STM JOURNALS

Recent Trends in
Programming Languages

ISSN: 2455-1821
Volume 11, Issue 3, 2024
September–December
DOI (Journal): 10.37591/RTPL

https://journals.stmjournals.com/rtpl

Review RTPL

Implementation of the Tridiagonal Matrix Algorithm (TDMA) in C: A Practical Approach

Navya Jain[1], Rishika Chauhan[2], Pankaj Dumka[3,*]

Abstract

This paper presents a practical implementation of the tridiagonal matrix algorithm (TDMA), also known as the Thomas algorithm, using the C programming language. The TDMA is a commonly used algorithm for solving systems of linear equations where the coefficient matrix is tridiagonal. The paper draws a detailed step-by-step process of the algorithm's development, from forward elimination to backward substitution, with a focus on minimizing computational difficulty compared to standard Gaussian elimination. The simplicity and efficiency of TDMA make it an ideal choice in many engineering applications, such as structural analysis, heat conduction, and fluid dynamics, where large tridiagonal systems often arise. The C programming language is highlighted as a standard platform for implementing the TDMA due to its performance benefits, low-level memory management, and precision in handling numerical problems. The paper provides a comprehensive explanation of the algorithm's core components, including vector initialization, solution printing, and the main TDMA solver function. Several practical examples were used to demonstrate the effectiveness of the implementation, validating the approach with commonly met tridiagonal systems in engineering scenarios. This study contributes to numerical analysis and engineering computation by offering a robust and efficient solution to tridiagonal systems. The implementation's modularity and clear function structure also make it adaptable for educational and professional purposes, allowing for easy integration into more complex numerical simulations.

Keywords: TDMA, Thomas algorithm, C programming, tridiagonal matrix, numerical analysis

INTRODUCTION

Tridiagonal systems of equations often appear in numerical analysis and engineering applications, particularly when solving problems related to finite difference methods for partial differential equations [1]. A tridiagonal matrix has nonzero elements only on its main diagonal, a diagonal above, and a diagonal below it. The tridiagonal matrix algorithm (TDMA), also known as the Thomas algorithm, is a shortened form of Gaussian elimination that solves this system of equations [2]. However, the problem with Gaussian elimination is that it has a high computational cost when solving a tridiagonal matrix [3]. This paper details the implementation of TDMA in C, explains the approach, and demonstrates the algorithm using numerical examples.

***Author for Correspondence**
Pankaj Dumka
E-mail: p.dumka.ipec@gmail.com

[1]Student, Department of Computer Science and Engineering, Jaypee University of Engineering and Technology, Raghogarh-Vijaypur, Guan, Madhya Pradesh, India.
[2]Assistant Professor, Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, Raghogarh-Vijaypur, Guan, Madhya Pradesh, India.
[3]Assistant Professor, Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Raghogarh-Vijaypur, Guan, Madhya Pradesh, India

Received Date: October 09, 2024
Accepted Date: October 14, 2024
Published Date: November 04, 2024

Citation: Navya Jain, Rishika Chauhan, Pankaj Dumka. Implementation of the Tridiagonal Matrix Algorithm (TDMA) in C: A Practical Approach. Recent Trends in Programming Languages. 2024; 11(3): 36–43p.

C programming is essential for engineers when solving numerical problems, such as the tridiagonal matrix, because it offers high performance, fine control over memory, and effective execution [4, 5]. Engineers often deal with large datasets and complex mathematical calculations, where computational speed and precision are important. The ability of C to work closely with hardware [6], manage resources

efficiently, and optimize algorithms makes it ideal for solving time-sensitive, resource-constrained problems in areas such as structural analysis, fluid mechanics, and thermal modeling [7, 8]. Additionally, learning C helps engineers develop a strong foundation for programming concepts, enabling them to better understand and implement numerical methods in real-world applications [9].

TRIDIAGONAL MATRIX ALGORITHM

The TDMA algorithm reduces the computational complexity compared to the ordinary Gaussian elimination method. Moreover, one does not need to store elements in a matrix manner. Elements can be stored in a linear fashion (such as a vector), that is, one index can solve the purpose. A typical tridiagonal matrix can be represented as [1]

$$\begin{bmatrix} d_1 & a_1 & & & & & \\ b_2 & d & a_2 & & & & \\ & b & d_3 & a_3 & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & b_{n-1} & d_{n-1} & a_{n-1} \\ & & & & & b & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} \tag{1}$$

In the equation form for some i[th] node (as shown in Figure 1), matrix can be written as

$$b_i x_{i-1} + d_i x_i + a_i x_{i+1} = c_i \tag{2}$$

One must remember that b_1 and a_n both are zero. For better understanding, one can write Equation (2) as:

$$d_i x_i = c_i + (-b_i) x_{i-1} + (-a_i) x_{i+1} \tag{3}$$

For different i, one can understand how different x depends on other x as shown in Table 1.

So based on Table 1, one can write the i^{th} unknown in terms of $(i + 1)^{th}$e unknown in the form of the below-mentioned recurrence formula:

$$x_1 = \alpha_i x_{i+1} + \beta_i \tag{4}$$

Where α_i and β_i are the constants that are not known so far. Now the next step can be to write Equation (4) for (i-1)[th] term as follows:

$$x_{i-1} = \alpha_{i-1} x_i + \beta_{i-1} \tag{5}$$

Substituting the value of x_{i-1} from Equation (5) into Equation (2) and solve for α_i and β_i.

$$b_i(\alpha_{i-1} x_i + \beta_{i-1}) + d_i x_i + a_i x_{i+1} = c_i \tag{6}$$

$$(b_i \alpha_{i-1} + d_i) x_i = -a_i x_{i+1} + c_i - b_i \beta_{i-1} \tag{7}$$

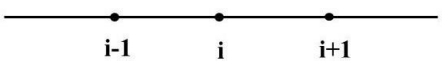

Figure 1. Node locations.

Table 1. Functional dependencies of ϕ.

i	x	Depends on
1	$x_1 = \frac{c_1+0+(-a_1)x_2}{d_1}$	$f(x_2)$
2	$x_2 = \frac{c_2+(-b_2)x_1+(-a_2)x_3}{d_2}$	$f(x_3)$
n	$But\ x_1 = f(x_2)$ $x_n = c_n + (-b_n)x_{n-1} + 0$ $But\ x_{n-1} = f(x_n)$	Constant

$$x_i = \frac{-a_i}{(b_i\alpha_{i-1}+d_i)}x_{i+1} + \frac{c_i - b_i\beta_{i-1}}{(b_i\alpha_{i-1}+d_i)} \quad (8)$$

Now comparing Equation (8) with Equation (4) returns the values of P_i and Q_i, as follows:

$$\alpha_i = \frac{-a_i}{(b_i\alpha_{i-1}+d_i)} \quad \beta_i = \frac{c_i - b_i\beta_{i-1}}{(b_i\alpha_{i-1}+d_i)} \quad (9)$$

Equation (9) tells that the value of α at i^{th} level can be known once its value at $(i-1)^{th}$ level is known. But it has already been mentioned that b_1 is zero, so α_1 and β_i can be evaluated as:

$$\alpha_1 = -\frac{a_1}{d_1} \quad \beta_1 = \frac{c_1}{d_1} \quad (10)$$

Now, as at location 1, the values of α and β are known, so using Equation (9) one can evaluate the values for other locations using Equation (9). In the end, one will get $\alpha_n = 0$ and $x_n = \beta_n$. Based on x_n the back substitution (Equation (4)) will result in x_{n-1} and other terms up to the first term, i.e., x_1.

METHODOLOGY

The core of our implementation is the TDMA function in C, which accepts five vectors representing the tridiagonal matrix and right-hand-side vector. The algorithm proceeds through forward elimination and backward substitution to solve the system. The matrix system solved using the TDMA algorithm is as follows:

$$d_1x_1 + a_1x_2 = c_1$$

$$b_ix_{i-1} + d_ix_i + a_ix_{i+1} = c_i \; for \; 2 \le i \le n-1$$

$$b_nx_{n-1} + d_nx_n = c_n$$

Where, b is the sub-diagonal (below diagonal) vector, d is the diagonal vector, a is the super-diagonal (above diagonal) vector, c is the right-hand-side vector, and x is the unknown vector to be solved. Mainly two steps are implemented in this algorithm viz. (i) forward elimination to eliminate the sub-diagonal terms and (ii) backward substitution to solve for the unknowns.

The program is divided into several functions, each of which performs a particular task, thereby making the code more modular and easier to understand. The code consists of the following components.

1. *number_array:* Initialize the elements of vectors (arrays) used in the TDMA process.
2. *arr_print:* Prints the final solution vector of unknowns.
3. *TDMA:* The main algorithm to solve the system of tridiagonal equations.
4. *main:* The driver function takes user input, sets up the vectors, calls the TDMA solver, and outputs the results.

The stepwise explanation of the code is as follows:

1. *Function to create vectors (number_array):* This function initializes each element of the array (vector) with a specified value. It simplifies the task of setting up arrays, such as diagonal and off-diagonal vectors, by assigning the same value to each element.

```
void number_array(int n, float value, float *x)
        {
              for(int i=0; i<n; i++)
            {
                    x[i] = value;
              }
        }
```

   i. *Parameters:*
      - *n:* The size of an unknown array (vector).
      - *value:* The number whose array has to be created.
      - *x[]:* The array to be filled with the value.

ii. *Example:* If one wants to initialize a diagonal vector with a value of 5, which has four elements, then this function will set all the elements of the vector to 5.
iii. *Working:* The function loops through each element of the array x[] and assigns the value value to it. The loop runs from 0 to n-1 to cover all elements of the array.

2. *Function to print a vector (arr_print):* This function is created to display the contents of the array, making it easier to see the final solution once the TDMA process is complete.

```
void arr_print(int n, float x[])
{
                for (int i=0; i<n; i++)
               {
                        printf("x[%d] = %f\n", i, x[i]);
                }
}
```

i. *Parameters:*
   - n: Array size.
   - x[]: Array (solution vector) whose elements are to be printed.
ii. *Example output:* For an array of size three with values [1.5, 2.3, 3.7], the output is as follows:
   - x[0] = 1.500000
   - x[1] = 2.300000
   - x[2] = 3.700000
iii. *Working:* The function loops through the array and prints each element in a formatted manner, along with its index.

3. *TDMA solver (TDMA):* This is the primary part of the program that implements the actual algorithm to solve a system of tridiagonal equations. It consists of two main steps: (i) forward elimination and (ii) backward substitution.

```
void TDMA(int n, float b[], float d[], float a[], float c[], float
*x) {
                float alpha[n], beta[n];   // Intermediate variables

                // Step 1: Setup for the first elements of 'a' and 'b' toavoid out-of-bound access.
                a[n-1] = 0;
                b[0] = 0;

                // Step 2: Calculate the first alpha and beta valuesalpha[0] = -a[0] / d[0];
                beta[0] = c[0] / d[0];
                // Step 3: Forward elimination - calculate the remainingalpha and beta values
                for (int i=1; i<n; i++) {
                        alpha[i] = -a[i] / (b[i] * alpha[i-1] + d[i]);
                        beta[i] = (c[i] - b[i] * beta[i-1]) / (b[i] *alpha[i-1] + d[i]);
                }

                // Step 4: Backward substitution to compute the solutionvector 'x'
                x[n-1] = beta[n-1]; // Last element of the solution for (int i=n-2; i>-1; i--) {
                        x[i] = alpha[i] * x[i+1] + beta[i]; // Substituting backwards
                }
        }
```

i. *Parameters:*
   - n: The number of unknowns.
   - b[]: Sub-diagonal vector (i.e., vector below the main diagonal).
   - d[]: The diagonal vector (main diagonal vector).
   - a[]: Super-diagonal vector (i.e., the vector above the main diagonal).
   - c[]: The right-hand side vector.
   - x[]: The array to store the solution vector.

ii. *Explanation:*
- *Step 1: initialize:* Because the first element of b[] and the last element of a[] are not used in the matrix setup, we initialize them to zero. This avoids access to invalid memory locations later in the code.
- *Step 2: forward elimination:* The TDMA algorithm eliminates elements from the lower diagonal by computing alpha[] and beta[], which are intermediate variables that store the modified coefficients. This helps simplify the matrix to an upper triangular form. Alpha [] and beta[] were evaluated using Equation (9):
- *Step 3: backward substitution:* Once the matrix is simplified to an upper triangular form, the backward substitution process begins. The last element of the solution x[] is set to the last beta[] value, which is then used to calculate the rest of the solution using Equation (9). This formula substitutes for the solution from the last element upward.

4. *Main function–main:* The main function drives the entire program, collects user input, sets up the problem, and calls the TDMA solver. The final solution was printed.

```
int main()
{
        // Step 1: Get the number of unknownsint n;
        printf("Enter number of unknowns: ");scanf("%d", &n);

        // Step 2: Define vectors for the tridiagonal systemfloat b[n], d[n], a[n], c[n], x[n];

        // Step 3: Get user input for matrix elementsfloat b1, d1, a1, c1;
        printf("Enter elemental values of b, d, a, and c\n");scanf("%f %f %f %f", &b1,
        &d1, &a1, &c1);

        // Step 4: Initialize the vectors using the number_array function
        number_array(n,     b1,     b);
        number_array(n,     d1,     d);
        number_array(n,     a1,     a);
        number_array(n, c1, c);

        // Step 5: Call the TDMA function to solve the systemTDMA(n, b, d, a, c, x);

        // Step 6: Print the solution using the arr_print functionarr_print(n, x);

        return 0;
}
```

i. *Explanation*
- First, the program asks the user for the number of unknowns.
- It then prompts for the values to fill the diagonal (d), sub-diagonal (b), super-diagonal (a), and right-hand-side (c).
- The vectors are initialized using the number_array function.
- Finally, the TDMA solver is called, and the solution is printed.

IMPLEMENTATION OF TDMA FUNCTION IN DIFFERENT PROBLEMS

To validate the implementation, we tested the algorithm using tridiagonal matrix examples commonly found in engineering problems.

Example 1. Solve the following Tridiagonal matrix using the Thomas algorithm:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

Solution: Because there were five unknowns, n=5. Then, we set a, b, c, and d vectors and call the function TDMA.

Input parameters	Output
Below diagonal vector element = -1 Diagonal vector element = 2 Above diagonal vector element = -1 Right-hand vector element =1	Enter the number of unknowns: 5 Enter elemental values of b, d, a, and c -1 2 -1 1 x[0] = 2.500000 x[1] = 4.000000 x[2] = 4.500000 x[3] = 4.000000 x[4] = 2.500000

Example 2. Solve the following Tridiagonal matrix using the Thomas algorithm:

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 2 & 4 & 1 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 0 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 10 \end{bmatrix}$$

Solution: Because there were five unknowns, n=4. Then, we set a, b, c, and d vectors and call the function TDMA.

Input parameters	Output
Below diagonal vector element = 2 Diagonal vector element = 4 Above diagonal vector element = 1 Right-hand vector element =10	Enter the number of unknowns: 4 Enter elemental values of b, d, a, and c 2 4 1 10 x[0] = 2.256098 x[1] = 0.975610 x[2] = 1.585366 x[3] = 1.707317

Example 3: Find the steady-state temperature distribution in a rod 1 m in length. The left and right temperatures were 700 K and 350 K, respectively. It is assumed that the rod is 1D with no heat generation.

Solution:

The 1D steady heat conduction equation with no heat generation is given by:

$$\frac{\partial^2 T}{\partial x^2} = 0$$

Discretizing it for any i^{th} node with second-order accuracy will give:

$$T_{i-1} - 2T_i + T_{i+1} = 0$$

The boundary condition:

$T_1 = 700.0$ $T_2 = 350.0$

For five nodes (the number can be any), the solution becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 700 \\ 0 \\ 0 \\ 0 \\ 350 \end{bmatrix}$$

In this problem, we have to slightly modify the main function, where we are generating the values of vectors b, d, a, and c. In the diagonal below, the last element is zero, the above diagonal has the first element zero, and the diagonal vector has the first and last elements as 1. In addition, in the Right-Hand Side (RHS) vector, the first and last elements take the values of the boundary conditions, so they can be updated in the program itself.

main function	**Output**

int main()
 	
{
int n;
printf(“Enter number of unknowns: ”); scanf(“%d”,&n);
 	
float b[n],d[n],a[n],c[n],x[n];
 	
// supplying elements to b, d, a, and c vectors
 	
float b1,d1,a1,c1;
 	
printf(“Enter elemental values of b, d, a, and c\n”);
scanf(“%f %f %f %f”,&b1, &d1, &a1, &c1);
number_array(n,b1,b);
b[n-1] = 0; // changed last term
 	
number_array(n,d1,d);
d[0] = 1; // changed first term d[n-1] = 1; // changed last term
 	
number_array(n,a1,a);
a[0] = 0; // changed first term
 	
number_array(n,c1,c);
c[0] = 700; // changed first term c[n-1] = 350; // changed last term
 	
TDMA(n,b,d,a,c,x); arr_print(n,x); return 0;
}
	Enter the number of unknowns: 5
	Enter elemental values of b, d, a, and c
	1
	-2
	1
	0
	x[0] = 700.000000
	x[1] = 612.500000
	x[2] = 525.000000
	x[3] = 437.500031
	x[4] = 350.000000

CONCLUSIONS

In this study, the implementation of the Thomas algorithm (TDMA) using C programming is presented. The algorithm was specifically designed to efficiently solve the tridiagonal matrices. A comprehensive explanation of the algorithm, accompanied by the C code for the TDMA function, was presented. To demonstrate its practical application, three example problems were solved, demonstrating how the function can be applied to real-life scenarios. This approach offers a clear and accessible pathway for beginners in numerical methods to understand how to set up and solve equations programmatically using C.

REFERENCES

1. Strang G. Introduction to Linear Algebra, 5th edition. Wellesley, Massachusetts: Wellesley-Cambridge Press; Cambridge Press; 2022.
2. Salih A. Tridiagonal matrix algorithm. Department of Aerospace Engineering, Indian Institute of Space Science and Technology, Thiruvananthapuram. October 2010.
3. Dumka P, Dumka R, Mishra DR. Numerical methods using Python (for scientists and engineers). Noida, India: Blue Rose Publishers; 2022.
4. Strikwerda JC. Finite Difference Schemes and Partial Differential Equations. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics; 2004. DOI: 10.1137/1.9780898717938.fm.
5. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C: The Art of Scientific Computing. 2nd ed. New York: Cambridge University Press; 1992.
6. Samant SS, Xia J, Muyan-Özçelik P, Owens JD. High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy. Med Phys. 2008;35:3546–53. DOI: 10.1118/1.2948318, PubMed: 18777915.
7. Dumka P, Dumka R. Basics of C for Engineers: A Quick Introduction. New Delhi: Apna Publisher; 2023.
8. Barnett RH, Cox S, O'Cull L. Embedded C Programming and the Atmel AVR. Boston, Massachusetts, United States: Delmar Cengage Learning; 2002.
9. Epperson JF. An Introduction to Numerical Methods and Analysis. New Jersey, United States: Wiley–Blackwell; 2007.

