# Tridiagonal Matrix Algorithm

Tamas Kis | tamas.a.kis@outlook.com | https://tamaskis.github.io

## CONTENTS

1. Tridiagonal Matrix Algorithm 2
    1.1 Tridiagonal Linear Systems 2
    1.2 Tridiagonal Matrix Algorithm: Vector Implementation 2
    1.3 Tridiagonal Matrix Algorithm: Matrix Implementation 4
        1.3.1 Naive Version 4
        1.3.2 Better Version 5
        1.3.3 Best Version 6

References 7

---

Copyright © 2021 Tamas Kis

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
1 TRIDIAGONAL MATRIX ALGORITHM

1.1 Tridiagonal Linear Systems

A tridiagonal linear system is one of the form

\[ \mathbf{A} \mathbf{x} = \mathbf{d} \]  

(1)

where

\[
\begin{bmatrix}
    a_1 & b_1 & c_1 \\
    b_2 & a_2 & c_2 \\
    \vdots & \vdots & \vdots \\
    a_{n-2} & b_{n-1} & c_{n-2} \\
    a_{n-1} & b_n & c_{n-1}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_{n-1} \\
    x_n
\end{bmatrix}
=
\begin{bmatrix}
    d_1 \\
    d_2 \\
    \vdots \\
    d_{n-1} \\
    d_n
\end{bmatrix}
\]

(2)

and where \( \mathbf{A} \in \mathbb{R}^{n \times n} \) and \( \mathbf{x}, \mathbf{d} \in \mathbb{R}^n \). Owing to the fact that it only has three nonzero diagonals, the matrix \( \mathbf{A} \) is referred to as a tridiagonal matrix\(^1\). The tridiagonal vectors \( \mathbf{a} \in \mathbb{R}^{n-1}, \mathbf{b} \in \mathbb{R}^n, \) and \( \mathbf{c} \in \mathbb{R}^{n-1} \) are defined below in Eq. (3).

\[
\mathbf{a} = \begin{bmatrix}
    a_1 \\
    \vdots \\
    a_{n-1}
\end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix}
    b_1 \\
    \vdots \\
    b_n
\end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix}
    c_1 \\
    \vdots \\
    c_{n-1}
\end{bmatrix}
\]

(3)

These tridiagonal vectors form the tridiagonal matrix \( \mathbf{A} \), as shown in Eq. (1)\(^1\).

The tridiagonal matrix algorithm (also known as the Thomas algorithm) is an algorithm that can efficiently solve the tridiagonal linear system for \( \mathbf{x} \). There are two general implementations of this algorithm; one whose inputs are the tridiagonal vectors \( \mathbf{a}, \mathbf{b}, \) and \( \mathbf{c} \), and the other which operates directly on the tridiagonal matrix \( \mathbf{A} \). We name these algorithms accordingly:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reason for Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>tridiagonal_vector (Algorithm 1)</td>
<td>The tridiagonal vectors, ( \mathbf{a}, \mathbf{b}, ) and ( \mathbf{c} ), are input to this function.</td>
</tr>
<tr>
<td>tridiagonal_matrix (Algorithm 4)</td>
<td>The tridiagonal matrix, ( \mathbf{A} ), is input to this function.</td>
</tr>
</tbody>
</table>

Two additional algorithms (Algorithms 2 and 3) are also detailed, but these primarily serve as stepping stones towards developing Algorithm 4.

1.2 Tridiagonal Matrix Algorithm: Vector Implementation

The tridiagonal matrix algorithm defined by Algorithm 1 below is adapted from \([1, 3, 4]\).

---

\(^1\) In many references, a tridiagonal matrix is often defined with one of the following two convention:

\[
\mathbf{A} = \begin{bmatrix}
    a_1 & b_1 & c_1 \\
    a_2 & b_2 & c_2 \\
    \vdots & \vdots & \vdots \\
    a_{n-2} & b_{n-1} & c_{n-2} \\
    a_{n-1} & b_n & c_{n-1}
\end{bmatrix}
\]

The first is typically used when defining a tridiagonal linear system \([2]\), while the second is used almost exclusively when defining the tridiagonal matrix algorithm \([3, 4]\). However, for the second convention above, the \( a_i \)'s range from \( a_2 \) to \( a_{n-1} \), which is inconvenient from a programming standpoint; therefore, I defined them here as ranging from \( a_1 \) to \( a_{n-1} \). This convention is also reflected in Algorithms 1, 2, and 3.
**Algorithm 1: tridiagonal_vector**

Solves the tridiagonal linear system $Ax = d$ for $x$ using the vector implementation of the tridiagonal matrix algorithm.

**Inputs:**
- $a \in \mathbb{R}^{n-1}$ - tridiagonal vector
- $b \in \mathbb{R}^{n}$ - tridiagonal vector
- $c \in \mathbb{R}^{n-1}$ - tridiagonal vector
- $d \in \mathbb{R}^{n}$ - vector

**Procedure:**
1. Determine $n$, given that $d \in \mathbb{R}^{n}$.
2. Preallocate the vector $x \in \mathbb{R}^{n}$.
3. Forward elimination.
   
   ```plaintext
   for $i = 2$ to $n$
   
   \[ w = \frac{a_{i-1}}{b_{i-1}} \]
   \[ b_i = b_i - wc_{i-1} \]
   \[ d_i = d_i - wd_{i-1} \]
   
   end
   ```

4. Backward substitution.
   
   ```plaintext
   x_n = \frac{d_n}{b_n}
   
   for $i = n - 1$ to $1$ by $-1$
   
   \[ x_i = d_i - c_i x_{i+1} \]
   \[ b_i \]
   
   end
   ```

5. Solution of tridiagonal linear system.

   ```plaintext
   return x
   ```

**Outputs:**
- $x \in \mathbb{R}^{n}$ - solution of the tridiagonal linear system $Ax = d$

**Note:**
- The tridiagonal matrix ($A$) for the tridiagonal linear system ($Ax = d$) is defined in terms of the tridiagonal vectors ($a$, $b$, and $c$) as

   \[
   A = \begin{bmatrix}
   b_1 & c_1 &  &  & \\
   a_2 & b_2 & c_2 &  & \\
   & a_3 & \ddots & \ddots & \\
   & & \ddots & \ddots & \ddots \\
   a_{n-1} & b_{n-1} & c_{n-2} &  & \\
   & a_n & b_n & c_{n-1} & \\
   \end{bmatrix}
   \]
1.3 Tridiagonal Matrix Algorithm: Matrix Implementation

The tridiagonal matrix algorithm essentially takes the tridiagonal vector algorithm from Section 1.2 and adapts it to the case where we input the tridiagonal matrix \( (A) \) instead of the tridiagonal vectors \((a, b, \text{ and } c)\). The algorithms presented in Sections 1.3.1 and 1.3.2 are stepping stones towards the shortest implementation (i.e. the own that should actually be implemented in code) presented in Section 1.3.3.

1.3.1 Naive Version

The implementation of the tridiagonal matrix algorithm provided by Algorithm 2 is a rather naive one where we extract the tridiagonal vectors \((a, b, \text{ and } c)\) one-by-one from the tridiagonal matrix \(A\).

**Algorithm 2: tridiagonal_matrix_naive**

Solves the tridiagonal linear system \(Ax = d\) for \(x\) using the matrix implementation of the tridiagonal matrix algorithm (naive version).

**Inputs:**
- \(A \in \mathbb{R}^{n \times n}\) - tridiagonal matrix
- \(d \in \mathbb{R}^n\) - vector

**Procedure:**
1. Determine \(n\), given that \(d \in \mathbb{R}^n\).
2. Preallocate the vectors \(a \in \mathbb{R}^{n-1}\), \(b \in \mathbb{R}^{n}\), \(c \in \mathbb{R}^{n-1}\), and \(x \in \mathbb{R}^n\).
3. Extract \(a\) from \(A\).
   
   \[
   \text{for } i = 2 \text{ to } n \\
   \quad a_{i-1} = A_{i,i-1} \\
   \text{end}
   \]

4. Extract \(b\) from \(A\).
   
   \[
   \text{for } i = 1 \text{ to } n \\
   \quad b_i = A_{i,i} \\
   \text{end}
   \]

5. Extract \(c\) from \(A\).
   
   \[
   \text{for } i = 2 \text{ to } n \\
   \quad c_{i-1} = A_{i-1,i} \\
   \text{end}
   \]

   
   \[
   \text{for } i = 2 \text{ to } n \\
   \quad w = \frac{a_{i-1}}{b_{i-1}} \\
   \quad b_i = b_i - wc_{i-1} \\
   \quad d_i = d_i - wd_{i-1} \\
   \text{end}
   \]
7. Backward substitution.

\[
x_n = \frac{d_n}{b_n}
\]

for \( i = n - 1 \) to 1 by -1

\[
x_i = \frac{d_i - c_i x_{i+1}}{b_i}
\]

end

8. Solution of tridiagonal linear system.

return \( x \)

Outputs:

- \( x \in \mathbb{R}^n \) - solution of the tridiagonal linear system \( Ax = d \)

### 1.3.2 Better Version

We can save some computational effort by reducing the number of for loops in Algorithm 2. For smaller systems, this doesn’t make a huge impact, but for larger systems, it can halve the time it takes to solve. We can note that four of the loops go “forward” in \( i \), so we can combine them (with the caveat that we must extract \( b_1 \) separately since its loop starts from 1 and not 2). Defining this “better” algorithm,

**Algorithm 3: tridiagonal_matrix_better**

Solves the tridiagonal linear system \( Ax = d \) for \( x \) using the matrix implementation of the tridiagonal matrix algorithm (better version).

**Inputs:**

- \( A \in \mathbb{R}^{n \times n} \) - tridiagonal matrix
- \( d \in \mathbb{R}^n \) - vector

**Procedure:**

1. Determine \( n \), given that \( d \in \mathbb{R}^n \).
2. Preallocate the vectors \( a \in \mathbb{R}^{n-1} \), \( b \in \mathbb{R}^n \), \( c \in \mathbb{R}^{n-1} \), and \( x \in \mathbb{R}^n \).
3. Extract first element of \( b \) from \( A \).

\[
b_1 = A_{1,1}
\]

4. Forward loop.

\[
\text{for } i = 2 \text{ to } n
\]
(a) Extract relevant elements of $a$, $b$, and $c$ from $A$.

\[ a_{i-1} = A_{i,i-1} \]
\[ b_i = A_{i,i} \]
\[ c_{i-1} = A_{i-1,i} \]

(b) Forward elimination.

\[ w = \frac{a_{i-1}}{b_{i-1}} \]
\[ b_i = b_i - wc_{i-1} \]
\[ d_i = d_i - wd_{i-1} \]

5. Backward loop (backward substitution).

\[ x_n = \frac{d_n}{b_n} \]

\[ \text{for } i = n - 1 \text{ to } 1 \text{ by } -1 \]
\[ x_i = \frac{d_i - cx_{i+1}}{b_i} \]
\[ \text{end} \]

6. Solution of tridiagonal linear system.

\[ \text{return } x \]

**Outputs:**
- $x \in \mathbb{R}^n$ - solution of the tridiagonal linear system $Ax = d$

### 1.3.3 Best Version

Finally, instead of defining/preallocating the vectors $a$, $b$, and $c$, we can access the elements of $A$ directly. We refer to this as the “best” implementation; it is best in the sense that it requires the least lines of code. It is also generally faster than the implementation presented in Section 1.3.2.

**Algorithm 4: tridiagonal_matrix**

Solves the tridiagonal linear system $Ax = d$ for $x$ using the matrix implementation of the tridiagonal matrix algorithm.

**Inputs:**
- $A \in \mathbb{R}^{n \times n}$ - tridiagonal matrix
- $d \in \mathbb{R}^n$ - vector

**Procedure:**
1. Determine $n$, given that $d \in \mathbb{R}^n$.
2. Preallocate the vector $x \in \mathbb{R}^n$.
3. Forward elimination.
for $i = 2$ to $n$
\[
\begin{align*}
    w &= \frac{A_{i,i-1}}{A_{i-1,i-1}} \\
    A_{i,i} &= A_{i,i} - wA_{i-1,i} \\
    d_i &= d_i - wd_{i-1}
\end{align*}
\]
end

4. Backward substitution.
\[
x_n = \frac{d_n}{A_{n,n}}
\]
\[
\text{for } i = n - 1 \text{ to } 1 \text{ by } -1
\]
\[
x_i = \frac{d_i - A_{i,i+1}x_{i+1}}{A_{i,i}}
\]
end

5. Solution of tridiagonal linear system.

return $x$

Outputs:
- $x \in \mathbb{R}^n$ - solution of the tridiagonal linear system $Ax = d$
REFERENCES


