Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER Applied Mathematics and Computation 175 (2006) 748–761

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

Row-wise algorithms of ST decomposition ☆

Celso José Cordeiro [a], Jin Yun Yuan [b,*]

[a] *Centro de Estudos em Engenharia Civil—CESEC, Universidade Federal do Paraná, Centro Politécnico, CP: 19.081, 81531-990, Curitiba, PR, Brazil*

[b] *Departamento de Matemática, Universidade Federal do Paraná, Centro Politécnico, CP: 19.081, 81531-990, Curitiba, PR, Brazil*

Abstract

New row-wise algorithms for the ST decomposition are established here. The new algorithms require just a row of A and two triangular solvers at each step instead of three triangular solvers in Golub–Yuan algorithms (BIT, 42 (2002) 814–822). Therefore, multiplication and storage requirements of the ST decomposition can greatly be saved in the row-wise algorithms. Some numerical comparisons are presented. It follows from numerical experiments that the new algorithms save at least 40% CPU time compared with other algorithms of the ST decomposition. In terms of numerical experiments, the numerical stability of the new algorithms is reasonable.
© 2005 Elsevier Inc. All rights reserved.

Keywords: ST decomposition; Golub–Yuan algorithm; Modified ST algorithm; *LU* factorization; Cholesky decomposition; Numerical stability; Row-wise ST algorithm

☆ The work of the second author was partially support by CNPq, Fundação Araucária, Instituto Milênio, Brazil.

* Corresponding author.
E-mail addresses: cordeiro@cesec.ufpr.br (C.J. Cordeiro), jin@mat.ufpr.br (J.Y. Yuan).

0096-3003/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2005.07.064

1. Introduction

Matrix decomposition is a very important tool in matrix computations and its applications. The LU decomposition is a well-known dynamic process to transform matrix A into a product of triangular matrices whenever the decomposition exists, that is, A can be numerically factored into lower and upper triangular matrices L and U such that

$$A = LU. \tag{1}$$

We can simplify the decomposition significantly by taking into account the symmetry for symmetric and positive definite matrices. This special decomposition is called Cholesky decomposition. The Cholesky decomposition constructs a lower triangular matrix L such that

$$A = LL^T. \tag{2}$$

Recently, a new matrix decomposition, ST decomposition, was proposed by Golub and Yuan [7,8]. By this decomposition, a nonsingular matrix A can be decomposed into a product of a triangular matrix and a symmetric positive definite matrix, that is, there exist lower triangular T and L such that $TA = LL^T$. Golub and Yuan have established two algorithms for the decomposition as follows:

Algorithm 1.1 (*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),
    $\lambda_{k+1} = \sqrt{\mu}$,
    $t_{k+1} = T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1}\hat{l}_{k+1})$.
```

Algorithm 1.2 (*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),
    $\lambda_{k+1} = \sqrt{\eta}$,
    $\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1}l_{k+1})$.
```

The numerical behavior of the Golub–Yuan algorithms was studied in [15,16]. Several types of matrices with different sizes were tested and compared with the LU (or Cholesky) decomposition without pivoting. From their test results, the ST decomposition gives reasonable numerical stability, specially for sparse matrices. Some modifications for Golub–Yuan algorithm were given in [16]. Golub–Yuan algorithms and their modifications [15,16] are expensive because at each step they need three triangular solvers. This is one of motivations for us to set up new cheap algorithm of the ST decomposition. Another motivation is that only one or several rows of matrix A are available in many real applications. Algorithms in [7,16] are block-wise which are not applicable in such cases. So row-wise algorithm is necessary. Therefore, our main contribution here is to propose some row-wise algorithms for the ST decomposition. The new row-wise algorithms are much cheaper than Golub–Yuan algorithms and their modification in [16] because at each step, only two triangular solvers are required. Meantime, some numerical tests and comparisons for several types of matrices are given. The rest of the paper is organized as follows. Three new row-wise algorithms of the ST decomposition are established in Section 2. Numerical test results are presented in Section 3. Some comments are given in the last section.

2. Row-wise algorithms

Assume that

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

and

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

It follows from $TA = LL^T$ in row-wise form that:

$$T_k A_k = L_k L_k^T, \tag{3}$$

$$T_k \tilde{a}_{k+1} = L_k l_{k+1}, \tag{4}$$

$$T_k \tilde{A}_k = L_k \tilde{L}_k^T, \tag{5}$$

$$t_{k+1}^T A_k + \tau_{k+1} a_{k+1}^T = l_{k+1}^T L_k^T, \tag{6}$$

$$t_{k+1}^T \tilde{a}_{k+1} + \tau_{k+1}\alpha_{k+1} = l_{k+1}^T l_{k+1} + \lambda_{k+1}^2, \tag{7}$$

$$t_{k+1}^T \tilde{A}_k + \tau_{k+1}\breve{a}_{k+1}^T = l_{k+1}^T \tilde{L}_k^T + \lambda_{k+1}\breve{l}_{k+1}^T. \tag{8}$$

From expression (6), we obtain

$$A_k^T t_{k+1} + \tau_{k+1} a_{k+1} = L_k l_{k+1}. \tag{9}$$

In terms of (9), there is

$$t_{k+1} = A_k^{-T}(L_k l_{k+1} - \tau_{k+1} a_{k+1}). \tag{10}$$

From (3), it follows that:

$$A_k^{-T} = T_k^T L_k^{-T} L_k^{-1}. \tag{11}$$

Substituting (11) in (10), we obtain

$$t_{k+1} = T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}). \tag{12}$$

It follows from (7) that:

$$\tilde{a}_{k+1}^T t_{k+1} + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2. \tag{13}$$

Substituting (12) in (13), we attain

$$\tilde{a}_{k+1}^T T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) + \tau_{k+1}\alpha_{k+1} = \|l_{k+1}\|_2^2 + \lambda_{k+1}^2. \tag{14}$$

In terms of (4), there is

$$\tilde{a}_{k+1}^T = l_{k+1}^T L_k^T T_k^{-T}. \tag{15}$$

Substituting (15) in (14), we obtain

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

that is,

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

Then, there is

$$\lambda_{k+1}^2 = \tau_{k+1}(\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}). \tag{16}$$

Note that τ_{k+1} should have the same sign as $(\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1})$, because $\lambda_{k+1}^2 > 0$.

It follows from (8) in (12) that:

$$\lambda_{k+1}\breve{l}_{k+1} = \tilde{A}_k^T T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1} L_k^{-1} a_{k+1}) + \tau_{k+1}\breve{a}_{k+1} - \tilde{L}_k l_{k+1}. \tag{17}$$

Since

$$\tilde{A}_k^T = \tilde{L}_k L_k^T T_k^{-T}, \tag{18}$$

we obtain

$$\breve{l}_{k+1} = \frac{\tau_{k+1}}{\lambda_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k L_k^{-1} a_{k+1}). \tag{19}$$

We can rewrite (16) in the following form:

$$\frac{\tau_{k+1}}{\lambda_{k+1}} = \frac{\lambda_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}}. \tag{20}$$

Substituting now (20) in (19), we obtain

$$\breve{l}_{k+1} = \lambda_{k+1} \frac{\breve{a}_{k+1} - \tilde{L}_k L_k^{-1} a_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}}. \tag{21}$$

In conclusion, a row-wise algorithm for the ST decomposition $TA = LL^T$ can be established as follows:

Algorithm 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
    $l_{k+1} = L(k+1, 1{:}k)^T$,
    $L_k = L(1{:}k, 1{:}k)$,
    $\tilde{L}_k = L(k+1:n, 1:k)$,
    $\alpha_{k+1} = A(k+1, k+1)$,
    $a_{k+1} = A(k+1, 1{:}k)^T$,
    $\breve{a}_{k+1} = A(k+1, k+1:n)^T$,
    $\hat{l}_{k+1} = L_k^{-1} a_{k+1}$,
    $\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T \hat{l}_{k+1}$,
Choose $\lambda_{k+1} > 0$ such that $LL^T = S$
    $\tau_{k+1} = \frac{\lambda_{k+1}^2}{\mu_{k+1}}$,
    $\breve{l}_{k+1} = \frac{\lambda_{k+1}}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1})$,
    $t_{k+1} = T_k^T L_k^{-T}(l_{k+1} - \tau_{k+1}\hat{l}_{k+1})$.
    $L(k+1, k+1) = \lambda_{k+1}$,
    $L(k+2:n, k+1) = \breve{l}_{k+1}$,
    $T_{k+1} = \begin{pmatrix} T_k & 0 \\ t_{k+1}^T & \tau_{k+1} \end{pmatrix}$.
```

Similarly, from the row-wise form of the decomposition $A = TLL^T$, it follows that:

$$T_k L_k L_k^T = A_k, \tag{22}$$

$$T_k L_k l_{k+1} = \tilde{a}_{k+1}, \tag{23}$$

$$T_k L_k \tilde{L}_k^T = \tilde{A}_k, \tag{24}$$

$$t_{k+1}^T L_k L_k^T + \tau_{k+1} l_{k+1}^T L_k^T = a_{k+1}^T, \tag{25}$$

$$t_{k+1}^T L_k l_{k+1} + \tau_{k+1}(l_{k+1}^T l_{k+1} + \lambda_{k+1}^2) = \alpha_{k+1}, \tag{26}$$

$$t_{k+1}^T L_k \tilde{L}_k^T + \tau_{k+1}(l_{k+1}^T \tilde{L}_k^T + \lambda_{k+1} \breve{l}_{k+1}^T) = \breve{a}_{k+1}^T, \tag{27}$$

from which we can easily obtain

$$l_{k+1} = L_k^{-1} T_k^{-1} \tilde{a}_{k+1}, \tag{28}$$

$$t_{k+1} = L_k^{-T}(L_k^{-1} a_{k+1} - \tau_{k+1} l_{k+1}), \tag{29}$$

$$\tau_{k+1}\lambda_{k+1}^2 = \alpha_{k+1} - a_{k+1}^T L_k^{-T} l_{k+1}, \tag{30}$$

$$\breve{l}_{k+1} = \lambda_{k+1} \frac{\breve{a}_{k+1} - \tilde{L}L_k^{-1} a_{k+1}}{\alpha_{k+1} - l_{k+1}^T L_k^{-1} a_{k+1}}. \tag{31}$$

Therefore, another row-wise algorithm for $A = TLL^T$ is proposed as follows:

Algorithm 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
    $l_{k+1} = L(k+1, 1{:}k)^T$,
    $L_k = L(1{:}k, 1{:}k)$,
    $\tilde{L}_k = L(k+1:n, 1:k)$,
    $\alpha_{k+1} = A(k+1, k+1)$,
    $a_{k+1} = A(k+1, 1{:}k)^T$,
    $\breve{a}_{k+1} = A(k+1, k+1:n)^T$,
    $\hat{l}_{k+1} = L_k^{-1} a_{k+1}$,
    $\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T \hat{l}_{k+1}$,
Choose $\lambda_{k+1} > 0$ such that $LL^T = S$
    $\bar{\tau}_{k+1} = \frac{\mu_{k+1}}{\lambda_{k+1}^2}$,
    $\breve{l}_{k+1} = \frac{\lambda_{k+1}}{\mu_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k \hat{l}_{k+1})$,
    $\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1} l_{k+1})$,
    $L(k+1, k+1) = \lambda_{k+1}$,
    $L(k+2:n, k+1) = \breve{l}_{k+1}$,
    $T_{k+1} = \begin{pmatrix} T_k & 0 \\ \dot{t}_{k+1}^T & \bar{\tau}_{k+1} \end{pmatrix}$.
```

Note that Algorithms 2.1 and 2.2 require only two triangular solvers at each step, but Algorithms (1.1) and (1.2) need three ones.

For implementations of algorithms above, we have to know how to choose $\lambda_{k+1} > 0$. One important factor is that $\frac{\lambda_{k+1}}{\mu_{k+1}}$ and $\bar{\tau}_{k+1} = \frac{\mu_{k+1}}{\lambda_{k+1}^2}$ should be well controlled such that the algorithms do not break down. For this purpose, we suggest the following rules to choose λ_{k+1}:

1. If $|\mu_{k+1}| > 1$, set $\lambda_{k+1} = 1$ which implies $\bar{\tau}_{k+1} = \mu_{k+1}$ and $\left|\frac{\lambda_{k+1}}{\mu_{k+1}}\right| < 1$.
2. The case of $|\mu_{k+1}| < 1$ is a little bit more complicated. An alternative way is $\lambda_{k+1} = \sqrt{|\mu_{k+1}|}$ that implies $\bar{\tau}_{k+1} = \text{sign}(\mu_{k+1})$ and $\left|\frac{\lambda_{k+1}}{\mu_{k+1}}\right| = \frac{1}{\lambda_{k+1}}$.

With these considerations, Algorithm 2.2 can be rewritten in the following manner:

Algorithm 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
    $l_{k+1} = L(k+1, 1{:}k)^T$,
    $L_k = L(1{:}k, 1{:}k)$,
    $\tilde{L}_k = L(k+1:n, 1:k)$,
    $\alpha_{k+1} = A(k+1, k+1)$,
    $a_{k+1} = A(k+1, 1{:}k)^T$,
    $\breve{a}_{k+1} = A(k+1, k+1:n)^T$,
    $\hat{l}_{k+1} = L_k^{-1}a_{k+1}$,
    $\mu_{k+1} = \alpha_{k+1} - l_{k+1}^T\hat{l}_{k+1}$,

    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
        $\lambda_{k+1} = \sqrt{|\mu_{k+1}|}$,
        $\bar{\tau}_{k+1} = \text{sign}(\mu_{k+1})$,
        $\breve{l}_{k+1} = \frac{\text{sign}(\mu_{k+1})}{\lambda_{k+1}}(\breve{a}_{k+1} - \tilde{L}_k\hat{l}_{k+1})$,
    end
    $\dot{t}_{k+1} = L_k^{-T}(\hat{l}_{k+1} - \bar{\tau}_{k+1}l_{k+1})$,
end
$L(k+1, k+1) = \lambda_{k+1}$,
$L(k+2:n, k+1) = \breve{l}_{k+1}$,
$T_{k+1} = \begin{pmatrix} T_k & 0 \\ \dot{t}_{k+1}^T & \bar{\tau}_{k+1} \end{pmatrix}$.
```

3. Numerical tests

We shall give numerical tests and comparison of our algorithm and most used matrix decompositions for several types of matrices (well-conditioned, ill-conditioned, dense or sparse). In Subsection 3.1, test matrix information is given (more details are given in [1–4,6,9,10,12,14,17–19]. In Subsection 3.2, the comparison results among the algorithm (2.3), the algorithms ST, MST (Modified ST) algorithm developed [15,16], the LU decomposition without pivoting (see [5], p. 163), Cholesky decomposition (see [5], p. 174) and the QR decomposition (see [5], p. 209) which were implemented in Matlab code by us. All tests were made by Matlab 6.0 in computers Pentium (III), 800 MHz, 512 MB of RAM memory, Mainboard PC CHIPS 748 and HD with 20 GB. For the simplicity, we introduce the following notations:

- For the algorithm (2.3) (New ST II)

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

- For ST and MST decomposition

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

- For LU decomposition

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

- For Cholesky decomposition

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

- For QR decomposition

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

3.1. Test matrices

We tested the following types of matrices:

1. *Circulant matrix*: $C = circul(V)$ is the circulant matrix [3] 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

Table 1
Circulant matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5810	4.5743e−14	17.7950	5.9004e−13	86.8550	1.0011e−12
ST	0.7910	2.3752e−12	31.5560	4.1408e−11	144.3170	1.4416e−10
MST	0.9620	2.8104e−13	29.5530	4.0434e−12	143.7860	1.7749e−11
LU	0.3200	1.0355e−15	4.7870	2.0809e−15	32.5170	2.7667e−15
QR	1.8030	1.4630e−15	54.7090	2.7197e−15	268.9060	4.5613e−15

Table 2
Door's matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5600	0	17.8660	0	86.1040	0
ST	0.8010	3.2568e−13	28.4710	9.4164e−13	137.5680	1.1941e−12
MST	0.9110	4.9038e−13	29.3520	9.5221e−13	143.0260	1.1514e−12
LU	0.2710	0	4.7970	0	39.0060	0
QR	1.7920	5.4334e−16	54.7890	4.7733e−16	270.3890	4.7904e−16

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 $W = [1\ \ t\ \ t^2\ \ \cdots\ \ t^n]$ is an eigenvalue of C, and $W(n{:}-1{:}1)$ is an eigenvector of C (Table 1).

2. *Door's matrix*: Door matrix (see [4])—diagonally dominant, ill-conditioned, tridiagonal. $[C, D, E] = \mathrm{dorr}(n, \theta)$ returns the vectors which defines a row diagonally dominant, tridiagonal M-matrix that is ill-conditioned for small values of the parameter $\theta \geqslant 0$ (Table 2).
   If only one output parameter is supplied then $C = \mathrm{full}(\mathrm{tridiag}(C, D, E))$, i.e., the matrix itself is returned.
   The columns of $\mathrm{inv}(C)$ vary greatly in norm. θ defaults to 0.01. The amount of diagonal dominance is given by (ignoring routing errors):
   $$\mathrm{comp}(C) * \mathrm{ones}(n, 1) = \theta * (n + 1)^2 * [1\ \ 0\ \ 0\ \ \ldots\ \ 0\ \ 1]'.$$
3. *Hilbert's matrix*: Hilbert matrix [1,9] is the n-by-n matrix with elements $1/(i + j - 1)$. It is a famous example of a ill-conditioned matrix. condition number grows exponentially as $\exp(3.5 * n)$ (Table 3).
4. *Moler matrix*: Moler matrix [11] is symmetric and positive definite with the form $A = U^T U$ where $U = \mathrm{triw}(n, \alpha)$. For $\alpha = -1$ (the default) $A(i,j) = min(i,j) - 2$, $A(i, i) = i$ (Table 4).

Table 3
Hilbert's matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 437$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5200	1.0610e−09	17.8460	1.4987e−08	57.4830	4.0805e−08
ST	0.8910	1.8508e−04	30.4040	1.1684e+03	94.4760	5.7171e+04
MST	0.8210	3.8937e−07	30.2730	2.6894e−05	94.9960	1.9576e−04
LU	0.3600	1.3013e−16	4.7970	2.0539e−16	12.6780	2.3682e−16
Cho	Fails	Fails	Fails	Fails	Fails	Fails
QR	1.8230	2.2976e−15	54.7180	4.0471e−15	175.5220	1.7678e−15

Table 4
Moler matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5510	0	17.8560	0	86.8750	0
ST	0.8810	0	29.5820	0	144.2080	0
MST	0.8410	0	29.5020	0	144.8580	0
LU	0.3200	0	4.7670	0	64.5330	0
Cho	0.3600	0	3.9050	0	12.9890	0
QR	1.7520	7.7197e−15	55.0890	2.3728e−14	273.8930	4.8580e−14

Table 5
Pie's matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5510	3.4894e−16	17.8160	5.6284e−16	86.9150	6.6973e−16
ST	0.8320	9.2500e−15	29.5130	2.6345e−14	144.4780	3.3878e−14
MST	0.8310	9.2500e−15	29.5320	2.6345e−14	143.9270	3.3878e−14
LU	0.2700	2.5755e−16	4.7270	6.3363e−16	80.4760	7.7635e−16
QR	1.7430	1.1739e−15	54.6290	9.5640e−15	268.8470	2.1778e−14

5. *Pie's matrix*: Pie's matrix is the matrix A [13] 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) $= 5 \times 10^4$ (Table 5).
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 [6], Section 4.5.4) (Table 6).

Table 6
Poisson's matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 529$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.4910	4.1372e−17	22.6620	6.0286e−17	102.3070	6.9183e−17
ST	0.8510	8.7313e−17	36.2320	1.1019e−16	164.6770	1.2472e−16
MST	0.8920	1.0400e−16	36.3920	1.2116e−16	164.7270	1.2960e−16
LU	0.2700	9.1007e−17	5.8390	1.5524e−16	81.0060	1.6950e−16
Cho	0.3700	7.9393e−17	4.7360	7.5996e−17	15.1010	7.9462e−17
QR	1.8020	9.4584e−16	69.2990	1.1766e−15	322.7850	1.2365e−15

7. *Prolate matrix*: Prolate matrix is symmetric (see [18]), ill-conditioned Toeplitz matrix. The $A = \text{prolate}(n, w)$ 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 (Table 7).
8. *Tridiagonal matrix*: Tridiagonal matrix (sparse) (see [14,17]) is the 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 $\text{tridiag}(n, c, d, e)$, 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 [17]

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

   The $\text{tridiag}(n)$ is the same as $\text{tridiag}(n, -1, 2, -1)$, which is a symmetric positive definite M-matrix (the negative of the second difference matrix) (Table 8).

Table 7
Prolate matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5100	1.8815e−07	17.8650	3.8153e−06	86.8950	3.6374e−06
ST	0.8310	1.4000e−03	29.3920	1.4300e−02	143.7560	4.0400e−02
MST	0.9310	9.7909e−05	29.6530	8.4158e−05	144.3780	3.8000e−03
LU	0.3200	1.1156e−15	4.9170	9.9488e−16	20.5590	1.0335e−15
Cho	Fails	Fails	Fails	Fails	Fails	Fails
QR	1.6930	1.2364e−15	54.8890	2.1301e−15	272.9720	2.5952e−15

Table 8
Tridiagonal matrix

Alg.	Size					
	$n = 100$		$n = 300$		$n = 500$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.5500	6.4206e−18	17.8860	4.5350e−18	86.4640	3.5120e−18
ST	0.8310	7.5151e−17	28.5310	7.9330e−17	138.8600	7.6569e−17
MST	0.8910	6.7054e−17	29.0220	6.8183e−17	138.6100	7.0215e−17
LU	0.2800	0	4.6970	0	67.4070	0
Cho	0.3610	7.8897e−17	3.9650	8.0531e−17	13.1090	7.7317e−17
QR	1.7320	4.7948e−16	54.6080	4.7447e−16	272.7530	4.7657e−16

Table 9
Wathen's matrix

Alg.	Size					
	$n = 96$		$n = 341$		$n = 560$	
	CPU	Error	CPU	Error	CPU	Error
NST	0.4300	7.4213e−17	26.7580	8.7363e−17	121.7850	8.8296e−17
ST	0.7520	1.1465e−16	42.5810	1.3154e−16	197.1640	1.1065e−16
MST	0.7620	1.1465e−16	42.6910	9.7096e−17	198.0650	1.0833e−16
LU	0.2500	1.2258e−16	6.4800	1.6593e−16	87.8760	1.6299e−16
Cho	0.3500	6.4657e−17	5.2980	7.5805e−17	17.3950	7.3758e−17
QR	1.5020	7.9502e−16	81.8780	1.2533e−15	392.4940	1.2331e−15

9. *Wathen's matrix*: Whathen matrix (see [19]) is a finite element matrix (sparse, random entries). The $A = \text{wathen}(nx, ny)$ is a sparse random n-by-n finite element matrix where $n = 3 \cdot nx \cdot ny + 2 \cdot nx + 2 \cdot ny + 1$. 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, $\text{rho}(nx, ny)$, which is chosen randomly in this routine. In particular, if $D = \text{diag}(\text{diag}(A))$, then $0.25 \leqslant \text{eig}(\text{inv}(D)A) \leqslant 4.5$ for any positive integer nx and ny and densities $\text{rho}(nx, ny)$. This diagonally scaled matrix is returned by $\text{wanthen}(nx, ny, 1)$ (Table 9).

3.2. Comparison results

In this subsection, we will present the numerical results obtained by algorithms described previously. All results are given in the following tables where NST, ST, MST, LU, Cho and QR mean Algorithm 2.3 [2], Golub–Yuan algorithm, modified ST algorithm [16], the LU decomposition, Cholesky decomposition and the QR decomposition, respectively, CPU means CPU time, and error was defined in the previous subsection for each decomposition. Alg. is abbreviation of algorithm.

4. Comments

Since main advantage of Algorithms 2.1–2.3 is that at each step, only one row is treated, they are preferable for many applications such as tomography. The new algorithms are the cheapest one among all current ST decomposition algorithms because they involve only two triangular solvers at each step. Our numerical experiments confirmed this point. It follows from tables in Subsection 3.2 that our new algorithm saves 40% CPU time compared with Golub–Yuan algorithms and modified ST algorithm given by Santiago and Yuan [16]. New algorithms possess nice numerical stability for all test matrices (see all tables in Subsection 3.2). Compared with the *LU* decomposition, all ST algorithms are expensive which is price for keeping symmetric structure. For well-conditioned matrices, the *LU* decomposition and Cholesky decomposition are better. But for ill-conditioned matrices, the row-wise algorithms are better than the Cholesky decomposition from Tables 3 and 7. As pointed in [8], the main advantage of the ST decomposition is that the decomposition is a nice preconditioner for many applications.

References

[1] M.D. Choi, Tricks or treats with the Hilbert matrix, Am. Math. Monthly 90 (1983) 301–312.
[2] C.J. Cordeiro, New Algorithms to S&T Decomposition, Master Dissertation, UFPR, Curitiba, Paraná, Brazil, 2002.
[3] P.J. Davis, Circulante Matrices, John Wiley, 1977.
[4] F.W. Door, An example of ill-conditioning in the numerical solution of singular perturbation problems, Math. Comp. 25 (1971) 271–283.
[5] L.V. Fausett, Applied Numerical Analysis Using Matlab, Prentice-Hall Inc., 1999.
[6] G.H. Golub, C. Greif, Techniques for solving KKT systems with ill-conditioned or singular (1,1) block, BIT 40th Meeting, Lund, Sweden, 9–12 August, 2000.
[7] G.H. Golub, J.Y. Yuan, ST: Symmetric-triangular decomposition and its applications. Part I: Theorems and algorithms, BIT 42 (2002) 814–822.
[8] G.H. Golub, J.Y. Yuan, ST: Symmetric-triangular decomposition and its applications. Part II: Applications, BIT 40th. Meeting, Lund, 2000.
[9] N.J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996, Section 26.1.
[10] D.E. Knuth, The art of computer programming, second ed., Fundamental Algorithms, Vol. 1, Addison-Wesley, Reading, Massachusetts, 1973, p. 37.
[11] J.C. Nash, Compact Numerical Mathods for Computer: Linear Algebra and Function Minimization, second ed., Adam Hilger, Bristol, 1990 (Appendix 1).
[12] M. Newman, J. Todd, The evaluation of matrix inversion programs, J. Soc. Indust. Appl. Math. 6 (1958) 466–476.

[13] M.L. Pei, A test matrix for inversion procedures, Commun. ACM 5 (1962) 508.
[14] D.E. Rutherford, Some continuant determinants arising in physics and chemistry II, Proc. Royal Soc. Edin. 63 A (1952) 232–241.
[15] C.D. Santiago, Numerical Experiments For S&T Decomposition, Master Dissertation, UFPR, Curitiba, Paraná, Brazil, 2001.
[16] C.D. Santiago, J.Y. Yuan, Modified ST algorithms and numerical experiments, numerical algorithms, Appl. Numer. Math. 47 (2003) 237–253.
[17] J. Todd, Basic numerical mathematics, Numerical Algebra, Vol. 2, Birkhauser, Basel, and Academic Press, New York, 1977, p. 155.
[18] J.M. Varah, The Prolate Matrix, Linear Alg. Appl. 187 (1993) 269–278.
[19] A.J. Wathen, Realistic eigenvalue bounds for the Galerkin mass matrix, IMA J. Numer. Anal. 7 (1987) 449–457.