CELSO JOSÉ CORDEIRO

NEW ALGORITHMS TO S&T DECOMPOSITION

CURITIBA

2002

CELSO JOSÉ CORDEIRO

NEW ALGORITHMS TO S&T DECOMPOSITION

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciências, pelo Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Setores de Ciências Exatas e de Tecnologia da Universidade Federal do Paraná.

Orientador: Prof. Dr. Jin Yun Yuan.

CURITIBA

2002

CELSO JOSÉ CORDEIRO

NEW ALGORITHMS TO S&T DECOMPOSITION

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciências, pelo Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Setores de Ciências Exatas e de Tecnologia da Universidade Federal do Paraná.

Orientador: Prof. Dr. Jin Yun Yuan.

CURITIBA

2002

TERMO DE APROVAÇÃO

CELSO JOSÉ CORDEIRO

NEW ALGORITHMS TO S&T DECOMPOSITION

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre em Ciências no Curso de Pós-Graduação em Métodos Numéricos em Engenharia, Setores de Ciências Exatas e de Tecnologia da Universidade Federal do Paraná, pela seguinte banca examinadora:

Coordenador do Curso:

Prof. Dra. Maria Teresinha Arns Steiner.

Orientador:

Prof. Dr. Jin Yun Yuan.
Depto. de Matemática - UFPR

Prof. Dr. Mário Cesar Zambaldi
Depto. de Matemática - UFSC

Prof. Dr. Luiz Carlos Matioli
Depto. de Matemática - UFPR

Curitiba, junho de 2002.

For my beloved children, Henrique, Aliny, Victor,
to the memory of my father, Francisco,
and to may mother Inês.
That God always guards them.

Acknowledgements

I am grateful my mother, Inês José Maria Rocha Cordeiro and to my children Henrique Cordeiro, Aliny Xavier Cordeiro, and Victor Cordeiro for their precious collaboration and understanding for all the working hours and study that I had to be absence of their conviviality.

To Prof. Dr. Jin Yun Yuan, my supervisor, for the proposal of the theme, collaboration, orientation and great readiness during the course and in the progresses of the works.

To Prof. Dr. Yin Jiahong, for his great help in the verifications of the obtained results and suggestions presented in the development of this dissertation.

To the colleagues, Cosmo D. Santiago who gave me great help during this work, and also to Ricardo A. D. Zanardini, Waléria Cecílio and Ingrid Milléo, with whom I shared many hours of studies, seminars, and discussions, which is very profitable for me.

To Prof. Dr. Marli Cardia for her great competence in teaching which leads me to have the willing of learning more and more on Linear Algebra and Optimization.

To the Center of Studies in Civil Engineering - CESEC, where we are offered wonderful infrastructure for the accomplishment of the works, and finally to everybody that collaborated direct or indirectly to the development of this work.

One night a man had a dream. He dreamed that he was walking along the beach with the Lord, and he began to see scenes from his life. For each scene, he noticed two sets of footprints in the sand, one belonging to him, and the other to the Lord.

He notice that sometimes along the path of his life there was only one set of footprints, and that it happened exactly when he was having some kind of trouble.

This really bothered him and he asked the Lord about it, "Lord, why did you leave me when I needed you most? I can't understand why you left me in times of trouble. Exactly when I needed you near me, you were not there".

The Lord answered him, "My son, my precious child, I love you and I never left you. When you were in trouble, during the hard times of your life, when you see only one set of footprints, it was then that I carried you".

(Author unknown.)

Contents

List of Tables

Resumo

Este trabalho tem como maior objetivo fazer uma nova análise e propor novos algoritmos para a Decomposição ST (Symmetric Triangular Decomposition) desenvolvido por Golub e Yuan em 2000.

A decomposição ST decompõem uma matriz A quadrada e não singular (em geral, não simétrica) em um produto de matrizes $A = ST$ onde T é uma matriz triangular e S é uma matriz simétrica e definida positiva. Em seus trabalhos Golub e Yuan apresentam dois algoritmos que fazem a decomposição ST em blocos; são necessários para isso armazenar a matriz A e resolver três sistemas lineares a cada passo da decomposição.

Em nossa nova análise descobrimos que a decomposição ST pode ser feita por linhas e isso apresenta uma grande vantagem, pois não precisamos armazenar a matriz A porque utilizamos apenas uma linha de A e resolvemos dois sistemas lineares a cada passo do processo.

Desenvolvemos dois novos algoritmos que apresentam um menor custo computacional e melhores resultados numéricos. Nossos resultados foram comparados com: Decomposição ST, Decomposição MST (ST Modificado), Decomposição LU sem pivotiamento, Decomposição $Cholesky$ e Decomposição QR.

Abstract

The main objective of this work is to propose new algorithms of the ST Decomposition (Symmetric-Triangular Decomposition) developed by Golub and Yuan in 2000.

The decomposition ST decomposes a square and nonsingular (in general, non-symmetric) matrix A into a matrix product $A = ST$ where T is triangular, and S is symmetrical and positive definite. Golub and Yuan present two algorithms to obtain the decomposition ST in block-wise. Then it is necessary to store the matrix A and to solve three linear systems to each step of the decomposition.

In our new algorithms, we discover that the decomposition ST can be attained by row-wise which needs just a row of A and solver of two linear systems at each step. By this way we can save multiplication and storage requirements.

We develop two new algorithms that are cheaper and more stable. Our results are compared to: ST Decomposition, MST Decomposition (Modified ST), LU Decomposition without pivoting, *Cholesky* Decomposition, and QR Decomposition for several types of test matrices.

Chapter 1

Introduction

Given an $n \times n$ nonsingular matrix A and n-vector b, we have to find an n-vector x such that

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

For solving the problem (1.1) there are two types of methods: Direct Methods and Iterative Methods. The direct method is to use matrix decomposition to get a simple and equivalent system. It is well-known that the symmetry and positivity of the matrix A posses very nice properties for both direct method and iterative method in scientific computing. It is much easier to solve the problem (1.1) with symmetric or symmetric and positive definite matrices.

There is the famous Cholesky decomposition for symmetric and positive definite A (see, section 2.2). When A is very large and sparse symmetric and positive definite, the conjugate gradient method is an excellent iterative method [21].

We meet many difficulties to solve nonsymmetric or symmetric indefinite problems even if there are some direct methods and iterative methods, such as the LU decomposition, the Gaussian Elimination, the QR decomposition, the QZ decomposition, the Lanczos method, the $GMRES$ method and various generalizations of the conjugate gradient method [3, 7, 17]. An important application example of a symmetric

indefinite problem is the augmented system

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} p \\ q \end{pmatrix}, \tag{1.2}$$

which appears is many areas, such as, constrained optimization [3, 1, 44], generalized least squares problems [45, 46, 47, 48, 37, 38, 50, 20], numerical methods for partial differential equations [9, 10, 16]. Since the system is large sparse, several preconditioned iterative methods have been proposed for solving it recently. Generally, there are two kinds of B as preconditioner [45, 46, 47, 48, 37, 38, 50, 20, 28], or Shur complement of (1.2) as preconditioner [9, 10, 14, 49]. For the first case, it is not easy to find a right submatrix of B as a good preconditioner. For the second case, the Shur complement is always ill-conditioned and it is not easily inverted. For the finite element approximation to Navier-Stokes equation, there are some easy ways to approximate the Shur complement because A and B have some special relationships. For general augmented systems, however, there is not a good way to approximate the Shur complement.

In real applications, we also meet many nonsymmetric problems [49]. Recently we have to solve large sparse nonsymmetric linear systems or symmetric indefinite linear systems, nonsymmetric generalized eigenvalue problems and other nonsymmetric problems. Since the properties of nonsymmetric matrices are quite different from that of the symmetric case, we meet many serious difficulties in many applications for numerical analysis.

Therefore, to solve the problems mentioned above we will modify our problem (1.1) in the following way:

$$Ax = b \tag{1.3}$$

$$TAx = Tb \tag{1.4}$$

$$Sx = \tilde{b} \tag{1.5}$$

where T is triangular matrix and $TA = S = LL^T$ is a symmetric positive definite

matrix and L is a lower triangular matrix with positive diagonal entries. The matrix T can also be defined as a preconditioner for the problem (1.1).

The eigenvalue problem typically arises in the explicit solution and stability analysis of a homogeneous system of first-order differential equations. The stability requires only implicit knowledge of eigenvalues, whereas the explicit solution requires eigenvalues and eigenvector explicitly.

Applications such as buckling problems, stock market analysis, and study of dynamic systems behavior, require computations of only a few eigenvalues and eigenvectors, usually the few largest or smallest ones.

In many practical instances, the matrix A is symmetric, and thus the eigenvalue problem becomes a symmetric eigenvalue problem. A great number of eigenvalue problems arising in engineering applications are, however, generalized eigenvalue problems, as stated next.

$$Ax = \lambda Bx, \tag{1.6}$$

where A and B are $n \times n$ matrices. There are several efficient methods to solve the problem (1.6) when A and B are symmetric, and B is positive definite too. A few papers discussed the generalized eigenvalue problem with A or B which are not symmetric. For a large fluid-structure interaction problem [33] and some differential equation problems [15], we meet the generalized eigenvalue problem with nonsymmetric A and B. Now, we can reduce the nonsymmetric generalized eigenvalue problem to a symmetric generalized eigenvalue problem by ST decomposition. Then, many numerical methods [51, 12, 39, 11, 26] for symmetric eigenvalue problems can be applied to solve nonsymmetric generalized eigenvalue problems.

Suppose that there exist T and L such that $TB = LL^T$. Then,

$$TAx = \lambda LL^T x, \tag{1.7}$$

which can be easily solved by many well-known and good methods.

The outline of this word is as follows. In Chapter 2, we will show some basic concepts related to matrix decomposition, solution of triangular linear systems and

the ST Decomposition. In Chapter 3, we will propose new algorithms of the ST decomposition. In Chapter 4 the numerical tests are presented and Chapter 5 the conclusions and the future works suggestions.

Chapter 2

Some basic knowledge on Matrix Decomposition

In this chapter, we review some well-known methods for a linear solving system and present a ST Decomposition summary.

2.1 Gaussian Elimination and LU Factorization

Consider the problem of solving the linear system of n equations in n unknowns:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & = & b_2 \\ & & & & & & & \vdots & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \ldots & + & a_{nn}x_n & = & b_n \end{array}$$

or, in matrix notation,

$$Ax = b$$

where $A = (a_{ij})$ and $b = (b_1, \ldots, b_n)^T$.

A well-known approach to solving the problem is the classical elimination scheme known as **Gaussian elimination**. The basic idea is to reduce the system to an equivalent upper triangular system which can be solved easily by the back substitution algorithm (Algorithm B.0.1).

The **reduction process** consists of $n-1$ steps. In the following,

$$b^{(k)} = \begin{pmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_n^{(k)} \end{pmatrix}, \qquad A^{(k)} = \left(a_{ij}^{(k)}\right),$$

$$A^{(0)} = A, \qquad b^{(0)} = b$$

Step 1:

At step 1, the unknown x_1 is eliminated from the second through the nth equations. This is done by multiplying the fist equation by

$$\left(-\frac{a_{21}}{a_{11}}\right), \left(-\frac{a_{31}}{a_{11}}\right), \ldots, \left(-\frac{a_{n1}}{a_{11}}\right)$$

and adding it to the second through nth equations. The quantities

$$m_{i1} = -\frac{a_{i1}}{a_{11}}, \qquad i = 2, \ldots, n$$

are called **multipliers**. At the end of step 1, the system $Ax = b$ becomes $A^{(1)}x = b^{(1)}$, where the entries of $A^{(1)} = (a_{ij}^{(1)})$ and those of $b^{(1)}$ are related to the entries of A and b as follows:

$$\begin{aligned} b_i^{(1)} &= a_{ij} + m_{i1}a_{1j} && (i = 2, \ldots, n; \quad j = 2, \ldots, n) \\ b_i^{(1)} &= b_i + m_{i1}b_1 && (i = 2, \ldots, n) \end{aligned}$$

(*Note that* $a_{21}^{(1)}, a_{31}^{(1)}, \ldots, a_{n1}^{(1)}$ *are all zero.*)

Step 2:

At step 2, x_2 is eliminated from the third through the nth equations of $A^{(1)}x = b^{(1)}$ by multiplying the second equations of $A^{(1)}x = b^{(1)}$ by the multiples

$$m_{i2} = -\frac{a_{i2}^{(1)}}{a_{22}^{(1)}}, \qquad i = 3, \ldots, n$$

and adding the result to the third through nth equations. The system now becomes $A^{(2)}x = b^{(2)}$, whose entries are given as follows:

$$\begin{aligned} a_{ij}^{(2)} &= a_{ij}^{(1)} + m_{i2}a_{2j}^{(1)} && (i = 3, \ldots, n; \quad j = 3, \ldots, n) \\ b_i^{(2)} &= b_i^{(1)} + m_{i2}b_2^{(1)} && (i = 3, \ldots, n) \\ a_{i2}^{(2)} &= 0, && (i = 3, \ldots, n) \end{aligned}$$

The other entries of $A^{(2)}$ and those of $b^{(2)}$ are the same as those of $A^{(1)}$ and $b^{(1)}$ respectively.

Step k:

At step k, the $(n-k)$ multipliers

$$m_{ik} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k+1, \ldots, n$$

are formed; using them, x_k is eliminated from the $(k+1)$st through the nth equation of $A^{(k-1)}x = b^{(k-1)}$. The entries of $A^{(k)}$ and those of $b^{(k)}$ are given by

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} + m_{ik}a_{kj}^{(k-1)} && (i = k+1, \ldots, n; \quad j = k+1, \ldots, n) \\ b_i^{(k)} &= b_i^{(k-1)} + m_{ik}b_k^{(k-1)} && (i = k+1, \ldots, n) \\ a_{ik}^{(k)} &= 0, && (i = k+1, \ldots, n) \end{aligned}$$

The other entries of $A^{(k)}$ and $b^{(k)}$ are same as those of $A^{(k-1)}$ and $b^{(k-1)}$ respectively.

Step $n-1$*:*

At the end of the $(n-1)$st step, the reduced matrix $A^{(n-1)}$ is upper triangular and the original vector b is transformed to $b^{(n-1)}$.

Now we present the pseudocodes of Gauss elimination and some observations.

The following observation will help to write the pseudocodes:

1. There are $n-1$ steps:
$$k = 1, 2, \ldots, n-1$$
2. For each value of k, there are $n-k$ multipliers: m_{ik}, $(i = k+1, \ldots, n)$.
3. For each value of k, only $(n-k)^2$ entries of $A^{(k)} = (a_{ij}^k)$ are modified $(i = k+1, \ldots, n; j = k+1, \ldots, n)$. The $n-k$ entries below the $(k,k)th$ entry of the kth column are zeros, and the remaining other entries that are not modified remain the same as those of the corresponding entries of $A^{(k-1)}$.

ALGORITHM 2.1.1 *Basic Gaussian Elimination*

For $k = 1, 2, \ldots, n-1$ *do*

  For $i = 1, 2, \ldots, n$ *do*

    $m_{i,k} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$ *(assuming that* $a_{kk}^{(k-1)} \neq 0$*).*

    $b_i^{(k)} = b_i^{(k-1)} + m_{ik} b_k^{(k-1)}$

    For $j = k+1, \ldots, n$ *do*

    $a_{ij}^{(k)} = a_{ij}^{(k-1)} + m_{ik} a_{kj}^{(k-1)}$

(Note that $A = (a_{ij}) = (a_{ij}^{(0)})$, $b = b^{(0)}$*.)*

Remarks:

1. The basic Gaussian elimination algorithm is commonly known as the *Gaussian elimination algorithm without row interchanges or the Gaussian elimination algorithm without pivoting.*
2. The basic Gaussian elimination algorithm as presented here is *not* commonly used in practice. Two practical variations of this algorithm, known as Gaussian elimination with partial and complete pivoting, are described in several literatures.
3. *We have assumed that the quantities* $a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}$ *are different from zero. If any of them is zero, the algorithm will stop.*

Triangular factorization of a matrix. It is convenient to keep the multipliers m_{ij}, since we often want to solve $Ax = b$ with the same A but a different vector b. In the computer the elements $a_{ij}^{(k+1)}$, $j \geq i$, always are stored into the storage for $a_{ij}^{(k)}$. The elements below the diagonal are being zeroed, and this provides a convenient storage for the elements m_{ij}. Store m_{ij} into the space originally used to store a_{ij}, $i > j$.

There is yet another reason for looking at the multipliers m_{ij} as the elements of a matrix. First, introduce the lower triangular matrix

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & \cdots & 0 \\ -m_{31} & -m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & -m_{n2} & \cdots & -m_{n,n-1} & 1 \end{pmatrix}$$

THEOREM 2.1.1 *If L and U are the lower and upper triangular matrices defined previously using Gaussian elimination, then*

$$A = LU \tag{2.1}$$

See the proof of this theorem in [2] p. 511.

The decomposition (2.1.1) is an important result, and extensive use is made of it in developing variants of Gaussian elimination for special classes of matrices. But for the moment we give only the following corollary.

COROLLARY 2.1.2 *With the matrices A, L and U as in Theorem 2.1.1,*

$$\begin{aligned} det(A) &= u_{11}u_{22}\cdots u_{nn} \\ &= a_{11}^{(1)}a_{22}^{(2)}\cdots a_{nn}^{(n)} \end{aligned}$$

See the proof of this corollary in [2] p. 512.

2.2 Cholesky Factorization

For a symmetric positive definite matrix A, there exists a unique factorization

$$A = HH^T$$

where H is a lower triangular matrix with positive diagonal entries. The factorization is called the **Cholesky factorization**, after the French engineer Cholesky (1875-1918). The existence of the Cholesky factorization for a symmetric positive definite matrix A can be seen either by LU factorization of A or by finding the matrix H directly from the preceding relation.

THEOREM 2.2.1 *The Cholesky Factorization Theorem*
Let A be a symmetric positive definite matrix. Then A can be written uniquely in the form

$$A = HH^T$$

where H is lower triangular with positive diagonal entries. An explicit expression for H is given by

$$H = LD^{1/2}$$

where L is the unit lower triangular matrix in the LU factorization of A obtained by Gaussian elimination without pivoting and

$$D^{1/2} = diag\left(u_{11}^{1/2}, \ldots, u_{nn}^{1/2}\right)$$

ALGORITHM 2.2.1 *Cholesky Algorithm*
Given an $n \times n$ symmetric positive definite matrix A, the following computes the Cholesky factor H. The matrix H is computed row by row and is stored in the lower triangular part of A.

For $k = 1, 2, \ldots, n$ do

For $i = 1, 2, \ldots, k-1$ do

$$a_{ki} \equiv h_{ki} = \frac{1}{h_{ii}}\left(a_{ki} - \sum_{j=1}^{i-1} h_{ij}h_{kj}\right)$$

$$a_{kk} \equiv h_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{i-1} h_{kj}^2}$$

Solution of $Ax = b$ Using the Cholesky Factorization
Having the Cholesky factorization $A = HH^T$ at hand, we can transform a positive definite linear system $Ax = b$ into a lower triangular system $Hy = b$ where $y = H^T x$.

ALGORITHM 2.2.2 *The Cholesky Algorithm for the Positive Definite System $Ax = b$*

Step 1: Find the Cholesky factorization of

$$A = HH^T$$

using Algorithm 2.2.1

Step 2: Solve the lower triangular system for y:

$$Hy = b$$

using Algorithm B.0.2

Step 3: Solve the upper triangular system for x:

$$H^T x = y$$

using Algorithm B.0.1

2.3 Householder Matrices and QR Factorization

To compute the QR factorization of a general matrix we will now show how the Householder transformation can be used.

To reduce a matrix to upper triangular form we need to introduce zeros below its main diagonal. This can be done by premultiplying a matrix by a sequence of special matrices. One class of special matrices is the Householder transformation matrices. It can introduce zeros in a vector. Specifically, given a vector $v \in \mathbb{R}^n$ one can find a matrix H, such that, Hv is a multiple of e_1, the first column of I_n. This matrix H will be presented in the following lemma (See Lawson an Hanson [25] and Stewart [40]).

LEMMA 2.3.1 *Given $0 \neq v \in \mathbb{R}^n$, there exists an orthogonal matrix H such that*

$$Hv = -\sigma e_1,$$

with

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and

$$\sigma = \begin{cases} +\|v\|_2 & if \quad v_1 \geq 0 \\ -\|v\|_2 & if \quad v_1 < 0 \end{cases}$$

where v_1 is the first component of v.

Proof: Define

$$u = v + \sigma e_1,$$

and

$$H = I_n - \frac{2uu^T}{u^T u}. \tag{2.2}$$

One can directly verify that the matrix H of (2.2) is symmetric and orthogonal. Since $v^T v = \sigma^2$,

$$\begin{aligned} \frac{2}{v^t v} &= \frac{2}{(v+\sigma e_1)^T (v+\sigma e_1)} \\ &= \frac{2}{v^T v + 2\sigma v_1 + \sigma^2} \\ &= \frac{2}{2\sigma^2 + 2\sigma v_1} = \frac{1}{\sigma^2 + \sigma v_1} \end{aligned}$$

Hence,

$$\begin{aligned} Hv = v - \frac{uu^T v}{\sigma^2+\sigma v_1} &= v - \frac{(v+\sigma e_1)(v+\sigma e_1)^T v}{\sigma^2+\sigma v_1} \\ &= v - \frac{(v+\sigma e_1)(v^T v + \sigma e_1^T v)}{\sigma^2+\sigma v_1} \\ &= v - (v + \sigma e_1) = -\sigma e_1. \end{aligned}$$

□

This transformation can be seen geometrically as a reflection in the $(n-1)$-dimensional subspace, orthogonal to the vector u. That is, when a vector v is multiplied by H, it is reflected in the hyperplane $span\{u\}^{\perp}$. This is why, Householder transformation is also known as Householder reflection.

Hence, given a matrix $A \in \mathbb{R}^{m \times n}$, a sequence of matrices $H_1, H_2, \ldots, H_r$ ($r = min\{m-1, n\}$) can be computed such that the product $R_{r+1} = H_r H_{r-1} \ldots H_1 A$ is upper triangular.

The essence of the algorithm that will be presented below can be expressed by a small example (see [17]). Suppose $m = 6$, $n = 5$, and assume that Householder

matrices H_1 and H_2 have been computed so that

$$H_2H_1A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}.$$

Concentrating on the boldfaced entries, we next determine a Householder matrix $\tilde{H}_3 \in \mathbb{R}^{4\times 4}$ such that

$$\tilde{H}_3 \begin{pmatrix} \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \times \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

If $H_3 = diag(I_2, \tilde{H}_3)$, then

$$H_3H_2H_1A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}.$$

After these 5 steps we obtain an upper triangular matrix $H_5H_4 \ldots H_1A = R$ and then by setting $Q = H_1 \ldots H_5$ we obtain the factorization $A = QR$.

Formally, we have (see Stewart [40])

ALGORITHM 2.3.1 *Householder QR factorization*
begin

1. *Input* $A = (a_{ij}), m, n, r = min\{m-1, n\}$
2. *for* $k = 1, r$

$\sigma = sign(a_{kk})\sqrt{a_{kk}^2 + \ldots + a_{mk}^2}$

if $\sigma = 0$ *then go to 2.*

$a_{kk} = a_{kk} + \sigma$

$\pi_k = \sigma a_{kk}$

$\rho_k = -\delta$

for $j = k+1, n$

$\tau = \pi_k^{-1} \sum_{i=k}^{m} a_{ik} a_{ij}$

for $i = k, m$

$a_{ij} = a_{ij} - \tau a_{ik}$

end

end

end

3. *if* $m \leq n$ *then* $\rho_m = a_{mm}$
4. *Output*

- π_k, $k = 1, \ldots, r$;
- $A = (a_{ij})$ *where*
  - *the strict upper triangular part of A contains the elements of R;*
  - *the lower triangular part of A contains the vectors* u_k, $k = 1, \ldots, r$;
- ρ_k, $k = 1, \ldots, r$, *where* rho_k *is the kth element of R.*

end

In the above algorithm matrix A is overwritten by the elements in the strict upper triangular part of R and by the u_k's. If we take $\tilde{H}_k = I_k - \pi_k^{-1} u_k u_k^T$ and $H_k = (diag I_k, \tilde{H}_k)$, matrix Q can be retrieved by computing $Q = H_r H_{r-1}, \ldots, H_1$.

2.4 The ST Decomposition and Algorithms

Now we will present the basic ideas of ST decomposition developed by Gene H. Golub[1] and Jin Yun Yuan[2] in [18, 19]. The main idea of the new decomposition is to transform a square nonsingular and nonsymmetric matrix into a symmetric and positive defined matrix because the symmetric and positive definite matrix has many advantages in the resolution of a linear system.

In this section, the main theorems of the symmetric and triangular decomposition will be presented.

THEOREM 2.4.1 *For every nonsingular and nonsymmetric $n \times n$ matrix A, whose leading principal submatrices are nonsingular, there exist a symmetric matrix S and a unit triangular matrix T such that*

$$A = ST. \tag{2.3}$$

THEOREM 2.4.2 *For every nonsingular and nonsymmetric $n \times n$ matrix A, whose leading principal submatrices are nonsingular, there exist a symmetric matrix S and a unit triangular matrix T such that*

$$A = TS. \tag{2.4}$$

Proof: The proof of these two theorems is given in [18]. □

These two theorems show the existence of the unitary triangular matrix T and nonsingular matrix S.

We will now give an example for the previous theorems taken from [18].

[1]Department of Computer Science Stanford University, USA.
[2]Department of Mathematic - UFPR, Brazil.

EXAMPLE 2.4.1 *Suppose that the matrix A is p-cyclic matrix and is given by*

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & \cdots & a_{1p} \\ a_{21} & a_{22} & \ddots & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & a_{p,p-1} & a_{pp} \end{pmatrix}.$$

Here we consider $p = 4$. Then we have

$$S = \begin{pmatrix} a_{11} & a_{21} & 0 & 0 \\ a_{21} & \hat{a}_{22} & a_{23} & 0 \\ 0 & a_{23} & \hat{a}_{33} & a_{34} \\ 0 & 0 & a_{34} & \hat{a}_{44} \end{pmatrix} \quad \text{and} \quad T = \begin{pmatrix} 1 & t_{12} & 0 & t_{14} \\ 0 & 1 & t_{23} & t_{24} \\ 0 & 0 & 1 & t_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

such that

$$AT = S,$$

where

$$t_{12} = \frac{a_{21}}{a_{11}}, \qquad t_{14} = -\frac{a_{14}}{a_{11}},$$

$$t_{23} = \frac{a_{23}}{a_{22}}, \qquad t_{24} = -\frac{t_{14}a_{21}}{a_{22}},$$

$$t_{34} = \frac{a_{34} - a_{33}t_{34}}{a_{33}}, \qquad \hat{a}_{22} = a_{22} + t_{12}a_{21},$$

$$\hat{a}_{33} = a_{33} + t_{23}a_{23}, \qquad \text{and} \qquad \hat{a}_{44} = a_{44} + t_{34}a_{34}.$$

Now we shall allow the matrix T change from unit triangular to a triangular matrix so that S is symmetric and positive definite.

Now the conditions of the previous theorems will be relaxed to attain the following general theorems.

THEOREM 2.4.3 *For every nonsingular and nonsymmetric $n \times n$ matrix A, whose leading principal submatrices are nonsingular, there exist a triangular matrix T and a symmetric and positive definite matrix S such that*

$$A = TS.$$

Note that the matrices S and T in Theorem 2.4.3 are positive definite and general triangular respectively, but just symmetric and unit triangular respectively in Theorem 2.4.4. Similarly, we can show the following results as well.

THEOREM 2.4.4 *For every nonsingular and nonsymmetric $n \times n$ matrix A, whose leading principal submatrices are nonsingular, there exist a triangular matrix T and a symmetric and positive definite matrix S such that*

$$A = ST.$$

For the nonsingularity of the matrix T, we can write the theorems (2.4.3) and (2.4.4) in the following forms:

$$T^{-1}A = S$$

$$AT^{-1} = S.$$

Making $T^{-1} = T$ now, we obtain:

THEOREM 2.4.5 *For every nonsingular and nonsymmetric $n \times n$ matrix A, whose leading principal submatrices are nonsingular, there exist a triangular matrix T and a symmetric and positive definite matrix S such that*

$$TA = S = LL^T,$$

or

$$AT = S = LL^T.$$

Proof: The proof of these two theorems is given in [18]. □

We will show now the general ideas that resulted in the construction of the algorithms ST idealized by Golub and Yuan.

Since S is symmetric and positive definite, there exists a lower triangular matrix L such that $S = LL^T$. Suppose that there exist lower triangular matrices T and L such that $TA = LL^T$. Assume that

$$T = \begin{pmatrix} T_k & 0 & 0 \\ t_{k+1}^T & \tau_{k+1} & 0 \\ \tilde{T}_k & \breve{t}_{k+1} & \hat{T}_{n-k} \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} L_k & 0 & 0 \\ l_{k+1}^T & \lambda_{k+1} & 0 \\ \tilde{L}_k & \breve{l}_{k+1} & \hat{L}_{n-k} \end{pmatrix}.$$

Consequently, A has the following form

$$A = \begin{pmatrix} A_k & \tilde{a}_{k+1} & \tilde{A}_k \\ a_{k+1}^T & \alpha_{k+1} & \breve{a}_{k+1}^T \\ \breve{A}_{n-k} & \hat{a}_{n-k} & \hat{A}_{n-k} \end{pmatrix}.$$

It follows form $LL^T = TA$ that

$$\begin{pmatrix} L_k & 0 \\ l_{k+1}^T & \lambda_{k+1} \end{pmatrix} \begin{pmatrix} L_k^T & l_{k+1} \\ 0 & \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} T_k & 0 \\ t_{k+1}^T & \tau_{k+1} \end{pmatrix} \begin{pmatrix} A_k & \tilde{a}_{k+1} \\ a_{k+1}^T & \alpha_{k+1} \end{pmatrix} \tag{2.5}$$

from algebraic operations of matrices in (2.5) where we removed the following elements:

$$l_{k+1} = L_k^{-1} T_k \tilde{a}_{k+1}, \tag{2.6}$$

$$t_{k+1} = T_k^T L_k^{-T} (l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) \tag{2.7}$$

$$\tau_{k+1}(\alpha_{k+1} - a_{k+1}^T L_k^{-T} l_{k+1}) = \lambda_{k+1}^2 \tag{2.8}$$

By the nonsingularity of A, $\alpha_{k+1} - a_{k+1}^T L_k^{-T} l_{k+1} \neq 0$. So, we take $\tau_{k+1} \neq 0$ such that $\tau_{k+1}(\alpha_{k+1} - a_{k+1}^T L_k^{-T} l_{k+1}) > 0$. Then, the method is well defined. Therefore, we have the following algorithm to decompose A as a product of triangular matrix T and symmetric and positive definite matrix $TA = S = LL^T$ (theorem 2.4.5). The

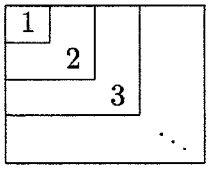

Figure 2.1: Manner of ST Decomposition

method to find T and L, which was called the escalator method (see [23], page 127), is shown in Figure 2.1.

With these ideas, we obtain the following algorithms easily:

ALGORITHM 2.4.1 *(Golub-Yuan ST I)*
Set τ_1 *such that* $\tau_1\alpha_1 > 0$ *and* $\lambda_1 = \sqrt{\tau_1\alpha_1}$;

For $k = 1, 2, \ldots, n-1$

$$l_{k+1} = L_k^{-1} T_k \tilde{a}_{k+1},$$
$$\hat{l}_{k+1} = L_k^{-1} a_{k+1},$$
$$s = \alpha_{k+1} - \hat{l}_{k+1}^T l_{k+1},$$

Choose $\tau_{k+1} \neq 0$ *such that* $\mu = \tau_{k+1} s > 0$ *(or big enough),*

$$\lambda_{k+1} = \sqrt{\mu},$$
$$t_{k+1} = T_k^T L_k^{-T} (l_{k+1} - \tau_{k+1} \hat{l}_{k+1}).$$

We can also derive another version of the decomposition as $TLL^T = A$ (theorem 2.4.3) as follows.

ALGORITHM 2.4.2 *(Golub-Yuan ST II)*
Set τ_1 *such that* $\tau_1\alpha_1 > 0$ *and* $\lambda_1 = \sqrt{\frac{\alpha_1}{\tau_1}}$;

For $k = 1, 2, \ldots, n-1$

$$l_{k+1} = L_k^{-1} T_k^{-1} \tilde{a}_{k+1},$$
$$\hat{l}_{k+1} = L_k^{-1} a_{k+1},$$
$$s = \alpha_{k+1} - \hat{l}_{k+1}^T l_{k+1},$$

Choose $\bar{\tau}_{k+1} \neq 0$ *such that* $\eta = s/\bar{\tau}_{k+1} > 0$ *(or big enough),*

$$\lambda_{k+1} = \sqrt{\eta},$$
$$\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1} l_{k+1}).$$

These two algorithms require $2n^3/3$ flops, because they solve three linear systems to each step.

Chapter 3

New Algorithms for the ST Decomposition

We will make a new analysis of the ST Decomposition to obtain more stable and faster algorithms and that just uses a row of the matrix A.

3.1 A new analysis of the method

Let us suppose that matrices have the following forms:

$$A = \begin{pmatrix} A_k & \tilde{a}_{k+1} & \tilde{A}_k \\ a_{k+1}^T & \alpha_{k+1} & \breve{a}_{k+1}^T \\ \breve{A}_{n-k} & \hat{a}_{n-k} & \hat{A}_{n-k} \end{pmatrix} \quad , \quad T = \begin{pmatrix} T_k & 0 & 0 \\ t_{k+1}^T & \tau_{k+1} & 0 \\ \tilde{T}_k & \breve{t}_{k+1} & \hat{T}_{n-k} \end{pmatrix} \quad \textit{and}$$

$$L = \begin{pmatrix} L_k & 0 & 0 \\ l_{k+1}^T & \lambda_{k+1} & 0 \\ \tilde{L}_k & \breve{l}_{k+1} & \hat{L}_{n-k} \end{pmatrix}.$$

Making the ST Decomposition ($TA = LL^T$), we have:

$$\begin{pmatrix} T_k & 0 & 0 \\ t_{k+1}^T & \tau_{k+1} & 0 \\ \tilde{T}_k & \breve{t}_{k+1} & \hat{T}_{n-k} \end{pmatrix} \begin{pmatrix} A_k & \tilde{a}_{k+1} & \tilde{A}_k \\ a_{k+1}^T & \alpha_{k+1} & \breve{a}_{k+1}^T \\ \breve{A}_{n-k} & \hat{a}_{n-k} & \hat{A}_{n-k} \end{pmatrix} = \begin{pmatrix} L_k & 0 & 0 \\ l_{k+1}^T & \lambda_{k+1} & 0 \\ \tilde{L}_k & \breve{l}_{k+1} & \hat{L}_{n-k} \end{pmatrix} \begin{pmatrix} L_k^T & l_{k+1} & \tilde{L}_k^T \\ 0 & \lambda_{k+1} & \breve{l}_{k+1}^T \\ 0 & 0 & \hat{L}_{n-k}^T \end{pmatrix}$$

Operating the matrix products, we obtain:

$$\begin{pmatrix} T_k A_k & T_k \tilde{a}_{k+1} & T_k \hat{A}_k \\ t_{k+1}^T A_k + \tau_{k+1} a_{k+1}^T & t_{k+1}^T \tilde{a}_{k+1} + \tau_{k+1}\alpha_{k+1} & t_{k+1}^T \hat{A}_k + \tau_{k+1} \breve{a}_{k+1}^T \\ \tilde{T}_k A_k + \breve{t}_{k+1} a_{k+1}^T + \hat{T}_{n-k} \breve{A}_{n-k} & \tilde{T}_k \tilde{a}_{k+1} + \breve{t}_{k+1}\alpha_{k+1} + \hat{T}_{n-k}\tilde{a}_{n-k} & \tilde{T}_k \tilde{A}_k + \breve{t}_{k+1}\breve{a}_{k+1}^T + \hat{T}_{n-k}\tilde{A}_{n-k} \end{pmatrix} =$$

$$= \begin{pmatrix} L_k L_k^T & L_k l_{k+1} & L_k \tilde{L}_k^T \\ l_{k+1}^T L_k^T & l_{k+1}^T l_{k+1} + \lambda_{k+1}^2 & l_{k+1}^T \tilde{L}_k^T + \lambda_{k+1} \breve{l}_{k+1}^T \\ \tilde{L}_k L_k^T & \tilde{L}_k l_{k+1} + \breve{l}_{k+1}\lambda_{k+1} & \tilde{L}_k \tilde{L}_k^T + \breve{l}_{k+1}\breve{l}_{k+1}^T + \hat{L}_{n-k}\hat{L}_{n-k}^T \end{pmatrix} \tag{3.1}$$

It follows from the first two lines that

$$T_k A_k = L_k L_k^T, \tag{3.2}$$

$$T_k \tilde{a}_{k+1} = L_k l_{k+1}, \tag{3.3}$$

$$T_k \tilde{A}_k = L_k \tilde{L}_k^T, \tag{3.4}$$

$$t_{k+1}^T A_k + \tau_{k+1} a_{k+1}^T = l_{k+1}^T L_k^T, \tag{3.5}$$

$$t_{k+1}^T \tilde{a}_{k+1} + \tau_{k+1}\alpha_{k+1} = l_{k+1}^T l_{k+1} + \lambda_{k+1}^2, \tag{3.6}$$

$$t_{k+1}^T \tilde{A}_k + \tau_{k+1} \breve{a}_{k+1}^T = l_{k+1}^T \tilde{L}_k^T + \lambda_{k+1} \breve{l}_{k+1}^T. \tag{3.7}$$

From expression (3.5), we have

$$(t_{k+1}^T A_k + \tau_{k+1} a_{k+1}^T)^T = (l_{k+1}^T L_k^T)^T$$

$$A_k^T t_{k+1} + \tau_{k+1} a_{k+1} = L_k l_{k+1} \tag{3.8}$$

In terms of (3.8), there is

$$t_{k+1} = A_k^{-T}(L_k l_{k+1} - \tau_{k+1} a_{k+1}) \tag{3.9}$$

From (3.2), it follows that

$$(T_k A_k)^T = (L_k L_k^T)^T$$

$$A_k^T T_k^T = L_k L_k^T$$

$$(A_k^T T_k^T)^{-1} = (L_k L_k^T)^{-1}$$

$$T_k^{-T} A_k^{-T} = L_k^{-T} L_k^{-1}$$

$$A_k^{-T} = T_k^T L_k^{-T} L_k^{-1} \tag{3.10}$$

Substituting (3.10) in (3.9), we obtain:

$$t_{k+1} = T_k^T L_k^{-T} L_k^{-1} (L_k l_{k+1} - \tau_{k+1} a_{k+1})$$

$$t_{k+1} = T_k^T L_k^{-T} (l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) \tag{3.11}$$

Writing (3.6) in another manner, we obtain

$$(t_{k+1}^T \tilde{a}_{k+1} + \tau_{k+1}\alpha_{k+1})^T = (l_{k+1}^T l_{k+1} + \lambda_{k+1}^2)^T$$

$$\tilde{a}_{k+1}^T t_{k+1} + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2 \tag{3.12}$$

Substituting (3.11) in (3.12), we attain

$$\tilde{a}_{k+1}^T T_k^T L_k^{-T} (l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2 \tag{3.13}$$

In terms of (3.3), there is

$$\tilde{a}_{k+1}^T = l_{k+1}^T L_k^T T_k^{-T} \tag{3.14}$$

Substituting (3.14) in (3.13), we obtain

$$l_{k+1}^T L_k^T T_k^{-T} T_k^T L_k^{-T} (l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2$$

$$\|l_{k+1}\|_2^2 - \tau_{k+1} l_{k+1}^T L_k^{-1} a_{k+1} + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2$$

$$\lambda_{k+1}^2 = \tau_{k+1}(\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}) \tag{3.15}$$

The expression (3.15) obtained above is extremely important because it relates to variables λ_{k+1} and τ_{k+1}. In this expression it is observed that τ_{k+1} should have the same sign of $(\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1})$, because $\lambda_{k+1}^2 > 0$.

Continuing the process to find other relationships, we know

$$\lambda_{k+1} \breve{l}_{k+1}^T = t_{k+1}^T \tilde{A}_k + \tau_{k+1} \breve{a}_{k+1}^T - l_{k+1}^T \tilde{L}_k^T$$

$$\lambda_{k+1}\breve{l}_{k+1} = \tilde{A}_k^T t_{k+1} + \tau_{k+1}\breve{a}_{k+1} - \tilde{L}_k l_{k+1} \tag{3.16}$$

Substituting t_{k+1} for (3.11) in (3.16), we obtain:

$$\lambda_{k+1}\breve{l}_{k+1} = \tilde{A}_k^T T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1}L_k^{-1}a_{k+1}) + \tau_{k+1}\breve{a}_{k+1} - \tilde{L}_k l_{k+1} \tag{3.17}$$

Now from (3.4), it follows that

$$\tilde{A}_k^T = \tilde{L}_k L_k^T T_k^{-T} \tag{3.18}$$

Substituting (3.18) in (3.17), we obtain:

$$\lambda_{k+1}\breve{l}_{k+1} = \tilde{L}_k L_k^T T_k^{-T} T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1}L_k^{-1}a_{k+1}) + \tau_{k+1}\breve{a}_{k+1} - \tilde{L}_k l_{k+1}$$

$$\lambda_{k+1}\breve{l}_{k+1} = \tilde{L}_k l_{k+1} - \tau_{k+1}\tilde{L}_k L_k^{-1}a_{k+1} + \tau_{k+1}\breve{a}_{k+1} - \tilde{L}_k l_{k+1}$$

$$\lambda_{k+1}\breve{l}_{k+1} = \tau_{k+1}(\breve{a}_{k+1} - \tilde{L}_k L_k^{-1}a_{k+1})$$

$$\breve{l}_{k+1} = \frac{\tau_{k+1}}{\lambda_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k L_k^{-1}a_{k+1}) \tag{3.19}$$

We can write (3.15) in the following form

$$\lambda_{k+1}\lambda_{k+1} = \tau_{k+1}(\alpha_{k+1} - l_{k+1}^T L_k^{-1}a_{k+1})$$

$$\frac{\tau_{k+1}}{\lambda_{k+1}} = \frac{\lambda_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1}a_{k+1}} \tag{3.20}$$

Substituting now (3.20) in (3.19), we obtain

$$\breve{l}_{k+1} = \lambda_{k+1}\frac{\breve{a}_{k+1} - \tilde{L}_k L_k^{-1}a_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1}a_{k+1}} \tag{3.21}$$

Making now the ST Decomposition in the following $A = TLL^T$, we have

$$\begin{pmatrix} A_k & \tilde{a}_{k+1} & \tilde{A}_k \\ a_{k+1}^T & \alpha_{k+1} & \breve{a}_{k+1}^T \\ \breve{A}_{n-k} & \hat{a}_{n-k} & \hat{A}_{n-k} \end{pmatrix} = \begin{pmatrix} T_k & 0 & 0 \\ t_{k+1}^T & \tau_{k+1} & 0 \\ \tilde{T}_k & \breve{t}_{k+1} & \hat{T}_{n-k} \end{pmatrix} \begin{pmatrix} L_k & 0 & 0 \\ l_{k+1}^T & \lambda_{k+1} & 0 \\ \tilde{L}_k & \breve{l}_{k+1} & \hat{L}_{n-k} \end{pmatrix} \begin{pmatrix} L_k^T & l_{k+1} & \tilde{L}_k^T \\ 0 & \lambda_{k+1} & \breve{l}_{k+1}^T \\ 0 & 0 & \hat{L}_{n-k}^T \end{pmatrix}.$$

Making the convenient products to obtain the following equations which results in our second algorithm as follows.

$$T_k L_k L_k^T = A_k, \tag{3.22}$$

$$T_k L_k l_{k+1} = \tilde{a}_{k+1}, \tag{3.23}$$

$$T_k L_k \tilde{L}_k^T = \tilde{A}_k, \tag{3.24}$$

$$t_{k+1}^T L_k L_k^T + \tau_{k+1} l_{k+1}^T L_k^T = a_{k+1}^T, \tag{3.25}$$

$$t_{k+1}^T L_k l_{k+1} + \tau_{k+1}(l_{k+1}^T l_{k+1} + \lambda_{k+1}^2) = \alpha_{k+1}, \tag{3.26}$$

$$t_{k+1}^T L_k \tilde{L}_k^T + \tau_{k+1}(l_{k+1}^T \tilde{L}_k^T + \lambda_{k+1} \breve{l}_{k+1}^T) = \breve{a}_{k+1}^T. \tag{3.27}$$

With the expressions above, we easily arrive at the following relations

$$l_{k+1} = L_k^{-1} T_k^{-1} \tilde{a}_{k+1}, \tag{3.28}$$

$$t_{k+1} = L_k^{-T}(L_k^{-1} a_{k+1} - \tau_{k+1} l_{k+1}), \tag{3.29}$$

$$\tau_{k+1}\lambda_{k+1}^2 = \alpha_{k+1} - a_{k+1}^T L_k^{-T} l_{k+1}, \tag{3.30}$$

$$\breve{l}_{k+1} = \lambda_{k+1} \frac{\breve{a}_{k+1} - \tilde{L} L_k^{-1} a_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}}. \tag{3.31}$$

3.2 The new algorithms

Considering the facts seen in section (3.1) we can write these two new algorithms using the row k of the matrix A.

This algorithm makes the following transformation $TA = LL^T$.

ALGORITHM 3.2.1 *(New ST I)*
Set $\tau_1\alpha_1 > 0$, $\lambda_1 = \sqrt{\tau_1\alpha_1}$ *and* $L(2:n,1) = \frac{\lambda_1}{\alpha_1} A(1,2:n)^T$;

For $k = 1, 2, \ldots, n-1$, *do*

$$l_{k+1} = L(k+1, 1:k)^T,$$
$$\hat{l}_{k+1} = L_k^{-1} a_{k+1},$$
$$\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T \hat{l}_{k+1},$$

Choose $\lambda_{k+1} > 0$ *such that* $LL^T = S$

$$\tau_{k+1} = \frac{\lambda_{k+1}^2}{\mu_{k+1}},$$
$$\breve{l}_{k+1} = \frac{\lambda_{k+1}}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1}),$$
$$t_{k+1} = T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1}\hat{l}_{k+1}).$$

This algorithm makes the following transformation $A = TLL^T$.

ALGORITHM 3.2.2 *(New ST II)*
Set $\tau_1\alpha_1 > 0$, $\lambda_1 = \sqrt{\frac{\alpha_1}{\tau_1}}$ *and* $L(2:n, 1) = \frac{\lambda_1}{\alpha_1} A(1, 2:n)^T$;

For $k = 1, 2, \ldots, n-1$, *do*

$$l_{k+1} = L(k+1, 1:k)^T,$$
$$\hat{l}_{k+1} = L_k^{-1} a_{k+1},$$
$$\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T \hat{l}_{k+1},$$

Choose $\lambda_{k+1} > 0$ *such that* $LL^T = S$

$$\bar{\tau}_{k+1} = \frac{\mu_{k+1}}{\lambda_{k+1}^2},$$
$$\breve{l}_{k+1} = \frac{\lambda_{k+1}}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1}),$$
$$\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1} l_{k+1}).$$

Remark:

Since these two algorithms solve two linear systems at each step, they are faster than the algorithms (2.4.1) and (2.4.2). The most important fact is that they just

use a row of the matrix A at each step which leads to nonnecessity to store the whole matrix A for many special applications.

As the ST decomposition is not unique, we will make a more detailed analysis in Algorithm (3.2.2) to improve its numerical performance.

To decide what is $\lambda_{k+1} > 0$, we have to analyze two subproblems.

1. $\breve{l}_{k+1} = \frac{\lambda_{k+1}}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k\hat{l}_{k+1})$ is a vector. Therefore, $\frac{\lambda_{k+1}}{\mu_{k+1}}$ should be controlled so that $\breve{l}_{k+1}$ does not generate large numbers which leads to breakdown problems.

2. $\bar{\tau}_{k+1} = \frac{\mu_{k+1}}{\lambda_{k+1}^2}$ is a real number to be controlled so that we don't have breakdown problems.

An alternative to control the referred problems is the following

i. If $|\mu_{k+1}| > 1$ we will assume that $\lambda_{k+1} = 1$ because it implies $\bar{\tau}_{k+1} = \mu_{k+1}$ and $|\frac{\lambda_{k+1}}{\mu_{k+1}}| < 1$.

ii. If $|\mu_{k+1}| < 1$ the case is a little bit more complicated. An alternative way is $\lambda_{k+1} = \sqrt{|\mu_{k+1}|}$ that implies $\bar{\tau}_{k+1} = sign(\mu_{k+1})$ and $|\frac{\lambda_{k+1}}{\mu_{k+1}}| = \frac{1}{\lambda_{k+1}}$.

With these considerations, the Algorithm (3.2.2) can be rewritten in the following manner.

ALGORITHM 3.2.3 *(New ST II)*

Set $\tau_1\alpha_1 > 0$, $\lambda_1 = \sqrt{\frac{\alpha_1}{\tau_1}}$ *and* $L(2:n,1) = \frac{\lambda_1}{\alpha_1}A(1,2:n)^T$;

For $k = 1, 2, \ldots, n-1$, *do*

$$l_{k+1} = L(k+1, 1:k)^T,$$
$$\hat{l}_{k+1} = L_k^{-1} a_{k+1},$$
$$\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T \hat{l}_{k+1},$$

if $|\mu_{k+1}| > 1$

$$\lambda_{k+1} = 1,$$
$$\bar{\tau}_{k+1} = \mu_{k+1},$$
$$\breve{l}_{k+1} = \frac{1}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1}),$$

else

$$\lambda_{k+1} = \sqrt{|\mu_{k+1}|},$$
$$\bar{\tau}_{k+1} = sign(\mu_{k+1}),$$
$$\breve{l}_{k+1} = \frac{sign(\mu_{k+1})}{\lambda_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1}),$$

end

$$\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1} l_{k+1}),$$

end

See the implementation of this algorithm in Appendix A.

EXAMPLE 3.2.1 *Consider the Door's matrix 4-by-4 (see 4.1.2)*

$$A = \begin{pmatrix} 2 & -7/4 & 0 & 0 \\ -1/4 & 1 & -3/4 & 0 \\ 0 & -3/4 & 1 & -1/4 \\ 0 & 0 & -7/4 & 2 \end{pmatrix}.$$

We will make the ST Decomposition using the algorithm 3.2.1.

Solution:

Step 0 $A_1 = (2 \quad -\frac{7}{4} \quad 0 \quad 0)$,

$\tau_1 = \alpha_1 = 2$,

$\lambda_1 = 2$,

$L(2:4,1) = (-\frac{7}{4} \quad 0 \quad 0)^T$,

Output

$$L_1 = \begin{pmatrix} 2 \\ -\frac{7}{4} \\ 0 \\ 0 \end{pmatrix} \qquad \textit{and} \qquad T_1 = (2)$$

Step 1 $A_2 = (-\frac{1}{4} \quad 1 \quad -\frac{3}{4} \quad 0)$,

$l_2 = (-\frac{7}{4})$,

$\hat{l}_2 = (-\frac{1}{8})$,

$\mu_2 = \frac{25}{32}$,

$\lambda_2 = \frac{25}{32}$,

$\tau_2 = \frac{25}{32}$,

$$\breve{l}_2 = \begin{pmatrix} -\frac{3}{4} \\ 0 \end{pmatrix},$$

$t_2^T = (-\frac{423}{256})$,

Output

$$L_2 = \begin{pmatrix} 2 & 0 \\ -\frac{7}{4} & \frac{25}{32} \\ 0 & -\frac{3}{4} \\ 0 & 0 \end{pmatrix} \qquad \textit{and} \qquad T_2 = \begin{pmatrix} 2 & 0 \\ -\frac{423}{256} & \frac{25}{32} \end{pmatrix}$$

Step 2 $A_3 = (0 \quad -\frac{3}{4} \quad 1 \quad -\frac{1}{4})$,

$$l_3 = \begin{pmatrix} 0 \\ -\frac{3}{4} \end{pmatrix},$$

$$\hat{l}_3 = \begin{pmatrix} 0 \\ -\frac{24}{25} \end{pmatrix},$$

$\mu_3 = \frac{7}{25}$,

$\lambda_3 = \frac{7}{25}$,

$\tau_3 = \frac{7}{25}$,

$\breve{l}_3 = (-\frac{1}{4})$,

$$t_3^T = \begin{pmatrix} -\frac{1203}{20000} & -\frac{1203}{2500} \end{pmatrix},$$

Output

$$L_3 = \begin{pmatrix} 2 & 0 & 0 \\ -\frac{7}{4} & \frac{25}{32} & 0 \\ 0 & -\frac{3}{4} & \frac{7}{25} \\ 0 & 0 & -\frac{1}{4} \end{pmatrix} \qquad \textit{and} \qquad T_3 = \begin{pmatrix} 2 & 0 & 0 \\ -\frac{423}{256} & \frac{25}{32} & 0 \\ -\frac{1203}{20000} & -\frac{1203}{2500} & \frac{7}{25} \end{pmatrix}$$

Step 3 $A_4 = (0 \quad 0 \quad -\frac{7}{4} \quad 2)$,

$$l_4 = \begin{pmatrix} 0 \\ 0 \\ -\frac{1}{4} \end{pmatrix}$$

$$\hat{l}_4 = \begin{pmatrix} 0 \\ 0 \\ -\frac{25}{4} \end{pmatrix}$$

$\mu_4 = \frac{7}{16}$

$\lambda_4 = \frac{7}{16}$

$\tau_4 = \frac{7}{16}$

$t_4^T = \begin{pmatrix} \frac{477}{1600} & \frac{477}{200} & \frac{159}{64} \end{pmatrix}$

Output

$$L_4 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -\frac{7}{4} & \frac{25}{32} & 0 & 0 \\ 0 & -\frac{3}{4} & \frac{7}{25} & 0 \\ 0 & 0 & -\frac{1}{4} & \frac{7}{16} \end{pmatrix} \quad \textit{and} \quad T_4 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -\frac{423}{256} & \frac{25}{32} & 0 & 0 \\ -\frac{1203}{20000} & -\frac{1203}{2500} & \frac{7}{25} & 0 \\ \frac{477}{1600} & \frac{477}{200} & \frac{159}{64} & \frac{7}{16} \end{pmatrix}$$

EXAMPLE **3.2.2** *Let us consider the Door's matrix 4-by-4*

$$A = \begin{pmatrix} 2 & -7/4 & 0 & 0 \\ -1/4 & 1 & -3/4 & 0 \\ 0 & -3/4 & 1 & -1/4 \\ 0 & 0 & -7/4 & 2 \end{pmatrix},$$

with application of the algorithm 3.2.2.

Resolution:

Step 0 $A_1 = (2 \quad -\frac{7}{4} \quad 0 \quad 0)$,

$\bar{\tau}_1 = \alpha_1 = 2$,

$\lambda_1 = 1$,

$L(2:4,1) = (-\frac{7}{8} \quad 0 \quad 0)^T$,

Output

$$L_1 = \begin{pmatrix} 2 \\ -\frac{7}{8} \\ 0 \\ 0 \end{pmatrix} \quad \textit{and} \quad \bar{T}_1 = (2)$$

Step 1 $A_2 = (-\frac{1}{4} \quad 1 \quad -\frac{3}{4} \quad 0)$,

$l_2 = (-\frac{7}{8})$,

$\hat{l}_2 = (-\frac{1}{4})$,

$\mu_2 = \frac{25}{32}$,

$\lambda_2 = \frac{1393}{1576}$,

$\bar{\tau}_2 = 1$,

$\breve{l}_2 = \begin{pmatrix} -\frac{605}{713} \\ 0 \end{pmatrix}$,

$t_2^T = (\frac{5}{8})$,

Output

$$L_2 = \begin{pmatrix} 2 & 0 \\ -\frac{7}{8} & \frac{1393}{1576} \\ 0 & -\frac{605}{713} \\ 0 & 0 \end{pmatrix} \qquad \textit{and} \qquad \bar{T}_2 = \begin{pmatrix} 2 & 0 \\ \frac{5}{8} & 1 \end{pmatrix}$$

Step 2 $A_3 = (0 \quad -\frac{3}{4} \quad 1 \quad -\frac{1}{4})$,

$l_3 = \begin{pmatrix} 0 \\ -\frac{605}{713} \end{pmatrix}$,

$\hat{l}_3 = \begin{pmatrix} 0 \\ -\frac{605}{713} \end{pmatrix}$,

$\mu_3 = \frac{7}{25}$,

$\lambda_3 = \frac{1071}{2024}$,

$\bar{\tau}_3 = 1$,

$\breve{l}_3 = (-\frac{506}{1071})$,

$t_3^T = \begin{pmatrix} 0 & 0 \end{pmatrix}$,

Output

$$L_3 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{7}{8} & \frac{1393}{1576} & 0 \\ 0 & -\frac{605}{713} & \frac{1071}{2024} \\ 0 & 0 & -\frac{506}{1071} \end{pmatrix} \qquad \textit{and} \qquad \bar{T}_3 = \begin{pmatrix} 2 & 0 & 0 \\ \frac{5}{8} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Step 3 $A_4 = (0 \quad 0 \quad -\frac{7}{4} \quad 2)$,

$$l_4 = \begin{pmatrix} 0 \\ 0 \\ -\frac{506}{1071} \end{pmatrix}$$

$$\hat{l}_4 = \begin{pmatrix} 0 \\ 0 \\ -\frac{506}{153} \end{pmatrix}$$

$\mu_4 = \frac{7}{16}$

$\lambda_4 = \frac{506}{765}$

$\bar{\tau}_4 = 1$

$t_4^T = \begin{pmatrix} -\frac{9}{2} & -\frac{36}{7} & -\frac{75}{14} \end{pmatrix}$

Output

$$L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{7}{8} & \frac{1393}{1576} & 0 & 0 \\ 0 & -\frac{605}{713} & \frac{1071}{2024} & 0 \\ 0 & 0 & -\frac{506}{1071} & \frac{506}{765} \end{pmatrix} \qquad \textit{and} \qquad \bar{T}_4 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ \frac{5}{8} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{9}{2} & -\frac{36}{7} & -\frac{75}{14} & 1 \end{pmatrix}$$

Chapter 4

Numerical Tests

In the section 4.1, we will make brief comments on the test matrices used in our work. In the section 4.2, we will summarize the comparison results among the algorithm (3.2.3), the algorithms *ST*, *MST* (Modified *ST*) algorithm developed by Santiago and Yuan in 2001 (see [35, 36]), *LU* Decomposition without pivoting (see [13], page 163), *Cholesky* Decomposition (see [13], page 174) and *QR* Decomposition (see [13], page 209). In the comparisons, we will display the machine using time and the relative error.

All tests were made in computers Pentium (III), 800 MHz, 512 MB of RAM memory, Mainboard PC CHIPS 748 and HD with 20 GB.

The relative errors were analyzed in the following forms:

- For the algorithm (3.2.3) (New *ST II*)

$$error = \frac{\|A - TLL^T\|_F}{\|A\|_F}$$

- For *ST* and *MST* Decomposition

$$error = \frac{\|A - T^{-1}LL^T\|_F}{\|A\|_F}$$

- For LU Decomposition

$$error = \frac{\|A - LU\|_F}{\|A\|_F}$$

- For *Cholesky* Decomposition

$$error = \frac{\|A - LL^T\|_F}{\|A\|_F}$$

- For QR Decomposition

$$error = \frac{\|A - QR\|_F}{\|A\|_F}$$

4.1 Matrices Test

4.1.1 Circulant Matrix

$C = circul(V)$ is the circulant matrix (see [6]) whose first row is V. (A circulant matrix has the property that each row is obtained from the previous one by cyclically permuting the entries one step forward; it is a special Toeplitz matrix in which the diagonals 'wrap round'.)

Special case: if V is a scalar the $C = circul(1 : V)$.

The eigensystem of C (n-by-n) is known explicitly. If t is an nth root of unity, then the inner product of V with $W = [1 \quad t \quad t^2 \quad \ldots \quad t^n]$ is an eigenvalue of C, and $W(n : -1 : 1)$ is an eigenvector of C.

MATLAB CODE

```
function C = circul(v)
n = max(size(v));
if   n==1

     n = v;
     v = 1:n;
```

```
end
v = v(:).';              % Make sure v is a row vector.
C = toeplitz( [ v(1)    v(n:-1:2) ], v );
```

4.1.2 Door's Matrix

Door matrix (see [8]) - diagonally dominant, ill-conditioned, tridiagonal. $[C, D, E] = dorr(n, \theta)$ returns the vectors which defines a row diagonally dominant, tridiagonal M-matrix that is ill-conditioned for small values of the parameter $\theta \geq 0$.

If only one output parameter is supplied then $C = full(tridiag(C, D, E))$, i.e., the matrix itself is returned.

The columns of $inv(C)$ vary greatly in norm. θ defaults to 0.01. The amount of diagonal dominance is given by (ignoring routing errors):

$$comp(C) * ones(n, 1) = \theta * (n + 1)^2 * [1 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 1]'.$$

MATLAB CODE

```
function [c, d, e] = dorr(n, theta)

if   nargin < 2,    theta = 0.01;    end
c = zeros(n, 1);       e = c;       d = c;
% All length n for convenience. Make c, e of length n-1 later.

h = 1/(n+1);
m = floor( (n+1)/2 );
term = theta/h^2;

i = (1:m)';

    c(i) = -term*ones(m,1);
    e(i) = c(i) - (0.5-i*h)/h;
    d(i) = -(c(i) + e(i));
```

```
i = (m+1:n)';

    e(i) = -term*ones(n-m,1);
    c(i) = e(i) + (0.5-i*h)/h;
    d(i) = -(c(i) + e(i));

c = c(2:n);
e = e(1:n-1);


if  nargout <= 1

    c = tridiag(c, d, e);

end
```

4.1.3 Hilbert's Matrix

Hilbert matrix (see [24, 4, 30, 22]). hilb(n) is the n-by-n matrix with elements $1/(i+j-1)$. It is a famous example of a ill-conditioned matrix. cond(hilb(n)) grows exponentially as EXP(3.5*n).

MATLAB CODE

```
function H = hilb(n)
if  n == 1

    H = 1;

else

    H = cauchy( (1:n) - .5 );

end
```

4.1.4 Moler Matrix

Moler matrix (see [29]) - symmetric positive definite. $A = moler(n, alpha)$ is the symmetric and positive definite n-by-n matrix $U' * U$ where $U = triw(n, alpha)$. For $alpha = -1$ (the default) $A(i,j) = min(i,j) - 2$, $A(i,i) = i$.

MATLAB CODE

```
function A = moler(n, alpha)
if  nargin == 1,  alpha = -1;  end
A = triw(n, alpha)'*triw(n, alpha);
```

4.1.5 Pie's Matrix

Pie's matrix A (see [32]) with $a_{ii} = \alpha$, $a_{ij} = 1$ for $i \neq j$. The matrix becomes ill-conditioned when α is close to 1. For example, when $\alpha = 0.9999$ (the default) and $n = 5$, $Cond(A) = 5 \times 10^4$.

MATLAB CODE

```
function A=piematrix(n,alpha)
if  nargin == 1,  alpha = .9999;  end
for  i = 1:n
    for  j = 1:n
        if  i==j
            A(i,j) = alpha;
        else
            A(i,j) = 1;
        end
    end
end
```

4.1.6 Poisson's Matrix

Block tridiagonal matrix from Poisson's equation (sparse). The poisson(n) is a block tridiagonal matrix of order n^2 resulting from discretizing Poisson's equation with the 5-point operator on an n-by-n mesh (see [16], section 4.5.4).

MATLAB CODE

```
function A=poisson(n)
S = tridiag(n, -1, 2, -1);
I = speye(n);
A = kron(I,S) + kron(S,I);
```

4.1.7 Prolate Matrix

Prolate matrix - symmetric (see [42]), ill-conditioned Toepliz matrix. The $A = prolate(n, w)$ is the n-by-n prolate matrix with parameter w.

If $0 < w < 0.5$ then

- A is positive definite;
- The eigenvalues of A are distinct, lie in $(0, 1)$, and tend to cluster around 0 and 1.

The w defaults to 0.25.

MATLAB CODE

```
function A=prolate(n, w)
if  nargin == 1,  w = 0.25;  end
a = zeros(n,1);
a(1) = 2*w;
a(2:n) = sin( 2*pi*w*(1:n-1) ) ./ ( pi*(1:n-1) );
A = toeplitz(a);
```

4.1.8 Tridiagonal Matrix

Tridiagonal matrix (sparse)(see [34, 41]). The $tridiag(x, y, z)$ is the tridiagonal matrix with subdiagonal x, diagonal y and superdiagonal z. The x and z are vectors whose dimension is one less than that of y. Alternatively $tridiag(n, c, d, e)$, where c, d and e are all scalars, yields the Toeplitz tridiagonal matrix of order n with subdiagonal elements c, diagonal elements d and superdiagonal elements e. This matrix has eigenvalues (Todd 1977)

$$d + 2 * sqrt(c * e) * cos(k * pi/(n + 1)), k = 1 : n.$$

The $tridiag(n)$ is the same as $tridiag(n, -1, 2, -1)$, which is a symmetric positive definite M-matrix (the negative of the second difference matrix).

MATLAB CODE

```
function A=tridiag(n, x, y, z)
if  nargin == 1,   x = -1;   y = 2;   z = -1;   end
if  nargin == 3,   z = y;    y = x;   x = n;    end

x = x(:);   y = y(:);   z = z(:);      % Force column vector.

if max( [ size(x) size(y) size(z) ] ) == 1

    x = x*ones(n-1,1);
    z = z*ones(n-1,1);
    y = y*ones(n,1);

else

    [nx, m] = size(x);
    [ny, m] = size(y);
    [nz, m] = size(z);
    if  (ny - nx - 1) | (ny - nz - 1)
```

```
            error('Dimensions of vector arguments are incorrect.')
        end
    end
% T = diag(x, -1) + diag(y) + diag(z, 1);    % For non-sparse matrix.
n = max(size(y));
T = spdiags([ [x;0] y [0;z] ], -1:1, n, n);
```

4.1.9 Wathen's Matrix

Whaten's matrix (see [43]) is a finite element matrix (sparse, random entriens). The $A = wathen(nx, ny)$ is a sparse random n-by-n finite element matrix where $n = 3.nx.ny + 2.nx + 2.ny + 1$. A is precisely the consistent mass matrix for a regular nx-by-ny grid of 8-node (serendipity) elements in 2 space dimensions. A is symmetric positive definite for any (positive) values of the density, $rho(nx, ny)$, which is chosen randomly in this routine. In particular, if $D = diag(diag(A))$, then $0.25 \leq eig(inv(D)A) \leq 4.5$ for any positive integer nx and ny and densities $rho(nx, ny)$. This diagonally scaled matrix is returned by $wanthen(nx, ny, 1)$.

MATLAB CODE

```
function A = wathen(nx, ny, k)
if  nargin < 2,  error('Two dimensioning arguments must be specified.'),  end
e1 = [6  -6  2  -8;-6  32  -6  20;2  -6  6  -6;-8  20  -6  32];
e2 = [3  -8  2  -6;-8  16  -8  20;2  -8  3  -8;-6  20  -8  16];
e = [e1  e2;  e2'  e1]/45;
n = 3*nx*ny+2*nx+2*ny+1;
A = sparse(n,n);
RHO = 100*rand(nx,ny);
for  j = 1:ny
    for  i = 1:nx
```

```
            nn(1) = 3*j*nx + 2*i + 2*j + 1;
            nn(2) = nn(1) - 1;
            nn(3) = nn(2) - 1;
            nn(4) = (3*j - 1)*nx + 2*j + i - 1;
            nn(5) = 3*(j - 1)*nx + 2*i + 2*j - 3;
            nn(6) = nn(5) + 1;
            nn(7) = nn(6) + 1;
            nn(8) = nn(4) + 1;
            em = e*RHO(i, j);
                for   krow = 1:8
                    for   kcol = 1:8
                        A(nn(krow),nn(kcol)) = A(nn(krow),nn(kcol))
                        + em(krow,kcol);
                    end
                end
        end
    end
    if   k == 1
       A = diag(diag(A)) \ A;
    end
```

4.2 Numeric Results

In this section we will present the numerical results obtained with algorithms described previously.

The tables 4.1, 4.2, and 4.3 represent the numerical results for the Circulating Matrix.

The tables 4.4, 4.5, and 4.6 represent the numeric results for the Door's Matrix.

Decomposition	CPU time	Relative Error
NST	0.5810	4.5743e-14
ST	0.7910	2.3752e-12
MST	0.9620	2.8104e-13
LU	0.3200	1.0355e-15
QR	1.8030	1.4630e-15

Table 4.1: Circulante Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.7950	5.9004e-13
ST	31.5560	4.1408e-11
MST	29.5530	4.0434e-12
LU	4.7870	2.0809e-15
QR	54.7090	2.7197e-15

Table 4.2: Circulante Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	86.8550	1.0011e-12
ST	144.3170	1.4416e-10
MST	143.7860	1.7749e-11
LU	32.5170	2.7667e-15
QR	268.9060	4.5613e-15

Table 4.3: Circulante Matrix, order 500.

Decomposition	CPU time	Relative Error
NST	0.5600	0
ST	0.8010	3.2568e-13
MST	0.9110	4.9038e-13
LU	0.2710	0
QR	1.7920	5.4334e-16

Table 4.4: Door's Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8660	0
ST	28.4710	9.4164e-13
MST	29.3520	9.5221e-13
LU	4.7970	0
QR	54.7890	4.7733e-16

Table 4.5: Door's Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	86.1040	0
ST	137.5680	1.1941e-12
MST	143.0260	1.1514e-12
LU	39.0060	0
QR	270.3890	4.7904e-16

Table 4.6: Door's Matrix, order 500.

The tables 4.7, 4.8, and 4.9 represent the numerical results for the Hilbert's Matrix.

Remark: This matrix is theoretically symmetric and positive definite, but presents numerical problems due to its ill-conditioned.

The tables 4.10, 4.11, and 4.12 represent the numerical results for the Moler Matrix.

The tables 4.13, 4.14, and 4.15 represent the numerical results for the Pie's Matrix.

The tables 4.16, 4.17, and 4.18 represent the numerical results for the Poisson's Matrix.

The tables 4.19, 4.20, and 4.21 represent the numerical results for the Prolate

Decomposition	CPU time	Relative Error
NST	0.5200	1.0610e-09
ST	0.8910	1.8508e-04
MST	0.8210	3.8937e-07
LU	0.3600	1.3013e-16
Cholesky	******	******
QR	1.8230	2.2976e-15

Table 4.7: Hilbert's Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8460	1.4987e-08
ST	30.4040	1.1684e+03
MST	30.2730	2.6894e-05
LU	4.7970	2.0539e-16
Cholesky	******	******
QR	54.7180	4.0471e-15

Table 4.8: Hilbert's Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	57.4830	4.0805e-08
ST	94.4760	5.7171e+04
MST	94.9960	1.9576e-04
LU	12.6780	2.3682e-16
Cholesky	******	******
QR	175.5220	1.7678e-15

Table 4.9: Hilbert's Matrix, order 437.

Decomposition	CPU time	Relative Error
NST	0.5510	0
ST	0.8810	0
MST	0.8410	0
LU	0.3200	0
Cholesky	0.3600	0
QR	1.7520	7.7197e-15

Table 4.10: Moler Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8560	0
ST	29.5820	0
MST	29.5020	0
LU	4.7670	0
Cholesky	3.9050	0
QR	55.0890	2.3728e-14

Table 4.11: Moler Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	86.8750	0
ST	144.2080	0
MST	144.8580	0
LU	64.5330	0
Cholesky	12.9890	0
QR	273.8930	4.8580e-14

Table 4.12: Moler Matrix, order 500.

Decomposition	CPU time	Relative Error
NST	0.5510	3.4894e-16
ST	0.8320	9.2500e-15
MST	0.8310	9.2500e-15
LU	0.2700	2.5755e-16
QR	1.7430	1.1739e-15

Table 4.13: Pie's Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8160	5.6284e-16
ST	29.5130	2.6345e-14
MST	29.5320	2.6345e-14
LU	4.7270	6.3363e-16
QR	54.6290	9.5640e-15

Table 4.14: Pie's Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	86.9150	6.6973e-16
ST	144.4780	3.3878e-14
MST	143.9270	3.3878e-14
LU	80.4760	7.7635e-16
QR	268.8470	2.1778e-14

Table 4.15: Pie's Matrix, order 500.

Decomposition	CPU time	Relative Error
NST	0.4910	4.1372e-17
ST	0.8510	8.7313e-17
MST	0.8920	1.0400e-16
LU	0.2700	9.1007e-17
Cholesky	0.3700	7.9393e-17
QR	1.8020	9.4584e-16

Table 4.16: Poisson's Matrix, order 100 ($n = 10$).

Decomposition	CPU time	Relative Error
NST	22.6620	6.0286e-17
ST	36.2320	1.1019e-16
MST	36.3920	1.2116e-16
LU	5.8390	1.5524e-16
Cholesky	4.7360	7.5996e-17
QR	69.2990	1.1766e-15

Table 4.17: Poisson's Matrix, order 324 ($n = 18$).

Decomposition	CPU time	Relative Error
NST	102.3070	6.9183e-17
ST	164.6770	1.2472e-16
MST	164.7270	1.2960e-16
LU	81.0060	1.6950e-16
Cholesky	15.1010	7.9462e-17
QR	322.7850	1.2365e-15

Table 4.18: Poisson's Matrix, order 529 ($n = 23$).

Decomposition	CPU time	Relative Error
NST	0.5100	1.8815e-07
ST	0.8310	1.4000e-03
MST	0.9310	9.7909e-05
LU	0.3200	1.1156e-15
Cholesky	******	******
QR	1.6930	1.2364e-15

Table 4.19: Prolate Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8650	3.8153e-06
ST	29.3920	1.4300e-02
MST	29.6530	8.4158e-05
LU	4.9170	9.9488e-16
Cholesky	******	******
QR	54.8890	2.1301e-15

Table 4.20: Prolate Matrix, order 300.

Matrix.

The tables 4.22, 4.23, and 4.24 represent the numerical results for the Tridiagonal Matrix.

The tables 4.25, 4.26, and 4.27 represent the numerical results for the Wathen's Matrix.

Decomposition	CPU time	Relative Error
NST	86.8950	3.6374e-06
ST	143.7560	4.0400e-02
MST	144.3780	3.8000e-03
LU	20.5590	1.0335e-15
Cholesky	******	******
QR	272.9720	2.5952e-15

Table 4.21: Prolate Matrix, order 500.

Decomposition	CPU time	Relative Error
NST	0.5500	6.4206e-18
ST	0.8310	7.5151e-17
MST	0.8910	6.7054e-17
LU	0.2800	0
Cholesky	0.3610	7.8897e-17
QR	1.7320	4.7948e-16

Table 4.22: Tridiagonal Matrix, order 100.

Decomposition	CPU time	Relative Error
NST	17.8860	4.5350e-18
ST	28.5310	7.9330e-17
MST	29.0220	6.8183e-17
LU	4.6970	0
Cholesky	3.9650	8.0531e-17
QR	54.6080	4.7447e-16

Table 4.23: Tridiagonal Matrix, order 300.

Decomposition	CPU time	Relative Error
NST	86.4640	3.5120e-18
ST	138.8600	7.6569e-17
MST	138.6100	7.0215e-17
LU	67.4070	0
Cholesky	13.1090	7.7317e-17
QR	272.7530	4.7657e-16

Table 4.24: Tridiagonal Matrix, order 500.

Decomposition	CPU time	Relative Error
NST	0.4300	7.4213e-17
ST	0.7520	1.1465e-16
MST	0.7620	1.1465e-16
LU	0.2500	1.2258e-16
Cholesky	0.3500	6.4657e-17
QR	1.5020	7.9502e-16

Table 4.25: Wathen's Matrix, order 96 ($nx = ny = 5$).

Decomposition	CPU time	Relative Error
NST	26.7580	8.7363e-17
ST	42.5810	1.3154e-16
MST	42.6910	9.7096e-17
LU	6.4800	1.6593e-16
Cholesky	5.2980	7.5805e-17
QR	81.8780	1.2533e-15

Table 4.26: Wathen's Matrix, order 341 ($nx = ny = 10$).

Decomposition	CPU time	Relative Error
NST	121.7850	8.8296e-17
ST	197.1640	1.1065e-16
MST	198.0650	1.0833e-16
LU	87.8760	1.6299e-16
Cholesky	17.3950	7.3758e-17
QR	392.4940	1.2331e-15

Table 4.27: Wathen's Matrix, order 560 ($nx = ny = 13$).

Chapter 5

Conclusions and future works

The central objective of this work is to propose two new algorithms for the ST decomposition proposed firstly by Golub and Yuan in [18].

Our numerical experiments show that our new algorithms *NST* (New ST Decomposition) are more efficient and cheaper than Golub-Yuan Algorithm and *MST* Algorithm (Santiago and Yuan [35]) because they give smaller relative error and less computational time.

Compared with the QR Decomposition our algorithms are always cheaper in terms of computational time, and with smaller relative error in many cases.

Compared with the LU Decomposition the *NST* Decomposition loses in computation time, because it needs more operations. However it is important to observe that in some cases our algorithms give smaller relative error.

For the *Cholesky* Decomposition our method loses a lot in computation time. But it has an excellent performance for ill-conditioned matrices with which the *Cholesky* Decomposition does not work.

Therefore, it follows our analysis and numerical tests that the *NST* Decomposition posses well computational performance even compared with existing matrix decompositions since it is new-born decomposition. Also the new algorithms require only one row at each step which can treat very huge matrices. This version can also be generalized to column version. Of course it still requires more improvements and

some applications. Hence our suggested future work are as follows.

Future works suggestions

- To apply the *NST* Decomposition to improve the solution of the system $Ax = b$ using the iterative refinement.
- To research the possibility with pivoting technique for the *NST* method to improve its numerical stability.
- To develop the incomplete decomposition for sparse matrices with strategy of fill-in reduction.
- To apply the decomposition for solving nonsymmetric generalized eigenvalue problems.
- To give roundoff error analysis and to do more tests.
- To improve the *NST* Algorithms further.

Appendix A

The codes for ST Decomposition

Implementation of the Algorithm (2.4.1)

```
function [L, T] = STyuan(A)

%   ST Decomposition
%       This routine decomposes the matrix A in the form T*A=S=L*L'
%       Where,
%           T is lower triangular matrix, L is the factor of Cholesky, and
%           S is symmetric positive definite matrix.

n = length(A);
T(1,1) = A(1,1);
L(1,1) = sqrt(T(1,1)*A(1,1));

t0 = clock;
for   k = 1:n-1

      L(k+1,1:k) = (L(1:k,1:k)\(T(1:k,1:k)*A(1:k,k+1)))';
      LL = L(1:k,1:k)\A(k+1,1:k)';
```

```
        s = A(k+1,k+1) - L(k+1,1:k)*LL;

            if    abs( s ) < 0.1e-18

                T(k+1,k+1) = 1;

            else

                T(k+1,k+1) = sign(s);

            end

        L(k+1,k+1) = sqrt( T(k+1,k+1)*s );
        T(k+1,1:k) = (T(1:k,1:k)'*(L(1:k,1:k)'\(L(k+1,1:k)'-T(k+1,k+1)*LL)))';

end
cpu_time = etime(clock,t0)                          % verify the cputime
error_st = norm(A-T\L*L','fro')/norm(A,'fro')       % compute the error
```

Implementation of the Algorithm MST

(For more details to see [35])

```
function mst

%   Modified ST Decomposition
%       This routine decomposes the matrix A in the form T*A=S=L*L'
%       which actualization of the elements of the diagonal of T.
%       Where,
%           T is lower triangular matrix, L is the factor of Cholesky, and
%           S is symmetric positive definite matrix.

[A,OP,op,eta] = testes;      % Function with test matrices

n = length(A);
T(1,1) = A(1,1);
L(1,1) = sqrt(T(1,1)*A(1,1));
tic
for   k = 1:n-1

      L(k+1,1:k) = (L(1:k,1:k)\(T(1:k,1:k)*A(1:k,k+1)))';
      LL = L(1:k,1:k)\A(k+1,1:k)';
      s = A(k+1,k+1) - L(k+1,1:k)*LL;
      if   abs( s ) < 0.1e-18 % tolerance to avoid break-down problems

           T(k+1,k+1) = 1;

      else

           T(k+1,k+1) = sign( s )*eta;
           % Take the matrix an makes the appropriate updating
           if op==2 | op==15
```

```
            eta = norm(L(k+1,1:k)/2*k);
        elseif op==5 | op==13 | op==14 | op==21
            eta = norm(L(k+1,:),1);
        elseif op==1 | op==18
            eta = norm(L(k+1,:));
        elseif op==8 | op==17 | op==28
            eta = 2;
        end
    end
    L(k+1,k+1) = sqrt( T(k+1,k+1)*s );
    T(k+1,1:k) = (T(1:k,1:k)'*(L(1:k,1:k)'\(L(k+1,1:k)'-T(k+1,k+1)*LL)))';
end
tST = toc                                          % verify the cputime
erSTr = norm(A-T\L*L','fro')/norm(A,'fro')         % compute the error
```

Implementation of the Algorithm (3.2.3) (New *ST II*)

```
function [L, T] = NST(A)

%   New ST Decomposition
%       This routine decomposes the matrix A in the form A=T*S=T*L*L'
%       which actualization of the elements of the diagonal of L.
%       Where,
%           T is lower triangular matrix, L is the factor of Cholesky, and
%           S is symmetric positive definite matrix.

n = length(A);
T(1,1) = A(1,1);
L(1,1) = 1;
L(2:n,1) = A(1,2:n)'/A(1,1);

t0 = clock;
for   k = 1:n-1

      lk_1 = L(k+1,1:k)';
      lk_1hat = L(1:k,1:k)\A(k+1,1:k)';
      mu = A(1+k,1+k) - lk_1'*lk_1hat;

      if   abs(mu) > 1

           L(k+1,k+1) = 1;
           T(k+1,k+1) = mu;
           L(k+1:n,k+1) = (A(1+k,1+k:n)' - L(k+1:n,1:k) * lk_1hat)/mu;

      else

           L(k+1,k+1) = sqrt(abs(mu));
```

```
        T(k+1,k+1) = sign(mu);
        L(k+1:n,k+1) = sign(mu)*(A(1+k,1+k:n)' - L(k+1:n,1:k) * lk_1hat)/
        ...L(k+1,k+1);

    end

    T(k+1,1:k) = ((L(1:k,1:k)'\(lk_1hat-T(k+1,k+1)*L(k+1,1:k)')))';

end
cpu_time = etime(clock,t0)                          % verify the time
error_nst = norm(A-T*L*L','fro')/norm(A,'fro')      % compute the error
```

Appendix B

Solution of Triangular System

Consider a system

$$Ty = b \tag{B.1}$$

where $T = (t_{ij})_{n\times n}$ is a nonsingular upper triangular matrix and $y = (y_1, y_2, \ldots, y_n)^T$. That is

$$\begin{array}{ccccccccccc}
t_{11}y_1 & + & t_{12}y_2 & + & t_{13}y_3 & + & \cdots & + & t_{1n}y_n & = & b_1 \\
 & & t_{22}y_2 & + & t_{23}y_3 & + & \cdots & + & t_{2n}y_n & = & b_2 \\
 & & & & t_{33}y_3 & + & \cdots & + & t_{3n}y_n & = & b_3 \\
 & & & & & & \ddots & & \cdots & \vdots & \\
 & & & & & & t_{n-1,n-1}y_{n-1} & + & t_{n-1,n}y_n & = & b_{n-1} \\
 & & & & & & & & t_{n,n}y_n & = & b_n
\end{array} \tag{B.2}$$

where each $t_{ii} \neq 0$, for $i = 1, 2, \ldots, n$.

By back substitution, there is the following algorithm,

ALGORITHM B.0.1 *Back Substitution*

Input Data: $T = (t_{ij})$, *an* $n \times n$ *nonsingular upper triangular matrix, and* b *an* n*-vector*

Step 1: Compute $y_n = \frac{b_n}{t_{nn}}$

Step 2: Compute y_{n-1} *through* y_1 *successively:*

$$y_i = \frac{1}{t_{ii}} \left(b_i - \sum_{j=i+1}^{n} t_{ij} y_j \right), \qquad i = n-1, \ldots, 2, 1$$

Output Data: $y = (y_1, y_2, \ldots, y_n)^T$

Pseudocodes

For $i = n, n-1, \ldots, 3, 2, 1$ do

$$y_i = \frac{1}{t_{ii}} \left(b_i - \sum_{j=i+1}^{n} t_{ij} y_j \right)$$

Note: When $i = n$, the summation ($\sum$) is skipped.

For lower triangular system

$$Lx = b, \tag{B.3}$$

we have.

ALGORITHM B.0.2 *Forward Elimination*

For $i = 1, 2, \ldots, n$ *do*

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right)$$

Bibliography

[1] M. Arioli, I.S. Duff and P.P.M. de Rijk, *On the Augmented System Approach to Sparse Least-Squares Problems*, Numer. Math., 55(1989) 667-684.

[2] K.E. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed, John Wiley & Sons, USA, 1989.

[3] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[4] M.D. Choi, *Tricks or treats with the Hilbert matrix*, Amer. Math. Monthly, 90 (1983), pp. 301-312.

[5] B.N. Datta, *Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Company, California 93950, USA, 1994.

[6] P.J. Davis, *Circulante Matrices*, John Wiley, 1977.

[7] J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[8] F.W. Door, *An example of ill-conditioning in the numerical solution of singular perturbation problems*, Math. Comp., 25 (1971), pp. 271-283.

[9] H. Elman and D. Silvester, *Fast Nonsymmetric Iterations and Preconditioning for Navier-Stokes equations*, SIAM J. Sci. Comput., 17(1996) 33-46.

[10] H. Elman, D.J. Silvester and A. Wathen, *Iterative Methods for Problems in Computatational Fluid Dynamics, Iterative Methods in Scientific Computing*,

R. Chan, T. Chan and G. Golub eds., Springer-Verlag, Singapore, 1997, pp. 271-327.

[11] L. Elsner, A. Fasse, and E. Langmann, *A divide-and-conquer method for the tridiagonal generalized eigenvalue problem*, JCAM, 86(1997) 141-148.

[12] D.J. Evans and J. Shanehchi, *Preconditioned iterative methods for the generalized eigenvalue problem with large sparse matrices*, Preconditioning Methodes: Analysis and Appications, 379-400, Topics in Comput. Math., 1. Corden & Breach, New York, 1983.

[13] L.V. Fausett. *Applied Numerical Analysis Using Matlab*, Prentice-Hall, Inc., ISBN: 0-13-319849-9, 1999.

[14] B. Fischer, A. Ramage, D.J. Silvester and A.J. Wathen, *Minimum Residual Methods for Augmented Systems*, BIT, 38(1998) 527-543.

[15] R. Fletcher and D.F. Griffiths, *The generalized eigenvalue problem for certain unsymmetric band matrices*, LAA, 29(1980) 139-149.

[16] G.H. Golub and C. Greif, *Techniques for solving KKT systems with ill-conditioned or singula (1,1) block*, BIT 40th Meeting, Lund, Sweden, 9-12, August, 2000.

[17] G.H. Golub and C.F. Van Loan, *Matrix Computation*, 3rd Edition, The Johns Hopkins University Press, Baltimore and London, 1996.

[18] G.H. Golub and J.Y. Yuan, *ST: Symmetric-Triangular Decomposition and its Applications Part I: Theorems and algorithms*, BIT *(accepted)*, 2000.

[19] G.H. Golub and J.Y. Yuan, *ST: Symmetric-Triangular Decomposition and its Applications Part II: Applications*, BIT 40th. Meeting, Lund, 2000.

[20] G.H. Golub, X. Wu and J.Y. Yuan, *SOR-Like Methods for Augmented Systems*, BIT, to appear.

[21] M.R. Hestenes and E. Stiefel, *Methods of conjugate gradient for solving linear systems*, J. Reserch of the National Bureau of Standasds, 49(1952) 409-436.

[22] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996; sec. 26.1.

[23] A.S. Householder, *The Theory of Matrices in Numerical Analysis*, Blaisdell Publishing Company/ A Division of Ginn And Company, New York, 1964.

[24] D.E. Knuth, *The Art of Computer Programming*, Volume 1, Fundamental Algorithms, second edition, Addison-Wesley, Reading, Massachusetts, 1973, p.37.

[25] C.L. Lawson, R.J. Ranson, *Solving Least Squares Problems*, Philadelphia: SIAM, 1995.

[26] C.K. Li and R. Mathias, *Generalized eigenvalues of a definite Hermitian matrix pair*, LAA, 271(1998) 309-321.

[27] D. Luenberger, *Introduction to Linear and Nonlinear Programming.* New York: Addison Wesley, 1973.

[28] S.G. Nash and A. Sofer, *Preconditioned Reduced Matrices*, SIAM J. Matrix Anal. Appl., 17(1996) 47-68.

[29] J.C. Nash, *Compact Numerical Mathods for Computer: Linear Algebra and Function Minimization*, second edition, Adam Hilger, Bristol, 1990 (Appendix 1).

[30] M. Newman and J. Todd, *The evaluation of matrix inversion programs*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 466-476.

[31] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems.* New York: Plenum Press, 1988a.

[32] M.L. Pei, *A test matrix for inversion procedures*, Comm. ACM, 5 (1962), p. 508.

[33] C. Rajakumar and C.R. Rogers, *The Lanczos algorithm applied to unsymmetric generalized eigenvalue problem*, Internat. J. Numer. Methods Engrg. 32(1991) 1009-1026.

[34] D.E. Rutherford, *Some continuant determinants arising in physics and chemistry II*, Proc. Royal Soc. Edin., 63, A (1952), pp. 232-241.

[35] C.D. Santiago, *Numerical Experiments For S&T Decomposition*, Master Dissertation, UFPR, Curitiba, Parana, Brazil, 2001.

[36] C.D. Santiago and J.Y. Yuan, *Modified ST Algorithms and Numerical Experiments*, Numerical Algorithms (accepted) 2002.

[37] C.H. Santos, B.P.B. Silva and J.Y. Yuan, *Block SOR Methods for Rank Deficient Least Squares Problems*, JCAM, 100(1998) 1-9.

[38] C.H. Santos and J.Y. Yuan, *Preconditioned Conjugate Gradient Methods for Rank Deficient Least Squares Problems*, Inter. J. Comput. Math., 72(1999) 509-518.

[39] D.C. Sorensen, *Truncated QR methods for large scale generalized eigenvalue problems*, ETNA, 7(1998) 141-162.

[40] G.W. Stewart, *Introduction to matrix computations*, San Diego, California: Academic Press, Inc., 1973.

[41] J. Todd, Basic Numerical Mathematics, Vol. 2: Numerical Algebra, Birkhauser, Basel, and Academic Press, New York, 1977, p. 155.

[42] J.M. Varah. *The Prolate Matrix.* Linear Algebra and Appl., 187:269-278, 1993.

[43] A.J. Wathen, *Realistic eigenvalue bounds for the Galerkin mass matrix*, IMA J. Numer. Anal., 7 (1987), pp. 449-457.

[44] S. Wright, *Stability of Augmented System Factorizations in Interior-point Methods*, SIAM J. Matrix Anal. Appl., 18(1997) 191-222.

[45] J.Y. Yuan, *Iterative Methods for Generalized Least Squares Problems*, Ph.D. Thesis, IMPA, Rio de Janeiro, Brazil, 1993.

[46] J.Y. Yuan, *Numerical Methods for Generalized Least Squares Problems*, JCAM, 66(1996) 571-584.

[47] J.Y. Yuan and A.N. Iusem, *Preconditioned Conjugate Gradient Method for Generalized Least Squares Problems*, JCAM, 71(1996) 287-297.

[48] J.Y. Yuan and A.N. Iusem, *Preconditioned SOR methods for Generalized Least Squares Problems*, Acta Mathematicae Appicadae Sinica 16(2000).

[49] J.Y. Yuan, G.H. Golub and R. Plemmons, *Left Conjugate Gradient Method for Nonsymmetric Linear Systems*, Technical Report, SCCM, Stanfor University, 1999.

[50] J.Y. Yuan and X.Q. Jin, *Direct Iterative Methods for Rank Deficient Generalized Least Squares Problems*, JCAM, 18(2000) 437-446.

[51] T. Zhang, K.H. Law, G.H Golub, *On the homotopy method for perturbed symmetric generalized eigenvalue problems*, SIAM J. Sci. Comput., 19(1998) 1625-1645.