CLASSIC TWO-STEP DURBIN-TYPE AND LEVINSON-TYPE ALGORITHMS FOR SKEW-SYMMETRIC TOEPLITZ MATRICES

IYAD T. ABU-JEIB

Abstract. We present efficient classic two-step Durbin-type and Levinson-type algorithms for even order skew-symmetric Toeplitz matrices.

AMS classifications: 65F05, 15A57
Keywords: skew-symmetric Toeplitz matrix; Levinson algorithm; Durbin algorithm; skew-centrosymmetric

1. Introduction

We extend the classic Durbin's algorithm and the classic Levinson's algorithm for symmetric Toeplitz matrices to skew-symmetric Toeplitz matrices. Levinson's algorithm is for solving the system $Ax = b$, where A is an $n \times n$ real symmetric Toeplitz matrix (with some restrictions on A) and x and b are $n \times 1$ vectors. In our algorithms, we present an $O(n^2)$ method to solve $HX = B$, where H is a finite $n \times n$ real skew-symmetric Toeplitz matrix (with some restrictions) and B is $n \times 2$ (to solve $Hx = b$, where b is $n \times 1$, simply let b be one of the columnns of B). This method can be used to invert nonsingular skew-symmetric Toeplitz matrices. Our algorithms are different than the algorithms presented in [?]. They are simple and they use very similar techniques to those used by Durbin and Levinson. They are easy to derive and easy to implement. They are also two-step. In addition, the restrictions we place on the matrix of coefficients are less than the restrictions in Durbin's algorithm and in Levinson's algorithm. Levinson's algorithm and a two-step version of it can be found in [?, ?, ?]. We recall here that well-known researchers thought that the classic Durbin's algorithm and the classic Levinson's algorithm can not be generalized to skew-symmetric Toeplitz matrices. But, we managed to generalize them. Our algorithms were tested on skew-symmetric Toeplitz matrices that appear in Sinc methods. They were tested on matrix S_n of $I_n^{(-1)}$, where

$$I_n^{(-1)} = [\eta_{ij}]_{i,j=1}^{n}$$

where $\eta_{ij} = e_{i-j}$, $e_k = \dfrac{1}{2} + s_k$, and $s_k = \displaystyle\int_0^k \operatorname{sinc}(x)dx$, where

$$\operatorname{sinc}(x) = \begin{cases} \dfrac{\sin(\pi x)}{\pi x}, & \text{for } x \neq 0 \\ 1, \text{ for } x = 0. \end{cases}$$

Thus, $I_n^{(-1)}$ can be expressed in the form

$$I_n^{(-1)} = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} + S_n,$$

where $\left[\frac{1}{2}\right]$ is the $n \times n$ matrix whose elements are all equal to $\frac{1}{2}$. We tested the algorithms also on matrix $I_n^{(1)}$ of Sinc methods. $I_n^{(1)}$ is an $n \times n$ skew-symmetric Toeplitz matrix defined as follows

$$I_n^{(1)} = \begin{bmatrix} 0 & -1 & \frac{1}{2} & ... & \frac{(-1)^{n-1}}{n-1} \\ 1 & 0 & -1 & ... & \frac{(-1)^{n-2}}{n-2} \\ -\frac{1}{2} & 1 & 0 & ... & \frac{(-1)^{n-3}}{n-3} \\ \vdots & \vdots & \vdots & ... & \vdots \\ \frac{(-1)^n}{n-1} & \frac{(-1)^{n-1}}{n-2} & \frac{(-1)^{n-2}}{n-3} & ... & 0 \end{bmatrix}.$$

Many Sinc methods are dependent on Sinc matrices. For more about the matrices of Sinc methods and about Sinc methods, see [?, ?, ?, ?, ?, ?].

2. Preliminaries

We employ the following notation. We denote the transpose of a matrix A by A^T. As usual, I_k denotes the $k \times k$ identity matrix. When counting flops, we treat addition/subtraction the same as multiplication/division. By the *main counterdiagonal* (or simply *counterdiagonal*) of a square matrix we mean the positions which proceed diagonally from the last entry in the first row to the first entry in the last row.

Definition 2.1. The *counteridentity* matrix, denoted J, is the square matrix whose elements are all equal to zero except those on the counterdiagonal, which are all equal to 1.

We note that multiplying a matrix A by J from the left results in reversing the rows of A and multiplying A by J from the right results in reversing the columns of A. Throughout this paper, we will denote the $k \times k$ counteridentity matrix by J_k. Note that multiplying a matrix or a vector by J does not contribute to the running time.

Definition 2.2. A matrix A is *skew-centrosymmetric* if $JAJ = -A$, and *Toeplitz* if the elements along each diagonal are equal.

Note that skew-symmetric Toeplitz matrices are skew-symmetric skew-centrosymmetric. Note also that if T_n is an $n \times n$ skew-symmetric Toeplitz matrix, then T_n has the following form

$$T_n = \begin{bmatrix} 0 & \sigma_1 & \sigma_2 & ... & \sigma_{n-1} \\ -\sigma_1 & 0 & \sigma_1 & ... & \sigma_{n-2} \\ -\sigma_2 & -\sigma_1 & 0 & ... & \sigma_{n-3} \\ \vdots & \vdots & \vdots & ... & \vdots \\ -\sigma_{n-1} & -\sigma_{n-2} & -\sigma_{n-3} & ... & 0 \end{bmatrix}.$$

The above form is the form we will refer to in the next section. Note that the first row (excluding the first element) *generates* (determines) T_n, i.e. the vector $h_n = [\sigma_1, \sigma_2, \cdots, \sigma_{n-1}]^T$ is a *generator* of T_n. We will assume that $\{T_n\}$, $n \in 2\mathbb{Z}^+$, is a family of real skew-symmetric Toeplitz matrices. Note that to solve $T_n X = B$, where T_n is $n \times n$ and B is $n \times 2$, we need T_{n+2}. To be specific, we need only the elements σ_n and σ_{n+1} from the generator h_{n+2} of T_{n+2}. This is not a shortcome of the algorithm because most of the skew-symmetric Toeplitz matrices

that arise in applications are of the form we mentioned above (i.e. they appear as classes/sequences of matrices) as it is the case with the skew-symmetric Toeplitz matrices that appear in Sinc methods. For example, in matrix S_n of $I_n^{(-1)}$ described in the introduction, $\sigma_k = \int_0^{-k} \text{sinc}(x)dx$, and in matrix $I_n^{(1)}$, $\sigma_k = \frac{(-1)^k}{k}$. Thus, σ_k is defined for all $k \in \mathbb{Z}^+$. Also, we recall that Durbin and Levinson used similar teqchniques in their algorithms; i.e., to solve $H_n x = b$, where H_n is an $n \times n$ real symmetric Toeplitz matrix, they use H_{n+1}.

Definition 2.3. Let A be an $n \times n$ matrix. The *leading principal matrix* of A of order k is the matrix formed from A by deleting the last $n-k$ columns and the last $n-k$ rows of A.

3. Algorithms for Skew-Symmetric Toeplitz Matrices

Throughout the rest of the paper, let k be even, and let T_k be a $k \times k$ real skew-symmetric Toeplitz matrix and assume T_i, $\forall i \in \{2, 3, \cdots, k\} \cap 2\mathbb{Z}$, is non-singular (i.e. all leading principal matrices of T_k of even order are nonsingular). We recall that Durbin and Levinson have stronger restrictions in their algorithms. We note also that it happens sometimes that all even-order matrices of a family of skew-symmetric Toeplitz matrices are non-singular as it is the case with matrix S_n of $I_n^{(-1)}$ and matrix $I_n^{(1)}$. For the proofs, see [?, ?]. Note that odd-order skew-symmetric (and odd-order skew-centrosymmetric matrices) are singular, and hence, it is essential to have a two-step algorithm that skips the odd-order matrices. Thus, our algorithm is a two-step algorithm because it moves from order k to order $k+2$ instead of moving from order k to order $k+1$. Now note that T_{k+2} can be written as

$$T_{k+2} = \begin{bmatrix} T_k & J_k R_k \\ -R_k^T J_k & T_2 \end{bmatrix},$$

where $R_k = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \\ \vdots & \vdots \\ \sigma_k & \sigma_{k+1} \end{bmatrix}$. Once again, in each step we will move from T_k to T_{k+2} instead of T_{k+1}. We start with T_2. Now, we extend Durbin's algorithm. If we know the solution of $T_k Y = R_k$, where Y is $k \times 2$, then we can know the solution of

$$\begin{bmatrix} T_k & J_k R_k \\ -R_k^T J_k & T_2 \end{bmatrix} \begin{bmatrix} Z \\ W \end{bmatrix} = \begin{bmatrix} R_k \\ S_k \end{bmatrix},$$

where Z is $k \times 2$, W is 2×2, and $S_k = \begin{bmatrix} \sigma_{k+1} & \sigma_{k+2} \\ \sigma_{k+2} & \sigma_{k+3} \end{bmatrix}$. Note that T_k and J_k are $k \times k$, R_k and Z are $k \times 2$, and T_2 and S_k are 2×2. Now note that

$$T_k Z + J_k R_k W = R_k, \text{ and } -R_k^T J_k Z + T_2 W = S_k.$$

Thus, $Z = Y + J_k Y W$ and $W = (T_2 - R_k^T Y)^{-1}(S_k + R_k^T J_k Y)$. Note that $T_2 - R_k^T Y$ is nonsingular, because

$$H_k^T T_{k+2} H_k = \begin{bmatrix} T_k & 0 \\ Y^T J_k T_k - R_k^T J_k & T_2 - R_k^T Y \end{bmatrix},$$

where

$$H_k = \begin{bmatrix} I_k & J_k Y \\ 0 & I_2 \end{bmatrix}.$$

(Note that $T_kJ_kY + J_kR_k = 0$ and $Y^TJ_kT_kJ_kY + Y^TR_k = 0$ also.) Hence, $\det(T_{k+2}) = \det(T_k) \cdot \det(T_2 - R_k^TY)$. Now since we are assuming T_{k+2} is nonsingular, then $T_2 - R_k^TY$ is nonsingular. Thus, our $O(n^2)$ algorithm is (we note that the inverse we have to find in this algorithm and in all other algorithms is for a 2×2 matrix):

Algorithm 1:

Input: n (an even positive integer), $\sigma = [\sigma_1\ \sigma_2\ \cdots\ \sigma_{n+1}]^T$ (a generator of a real skew-symmetric Toeplitz matrix T_{n+2}).

$$Y_2 = \begin{bmatrix} -\sigma_2/\sigma_1 & -\sigma_3/\sigma_1 \\ 1 & \sigma_2/\sigma_1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}$$

for $k = 2, \cdots, n-2$, *step 2*

Let J_k be the counteridentity matrix of order k.

$$R_k = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \\ \vdots & \vdots \\ \sigma_k & \sigma_{k+1} \end{bmatrix}$$

$$S_k = \begin{bmatrix} \sigma_{k+1} & \sigma_{k+2} \\ \sigma_{k+2} & \sigma_{k+3} \end{bmatrix}$$

$$P_k = (T_2 - R_k^TY_k)^{-1}$$

$$W_k = P_k(S_k + R_k^TJ_kY_k)$$

$$Z_k = Y_k + J_kY_kW_k$$

$$Y_{k+2} = \begin{bmatrix} Z_k \\ W_k \end{bmatrix}$$

end for

Output: Y_n (the solution of $T_nY_n = R_n$, where T_n is the skew-symmetric Toeplitz matrix whose generator is $[\sigma_1\ \sigma_2\ \cdots\ \sigma_{n-1}]^T$ and R_n is as before).

It is easy to see that the running time of the previous algorithm is $\sum_{k=2}^{n-2} 24k = 6n^2 + O(n)$.

Remark: We remind the reader that we define a flop to be one addition *or* one multiplication, while some people define it to be one addition *and* one multiplication. If we define the flop to be one addition and one multiplication, then the running time of the previous algorithm will be almost half of the running time we have above.

Now we derive a Levinson-type algorithm. Let $B = \begin{bmatrix} b_1 & c_1 \\ b_2 & c_2 \\ \vdots & \vdots \\ b_k & c_k \end{bmatrix}$. Assume we have the solution of $T_kX = B$, where X is $k \times 2$, and assume also we have the solution (from the previous algorithm) of $T_kY = R_k$, where Y is $k \times 2$. Now, we want to solve the next even higher order equation:

$$\begin{bmatrix} T_k & J_kR_k \\ -R_k^TJ_k & T_2 \end{bmatrix} \begin{bmatrix} V \\ M \end{bmatrix} = \begin{bmatrix} B \\ C \end{bmatrix},$$

where V is $k \times 2$, M and C are 2×2, and $C = \begin{bmatrix} b_{k+1} & c_{k+1} \\ b_{k+2} & c_{k+2} \end{bmatrix}$. Then, we will have

$$T_k V + J_k R_k M = B, \ -R_k^T J_k V + T_2 M = C.$$

Thus, $V = X + J_k Y M$ and $M = (T_2 - R_k^T Y)^{-1}(C + R_k^T J_k X)$. Now, all we need to do is to execute the steps above in parallel with the steps for solving $T_k Y = R_k$. Therefore, our $O(n^2)$ algorithm is:

Algorithm 2 (Classic Levinson-Type Algorithm):

Input: n (an even positive integer), $b = [b_1 \ b_2 \ \cdots \ b_n]^T$, $c = [c_1 \ c_2 \ \cdots \ c_n]^T$ (b and c are the vectors of constant terms), $\sigma = [\sigma_1 \ \sigma_2 \ \cdots \ \sigma_{n+1}]^T$ (a generator of a real skew-symmetric Toeplitz matrix T_{n+2}).

$Y_2 = \begin{bmatrix} -\sigma_2/\sigma_1 & -\sigma_3/\sigma_1 \\ 1 & \sigma_2/\sigma_1 \end{bmatrix}$

$T_2 = \begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}$

$X_2 = \begin{bmatrix} -b_2/\sigma_1 & -c_2/\sigma_1 \\ b_1/\sigma_1 & c_1/\sigma_1 \end{bmatrix}$

for $k = 2, \cdots, n-2$, *step 2*

Let J_k be the counteridentity matrix of order k.

$R_k = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \\ \vdots & \vdots \\ \sigma_k & \sigma_{k+1} \end{bmatrix}$

$S_k = \begin{bmatrix} \sigma_{k+1} & \sigma_{k+2} \\ \sigma_{k+2} & \sigma_{k+3} \end{bmatrix}$

$P_k = (T_2 - {R_k}^T Y_k)^{-1}$

$W_k = P_k(S_k + R_k^T J_k Y_k)$

$Z_k = Y_k + J_k Y_k W_k$

$M_k = P_k(C_k + R_k^T J_k X_k)$

$V_k = X_k + J_k Y_k M_k$

$Y_{k+2} = \begin{bmatrix} Z_k \\ W_k \end{bmatrix}$

$X_{k+2} = \begin{bmatrix} V_k \\ M_k \end{bmatrix}$

end for

Output: X_n (the solution of $T_n X_n = B_n$, where $B_n = [b, c]$ and T_n is the skew-symmetric Toeplitz matrix whose generator is $[\sigma_1 \ \sigma_2 \ \cdots \ \sigma_{n-1}]^T$).

It is easy to see that the running time of the previous algorithm is $\sum_{k=2}^{n-2} 40k = 10n^2 + O(n)$. Note that our algorithm solves for two vectors of constant terms at once. I.e. if we want to solve $T_n x = b$ and $T_n y = c$, where b and c are $n \times 1$ and n is even, then we solve the system $T_n Z = D$, where D is an $n \times 2$ matrix whose first column is b and second column is c. Then, x will be the first column of Z and y the second column. Solving $T_n Z = D$ by our method costs $10n^2 + O(n)$. Thus, obtaining x costs $5n^2 + O(n)$ and obtaining y has the same cost.

4. Examples

In the following example we solve (calculations are done by Octave which is a math-oriented programming language similar to MATLAB) $S_6 X = D$, where S_6 is

the 6×6 matrix of $I^{(-1)}$ mentioned in the introduction, i.e.

$$S_6 = \begin{bmatrix} 0.00000 & -0.58949 & -0.45141 & -0.53309 & -0.47497 & -0.52011 \\ 0.58949 & 0.00000 & -0.58949 & -0.45141 & -0.53309 & -0.47497 \\ 0.45141 & 0.58949 & 0.00000 & -0.58949 & -0.45141 & -0.53309 \\ 0.53309 & 0.45141 & 0.58949 & 0.00000 & -0.58949 & -0.45141 \\ 0.47497 & 0.53309 & 0.45141 & 0.58949 & 0.00000 & -0.58949 \\ 0.52011 & 0.47497 & 0.53309 & 0.45141 & 0.58949 & 0.00000 \end{bmatrix},$$

$$D = \begin{bmatrix} 1 & -3 \\ 2 & -7 \\ 3 & 6 \\ 4 & 4 \\ 5 & -8 \\ 6 & 2 \end{bmatrix},$$

and the generator σ of S_8 is $[-0.58949 \ -0.45141 \ -0.53309 \ -0.47497 \ -0.52011 \ -0.48321 \ -0.51442]^T$. Note that the inputs to the algorithm are 6, b, c, and σ, where b is the first column of D and c is the second column of D.

Initializations:

$$Y_2 = \begin{bmatrix} -0.76577 & -0.90433 \\ 1.00000 & 0.76577 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 0.00000 & -0.58949 \\ 0.58949 & 0.00000 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 3.3928 & -11.8747 \\ -1.6964 & 5.0891 \end{bmatrix}$$

Iterations:

k = 2:

$$R_2 = \begin{bmatrix} -0.58949 & -0.45141 \\ -0.45141 & -0.53309 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} -0.53309 & -0.47497 \\ -0.47497 & -0.52011 \end{bmatrix}$$

$$C_2 = \begin{bmatrix} 3 & 6 \\ 4 & 4 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} -3.0088e-17 & 1.2872e+00 \\ -1.2872e+00 & 9.1969e-17 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} -0.66695 & -0.49387 \\ 1.00000 & 0.66695 \end{bmatrix}$$

$$Z_2 = \begin{bmatrix} -0.66695 & -0.88747 \\ 0.60640 & 0.54082 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 3.8063 & 10.3397 \\ -3.1773 & -10.7611 \end{bmatrix}$$

$$V_2 = \begin{bmatrix} 4.7659 & -9.7754 \\ -1.7378 & 6.9029 \end{bmatrix}$$

$$Y_4 = \begin{bmatrix} -0.66695 & -0.88747 \\ 0.60640 & 0.54082 \\ -0.66695 & -0.49387 \\ 1.00000 & 0.66695 \end{bmatrix}$$

$$X_4 = \begin{bmatrix} 4.7659 & -9.7754 \\ -1.7378 & 6.9029 \\ 3.8063 & 10.3397 \\ -3.1773 & -10.7611 \end{bmatrix}$$

k = 4:

$$R_4 = \begin{bmatrix} -0.58949 & -0.45141 \\ -0.45141 & -0.53309 \\ -0.53309 & -0.47497 \\ -0.47497 & -0.52011 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} -0.52011 & -0.48321 \\ -0.48321 & -0.51442 \end{bmatrix}$$

$$C_4 = \begin{bmatrix} 5 & -8 \\ 6 & 2 \end{bmatrix}$$

$$P_4 = \begin{bmatrix} -1.1544e-16 & 1.2270e+00 \\ -1.2270e+00 & 8.3570e-17 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} -0.63828 & -0.42637 \\ 1.00000 & 0.63828 \end{bmatrix}$$

$$Z_4 = \begin{bmatrix} -0.63828 & -0.88814 \\ 0.53823 & 0.50995 \\ -0.51318 & -0.40723 \\ 0.53823 & 0.38486 \end{bmatrix}$$

$$M_4 = \begin{bmatrix} 4.6033 & 3.8665 \\ -4.6840 & 6.5775 \end{bmatrix}$$

$$V_4 = \begin{bmatrix} 6.2453 & -1.5221 \\ -2.4946 & 1.0757 \\ 4.0645 & 16.2416 \\ -2.0906 & -19.1772 \end{bmatrix}$$

$$Y_6 = \begin{bmatrix} -0.63828 & -0.88814 \\ 0.53823 & 0.50995 \\ -0.51318 & -0.40723 \\ 0.53823 & 0.38486 \\ -0.63828 & -0.42637 \\ 1.00000 & 0.63828 \end{bmatrix}$$

$$X_6 = \begin{bmatrix} 6.2453 & -1.5221 \\ -2.4946 & 1.0757 \\ 4.0645 & 16.2416 \\ -2.0906 & -19.1772 \\ 4.6033 & 3.8665 \\ -4.6840 & 6.5775 \end{bmatrix}$$

End of Iterations
The solution X is

$$\begin{bmatrix} 6.2453 & -1.5221 \\ -2.4946 & 1.0757 \\ 4.0645 & 16.2416 \\ -2.0906 & -19.1772 \\ 4.6033 & 3.8665 \\ -4.6840 & 6.5775 \end{bmatrix}.$$

We note that the solution above is exactly the same as the solution obtained from solving the system using Maple. The two solutions even match for a much higher number of decimal places.

As another example, we solve the system $I_8^{(1)}X = D$, where D is an 8×2 matrix whose first solumn is the 8×1 zero vector (call this vector b) and whose second column is $c = I_8^{(1)}e$, where e is the 8×1 vector of ones; i.e. (rounded up to 16 digits)

$$c = \begin{bmatrix} -0.759523809523809 \\ 0.383333333333333 \\ -0.283333333333333 \\ 0.250000000000000 \\ -0.250000000000000 \\ 0.283333333333333 \\ -0.383333333333333 \\ 0.759523809523809 \end{bmatrix}.$$

Note that D was chosen so that the true solution is the matrix whose first column is the 8×1 zero vector and whose second column is the 8×1 vector of ones. Solving the system by our algorithm with input 8, b, c, and σ, where

$$\sigma = \begin{bmatrix} -1 \\ 1/2 \\ -1/3 \\ 1/4 \\ -1/5 \\ 1/6 \\ -1/7 \\ 1/8 \\ -1/9 \end{bmatrix},$$

gives us the solution

$$\begin{bmatrix} 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \\ 0.000000000000000 & 1.000000000000000 \end{bmatrix}.$$

5. Improved Algorithms

First, we present a $4n^2+O(n)$ Durbin-type algorithm. The idea here is to reduce the cost of computing $R_k^T Y_k$. Here we will use the same notation (but we will replace Y by Y_k, Z by Z_k and W by W_k) we used when we derived the first two algorithms. Now consider

$$R_k^T Y_k = \begin{bmatrix} R_{k-2}^T & S_{k-2}^T \end{bmatrix} \begin{bmatrix} Z_{k-2} \\ W_{k-2} \end{bmatrix}.$$

But, $Z_{k-2} = Y_{k-2} + J_{k-2}Y_{k-2}W_{k-2}$. Thus,

$$R_k^T Y_k = R_{k-2}^T Y_{k-2} + (T_2 - R_{k-2}^T Y_{k-2})W_{k-2}^2.$$

Therefore, we can use the previously computed values of W and R^TY to compute the new value of R^TY which we will call E.

Algorithm 3 (Classic Durbin-Type Algorithm):

Input: n (an even positive integer), $\sigma = [\sigma_1\ \sigma_2\ \cdots\ \sigma_{n+1}]^T$ (a generator of a real skew-symmetric Toeplitz matrix T_{n+2}).

$Y_2 = \begin{bmatrix} -\sigma_2/\sigma_1 & -\sigma_3/\sigma_1 \\ 1 & \sigma_2/\sigma_1 \end{bmatrix}$

$T_2 = \begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}$

$R_2 = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \end{bmatrix}$

$E = R_2^T Y_2$

$W = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

for $k = 2, \cdots, n-2$, *step 2*

Let J_k be the counteridentity matrix of order k.

$R_k = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \\ \vdots & \vdots \\ \sigma_k & \sigma_{k+1} \end{bmatrix}$

$S_k = \begin{bmatrix} \sigma_{k+1} & \sigma_{k+2} \\ \sigma_{k+2} & \sigma_{k+3} \end{bmatrix}$

$E = E + (T_2 - E)W^2$

$P_k = (T_2 - E)^{-1}$

$W = P_k(S_k + R_k^T J_k Y_k)$

$Z_k = Y_k + J_k Y_k W$

$Y_{k+2} = \begin{bmatrix} Z_k \\ W \end{bmatrix}$

end for

Output: Y_n (the solution of $T_n Y_n = R_n$, where T_n is the skew-symmetric Toeplitz matrix whose generator is $[\sigma_1\ \sigma_2\ \cdots\ \sigma_{n-1}]^T$ and R_n is as before).

It is easy to see that the running time of the previous algorithm is $\sum_{k=2}^{n-2} 16k = 4n^2 + O(n)$.

Algorithm 4 (Improved Classic Levinson-Type Algorithm):

Input: n (an even positive integer), $b = [b_1\ b_2\ \cdots\ b_n]^T$, $c = [c_1\ c_2\ \cdots\ c_n]^T$ (b and c are the vectors of constant terms), $\sigma = [\sigma_1\ \sigma_2\ \cdots\ \sigma_{n+1}]^T$ (a generator of a real skew-symmetric Toeplitz matrix T_{n+2}).

$$Y_2 = \begin{bmatrix} -\sigma_2/\sigma_1 & -\sigma_3/\sigma_1 \\ 1 & \sigma_2/\sigma_1 \end{bmatrix}$$
$$T_2 = \begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}$$
$$X_2 = \begin{bmatrix} -b_2/\sigma_1 & -c_2/\sigma_1 \\ b_1/\sigma_1 & c_1/\sigma_1 \end{bmatrix}$$
$$R_2 = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \end{bmatrix}$$
$E = R_2^T Y_2$
$$W = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
for $k = 2, \cdots, n-2$, *step 2*

Let J_k be the counteridentity matrix of order k.

$$R_k = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \\ \vdots & \vdots \\ \sigma_k & \sigma_{k+1} \end{bmatrix}$$
$$S_k = \begin{bmatrix} \sigma_{k+1} & \sigma_{k+2} \\ \sigma_{k+2} & \sigma_{k+3} \end{bmatrix}$$
$E = E + (T_2 - E)W^2$
$P_k = (T_2 - E)^{-1}$
$W = P_k(S_k + R_k^T J_k Y_k)$
$Z_k = Y_k + J_k Y_k W$
$M_k = P_k(C_k + R_k^T J_k X_k)$
$V_k = X_k + J_k Y_k M_k$
$$Y_{k+2} = \begin{bmatrix} Z_k \\ W \end{bmatrix}$$
$$X_{k+2} = \begin{bmatrix} V_k \\ M_k \end{bmatrix}$$

end for

Output: X_n (the solution of $T_n X_n = B_n$, where $B_n = [b, c]$ and T_n is the skew-symmetric Toeplitz matrix whose generator is $[\sigma_1\ \sigma_2\ \cdots\ \sigma_{n-1}]^T$).

It is easy to see that the running time of the previous algorithm is $\sum_{k=2}^{n-2} 32k = 8n^2 + O(n)$. Note that our algorithm solves for two vectors of constant terms at once. I.e. if we want to solve $T_n x = b$ and $T_n y = c$, where b and c are $n \times 1$ and n is even, then we solve the system $T_n Z = D$, where D is an $n \times 2$ matrix whose first column is b and second column is c. Then, x will be the first column of Z and y the second column. Solving $T_n Z = D$ by our method costs $8n^2 + O(n)$. Thus, obtaining x costs $4n^2 + O(n)$ and obtaining y has the same cost.

Solving the first system we solved in Section **??** by the improved Levinson algorithm (using Octave) gives the same solution we got in that section. All variables we get here are the same as those in that example except that we do not have W_k here, and we have the following additional ones.

In the initializations part, we have

$$R_2 = \begin{bmatrix} -0.58949 & -0.45141 \\ -0.45141 & -0.53309 \end{bmatrix}$$

$$E = \begin{bmatrix} -5.5511e-17 & 1.8742e-01 \\ -1.8742e-01 & 1.8160e-17 \end{bmatrix}$$

$$W = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

When $k = 2$ in the iterations, we get

$$E = \begin{bmatrix} -5.5511e-17 & 1.8742e-01 \\ -1.8742e-01 & 1.8160e-17 \end{bmatrix}$$

$$W = \begin{bmatrix} -0.66695 & -0.49387 \\ 1.00000 & 0.66695 \end{bmatrix}.$$

When $k = 4$ in the iterations, we get

$$E = \begin{bmatrix} -5.8234e-17 & 2.2553e-01 \\ -2.2553e-01 & 1.5619e-17 \end{bmatrix}$$

$$W = \begin{bmatrix} -0.63828 & -0.42637 \\ 1.00000 & 0.63828 \end{bmatrix}.$$

6. An Octave Program for the Improved Levinson-Type Algorithm

We note that we do not have to compute the counteridentity matrix, J, in the program below. We can write the program without it. We included it in the program for the sake of clarity. We note also that the program can be shortened if we use Octave's built-in functions and operators. But, we decided to write it as above to make it easy to understand for readers who do not know Octave.

```
#
1;
function J = Counter(n)
# usage: J = Counter(n)
# description: Creates the counteridentity matrix of order n.
J=zeros(n);
for i=1:n
        J(i,n-i+1)=1;
endfor;
endfunction
function Z = solve(sigma,D)
# description : Solves the system T_nZ = D, where n is even and
# T_n is a real skew-symmetric Toeplitz matrix generated by
# [sigma(1) sigma(2) ... sigma(n-1)]^T.
# The input sigma = [sigma(1) sigma(2) ... sigma(n+1)]^T
# is the generator of T_{n+2}. It is an (n+1) x 1 column vector.
# The input D is an n x 2 matrix (matrix of constant terms).
# usage: Z = solve(sigma,D)
n = rows(sigma)- 1;
b = D(:,1); # b is the first column of D.
c = D(:,2); # c is the second column of D.
Y = [-sigma(2)/sigma(1),-sigma(3)/sigma(1);1,sigma(2)/sigma(1)];
T2 = [0,sigma(1);-sigma(1),0];
```

```
X = [-b(2)/sigma(1),-c(2)/sigma(1);b(1)/sigma(1),c(1)/sigma(1)];
R = [sigma(1),sigma(2);sigma(2),sigma(3)];
E = R' * Y; # The prime is used in Octave for transpose.
W = zeros(2,2); # W is the 2 × 2 zero matrix.
for k = 2:n-2
        if (rem(k, 2) != 0)
                continue;
        endif;
        R = zeros(k,2);
        for i=1:k
                R(i,1)=sigma(i);
                R(i,2) = sigma(i+1);
        endfor;
        S = zeros(2,2);
        S(1,1) = sigma(k+1);
        S(1,2) = sigma(k+2);
        S(2,1) = sigma(k+2);
        S(2,2) = sigma(k+3);
        C = zeros(2,2);
        C(1,1) = b(k+1);
        C(2,1) = b(k+2);
        C(1,2) = c(k+1);
        C(2,2) = c(k+2);
        J = Counter(k);
        E = E + (T2 - E) * W * W;
        P = inv(T2 - E);
        W = P * (S + R' * J * Y);
        Z = Y + J * Y * W;
        M = P * (C + R' * J * X);
        V = X + J * Y * M;
        Y = [Z;W];
        X = [V;M];
endfor;
Z = X;
endfunction;
```

References

[1] I. T. Abu-Jeib and T. S. Shores, *On properties of matrix $I^{(-1)}$ of Sinc methods,* New Zealand J. Math. **32** (2003) 1-10.

[2] D. Delsarte and Y. Genin, *The split Levinson algorithm,* IEEE Transactions on Acoustics Speech and Signal Processing ASSP-**34** (1986) 470-477.

[3] P. Gierke, *Ph.D. thesis,* University of Nebraska-Lincoln, 1999.

[4] G. Heinig and K. Rost, *Fast algorithms for skewsymmetric Toeplitz matrices,* Toeplitz matrices and singular integral equations (Pobershau, 2001) 193-208, Oper. Theory Adv. Appl., **135**, Birkhäuser, Basel, 2002.

[5] J. Lund and K. Bowers, *Sinc Methods for Quadrature and Differential Equations*, SIAM, Philadelphia, 1992.

[6] A. Melman, *The even-odd split Levinson algorithm for Toeplitz systems,* SIAM J. Matrix Anal. Appl. **23,** 1 (2001) 256-270.

[7] A. Melman, *A two step even-odd split Levinson algorithm for Toeplitz systems,* Linear Algebra Appl. **338** (2001) 219-237.
[8] F. Stenger, *Numerical Methods Based on Sinc and Analytic Functions*, Springer-Verlag, New York, 1993.
[9] F. Stenger, *Collocating convolutions*, Math. Comp. **64** (1995) 211-235.
[10] F. Stenger, *Matrices of sinc methods*, J. Comput. Appl. Math. **86** (1997) 297-310.

Department of Mathematics and Computer Science, Fenton Hall, SUNY College at Fredonia, Fredonia, NY 14063, USA
E-mail address: abu-jeib@cs.fredonia.edu