$O(n)$ working precision inverses for symmetric tridiagonal Toeplitz matrices with $O(1)$ floating point calculations.

Manuel Radons

November 29, 2016
Abstract: A well known numerical task is the inversion of large symmetric tridiagonal Toeplitz matrices, i.e., matrices whose entries equal $a$ on the diagonal and $b$ on the extra diagonals ($a, b \in \mathbb{R}$). The inverses of such matrices are dense and there exist well known explicit formulas by which they can be calculated in $O(n^2)$. In this note we present a simplification of the problem that has proven to be rather useful in everyday practice: If $|a| > 2|b|$, that is, if the matrix is strictly diagonally dominant, its inverse is a band matrix to working precision and the bandwidth is independent of $n$ for sufficiently large $n$. Employing this observation, we construct a linear time algorithm for an explicit tridiagonal inversion that only uses $O(1)$ floating point operations. On the basis of this simplified inversion algorithm we outline the cornerstones for an efficient parallelizable approximative equation solver.

Keywords Tridiagonal, symmetric, Toeplitz, Explicit inverse, Working precision, Linear time, Equation solving

0.1 Introduction

The inversion of symmetric tridiagonal Toeplitz matrices is a rather prominent task in applied mathematics. In general, the inverse of such matrices is dense and there exist explicit formulas, such as those presented in [1], that allow performance of the inversion in $O(n^2)$. However, for large $n$ ($\geq 100,000$), which one encounters frequently in numerical applications, this quadric complexity is too costly.

In the second section of this note we derive a simplified inversion formula for strictly diagonally dominant matrices of the (symmetric tridiagonal Toeplitz) form (0.1) and simplify it further in section three. We then use this simplified form to devise a Matlab style pseudocode for an explicit inversion with $O(n)$ read/write operations and integer additions (in the form of running indices) that uses only $O(1)$ floating point operations. In the final section we use the simplified inverse structure to outline an approximative equation solver that runs in roughly $4n$ fused multiply-adds and is communication free and thus parallelizable in a straightforward fashion.

0.2 First Simplification of Explicit Inversion

It is shown in [1, Cor. 4.2] that the inverse of an $n \times n$ tridiagonal matrix of the form

$$
T = \begin{bmatrix}
a & b \\
b & a & b \\
& \ddots & \ddots & \ddots \\
& & b & a & b \\
& & & b & a
\end{bmatrix},
$$

(0.1)
is given by the formula

\[
(T^{-1})_{ij} = \begin{cases} 
(-1)^{i+j} \frac{1}{\tau} \frac{U_{i-j}(a/2b)U_{n-i}(a/2b)}{U_n(a/2b)} & \text{if } i \leq j \\
(-1)^{i+j} \frac{1}{\tau} \frac{U_{i-j}(a/2b)U_{n-i}(a/2b)}{U_n(a/2b)} & \text{if } i > j
\end{cases}
\]

with

\[
U_n(x) = \frac{r_{+}^{n+1} - r_{-}^{n+1}}{r_{+} - r_{-}},
\]

where

\[
r_{\pm} = x \pm \sqrt{x^2 - 1}
\]

are the two solutions of the quadratic equation \(r^2 - 2xr + 1 = 0\). Let \(|x| > 1\), i.e., \(|a| > 2|b|\). Then the solutions \(r_{\pm}\) are real and we have

\[
r_{+}r_{-} = \left(x + \sqrt{x^2 - 1}\right)\left(x - \sqrt{x^2 - 1}\right) = x^2 - \left(\sqrt{x^2 - 1}\right)^2 = 1.
\]

Consequently,

\[
r_{+} = \frac{1}{r_{-}}, \text{ which implies } \log |r_{+}| = -\log |r_{-}|
\]

for arbitrary bases of the logarithm. Moreover, we either have

\[
|r_{+}| > 1 \text{ or } |r_{-}| > 1.
\]

We assume, without loss of generality, the first case, i.e., \(|r_{+}| > 1\). Moreover, we define

\[
\phi := \log_2 |r_{+}| > 0 \quad \text{and} \quad \tau := b(r_{+} - r_{-}).
\]

Let \(\delta \in \mathbb{N}_{>0}\). We call an arithmetic whose smallest representable magnitude is \(2^{-\delta}\) a \(\delta\)-arithmetic. In this sense, for example, single precision using subnormal representation is a 149-arithmetic (see, e.g., [4]).

**Definition 1.** We say \(s, t \in \mathbb{R}\) are numerically equal in \(\delta\)-arithmetic, if \(|s - t| < 2^{-\delta}\).

In the following we assume \(i \leq j\). Due to the symmetry of \(T\) – and thus of \(T^{-1}\) – analogous results follow for the case \(i > j\). Given said assumption it holds

\[
(T^{-1})_{ij} \overset{\text{(0.5)}}{=} (-1)^{i+j} \frac{1}{\tau} \frac{r_{+}^{n+i+j+1} - r_{-}^{n-i-j+1}}{r_{+}^{n+1} - r_{-}^{n+1}}
\]

\[
\overset{\text{(0.6)}}{=} (-1)^{i+j} \frac{1}{\tau} \frac{r_{+}^{n+i+j+1} - r_{-}^{n-i-j+1} - r_{+}^{n-i+j+1} + r_{-}^{n-i+j+1}}{r_{+}^{n+1} - r_{-}^{n+1}}
\]

Assume \(i \geq \left\lceil \frac{n}{2} \right\rceil\). We investigate the difference between the exact value (0.6) of \((T^{-1})_{ij}\) and the simplified term

\[
(-1)^{i+j} \frac{1}{\tau} \left(r_{+}^{i-j} - r_{-}^{2(n+1)+i+j}\right)
\]

for large \(n\).
Lemma 1. Let \( i \geq \left\lceil \frac{n}{2} \right\rceil \) and \( 10 \leq k \in \mathbb{N} \) and define
\[
N := \left\lceil \frac{k}{\phi} \rightceil. \tag{0.8}
\]

Then for all \( n \geq N \) it holds
\[
\left| (T^{-1})_{ij} - (-1)^{i+j} \frac{1}{\tau} \left( r_+^{i-j} - r_-^{2(n+1)+i+j} \right) \right| < 3 \left| \frac{1}{\tau 2^k} \right| \tag{0.9}
\]

Proof. The left hand term in (0.9) equals:
\[
\begin{align*}
(T^{-1})_{ij} - (-1)^{i+j} & \frac{1}{\tau} \left( r_+^{n+i-j+1} - r_-^{n+i-j-1} \right) \\
& = (T^{-1})_{ij} - \left( (T^{-1})_{ij} \frac{r_+^{n+1} - r_-^{n-1}}{r_+^{n+1}} - (-1)^{i+j} \frac{r_+^{2(n+1)-i+j} - r_-^{i-j}}{\tau} \right) \\
& = (T^{-1})_{ij} \frac{r_+^{2(n+1)} - (-1)^{i+j} \frac{r_+^{2(n+1)-i+j} - r_-^{i-j}}{\tau}}{r_+^{2(n+1)} - 1} \\
& \leq \frac{r_+^{i-j} - r_-^{i-j} - r_+^{2(n+1)+i+j} + r_+^{2(n+1)-i+j}}{\tau (r_+^{2(n+1)} - 1)} + \frac{r_+^{2(n+1)-i+j} - r_-^{i-j}}{\tau} \\
& =: \xi_1 + \xi_2.
\end{align*}
\]

Noting that \( 2^{\phi(n+1)} > 0 \), for all \( n \geq N \) it holds
\[
2^{1.9\phi n} < 2^{\phi(n+1)} - 1 \quad \text{and} \quad 2^k \leq 2^{\phi n}. \tag{0.10}
\]

Keeping in mind that by hypothesis \( i \geq \left\lceil \frac{n}{2} \right\rceil \), we have
\[
\begin{align*}
\xi_1 < & \frac{1}{|\tau|} \left| \frac{r_+^{i-j} - r_-^{i-j} - r_+^{2(n+1)+i+j} + r_+^{2(n+1)-i+j}}{r_+^{1.9n}} \right| \\
& < \frac{1}{|\tau|} \left| \frac{r_+^{0} - r_-^{n} - r_+^{2} + r_+^{1.5n}}{r_+^{1.9n}} \right| \\
& < 4 \left| \frac{r_-^{1.9n}}{\tau} \right| < \left| \frac{r_-^{n}}{\tau} \right|. \tag{0.11}
\end{align*}
\]

Moreover,
\[
\begin{align*}
\xi_2 & < \left| \frac{r_+^{1.5n} - r_-^{n}}{\tau} \right| < 2 \left| \frac{r_-^{n}}{\tau} \right|. \tag{0.12}
\end{align*}
\]

Together, (0.11) and (0.12) yield
\[
\xi_1 + \xi_2 < 3 \left| \frac{r_-^{n}}{\tau} \right| \leq 3 \left| \frac{1}{\tau 2^k} \right|.
\]

This completes the proof.
An analogous statement immediately follows for $i < \left\lfloor \frac{n}{2} \right\rfloor$.

**Lemma 2.** Let $i < \left\lfloor \frac{n}{2} \right\rfloor$ and $10 \leq k \in \mathbb{N}$ and define

$$
N := \left\lfloor \frac{k}{\varphi} \right\rfloor .
$$

(0.13)

Then for all $n \geq N$ it holds

$$
\left| (T^{-1})_{ij} - (-1)^{i+j} \frac{1}{T} (r_i^{i-j} - r_i^{-i-j}) \right| < 3 \left| \frac{1}{\tau 2^k} \right| .
$$

(0.14)

**Proof.** This follows from Lemma 1 and the persymmetry of $T$ and thus of $T^{-1}$. \hfill \square

Combining Lemma 1 and 2 we get:

**Proposition 1.** Assuming $i \leq j$, let

$$
k_\delta \geq \max(10, \delta + \log_2 |\tau| - \log_2 3) \quad \text{and} \quad N_\delta := \left\lfloor \frac{k_\delta}{\varphi} \right\rfloor .
$$

Moreover, define

$$
(T^{-1})_{ij} = \begin{cases}
(-1)^{i+j} \frac{1}{T} (r_i^{i-j} - r_i^{-i-j}) & \text{if } i < \left\lfloor \frac{n}{2} \right\rfloor \\
(-1)^{i+j} \frac{1}{T} (r_i^{i-j} - r_i^{-2(n+1)+i+j}) & \text{if } i \geq \left\lfloor \frac{n}{2} \right\rfloor .
\end{cases}
$$

Then, for all $n \geq N_\delta$, $(T^{-1})_{ij}$ and $(T^{-1})_{ij}$ are numerically equal in $\delta$-arithmetic.

### 0.3 Further Simplification and Algorithm

Due to the bisymmetry of $T$ – and thus of $T^{-1}$ – it suffices to calculate the entries of one of the four sections of the subdivision of $T^{-1}$ which is induced by the main- and the counter-diagonal. All other entries can be derived using the appropriate reflections. Our attention will thus be restricted to the case $n \geq N_\delta$, $i \leq j$ and $i + j \leq n + 1$, which especially means that $i \leq (n + 1)/2$.

The first choice in the design of an actual algorithm for the inverse calculation is to decide whether to first perform the subtraction of the terms in the brackets and then multiply the result with $(-1)^{i+j} \frac{1}{T}$ or vice versa. We investigate the options:

At first note that the value of the $r_i^{i-j}$ and thus of the $(-1)^{i+j} \frac{1}{T} r_i^{i-j}$ exclusively depends on the horizontal distance of the corresponding entry from the main diagonal. They numerically equal zero if either

$$
| i - j | < -\frac{\delta}{\varphi} \quad \text{or} \quad | i - j | < -\frac{\delta}{\varphi} + \frac{\log_2 |\tau|}{\varphi} =: -\alpha_\delta .
$$
respectively. Recalling that by assumption $|r_+| > 1$, we have

$$|r_+^{i-j}| > |r_+^{-i-j}|.$$  \hspace{1cm} (0.15)

Hence, if $r_+^{i-j} / \frac{1}{\tau} r_+^{-i-j}$ is numerically zero, then so is $r_+^{-i-j} / \frac{1}{\tau} r_+^{i-j}$ – which implies that for both possible approaches the $T_{\delta}^{-1}$ are band matrices with constant bandwidth, say $b_{r,\delta}$ [which is necessarily $\leq \max(\delta/\phi, \alpha_{\delta})$], for all $n \geq N_{\delta}$.

Moreover, since $i \leq j$ and thus $2i \leq i + j$, the terms $r_+^{-i-j}$ and $(-1)^{i+j} \frac{1}{\tau} r_+^{i-j}$ numerically equal zero if either

$$-2i < -\frac{\delta}{\phi} \quad \text{or} \quad -2i < -\alpha_{\delta}.$$  

This implies that for all $n \geq \max(N_{\delta}, \frac{\delta}{\phi}, \alpha_{\delta})$ the number of rows for which the $(T_{\delta}^{-1})_{ij}$ do not simply equal $(-1)^{i+j} \frac{1}{\tau} r_+^{i-j}$ is a constant, say $c_{r,\delta}$. The following pseudocode, which we will refine in the sequel, already establishes the statement of this article’s title: that the explicit inverse can be calculated, up to working precision, in $O(n)$, using only $O(1)$ floating point operations.

Algorithm 1 Outline explicit tridiagonal inversion

1: $T_{\delta}^{-1} = \text{zeros}(n, n)$
2: for $i = 1 : c_{r,\delta}$ do
3:     for $j = i : i + b_{r,\delta} - 1$ do
4:         $T_{\delta}^{-1}(i, j) = (-1)^{i+j} \frac{1}{\tau} (r_+^{i-j} - r_+^{-i-j})$
5:     end for
6: end for
7: OtherValues = $\text{zeros}(b_{r,\delta}, 1)$
8: for $i = 1 : b_{r,\delta}$ do
9:     OtherValues ($i$) = $(-1)^{1+i} \frac{1}{\tau} r_+^{1-i}$
10: end for
11: for $i = c_{r,\delta} + 1 : \left\lceil \frac{n}{2} \right\rceil$ do
12:     $T_{\delta}^{-1}(i, i : i + b_{r,\delta} - 1)$ = OtherValues($i$)
13: end for
14: Use symmetries of $T_{\delta}^{-1}$ to fill up the rest of the matrix.
15: return $T_{\delta}^{-1}$

As a first step towards the refinement of the algorithm outline we note that if $|x| > 1$, then $|x| > |\sqrt{x^2 - 1}|$. Which implies

$$\text{sign} \left( r_+ \right) = \text{sign} \left( x \right),$$

where sign denotes the signum function.

We now distinguish the two cases $x < 0$, which is the case if and only if $a$ and $b$ have different signs, and $x > 0$. 

6
\[ x < 0 : \] Then
\[ (-1)^{i+j}r_{-}^{i-j} = (-1)^{i+j}(-1)^{i-j}|r_+|^{i-j} = (-1)^{2i}|r_+|^{i-j} = |r_+|^{i-j}. \]

Recalling (0.15),
\[ \text{sign} \left( r_{-}^{i-j} - r_{+}^{i-j} \right) = \text{sign} \left( r_{+}^{i-j} \right). \]

Hence, \( T^{-1} \) is either nonnegative or nonpositive, depending on the sign of \( \tau \).
Moreover, \( i - j \) is odd if and only if \( i + j \) \((= i - j + 2j)\) is odd. Thus it holds
\[ \text{sign} \left( r_{+}^{i-j} \right) = \text{sign} \left( r_{-}^{i-j} \right), \]
which, by (0.16), implies \((-1)^{i+j}r_{+}^{i-j} = |r_+|^{-i-j}\) and thus
\[ (-1)^{i+j} \frac{1}{\tau} \left( r_{+}^{i-j} - r_{-}^{i-j} \right) = \frac{1}{\tau} \left( |r_+|^{i-j} - |r_+|^{-i-j} \right). \]

\[ x > 0 : \] Then
\[ (-1)^{i+j} \frac{1}{\tau} \left( r_{+}^{i-j} - r_{-}^{i-j} \right) = (-1)^{i+j} \frac{1}{\tau} \left( |r_+|^{i-j} - |r_+|^{-i-j} \right), \]
which implies that the diagonals of \( T^{-1} \) have alternating signs. The relevance of this
observation lies in the fact that it allows us to assign the signs diagonal-wise and thus
circumvent the effort of the sign calculations.

We may now answer the initial question, whether we should first subtract or multiply.
For a single term \( \frac{1}{\tau} \left( r_{+}^{i-j} - r_{-}^{i-j} \right) \) it saves a multiplication to perform the subtraction of
\( r_{+}^{i-j} \) and \( r_{-}^{i-j} \) first. But, defining \( b_{r,\delta} := \alpha_{\delta} \), the set
\[ B := \left\{ \frac{1}{\tau} r_{+}^{i-j} : 1 \leq j \leq b_{r,\delta} \right\} \]
contains all terms \( \frac{1}{\tau} r_{+}^{i-j} \) and \( \frac{1}{\tau} r_{-}^{i-j} \) that do not equal zero numerically. So it suffices to
perform \( b_{r,\delta} \) floating point multiplications to compute the values in \( B \).

As for the subtractions: Since it holds \((i-j)-(i-j) = 2i \geq 2\), let \( c_{r,\delta} := b_{r,\delta} - 2 \). The
number of tuples \((i,j)\) with \( i,j \geq 1 \) and \( i \leq j \) such that \( i + j \leq c_{r,\delta} \) – which is also the
number of \( \frac{1}{\tau} r_{+} - i - j \) that do not equal zero numerically and thus the number of floating
point subtractions to be performed – equals \( (c_{r,\delta} - 1) + (c_{r,\delta} - 2) + \cdots + 1 = c_{r,\delta}(c_{r,\delta} - 1)/2 \)
tuples \((i,j)\) such that \( i + j \leq l \).

Hence, performing the multiplications first and then the additions, we merely need to
perform \( \alpha_{\delta} \) floating point additions and \((\alpha_{\delta} - 2)(\alpha_{\delta} - 3)/2 \) floating point multiplications.

These observations are summarized in the refined pseudocode. Note that the algorithm
is optimized for a minimal operations count, not memory efficiency. In real life situations
one would certainly not store the inverse in an \( n \times n \) array. However, the building blocks
of the algorithm can be extracted and plugged into a memory efficient implementation.
Algorithm 2 Refined explicit tridiagonal inversion

1: procedure EXPLICIT INVERSE(a, b, n, δ)  \(\triangleright \) \(\delta \) is a precision parameter.
2: \(r_+ = a/2b\)
3: \(r = -r_+\)  \(\triangleright \) The sign switch allows to omit computation of the powers of \(-1\).
4: \(b_{r,\delta} = \alpha_\delta\)
5: Values = zeros(1, \(b_{r,\delta}\))
6: Values(1) = 1/\(\tau\)
7: for \(i = 2 : b_{r,\delta}\) do
8: \(\triangleright \) Inefficient. Will be stored differently in real life applications.
9: \(\triangleright \) 1st use of persymmetry.
10: \(T_\delta^{-1} = \text{zeros}(n, n)\)
11: for \(i = 1 : \left\lceil \frac{n}{2} \right\rceil \) do
12: \(\triangleright \) 1st use of symmetry.
13: end for
14: \(c_{r,\delta} = b_{r,\delta} - 2\)
15: for \(i = 1 : c_{r,\delta}\) do
16: \(\triangleright \) 2nd use of symmetry.
17: \(\triangleright \) 2nd loop instead of evaluation \(\min(1, i - b_{r,\delta} + 1)\).
18: for \(j = i : b_{r,\delta} - 1\) do
19: \(\triangleright \) 2nd loop instead of evaluation \(\min(1, i - b_{r,\delta} + 1)\).
20: \(\triangleright \) 2nd use of persymmetry.
21: \(\triangleright \) 2nd loop instead of evaluation \(\min(n, i + b_{r,\delta} - 1)\).
22: \(\triangleright \) 1st use of persymmetry.
23: end for
24: \(T_\delta^{-1}(i, j) = T_\delta^{-1}(j, i)\)
25: end for
26: for \(i = n - b_{r,\delta} + 1 : n\) do
27: \(\triangleright \) 1st use of symmetry.
28: end for
29: \(T_\delta^{-1}(i, j) = T_\delta^{-1}(j, i)\)
30: \(\triangleright \) 2nd use of symmetry.
31: end for
32: for \(j = i : n\) do
33: \(\triangleright \) 2nd loop instead of evaluation \(\min(n, i + b_{r,\delta} - 1)\).
34: \(T_\delta^{-1}(i, j) = T_\delta^{-1}(j, i)\)
35: \(\triangleright \) 2nd loop instead of evaluation \(\min(1, i - b_{r,\delta} + 1)\).
36: end for
37: return \(T_\delta^{-1}\)
38: end procedure
For this we note that \( A := T_\delta^{-1} \) can be decomposed into three blocks \( B, B', \) and \( C \) as follows: Let \( w \) be the number of rows in \( A \), where one or more nonzero entries are a sum (\( w \) is even due to the bisymmetry of \( A \)) and define

\[
d := \max \left( \frac{w}{2}, \frac{b_r, \delta}{2} \right).
\]

Then designate the first and last \( d \) rows of \( A \) as \( B \) and \( B' \), respectively. The block in between is \( C \). Clearly, \( B' \) equals \( B \) rotated by \( 180^\circ \). It thus suffices to store one of these blocks, which costs \( O(1) \) memory (more precisely \( O(\delta) \) – but this is the same for all practical purposes). Let \( d < i < n - d \). Then shifting the nonzero entries of the \( i \)-th row of \( A \) by one to the right yields row \( i + 1 \). Hence it suffices to store the nonzero entries of a single row of \( C \), which gives an overall memory requirement of \( O(1) \).

Moreover, a feature of the algorithm which has proven to be immensely practical is that said values have to be computed (and stored) only once and can then be used to construct the inverse for any sufficiently large \( n \).

### 0.4 Equation solving (outline) and outlook

Define \( A, B, B', C, d, r \) and the bandwidth \( b_r, \delta := \alpha \) as in the last section. We want to (approximatively) solve a linear system of the form

\[
T x = b \iff x = T^{-1} b \approx T_\delta^{-1} b = Ab.
\]

A naive but inefficient approach would be to simply multiply \( b \) with \( A \) (inefficient, since the bandwidth \( b_r, \delta \) of \( A \) is likely larger than the constants in current algorithms for the solution of tridiagonal systems such as those presented in \([2] \) and \([3] \)).

We propose another procedure: Decompose \( x \) into \( x^{(1)} := Bb, x^{(2)} := B'b \) and \( x^{(3)} := Cb \). The vectors \( x^{(1)} \) and \( x^{(2)} \) can be computed in \( O(\delta) \approx O(1) \). Let \( d < i < n - d \). Then

\[
x^{(3)}_{i-d} = e_i^T A b = \sum_{j=i-a}^{i+a} a_{ij} b_j = \sum_{j=i-\alpha}^{i} a_{ij} b_j + \sum_{j=i+1}^{i+a} a_{ij} b_j =: \varphi_i + \varphi'_i
\]

and it can easily be verified that

\[
x^{(3)}_{i-d+1} = \sum_{j=i-\alpha+1}^{i+1} a_{ij} b_j + \sum_{j=i+2}^{i+a+1} a_{ij} b_j
\]

\[
= \frac{\varphi_i - a_{i,i-\alpha+1} \cdot b_{i-\alpha}}{r} + r \cdot \varphi'_i + a_{i+1,i+\alpha+1} \cdot b_{i+\alpha+1}.
\]

This row-to-row update can be refined into a repeatedly applicable update scheme. For this we remark that the block \( C \) is Toeplitz. Now choose \( d < i < n - d \) arbitrarily and define

\[
b' := a_{i,i-\alpha} \cdot b \quad \text{and} \quad b'' := a_{ii} \cdot b.
\]
These rescalings cost $2n$ multiplications. (Albeit, since we are scaling vectors, these can likely be implemented more efficiently than $2n$ arbitrary multiplications.) Now define

$$
\varphi_{i+1} := \frac{\varphi_i - b_i' - \alpha r}{r} + b_i''_{i+1}
$$

and

$$
\varphi_i'_{i+1} := r \cdot \varphi_i' - b_{i+1}' + b_i + \alpha.
$$

Performing the updates in this manner costs $2n$ additions plus $2n$ fused multiply-adds plus another $n$ additions of the $\varphi_i$ and $\varphi_i'$. One might argue (this has to be investigated further) that the subtraction of $b_i' - \alpha$ in the first equality can be omitted due to its negligible numerical impact. This would spare $n$ subtractions and reduce the overall cost of the so-performed update scheme to roughly $4n + \mathcal{O}(1)$ multiplications and additions, which is about in line with the computational cost of the approximative tridiagonal Toeplitz solver presented in [3] – albeit the aforementioned algorithm by McNally et. al. does not require symmetry.

The proposed update scheme has the following advantages and disadvantages:

- **Pro**: Easy to implement.
- **Pro**: Fast.
- **Pro**: Communication free if block $C$ is decomposed further and thus parallelizable.
- **Con**: Possibly unstable.

To give definitive statements about the practical usefulness of the above update scheme, these aspects have to be be studied further. Another possibly productive line of investigation is the extension of the statements of this note to non-symmetric Toeplitz matrices. We will follow up on these questions.
Bibliography

[1] C.M. da Fonseca and J. Petronilho, Explicit inverses of some tridiagonal matrices, Linear Algebra and its Applications, 325, 7-21 (2001)

[2] R.W. Hockney, A fast direct solution of Poissons equation using Fourier analysis, Journal of the ACM, 12(1), 95-113 (1965)

[3] J.M. McNally, L.E. Garey and R.E. Shaw, A communication-less parallel algorithm for tridiagonal Toeplitz systems, Journal of Computational and Applied Mathematics, 212, 260 - 271 (2008)

[4] IEEE Task P754, ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic, pub-IEEE-STD, 1985