FAST SOLVERS FOR TRIDIAGONAL TOEPLITZ LINEAR SYSTEMS

ZHONGYUN LIU *, SHAN LI *, YI YIN†, AND YULIN ZHANG‡

Abstract. Let A be a tridiagonal Toeplitz matrix denoted by $A = Tritoep(\beta, \alpha, \gamma)$. The matrix A is said to be: *strictly diagonally dominant* if $|\alpha| > |\beta| + |\gamma|$, *weakly diagonally dominant* if $|\alpha| \geq |\beta| + |\gamma|$, *sub-diagonally dominant* if $|\beta| \geq |\alpha| + |\gamma|$ and *super-diagonally dominant* if $|\gamma| \geq |\alpha| + |\beta|$. In this paper, we consider the solution of a tridiagonal Toeplitz system $A\mathbf{x} = \mathbf{b}$, where A is sub-diagonally dominant, super-diagonally dominant or weakly diagonally dominant, respectively. We first consider the case of A being sub-diagonally dominant. We transform A into a block 2×2 matrix by an elementary transformation and then solve such a linear system by using the block LU factorization. Compared with the LU factorization method with pivoting, our algorithm takes less flops, needs less memory storage and data transmission. In particular, our algorithm outperforms the LU factorization method with pivoting in terms of computing efficiency. Then we deal with super-diagonally dominant and weakly diagonally dominant cases, respectively. Numerical experiments are finally given to illustrate the effectiveness of our algorithms.

Key words. Tridiagonal Toeplitz matrices, Diagonally dominant, Schur complement, Block LU factorization, Pivoting.

AMS subject classifications. 15A23, 15B05, 65F05, 65F10

1. Introduction.

Consider the direct solution of the following nonsingular linear system

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where A is an $n \times n$ tridiagonal Toeplitz matrix of the following form

$$A = \begin{bmatrix} \alpha & \gamma & & & & \\ \beta & \alpha & \gamma & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta & \alpha & \gamma \\ & & & & \beta & \alpha \end{bmatrix}. \tag{1.2}$$

Toeplitz matrices are a class of special structured matrices, which arise from a variety of applications in mathematics, scientific computing and engineering, for instance, image restoration storage problems in signal processing, algebraic differential equation, time series and control theory. Exploiting their structure, some fast and/or accurate algorithms for Toeplitz systems and eigen-problems have been developed, see for instance [3, 4, 8, 11, 15] and references therein.

There are two main types of methods for solving Toeplitz systems: direct methods and iterative methods. Iterative methods consist of classical splitting iteration methods and Krylov subspace iteration methods. Classical splitting iteration methods for Toeplitz systems require efficient splittings which depend on the structure and property of coefficient matrices, for example, Gauss-Seidel and SOR splittings [20] for H-matrices and Hermitian positive definite matrices, circulant and skew circulant splitting for positive definite matrices [13, 17], trigonometric transform splitting for real symmetric positive definite matrices [14], real symmetric and skew symmetric splitting for real positive definite matrices [5,9] which is a tailor of Hermitian

*School of Mathematics and Statistics, Changsha University of Science and Technology, Changsha 410076, P. R. China. (Zhongyun Liu: Male, Prof./Dr., Tutor of PhD student, Major: Numerical Linear Algebra, Email: liuzhongyun@263.net)

†Department of Basic Courses, Hunan College of Information, Changsha, 410200, P. R. China (yinyi@mail.hniu.cn)

‡Centro de Matemática, Universidade do Minho, 4710-057 Braga, Portugal (Yulin Zhang: zhang@math.uminho.pt).

and skew Hermitian splitting for general positive definite matrices [1], and so on. Krylov subspace iteration methods include CG method for Hermitian positive definite matrices and GMRES methods for non-Hermitian positive definite matrices as well as their variants [3, 4, 20]. One important property of iteration methods is that only matrix-vector multiplications are involved at each iteration, which can be obtained in $O(n \log n)$ operations by the FFT or DCT and DST, see for instance [3, 4, 8, 12, 17], and another important property is that iteration methods are efficient and robust numerically, and suited for solving large sparse linear systems. Direct methods (see [8]) are mainly applicable for small and medium model problems but often too expensive to be practical for large sparse problems. Furthermore, direct methods may suffer from numerical instability and loss of accuracy of solutions [4].

Tridiagonal Toeplitz linear systems have their own applications in mathematical science and engineering field [18–20]. Due to their special structure, fast and/or accurate solvers have intrigued researchers for decades, see for example [6, 19, 21, 22] for serial algorithms and [7, 10, 16] for parallel algorithms. Those algorithms are all numerically reliable, if the coefficient matrix A is strictly diagonally dominant. Otherwise, a pivoting strategy will be necessary for the LU factorization of A, see for example [2]. It is known that a pivoting strategy may cause a lot of nonzero fill-ins, which will result in more memory storage required and more float-point operations increased. This motivates us to consider new methods for solving (1.1), where A is assumed to be sub-diagonally dominant, super-diagonally dominant or weakly diagonally dominant. Such classes of matrices frequently appear in numerical solution to one-dimension convection-diffusion equation via finite difference scheme, see for instance [20].

This paper is organized as follows. In next section, we develop some fast algorithms for solving (1.1) with the coefficient matrix A being strictly sub-diagonally dominant, strictly super-diagonally dominant and weakly diagonally dominant, respectively. Numerical tests are illustrated in Section 3 to show the effectiveness of our algorithms. A brief conclusion is drawn in Section 4, and the acknowledgements are written in the last section.

2. Fast algorithms. We begin with the strictly sub-diagonally dominant case. Let

$$C = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

be the shifted circulant matrix. It is easy to verify that $\hat{A} = CA$ has the following block 2×2 structure

$$\hat{A} = \left[\begin{array}{cccc|c} \beta & \alpha & \gamma & & \\ & \beta & \alpha & \ddots & \\ & & \ddots & \ddots & \gamma \\ & & & \beta & \alpha \\ \hline \alpha & \gamma & & & 0 \end{array}\right] \equiv \left[\begin{array}{c|c} A_{11} & \mathbf{p} \\ \hline \mathbf{w^T} & 0 \end{array}\right] \tag{2.1}$$

and admits the following bock 2×2-LU factorization

$$\hat{A} = \begin{bmatrix} I & 0 \\ \mathbf{w^T} A_{11}^{-1} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & \mathbf{p} \\ 0 & -\mathbf{w^T} A_{11}^{-1} \mathbf{p} \end{bmatrix}, \tag{2.2}$$

where $-\mathbf{w^T} A_{11}^{-1} \mathbf{p}$ is the so-called Schur complement of $\hat{A}$ corresponding to A_{11}.

Equivalently, the problem for solving (1.1) becomes solving

$$\hat{A}\mathbf{x} = \hat{\mathbf{b}} \tag{2.3}$$

where $\hat{\mathbf{b}} = (b_2, b_3, \ldots, b_n, b_1)^T$.

Partitioning $\mathbf{x}$ and $\hat{\mathbf{b}}$ into the following forms

$$\mathbf{x} = \left[\frac{\mathbf{x}_1}{x_n}\right] \quad \text{and} \quad \hat{\mathbf{b}} = \left[\frac{\mathbf{b_2}}{b_1}\right],$$

and using the block LU factorization (2.2), we can get the solution to (1.1) by solving

$$\begin{cases} A_{11}\mathbf{x_1} + x_n\mathbf{p} & = \mathbf{b_2} \\ \mathbf{w^T}A_{11}^{-1}\mathbf{b_2} - \mathbf{w^T}A_{11}^{-1}\mathbf{p}x_n & = b_1 \end{cases} \tag{2.4}$$

In order to solve (2.4), we first solve

$$\begin{cases} A_{11}\mathbf{u} = \mathbf{p} \\ A_{11}\mathbf{v} = \mathbf{b_2} \end{cases} \tag{2.5}$$

by the following algorithm

Algorithm 1 Backward substitution for solving $A_{11}\mathbf{z} = \mathbf{p}$.

1: Input $\beta, \alpha, \gamma, \mathbf{p}$;
2: Compute $\alpha_1 = \frac{\alpha}{\beta}, \gamma_1 = \frac{\gamma}{\beta}$;
3: Compute $z_{n-1} = \alpha_1; z_{n-2} = \gamma_1 - \alpha_1 z_{n-1}$;
4: For $i = 2, \ldots, n-2$,
compute $z_{n-i-1} = -\alpha_1 z_{n-i} - \gamma_1 z_{n-i+1}$;
5: Output $\mathbf{z} = [z_1, z_2, \ldots, z_{n-1}]^T$.

Analogously, we can solve $A_{11}\mathbf{z} = \mathbf{b_2}$.

Once we get the solutions $\mathbf{u}$ and $\mathbf{v}$ to systems in Equation (2.5), we can form the solution to (1.1) given by

$$\begin{cases} \mathbf{x_1} = \mathbf{v} - x_n\mathbf{u} \\ x_n = (\mathbf{w^T}\mathbf{v} - b_1)/\mathbf{w^T}\mathbf{u}. \end{cases} \tag{2.6}$$

We remark here that to obtain the solution (2.6) we need to solve two auxiliary linear systems in (2.5), where A_{11} is an upper triangular tridiagonal matrix. Obviously, the vectors $\mathbf{u}$ and $\mathbf{v}$ can be easily obtained by a backward substitution. Also, due to the strictly diagonally dominance of A_{11}, both computed vectors $\mathbf{u}$ and $\mathbf{v}$ are reliable.

Based on the above analysis, we can now reformulate our algorithm for solve (1.1) as follows.

Algorithm 2 An algorithm for solving tridiagonal Toeplitz linear system $A\mathbf{x} = \mathbf{b}$.

1: Input $\beta, \alpha, \gamma, \mathbf{w}, \mathbf{p}, \mathbf{b}_2, b_1$;
2: Call Algorithm 1 to solve $A_{11}\mathbf{u} = \mathbf{p}$, $A_{11}\mathbf{v} = \mathbf{b}_2$ defined in (2.5);
3: Compute x_n and $\mathbf{x_1}$ according to (2.6);
4: Output $\mathbf{x} = [\mathbf{x_1}^T x_n]^T$.

In algorithm 2, when computing the $\mathbf{x_1}$ and x_n, $\mathbf{u}$ and $\mathbf{v}$ are used twice, $\mathbf{w^T u}$ and $\mathbf{w^T v}$ are used only once, so the influence of the error propagation can be neglected, therefore x_n and $\mathbf{x_1}$ are reliable. Thus we can conclude that our algorithm for solving such a class of linear systems is numerically stable and the computed solutions are reliable.

For the computational complexity, our algorithm takes about $12n$ flops, less than $13n$ flops required for the LU factorization method with pivoting. For memory storage, our algorithm needs store 2 n-vectors only, less than 5 n-vectors required to store for the LU factorization method with pivoting. In particular, our algorithm needs less data transmission. It only reads one vector (the right-hand side vector) and writes one vector (the solution), but the LU factorization method with pivoting needs read 5 vectors. As we know modern computers have multi-level memory hierarchy, there are different storage levels such as the smaller high-speed cache and larger low-speed disk storage. During computations, data transfer in different levels of cache memory. Thus, algorithms with less data transmission may show better computating performance. This makes the algorithm more efficient than LU factorization method with pivoting. The efficiency of our algorithm is illustrated by numerical results in next section.

Now, we consider the strictly superdiagonally dominant case. Let J be the exchange matrix with ones on the cross diagonal (bottom left to top right) and zeros elsewhere, i.e.,

$$J = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

Because A is strictly superdiagonally dominant, $\tilde{A} = JAJ = Tritoep(\gamma, \alpha, \beta)$ becomes strictly subdiagonally dominant. Therefore we can transform the original linear system (1.1) into the following new one

$$\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \tag{2.7}$$

where $\tilde{\mathbf{x}} = J\mathbf{x}$ and $\tilde{\mathbf{b}} = J\mathbf{b}$. Thus, combining Algorithm 2 yields the following Algorithm 3 for solving (1.1).

Algorithm 3 An algorithm for solving tridiagonal Toeplitz linear system $A\mathbf{x} = \mathbf{b}$.

1: Input $\gamma, \alpha, \beta, \tilde{\mathbf{b}}$;
2: Call Algorithm 2 to obtain the solution $\tilde{\mathbf{x}}$ to (2.7);
3: Output $\mathbf{x} = J\tilde{\mathbf{x}}$.

Since J is a permutation matrix, the stability, computational complexity and memory storage of Algorithm 3 are the same as Algorithm 2.

Finally, we turn to the case of A being weakly diagonally dominant. It is known that in this case A is an H-matrix which admits an LU factorization without pivoting. Obviously, this LU factorization does not cause any nonzero fill-ins, but needs more memory units, see [6] for a similar analysis when A is strictly diagonally dominant. To avoid this problem, we adopt the following strategy:

If $\beta > \gamma$, then we call Algorithm 2 for solving (1.1), if $\beta < \gamma$, then we call Algorithm 3 for solving (1.1).

Algorithm 4 An algorithm for solving tridiagonal Toeplitz linear system $A\mathbf{x} = \mathbf{b}$.

1: Input γ, α, β, $\mathbf{b}$ (if $\beta > \gamma$) or $\tilde{\mathbf{b}}$ (if $\beta < \gamma$);
2: Call Algorithm 2 (if $\beta > \gamma$) or Algorithm 3 (if $\beta < \gamma$) to obtain the solution $\mathbf{x}$ to (1.1);
3: Output $\mathbf{x}$.

The computational complexity and memory storage of Algorithm 4 are about the same as Algorithm 2 or 3.

3. Numerical experiments. In this section, we use some examples to illustrate the effectiveness of our algorithms.

All the numerical tests were done on a Founder desktop PC with Intel(R) Core(TM) i3-2310M CPU @2.10 by Matlab R2009(b) with a machine precision of 10^{-16}.

For convenience, throughout our numerical experiments, we denote by $R = ||\mathbf{b} - A\mathbf{x}||/||\mathbf{b}||$, CPU, NEW, PLU, LU the relative residual error, computing time (in seconds), our new algorithm, LU factorization method with pivoting and LU factorization method. In all tables, the CPU is the average value of computing times required by performing the corresponding algorithm 10 times, the right hand side vector $\mathbf{b}$ is taken to be $A\mathbf{e}$ or $A\mathbf{x}^*$ with $\mathbf{x}^* = rand(n, 1)$.

All tested matrices are arising from the following one-dimension convection-diffusion equation,

$$\begin{cases} -au'' + bu' = 0, \ x \in (0,1) \\ u(0) = 0, \ u(1) = 1 \end{cases}$$

where the coefficients a and b are constant.

When we use centered difference schemes for the second order derivative and the first order derivative, we get the following example

Example 1: $A = Tritoep(-1-c, \ 2, \ -1+c)$.

When we use centered difference schemes for the second order derivative and the backward difference scheme for the first order derivative, we get the following example

Example 2: $A = Tritoep(-1-c, \ 2+c, \ -1)$.

When we use centered difference schemes for the second order derivative and the forward difference scheme for the first order derivative, we get the following example

Example 3: $A = Tritoep(-1, \ 2-c, \ -1+c)$.

In numerical experiments, we test numerous matrices in Examples 1-3 with different order n and various parameter c. We find that all experiments have a similar numerical behavior. We therefore select some of them listed in following tables.

Numerical results are illustrated in Tables 3.1-3.2 for Example 1 with $c = 12.5$ and $c = 2.5$, Tables 3.3-3.4 for Example 2 with $c = -6.5$ and $c = -9.5$ (where those matrices are subdiagonally dominant) and Tables 3.5-3.6 for Example 3 with $c = 5.5$ and $c = 7.5$ (in which both matrices A are superdiagonally dominant). In those six cases, we also test the LU factorization method with pivoting for comparison. We observe that our method for Examples 1-2 and its variant for Example 3 perform much faster than the LU factorization method with pivoting and produces more accurate solutions for $\mathbf{b} = A\mathbf{e}$ than the latter in all cases. As for $\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n, 1))$, both methods produce solutions with almost the same accuracy.

TABLE 3.1
Comparisons of our algorithm with PLU for Example 1 with $c = 12.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	3.028e-2	9.711e-16	1.484e-1	7.263e-14	3.081e-2	1.962e-16	1.498e-1	1.026e-16
2^{22}	2.446e-1	9.711e-16	1.201e+0	2.052e-13	2.506e-1	1.742e-16	1.216e+0	1.339e-16
2^{24}	9.711e-1	9.711e-16	4.797e+0	4.103e-13	1.018e+0	1.694e-16	4.880e+0	2.165e-16

TABLE 3.2
Comparisons of our algorithm with PLU for Example 1 with $c = 2.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	2.984e-2	7.648e-16	1.473e-1	1.056e-15	3.130e-2	1.630e-16	1.498e-1	1.719e-16
2^{22}	2.456e-1	7.648e-16	1.186e+0	1.056e-15	2.498e-1	1.748e-16	1.197e+0	1.449e-16
2^{24}	9.685e-1	7.648e-16	4.765e+0	1.056e-15	9.978e-1	1.827e-16	4.782e+0	1.730e-16

TABLE 3.3
Comparisons of our algorithm with PLU for Example 2 with $c = -6.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	3.003e-2	2.632e-16	1.466e-1	5.051e-14	3.056e-2	1.654e-16	1.486e-1	1.021e-16
2^{22}	2.440e-1	2.632e-16	1.187e+0	1.428e-13	2.444e-1	1.646e-16	1.190e+0	1.651e-16
2^{24}	9.610e-1	2.632e-16	4.728e+0	2.856e-13	9.747e-1	1.688e-16	4.764e+0	1.017e-16

TABLE 3.4
Comparisons of our algorithm with PLU for Example 2 with $c = -9.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	3.080e-2	5.374e-16	1.519e-1	3.757e-14	3.166e-2	1.782e-16	1.485e-1	1.181e-16
2^{22}	2.156e-1	5.374e-16	1.197e+0	1.063e-13	2.497e-1	1.735e-16	1.195e+0	2.432e-16
2^{24}	9.775e-1	5.374e-16	4.799e+0	2.125e-13	9.802e-1	1.675e-16	4.801e+0	3.297e-16

TABLE 3.5
Comparisons of our algorithm with PLU for Example 3 with $c = 5.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	3.090e-2	6.812e-16	1.492e-1	3.489e-14	3.180e-2	1.949e-16	1.492e-1	1.059e-16
2^{22}	2.457e-1	6.812e-16	1.218e+0	9.866e-14	2.500e-1	1.637e-16	1.193e+0	1.953e-16
2^{24}	9.789e-1	6.812e-16	4.807e+0	1.973e-13	9.970e-1	1.632e-16	4.803e+0	1.959e-16

TABLE 3.6
Comparisons of our algorithm with PLU for Example 3 with $c = 7.5$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
n	NEW		PLU		NEW		PLU	
	CPU	R	CPU	R	CPU	R	CPU	R
2^{19}	3.072e-2	3.480e-16	1.481e-1	1.225e-14	3.130e-2	1.703e-16	1.501e-1	1.456e-16
2^{22}	2.441e-1	3.480e-16	1.199e+0	3.458e-14	2.523e-1	1.677e-16	1.193e+0	1.749e-16
2^{24}	9.839e-1	3.480e-16	4.784e+0	6.915e-14	9.879e-1	1.819e-16	4.794e+0	2.591e-16

Numerical results for fixed $n = 2^{22}$ and various c are illustrated in Table 3.7 for Example 1 and Table 3.8 for Example 2 (where all those matrices are weakly diagonally dominant)

TABLE 3.7
Comparisons of our algorithm with LU for Example 1 with $n = 2^{22}$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
c	NEW		LU		NEW		LU	
	CPU	R	CPU	R	CPU	R	CPU	R
0.1	8.801e-2	1.304e-12	8.104e-1	2.912e-13	8.702e-2	5.566e-16	8.069e-2	1.898e-16
0.2	8.841e-2	7.268e-13	8.067e-1	1.577e-13	8.663e-2	4.568e-16	8.073e-1	1.313e-16
0.3	8.921e-2	7.408e-13	8.072e-1	2.310e-13	8.692e-2	2.591e-16	8.075e-1	1.542e-16
0.4	8.852e-2	3.806e-13	8.090e-1	2.144e-13	8.713e-2	2.297e-16	8.08e-1	1.875e-16
0.5	8.781e-2	1.438e-15	8.064e-1	6.549e-16	8.731e-2	1.503e-16	8.081e-1	1.135e-16
0.6	8.823e-2	1.688e-15	8.068e-1	9.107e-16	8.701e-2	1.698e-16	8.043e-1	1.520e-16
0.7	8.931e-2	2.888e-13	8.098e-1	2.908e-13	8.667e-2	1.462e-16	8.056e-1	2.243e-16
0.8	8.851e-2	1.256e-13	8.075e-1	9.416e-14	8.632e-2	1.606e-16	8.059e-1	1.356e-16
0.9	8.801e-2	2.106e-13	8.097e-1	2.544e-13	8.651e-2	1.172e-16	8.059e-1	2.101e-16

TABLE 3.8
Comparisons of our algorithm with LU for Example 2 with $n = 2^{22}$

	$\mathbf{b} = A\mathbf{e}$				$\mathbf{b} = Ax^*(\mathbf{x}^* = rand(n,1))$			
c	NEW		LU		NEW		LU	
	CPU	R	CPU	R	CPU	R	CPU	R
−0.9	8.801e-2	3.491e-13	8.074e-1	3.720e-13	8.691e-2	1.769e-16	8.081e-1	1.748e-16
−0.8	8.823e-2	2.605e-13	8.067e-1	1.115e-13	8.823e-2	2.459e-16	8.093e-1	1.474e-16
−0.7	8.812e-2	3.984e-13	8.068e-1	1.089e-13	8.701e-2	2.682e-16	8.084e-1	1.455e-16
−0.6	8.821e-2	2.962e-15	1.449e0	6.896e-16	8.761e-2	2.641e-16	1.452e0	2.285e-15
−0.5	8.912e-2	6.185e-16	8.066e-1	3.580e-16	8.751e-2	4.428e-16	8.070e-1	1.304e-16
−0.4	8.841e-2	3.762e-16	8.066e-1	1.304e-16	8.821e-2	1.708e-16	8.060e-1	1.619e-16
−0.3	8.762e-2	2.982e-15	8.072e-1	3.315e-13	8.712e-2	4.049e-16	8.088e-1	1.888e-16
−0.2	8.891e-2	6.934e-13	8.082e-1	2.082e-13	8.912e-2	1.435e-15	8.082e-1	1.601e-16
−0.1	8.791e-2	2.851e-13	8.082e-1	2.390e-13	8.701e-2	1.008e-15	8.069e-1	1.862e-16
0.1	9.011e-2	2.879e-13	8.050e-1	3.504e-13	8.671e-2	1.458e-15	8.083e-1	1.822e-16
0.2	8.821e-2	1.901e-13	8.074e-1	3.045e-13	8.751e-2	1.914e-16	8.091e-1	1.714e-16
0.3	8.821e-2	4.809e-13	1.452e0	1.379e-10	8.731e-2	2.062e-16	1.452e0	1.379e-13
0.4	8.901e-2	7.538e-13	8.081e-1	1.748e-13	8.712e-2	8.243e-16	8.083e-1	1.715e-16
0.5	9.071e-2	5.498e-13	8.098e-1	2.185e-13	8.912e-2	2.172e-16	8.089e-1	1.635e-16
0.6	8.801e-2	7.429e-13	8.072e-1	3.747e-13	8.701e-2	7.518e-16	8.102e-1	1.966e-16
0.7	8.820e-2	4.714e-13	8.075e-1	2.306e-13	8.812e-2	5.488e-16	8.089e-1	1.553e-16
0.8	8.841e-2	3.035e-13	1.452e0	4.513e-10	8.911e-2	1.808e-16	1.455e0	2.411e-13
0.9	8.871e-2	1.805e-15	8.081e-1	3.604e-13	8.701e-2	2.079e-16	8.105e-1	2.269e-16
1	8.941e-2	6.185e-16	8.078e-1	3.158e-16	8.741e-2	4.721e-16	8.064e-1	1.304e-16
3	8.841e-2	0	8.078e-1	1.654e-13	8.751e-2	1.518e-16	8.090e-1	1.241e-16
6	8.761e-2	6.431e-14	8.086e-1	6.431e-14	8.741e-2	1.544e-16	8.082e-1	1.204e-16
9	8.731e-2	4.526e-14	8.072e-1	1.131e-13	8.691e-2	1.772e-16	8.069e-1	1.591e-16

From Tables 3.7-3.8, we can see our method works much faster than the LU factorization method, while solutions produced by our new method are about as accurate as those obtained by the LU factorization method.

4. Conclusions. We have explored the special structure of tridiagonal Toeplitz matrices to develop an effective algorithm for solving tridiagonal Toeplitz linear systems. We first transformed the original system into a new one by an elementary transformation. The coefficient matrix of the new system becomes a block 2×2 matrix whose principal leading block is an upper triangular tridiagonal Toeplitz matrix of order $n-1$. Based on this block 2×2 structure, we then proposed an new algorithm.

Compared with the LU factorization method, some significant advantages of our algorithm can be found. One is that our algorithm takes less flops. Another is that less memory storage and data transmission are required to perform our algorithm.

Numerical tests were shown that our algorithm fully outperforms the LU factorization method with pivoting in terms of computing efficiency and accuracy of the produced solution, when solving a tridiagonal Toeplitz system whose coefficient matrix is subdiagonally or superdiagonally dominant. In addition, we found numerically that our method performs as well as the LU factorization method when solving a tridiagonal Toeplitz system whose coefficient matrix is weakly diagonally dominant.

5. Acknowledgement. The authors would like to thank the supports of the National Natural Science Foundation of China under Grant No. 11371075, the Hunan Key Laboratory of mathematical modeling and analysis in engineering, the research innovation program of Changsha University of Science and Technology for postgraduate students under Grant(CX2019SS34), the Portuguese Funds through FCT-Fundacão para a Ciência, within the Project UIDB/00013/2020 and UIDP/00013/2020.

REFERENCES

[1] Z.-Z. Bai, G. H. Golub and M. K. Ng, *Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems*, SIAM J. Matrix Anal. Appl., 24 (2003): 603-626.
[2] J. R. Bunch, R. F. Marcia, *A pivoting strategy for symmetric tridiagonal matrices*, J. Numer. Linear Algebra, 12 (2000): 911-922.
[3] R. Chan and X.-J. Jin, *An introduction to iterative Toeplitz solvers.* SIAM, Philadelphia, PA, USA, 2007.
[4] R. Chan and M. K. Ng, *Conjugate gradient methods for Toeplitz systems,* SIAM Rev. 38 (1996): 427-482.
[5] F. Chen and Y.-L. Jiang, *On HSS and AHSS iteration methods for nonsymmetric positive definite Toeplitz systems*, J. Comput. Appl. Math., 234 (2010): 2432-2440.
[6] L. Du, T. Sogabe and S.-L. Zhang, *A fast algorithm for solving tridiagonal quasi-Toeplitz linear systems*, Appl. Math. Lett. 75 (2017): 74-81.
[7] L. E. Garey and R. E. Shaw, *A parallel method for linear equations with tridiagonal Toeplitz coefficient matrices*, Comput. Math. Appl., 42 (2001): 1-11.
[8] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore and London, 3rd edition, 1996.
[9] C.-Q. Gu and Z.-L. Tian, *On the HSS iteration methods for positive definite Toeplitz linear systems*, J. Comput. Appl. Math., 224 (2009): 709-718.
[10] H. J. Kim, *A parallel algorithm solving a tridiagonal Toeplitz linear system*, Parallel Comput., 13 (1990): 289-294.
[11] Z.-Y. Liu, L. Chen and Y.-L. Zhang, *The reconstruction of an Hermitian Toeplitz matrices with prescribed eigenpairs*, J. Syst. Sci. Complex., 23 (2010): 961-970.
[12] Z.-Y. Liu, S.-H. Chen, W.-J. Xu and Y.-L. Zhang, *The eigen-structures of real (skew) circulant matrices with some applications*, Comput. Appl. Math., 38 (2019) : 178 (https://doi.org/10.1007/s40314-019-0971-9).
[13] Z.-Y. Liu, X.-R. Qin, N.-C. Wu and Y.-L. Zhang, *The shifted classical circulant and skew circulant splitting iterative methods for Toeplitz matrices*, Canad. Math. Bull., 60 (2017): 807-815.
[14] Z.-Y. Liu, N.-C. Wu, X.-R. Qin and Y.-L. Zhang, *Trigonometric transform splitting methods for real symmetric Toeplitz systems*, Comput. Math. Appl., 75 (2018): 2782-2794.
[15] Z.-Y. Liu, Y.-L. Zhang, C. Ferreira and R. Ralha, *On inverse eigenvalue problems for block Toeplitz matrices with Toeplitz blocks*, Appl. Math. Comput., 216 (2010): 1819-1830.
[16] J. M. McNally, L. E. Garey and R. E. Shaw, *A split-correct parallel algorithm for solving tridiagonal symmetric toeplitz systems*, INT. J. Comput. Math., 75 (2000): 303-313.
[17] M. K. Ng, *Circulant and skew-circulant splitting methods for Toeplitz systems,* J. Comput. Appl. Math., 159 (2003): 101-108.
[18] S. Noschese, L. Pasquini and L. Reichel, *Tridiagonal Toeplitz matrices: properties and novel applications*, Numer. Linear Algebra Appl., 20 (2013): 302-326.

[19] O. Rojo, *A new method for solving symmetric circulant tridiagonal systems of linear equations*, J. Parllel Distr. Com., 20 (1990): 61-67.

[20] Y. Saad, *Iterative methods for sparse Linear systems*, 2nd edition, SIAM, Philadelphia, PA, USA, 2003.

[21] A. V. Terekhov, *Parallel dichotomy algorithm for solving tridiagonal system of linear equations with multiple right-hand sides*, J. Parllel Distr. Com., 87 (2015): 102-108.

[22] W.-M. Yan and K.-L. Chung, *A fast algorithm for solving special tridiagonal systems*, Computing, 52 (1994): 203-211.