THE EIGEN-STRUCTURES OF REAL (SKEW) CIRCULANT MATRICES WITH SOME APPLICATIONS

ZHONGYUN LIU*, SIHENG CHEN*, WEIJIN XU*, AND YULIN ZHANG†

Abstract. The circulant matrices and skew-circulant matrices are two special classes of Toeplitz matrices and play vital roles in the computation of Toeplitz matrices. In this paper, we focus on real circulant and skew-circulant matrices. We first investigate their real Schur forms, which are closely related to the family of discrete cosine transform (DCT) and discrete sine transform (DST). Using those real Schur forms, we then develop some fast algorithms for computing real circulant, skew-circulant and Toeplitz matrix-real vector multiplications. Also, we develop a DCT-DST version of circulant and skew-circulant splitting (CSCS) iteration for real positive definite Toeplitz systems. Compared with the fast Fourier transform (FFT) version of CSCS iteration, the DCT-DST version is more efficient and saves a half storage. Numerical experiments are presented to illustrate the effectiveness of our method.

Key words. Real Schur form, real circulant matrices, real skew-circulant matrices, real Toeplitz matrices, CSCS iteration.

AMS subject classifications. 15A23, 65F10, 65F15.

1. Introduction.

Recall that a matrix $T = (t_{jk})_{j,k=0}^{n-1}$ is said to be *Toeplitz* if $t_{jk} = t_{j-k}$; a matrix $C = (c_{jk})_{j,k=0}^{n-1}$ is said to be *circulant* if $c_{jk} = c_{j-k}$ and $c_{-l} = c_{n-l}$ for $1 \le l \le n-1$; and a matrix $S = (s_{jk})_{j,k=0}^{n-1}$ is said to be *skew-circulant* if $s_{jk} = s_{j-k}$ and $s_{-l} = -s_{n-l}$ for $1 \le l \le n-1$.

Toeplitz matrices arise in a variety of applications in mathematics, scientific computing and engineering, for instance, signal processing, algebraic differential equation, time series and control theory, see e.g. [3] and a large literature therein. Those applications have motivated both mathematicians and engineers to develop specific algorithms for solving Toeplitz systems for instance [3, 6, 13] and references therein.

The discrete Fourier transform (DFT) matrix $F = (F_{jk})$ is defined by

$$F_{jk} = \frac{1}{\sqrt{n}}\omega^{-jk}, \quad j,\ k = 0, 1, \cdots, n-1, \text{ where } \omega = \exp(\frac{2\pi}{n}i),\ i = \sqrt{-1}. \tag{1.1}$$

It is known that any circulant matrix C and skew-circulant matrix S possess the following Schur canonical forms [3, 10, 5], respectively,

$$C = F\Lambda F^* \quad \textit{and} \quad S = \tilde{F}\tilde{\Lambda}\tilde{F}^*, \tag{1.2}$$

*School of Mathematics and Statistics, Changsha University of Science and Technology, Changsha 410076, P. R. China (liuzhongyun@263.net, mathlife@sina.cn, 649527704@qq.com).

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

where $\tilde{F} = DF^*$ is an unitary matrix with $D = \mathrm{diag}\big(1, e^{\frac{\pi}{n}i}, \cdots, e^{\frac{(n-1)\pi}{n}i}\big)$, Λ and $\tilde{\Lambda}$ are diagonal matrices, holding the eigenvalues of C and S respectively. Moreover, Λ and $\tilde{\Lambda}$ can be obtained in $O(n \log n)$ operations by using two FFTs of the first rows of C and S, respectively.

Due to the Schur canonical forms (1.2) of C and S, the products $C\boldsymbol{x}$ or $S\boldsymbol{x}$ for any vector $\boldsymbol{x}$ can be computed by 3FFTs (1 FFT for computing eigenvalues) in $O(n \log n)$ operations.

Very often, circulant and skew circulant matrices are used to deal with Toeplitz issues. An important property is that a Toeplitz matrix T can be split into the following circulant and skew-circulant splitting (CSCS)[10]

$$T = C + S \tag{1.3}$$

where $C = (c_{jk})$ is a circulant matrix and $S = (s_{jk})$ is a skew-circulant matrix, which are defined as follows.

$$c_{jk} = \begin{cases} \frac{1}{2}t_0, & \text{if } j = k, \\ \frac{(t_{j-k}+t_{j-k-n})}{2}, & \text{otherwise,} \end{cases} \quad \text{and} \quad s_{jk} = \begin{cases} \frac{1}{2}t_0, & \text{if } j = k, \\ \frac{t_{j-k}-t_{j-k-n}}{2}, & \text{otherwise.} \end{cases}$$

Actually, due to $T\boldsymbol{x} = C\boldsymbol{x} + S\boldsymbol{x}$, $T\boldsymbol{x}$ can be computed by 6 FFTs of n-vector. Also, any linear system of equations $C\boldsymbol{x} = \boldsymbol{b}$ ($S\boldsymbol{x} = \boldsymbol{b}$) that contains circulant matrices (skew-circulant matrices) may be quickly solved by using the FFT. However, all operations, due to FFTs, are involved into complex arithmetics, even if C (S) and $\boldsymbol{b}$ are real. Now, one may ask when C and S are real, could we find an analogue of (1.2) for C and S to avoid complex arithmetics in matrix-vector multiplication? and/or when C (S) and $\boldsymbol{b}$ are real, could we develop an algorithm which only involves real arithmetics for solving $C\boldsymbol{x} = \boldsymbol{b}$ ($S\boldsymbol{x} = \boldsymbol{b}$)? This is the main motivation of this paper.

The organization of this paper is as follows. In the next section, by exploring the eigen-structures of C and S, we will give the real Schur forms of the circulant matrix C and the skew-circulant matrix S. In Sections 3 and 4, with the real schur forms, we will develop a real method to fast calculate Toeplitz matrix-vector multiplication and an algorithm based on the CSCS iteration in [10] to solve $T\boldsymbol{x} = \boldsymbol{b}$ by real arithmetics. Numerical experiments are presented in Section 5 to show the effectiveness of our method. A brief conclusion and the acknowledgements are finally followed.

2. The Real Schur Forms of Real (Skew) Circulant Matrices. In the section, making use of the eigen-structures of a real circulant matrix C and a real skew-circulant matrix S, we develop their corresponding real Schur forms.

2.1. Preliminaries. Let's begin with some basic definitions. For convenience, throughout the paper, we define J_n the permutation matrix of order n with ones on the cross diagonal (bottom left to top right) and zeros elsewhere, and P_{pq} is the $(q-p) \times n$ restriction matrix satisfying $P_{pq}\,(x_j)_{j=0}^{n-1} = (x_j)_{j=p}^{q-1}$, $q > p$.

Definition 2.1. [8] *A vector $\boldsymbol{x} \in \mathbb{R}^n$ is said to be symmetric if $J_n\boldsymbol{x} = \boldsymbol{x}$ and skew-symmetric if $J_n\boldsymbol{x} = -\boldsymbol{x}$.*

Now, let's recall the definitions of DCTs and DSTs. The family of discrete trigonometric transforms consists of 8 versions of DCTs and corresponding 8 versions of DSTs [15, 12, 11]. In this paper, we only need four versions of them which will be used in the sequel.

DEFINITION 2.2. *The* DCT-I, DCT-II, DCT-V and DCT-VI *matrices are defined as follows.*

$$\mathscr{C}^{\mathrm{I}}_{n+1} = \sqrt{\frac{n}{2}}\left[\tau_j\tau_k\cos\frac{jk\pi}{n}\right]_{j,k=0}^{n}, \qquad \mathscr{C}^{\mathrm{V}}_{n} = \frac{2}{\sqrt{2n-1}}\left[\tau_j\tau_k\cos\frac{2jk\pi}{2n-1}\right]_{j,k=0}^{n-1},$$

$$\mathscr{C}^{\mathrm{II}}_{n} = \sqrt{\frac{n}{2}}\left[\tau_j\cos\frac{j(2k+1)\pi}{2n}\right]_{j,k=0}^{n-1}, \qquad \mathscr{C}^{\mathrm{VI}}_{n} = \frac{2}{\sqrt{2n-1}}\left[\tau_j\iota_k\cos\frac{j(2k+1)\pi}{2n-1}\right]_{j,k=0}^{n-1},$$

where

$$\tau_{l(l=j,\ k)} = \begin{cases} 1, & \text{if} \quad l\neq 0 \quad \text{and} \quad l\neq n \\ \frac{1}{\sqrt{2}}, & \text{if} \quad l=0 \quad \text{or} \quad l=n, \end{cases} \quad \text{and} \quad \iota_k = \begin{cases} 1, & \text{if} \quad k\neq n-1 \\ \frac{1}{\sqrt{2}}, & \text{if} \quad k=n-1. \end{cases}$$

DEFINITION 2.3. *The* DST-I, DST-II, DST-V and DST-VI *matrices are defined as follows.*

$$\mathscr{S}^{\mathrm{I}}_{n-1} = \sqrt{\frac{2}{n}}\left[\sin\frac{jk\pi}{n}\right]_{j,k=1}^{n-1}, \qquad \mathscr{S}^{\mathrm{V}}_{n-1} = \frac{2}{\sqrt{2n-1}}\left[\sin\frac{2jk\pi}{2n-1}\right]_{j,k=1}^{n-1},$$

$$\mathscr{S}^{\mathrm{II}}_{n} = \sqrt{\frac{2}{n}}\left[\tau_j\sin\frac{j(2k-1)\pi}{2n}\right]_{j,k=1}^{n}, \qquad \mathscr{S}^{\mathrm{VI}}_{n-1} = \frac{2}{\sqrt{2n-1}}\left[\sin\frac{j(2k-1)\pi}{2n-1}\right]_{j,k=1}^{n-1},$$

where τ_j *is defined as in Definition* 2.2.

Note that all those transform matrices are all orthogonal.

2.2. Real Circulant Matrices. Let us first to investigate the real eigen-structure of a real circulant matrix C. It is shown in [5, 9] that the eigenvalues of a real circulant matrix can be arranged in the following order

1. $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \cdots, \lambda_{m-1}, \lambda_m, \bar{\lambda}_{m-1}, \cdots, \bar{\lambda}_1]^T$, where $\lambda_0, \lambda_m \in \mathbb{R}$ and $n = 2m$,
2. $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \cdots, \lambda_m, \bar{\lambda}_m, \cdots, \bar{\lambda}_1]^T$, where $\lambda_0 \in \mathbb{R}$ and $n = 2m+1$.

Partitioning $F^* = [\ \boldsymbol{f}_0, \cdots, \boldsymbol{f}_{n-1}]$, we have $\boldsymbol{f}_{n-k} = \bar{\boldsymbol{f}}_k$. For any eigenvalue λ_k, $C\boldsymbol{f}_k = \lambda_k\boldsymbol{f}_k$ means $C(\boldsymbol{f}_k + \bar{\boldsymbol{f}}_k) = \lambda_k\boldsymbol{f}_k + \bar{\lambda}_k\bar{\boldsymbol{f}}_k$ and $C(\boldsymbol{f}_k - \bar{\boldsymbol{f}}_k) = \lambda_k\boldsymbol{f}_k - \bar{\lambda}_k\bar{\boldsymbol{f}}_k$.

If we denote $\lambda_k = \alpha_k + i\beta_k$ and $\boldsymbol{f}_k = \hat{\boldsymbol{c}}_k + i\hat{\boldsymbol{s}}_k$, then we have

$$C[\ \hat{\boldsymbol{c}}_k,\ \hat{\boldsymbol{s}}_k\ ] = [\ \hat{\boldsymbol{c}}_k,\ \hat{\boldsymbol{s}}_k\ ]\begin{bmatrix} \alpha_k & \beta_k \\ -\beta_k & \alpha_k \end{bmatrix},$$

where α_k and β_k, $\hat{\boldsymbol{c}}_k$ and $\hat{\boldsymbol{s}}_k$ are the real and pure imaginary parts of λ_k and $\boldsymbol{f}_k$, for $k = 0, \cdots, n-1$.

Now, we show how to construct an orthogonal matrix U which transforms C into its real Schur form. Notice that $\hat{\boldsymbol{c}}_k = \hat{\boldsymbol{c}}_{n-k}$ and $\hat{\boldsymbol{s}}_k = -\hat{\boldsymbol{s}}_{n-k}$, for $k = 0, \cdots, n-1$, so we need only

normalize the first half of the vectors $\hat{\boldsymbol{c}}_k$ and $\hat{\boldsymbol{s}}_k$ to get an orthogonal U. Namely, U can be chosen as follows,

$$U = \begin{cases} \left[\ \hat{\boldsymbol{c}}_0, \sqrt{2}\ \hat{\boldsymbol{c}}_1, \cdots, \sqrt{2}\ \hat{\boldsymbol{c}}_{m-1}, \hat{\boldsymbol{c}}_m, \sqrt{2}\ \hat{\boldsymbol{s}}_{m-1}, \cdots, \sqrt{2}\ \hat{\boldsymbol{s}}_1\right], & n = 2m, \\ \left[\ \hat{\boldsymbol{c}}_0, \sqrt{2}\ \hat{\boldsymbol{c}}_1, \cdots, \sqrt{2}\ \hat{\boldsymbol{c}}_m, \sqrt{2}\ \hat{\boldsymbol{s}}_m, \cdots, \sqrt{2}\ \hat{\boldsymbol{s}}_1\right], & n = 2m+1. \end{cases} \tag{2.1}$$

A straightforward calculation shows that $U^T C U \equiv \Omega$ is real and has the following structure:

$$\Omega_{2m} = \left[\begin{array}{c|ccc|c|ccc} \alpha_0 & & & & & & & \\ \hline & \alpha_1 & & & & & & \beta_1 \\ & & \ddots & & & & \iddots & \\ & & & \alpha_{m-1} & & \beta_{m-1} & & \\ \hline & & & & \alpha_m & & & \\ \hline & & & -\beta_{m-1} & & \alpha_{m-1} & & \\ & & \iddots & & & & \ddots & \\ & -\beta_1 & & & & & & \alpha_1 \end{array}\right] \tag{2.2}$$

or

$$\Omega_{2m+1} = \left[\begin{array}{c|ccc|cc} \alpha_0 & & & & & \\ \hline & \alpha_1 & & & & \beta_1 \\ & & \ddots & & \iddots & \\ & & & \alpha_m & \beta_m & \\ \hline & & & -\beta_m & \alpha_m & \\ & & \iddots & & \ddots & \\ & -\beta_1 & & & & \alpha_1 \end{array}\right], \tag{2.3}$$

which can be transformed into the *real Schur canonical form* by a permutation. Therefore we also refer to (2.2) or (2.3) as the *real Schur form* of C. This leads to the following theorem.

THEOREM 2.4 (**Real Schur form of real circulant matrices**). *Let U be defined as in* (2.1). *If the circulant matrix C is real, then $U^T C U = \Omega$ is the real Schur form of C.*

Proof. The proof can be directly given by the above analysis and thus omitted. □

2.3. Real Skew-Circulant Matrices. Similarly, the eigenvalues of a real skew-circulant matrix S can be arranged in the following order

1. $\tilde{\boldsymbol{\lambda}} = [\tilde{\lambda}_0, \cdots, \tilde{\lambda}_{m-1}, \bar{\tilde{\lambda}}_{m-1}, \cdots, \bar{\tilde{\lambda}}_0]^T$, where $n = 2m$,
2. $\tilde{\boldsymbol{\lambda}} = [\tilde{\lambda}_0, \cdots, \tilde{\lambda}_{m-1}, \tilde{\lambda}_m, \bar{\tilde{\lambda}}_{m-1}, \cdots, \bar{\tilde{\lambda}}_0]^T$, where $\tilde{\lambda}_m \in \mathbb{R}$ and $n = 2m+1$.

The next procedure is very much like section 2.2. Let $\tilde{F}^* = [\ \tilde{\boldsymbol{f}}_0, \cdots, \tilde{\boldsymbol{f}}_{n-1}]$. Analogously, denoting $\tilde{\lambda}_k = \tilde{\alpha}_k + i\tilde{\beta}_k$ and $\tilde{\boldsymbol{f}}_k = \tilde{\boldsymbol{c}}_k + i\tilde{\boldsymbol{s}}_k$, then we have $S[\ \tilde{\boldsymbol{c}}_k,\ \tilde{\boldsymbol{s}}_k\ ] = [\ \tilde{\boldsymbol{c}}_k,\ \tilde{\boldsymbol{s}}_k\ ] \begin{bmatrix} \tilde{\alpha}_k & \tilde{\beta}_k \\ -\tilde{\beta}_k & \tilde{\alpha}_k \end{bmatrix}$.

Due to $\tilde{\boldsymbol{c}}_k = \tilde{\boldsymbol{c}}_{n-k-1}$ and $\tilde{\boldsymbol{s}}_k = -\tilde{\boldsymbol{s}}_{n-k-1}$, for $k = 0, \cdots, n-1$, we can choose an orthogonal

matrix $\tilde{U}$ as follows,

$$\tilde{U} = \begin{cases} \left[\ \sqrt{2}\ \tilde{\boldsymbol{c}}_0, \cdots, \sqrt{2}\ \tilde{\boldsymbol{c}}_{m-1}, \sqrt{2}\ \tilde{\boldsymbol{s}}_{m-1}, \cdots, \sqrt{2}\ \tilde{\boldsymbol{s}}_0\right], & \text{if } \ n = 2m, \\ \left[\ \sqrt{2}\ \tilde{\boldsymbol{c}}_0, \cdots, \sqrt{2}\ \tilde{\boldsymbol{c}}_{m-1}, \tilde{\boldsymbol{c}}_m, \sqrt{2}\ \tilde{\boldsymbol{s}}_{m-1}, \cdots, \sqrt{2}\ \tilde{\boldsymbol{s}}_0\right], & \text{if } \ n = 2m+1, \end{cases} \tag{2.4}$$

which makes $\tilde{U}^T S \tilde{U} \equiv \Sigma$ being of the following structure.

$$\Sigma_{2m} = \left[\begin{array}{ccc|ccc} \tilde{\alpha}_0 & & & & & \tilde{\beta}_0 \\ & \ddots & & & \iddots & \\ & & \tilde{\alpha}_{m-1} & \tilde{\beta}_{m-1} & & \\ \hline & & -\tilde{\beta}_{m-1} & \tilde{\alpha}_{m-1} & & \\ & \iddots & & & \ddots & \\ -\tilde{\beta}_0 & & & & & \tilde{\alpha}_0 \end{array}\right] \tag{2.5}$$

or

$$\Sigma_{2m+1} = \left[\begin{array}{ccc|c|ccc} \tilde{\alpha}_0 & & & & & & \tilde{\beta}_0 \\ & \ddots & & & & \iddots & \\ & & \tilde{\alpha}_{m-1} & & \tilde{\beta}_{m-1} & & \\ \hline & & & \tilde{\alpha}_m & & & \\ \hline & & -\tilde{\beta}_{m-1} & & \tilde{\alpha}_{m-1} & & \\ & \iddots & & & & \ddots & \\ -\tilde{\beta}_0 & & & & & & \tilde{\alpha}_0 \end{array}\right]. \tag{2.6}$$

Similarly, we refer to (2.5) or (2.6) as the *real Schur form* of S. Based on the above analysis, we conclude the following theorem.

THEOREM 2.5 (**Real Schur form of real skew-circulant matrices**). *Let $\tilde{U}$ be defined as in* (2.4). *If the skew-circulant matrix S is real, then $\tilde{U}^T S \tilde{U} = \Sigma$ is the real Schur form of S.*

We remark here that different from $C = F\Lambda F^*$ (i.e., C is factorized into the product of 3 complex matrices), Theorems 2.4 - 2.5 tell us that both C and S can be factorized into the products of 3 real matrices, respectively. This fact allows us to fast calculate matrix-vector multiplication and solve $C\boldsymbol{x} = \boldsymbol{b}$ and $S\boldsymbol{x} = \boldsymbol{b}$ by only real operations. In the next two sections, we will derive this strategy.

3. Fast Matrix-vector Multiplication. In this section, we first reduce the matrices U and $\tilde{U}$ into simpler forms by exploiting the structures of U and $\tilde{U}$, then show how to fast compute Ω in (2.2) or (2.3) and Σ in (2.5) or (2.6), and finally develop fast algorithms for computing $C\boldsymbol{x}$, $S\boldsymbol{x}$ and $T\boldsymbol{x}$.

Recall Definition 2.1, $P_{1n}\hat{\boldsymbol{c}}_k$ (remove the first entry of $\hat{\boldsymbol{c}}_k$) is a symmetric vector and $P_{1n}\hat{\boldsymbol{s}}_k$ (remove the first entry of $\hat{\boldsymbol{s}}_k$) is a skew-symmetric vector. Notice that if we delete the first row of U, then the first $m+1$ columns of the submatrix are all symmetric, and the last columns are all skew-symmetric. Therefore, the U of (2.1) can be partitioned into the following form

$$U_{2m} = \begin{bmatrix} \sigma_1 \boldsymbol{q}_{m+1}^T & \mathbf{0} \\ \hat{\mathscr{C}} & -\mathscr{S}_{m-1}^{\mathrm{I}} J_{m-1} \\ \sigma_1 \boldsymbol{v}_{m+1}^T & \mathbf{0} \\ J_{m-1}\hat{\mathscr{C}} & J_{m-1}\mathscr{S}_{m-1}^{\mathrm{I}} J_{m-1} \end{bmatrix} \text{ and } \quad U_{2m+1} = \begin{bmatrix} \sigma_1 \boldsymbol{p}_{m+1}^T & \mathbf{0} \\ \hat{\mathscr{C}} & -\mathscr{S}_m^{\mathrm{V}} J_m \\ J_m\hat{\mathscr{C}} & J_m\mathscr{S}_m^{\mathrm{V}} J_m \end{bmatrix}$$

where $\sigma_1 = \sqrt{\frac{2}{n}}$, $\sigma_2 = \frac{1}{\sqrt{2}}$, $\boldsymbol{p}_{m+1} = (\frac{1}{\sqrt{2}}, 1, \cdots, 1)^T$, $\boldsymbol{q}_{m+1} = (\frac{1}{\sqrt{2}}, 1, \cdots, 1, \frac{1}{\sqrt{2}})^T$, $\boldsymbol{v}_{m+1} = (\frac{1}{\sqrt{2}}, -1, \cdots, (-1)^{m-1}, \frac{(-1)^m}{\sqrt{2}})^T$, and

$$\hat{\mathscr{C}} = \begin{cases} \sigma_2 P_{1,m} \mathscr{C}_{m+1}^{\mathrm{I}} \in \mathbb{R}^{(m-1)\times(m+1)}, & \text{if } \ n = 2m, \\ \sigma_2 P_{1,m+1} \mathscr{C}_{m+1}^{\mathrm{V}} \ \in \mathbb{R}^{m\times(m+1)}, & \text{if } \ n = 2m+1. \end{cases}$$

Similarly, the $\tilde{U}$ of (2.4) can be partitioned into the following form

$$\tilde{U}_{2m} = \begin{bmatrix} \sigma_1 \boldsymbol{e}^T & \mathbf{0} \\ \tilde{\mathscr{C}} & \mathscr{S}_m^{\mathrm{II}} J_m \\ \mathbf{0} & \sigma_1 \boldsymbol{u}_m^T J_m \\ -J_{m-1}\tilde{\mathscr{C}} & J_{m-1}\mathscr{S}_m^{\mathrm{II}} J_m \end{bmatrix} \text{ and } \ \tilde{U}_{2m+1} = \begin{bmatrix} \sigma_1 \boldsymbol{p}_{m+1}^T & \mathbf{0} \\ \tilde{\mathscr{C}} & \mathscr{S}_m^{\mathrm{VI}} J_m \\ -J_m\tilde{\mathscr{C}} & J_m\mathscr{S}_m^{\mathrm{VI}} J_m \end{bmatrix}$$

where $\boldsymbol{u}_m = (1, -1, \cdots, (-1)^{m-1})^T$ and

$$\tilde{\mathscr{C}} = \begin{cases} \sigma_2 P_{1m} \mathscr{C}_m^{\mathrm{II}} \ \in \mathbb{R}^{(m-1)\times m}, & \text{if } \ n = 2m, \\ \sigma_2 P_{1m} \mathscr{C}_{m+1}^{\mathrm{VI}} \ \in \mathbb{R}^{m\times(m+1)}, & \text{if } \ n = 2m+1. \end{cases}$$

Now we construct an orthogonal matrix Q of the form

$$Q = \begin{cases} \frac{1}{\sqrt{2}} \begin{bmatrix} \sqrt{2} & & \\ & I_{m-1} & J_{m-1} \\ & -J_{m-1} & I_{m-1} \end{bmatrix} \begin{matrix} \\ \\ \\ \end{matrix}, \text{if } \ n = 2m, \\ \frac{1}{\sqrt{2}} \begin{bmatrix} \sqrt{2} & & \\ & I_m & J_m \\ & -J_m & I_m \end{bmatrix}, \qquad \text{if } \ n = 2m+1. \end{cases} \tag{3.1}$$

Then we can get the following simple formulas.

THEOREM 3.1. *Let U, $\tilde{U}$ and Q be defined as in* (2.1), (2.4) *and* (3.1), *respectively. Then we have*

$$QU = \begin{cases} \begin{bmatrix} \mathscr{C}_{m+1}^{\mathrm{I}} & \\ & J_m \mathscr{S}_{m-1}^{\mathrm{I}} J_m \end{bmatrix}, \text{if } \ n = 2m, \\ \begin{bmatrix} \mathscr{C}_{m+1}^{\mathrm{V}} & \\ & J_m \mathscr{S}_m^{\mathrm{V}} J_m \end{bmatrix}, \quad \text{if } \ n = 2m+1. \end{cases} \tag{3.2}$$

and

$$Q^T\tilde{U} = \begin{cases} \begin{bmatrix} \mathscr{C}_m^{\rm II} & \\ & J_m\mathscr{S}_m^{\rm II}J_m \end{bmatrix}, & \text{if } \ n = 2m. \\ \begin{bmatrix} \mathscr{C}_{m+1}^{\rm VI} & \\ & J_m\mathscr{S}_m^{\rm VI}J_m \end{bmatrix}, & \text{if } \ n = 2m+1. \end{cases} \tag{3.3}$$

Proof. The proof of this theorem can be completed by a tedious straightforward calculation and thus omitted. □

The Theorem 3.1 provides us a fast way to the calculation of $U\boldsymbol{x}$, it can be obtained by 1 DCT-I of $(m+1)$-vector and 1 DST-I of $(m-1)$-vector if $n = 2m$, and 1 DCT-V of $(m+1)$-vector and 1 DST-V of m-vector if $n = 2m+1$. The product $\tilde{U}\boldsymbol{x}$ can be obtained by a similar mode by employing the second and sixth versions of DCT and DST.

The entries of Ω and Σ can be computed by

$$\Omega U^T\boldsymbol{e}_1 = (QU)^TQC\boldsymbol{e}_1 \ \text{ and } \ \Sigma\tilde{U}^T\boldsymbol{e}_1 = (Q^T\tilde{U})^TQ^TS\boldsymbol{e}_1, \tag{3.4}$$

where $\boldsymbol{e}_1 = (1, 0, \cdots, 0)^T$.

The left-hand sides of (3.4) are as follows, respectively,

$$\sqrt{n}\Omega U^T\boldsymbol{e}_1 = \begin{cases} (\frac{\alpha_0}{\sqrt{2}}, \alpha_1, \cdots, \alpha_{m-1}, \frac{\alpha_m}{\sqrt{2}}, -\beta_{m-1}, \cdots, -\beta_1)^T, n = 2m, \\ (\frac{\alpha_0}{\sqrt{2}}, \alpha_1, \cdots, \alpha_m, -\beta_m, \cdots, -\beta_1)^T, \qquad n = 2m+1, \end{cases} \tag{3.5}$$

and

$$\sqrt{n}\Sigma\tilde{U}^T\boldsymbol{e}_1 = \begin{cases} (\tilde{\alpha}_0, \cdots, \tilde{\alpha}_{m-1}, -\tilde{\beta}_{m-1}, \cdots, -\tilde{\beta}_0)^T, \qquad n = 2m, \\ (\tilde{\alpha}_0, \cdots, \tilde{\alpha}_{m-1}, \frac{\tilde{\alpha}_m}{\sqrt{2}}, -\tilde{\beta}_{m-1}, \cdots, -\tilde{\beta}_0)^T, n = 2m+1. \end{cases} \tag{3.6}$$

This means we only need 1 DCT and 1 DST of about $\frac{n}{2}$-vector to get Ω or Σ.

Now, we show the calculations of $C\boldsymbol{x}$ and $S\boldsymbol{x}$ for any real n-vector $\boldsymbol{x}$ using DCT and DST.

According to Theorem 2.4 and Theorem 3.1, $C\boldsymbol{x}$ can be easily obtained by three DSTs and three DCTs (version I or V) of about $\frac{n}{2}$-vector. As for the storage required, we need one temporary n-vector and an extra n-vector for storing Ω. In fact, we don't need to compute and store U and Ω explicitly. It can be written as the following Algorithm 1.

If we compute $C\boldsymbol{x}$ by (1.2), it requires three FFTs of n-vector, and one temporary complex n-vector and an extra complex n-vector for storing Λ, equivalently, two temporary real n-vectors and two extra real n-vectors for storing Λ.

Similarly, according to Theorem 2.5 and Theorem 3.1, we develop the following Algorithm 2 for computing the product $S\boldsymbol{x}$, which can be obtained by three DSTs and three DCTs (version II or VI) of about $\frac{n}{2}$-vector.

From (1.3), we have that a Toeplitz matrix-vector multiplication $T\boldsymbol{x} = C\boldsymbol{x} + S\boldsymbol{x}$ can be fast calculated by employing the Algorithms 1 - 2.

Algorithm 1 To calculate $C\boldsymbol{x}$

1: Compute $\boldsymbol{v} = Q\boldsymbol{c}_1$ directly.
2: Compute $\hat{\boldsymbol{v}} = (QU)^T\boldsymbol{v}$ by DCT and DST.
3: Form Ω.
4: Compute $\boldsymbol{y}_1 = Q\boldsymbol{x}$ directly.
5: Compute $\boldsymbol{y}_2 = (QU)^T\boldsymbol{y}_1$ by DCT and DST.
6: Compute $\boldsymbol{y}_3 = \Omega\boldsymbol{y}_2$ directly.
7: Compute $\boldsymbol{y}_4 = (QU)\boldsymbol{y}_3$ by DCT and DST.
8: Compute $Q^T\boldsymbol{y}_4$, i.e., $C\boldsymbol{x}$.

Algorithm 2 To calculate $S\boldsymbol{x}$

1: Compute $\boldsymbol{u} = Q^T\boldsymbol{s}_1$ directly.
2: Compute $\hat{\boldsymbol{u}} = (Q^T\tilde{U})^T\boldsymbol{u}$ by DCT and DST.
3: Form Σ.
4: Compute $\boldsymbol{z}_1 = Q^T\boldsymbol{x}$ directly.
5: Compute $\boldsymbol{z}_2 = (Q^T\tilde{U})^T\boldsymbol{z}_1$ by DCT and DST.
6: Compute $\boldsymbol{z}_3 = \Sigma\boldsymbol{z}_2$ directly.
7: Compute $\boldsymbol{z}_4 = (Q^T\tilde{U})\boldsymbol{y}_3$ by DCT and DST.
8: Compute $Q\boldsymbol{z}_4$, i.e., $S\boldsymbol{x}$.

4. Solving $T\boldsymbol{x} = \boldsymbol{b}$ by the CSCS iteration. Consider the iterative solution to a large scale system of linear equations

$$T\boldsymbol{x} = \boldsymbol{b}, \tag{4.1}$$

where $T \in \mathbb{R}^{n\times n}$ is a Toeplitz matrix and $\boldsymbol{b} \in \mathbb{R}^n$.

Based on the splitting (1.3), Ng proposed in [10] the following CSCS iteration for solving (4.1).

The CSCS iteration: *Given an initial guess $\boldsymbol{x}^{(0)}$, for $k = 0, 1, \cdots$, until $\{\boldsymbol{x}^{(k)}\}$ converges, compute*

$$\begin{cases} (\theta I + C)\boldsymbol{x}^{(k+\frac{1}{2})} = (\theta I - S)\boldsymbol{x}^{(k)} + \boldsymbol{b}, \\ (\theta I + S)\boldsymbol{x}^{(k+1)} = (\theta I - C)\boldsymbol{x}^{(k+\frac{1}{2})} + \boldsymbol{b}, \end{cases} \tag{4.2}$$

where θ is a given positive constant.

It is shown in [10] that the CSCS iteration converges unconditionally, if both C and S are positive definite.

Then applying (1.2) to (4.2), we get

The FFT version of CSCS iteration: *Given an initial guess $\boldsymbol{x}^{(0)}$, for $k = 0, 1, \cdots$,*

until $\{\boldsymbol{x}^{(k)}\}$ *converges, compute*

$$\begin{cases} F(\theta I+\Lambda)F^*\boldsymbol{x}^{(k+\frac{1}{2})} = \tilde{F}(\theta I-\tilde{\Lambda})\tilde{F}^*\boldsymbol{x}^{(k)}+\boldsymbol{b}, \\ \tilde{F}(\theta I+\tilde{\Lambda})\tilde{F}^*\boldsymbol{x}^{(k+1)} = F(\theta I-\Lambda)F^*\boldsymbol{x}^{(k+\frac{1}{2})}+\boldsymbol{b}. \end{cases} \tag{4.3}$$

where θ *is a given positive constant.*

In the preparatory stage, two FFTs of n-vector for computing Λ and $\tilde{\Lambda}$ are required. In the iterative stage, six FFTs of n-vector for solving (4.3) are needed. Therefore the computational complexity is $O(n\log n)$ complex flops at each iteration. However, all operations, due to FFTs, are involved into complex arithmetics, even if T and $\boldsymbol{b}$ are real.

In this section, we develop the DCT-DST version of (4.2) based on the DCT and DST. Also, we compare the computational cost of our version with the FFT version of the CSCS iteration (4.3).

Note by Theorem 2.4 and Theorem 2.5 that the CSCS iteration (4.2) can be reformulated as the following form.

$$\begin{cases} U(\theta I+\Omega)U^T\boldsymbol{x}^{(k+\frac{1}{2})} = \tilde{U}(\theta I-\Sigma)\tilde{U}^T\boldsymbol{x}^{(k)}+\boldsymbol{b}, \\ \tilde{U}(\theta I+\Sigma)\tilde{U}^T\boldsymbol{x}^{(k+1)} = U(\theta I-\Omega)U^T\boldsymbol{x}^{(k+\frac{1}{2})}+\boldsymbol{b}. \end{cases} \tag{4.4}$$

The equation (4.4) can be further reduced into a simpler form due to Theorem 3.1. For example, consider the case $n=2m$ (The odd case is similar to the even case), we have the following version,

The DCT-DST version of CSCS iteration: *Given an initial guess* $\boldsymbol{x}^{(0)}\in\mathbb{R}^n$*, compute* $\boldsymbol{x}^{(k)}$*, for* $k=0,1,\cdots$*, until* $\{\boldsymbol{x}^{(k)}\}$ *converges:*

$$\begin{cases} Q^T\begin{bmatrix}\mathscr{C}^{\mathrm{I}}_{m+1} & \\ & \mathscr{S}^{\mathrm{I}}_{m-1}\end{bmatrix}(\theta I+\Omega)\begin{bmatrix}\mathscr{C}^{\mathrm{I}}_{m+1} & \\ & \mathscr{S}^{\mathrm{I}}_{m-1}\end{bmatrix}(Q\boldsymbol{x}^{(k+\frac{1}{2})}) \\ = Q\begin{bmatrix}\mathscr{C}^{\mathrm{II}}_{m} & \\ & \mathscr{S}^{\mathrm{II}}_{m}\end{bmatrix}(\theta I-\Sigma)\begin{bmatrix}\mathscr{C}^{\mathrm{II}}_{m} & \\ & \mathscr{S}^{\mathrm{II}}_{m}\end{bmatrix}^T(Q^T\boldsymbol{x}^{(k)})+\boldsymbol{b}, \\ Q\begin{bmatrix}\mathscr{C}^{\mathrm{II}}_{m} & \\ & \mathscr{S}^{\mathrm{II}}_{m}\end{bmatrix}(\theta I+\Sigma)\begin{bmatrix}\mathscr{C}^{\mathrm{II}}_{m} & \\ & \mathscr{S}^{\mathrm{II}}_{m}\end{bmatrix}^T(Q^T\boldsymbol{x}^{(k+1)}) \\ = Q^T\begin{bmatrix}\mathscr{C}^{\mathrm{I}}_{m+1} & \\ & \mathscr{S}^{\mathrm{I}}_{m-1}\end{bmatrix}(\theta I-\Omega)\begin{bmatrix}\mathscr{C}^{\mathrm{I}}_{m+1} & \\ & \mathscr{S}^{\mathrm{I}}_{m-1}\end{bmatrix}(Q\boldsymbol{x}^{(k+\frac{1}{2})})+\boldsymbol{b}, \end{cases} \tag{4.5}$$

where θ *is a given constant.*

We emphasize here that our version has the same convergence rate and optimal parameter as the CSCS iteration does.

The computational complexity. The iteration (4.5) consists of the preparatory stage and computational stage. In preparatory stage, we only need to calculate the matrices Ω and Σ which can be obtained by two DCTs and DSTs of about $n/2$-vector, see (3.4), (3.5) and (3.6).

In the computational stage, we need to compute the inverses of $\theta I+\Omega$ and $\theta I+\Sigma$, respectively. Because the matrices $\theta I+\Omega$ and $\theta I+\Sigma$ are of special structure, their inverses also keep the same structure as the original matrices and can be easily obtained which only cost $O(n)$ flops. The remaining operations are all matrix-vector multiplications. It takes $O(n)$ flops to calculate the products $Q\boldsymbol{v}$, $Q^T\boldsymbol{v}$, $(\theta I+\Omega)^{-1}\boldsymbol{v}$, $(\theta I-\Omega)\boldsymbol{v}$, $(\theta I+\Sigma)^{-1}\boldsymbol{v}$ and $(\theta I-\Sigma)\boldsymbol{v}$. Therefore, the main cost of each iteration is six DCTs and six DSTs of about $n/2$-vectors. As is well known, the complexity of DCT-I, DCT-II and DST-II of an n-vector is $\frac{1}{2}n\log n$ multiplications and $\frac{3}{2}n\log n$ additions, while DST-I of an n-vector requires $\frac{1}{2}n\log n$ multiplications and $2n\log n$ additions, see [2, 16, 14]. Thus the *computational complexity* of one iteration for solving (4.5) is $\frac{25}{2}n\log n$. Note that to perform a FFT of an n-vector requires $5n\log n$ flops, see, for example, [6]. Therefore, the *computational complexity* of one iteration for solving (4.3) is $30n\log n$. This means our method can save about half operations as compared with the FFT version of the CSCS iteration.

5. Numerical Examples. All the numerical tests are done on a Founder desktop PC with quad-core Intel(R) Core(TM) i7-4790 CPU 3.60 GHz with MATLAB 7.11.0(R2010b).

In all tests, we take the right-hand side $\boldsymbol{b}$ of (4.1) to be $(1,\cdots,1)^T$ and the initial guess $\boldsymbol{x}^{(0)}$ to be the zero vector. All tests are performed with double precision, and terminated when the current iterate satisfies $\frac{\|\boldsymbol{r}^{(k)}\|_2}{\|\boldsymbol{r}^{(0)}\|_2}\leq 10^{-7}$, or when the number of iterations is over 500, where $\boldsymbol{r}^{(k)}$ is the residual vector of the system (4.1) at the current iterate $\boldsymbol{x}^{(k)}$, and $\boldsymbol{r}^{(0)}$ is the initial one.

To show the effectiveness of our version (4.5), we give some comparisons of the elapsed CPU time among the iteration (4.5), the FFT version of CSCS iteration (4.3), and the AHSS iteration [4] for solving real positive definite Toeplitz systems, whose generating functions are listed as follows.

EXAMPLE 5.1. [10] $t_k=(1+\mid k\mid)^{-p}, k=0,\ \ \pm1,\cdots\pm(n-1)$.

EXAMPLE 5.2. [4] $f(x)=5+x^2+2cos(3x)+i(x+sinx),\ x\in[-\pi,\pi]$.

EXAMPLE 5.3. [4] $f(x)=10+8cosx+i2sin(5x),\ x\in[-\pi,\pi]$.

The generated Toeplitz matrix T_n is symmetric positive definite in Example 5.1, and non-symmetric positive definite in Examples 5.2-5.3. Therefore, all versions of the CSCS iteration are convergent unconditionally.

In all tables, we denote the order of the matrix T, the parameter of the iteration, spectral radius of the corresponding iteration matrix, the number of iterations by n, θ [1], ρ, N, respectively. Also, we denote the elapsed CPU times (10^{-2} seconds) of the FFT version of CSCS iteration, the DCT-DST version of CSCS iteration, the AHSS iteration based on FFT by t_s, t_r and t_a, respectively.

[1] We emphasize here that the true optimal parameter θ in the CSCS iteration is difficult to get, the parameters θ in all tables is experimentally approximately optimal. The parameters θ_1 and θ_2 in Table 5.3 and 5.4 are obtained from Corollary 4.1 in [4].

Table 5.1 Comparison between (4.3) and (4.5) for Example 5.1

n	p=0.9				p=1.1			
	θ	N	t_s	t_r	θ	N	t_s	t_r
4000	1.985	21	58.576	34.222	1.465	14	42.548	24.320
6000	2.095	22	140.69	71.187	1.555	14	86.545	46.634
8000	2.175	22	227.67	124.07	1.545	14	152.99	83.585

Table 5.2 Comparison between (4.3) and (4.5) for Examples 5.2-5.3

n	Example 5.2				Example 5.3			
	θ	N	t_s	t_r	θ	N	t_s	t_r
4000	3.680	5	17.133	10.108	3.890	9	29.907	18.628
6000	3.720	5	38.635	19.397	3.940	9	59.054	31.359
8000	3.705	5	66.787	32.603	3.925	8	95.722	49.814

The computational efficiency of two versions of CSCS iteration for solving Examples 5.1-5.3 is shown in the first two tables. As mentioned in the previous section, the spectral radius of iterative matrices of the two versions are the same, hence the numbers of iteration of them also remain the same. From Tables 5.1-5.2, we can see that when the order n of the matrix T becomes much larger, the iteration (4.5) works nearly twice as fast as the FFT version (4.3).

Also, we give a comparison of the computational efficiency between the iteration (4.5) and the AHSS iteration[2] developed by Gu [7] and Chen [4] who have tailored the HSS iteration proposed in [1] for real positive definite Toeplitz systems. Recall that any Toeplitz matrix T admits a Hermitian and skew-Hermitian splitting [7, 4] $T = H + \tilde{S}$, where $H = \frac{1}{2}(T + T^*)$ and $\tilde{S} = \frac{1}{2}(T - T^*)$. Then the resulting HSS iteration for solving real positive definite Toeplitz system is

The HSS iteration: *Given an initial guess* $\boldsymbol{x}^{(0)}$*, for* $k = 0, 1, \cdots$ *until* $\{\boldsymbol{x}^{(k)}\}$ *converges, compute*

$$\begin{cases} (\theta I + H)\boldsymbol{x}^{(k+\frac{1}{2})} = (\theta I - \tilde{S})\boldsymbol{x}^{(k)} + \boldsymbol{b}, \\ (\theta I + \tilde{S})\boldsymbol{x}^{(k+1)} = (\theta I - H)\boldsymbol{x}^{(k+\frac{1}{2})} + \boldsymbol{b}, \end{cases} \tag{5.1}$$

where θ *is a given positive constant.*

Because H is a symmetric Toeplitz matrix and $\tilde{S}$ is a skew-symmetric Toeplitz matrix, by using a unitary similarity transformation, each system of (5.1) can be reduced into two subsystems with about half sizes. Thus, one need to solve three Toeplitz-plus-Hankel subsystems. The complexity of each iteration is at least $O(n \log^2 n)$ if the superfast direct method is employed, and may be reduced to $O(\frac{n}{2} \log \frac{n}{2})$ if a preconditioned conjugate gradient method [4] is used. However, a good preconditioner is not easy to get. Moreover, the employ of FFTs makes the complex operations be involved for real system (4.1).

[2]The AHSS iteration in [4] involves two parameters and is faster than the HSS iteration proposed by Bai, et al. in [1].

Table 5.3 Comparison between (4.5) and AHSS iteration for Example 5.2

n	DCT-DST version			AHSS iteration		
	(θ, ρ)	N	t_r	$(\theta_1, \theta_2, \rho)$	N	t_a
256	(3.595, 0.1554)	6	1.7019	(7.1280, 7.1484, 0.2782)	9	2.0794
512	(3.765, 0.1656)	6	1.8638	(7.1444, 7.1550, 0.2813)	9	6.7190
1024	(3.865, 0.1718)	6	2.4665	(7.1532, 7.1586, 0.2830)	9	32.539

Table 5.4 Comparison between (4.5) and AHSS iteration for Example 5.3

n	DCT-DST version			AHSS iteration		
	(θ, ρ)	N	t_r	$(\theta_1, \theta_2, \rho)$	N	t_a
256	(3.585, 0.2806)	9	1.8882	(6.0035, 6.0005, 0.4916)	23	4.0687
512	(3.665, 0.2878)	9	2.0877	(6.0009, 6.0001, 0.4917)	23	11.418
1024	(3.735, 0.2971)	9	2.8626	(6.0002, 6.0000, 0.4917)	23	70.218

We remark here that the AHSS iteration does not work for symmetric case, Example 5.2 and 5.3 are used only. From the Tables 5.3-5.4, we can see that the spectral radius of the corresponding iteration matrix of (4.5) is much smaller than that of the AHSS iteration. Therefore, the number of iterations required of (4.5) is also less than that of the AHSS iteration. In particular, our iterative methods perform more efficiently for large n.

We must point out that in our tests, the functions $dct(\boldsymbol{x})$ and $dst(\boldsymbol{x})$, i.e., DCT-II and DST-I, are employed. In fact, in MATLAB, the DCT-II and DST-I are FFT-based algorithms for speedy computation. So the computational efficiency showed in practice is far lower than one in theoretical analysis, see the analysis of computational complexity in preceding sections. In case the real $dct(\boldsymbol{x})$ and $dst(\boldsymbol{x})$ are implemented in MATLAB in the future, it can be expected that our method will behave more efficiently.

6. Conclusion. In this paper, by exploiting the special eigen-structure of a real circulant matrix C and a real skew-circulant matrix S, we get their real Schur forms. Then, by means of DCT-DST, we further reduce the orthogonal matrices U and $\tilde{U}$ into simpler forms. As their applications, we first develop two new fast algorithms for computing matrix-vector multiplications $C\boldsymbol{x}$ and $S\boldsymbol{x}$, where only real arithmetics are involved. Then we reformulate the CSCS iteration for $T\boldsymbol{x} = \boldsymbol{b}$, and get the DCT-DST version of CSCS iteration (4.5), which only involves real arithmetics, is highly parallelizable and can be implemented on multiprocessors efficiently. Finally, some numerical examples are presented to show the reduction on the elapsed CPU times, compared with the iteration (4.3) and the AHSS iteration. In fact, our method can save a half storage and about half operations compared to the FFT version (4.3). Our proposed method shows obvious advantage especially in high order cases.

7. 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, and the Portuguese Funds through FCT-Fundacão para a Ciência, within the Project UID/ MAT/ 00013/2013.

REFERENCES

[1] Z. -Z. Bai, G. 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) pp. 603-626.

[2] V. Britanak, P. C. Yip and K.-R. Rao, *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*, Academic Press, Chennai, 2007.

[3] R. Chan and M. Ng, *Conjugate gradient methods for Toeplitz systems,* SIAM Rev. 38 (1996), pp. 427-482.

[4] F. Chen and Y.-L. Jiang, *On HSS and AHSS iteration methods for nonsymmetric positive definite Toeplitz systems*, J. Comput. Appl. Math., 234 (2010), pp. 2432-2440.

[5] M. T. Chu and R. J. Plemmons, *Real-valued, low rank, circulant approximation*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 645-659.

[6] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore and London, 3rd edition, 1996.

[7] C.-Q. Gu and Z.-L. Tian, *On the HSS iteration methods for positive definite Toeplitz linear systems*, J. Comput. Appl. Math., 224 (2009), pp. 709-718.

[8] G. Heinig and K. Rost, *Representations of Toeplitz-plus-Hankel matrices using trigonometric transformations with application to fast matrix-vector multiplication*, Linear Algebra Appl., 275-276 (1998), pp. 225-248.

[9] H. Karner, J. Schneid and C. W. Ueberhuber, *Spectral decomposition of real circulant matrices*, Linear Algebra Appl., 367 (2003), pp. 301-311.

[10] M. K. Ng, *Circulant and skew-circulant splitting methods for Toeplitz systems*, J. Comput. Applied Math., 159 (2003), pp. 101-108.

[11] K.-R. Rao and P. C. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Boston, 1990.

[12] G. Strang, *The Discrete Cosine Transform*, SIAM Review, 41 (1999), pp.135-147.

[13] Daniel B. Szyld, *An introduction to iterative Toeplitz solvers*, Math. Comput., 2009, Vol.78.

[14] M. Vetterli and H. Nussbaumer, *Simple FFT and DCT algorithms with reduced number of operations*, Signal Process., 6 (1984), pp. 267-278.

[15] Z. Wang and B. Hunt, *The discrete W-transform,* Appl. Math. Comput., 16 (1985), pp. 19-48.

[16] P. C. Yip and K. -R. Rao, *The Decimation-In-Frequency algorithms for a family of discrete sine and cosine transforms*, Circuits Systems and Signal Processing, 3 (1988) pp. 387-408.