On the Solution of an Almost Tridiagonal Linear Equations

Xingbo Wang[1,2], Jue Wang[3*]

[1] Department of Mechatronics, Foshan University, Guangdong, China.
[2] Guangdong College of Applied Science and Technology, Guangdong, China.
[3] Shenzhen Smoore Technology Co., Ltd., Guangdong, China.

* Corresponding author. Tel.: 8618928691407; email: jue.wang@smooretech.com
Manuscript submitted December 30, 2022; revised February 8, 2023; accepted March 10, 2023.
doi: 10.17706/ijapm.2023.13.3.34-43

Abstract: This paper makes an investigation on solving a kind of almost tridiagonal linear equations. By means of exploring more general a system of the Top-Bottom-Bordered Tridiagonal (TBBT) linear equations, the paper obtains an algorithm that costs less than the LU decomposition approach to solve the almost tridiagonal linear equations. Detail mathematical reasoning is presented to derive and analyze the algorithm. Numerical experiments show the algorithm can be applied to engineering application within expectation.

Keywords: Tridiagonal linear equations; Woodbury matrix identity, matrix Inverse, engineering computation

1. Introduction

When constructing a uniform cubic spline interpolation curves with the natural boundaries, the following linear Eq. (1) always occur.

$$\frac{1}{6}\begin{pmatrix} 6 & -12 & \boxed{6} & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & \vdots & 0 & 0 & \boxed{6} & -12 & 6 \end{pmatrix}\begin{pmatrix} \boldsymbol{P}_0 \\ \boldsymbol{P}_1 \\ \boldsymbol{P}_2 \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{P}_{n-1} \\ \boldsymbol{P}_n \\ \boldsymbol{P}_{n+1} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{Q}_1 \\ \boldsymbol{Q}_3 \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{Q}_{n-1} \\ \boldsymbol{Q}_n \\ 0 \end{pmatrix} \tag{1}$$

The coefficient matrix of the equations looks like a tridiagonal linear system (TLS) but actually not due to the two marked terms. It of course can be regarded to be a special case of the following matrix (2):

$$\begin{pmatrix} b_1 & c_1 & h_1 & h_2 & h_3 & \cdots & h_{n-3} & h_{n-2} \\ a_2 & b_2 & c_2 & & & & & \\ & a_3 & b_3 & c_3 & & & & \\ & & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & \\ & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ v_1 & v_2 & v_3 & \cdots & v_{n-3} & v_{n-2} & a_n & b_n \end{pmatrix} \tag{2}$$

This matrix as well as the following Eqs. (3)–(5) is called bordered tridiagonal matrices.

$$\begin{pmatrix} b_1 & c_1 & & & & & & h_1 \\ a_2 & b_2 & c_2 & & & & & h_2 \\ & a_2 & b_3 & c_3 & & & & h_3 \\ & & & \ddots & \ddots & & & \vdots \\ & & & \ddots & \ddots & \ddots & & h_{n-3} \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & h_{n-2} \\ & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ v_1 & v_2 & v_3 & \cdots & v_{n-3} & v_{n-2} & a_n & b_n \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} b_1 & c_1 & & & & & & u_1 \\ a_2 & b_2 & c_2 & & & & & u_2 \\ v_1 & a_3 & b_3 & c_3 & & & & u_3 \\ v_2 & & \ddots & \ddots & \ddots & & & \vdots \\ v_3 & & & \ddots & \ddots & \ddots & & u_{n-3} \\ \vdots & & & & a_{n-2} & b_{n-2} & c_{n-2} & u_{n-2} \\ v_{n-3} & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ v_{n-2} & & & \cdots & & & a_n & b_n \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} b_1 & c_1 & h_1 & h_2 & h_3 & \cdots & h_{n-3} & h_{n-2} \\ a_2 & b_2 & c_2 & & & & & \\ v_1 & a_3 & b_3 & c_3 & & & & \\ v_2 & & \ddots & \ddots & \ddots & & & \\ v_3 & & & \ddots & \ddots & \ddots & & \\ \vdots & & & & a_{n-2} & b_{n-2} & c_{n-2} & \\ v_{n-3} & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ v_{n-2} & & & \cdots & & & a_n & b_n \end{pmatrix} \tag{5}$$

The bordered tridiagonal matrices have frequently raised interests of researchers. Wang [1], Martin and Boyd [2], Karawia [3], El-Mikkawy and Atlan [4, 5], Jia and Li [6] researched (3), Martin and Boyd [2], Atlan and El-Mikkawy [5] investigated (4) and El-Mikkawy and Atlan [4] also investigated (5), leaving (2) unexplored. To meet the needs of theoretical study and engineering computing, this paper investigates (2). We first express the matrix (2) by the sum of a tridiagonal matrix A and the product of two matrices U and V, then find the inverse of matrix by the Woodbury's matrix identity. By such means, we find an algorithm faster than the LU factorization. The later sections introduce the details.

2. Woodbury Matrix Identity

The Woodbury matrix identity, or Woodbury formula, is one of the most applicable lemmas in finding the inverse of regular matrices, especially the sparse matrices. From classical textbooks like [7–9] to recent professional research [10], it can be found here and there. Suppose C is a regular $n \times n$ matrix and

$$C = A + UV^T \tag{6}$$

where A is an $n \times n$ matrix, U and V are n × m matrices with $m < n$; the Woodbury matrix identity states

$$C^{-1} = (A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1} \tag{7}$$

As a result, the solution of a linear system

$$C\boldsymbol{x} = \boldsymbol{f} \tag{8}$$

where x and f are $n \times 1$ matrices, can be expressed by

$$\boldsymbol{x} = (A+UV^T)^{-1}\boldsymbol{f} = A^{-1}\boldsymbol{f} - [A^{-1}U(I+V^TA^{-1}U)^{-1}V^TA^{-1}]\boldsymbol{f} \tag{9}$$

3. Problem and Solutions

Now back to the solution of (1), which is denoted by (8). For the purpose of solving more general cases, we consider the coefficient matrix C to be of the form (2), namely

$$C = \begin{pmatrix} b_1 & c_1 & h_1 & h_2 & h_3 & \cdots & h_{n-3} & h_{n-2} \\ a_2 & b_2 & c_2 & & & & & \\ & a_3 & b_3 & c_3 & & & & \\ & & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & \\ & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ v_1 & v_2 & v_3 & \cdots & v_{n-3} & v_{n-2} & a_n & b_n \end{pmatrix} \tag{10}$$

where $h_i, v_i (i = 1, 2, \ldots, n-2)$ are real numbers and

$$\begin{cases} |b_1| > |c_1| > 0 \\ |b_i| \geq |a_i| + |c_i| \quad a_i c_i \neq 0 (i = 2, 3, \cdots, n-1) \\ |b_n| > |a_n| > 0 \end{cases} \tag{11}$$

Let

$$A = \begin{pmatrix} b_1 & c_1 & & & & & & \\ a_2 & b_2 & c_2 & & & & & \\ & a_3 & b_3 & \ddots & & & & \\ & & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & & \\ & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & & & a_n & b_n \end{pmatrix} \tag{12}$$

$$U = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \vdots \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } V = \begin{pmatrix} 0 & v_1 \\ 0 & v_2 \\ h_1 & \vdots \\ h_2 & v_{n-2} \\ \vdots & 0 \\ h_{n-2} & 0 \end{pmatrix} \tag{13}$$

Then

$$C = A + UV^T \tag{14}$$

This means that the problem can be solved by once again using the Woodbury formula. In fact, as it was shown in [1],

$$\boldsymbol{x} = \boldsymbol{y} - (A^{-1}U(I + V^T A^{-1}U)^{-1}V^T)\boldsymbol{y} \tag{15}$$

provided that $\boldsymbol{y}$ is the solution of $A\boldsymbol{y} = \boldsymbol{f}$. Since A is a tridiagonal matrix that can be solved by the Thomas algorithm, our problem can surely be solved.

Now let

$$A^{-1} = \begin{pmatrix} \bar{a}_{11} & \bar{a}_{12} & \ldots & \bar{a}_{1n} \\ \bar{a}_{21} & \bar{a}_{22} & \ldots & \bar{a}_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ \bar{a}_{n1} & \bar{a}_{n2} & \ldots & \bar{a}_{nn} \end{pmatrix} \tag{16}$$

Then

$$A^{-1}U = \begin{pmatrix} \bar{a}_{11} & \bar{a}_{12} & \ldots & \bar{a}_{1n} \\ \bar{a}_{21} & \bar{a}_{22} & \ldots & \bar{a}_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ \bar{a}_{n1} & \bar{a}_{n2} & \ldots & \bar{a}_{nn} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \vdots \\ \vdots & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \bar{a}_{11} & \bar{a}_{1n} \\ \bar{a}_{21} & \bar{a}_{2n} \\ \vdots & \vdots \\ \bar{a}_{n1} & \bar{a}_{nn} \end{pmatrix} \tag{17}$$

$$V^T A^{-1}U = \begin{pmatrix} 0 & 0 & h_1 & h_2 & \ldots & h_{n-2} \\ v_1 & v_2 & \ldots & v_{n-2} & 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{a}_{11} & \bar{a}_{1n} \\ \bar{a}_{21} & \bar{a}_{2n} \\ \vdots & \vdots \\ \bar{a}_{n1} & \bar{a}_{nn} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n-2} h_i \bar{a}_{i+2,1} & \sum_{i=1}^{n-2} h_i \bar{a}_{i+2,n} \\ \sum_{i=1}^{n-2} v_i \bar{a}_{i,1} & \sum_{i=1}^{n-2} v_i \bar{a}_{i,n} \end{pmatrix} \tag{18}$$

and

$$I + V^T A^{-1}U = \begin{pmatrix} 1 + \sum_{i=1}^{n-2} h_i \bar{a}_{i+2,1} & \sum_{i=1}^{n-2} h_i \bar{a}_{i+2,n} \\ \sum_{i=1}^{n-2} v_i \bar{a}_{i,1} & 1 + \sum_{i=1}^{n-2} v_i \bar{a}_{i,n} \end{pmatrix} \tag{19}$$

Since (19) is a 2 × 2 matrix, it is easy to find its inverse $(I + V^T A^{-1}U)^{-1}$ provided that it is regular. Since (17) is an n × 2 matrix and V^T is a 2 × n matrix, the term $A^{-1}U(I + V^T A^{-1}U)^{-1}V^T$ is an n × n matrix, we can summarize the following algorithm to find a solution for the Eq. (8)

Algorithm for Solving Eq. (8)

Step 1. Calculate A^{-1} and $\boldsymbol{y} = A^{-1}\boldsymbol{f}$.

Step 2. Calculate $Z_1 = A^{-1}U$ by (17), obtaining Z_1 of order n × 2.

Step 3. Calculate $Z_2 = I + V^T Z_1$ by (19) and Z_2^{-1}, obtaining Z_2^{-1} of order 2 × 2.

Step 4. Calculate $Z_3 = Z_1 Z_2^{-1} V^T$, obtaining Z_3 of order n × n.

Step 5. Calculate $\boldsymbol{x} = (I - Z_3)\boldsymbol{y}$.

Remark 1. In the case of $h_2 = \ldots = h_{n-2} = 0$ and $v_1 = v_2 = \ldots = v_{n-3} = 0$, the Eq. (19) turns to be

$$I + V^T A^{-1} U = \begin{pmatrix} 1+\bar{a}_{31}h_1 & \bar{a}_{3n}h_1 \\ \bar{a}_{n-2,1}v_{n-2} & 1+\bar{a}_{n-2,1}v_{n-2} \end{pmatrix} \tag{20}$$

reducing the computations by $4(n-3)$ times.

4. Computational Efficiency

Seen from the steps of the computations proposed in the previous section, the operations of the computations are as follows:

(1) Computations of A^{-1} take $O(n^2+5n-5)$, according to [11].

(2) Computations of Z_1 take $O(2n)$ by (17).

(3) Computations of Z_2 and Z_2^{-1} take $O(4n+2)$ by (19).

(4) Computations of Z_3 takes $O(8n^2)$.

(5) Computations of final $\boldsymbol{x}$ takes $O(n^2+n)$

Consequently, the total cost is $O(10n^2+12n-3)$, which is the same cost as that in [6].

5. Numerical Examples

This section presents numerical examples for the algorithm proposed before. It first demonstrates the C++ language codes and then the examples. For convenience, matrix (2) is called a Top-Bottom-Bordered-Tridiagonal (TBBT) matrix, and thus the linear system from it is a TBBT system. Particularly, system (1) is called an almost tridiagonal one.

5.1. C/C++ Programming

The introduced algorithm is easy to be programmed with any computer advanced language or scripts such as C/C++, FORTRAN, Matlab and Maple. Here we propose C/C++ language because it is easily embedded into engineering systems of automations. To do, we designed a class CMatrix to deal with the matrix operations, especially the one to calculate the inverse of a tridiagonal matrix. Looking into literature, we found two algorithms that were introduced in [12] and [13] to calculate the inverse of a tridiagonal matrix. We compared the two algorithms and found that the one in [13] is easy to be converted into C++ language. We designed two member functions *invTriDiagMatrix* and *AlmostTridiagonalSolver* that particularly implement our algorithm and incorporated the solver into our project to develop CNC interpolator. Application shows the results are within expectation. Readers can see the codes in the appendix section.

5.2. Examples

As an example, we solve the almost tridiagonal system (1), which is frequently met with when constructing an interpolating B-spline curves of the natural boundary . In the equations, $\boldsymbol{Q}_i$ are points to be interpolated and $\boldsymbol{P}_j\,(j=0,1,\ldots,n+1)$ are control points of the B-spline curve. Taking $\boldsymbol{Q}_1(414.417,130.627)$, $\boldsymbol{Q}_2(394.420,151.128)$, $\boldsymbol{Q}_3(394.420,181.091)$, $\boldsymbol{Q}_4(417.048,203.169)$, $\boldsymbol{Q}_5(447.571,214.734)$, $\boldsymbol{Q}_6(478.093,203.169)$, $\boldsymbol{Q}_7(500.721,181.091)$, $\boldsymbol{Q}_8(500.721,151.128)$ and $\boldsymbol{Q}_9(480.724,130.627)$ yields two linear equations

$$\frac{1}{6}\begin{pmatrix} 6 & -12 & 6 & & & & & & & \\ 1 & 4 & 1 & & & & & & & \\ & 1 & 4 & 1 & & & & & & \\ & & 1 & 4 & 1 & & & & & \\ & & & 1 & 4 & 1 & & & & \\ & & & & 1 & 4 & 1 & & & \\ & & & & & 1 & 4 & 1 & & \\ & & & & & & 1 & 4 & 1 & \\ & & & & & & & 1 & 4 & 1 \\ & & & & & & & 6 & -12 & 6 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 0 \\ 414.417 \\ 394.420 \\ 394.420 \\ 417.048 \\ 447.571 \\ 478.093 \\ 500.721 \\ 500.721 \\ 480.724 \\ 0 \end{pmatrix}$$

and

$$\frac{1}{6}\begin{pmatrix} 6 & -12 & 6 & & & & & & & \\ 1 & 4 & 1 & & & & & & & \\ & 1 & 4 & 1 & & & & & & \\ & & 1 & 4 & 1 & & & & & \\ & & & 1 & 4 & 1 & & & & \\ & & & & 1 & 4 & 1 & & & \\ & & & & & 1 & 4 & 1 & & \\ & & & & & & 1 & 4 & 1 & \\ & & & & & & & 1 & 4 & 1 \\ & & & & & & & 6 & -12 & 6 \end{pmatrix}\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{pmatrix} = \begin{pmatrix} 0 \\ 130.627 \\ 151.128 \\ 181.091 \\ 203.169 \\ 214.734 \\ 203.169 \\ 181.091 \\ 151.128 \\ 130.627 \\ 0 \end{pmatrix}$$

These two equations are the form of Eq. (1). With the solver we developed, the solutions for the two are obtained by

$$X = (438.293,414.417,390.541,389.940,416.217,447.478,478.973,505.187,504.604,480.724,456.844)^T$$

and

$$Y = (113.134,130.627,148.120,183.660,203.787,220.208,203.787,183.660,148.120,130.627,113.134)^T$$

which lead to $\boldsymbol{P}_0 = (438.293,113.134)$, $\boldsymbol{P}_1 = (414.417,130.627)$, $\boldsymbol{P}_2 = (390.541,148.120)$, $\boldsymbol{P}_3 = (389.940,183.660)$, $\boldsymbol{P}_4 = (416.217,203.787)$, $\boldsymbol{P}_5 = (447.478,220.208)$, $\boldsymbol{P}_6 = (478.973,203.787)$, $\boldsymbol{P}_7 = (505.187,183.660)$ $\boldsymbol{P}_8 = (504.604,148.120)$, $\boldsymbol{P}_9 = (480.724,130.627)$ and $\boldsymbol{P}_{10} = (456.844,113.134)$.

The interpolated points $\boldsymbol{Q}_i (i = 1,2,...,9)$, the calculated control points $\boldsymbol{P}_j (j = 0,1,...,10)$ and the constructed B-spline curve are drawn with Fig. 1.

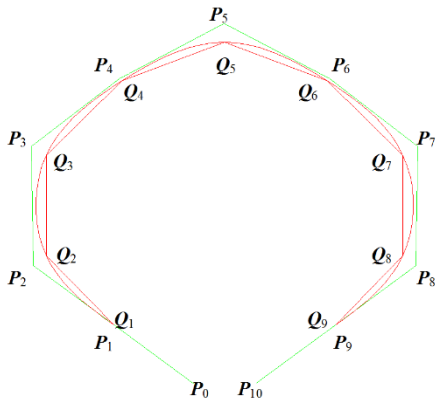

Fig. 1. Interpolated points, interpolating B-spline curve and calculated control points.

6. Conclusion and Future Work

As a frequently used linear system, the almost tridiagonal one was less investigated. This paper takes it to be a special case of the TBBT system and finds a way to solve the equations with computational efficiency. Compared to the Thomas algorithm or the generic Thomas algorithm, the introduced algorithm is less efficient although it is better than the LU algorithm. This leaves the future work of finding more efficient algorithm. Maybe new algorithm like the Thomas algorithm or those in [4, 5] can be found. Hope to see them in the future.

Appendix: C Language Codes

The following C language codes are implementation of the algorithm introduced in this paper. They are now a part of the class CMatirx we designed. Readers of interests can ask Prof. Xingbo WANG for the completed package. It should be pointed out that the function invTriDiagMatrix are converted from the Maple codes presented in [13].

```
////// Member function invTriDiagMatrix.
////// This is to find the inverse of a tridiagonal matrix.
////// Argument N: order the tridiagonal matrix.
////// Arguments a, d, and b: arrays to record the elements of the tridiagonal matrix as seen in [13].
///// Argument res: inverse of the calculated tridiagonal matrix.
void CMatrix::invTriDiagMatrix(int N, double *a, double *d, double *b, CMatrix &res)
{ int i, j;
   double *alpha=new double [N];
   double *beta=new double [N];
   CMatrix A(N);
   alpha[0]=d[0];
   beta[N-1]=d[N-1];
    for(i=1;i<N;i++)
       alpha[i]=(d[i]-b[i]*a[i-1]/alpha[i-1]);
    for(i=N-2;i>=0;i--)
       beta[i]=(d[i]-b[i+1]*a[i]/beta[i+1]);
    for(i=0;i<N;i++)
      A.SetElement(i,i,1/(alpha[i]-d[i]+beta[i]));

    ////For upper tridiagonal matrix (UTM)
       for (i=0;i<N-1;i++){
            for(j=i+1;j<N;j++){
               if(j>i) A.SetElement(j,i,-b[j]/beta[j]*A.GetElement(j-1,i));
             }
        } //// end of treating UTM
    ////For lower tridiagonal matrix (LTM)
     for(i=1;i<N;i++){
            for(j=i-1;j>=0;j--){
                  if(j<i)A.SetElement(j,i,-(a[j]/alpha[j]*A.GetElement(j+1,i)));
            }
     } //// end of treating LTM
     res=A;
    }

////// Member function AlmostTridiagonalSolver.
////// This is to solve the equation (8).
////// Argument N: order the tridiagonal matrix.
```

```
////// Argument f: n×1 matrix on the right side of (8).
////// Arguments a, d, and b: the same meaning as thoes in invTriDiagMatrix.
////// Argument h: array of h1,h2,...,hn-2 in matrix (10).
////// Argument v: array of v1,v2,...,vn-2 in matrix (10).
////// Argument X: n×1 matrix ,calculated unknowns of (8).

void CMatrix::AlmostTridiagonal(int N , CMatrix f, double *a, double *d, double *b,double *h,double *v,CMatrix
&X)
{
  CMatrix invA(N+2,N+2);      //// to store the inverse of A
  CMatrix Y(N+2,3);           //// to store y
  CMatrix U(N+2,2);           //// to store U
  CMatrix Z1(N+2,2);          //// to Z1
  CMatrix VT(2,N+2);         //// transpose of V
  CMatrix I(2,2);             ////I,    2×2 unit matrix
  CMatrix Z2(2,2);           ////Z2, 2×2 matrix
  CMatrix invZ2(2,2);          ////Z2, inverse of Z2
  CMatrix Z3(N+2,N+2);        ////Z3
  CMatrix II(N+2,N+2);        ////II, (N+2) × (N+2) unit matrix
// CMatrix MMM(N+2,3);      /////MMM, N+2 outputs

//// Step 1 of my algorithm
  CMatrix::invTriDiagMatrix(N+2, a, d, b,res); //// call invTriDiagMatrix
  Y=invA*f;                          //// calculate Y

////////////// temporary storages
double *m=new double [N+2];
double *xx=new double[N+2];
double *th=new double[N+2];
//////////////

m[0]=1;
for(int i=1;i<N+2;i++) m[i]=0;
for(int i=0;i<N+1;i++) xx[i]=0;
xx[N+1]=1;

int kk=0;
for(int j=0;j<N+2;j++) U.SetElement(j,kk,m[j]);    ////Set U

int cc=1;
for(int j=0;j<N+2;j++) U.SetElement(j,cc,xx[j]);

Z1=invA*U;        //// calculate Z1

int hj=0;
for(int j=0;j<N+2;j++) VT.SetElement(hj,j,h[j]);
int hm=1;
for(int j=0;j<N+2;j++) VT.SetElement(hm,j,v[j]);
double re[2], ke[2];
re[0]=1; re[1]=0; ke[0]=0; ke[1]=1;

int dd=0;
for(int j=0;j<2;j++) I.SetElement(dd,j,re[j]);
```

```
int fff=1;
for(int j=0;j<2;j++)I.SetElement(fff,j,ke[j]);

Z2=I+VT*Z1;              ////calculate Z2

////For invZ2
CMatrix B(2,2);
B.SetElement(0,0,Z2.GetElement(1,1));
B.SetElement(1,0,-Z2.GetElement(1,0));
B.SetElement(0,1,-Z2.GetElement(0,1));
B.SetElement(1,1,Z2.GetElement(0,0));

for(int i=0;i<2;i++)
for(int j=0;j<2;j++){
invZ2.SetElement(i,j,B.GetElement(i,j)/(Z2.GetElement(0,0)*Z2.GetElement(1,1)-Z2.GetElement(1,0)*Z2.GetElem
ent(0,1))+0);}

//// for Z3
Z3=Z1*invZ2*VT;

for(int i=0;i<N+2;i++) th[i]=1;
for(int i=0;i<N+2;i++)II.SetElement(i,i,th[i]);
//// for X
X=(II-Z3)*Y;
}
```

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

X. Wang contributes the theoretical reasoning for the algorithm; J. Wang contributes the programming work and numerical experiments. Both authors had approved the final version.

Funding

The research work is supported by Guangdong Engineering Center of Information Security for Intelligent Manufacturing System.

References

[1] Wang, X. B. (2009). A new algorithm with its scilab implementation for solution of bordered tridiagonal linear equations. *Proceedings 2009 IEEE International Workshop on Open Source Software for Scientific Computation* (pp. 11–14).

[2] Martin, A., & Boyd, I. D. (2010). Variant of the Thomas Algorithm for opposite-bordered tridiagonal systems of equations. *International Journal for Numerical Methods in Biomedical Engineering*, *26(6)*, 752–759.

[3] Karawia, A. A. (2013). New symbolic algorithms for solving a general bordered tridiagonal linear system. arXiv preprint, arXiv:1303.0738.

[4] El-Mikkawy, M., & Atlan, F. (2014). Algorithms for solving doubly bordered tridiagonal linear systems *British Journal of Mathematics & Computer Science*, *4(9)*, 1246.

[5] Atlan, F., & El-Mikkawy, M. E. A. (2015). A new symbolic algorithm for solving general

opposite-bordered tridiagonal linear systems. *Am. J. Comput. Math*, *5*, 258–266.

[6] Jia, J., & Li, S. (2015). On the inverse and determinant of general bordered tridiagonal matrices. *Computers & Mathematics with Applications*, *69(6)*, 503–509.

[7] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C, the Art of Scientific Computing*. Cambridge University Press.

[8] Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations*. The Johns Hopkins University Press.

[9] Strunk, W., & White, E. B. (1979). *The Elements of Style* (3rd ed.). New York: Macmillan.

[10] Petersen, K. B., & Pedersen, M. S. (2008). The matrix cookbook. *Technical University of Denmark*, *7(15)*, 510.

[11] El-Mikkawy, M. E. (2004). On the inverse of a general tridiagonal matrix. *Applied Mathematics and Computation*, *150(3)*, 669–679.

[12] Ran, R. S., Huang, T. Z., Liu, X. P., & Gu, T. X. (2009). An inversion algorithm for general tridiagonal matrix. *Applied Mathematics and Mechanics*, *30(2)*, 247–253.

[13] El-Mikkawy M E A, Karawia A. (2022, August). A Breakdown free numerical algorithm for inverting general tridiagonal matrices. arXiv preprint arXiv: arxiv-2208.12843.