{-1}, we will get the largest eigenvalue of A^{-1}, which is λ_n^{-1}.
A major limitation of power method is that it only find the largest eigenvalue. Similarly, the inverse power iteration sees only able to find the eigenvalue of smallest magnitude. Also power method fails when matrix has no dominant eigenvalue. As we are looking for more efficient and more convergent method for finding all eigenvalues so we switch to Cholesky and Jacobi methods.

3.6.2 Cholesky decomposition

Let $A = [a_{ij}]$ be any positive definite matrix then A can be factorized in the the form $A = LL^T$, this is nice form of LU factorization. This is possible when A is symmetric and positive definite ($x^T Ax > 0$ if $x^T x > 0$), with condition $U = L^T$. Equating row by row corresponding elements in the equation $A = LL^T$ we can determine elements of L, where L is diagonal matrix with positive entries.

$$\begin{pmatrix} a_{11} & \dots & \dots & a_{1n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \dots & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & \dots & 0 \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ l_{n1} & & \dots & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & & \dots & \dots & l_{n1} \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & \dots & \dots & l_{nn} \end{pmatrix}$$

We have in left side

$$a_{ij} = \sum_{k=1}^{j} l_{ik} l_{jk}, \qquad j = 1, 2, ..., i.$$

This gives

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}, \qquad j = 1, 2, ..., i-1.$$

Hence

$$l_{ii} = \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{1/2}.$$

Now we apply another approach to get partition, which is **Block Cholesky Factorization**.

$$\begin{pmatrix} a_{11} & b^T \\ b & \hat{A} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ h & \hat{G} \end{pmatrix} \begin{pmatrix} l_{11} & h^T \\ 0 & \hat{G}^T \end{pmatrix}$$

Next procedure is " how to find l_{11}, h, and $\hat{G}$? " For this compare left and right hand side entries, which implies

$$l_{11} = \sqrt{a}_{11}$$

$$h = \frac{b}{l_{11}}$$

For $\hat{G}$:

$$\hat{A} = hh^T + \hat{G}\hat{G}^T$$

$$\hat{G} = (\hat{A} - hh^T)\hat{G}^T$$

$\hat{G}$ is Choleski factor of

$$\hat{A} - hh^T$$

Here we present simple Cholesky method for LL^T factorization.

```
function [L]  = iqbal_chol(M)

n = length(M);    % calculate length of input matrix to use in for loop
L  = zeros(n,n);  % create a matrix of 0's
% loop through all the elements and do the required calculations
for i=1:n
   L(i,i) = sqrt(M(i,i) - L(i,:)*L(i,:)');
   for j=(i+1):n
       L(j,i) = (M(j,i) - L(i,:)*L(j,:)')/L(i,i);
   end
end

disp('This is L: ');  %Matlab display L'
disp(L);
 disp('This is L primed: ');
disp(L');
```

Example 8. *Find Cholesky decomposition of any square matrix M by using Matlab cods.*

Here is example for $n = 3$

```
M=rand(3)

M =

    0.0118    0.2987    0.4692
    0.8939    0.6614    0.0648
    0.1991    0.2844    0.9883

M=M*M'                    % Now M is symmetric

M =
```

```
    0.3095    0.2385    0.5511
    0.2385    1.2408    0.4302
    0.5511    0.4302    1.0974

 This is L:              % We display only final answer

    0.5564         0         0
    0.4287    1.0281         0
    0.9904    0.0054    0.3411
```

Example 9. *Here is another example for $n = 4$ to find Cholesky decomposition of any square matrix M by using Matlab cods.*

```
M=rand(4)               % M is random square matrix of order 4

M =

    0.5828    0.4329    0.5298    0.7833
    0.4235    0.2259    0.6405    0.6808
    0.5155    0.5798    0.2091    0.4611
    0.3340    0.7604    0.3798    0.5678

M=M*M'

M =                               % Now M is symmetric

    1.4214    1.2173    1.0234    1.1698
    1.2173    1.1042    0.7972    0.9431
    1.0234    0.7972    0.8582    0.9543
    1.1698    0.9431    0.9543    1.1564

 This is L:

    1.1922         0         0         0
    1.0211    0.2483         0         0
    0.8584   -0.3193    0.1393         0
    0.9812   -0.2366    0.2613    0.2633
```

Example 10. *Here is special example for $n = 3$ to find Cholesky decomposition of any square matrix M which is not symmetric we observe complex entries in L.*

```
M=rand(3)                 % M is random square matrix of order 3

M =                      % M is  not symmetric

    0.8180    0.2897    0.7271
    0.6602    0.3412    0.3093
    0.3420    0.5341    0.8385


 This is L:

   0.9044                  0                   0
   0.7300                  0 + 0.4378i         0
   0.3781                  0 - 0.5894i    0.5901
```

These are examples of simple Cholesky decomposition. In next chapter we will apply Cholesky iterative method for finding eigenvalue by converting a square matrix in diagonal form.

3.6.3 Jacobi Method

This is widely used method to find the eigenvalues and eigenvectors of a real symmetric matrix. It is easily understandable and reliable method that produces uniformly accurate answers for the results [3]. The algorithm is very accurate for matrices of order up to 10. In case of matrices of higher order like 20×20 it is little bit slow but results are quite acceptable.
Since all the eigenvalues of A are real and there exist a real orthogonal matrix S, such that, $S^{-1}AS = D$, (Diagonal matrix).
As D and A are similar, the diagonal elements of D are the eigenvalues of A. This method finds the spectral decomposition $A = VDV^T$ of a symmetric matrix. It finds eigenvalues and eigenvectors in one sweep. This algorithm is slow but stable.

Jacobi's method is based Jacobi rotation $G = R(i, j, \theta)$ such that the (i , j)-th entry of matrix G^tAG be zero. Symmetrically the (j , i)-th entry

will also be zero. We can write

$$B = G^t AG = \begin{pmatrix} & \vdots & & \vdots & \\ \dots & b_{ii} & \dots & b_{ij} & \dots \\ & \vdots & & \vdots & \\ \dots & b_{ji} & \dots & b_{jj} & \dots \\ & \vdots & & \vdots & \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & & & \\ & c & & s & \\ & & 1 & & \\ & & & 1 & \\ & -s & & c & \\ & & & & 1 \end{pmatrix} \begin{pmatrix} & \vdots & & \vdots & \\ \dots & a_{ii} & \dots & a_{ij} & \dots \\ & \vdots & & \vdots & \\ \dots & a_{ji} & \dots & a_{jj} & \dots \\ & \vdots & & \vdots & \end{pmatrix} \begin{pmatrix} 1 & & & & \\ & c & & -s & \\ & & 1 & & \\ & & & 1 & \\ & s & & c & \\ & & & & 1 \end{pmatrix}$$

in this calculation we make sure that $b_{ij} = b_{ji} = 0$ in terms of the elements of A and c, s. where

$$s = \left(\frac{1}{2} - \frac{\beta}{2\sqrt{1+\beta^2}} \right)^{1/2}$$

and

$$c = \left(\frac{1}{2} + \frac{\beta}{2\sqrt{1+\beta^2}} \right)^{1/2}$$

the value of β is found as

$$\beta = \frac{a_{ii} - a_{jj}}{2a_{ij}}$$

In the Jacobi algorithm we choose the rotation matrix and its transpose to get a zero at the required position.
We presents here some computational aspects of Jacobi method. Take a_{ij}, which is largest offdiagonal element. We construct matrix G by using values of c and s . Set $A_1 = G^t AG$. Repeat the process with new offdiagonal of A_1 matrix. In result we get

$$A_0 = A$$

$$A_{n+1} = G_n^t A_n G_n$$

To find the diagonal matrix D, take limit as,

$$\lim_{n \to \infty} A_n \to D$$

Our goal is to get sufficiently near to exact solution. This can be achieved by running the iteration until the off-diagonal elements are sufficiently small. In result we have

$$A \approx QDQ^t$$

where

$$Q = G_1 G_2 ... G_N,$$

which is spectral decomposition of the matrix A.

Example 11. *Find the eigenvalues and eigenvectors of the symmetric matrix by using Jacobi Method [5].*

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

Solution: Choose largest off-diagonal element that is '2' at (1,2), (1,3) and (2,3) positions. Choosing largest offdiagonal element is called " Classical Jacobi method". The usual approach is to eliminate each element in turn. We call a Jacobi sweep for every $\frac{n(n-1)}{2}$ plane rotations reducing all off-diagonal elements.
The rotation angle θ is given by $tan2\theta = \frac{2a_{12}}{a_{11}-a_{22}}$
For this problem $\theta = \frac{\pi}{4}$.Thus the orthogonal matrix S_1 is

$$S_1 = \begin{pmatrix} cos\pi/4 & -sin\pi/4 & 0 \\ sin\pi/4 & cos\pi/4 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Then the first rotation yields

$$D_1 = S_1^{-1} A S_1 = \begin{pmatrix} 3 & 0 & 2.82843 \\ 0 & -1 & 0 \\ 2.82843 & 0 & 1 \end{pmatrix}$$

The largest off-diagonal element of D_1 is now 2.82843 situated at $(1,3)$ position and hence the rotation angle is $\theta = 0.61548$.
The second orthogonal matrix S_2 is

$$S_2 = \begin{pmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{pmatrix} = \begin{pmatrix} 0.8165 & 0 & -0.57735 \\ 0 & 1 & 0 \\ 0.57735 & 0 & 0.8165 \end{pmatrix}$$

Thus the second rotation gives

$$D_2 = S_2^{-1} A S_2 = \begin{pmatrix} 5 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

which is diagonal matrix and eigenvalues are $5, -1, -1$.
Note: It is very rare for Jacobi method to converge in a finite number of steps. The eigenvectors are the columns of matrix S, where,

$$S = S_1 S_2 = \begin{pmatrix} 0.57735 & -0.70711 & -0.40825 \\ 0.57735 & 0.70711 & -0.40825 \\ 0.57735 & 0 & 0.8165 \end{pmatrix}$$

Example 12. *Find one-sweep Jacobi rotations for matrix A by using matlab codes.*

Here is a matlab m-file that does one-sweep of the Jacobi method.

```
function [A] = jac_sweep(A)

sz = size(A);
n = sz(1); % not checking square

for i=1:n-1,
    for j = i+1:n,
                      % zero out i,j and j,i elements
        indx = [i j];
        B = A(indx,indx);
        [c, s] = jac2(B);
        J = [c -s ; s c];
        Q = eye(n);
        Q(indx, indx) = J;
        A1 = Q'*A*Q;     % simple but very inefficient
        A1 = (A1+A1')/2;
        A = A1;
    end
end

   % compute the cos and sine for Jacobi rotation applied to B
   function [c s] = jac2(B)
       if abs(B(1,2)) > 1e-16,
        tau = (B(1,1) - B(2,2))/(2*B(1,2));
```

```
        t = min([ -tau + sqrt(1 + tau*tau),
        -tau - sqrt(1 + tau*tau)]) ;
        else
        t = 0;
       end
   c = 1/(sqrt(1+t*t));
   s = t*c;
```

Example 13. *Apply one-sweep Jacobi method to any square matrix A to see its results.*

We select randomly a square matrix A of order 4 and apply one-sweep Jacobi method after making it positive definite.

```
   A=rand(4)

  4.4510e-001  8.4622e-001  8.3812e-001  8.3180e-001
  9.3181e-001  5.2515e-001  1.9640e-002  5.0281e-001
  4.6599e-001  2.0265e-001  6.8128e-001  7.0947e-001
  4.1865e-001  6.7214e-001  3.7948e-001  4.2889e-001

A=A*A'          %  A is symmetric

  2.3085e+000  1.2938e+000  1.5400e+000  1.4299e+000
  1.2938e+000  1.3973e+000  9.1075e-001  9.6618e-001
  1.5400e+000  9.1075e-001  1.2257e+000  8.9411e-001
  1.4299e+000  9.6618e-001  8.9411e-001  9.5499e-001

jac_sweep(A)   % apply Jacobi sweep for A

ans =

  4.2181e-001  1.3568e-001  9.7757e-002 -1.5154e-001
  1.3568e-001  2.0282e-001  1.4772e-001 -3.1795e-001
  9.7757e-002  1.4772e-001  1.2182e-001  2.4443e-016
 -1.5154e-001 -3.1795e-001  2.4443e-016  5.1400e+000
```

Results In one-sweep all elements are not set to 0 at once, but each time one element is set to 0, the previous 0 may be made non-zero, so only the

$(3,4)$ and $(4,3)$ elements which is $8.9411e-001$ initially is selected and reduced in one sweep to $2.4443e-016$ which is sufficiently zero.

3.6.4 Singular Value Decomposition (SVD)

We are looking for the *best form* to reduce a matrix. The diagonal form is the best form of a matrix. As it is extremely easy to manipulate with diagonal matrices. In other words, if one is allowed to change bases independently, then one gets the best possible form one could hope for. This is called Singular Value Decomposition (SVD). It is most useful decomposition.
Definition Let $A \in M_{m,n}$. Then there are unitary matrices U and V of appropriate dimensions and a non-negative diagonal matrix $\Sigma \in M_{m,n}$ such that

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

The diagonal entries of Σ, ordered in non-increasing order, and

$$\sigma_{11} \geqslant \sigma_{22} \geqslant .... \geqslant \sigma_{qq} \geqslant 0$$

are the singular values of A where $q = min(m,n)$.

Theorem: Let $A \in M_{m,n}(F)$. Let $q = min(m,n)$. Then

1. There exist unitary $U \in M_m(F)$ and $V \in M_n(F)$ and real diagonal matrix $\Sigma \in M_{m,n}$ such that

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

where Σ is diagonal in the sense that

$$\sigma_{ij} = 0 \qquad if \quad i \neq j$$

We set

$$\sigma_i = \sigma_{ii}$$

2. The σ_i's are uniquely determined, and they are denoted by $\sigma_i(A)$ and are called the singular values of A.

3. The columns of U and V are called the left and right singular vector of A. If the σ_i are distinct then the singular vectors are unique up to multiplication by a number of modulus 1.

How to compute practically the SVD of A?

For any square matrix A , we have

$$A^*A = U\Sigma^2V^*$$

We can compute the eigenvalues of A^*A and then take square roots to get the singular values. It is not a good idea as we "square the condition number" each time.
Now we present another idea through a theorem.

Theorem 4. *Let $A \in M_{m,n}$. The corresponding Jordan-Wielandt matrix*

$$W_A = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \in M_{m+n}$$

has eigenvalues as

$$\pm\sigma_1(A), ..., \pm\sigma_q(A), 0, ..., 0$$

We have then

$$\begin{pmatrix} 0 & A^* \\ A & 0 \end{pmatrix} \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} A^*U & -A^*U \\ AV & AV \end{pmatrix} \tag{3.4}$$

From definition of SVD we can write $AV = U\Sigma$ also $A^* = (U\Sigma V^*)$
$\Rightarrow A^*U = V\Sigma$
Equation(3.3) is written as

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

$$= \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix}$$

this gives diagonal form of last matrix.
Further it can be written as

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

It proves that to calculate the singular values of A, we can calculate instead the eigenvalues of eigenvalues of W_A. It is useful connection between eigenvalues and singular values of any matrix A.

Chapter 4

Accuracy of Methods

In this chapter we shall discussing convergence of Cholesky and Jacobi methods through iterative schemes and shall compare the rate of convergence for both methods.

4.1 Cholesky Iterative Method to compute eigenvalues

We know by Cholesky factorization any square matrix A can be written $A = LL^T$. Now we want to construct a iterative Cholesky Algorithm for computation of eigenvalues of A_0. For this first we convert the given matrix into LL^T by using Cholesky method and then set A_1 equal to $L^T L$. Here we use the definition of similar matrices,i.e., similar matrices have same eigenvalues. By continuing the process we get a diagonalized matrix A_i where i is number of iterations. The diagonal entries are eigenvalue of matrix A_0. Here are mathematical steps which are being used in Matlab codes. Assume

$$A_0 > 0, \qquad (Positive\ definite)$$

Factorize into $A_0 = LL^T$
set $A_1 \leftarrow L^T L$
Again factorize for $A_1 = L_1 L_1^T$
then set $A_2 \leftarrow L_1^T L_1$
Again factorize for $A_1 = L_1 L_1^T$
Repeating this process until diagonalized L. Diagonal entries of this resultant matrix are eigenvalues of starting matrix A_0.

Matlab codes for Cholesky iterative method:
Here are Matlab codes for Cholesky iterative method to find eigen values of square matrix by using first converting it into tridiagonal form.

In first section we present Matlab codes which checks that all off-diagonal elements are zero. For this we set ϵ_1 as precondition. This process operates until we get diagonal matrix.

```
function d = isDiag(A,s)
 d=1;
 eps1=0.0001;
 for r=1:s
    for c=1:s
        if r==c
            continue
        elseif(A(r,c)>eps1)
            d=0;
            return
        else
            continue
        end
    end
end
```

Now consider second set of codes. We start by random square matrix A. Make it symmetric and then tridiagonal form. Now decompose it into Cholesky factors L. Then again set in the product of LL^T and apply Cholesky method for decomposition. In result we succeeded in converting more off-diagonal elements to zero. This is iterative method as it always take previous values and repeat again and again until all off-diagonal elements are reduced to zero.We can also set maximum number of iterations.

```
n=input ('enter value of n:  ');
 A=rand(n);       % take any square matrix of order n
  A=A*A';          % make A positive definite
   disp('The input array A is:')
   disp(A)
   A=hess(A);    % make A tridiagonal matrix
  [T,p]=chol(A); if p>0
    'Not a positive definite Matrix'
```

```
    return
end
 T=A;
for i=1:100         %max number of iterations
     L1=chol(T);
      A1=L1*L1'
      disp('iteration:  ')
      disp(i)
      disp('A1 after this iteration is:')
      disp( A1)
    if isDiag(A1,n)==0  % check for diagonal
        T=A1;
        continue
    else
        break
    end
end
 disp('     eigen(A)   eig using Cholesky')
  disp(sort([eig(A)     diag(A1)]))

'Total iterations  ', i
```

Example 14. *Find eigenvalues of any square matrix by using Cholesky-iterative method.*

Here is example for $n = 5$

```
  The input array A is:   % A is symmetric

    1.3838    1.0815    1.6375    0.8233    1.6341
    1.0815    2.0368    1.6443    1.1305    2.0806
    1.6375    1.6443    2.3864    1.4994    2.0649
    0.8233    1.1305    1.4994    1.1860    1.2218
    1.6341    2.0806    2.0649    1.2218    2.5416
```

```
iteration: 1

A1 after this iteration is:

    0.5077    0.2518         0         0         0
    0.2518    0.2890   -0.2307         0         0
         0   -0.2307    1.1774    1.9407         0
         0         0    1.9407    7.5603   -0.0168
         0         0         0   -0.0168    0.0001


iteration: 2

A1 after this iteration is:

    0.6326    0.1432         0         0         0
    0.1432    0.4883   -0.5259         0         0
         0   -0.5259    5.2674    3.7266         0
         0         0    3.7266    3.1463   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 3

A1 after this iteration is:

    0.6650    0.1215         0         0         0
    0.1215    1.0626   -1.6816         0         0
         0   -1.6816    7.6405    0.7044         0
         0         0    0.7044    0.1665   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 4

A1 after this iteration is:

    0.6872    0.1520         0         0         0
    0.1520    3.7583   -3.6578         0         0
         0   -3.6578    5.0233    0.0814         0
         0         0    0.0814    0.0657   -0.0000
```

```
         0         0         0   -0.0000    0.0000
```

```
iteration: 5

A1 after this iteration is:

    0.7208    0.3539         0         0         0
    0.3539    7.3167   -2.2674         0         0
         0   -2.2674    1.4359    0.0168         0
         0         0    0.0168    0.0611   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 6

A1 after this iteration is:

    0.8946    1.1142         0         0         0
    1.1142    7.8627   -0.7180         0         0
         0   -0.7180    0.7166    0.0049         0
         0         0    0.0049    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 7

A1 after this iteration is:

    2.2822    2.9975         0         0         0
    2.9975    6.5547   -0.2252         0         0
         0   -0.2252    0.6370    0.0015         0
         0         0    0.0015    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 8

A1 after this iteration is:

    6.2191    3.2103         0         0         0
```

```
    3.2103    2.6372   -0.1094         0         0
         0   -0.1094    0.6176    0.0005         0
         0         0    0.0005    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

iteration: 9

A1 after this iteration is:

    7.8763    1.2744         0         0         0
    1.2744    0.9922   -0.0860         0         0
         0   -0.0860    0.6054    0.0001         0
         0         0    0.0001    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 10

A1 after this iteration is:

    8.0824    0.4026         0         0         0
    0.4026    0.7954   -0.0749         0         0
         0   -0.0749    0.5960    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 11

A1 after this iteration is:

    8.1025    0.1247         0         0         0
    0.1247    0.7826   -0.0652         0         0
         0   -0.0652    0.5888    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 12

A1 after this iteration is:
```

```
    8.1044    0.0387         0         0         0
    0.0387    0.7862   -0.0564         0         0
         0   -0.0564    0.5833    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 13

A1 after this iteration is:
```

```
    8.1046    0.0121         0         0         0
    0.0121    0.7900   -0.0484         0         0
         0   -0.0484    0.5793    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 14

A1 after this iteration is:
```

```
    8.1046    0.0038         0         0         0
    0.0038    0.7930   -0.0413         0         0
         0   -0.0413    0.5763    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 15

A1 after this iteration is:
```

```
    8.1046    0.0012         0         0         0
    0.0012    0.7951   -0.0352         0         0
         0   -0.0352    0.5742    0.0000         0
         0         0    0.0000    0.0607   -0.0000
```

```
         0         0         0   -0.0000    0.0000
```

```
iteration: 16

A1 after this iteration is:

    8.1046    0.0004         0         0         0
    0.0004    0.7967   -0.0299         0         0
         0   -0.0299    0.5726    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 17

A1 after this iteration is:

    8.1046    0.0001         0         0         0
    0.0001    0.7978   -0.0253         0         0
         0   -0.0253    0.5715    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 18

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.7986   -0.0214         0         0
         0   -0.0214    0.5707    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 19

A1 after this iteration is:

    8.1046    0.0000         0         0         0
```

```
    0.0000    0.7992   -0.0181         0         0
         0   -0.0181    0.5701    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 20

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.7996   -0.0153         0         0
         0   -0.0153    0.5697    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 21

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.7999   -0.0129         0         0
         0   -0.0129    0.5694    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


 iteration: 22

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8001   -0.0109         0         0
         0   -0.0109    0.5692    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 23
```

```
A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8002   -0.0092         0         0
         0   -0.0092    0.5691    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 24

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8003   -0.0077         0         0
         0   -0.0077    0.5690    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 25

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8004   -0.0065         0         0
         0   -0.0065    0.5689    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 26

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8004   -0.0055         0         0
         0   -0.0055    0.5688    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 27

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8005   -0.0046         0         0
         0   -0.0046    0.5688    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 28

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8005   -0.0039         0         0
         0   -0.0039    0.5688    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 29

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8005   -0.0033         0         0
         0   -0.0033    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 30

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8005   -0.0028         0         0
         0   -0.0028    0.5687    0.0000         0
```

```
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

iteration: 31

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0023         0         0
         0   -0.0023    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

iteration: 32

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0020         0         0
         0   -0.0020    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 33

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0017         0         0
         0   -0.0017    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 34

A1 after this iteration is:

    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0014         0         0
```

```
         0   -0.0014    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 35

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0012         0         0
         0   -0.0012    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 36

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0010         0         0
         0   -0.0010    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 37

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0008         0         0
         0   -0.0008    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


 iteration: 38

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0007         0         0
         0   -0.0007    0.5687    0.0000         0
```

```
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 39

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0006         0         0
         0   -0.0006    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 40

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0005         0         0
         0   -0.0005    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 41

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0004         0         0
         0   -0.0004    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 42

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0004         0         0
         0   -0.0004    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
```

```
         0         0         0   -0.0000    0.0000


iteration: 43

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0003         0         0
         0   -0.0003    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

iteration: 44

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0003         0         0
         0   -0.0003    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 45

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0002         0         0
         0   -0.0002    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 46

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0002         0         0
         0   -0.0002    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000
```

```
iteration: 47

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0002         0         0
         0   -0.0002    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 48

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0001         0         0
         0   -0.0001    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000


iteration: 49

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0001         0         0
         0   -0.0001    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

iteration: 50

A1 after this iteration is:
    8.1046    0.0000         0         0         0
    0.0000    0.8006   -0.0001         0         0
         0   -0.0001    0.5687    0.0000         0
         0         0    0.0000    0.0607   -0.0000
         0         0         0   -0.0000    0.0000

    eigval(A)   eigval using Cholesky
    0.0000          0.0000
```

```
0.0607          0.0607
0.5687          0.5687
0.8006          0.8006
8.1046          8.1046
```

Comments: It is a nice example in which matrix converge to diagonal form in 50 iterations. We observe the fact that the off diagonal entries converge linearly to 0. Consider $(3, 2)$ entry of matrix A_1. In first iteration it is (-0.2307). Its value increases in first four iterations and then start decreasing in every iteration. The rate of convergence is very high for large values. After 10 iterations its value is (-0.0749), which is about three times decrease in stating value. After 20 iterations, the decrease is 15 times with respect to starting value and 5 times as compared to 10-th iteration. Similarly we notice that rate of decease is very high for 30-th and 40-th iterations. It is about 80 times and 500 times respectively. The rate of decrease for off diagonal elements is less when this value is sufficiently near to zero. For example in 41st iteration, the value of entry (2,3) is (-0.0004) which is same for next iteration. This means that rate of convergence is slow when element value is too small. In result of this continuous decrease, the value of entry $(3, 2)$ converges linearly to $\epsilon = 0.0001$. Similarly the other off-diagonal entries converges linearly to zero so the matrix A become diagonal. The diagonal entries are eigenvalues of matrix $diagA$ which are same as that of the eigenvalues of matrix A, computed from the computer.

Some more examples with different n-values

Example 15. *To find eigenvalue of matrix by using Cholesky iterative method.*

We take a small matrix of n = 2 in long format.

```
A1 =           % starting with symmetric matrix

   1.57489724733728   0.12141320187440
   0.12141320187440   0.01296269516289

   %After 89 iterations we get
   A1 =

   1.58427866132836   0.00000000000000
   0.00000000000000   0.00358128117180


ans =
   [eig(A)              (eig(diagA1)]
   0.00358128117180   0.00358128117180
   1.58427866132836   1.58427866132836
```

Eigenvalues in both columns are exactly same.

Example 16. *To find eigenvalue of matrix by using Cholesky iterative method with long format.*

Let us consider matrix of order $n = 4$ in long format.

```
A1 =                    % A is tridiagonal matrix

   0.21569190405489  -0.01008604686090                   0                   0
  -0.01008604686090   0.29671268251995   0.82660931243546                   0
                  0   0.82660931243546   2.55889895739011  -0.17379533705833
                  0                   0  -0.17379533705833   0.12450366947428
```

```
Total iterations =200        % after 200 iterations A1 is diagonal

  A1 =

   2.83881735297896  -0.00000000000000                   0                   0
  -0.00000000000000   0.21627220943165   0.00000000000001                   0
                  0   0.00000000000001   0.13981059419017  -0.00000000000000
                  0                   0  -0.00000000000000   0.00090705683845


ans =
   [eig(A)              diag(A1)]
   0.00090705683845   0.00090705683845
   0.13981059419017   0.13981059419017
   0.21627220943165   0.21627220943165
   2.83881735297896   2.83881735297896
```

Comments : After 200 iterations, which were fixed for this problem, the resulting matrix is not fully diagonal. Element at position $(2, 3)$ is (0.00000000000001), which is sufficiently near to zero. It means the matrix still needs more iterations to converge to diagonal form. Even then the calculated eigenvalues are exactly same the of matrix A.

Example 17. *To find eigenvalue of matrix by using Cholesky iterative method.*

Let us consider square matrix of order $n = 6$ in long format.

```
              % A is tridiagonal matrix

A1 =

  Columns 1 through 4

   0.19533323556977   0.01398283556730                   0                   0
   0.01398283556730   0.11082581131561  -0.07202155712020                   0
                  0  -0.07202155712020   0.16361942031202   0.20167753449254
                  0                   0   0.20167753449254   2.14017667644203
                  0                   0                   0   3.69095244262304
                  0                   0                   0                   0
```

```
  Columns 5 through 6

                  0                   0
                  0                   0
                  0                   0
   3.69095244262304                   0
  10.74368296306688  -0.15972018150605
  -0.15972018150605   0.00824155939965

  Total iterations = 200

  A1 =

  Columns 1 through 4

  12.11236890017097   0.00000000000000                   0                   0
   0.00000000000000   0.83161400302255  -0.00000000000000                   0
                  0  -0.00000000000000   0.20061461638438   0.00000316667893
                  0                   0   0.00000316667893   0.17514140273163
                  0                   0                   0   0.00000000000000
                  0                   0                   0                   0

  Columns 5 through 6

                  0                   0
                  0                   0
                  0                   0
   0.00000000000000                   0
   0.04204003036315  -0.00000000000000
  -0.00000000000000   0.00010071343329


ans =
    [eig(A)                diag(A1)]
   0.00010071343329   0.00010071343329
   0.04204003036315   0.04204003036315
   0.17514140233796   0.17514140273163
   0.20061461677805   0.20061461638438
   0.83161400302255   0.83161400302255
  12.11236890017097  12.11236890017097
```

Comments : This is an interesting example like previous one. After 200 iterations, which were fixed for this problem, the resulting matrix is not fully diagonal. Element at positions $(4, 3)$ and $(3, 4)$ is (0.00000316667893), which is s near to zero. It means that the matrix still needs more iterations to converge to diagonal form. Thus the calculated eigenvalues are not same like matrix A. Two calculated eigenvalues are different. The results for these value are correct up to 9 digits only. We can calculate more accurate eigenvalues by doing more iterations.

Example 18. *To find eigenvalue of matrix by using Cholesky iterative method.*

Let us consider matrix of order $n = 8$ in long format.

```
A1 =

  Columns 1 through 4

   0.01124719261622  -0.00104377027794                   0                   0
  -0.00104377027794   0.23514859215081   0.03007700177251                   0
                  0   0.03007700177251   0.46088278629551  -0.18483885526471
                  0                   0  -0.18483885526471   0.69015252137979
                  0                   0                   0   0.35631014869781
                  0                   0                   0                   0
                  0                   0                   0                   0
                  0                   0                   0                   0

  Columns 5 through 8

                  0                   0                   0                   0
                  0                   0                   0                   0
                  0                   0                   0                   0
   0.35631014869781                   0                   0                   0
   0.84326532838711  -0.53447440386778                   0                   0
  -0.53447440386778   3.85513262884968   5.40691823731328                   0
                  0   5.40691823731328  12.81628614402076  -0.69061828871510
                  0                   0  -0.69061828871510   0.13260218509142

                  Total iterations  =200
```

```
A1 =

  Columns 1 through 4

  15.38700744365881  -0.00000000000000                   0                   0
  -0.00000000000000   1.69362540608433   0.00000000000002                   0
                   0   0.00000000000002   0.95809189912697  -0.00000000000000
                   0                   0  -0.00000000000000   0.49903353284934
                   0                   0                   0   0.00000000000000
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0

  Columns 5 through 8

                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
   0.00000000000000                   0                   0                   0
   0.25640091859294  -0.00001175785708                   0                   0
  -0.00001175785708   0.22221989855918   0.00000000000000                   0
                   0   0.00000000000000   0.01709601436317  -0.00000000003717
                   0                   0  -0.00000000003717   0.01124226555658


   [eig(A)             diag(A1)]
   0.01124226555658   0.01124226555658
   0.01709601436317   0.01709601436317
   0.22221989451462   0.22221989855918
   0.25640092263750   0.25640091859294
   0.49903353284934   0.49903353284934
   0.95809189912697   0.95809189912697
   1.69362540608433   1.69362540608433
  15.38700744365880  15.38700744365881
```

Comments : This is almost same case as we discussed in previous example. The resulting matrix is not fully diagonal. Thus the calculated eigenvalues are not same like matrix A. There is a difference in three calculated and exact eigenvalues. It means the matrix still needs more iterations to converge to diagonal form.

4.2 Jacobi's Method for computing eigenvalues and eigenvectors

Consider the eigenvalue problem

$$AX = \lambda X$$

Start with the real symmetric matrix A. We want to construct the sequence of the orthogonal matrices $J_1, J_2, ..., J_n$ as follow:

$$D_0 = A$$

$$D_k = J_k' D_{k-1} J_k$$

for k=1,2,... Now we proceed to construct the sequence J_k, so that

$$lim_{k\to\infty} D_k = D = diag(\lambda_1, \lambda_2, ...\lambda_n)$$

The stoping criteria is when off-diagonal elements are sufficiently near to zero then halt further computation. In result we have

$$D_n \approx D.$$

The result produces

$$D_n = J_n' J_{n-1}' ... J_1' A J_1 J_2 ... J_{n-1} J_n \tag{4.1}$$

where

$$J = J_1 . J_2 ... J_{n-1} . J_n$$

then

$$J^{-1} A J = D_k$$

$$\Rightarrow AJ = JD_k \approx Jdaig(\lambda_1, \lambda_2, ...\lambda_n) \tag{4.2}$$

Let the columns of R be denoted by the vectors $X_1, X_2, ..., X_n$ then J can be expressed as a row vector of column vectors :

$$J = [X_1 \quad X_2 .... X_n]$$

Finally we get the relation

$$[AX_1 \quad AX_2 .... AX_n] \approx [\lambda_1 X_1 \quad \lambda_2 X_2 .... \lambda_n X_n]$$

this is relation between eigen vectors of J and corresponding eigenvalues λ_k.

Working procedure

Each step in Jacobi's method will accomplish the limited objective of reducing the two off-diagonal elements d_{ij} and d_{ji} to zero. At the same time there is reduction in sum squares of off-diagonal elements. For this first select row i and column j for which $a_{ij} \neq 0$. In second step it calculates the quantities c, s and θ. Then each time the algorithm set $D = D_1$ and then iterates the same procedure.
Let J_1 denote the first orthogonal matrix used. Suppose that;

$$D_1 = J_1' A J_1$$

reduces the elements d_{ij} and d_{ji} to zero, where J_1 is of the form

$$J_1 = \begin{pmatrix} 1 & & & & & \\ & c & & & s & \\ & & 1 & & & \\ & & & 1 & & \\ & -s & & & c & \\ & & & & & 1 \end{pmatrix}$$

Because sum-squares of off-diagonal elements is reducing, the sequence D_j converges to diagonal matrix D. The magnitude of off-diagonal elements are compared to ϵ, the pre-assigned tolerance. When all off-diagonal elements met this criteria we get diagonal matrix D. Diagonal entries of D are eigenvalues of initial matrix A. At end these calculated eigenvalues are compared to eigenvalues of matrix A computed by computer.

4.2.1 Here are Matlab codes for Jacobi's Method

```
% method to calculate eigenvalues by  jac_sweep iterations

format('long');    % set digit precision
n=input ('enter value of n:  '); % read matrix dimensions
A=randn(n);     % generate a random matrix of the size just read
A=A*A';         % calculate symmetric matrix
T=A;            % T is a temporary matrix for the coming calculations
disp('The input array A is:');
disp(A);        % display the original matrix
for i=1:100        %max Iteration to go through
    L1=jac_sweep(T); % use jac_sweep to zeros off-diagonal elements
     disp('iteration:  ')
     disp(i)

    if isDiag_ssq(L1,n)==0 % determine whether the resulting matrix is diagonal
        T=L1;        % continue if it is not
        continue
    else
        break
    end
end
% calculate, sort and display the eigen values of our matrices to compare
% the results
sort([eig(A) diag(L1)])
 'Total iterations  ', i

% function to calculate sum of square roots of off diagonal elements
% calls isDiag to determine whether a matrix is diagonal then calculates
function d = isDiag(A,s)
 d = 1;
 eps1=1e-12;
 ssq = 0;
 for r=1:s-1
     for c=r+1:s
          ssq = ssq + A(r,c)^2; % this is where it adds the values of square roo

     end
 end
```

```
% calculate and display the value of the sum of square roots
'Sqrt of ssq' sqrt(ssq) if sqrt(ssq) > eps1
    d = 0;
end
```

Example 19. *Use Matlab codes for Jacobi iterations to transform the symmetric matrices of order* 10×10 *to diagonal form.*

```
 The input array A is:  % after transforming into symmetric form
  Columns 1 through 4

   4.92710426689493  -0.73123145016046   1.24525579943457  -5.32406359697361
  -0.73123145016046   6.81880151866267  -0.54293841246707   0.66062742639089
   1.24525579943457  -0.54293841246707   5.54186551184006  -2.66189154132271
  -5.32406359697361   0.66062742639089  -2.66189154132271  17.75552652820081
  -0.03843179645927   4.23079660267863   2.58878779079049  -1.79535706393124
  -1.10406936493870  -2.96376960993967   2.25157227754740   0.80569204415550
  -1.81285344851994  -2.76930612417051   0.66097134550134   5.66664043634968
   2.45397888885462  -0.25877131970736  -2.24300075322014  -7.09568605948711
   4.13699337840298   1.13733527372539   2.37494300368035  -6.68966953927205
   0.06921060048142  -2.06507713955600   1.74873663660925  -2.00250153441941

  Columns 5 through 8

  -0.03843179645927  -1.10406936493870  -1.81285344851994   2.45397888885462
   4.23079660267863  -2.96376960993967  -2.76930612417051  -0.25877131970736
   2.58878779079049   2.25157227754740   0.66097134550134  -2.24300075322014
  -1.79535706393124   0.80569204415550   5.66664043634968  -7.09568605948711
   7.07523415313795  -1.56030631327115  -0.52038650945168  -0.39024019047969
  -1.56030631327115   5.12406014637303   2.92621808184685  -2.32052161823381
  -0.52038650945168   2.92621808184685   7.90963836595350  -2.34603603907912
  -0.39024019047969  -2.32052161823381  -2.34603603907912   7.16639485430050
   0.09797946434719  -1.26953768511102  -3.55902374495130   3.53024415350407
  -0.00292755601975   1.40772697327753   0.19640995960539  -0.44726870751898

  Columns 9 through 10

   4.13699337840298   0.06921060048142
   1.13733527372539  -2.06507713955600
```

```
   2.37494300368035   1.74873663660925
  -6.68966953927205  -2.00250153441941
   0.09797946434719  -0.00292755601975
  -1.26953768511102   1.40772697327753
  -3.55902374495130   0.19640995960539
   3.53024415350407  -0.44726870751898
  10.83716546981608  -0.95712002016701
  -0.95712002016701   1.74937502893952

Iteration: 1
Sqrt of ssq = 8.9427e+000

Iteration: 2
Sqrt of ssq = 1.9727e+000

Iteration: 3
Sqrt of ssq = 3.0269e-001

iteration: 4
Sqrt of ssq =  1.1834e-002

iteration: 5
Sqrt of ssq = 2.6852e-005

iteration: 6
Sqrt of ssq = 8.1106e-012

iteration: 7
Sqrt of ssq =  2.2204e-016

   [eig(A)       diag(L1)]
  3.3838e-002  3.3838e-002
  3.6222e-001  3.6222e-001
  9.5270e-001  9.5270e-001
  1.5914e+000  1.5914e+000
  3.0482e+000  3.0482e+000
  5.1223e+000  5.1223e+000
  7.8689e+000  7.8689e+000
  1.1215e+001  1.1215e+001
  1.4008e+001  1.4008e+001
  3.0703e+001  3.0703e+001
```

Comments: We get results in 7 iterations . Eigenvalues are exactly same. In first iteration $Sqrt\,of\ ssq = 8.9427e+000$ which is reduced to $Sqrt\,of\ ssq = 1.9727e+000$ in second run. This shows that Jacobi method is converging quadratically.

Example 20. *Use Matlab codes for Jacobi iterations to transform the symmetric matrices of order* 20×20 *to diagonal form.*

```
The input array A is:
 Columns 1 through 4

 18.33274113269596   8.69063532557255  -8.05080742697037  -0.96202282467986
  8.69063532557255  19.46968766495169  -5.39095457475524  -0.04317196901132
 -8.05080742697037  -5.39095457475524  14.43579663929603   1.93855268715616
 -0.96202282467986  -0.04317196901132   1.93855268715616  19.82384132725523
  4.80923555559124   3.71909912376392  -2.05369996319796  -6.29614390474461
  8.44585029786143   2.15758931270338  -0.41792313542981   1.11738001300005
  4.04219454083908   4.15139307014492   3.80635394253985   1.99624826423733
  4.50525189729215   4.92162864528229  -4.02871588487598   1.00598889768494
 -2.80917027670684   1.10436192374786   0.57792662993850   3.29202931182155
  4.49524067860350  -0.75430208080295  -4.24950641271152  -3.78643567441964
 -3.52103855354539   3.57750484830174   2.71752440158303   6.15786515520179
 -7.87996008113431   2.15109723190010   2.90011089067157   3.54609009021479
  4.80002265904681  -1.59105805368087   1.29356820990489   7.38470429023030
  1.66717040639530   1.14545917632083  -4.36092392862978  -1.91712736186076
 -3.80725360173308   7.05813185145915   6.02825993049226   3.20436113613495
  1.72471100875360   3.61060094661089  -3.33614466357950   4.75588750380413
  6.81572444425526   4.91771989320600  -5.15723268330519  -6.22541483185501
 -1.79094569115684   2.21104150087101   1.32024529078390   5.91887377065978
  7.19510238432702   5.88149749387292  -0.06753554887202  -1.18611708282151
 -3.01670335976680  -5.40505279830859  -3.46917850882857   0.01551779178417

 Columns 5 through 8

  4.80923555559124   8.44585029786143   4.04219454083908   4.50525189729215
  3.71909912376392   2.15758931270338   4.15139307014492   4.92162864528229
 -2.05369996319796  -0.41792313542981   3.80635394253985  -4.02871588487598
 -6.29614390474461   1.11738001300005   1.99624826423733   1.00598889768494
 14.36099946332437   3.73046883821772   1.10844810537455  -5.33900390221854
  3.73046883821772  24.38276377925606   5.65799421780889  -4.58252434307211
```

```
 1.10844810537455   5.65799421780889  13.66103168184937   1.97999449452723
-5.33900390221854  -4.58252434307211   1.97999449452723  15.00822013660078
-0.05521626329838   1.85219726366423  -7.29662813475638  -1.85849141290655
-2.43082953900282   3.05065135460699  -0.93762358626709  -3.21364576272254
-2.40268468385147  -3.10783986924079   6.36199769209926  -2.20192162924585
-1.89673904423745  -0.85346280807182   6.91710281913630   1.89641964906194
-2.07065108483332   7.49997231296922  13.00254144974241   1.58294777412672
 7.55480336539587  -1.32008757986820  -4.69947306080261  -3.92886713309874
-0.00596942006038  -2.96394598497866   3.06920670625717  -1.50690283174512
-4.25217716278094  -4.10668478685671  -0.82559016994131   9.70423242893581
 0.96514731283814   0.97727344005285  -8.21499784888196  -1.70909340333592
-3.83184464730678   6.55765762103624  -0.38964057450520   1.21426787642390
 1.76472662201430  10.64116320585426   3.93486714955786  -1.87052606954003
-2.86087258885245   1.79706122754855  -3.13443959529720   1.59561888208891

Columns 9 through 12

-2.80917027670684   4.49524067860350  -3.52103855354539  -7.87996008113431
 1.10436192374786  -0.75430208080295   3.57750484830174   2.15109723190010
 0.57792662993850  -4.24950641271152   2.71752440158303   2.90011089067157
 3.29202931182155  -3.78643567441964   6.15786515520179   3.54609009021479
-0.05521626329838  -2.43082953900282  -2.40268468385147  -1.89673904423745
 1.85219726366423   3.05065135460699  -3.10783986924079  -0.85346280807182
-7.29662813475638  -0.93762358626709   6.36199769209926   6.91710281913630
-1.85849141290655  -3.21364576272254  -2.20192162924585   1.89641964906194
15.31908761456532  -2.44512995368064  -1.93501853259340  -3.05911080290154
-2.44512995368064  19.44708718410749  -2.40514484831925  -6.37098563710744
-1.93501853259340  -2.40514484831925  19.30028747813050   9.27582299968460
-3.05911080290154  -6.37098563710744   9.27582299968460  20.72087175916059
-8.83303092955187  -0.43885426723912   5.07190293805150   3.92800372721324
 1.18154455303509   0.94372571020949  -4.16622801226983  -1.26616230824007
 0.32395599527496   1.24869106166008   5.04902937673208   9.81059790448825
 4.32860300715391  -1.30113353291271   2.43705352037408  -2.00333197999432
 2.91309522604263   5.05344205364850  -5.22306371835993 -10.38425712770274
 3.20029605264920   1.03281224227556  -1.74284279027901   5.09571414430093
-1.29984264363845   2.10106992889620  -3.88282213813449  -4.57936806239257
 5.35027212388616   0.80284270135759  -3.63448710105253   0.34165786764706

Columns 13 through 16

 4.80002265904681   1.66717040639530  -3.80725360173308   1.72471100875360
```

```
 -1.59105805368087   1.14545917632083   7.05813185145915   3.61060094661089
  1.29356820990489  -4.36092392862978   6.02825993049226  -3.33614466357950
  7.38470429023030  -1.91712736186076   3.20436113613495   4.75588750380413
 -2.07065108483332   7.55480336539587  -0.00596942006038  -4.25217716278094
  7.49997231296922  -1.32008757986820  -2.96394598497866  -4.10668478685671
 13.00254144974241  -4.69947306080261   3.06920670625717  -0.82559016994131
  1.58294777412672  -3.92886713309874  -1.50690283174512   9.70423242893581
 -8.83303092955187   1.18154455303509   0.32395599527496   4.32860300715391
 -0.43885426723912   0.94372571020949   1.24869106166008  -1.30113353291271
  5.07190293805150  -4.16622801226983   5.04902937673208   2.43705352037408
  3.92800372721324  -1.26616230824007   9.81059790448825  -2.00333197999432
 24.12522802600309  -3.41608222510974  -0.50117706718829   2.76400060312202
 -3.41608222510974  19.83312260680773   5.01613937346091  -3.63929156555895
 -0.50117706718829   5.01613937346091  21.07261881401281  -1.66159180249801
  2.76400060312202  -3.63929156555895  -1.66159180249801  17.04227540736978
 -6.95779821614710   1.87825028566875  -3.96396867475761   1.09177466684760
 -0.16965881003028  -3.29430437293934   5.12084378782793  -0.89842450071235
  6.40320120125264   1.94119135446451   2.74734505765918  -4.24963905501730
  0.12756734106068  -2.83122105320255  -4.77783909060213   5.02032919340751

 Columns 17 through 20

  6.81572444425526  -1.79094569115684   7.19510238432702  -3.01670335976680
  4.91771989320600   2.21104150087101   5.88149749387292  -5.40505279830859
 -5.15723268330519   1.32024529078390  -0.06753554887202  -3.46917850882857
 -6.22541483185501   5.91887377065978  -1.18611708282151   0.01551779178417
  0.96514731283814  -3.83184464730678   1.76472662201430  -2.86087258885245
  0.97727344005285   6.55765762103624  10.64116320585426   1.79706122754855
 -8.21499784888196  -0.38964057450520   3.93486714955786  -3.13443959529720
 -1.70909340333592   1.21426787642390  -1.87052606954003   1.59561888208891
  2.91309522604263   3.20029605264920  -1.29984264363845   5.35027212388616
  5.05344205364850   1.03281224227556   2.10106992889620   0.80284270135759
 -5.22306371835993  -1.74284279027901  -3.88282213813449  -3.63448710105253
-10.38425712770274   5.09571414430093  -4.57936806239257   0.34165786764706
 -6.95779821614710  -0.16965881003028   6.40320120125264   0.12756734106068
  1.87825028566875  -3.29430437293934   1.94119135446451  -2.83122105320255
 -3.96396867475761   5.12084378782793   2.74734505765918  -4.77783909060213
  1.09177466684760  -0.89842450071235  -4.24963905501730   5.02032919340751
 29.82838754136444  -1.26458282650169   2.20104361650244  -4.42658610991927
 -1.26458282650169  17.84150006836002   3.35641843894882  -3.22307705469232
  2.20104361650244   3.35641843894882  16.62589584989890  -3.80128969490291
```

```
  -4.42658610991927  -3.22307705469232  -3.80128969490291  13.69197701790477

Iteration: 1
 Sqrt of ssq = 3.9555e+001

Iteration: 2
 Sqrt of ssq = 1.3845e+001

Iteration:3
 Sqrt of ssq = 3.3652e+000

Iteration:4
Sqrt of ssq = 7.3924e-001

Iteration:5
Sqrt of ssq = 1.5517e-001

Iteration: 6
Sqrt of ssq = 5.1515e-003

Iteration: 7
 Sqrt of ssq = 9.2725e-006

Iteration: 8
 Sqrt of ssq = 2.0652e-015


  [eig(A)      eig(diag L1)]
  5.9679e-004  5.9679e-004
  5.2900e-002  5.2900e-002
  3.5223e-001  3.5223e-001
  1.0328e+000  1.0328e+000
  2.6762e+000  2.6762e+000
  2.9316e+000  2.9316e+000
  5.1808e+000  5.1808e+000
  6.5284e+000  6.5284e+000
  8.3462e+000  8.3462e+000
  1.2623e+001  1.2623e+001
  1.4645e+001  1.4645e+001
  1.7913e+001  1.7913e+001
  2.0490e+001  2.0490e+001
```

```
  2.1185e+001   2.1185e+001
  2.5473e+001   2.5473e+001
  3.2831e+001   3.2831e+001
  3.7543e+001   3.7543e+001
  4.2013e+001   4.2013e+001
  5.5268e+001   5.5268e+001
  6.7238e+001   6.7238e+001
```

Comments: By looking into all 8 iterations we notice that sum of squares of offdiagonal entries is reducing very fast. In each run they are converging quadratically.

Example 21. *Use Matlab codes for Jacobi's method to find eigenvalues of a symmetric matrix of order* 100×100 *by converting into to diagonal form.*

Here we present only final results to for simplicity.

```
Iteration:1
 Sqrt of ssq = 4.8436e+002

Iteration: 2
Sqrt of ssq = 1.7927e+002

Iteration:3
Sqrt of ssq = 5.2088e+001

Iteration:4
Sqrt of ssq = 1.5027e+001

Iteration: 5
Sqrt of ssq = 3.7894e+000

Iteration: 6
Sqrt of ssq = 5.9729e-001

iteration: 7
 Sqrt of ssq = 6.3850e-002

Iteration: 8
Sqrt of ssq =6.2185e-004
```

```
Iteration:9
 Sqrt of ssq = 9.0480e-007

Iteration: 10
 Sqrt of ssq = 1.0080e-013


  [eig(A)     eig(diag A)]
  1.0692e-002  1.0692e-002
  7.1967e-002  7.1967e-002
  9.4433e-002  9.4433e-002
  2.7659e-001  2.7659e-001
  3.9153e-001  3.9153e-001
  6.4258e-001  6.4258e-001
  9.2235e-001  9.2235e-001
  1.0307e+000  1.0307e+000
  1.7843e+000  1.7843e+000
  2.1695e+000  2.1695e+000
  2.5173e+000  2.5173e+000
  2.8806e+000  2.8806e+000
  3.4883e+000  3.4883e+000
  3.6898e+000  3.6898e+000
  4.9216e+000  4.9216e+000
  5.3648e+000  5.3648e+000
  6.4946e+000  6.4946e+000
  6.8548e+000  6.8548e+000
  7.2262e+000  7.2262e+000
  8.9053e+000  8.9053e+000
  1.0184e+001  1.0184e+001
  1.0600e+001  1.0600e+001
  1.1214e+001  1.1214e+001
  1.3334e+001  1.3334e+001
  1.5464e+001  1.5464e+001
  1.6300e+001  1.6300e+001
  1.7052e+001  1.7052e+001
  1.9204e+001  1.9204e+001
  2.0544e+001  2.0544e+001
  2.2796e+001  2.2796e+001
  2.3526e+001  2.3526e+001
  2.4927e+001  2.4927e+001
```

```
2.6809e+001  2.6809e+001
2.9533e+001  2.9533e+001
3.0385e+001  3.0385e+001
3.0883e+001  3.0883e+001
3.2936e+001  3.2936e+001
3.3861e+001  3.3861e+001
3.5061e+001  3.5061e+001
3.7552e+001  3.7552e+001
4.0822e+001  4.0822e+001
4.1795e+001  4.1795e+001
4.2036e+001  4.2036e+001
4.4028e+001  4.4028e+001
4.8187e+001  4.8187e+001
5.1844e+001  5.1844e+001
5.5792e+001  5.5792e+001
5.9542e+001  5.9542e+001
6.1548e+001  6.1548e+001
6.2222e+001  6.2222e+001
6.4556e+001  6.4556e+001
6.9297e+001  6.9297e+001
7.1777e+001  7.1777e+001
7.4904e+001  7.4904e+001
7.9578e+001  7.9578e+001
8.1615e+001  8.1615e+001
8.2583e+001  8.2583e+001
8.7462e+001  8.7462e+001
9.1724e+001  9.1724e+001
9.3655e+001  9.3655e+001
9.6224e+001  9.6224e+001
9.8530e+001  9.8530e+001
1.0197e+002  1.0197e+002
1.0879e+002  1.0879e+002
1.0983e+002  1.0983e+002
1.1618e+002  1.1618e+002
1.2290e+002  1.2290e+002
1.2651e+002  1.2651e+002
1.2997e+002  1.2997e+002
1.3650e+002  1.3650e+002
1.3939e+002  1.3939e+002
1.4081e+002  1.4081e+002
1.4820e+002  1.4820e+002
```

```
1.5583e+002  1.5583e+002
1.6110e+002  1.6110e+002
1.6411e+002  1.6411e+002
1.7380e+002  1.7380e+002
1.7706e+002  1.7706e+002
1.8696e+002  1.8696e+002
1.9173e+002  1.9173e+002
1.9468e+002  1.9468e+002
2.0102e+002  2.0102e+002
2.0810e+002  2.0810e+002
2.1546e+002  2.1546e+002
2.2183e+002  2.2183e+002
2.3335e+002  2.3335e+002
2.4015e+002  2.4015e+002
2.4397e+002  2.4397e+002
2.5792e+002  2.5792e+002
2.6546e+002  2.6546e+002
2.6987e+002  2.6987e+002
2.7875e+002  2.7875e+002
2.8824e+002  2.8824e+002
2.8883e+002  2.8883e+002
3.0179e+002  3.0179e+002
3.0708e+002  3.0708e+002
3.2280e+002  3.2280e+002
3.4641e+002  3.4641e+002
3.5484e+002  3.5484e+002
3.6748e+002  3.6748e+002
```

Comments: This a nice example in which a matrix of order 100×100 is diagonalized in just 10 iterations. Final results are very accurate. Thus the results are of Jacobi's method to find eigenvalues are more accurate and reliable.

4.3 Comparison of Cholesky Iterative Method and Jacobi's Method

Both methods are convergent. Cholesky method converges linearly while Jacobi method converges quadratically. Here we present an example to see the rate of convergent of both methods by applying on same matrix and will see the number of iterations and final results for eigenvalues.

Example 22. *Find eigenvalues of any square matrix A by using both Jacobi and Cholesky Methods and check number of iterations for each method.*

```
% codes for  jac_chol_iterat_compare.m

 n=input ('enter value of n:  ');
   A=rand(n);
    A=A*A';
                           % A is symmetric
    T=A;
 disp('The input array A is:')
  disp(A)
for i=1:100                     %max iterations
    J=jac_sweep(T);             % Jacobi rotations
      disp('iteration:  ')
      disp(i)
      disp('J after this iteration is:')
      disp( J)

    if isDiag(J,n)==0   % checking off-diagonal entries each time
        T=J;            % diagonal matrix
        continue
    else
        break
    end
end

 % now apply chol_tri to the same matrix to compare the results

 A=hess(A);        % make A tridiagonal matrix
 [T,p]=chol(A);    % permutation matrix
if p>0
```

```
    'Not a positive definite Matrix'
    return
end
 T=A;
for i=1:100              %max Iteration
    L1=chol(T);
    A1=L1*L1';          % set each time
      disp('iteration:  ')
      disp(i)
      disp('A1 after this iteration is:')
      disp( A1)
    if isDiag(A1,n)==0
        T=A1;          % Diagonal matrix
        continue
    else
        break
    end
end

disp('Jacobi:')         % final results
 disp( J)

disp('Cholesky')
 disp( A1)

 disp('     eigen(A)     eig using    % compare eigenvalues
Jacobi eig using Cholesky')
disp(sort([eig(A) diag(J) diag(A1)]))
```

Example 23. *Comparison Results for a randam matrix of order* (5×5).

```
enter value of n:  5

The input array A is:           % A is symmetric
    2.0869    1.6316    1.5179    1.9186    1.3259
    1.6316    2.2654    1.3120    1.6595    1.5064
    1.5179    1.3120    1.7033    1.4225    0.9348
    1.9186    1.6595    1.4225    2.2551    1.8170
```

```
    1.3259    1.5064    0.9348    1.8170    1.6679

iteration: 1

J after this iteration is:
    1.7158   -0.1753    0.3326   -0.1829   -2.8171
   -0.1753    0.6410    0.0633   -0.0613   -0.2495
    0.3326    0.0633    0.6124   -0.0025   -0.1216
   -0.1829   -0.0613   -0.0025    0.1682    0.0000
   -2.8171   -0.2495   -0.1216    0.0000    6.8412


iteration: 7        % after 7 iterations diagonal J

J after this iteration is:
    0.3024   -0.0000    0.0000   -0.0000   -0.0000
   -0.0000    0.6830   -0.0000    0.0000   -0.0000
    0.0000   -0.0000    0.8940   -0.0000   -0.0000
   -0.0000    0.0000   -0.0000    8.0991   -0.0000
   -0.0000   -0.0000   -0.0000   -0.0000    0.0001

  % Results of Cholesky iterative method

Iteration: 1

A1 after this iteration is:
    0.6610    0.1071         0         0         0
    0.1071    0.3983   -0.2028         0         0
         0   -0.2028    2.3323    2.9519         0
         0         0    2.9519    6.5866   -0.0243
         0         0         0   -0.0243    0.0004


%Results  after 71 iterations.

Iteration: 71

A1 after this iteration is:
    8.0991    0.0000         0         0         0
    0.0000    0.8940   -0.0001         0         0
         0   -0.0001    0.6830    0.0000         0
```

```
         0         0    0.0000    0.3024   -0.0000
         0         0         0   -0.0000    0.0001
```

```
eigen(A) eig using Jacobi  eig using Cholesky
   0.0001    0.0001          0.0001
   0.3024    0.3024          0.3024
   0.6830    0.6830          0.6830
   0.8940    0.8940          0.8940
   8.0991    8.0991          8.0991
```

Comments: We tested both methods on same random matrix of size (5×5). Maximum number of iterations are fixed at 100. Jacobi's method converges very fast and after 7 iterations diagonal matrix J is obtained. Where as Cholesky method needed 71 iterations to converge diagonal form. Eigenvalues obtained are exactly same by using Cholesky iterative method and Jacobi's method. In a few iterations Jacobi showed better results than Cholesky. Jacobi's method uses a lot more flops per iteration so it is not surprising that it converges in fewer iterations. Finally we come to conclusion that Jacobi's method is our better choice to compute eigenvalues of a real symmetric matrix.

Chapter 5

Applying Methods to compute weights and nodes of Gaussian quadrature

5.1 Introduction

As already discussed in chapter 1 that Gaussian quadrature is more accurate method for numerical integration. It calculated nodes and weights and then use these values for computing area under the integral, which is tedious. Using three-term recurrence relation we find connection between nodes and weights with eigenvalues and eigenvectors. This have transformed the numerical integration problem into eigenvalue problem.
In previous discussion we concluded with conclusion that Jacobi method is better choice for computation of eigenvalues. Here we are presenting Matlab codes for Jacobi rotation method and Gaussian quadrature method for computing nodes and weights. At the end there is a comparison of calculated nodes and weights by this algorithm and standard nodes and weights available online.

5.2 Computing Nodes and Weights by using Jacobi-Gaussian Matlab codes

Problem: Let us consider definite integral

$$I = \int_0^1 \exp(x)\mathrm{dx}$$

Its analytic value is 1.71828182845905.
By using relation of nodes and weights as discussed in chapter 1, we calculate λ-the eigenvalue and V- the eigen vector. These values gave nodes and weights very easily.
Here are Matlab codes for this algorithm.

```
format('long');
 n=input ('enter value of n:  '); % the size of tridiagonal matrix T
 alpha=zeros(1,n);       % entries of matrix T
beta=zeros(1,n);

for i=1:n
    beta(i)=1/2*(1-(2*i)^(-2))^-.5;  % calculating diagonal entry beta
end

t=zeros(n);
 for i=1:n-1
    t(i,i+1)=beta(i);      % to calculate matrix T
    t(i+1,i)=beta(i);
end

lambda=eig(t);        % calculating eigenvalues which are also node values
 [V D]=eig(t);        % calculating eigenvectors

for i=1:n
    w(i)=2*((V(1,i)^2));  % calculating weights from eigenvectors
end
```

These calculated value are presented in following table.

5.3 Comparison of Calculated and Standard Nodes of Gaussian quadrature

Comparison of Results		
n-values	Calculated Nodes	Standard Nodes
2	± 0.57735026918963	± 0.57735026918962576450914878
4	±0.33998104358486	±0.33998104358485626480266583
	± 0.86113631159405	±0.86113631159405257522394649
6	± 0.23861918608320	± 0.23861918608319690863050172
	±0.66120938646626	± 0.66120938646626451366139960
	±0.93246951420315	±0.93246951420315202781230155
8	± 0.18343464249565	±0.18343464249564980493947614
	± 0.52553240991633	±0.52553240991632898581773903
	±0.79666647741363	±0.79666647741362673959155390
	± 0.96028985649754	±0.96028985649753623168356091
10	± 0.14887433898163	± 0.14887433898163121088482605
	± 0.43339539412925	± 0.43339539412924719079926594
	± 0.67940956829902	± 0.67940956829902440623432744
	± 0.86506336668898	± 0.86506336668898451073209670
	± 0.97390652851717	± 0.97390652851717172007796401
12	± 0.12523340851147	± 0.12523340851146891547244140
	±0.36783149899818	± 0.36783149899818019375269151
	± 0.58731795428662	± 0.58731795428661744729670243
	±0.76990267419430	± 0.76990267419430468703689387
	±0.90411725637047	± 0.90411725637047485667846590
	±0.98156063424672	± 0.98156063424671925069054910
14	± 0.10805494870734	± 0.10805494870734366206624470
	± 0.31911236892789	± 0.31911236892788976043567180
	±0.51524863635815	± 0.51524863635815409196529072
	±0.68729290481169	± 0.68729290481168547014801980
	±0.82720131506977	±0.82720131506976499318979470
	± 0.92843488366357	± 0.92843488366357351733639110
	±0.98628380869681	± 0.98628380869681233884159732
16	± 0.09501250983764	± 0.09501250983763744018531933
	±0.28160355077926	± 0.28160355077925891323046052
	±0.45801677765723	± 0.45801677765722738634241947
	±0.61787624440264	± 0.61787624440264374844667183
	±0.75540440835500	± 0.75540440835500303389510125
	±0.86563120238783	± 0.86563120238783174388046792
	±0.94457502307323	± 0.94457502307323257607798841
	±0.98940093499165	± 0.98940093499164993259615420

Comparison of Results		
n-values	Calculated Nodes	Standard Nodes
18	±0.08477501304174	± 0.08477501304173530124226 19
	±0.25188622569151	± 0.25188622569150550958897 29
	±0.41175116146284	± 0.41175116146284264603593 18
	±0.55977083107395	± 0.55977083107394753460787 15
	±0.69168704306035	± 0.69168704306035320787489 11
	±0.80370495897252	± 0.80370495897252311568241 75
	± 0.89260246649756	± 0.89260246649755573920606 06
	±0.95582394957140	± 0.95582394957139775518119 59
	± 0.99156516842093	± 0.99156516842093094673001 60
20	± 0.07652652113350	± 0.07652652113349733375464 04
	±0.22778585114164	± 0.22778585114164507808049 62
	±0.37370608871542	±0.37370608871541956067254 82
	±0.51086700195083	± 0.51086700195082709800436 41
	±0.63605368072652	±0.63605368072651502545283 67
	±0.74633190646015	±0.74633190646015079261430 51
	±0.83911697182222	± 0.83911697182221882339452 91
	±0.91223442825133	± 0.91223442825132590586775 24
	±0.96397192727791	± 0.96397192727791379126766 61
	±0.99312859918510	± 0.99312859918509492478612 24
64	±0.02435029266342	± 0.02435029266342443250895 58
	±0.07299312178780	± 0.07299312178779903944954 29
	±0.12146281929612	± 0.12146281929612055447037 65
	±0.16964442042399	± 0.16964442042399281803731 36
	±0.21742364374001	± 0.21742364374000708414964 87
	±0.26468716220877	± 0.26468716220876741637396 42
	±0.31132287199021	± 0.31132287199021095615751 27
	±0.35722015833767	± 0.35722015833766811595044 26
	±0.40227015796399	± 0.40227015796399160369576 68
	±0.44636601725346	± 0.44636601725346408798494 77
	±0.48940314570705	± 0.48940314570705295747852 63
	±0.53127946401989	± 0.53127946401989455465801 39
	±0.57189564620263	± 0.57189564620263403428387 81
	±0.61115535517239	± 0.61115535517239325024885 30
	±0.64896547125466	± 0.64896547125465733985776 12
	±0.68523631305423	± 0.68523631305423324256355 84

Comparison of Results		
n-values	Calculated Nodes	Standard Nodes
64	±0.71988185017161	± 0.71988185017161 6108268489402
	±0.75281990726053	± 0.75281990726053 18966118638
	±0.78397235894334	± 0.78397235894334 14076102205
	±0.81326531512280	± 0.81326531512279 75597419233
	±0.84062929625258	± 0.84062929625258 03627516915
	±0.86599939815409	± 0.86599939815409 28197607834
	±0.88931544599511	± 0.88931544599511 41058534040
	±0.91052213707850	± 0.91052213707850 28057563807
	±0.92956917213194	± 0.92956917213194 39575821490 2
	±0.94641137485840	± 0.94641137485840 28160624815
	±0.96100879965205	± 0.96100879965205 37189186141
	±0.97332682778991	± 0.97332682778991 09637418535
	±0.98333625388463	± 0.98333625388462 59569312993
	±0.99101337147674	± 0.99101337147674 43207393824
	±0.99634011677196	± 0.99634011677195 52793469245
	±0.99930504173577	± 0.99930504173577 21394569056
100	±0.01562898442154	± 0.01562898442154 30828722167
	±0.04687168242159	± 0.04687168242159 16316149239
	±0.07806858281344	± 0.07806858281343 66366948174
	±0.10918920358006	± 0.10918920358006 11150034260
	±0.14020313723611	± 0.14020313723611 39732075146
	±0.17108008053860	± 0.17108008053860 32748875324
	±0.20178986409574	± 0.20178986409573 59972360489
	±0.23230248184497	± 0.23230248184497 39696495100
	±0.26258812037150	± 0.26258812037150 34791689293
	±0.29261718803847	± 0.29261718803847 19647375559
	±0.32236034390053	± 0.32236034390052 91517224766
	±0.35178852637242	± 0.35178852637242 17209723438
	±0.38087298162463	± 0.38087298162462 99567633625
	±0.40958529167830	± 0.40958529167830 15425288684
	±0.43789740217203	± 0.43789740217203 15131089780
	±0.46578164977336	± 0.46578164977336 80422492166
	±0.49321078920819	± 0.49321078920819 09335693088
	±0.52015801988176	± 0.52015801988176 30566468157
	±0.54659701206509	± 0.54659701206509 41674679943
	±0.57250193262138	± 0.57250193262138 11913168704
	±0.59784747024718	± 0.59784747024717 87212648065
	±0.62260886020371	±0.62260886020370 77716041908

Comparison of Results		
n-values	Calculated Nodes	Standard Nodes
100	±0.64676190851413	± 0.64676190851412927983263 03
	±0.67028301560314	± 0.67028301560314101580258 70
	±0.69314919935580	± 0.69314919935580196594864 79
	±0.71533811757306	± 0.71533811757305644645996 71
	±0.73682808980202	± 0.73682808980202070551242 77
	±0.75759811851971	± 0.75759811851970717603566 80
	±0.77762790964950	±0.77762790964949547562755 14
	±0.79689789239031	±0.79689789239031447638957 29
	±0.81538923833918	±0.81538923833917625439398 88
	±0.83308387988840	±0.83308387988840082354291 58
	±0.84996452787959	±0.84996452787959128429336 26
	±0.86601468849716	±0.86601468849716462341074 00
	±0.88121867938502	±0.88121867938501841557331 68
	±0.89556164497073	±0.89556164497072698669852 10
	±0.90902957098253	±0.90902957098252969046712 63
	±0.92160929814533	±0.92160929814533395266695 13
	±0.93328853504308	±0.93328853504307954592433 37
	±0.94405587013626	±0.94405587013625597796277 47
	±0.95390078292549	±0.95390078292549174284933 69
	±0.96281365425582	±0.96281365425581552729365 93
	±0.97078577576371	±0.97078577576370633193089 79
	±0.97780935848692	±0.97780935848691828855378 11
	±0.98387754070606	±0.98387754070605701549610 02
	±0.98898439524299	±0.98898439524299174800441 87
	±0.99312493703744	±0.99312493703744345965200 99
	±0.99629513473313	±0.99629513473312514918613 17
	±0.99849195063960	±0.99849195063959581840016 34
	±0.99971372677344	±0.99971372677344123367822 85

Comments: The calculated results are displayed in maximum length of 14 digits as per capacity of Matlab program. The results are matched with standard values calculated by "$NumericalIntegration_{P}avelHoloborodko$"and can be found at "*www.holoborodko.com/pavel/*". Calculated values are correct up to 14-digits length.

5.4 Comparison of Calculated and Standard Weights of Gaussian quadrature

Calculated values		
n-values	Calculated Weights	Standard Weights
2	1.00000000000000	1.0000000000000000000000
4	0.65214515486255	0.652145154862546142626936
	0.34785484513745	0.347854845137453857373063
6	0.17132449237917	0.171324492379170345040296
	0.36076157304814	0.360761573048138607569833
	0.46791393457269	0.467913934572691047389870
8	0.10122853629038	0.101228536290376259152531
	0.22238103445337	0.222381034453374470544356
	0.31370664587789	0.313706645877887287337962
	0.36268378337836	0.362683783378361982965150
10	0.06667134430869	0.066671344308688137593568
	0.14945134915058	0.149451349150580593145776
	0.21908636251598	0.219086362515982043995534
	0.26926671931000	0.269266719309996355091226
	0.29552422471475	0.295524224714752870173893
12	0.04717533638651	0.047175336386511827194616
	0.10693932599532	0.106939325995318430960254
	0.16007832854335	0.160078328543346226334652
	0.20316742672307	0.203167426723065921749064
	0.23349253653835	0.233492536538354808760849
	0.24914704581340	0.249147045813402785000562
14	0.03511946033175	0.035119460331751863031832
	0.08015808715976	0.080158087159760209805633
	0.12151857068790	0.121518570687903184689414
	0.15720316715819	0.157203167158193534569601
	0.18553839747794	0.185538397477937813741716
	0.20519846372130	0.205198463721295603965924
	0.21526385346316	0.215263853463157790195876
16	0.02715245941175	0.027152459411754094851780
	0.06225352393865	0.062253523938647892862843
	0.09515851168249	0.095158511682492784809925
	0.12462897125553	0.124628971255533872052476
	0.14959598881658	0.149595988816576732081501
	0.16915651939500	0.169156519395002538189312
	0.18260341504492	0.182603415044923588866763
	0.18945061045507	0.189450610455068496285396

Calculated values		
n-values	Calculated Weights	Standard Weights
18	0.02161601352648	0.021616013526483310313342 7
	0.04971454889497	0.049714548894969796453334 9
	0.07642573025489	0.076425730254889056529129 7
	0.10094204410629	0.100942044106287165562814 0
	0.12255520671148	0.122555206711478460184519 1
	0.14064291467065	0.140642914670650651204731 3
	0.15468467512627	0.154684675126265244925418 0
	0.16427648374583	0.164276483745832722986053 8
	0.16914238296314	0.169142382963143591840656 5
20	0.01761400713915	0.017614007139152118311862 0
	0.04060142980039	0.040601429800386941331040 0
	0.06267204833411	0.062672048334109063569506 5
	0.08327674157670	0.083276741576704748724758 1
	0.10193011981724	0.101930119817240435036750 1
	0.11819453196152	0.118194531961518417312377 4
	0.13168863844918	0.131688638449176626898494 5
	0.14209610931838	0.142096109318382051329298 3
	0.14917298647260	0.149172986472603746787828 7
	0.15275338713072	0.152753387130725850698084 3
64	0.00178328072170	0.001783280721696432947296 1
	0.00414703326056	0.004147033260562467635287 5
	0.00650445796898	0.006504457968978362856117 4
	0.00884675982636	0.008846759826363947723030 9
	0.01116813946013	0.011168139460131128818590 5
	0.01346304789672	0.013463047896718642598060 8
	0.01572603047602	0.015726030476024719321966 0
	0.01795171577570	0.017951715775697343085045 3
	0.02013482315353	0.020134823153530209372340 3
	0.02227017380838	0.022270173808383254159298 3
	0.02435270256871	0.024352702568710873338177 6
	0.02637746971505	0.026377469715054658671691 8
	0.02833967261426	0.028339672614259483227511 3
	0.03023465707240	0.030234657072402478867974 1
	0.03205792835485	0.032057928354851553585467 5
	0.03380516183714	0.033805161837141609391565 5

Calculated values		
n-values	Calculated Weights	Standard Weights
64	0.03547221325688	0.03547221325688238381069 31
	0.03705512854024	0.03705512854024004604041 51
	0.03855015317862	0.03855015317861562912896 25
	0.03995374113272	0.03995374113272034138665 69
	0.04126256324262	0.04126256324262352861015 63
	0.04247351512365	0.04247351512365358900733 98
	0.04358372452932	0.04358372452932345337682 79
	0.04459055816376	0.04459055816375656306013 47
	0.04549162792742	0.04549162792741814447977 10
	0.04628479658131	0.04628479658131441729595 32
	0.04696818281621	0.04696818281621001732532 63
	0.04754016571483	0.04754016571483030866228 22
	0.04799938859646	0.04799938859645830772812 62
	0.04834476223480	0.04834476223480295716976 95
	0.04857546744150	0.04857546744150342693479 91
	0.04869095700914	0.04869095700913972038336 54
100	0.00073463449051	0.00073463449050567173040 63
	0.00170939265352	0.00170939265351810523952 94
	0.00268392537155	0.00268392537155348241943 96
	0.00365596120133	0.00365596120132637518234 25
	0.00462445006342	0.00462445006342211935109 58
	0.00558842800387	0.00558842800386551515721 19
	0.00654694845085	0.00654694845084532276415 21
	0.00749907325547	0.00749907325546471157882 87
	0.00844387146967	0.00844387146966897140262 08
	0.00938041965369	0.00938041965369445795141 82
	0.01030780257487	0.01030780257486896958578 21
	0.01122511402319	0.01122511402318597711722 16
	0.01213145766298	0.01213145766297949740774 48
	0.01302594789297	0.01302594789297154228555 86
	0.01390771070372	0.01390771070371877268795 41
	0.01477588452744	0.01477588452744130176888 00
	0.01562962107755	0.01562962107754600272393 69
	0.01646808617614	0.01646808617614521264310 50
	0.01729046056832	0.01729046056832358243934 42
	0.01809594072213	0.01809594072212811666439 08
	0.01888373961338	0.01888373961337490455294 12
	0.01965308749444	0.01965308749443530586538 15

Calculated values		
n-values	Calculated Weights	Standard Weights
100	0.02040323264621	0.020403232646209432766838 9
	0.02113344211253	0.021133442112527641542672 3
	0.02184300241625	0.021843002416247386313953 7
	0.02253122025634	0.022531220256336272701797 0
	0.02319742318525	0.023197423185254121622488 9
	0.02384096026597	0.023840960265968205962560 4
	0.02446120270796	0.024461202707957052719975 0
	0.02505754448158	0.025057544481579589703764 2
	0.02562940291021	0.025629402910208116075642 0
	0.02617621923955	0.026176219239545676342308 7
	0.02669745918357	0.026697459183570962660384 7
	0.02719261344658	0.027192613446576880136491 6
	0.02766119822079	0.027661198220792388294204 2
	0.02810275565910	0.028102755659101173317648 3
	0.02851685432240	0.028516854322395097990936 8
	0.02890308960113	0.028903089601125203134876 2
	0.02926108411064	0.029261084110638276620119 0
	0.02959048805991	0.029590488059912642511754 5
	0.02989097959333	0.029890979593332830916836 8
	0.03016226510517	0.030162265105169144919068 7
	0.03040407952645	0.030404079526454820016507 9
	0.03061618658398	0.030616186583980448496459 4
	0.03079837903115	0.030798379031152590427713 9
	0.03095047885049	0.030950478850490988234063 5
	0.03107233742757	0.031072337427566516587810 2
	0.03116383569621	0.031163835696209906783818 3
	0.03122488425485	0.031224884254849357732376 5
	0.03125542345386	0.031255423453863356947642 5

Comments: The calculated weights are displayed in maximum length of 14 digits as per capacity of Matlab 7.1 program. The results are matched with standard values obtained by "$NumericalIntegration_{P}avelHoloborodko$" and can be found at "*www.holoborodko.com/pavel/*". Calculated values are correct up to 14-digits length.

5.5 Conclusion and Future work

In this project we solved a problem of numerical integration by using nodes and weights of Gaussian quadrature rule. This method is good but takes lot a of time and labor. To overcome this drawback we used the connection between nodes and weights of Gaussian quadratures with eigenvalue problem. We notice that computing nodes and weights by Jacobi method is very easy but as Matlab [7.1 version] read maximum 14 digits only so we could not get exact answer although it is very close to exact.
For future work we will devise an implementation of our method using high precision and will compare the basic method with the high precision method. Also in future work we will use norms to analyse the accuracy of the numerical methods we have used to compute eigenvalues and eigen vectors.

Bibliography

[1] James F. Epperson, *An introduction to numerical methods and analysis*, John Wiley and Sons,Inc, 2002.

[2] David Bau III Lloyd N. Trefethen, *Numerical linear algebra*, Society for Industrial and applied Mathematics, 1997.

[3] John H. Mathews and Kurtis K.Fink, *Numerical methods using matlab*, Prentice-Hall Inc, 2004.

[4] Carl D. Meyer, *Matrix analysis and applied linear algebra*, Society for Industrial and Applied Mathematics, 2000.

[5] Madhumangal Pal, *Numerical analysis for scientists and engineers*, Alpha Science International Ltd, 2007.