Tridiagonal Matrix Algorithm (Thomas Algorithm)

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

CONTENTS

1 TRIDIAGONAL MATRIX ALGORITHM (THOMAS ALGORITHM)

1.1 Tridiagonal Linear Systems

A **tridiagonal linear system** is one of the form

$$\boxed{\mathbf{A}\mathbf{x}=\mathbf{d}} \tag{1}$$

where

$$\boxed{\underbrace{\begin{bmatrix} b_1 & c_1 & & & & \\ a_1 & b_2 & c_2 & & & \\ & a_2 & \ddots & \ddots & & \\ & & \ddots & \ddots & c_{n-2} & \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}}_{\mathbf{d}}} \tag{2}$$

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 matrix algorithm** (also known as the **Thomas algorithm**) is an algorithm that can efficiently solve the tridiagonal linear system for $\mathbf{x}$. In this document, we introduce three different ways[2] to implement the tridiagonal matrix algorithm (Algorithms 1, 2, and 3). The first two implementations use three vectors, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, which we define as [1]

$$\boxed{\mathbf{a}=\begin{bmatrix} a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}, \quad \mathbf{b}=\begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \quad \mathbf{c}=\begin{bmatrix} c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}} \tag{3}$$

1.2 Slower Implementation

The tridiagonal matrix algorithm is shown below [1, 3, 4].

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

$$\mathbf{A}=\begin{bmatrix} a_1 & b_1 & & & & \\ c_1 & a_2 & b_2 & & & \\ & c_2 & \ddots & \ddots & & \\ & & \ddots & \ddots & b_{n-2} & \\ & & & c_{n-2} & a_{n-1} & b_{n-1} \\ & & & & c_{n-1} & a_n \end{bmatrix} \qquad \mathbf{A}=\begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & \ddots & \ddots & & \\ & & \ddots & \ddots & c_{n-2} & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix}$$

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, 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 and 2.

2 The `tridiagonal` function implements Algorithm 2.

Algorithm 1: `tridiagonal_slower`

Tridiagonal matrix algorithm (Thomas algorithm) (slower version).

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

Note:
- $\mathbf{A}$ and $\mathbf{d}$ define the tridiagonal linear system $\mathbf{Ax} = \mathbf{d}$.

Procedure:
1. Determine n, given that $\mathbf{d} \in \mathbb{R}^n$.
2. Preallocate vectors of size $n \times 1$ to store $\mathbf{b}$ and $\mathbf{x}$.
3. Preallocate vectors of size $(n-1) \times 1$ to store $\mathbf{a}$ and $\mathbf{c}$.
4. Extract $\mathbf{a}$ from $\mathbf{A}$.

   for $i = 2$ **to** n
   $\quad a_{i-1} = A_{i,i-1}$
   end

5. Extract $\mathbf{b}$ from $\mathbf{A}$.

   for $i = 1$ **to** n
   $\quad b_i = A_{i,i}$
   end

6. Extract $\mathbf{c}$ from $\mathbf{A}$.

   for $i = 2$ **to** n
   $\quad c_{i-1} = A_{i-1,i}$
   end

7. Forward elimination.

   for $2 = 1$ **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

8. 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

Return:

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

1.3 Faster Implementation

We can save some computational effort by reducing the number of for loops in Algorithm 1. 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 "faster" algorithm,

Algorithm 2: `tridiagonal`
Tridiagonal matrix algorithm (Thomas algorithm).

Given:

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

Note:

- $\mathbf{A}$ and $\mathbf{d}$ define the tridiagonal linear system $\mathbf{Ax} = \mathbf{d}$.

Procedure:

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

$$b_1 = A_{1,1}$$

5. Forward loop.

for $i = 2$ **to** n

(a) Extract relevant elements of $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ from $\mathbf{A}$.

$$a_{i-1} = A_{i,i-1}$$
$$b_i = A_{i,i}$$
$$c_{i-1} = A_{i-1,i}$$

(b) Forward elimination.

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

end

6. Backward loop (backward substitution).

$$x_n = \frac{d_n}{b_n}$$

$$\textbf{for } i = n-1 \textbf{ to } 1 \textbf{ by } -1$$
$$x_i = \frac{d_i - c_i x_{i+1}}{b_i}$$
$$\textbf{end}$$

Return:

- $\mathbf{x} \in \mathbb{R}^n$ - solution of the tridiagonal linear system $\mathbf{A}\mathbf{x} = \mathbf{d}$

1.4 Shortest Implementation

Finally, instead of defining/preallocating the vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, we can access the elements of $\mathbf{A}$ directly. We refer to this as the "shortest" implementation since it results in the least lines of code.

Algorithm 3: `tridiagonal_shortest`

Tridiagonal matrix algorithm (Thomas algorithm) (shortest implementation).

Given:

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

Note:

- $\mathbf{A}$ and $\mathbf{d}$ define the tridiagonal linear system $\mathbf{A}\mathbf{x} = \mathbf{d}$.

Procedure:

1. Determine n, given that $\mathbf{d} \in \mathbb{R}^n$.
2. Preallocate a vector of size $n \times 1$ to store $\mathbf{x}$.
3. Forward elimination.

$$\textbf{for } i = 2 \textbf{ to } n$$
$$w = \frac{A_{i,i-1}}{A_{i-1,i-1}}$$
$$A_{i,i} = A_{i,i} - wA_{i-1,i}$$
$$d_i = d_i - wd_{i-1}$$
$$\textbf{end}$$

4. Backward substitution.

$$x_n = \frac{d_n}{A_{n,n}}$$
$$\textbf{for } i = n-1 \textbf{ to } 1 \textbf{ by } -1$$
$$x_i = \frac{d_i - A_{i,i+1} x_{i+1}}{A_{i,i}}$$
$$\textbf{end}$$

Return:

- $\mathbf{x} \in \mathbb{R}^n$ - solution of the tridiagonal linear system $\mathbf{A}\mathbf{x} = \mathbf{d}$

For smaller systems, this implementation can actually be the fastest, since you only have to preallocate one vector ($\mathbf{x}$) instead of four ($\mathbf{x}$, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$). However, for large systems, it is costlier to traverse the matrix $\mathbf{A}$ during the substitution process than it is to preallocate, define, and traverse the vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$.

REFERENCES

[1] James Hateley. *Linear Systems of Equations and Direct Solvers*. MATH 3620 Course Reader (Vanderbilt University). 2019.

[2] *Tridiagonal matrix algorithm*. Wikipedia. Accessed: December 14, 2021. URL: https://en.wikipedia.org/wiki/Tridiagonal_matrix.

[3] *Tridiagonal matrix algorithm*. Wikipedia. Accessed: January 9, 2021. URL: https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm.

[4] *Tridiagonal matrix algorithm – TDMA (Thomas algorithm)*. CFD Online. Accessed: January 9, 2021. URL: https://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm).