J. Korean Soc. Ind. Appl. Math. Vol.29, No.2, 171–183, 2025 https://doi.org/10.12941/jksiam.2025.29.171

PRACTICAL IMPLEMENTATION OF BOUNDARY CONDITIONS IN THE THOMAS ALGORITHM

SOOBIN KWAK[1†]

[1]DEPARTMENT OF MATHEMATICS, KOREA UNIVERSITY, SOUTH KOREA
Email address: [†]soobin23@korea.ac.kr

ABSTRACT. The Thomas algorithm is widely used for efficiently solving tridiagonal systems that arise from finite difference discretizations of partial differential equations. While the core algorithm is well established, the practical treatment of boundary conditions, especially periodic boundary, often requires careful implementation to maintain numerical stability and accuracy. This paper presents a comprehensive discussion of practical strategies for incorporating various boundary conditions into the Thomas algorithm. We compare different implementation approaches, analyze their computational implications, and provide code-level insights to facilitate integration into numerical solvers. Benchmark tests on prototypical diffusion-type equations demonstrate the effectiveness and robustness of the proposed methods.

1. INTRODUCTION

Tridiagonal linear systems frequently arise in the numerical solution of partial differential equations (PDEs), especially when using finite difference discretizations of diffusion-type equations. The Thomas algorithm, also known as the tridiagonal matrix algorithm, is a classical direct solver for such systems, offering both computational efficiency and numerical stability. Owing to its simplicity and low computational cost, the Thomas algorithm remains a fundamental tool in the implementation of time-stepping schemes such as the implicit Euler [1] or alternating direction implicit (ADI) method [2].

Despite the algorithm's straightforward structure, its application requires careful handling of boundary conditions, particularly when the boundaries do not follow simple Dirichlet or Neumann specifications. Among these, periodic boundary conditions introduce cyclic coupling across the domain boundaries, violating the strict tridiagonal structure required by the standard Thomas algorithm. To address this issue, modifications such as the Sherman–Morrison formula are often employed, allowing for the efficient solution of cyclic tridiagonal systems via rank-one corrections.

Received June 15 2025; Revised June 19 2025; Accepted in revised form June 23 2025; Published online June 25 2025.

2020 *Mathematics Subject Classification.* 65M06.

Key words and phrases. Thomas algorithms, numerical boundary treatment, Sherman–Morrison correction.

† Corresponding author.

This paper focuses on the practical implementation and comparison of boundary condition treatments within the Thomas algorithm framework. We provide a detailed formulation of the algorithm under periodic [3], Dirichlet [4, 5], and Neumann [6, 7] boundary conditions, highlighting their respective impacts on matrix structure and solvability. Through illustrative examples, we demonstrate how to adapt the algorithm using matrix decomposition techniques and explain the implementation strategy in the context of two-dimensional problems solved with the operator splitting method (OSM).

To validate and compare the accuracy and stability of these implementations, we conduct a series of numerical experiments. These include convergence tests using the heat equation with an exact solution, as well as phase separation dynamics modeled by the Allen–Cahn (AC) equation [8, 9]. The experiments showcase how different boundary conditions influence numerical results, particularly in the evolution of sharp interfaces.

By bridging theoretical derivation and code-level practicality, this study aims to serve both as a reference for implementing boundary-aware Thomas solvers and as a guide for understanding the influence of boundary conditions in time-dependent PDE simulations.

The remainder of this paper is organized as follows. Section 2 introduces the numerical method and boundary condition treatments. Section 3 presents numerical experiments to validate and compare the proposed approach. A conclusion is given in Sec. 4, and the implementation code is provided in the Appendix.

2. Numerical Method

We consider the heat equation in a d-dimensional domain $\Omega \subset \mathbb{R}^d$:

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = \alpha \Delta u(\mathbf{x},t), \qquad \mathbf{x} \in \Omega, \quad t > 0. \tag{2.1}$$

Here, $u(\mathbf{x},t)$ denotes the temperature field, and $\alpha > 0$ is the diffusion coefficient. In this work, we focus on the two-dimensional case ($d = 2$), where the domain is given by $\Omega = (L_x, R_x) \times (L_y, R_y)$. The governing equation becomes

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \qquad (x,y) \in \Omega, \quad t > 0. \tag{2.2}$$

We assume periodic boundary conditions in both spatial directions. To solve the governing equation (2.2) numerically, the spatial domain is discretized using a uniform cell-centered grid. Let N_x and N_y denote the number of grid cells in the x- and y-directions, respectively. The spatial step size is given by $h = (R_x - L_x)/N_x = (R_y - L_y)/N_y$, and the cell centers are located at $x_i = L_x + (i - 0.5)h$, $i = 1, \ldots, N_x$ and $y_j = L_y + (j - 0.5)h$, $j = 1, \ldots, N_y$. We denote the numerical approximation of $u(x_i, y_j, t^n)$ by u_{ij}^n, where $t^n = n\Delta t$.

For each time step, the numerical solution is advanced from u_{ij}^n to u_{ij}^{n+1} through two intermediate substeps. The governing equation (2.2) is split into two parts using the OSM.

In the first substep, we solve the following equation implicitly in the x-direction:

$$\frac{u_{ij}^* - u_{ij}^n}{\Delta t} = \alpha \frac{u_{i-1,j}^* - 2u_{ij}^* + u_{i+1,j}^*}{h^2}. \tag{2.3}$$

In the second substep, the equation is solved implicitly in the y-direction using the intermediate solution u_{ij}^*:

$$\frac{u_{ij}^{n+1} - u_{ij}^*}{\Delta t} = \alpha \frac{u_{i,j-1}^{n+1} - 2u_{ij}^{n+1} + u_{i,j+1}^{n+1}}{h^2}. \tag{2.4}$$

.

Equation (2.3) can be rewritten in matrix form as

$$A\mathbf{u}_j^* = \mathbf{u}_j^n, \tag{2.5}$$

where

$$A = \begin{pmatrix} 1+2r & -r & 0 & \cdots & 0 & -r \\ -r & 1+2r & -r & 0 & \cdots & 0 \\ 0 & -r & 1+2r & -r & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -r & 1+2r & -r \\ -r & 0 & \cdots & 0 & -r & 1+2r \end{pmatrix}, \ \mathbf{u}_j^* = \begin{pmatrix} u_{1j}^* \\ u_{2j}^* \\ u_{3j}^* \\ \vdots \\ u_{N_x-1,j}^* \\ u_{N_xj}^* \end{pmatrix}, \text{ and } \mathbf{u}_j^n = \begin{pmatrix} u_{1j}^n \\ u_{2j}^n \\ u_{3j}^n \\ \vdots \\ u_{N_x-1,j}^n \\ u_{N_xj}^n \end{pmatrix}.$$

Here, $r = \alpha \Delta t / h^2$. The matrix A is cyclic tridiagonal due to the periodic boundary conditions, with nonzero elements in the top-right and bottom-left corners. Therefore, it cannot be directly handled by the standard Thomas algorithm. For the efficient solution of such systems, the cyclic tridiagonal matrix is decomposed into the sum of a modified tridiagonal matrix and a rank-one correction. Equation (2.5) rewrites $(\tilde{A} + pq^T)\mathbf{u}_j^* = \mathbf{u}_j^n$, where

$$\tilde{A} = \begin{pmatrix} 1+3r & -r & 0 & \cdots & 0 & 0 \\ -r & 1+2r & -r & 0 & \cdots & 0 \\ 0 & -r & 1+2r & -r & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -r & 1+2r & -r \\ 0 & 0 & \cdots & 0 & -r & 1+3r \end{pmatrix}, \ p = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \text{ and } q = \begin{pmatrix} -r \\ 0 \\ 0 \\ \vdots \\ 0 \\ -r \end{pmatrix}.$$

We compute the solution $\mathbf{u}_j^*$ using the Sherman–Morrison formula [10].

Theorem 2.1 (Sherman–Morrison Formula)**.** *Let $\tilde{A} \in \mathbb{R}^{n \times n}$ be a nonsingular matrix and let p, $q \in \mathbb{R}^n$ be column vectors. Then the solution $\mathbf{x}$ to the linear system*

$$(\tilde{A} + pq^T)\mathbf{x} = \mathbf{b}$$

is given by

$$\mathbf{x} = \tilde{A}^{-1}\mathbf{b} - \frac{\tilde{A}^{-1}pq^T\tilde{A}^{-1}\mathbf{b}}{1 + q^T\tilde{A}^{-1}p}.$$

In our case, we compute $\mathbf{f}_j = \tilde{A}^{-1}\mathbf{u}_j^n$, $\mathbf{g} = \tilde{A}^{-1}p$, and obtain the solution $\mathbf{u}_j^*$ as

$$\mathbf{u}_j^* = \mathbf{f}_j - \frac{q^T\mathbf{f}_j}{1+q^T\mathbf{g}} \cdot \mathbf{g}.$$

Since $\tilde{A}$ is a tridiagonal matrix, we solve the systems $\tilde{A}\mathbf{f}_j = \mathbf{u}_j^n$ and $\tilde{A}\mathbf{g} = p$ using the Thomas algorithm. A similar strategy is applied in the y-direction, where the corresponding tridiagonal systems are solved column-wise using the Thomas algorithm with appropriate periodic boundary corrections. Furthermore, even if the governing equation is extended to an N-dimensional domain, the use of the OSM allows for a straightforward extension by applying similar strategies along each spatial direction.

We note that the structure of the matrix A depends on the type of boundary condition imposed.

2.1. **Dirichlet Boundary Condition.** When Dirichlet boundary conditions are applied, the values at the boundaries are prescribed explicitly, such as $u_{0j} = a$, $u_{N_x+1,j} = b$. The unknowns are defined only on the interior nodes, and the resulting matrix A is strictly tridiagonal. The right-hand side vector $\mathbf{u}_j^n$ is modified to incorporate the Dirichlet boundary values:

$$A = \begin{pmatrix} 1+2r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -r & 1+2r & -r \\ 0 & \cdots & 0 & -r & 1+2r \end{pmatrix}, \qquad \mathbf{u}_j^n = \begin{pmatrix} u_{1j}^n + ra \\ u_{2j}^n \\ u_{3j}^n \\ \vdots \\ u_{N_x-1,j}^n \\ u_{N_xj}^n + rb \end{pmatrix}.$$

The standard Thomas algorithm can be directly applied in this case.

2.2. **Neumann Boundary Condition.** For homogeneous Neumann boundary conditions, which specify zero normal derivative at the boundaries, such as

$$\left.\frac{\partial u}{\partial x}\right|_{x=L_x} = 0, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=R_x} = 0,$$

a common approach is to introduce ghost points and approximate the boundary derivatives using central differences. This results in conditions like $u_{0j} = u_{1j}$ and $u_{N_x+1,j} = u_{N_xj}$, and modifies the first and last rows of the system. The resulting matrix A is given by:

$$A = \begin{pmatrix} 1+r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -r & 1+2r & -r \\ 0 & \cdots & 0 & -r & 1+r \end{pmatrix}.$$

Although A is not strictly tridiagonal, it retains a banded structure, and the Thomas algorithm can still be used.

Each boundary condition results in a different structure. Unlike the periodic case, neither Dirichlet nor Neumann boundary conditions introduce cyclic coupling. Therefore, the standard Thomas algorithm can be applied directly without the need for any correction or matrix modification.

3. NUMERICAL EXPERIMENTS

In this section, we perform a convergence test for the Thomas algorithm with periodic boundary conditions. We also compare the numerical behavior under Dirichlet, Neumann, and periodic boundary conditions by solving the Allen–Cahn (AC) equation. The AC equation is given by

$$\frac{\partial u}{\partial t} = \Delta u - \frac{F'(u)}{\epsilon^2},$$

where $u = u(x, y, t)$, $F(u) = 0.25(u^2 - 1)^2$ is a double-well potential, and $\epsilon > 0$ is a small parameter that controls the thickness of the diffuse interface.

To numerically solve the AC equation, we use an OSM, in which the linear diffusion term is treated implicitly using the Thomas algorithm, while the nonlinear reaction term is handled analytically [11]. Specifically, if u^{**} denotes the intermediate solution obtained after solving the diffusion step, then the final solution at the next time level is given by

$$u^{n+1} = \frac{u^{**}}{\sqrt{(1 - (u^{**})^2)\, e^{\left(-\frac{2\Delta t}{\epsilon^2}\right)} + (u^{**})^2}}.$$

This expression represents the exact solution of the reaction part of the equation, with initial u^{**}, and contributes to maintaining accuracy and stability when dealing with stiff nonlinearities. In our simulations, we use the interface thickness parameter $\epsilon = \epsilon_m$, where $\epsilon_m = mh/(2\sqrt{2}\tanh^{-1}(0.9))$ [12].

3.1. **Convergence test.**

To verify the accuracy of the proposed numerical method, we perform convergence tests using the two-dimensional heat equation with a known analytical solution. The computational domain is set to $\Omega = (-0.125, 1.875) \times (-0.25, 0.75)$. The initial condition is chosen as

$$u(x, y, 0) = \cos(\pi x)\cos(\pi y),$$

for which the exact solution is given by

$$u_{\text{exact}}(x, y, t) = \cos(\pi x)\cos(\pi y)e^{(-2\alpha\pi^2 t)}$$

with the diffusion coefficient $\alpha = 1$. We first perform simulations for a sequence of grid resolutions $N_x = N_y = 2^k$, where $k = 3, 4, \ldots, 8$, with a fixed time step size $\Delta t =$ 1.e-7 and

final time $T = 0.04$. The numerical solution is compared with exact solution at the final time, and the discrete ℓ_2-error is computed as

$$\|\mathbf{e}_h\|_{\ell_2} = \sqrt{\frac{1}{N_x N_y} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \left(u_{ij}^{N_t} - u_{\text{exact}}(x_i, y_j, N_t \Delta t) \right)^2},$$

where $N_t = T/\Delta t$ denotes number of iteration. Figure 1(a) shows the ℓ_2-error with respect to the mesh size h on a log-log scale. The observed convergence rate is approximately second order, which is consistent with the theoretical accuracy of the finite difference discretization and the OSM.

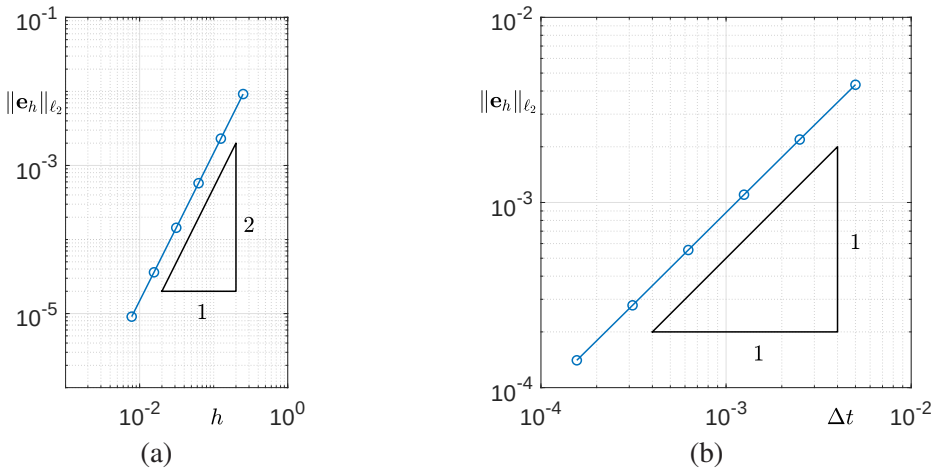

Figure 1: (a) Spatial convergence error with various spatial step size h. (b) Temporal convergence error with respect to time step Δt.

The detailed error values corresponding to each mesh size are reported in Table 1.

Table 1: Discrete ℓ_2-errors and observed convergence rates with respect to mesh size h.

h	1/4	1/8	1/16	1/32	1/64	1/128
$\|\mathbf{e}_h\|_{\ell_2}$	9.2087e-3	2.3034e-3	5.7596e-4	1.4406e-4	3.6081e-5	9.0866e-6
rate		1.9992	1.9997	1.9993	1.9973	1.9894

In addition, we investigate the temporal convergence errors by fixing the spatial grid size $h = 0.004$ and varying the time step $\Delta t = 0.01/2^k$ for $k = 1, 2, \ldots, 6$. The final time is set to $T = 0.04$, and the ℓ_2-error is evaluated similarly. As shown in Fig. 1(b), the method exhibits first-order convergence in time. Corresponding numerical values are summarized in Table 2.

Figure 2 illustrates the comparison between the initial condition, the numerical solution, and the exact solution at final time $T = 0.04$, used in the convergence test. Figure 2(a) displays

Table 2: Discrete ℓ_2-errors and observed convergence rates with respect to time step size Δt.

Δt	$1/200$	$1/400$	$1/800$	$1/1600$	$1/3200$	$1/6400$
$\|\mathbf{e}_h\|_{\ell_2}$	4.3254e-3	2.1885e-3	1.1017e-3	5.5360e-4	2.7838e-4	1.4047e-4
rate		0.9829	0.9902	0.9928	0.9918	0.9868

the initial condition, which is smooth and periodic over the computational domain. Figure 2(b) presents the numerical solution obtained using the Thomas algorithm with periodic boundary conditions, a spatial resolution of $h = 0.004$ and time step size $\Delta t = 1/6400$. Figure 2(c) shows the exact solution, which is used to evaluate the accuracy of the numerical approximation. The numerical solution in Fig. 2(b) matches closely with the exact solution in Fig. 2(c), capturing both the amplitude and the shape of the decaying cosine profile. This visual agreement confirms the second-order spatial accuracy and the robustness of the periodic Thomas algorithm.

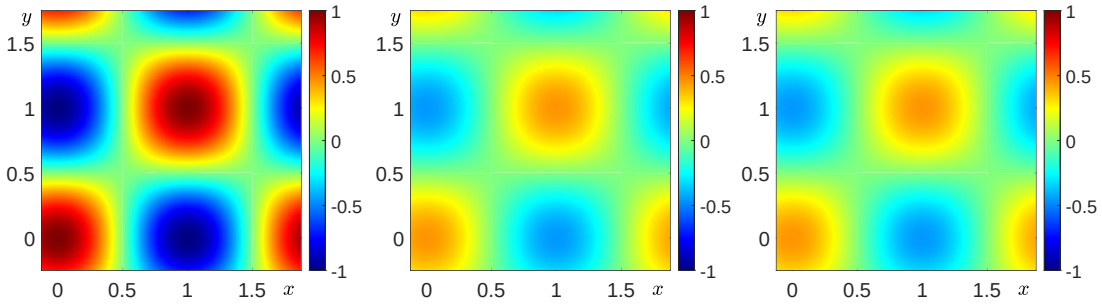

Figure 2: (a) Initial condition. (b) Numerical solution. (c) Exact solution.

3.2. **Comparison of different boundary conditions.** We investigate how different boundary conditions influence the numerical behavior of the AC equation. Using the same initial condition and discretization parameters, we compare the results under Dirichlet, Neumann, and periodic boundary conditions. These experiments are designed to highlight how boundary treatment affects the evolution and stability of the interface. In particular, we observe qualitative differences in interface motion near the domain boundaries, which reflect the intrinsic characteristics of each type of boundary condition.

The initial condition is chosen as a diffuse circular interface centered at $(0.4, 0.4)$, modeled by

$$u(x, y, 0) = \frac{1}{2} + \frac{1}{2} \tanh\left(\frac{r_0 - \sqrt{(x-0.4)^2 + (y-0.4)^2}}{2\sqrt{2}\epsilon} \right),$$

where $r_0 = 0.3$ and $\epsilon = \epsilon_4$. The computational domain is $(0, 2) \times (0, 2)$, discretized with $h = 0.01$, and the time step is set to $\Delta t = 0.5h^2$. The total simulation time is $T = 0.005$, and the AC equation is solved using the OSM introduced earlier.

Figure 3 compares the temporal evolution of the AC equation under three different boundary conditions: periodic, Dirichlet, and Neumann. All simulations start from the same initial condition, consisting of a circular interface centered near the lower-left corner of the domain. The three columns correspond to time snapshots at $t = 0$, $t = 40\Delta t$, and $t = 80\Delta t$, respectively.

In the periodic case (top row), the circular interface shrinks while maintaining its shape and reenters the domain smoothly from the opposite side as it crosses the boundary. This behavior confirms the correct implementation of periodic boundary conditions, preserving topological continuity. Under Dirichlet boundary conditions (middle row), the interface deforms significantly as it approaches the fixed boundaries. Since the solution is constrained to be -1 at the boundary, the circular profile becomes increasingly distorted and exhibits angular features where it encounters the boundary. This constraint enforces a lower potential value than the interior, causing an asymmetrical flattening of the interface and accelerating curvature change near the edge. Such artificial influence alters the natural curvature-driven evolution of the interface. In contrast, the Neumann boundary condition (bottom row) allows the interface to approach the domain boundary more naturally. The shape remains smooth and symmetric, and the interface motion slows down as it reaches the boundary, consistent with the zero-flux condition that prevents material from leaving or entering the domain. These results highlight the significant influence of boundary conditions on the phase evolution described by the AC equation. While periodic and Neumann boundaries preserve the natural curvature-driven behavior, Dirichlet boundaries with a fixed value of -1 introduce artificial distortion due to their enforced asymmetry.

To provide a quantitative assessment of the computational complexity associated with each boundary condition, Table 3 presents the measured CPU time for simulations conducted under Dirichlet, Neumann, and periodic boundary conditions. This simulation is performed on a desktop computer equipped with an Intel Core i9-13900K CPU and 32 GB of RAM, running MATLAB R2023b.

Table 3: CPU time (in seconds) for simulations under different boundary conditions: Dirichlet, Neumann, and periodic.

	Dirichelt	Neumann	periodic
CPU time	0.1131	0.1106	0.1203

4. Conclusion

This paper has presented a detailed and practical guide to implementing the Thomas algorithm under various boundary conditions, with particular emphasis on cases that are often overlooked in standard numerical treatments, such as periodic boundaries. For periodic systems, we demonstrated how to apply the Sherman–Morrison formula to efficiently and accurately

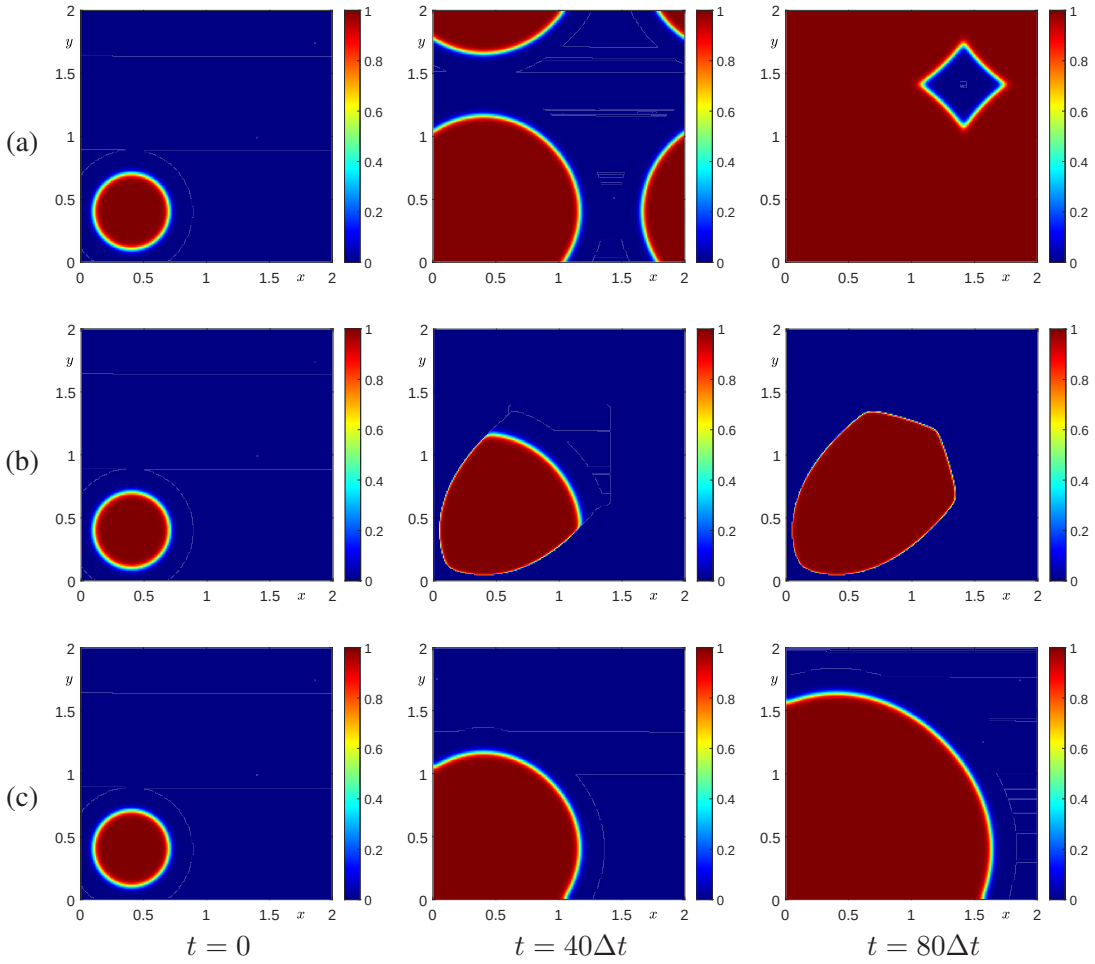

Figure 3: Temporal evolution of the AC equation with various boundary conditions. Rows correspond to different boundary conditions: (a) periodic, (b) Dirichlet, and (c) Neumann. Columns represent time at $t = 0$, $40\Delta t$, and $80\Delta t$.

solve the resulting cyclic tridiagonal systems. The governing equation is decomposed using an OSM and discretized using cell-centered finite differences. A series of numerical experiments was conducted to validate the proposed methods. Convergence tests with the heat equation confirmed the expected accuracy of the scheme. Simulations of the AC equation under Dirichlet, Neumann, and periodic boundary conditions further illustrated how boundary treatments can significantly affect the qualitative behavior of the interface. Beyond providing accurate results, the goal of this work is to serve as a clear and accessible reference for students and researchers who are learning to implement boundary-aware tridiagonal solvers. By carefully

documenting the modifications required for different boundary conditions and supporting the discussion with reproducible MATLAB code (available in the Appendix), we hope this work can aid future efforts in scientific computing and numerical PDEs. By carefully documenting the algorithmic modifications required for each type of boundary condition and supporting the discussion with reproducible MATLAB code (available in the Appendix), this work aims to provide a useful reference for students and practitioners implementing tridiagonal solvers in the context of PDEs.

ACKNOWLEDGMENTS

The author would like to thank editor and reviewers for helpful discussions and constructive comments that improved the manuscript.

APPENDIX

The code provided in the Appendix is also available online at `https://github.com/skwak/skwak.github.io`.

```
clear;

Lx = 0; Rx = 2; Ly = 0; Ry = 2; Nx = 200; Ny = Nx; h = (Rx-Lx)/Nx;
x = linspace(Lx+0.5*h,Rx-0.5*h,Nx);
y = linspace(Ly+0.5*h,Ry-0.5*h,Ny);
[X,Y] = ndgrid(x,y); ep = 4*h/(2*sqrt(2)*atanh(0.9));
dt = 0.5*h^2; T = 100*dt; Nt = round(T/dt);

% Initial condition
r0 = 0.3;
u = 0.5+0.5*tanh((r0-sqrt((X-0.4).^2+(Y-0.4).^2))/(2*sqrt(2)*ep));

% Boundary condition
bc = "periodic";

% Tridiagonal coefficients
r = dt/h^2;
ax = -r*ones(Nx,1); bx = (1+2*r)*ones(Nx,1); cx = -r*ones(Nx,1);
ay = -r*ones(Ny,1); by = (1+2*r)*ones(Ny,1); cy = -r*ones(Ny,1);
switch bc
    case "periodic"
        bx(1) = bx(1)-cx(Nx); bx(Nx) = bx(Nx)-ax(1);
        by(1) = by(1)-cy(Ny); by(Ny) = by(Ny)-ay(1);
    case "dirichlet"
        bx(1) = 1 + 2*r; bx(Nx) = 1 + 2*r;
        by(1) = 1 + 2*r; by(Ny) = 1 + 2*r;
        a = -1; b = -1;
    case "neumann"
```

```
        bx(1) = 1 + r; bx(Nx) = 1 + r;
        by(1) = 1 + r; by(Ny) = 1 + r;
end

% coefficients for Sherman-Morrison formula
px = zeros(Nx,1); px(1) = 1; px(Nx) = 1;
qx = zeros(Nx,1); qx(1) = ax(Nx); qx(Nx) = cx(1);
gx = thomas(ax,bx,cx,px);
py = zeros(Ny,1); py(1) = 1; py(Ny) = 1;
qy = zeros(Ny,1); qy(1) = ay(Ny); qy(Ny) = cy(1);
gy = thomas(ay,by,cy,py);

% Visualization
figure(1); clf; hold on; view(2); box on;
colormap jet; clim([0,1]); colorbar
surf(X,Y,u,'EdgeColor','none');
axis image; axis([Lx,Rx,Ly,Ry])

for n = 1:Nt
    u_star=zeros(Nx,Ny);
    for j = 1:Ny
        switch bc
            case "periodic"
                fj = thomas(ax,bx,cx,u(:,j));
                u_star(:,j) = fj-((qx'*fj)/(1+qx'*gx))*gx;
            case "dirichlet"
                fj = u(:,j);
                fj(1) = fj(1)+r*a; fj(end) = fj(end)+r*b;
                u_star(:,j) = thomas(ax,bx,cx,fj);
            case "neumann"
                u_star(:,j) = thomas(ax,bx,cx,u(:,j));
        end
    end
    for i = 1:Nx
        switch bc
            case "periodic"
                fi = thomas(ay,by,cy,u_star(i,:)');
                u(i,:) = (fi-((qy'*fi)/(1+qy'*gy))*gy)';
            case "dirichlet"
                fi = u_star(i,:)';
                fi(1) = fi(1)+r*a; fi(end) = fi(end)+r*b;
                u(i,:) = thomas(ay,by,cy,fi)';
            case "neumann"
                u(i,:) = thomas(ay,by,cy,u_star(i,:)')';
        end
```

```
    end
    u = u./sqrt((1-u.^2).*exp(-2*dt/ep^2)+u.^2);

    % Visualization
    if mod(n,10)==0
        figure(1); clf; hold on; view(2); box on;
        colormap jet; clim([0,1]); colorbar
        surf(X,Y,u,'EdgeColor','none');
        axis image; axis([Lx,Rx,Ly,Ry])
        drawnow
    end
end

function x = thomas(alpha,beta,gamma,f)
n = length(f); beta_=beta; f_=f;
x = zeros(n,1);
for i = 2:n
    mult = alpha(i)/beta_(i-1);
    beta_(i) = beta_(i)-mult*gamma(i-1);
    f_(i) = f_(i)-mult*f_(i-1);
end
x(n) = f_(n)/beta_(n);
for i = n-1:-1:1
    x(i) = (f_(i)-gamma(i)*x(i+1))/beta_(i);
end
end
```

REFERENCES

[1] B.I. Yun, *An improved implicit Euler method for solving initial value problems*, Journal of the Korean Society for Industrial and Applied Mathematics, **26**(3) (2022), 138–155.

[2] G. Birkhoff, R.S. Varga, and D. Young, *Alternating direction implicit methods*, Advances in computers 3, pp. 189–273, Elsevier, 1962.

[3] G. Russell, K.D. Harkins, T.W. Secomb, J.P. Galons, and T.P. Trouard, *A finite difference method with periodic boundary conditions for simulations of diffusion-weighted magnetic resonance experiments in tissue*, Physics in Medicine & Biology, **57**(4) (2012), N35.

[4] D. Jeong, Y. Nam, M. Bang, H. Kim, and J. Kim, *Reconstructing piecewise constant local volatility surfaces*, Journal of the Korean Society for Industrial and Applied Mathematics, **29**(1) (2025), 26–36.

[5] S. Kwak, S. Ham, J. Wang, H. Kim, and J. Kim, *Effective perpendicular boundary conditions in phase-field models using Dirichlet boundary conditions*, Engineering with Computers, (2025) 1–16.

[6] B.J. Szekeres, and F. Izsák, *A finite difference method for fractional diffusion equations with Neumann boundary conditions*, Open Mathematics, **13**(1) (2015), 000010151520150056.

[7] W. Dai, *A new accurate finite difference scheme for Neumann (insulated) boundary condition of heat conduction*, International Journal of Thermal Sciences, **49**(3) (2010), 571–579.

[8] S.M. Allen, and J.W. Cahn, *Ground state structures in ordered binary alloys with second neighbor interactions*, Acta Metallurgica, **20**(3) (1972), 423–433.

[9] Y. Kim, G. Ryu, and Y. Choi, *Fast and accurate numerical solution of Allen–Cahn equation*, Mathematical Problems in Engineering, **2021**(1) (2021), 5263989.
[10] J. Sherman, and W.J. Morrison, *Adjustment of an inverse matrix corresponding to a change in one element of a given matrix*, The Annals of Mathematical Statistics, **21(1)** (1950), 124–127.
[11] Y. Li, H.G. Lee, D. Jeong, and J. Kim, *An unconditionally stable hybrid numerical method for solving the Allen–Cahn equation*, Computers & Mathematics with Applications, **60**(6) (2010), 1591–1606.
[12] G. Lee and S. Lee, *Study on decouled projection method for Cahn–Hilliard equation*, Journal of the Korean Society for Industrial and Applied Mathematics, **27**(4) (2023), 272–280.