RESEARCH ARTICLE | JANUARY 15 2025

A cell structure implementation of the multigrid method for the two-dimensional diffusion equation

Yongho Choi; Youngjin Hwang; Soobin Kwak; Seokjun Ham; Jyoti; Hyundong Kim; Junseok Kim

AIP Advances 15, 015019 (2025)
https://doi.org/10.1063/5.0247042

A cell structure implementation of the multigrid method for the two-dimensional diffusion equation

Cite as: AIP Advances 15, 015019 (2025); doi: 10.1063/5.0247042
Submitted: 5 November 2024 • Accepted: 29 December 2024 •
Published Online: 15 January 2025

Yongho Choi,[1] Youngjin Hwang,[2] Soobin Kwak,[2] Seokjun Ham,[2] Jyoti,[3] Hyundong Kim,[4,5] and Junseok Kim[2,a)]

AFFILIATIONS

[1] Department of Computer and Information Engineering, Daegu University, Gyeongsan-si, Gyeongsangbuk-do 38453, Republic of Korea
[2] Department of Mathematics, Korea University, Seoul 02841, Republic of Korea
[3] The Institute of Basic Science, Korea University, Seoul 02841, Republic of Korea
[4] Department of Mathematics and Physics, Gangneung-Wonju National University, Gangneung 25457, Republic of Korea
[5] Institute for Smart Infrastructure, Gangneung-Wonju National University, Gangneung 25457, Republic of Korea

[a)] **Author to whom correspondence should be addressed:** cfdkim@korea.ac.kr. **URL:** https://mathematicians.korea.ac.kr/cfdkim/

ABSTRACT

To solve the two-dimensional diffusion equation using the finite difference method, we propose a simple MATLAB implementation of the multigrid method. The diffusion equation plays a fundamental role in modeling many significant physical phenomena and is ubiquitous in many governing equations. Some examples include the reaction–diffusion equations, the convection–diffusion equations, and others. These equations often lack analytical solutions or pose extreme challenges in finding them. Therefore, numerical techniques are indispensable for obtaining practical and accurate approximations for these equations. The multigrid method is known for its computational efficiency and effectiveness as an iterative technique for solving the discretized diffusion equation. Due to its popularity, the multigrid method has been implemented in several programming languages, such as Python, Java, C++, C, Fortran, and others. However, it is not easy for beginners to understand the implementation of the multigrid method due to its complex data structures and recursive routines. To resolve these difficulties, we develop a straightforward MATLAB implementation of the two-dimensional diffusion equation using a cell structure in MATLAB. This work provides an accessible and efficient framework for understanding and applying the multigrid method, thereby simplifying its implementation for researchers and practitioners.

I. INTRODUCTION

In this study, using MATLAB software,[1] we present a straightforward, intuitive, and easily applicable multigrid algorithm that replaces the recursive function with a for loop using a cell array structure. For a specific example of a MATLAB multigrid implementation, let us consider the two-dimensional (2D) diffusion equation for $(x, y) \in \Omega, t > 0$,

$$\frac{\partial u(x,y,t)}{\partial t} = \Delta u(x,y,t), \tag{1}$$

where Ω is the given 2D domain, and $u(x, y, t)$ is a substance concentration at (x, y) and t. The following equation is used:

$$\mathbf{n} \cdot \nabla u(x,y,t) = 0, \quad \mathbf{x} \in \partial\Omega, \quad t \geq 0,$$

where $\mathbf{n}$ is normal to $\partial\Omega$. The diffusion equation is essential for modeling processes involving heat transfer, chemical transport, and biological interactions, as it describes how substances spread over time. It serves as the foundation of many scientific and engineering applications, which makes it fundamental in understanding physical phenomena and predicting system behaviors in diverse fields, from environmental science to material engineering. Therefore, it is

essential to use fast and accurate numerical methods to solve the diffusion equation. Among these methods,[2–5] the multigrid method is a computational solver used to solve various partial differential equations (PDEs) in the field of computational physics and engineering. It is an iterative algorithm that efficiently solves the resolution of different scales of features in the solution. By using a hierarchy of grids, the multigrid technique can accelerate the convergence of iterative solvers and makes it particularly effective for problems with smooth and oscillatory components. The recursive function for a multigrid method is a very efficient routine; however, it can be particularly challenging for beginners to fully understand. The primary objective of this paper is to present a straightforward, intuitive, and easily implementable multigrid algorithm that replaces the recursive function with a for loop while using a cell array structure and to assist beginners and researchers in developing intuitive and simple numerical solution programs using the multigrid method under a unified platform, MATLAB, for program development.

This method is widely applied in various areas, including fluid dynamics, heat transfer, and structural mechanics, to obtain accurate and fast solutions to PDEs.[6–12] Many researchers have implemented multigrid methods using various computer programming languages such as Fortran 77,[13–15] C,[16] and C++.[17,18] Under the Dirichlet conditions on a Cartesian mesh with non-regular boundaries, Guillet and Teyssier[19] developed a multigrid solver for solving the Poisson equation. Gupta and Zhang[20] presented the high accuracy multigrid algorithm for the 3D convection–diffusion equation. The multigrid method[21] and parallel multigrid method[22] have been researched to solve the inverse problems related to the heat equation. The multigrid method has been implemented in the past with the MATLAB program.[23] In Ref. 24, the authors utilized graphics processing units (GPUs) and MATLAB 2010b. The multigrid codes were optimized using CUDA. In Ref. 25, the authors proposed an efficient multigrid-based topology optimization approach using MATLAB and CUDA. MATLAB released a parallel computing toolbox. This toolbox includes functions that allow for calling parallel thread execution (PTX) directly, starting from version 2010b or any newer versions. The handwritten CUDA kernel can be transformed into PTX code, and this PTX code can also be called a MATLAB function and perform parallel operations in MATLAB. Therefore, the authors showed that the computing speed was faster than central processing unit (CPU) computing by using GPU computing capable of parallel operation for the multigrid method.

The structure of this paper can be outlined as follows. Section II presents a multigrid algorithm using a cell structure in MATLAB to find the numerical approximation of the 2D diffusion equation. Section III presents the computational experiments to show the performance of the implemented numerical scheme. The conclusions drawn from the multigrid algorithm are provided in Sec. IV. In addition, we have included the MATLAB program for the 2D diffusion equation in the Appendix, which enables interested readers to customize and apply the multigrid program to suit their specific needs.

II. NUMERICAL SOLUTION ALGORITHM

Before explaining the multigrid method using a cell structure, we would like to explain how to define and use the cell structure in the MATLAB program. In MATLAB, a cell structure, or cell array, is a data type that allows you to store arrays of varying sizes and types. Unlike standard arrays, which require uniform data types and sizes, cell arrays can contain a mix of numbers, strings, vectors, matrices, and even other cell arrays, making them highly flexible for handling heterogeneous data. Each element in a cell array is accessed using curly braces, {·}, rather than the standard parentheses, (·), used for regular arrays. Let us consider the following example presented within a box.

```
>> C{1}=[1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]; C{2}=[1 2; 3 4];
>> C
C = 1x2 cell array
    {4x4 double}    {2x2 double}
>> C{1}
ans = 1     2     3     4
      5     6     7     8
      9    10    11    12
     13    14    15    16
>> C{2}
ans = 1     2
      3     4
>> C{1}(2,3)
ans = 7
```

A cell array is a data structure made up of indexed containers, known as cells, where each cell can hold data of any type and size. We define and access the contents of cells by indexing with curly braces, {}. The order of data is held as an index. In the above-mentioned example, C{1} contains a 4 × 4 matrix, and C{2} contains a 2 × 2 matrix. When we want to call the element in the second row and third column among the included elements in C{1}, then we use C{1}(2, 3) by enclosing indices in round parentheses, ().

For the 2D domain $\Omega = (a, b) \times (c, d)$, we define a uniform cell-centered discrete domain as $\Omega_h = \{(x_i, y_j) | x_i = a + (i - 0.5h)$,

$y_j = c + (j - 0.5h),\ i = 1, \ldots, N_x,\ j = 1, \ldots, N_y\}$, where $h = (b-a)/N_x = (d-c)/N_y$ is a uniform grid size. We note that N_x and N_y must be powers of 2 to use the multigrid method. Let $u_{ij}^n = u(x_i, y_j, n\Delta t)$, where Δt denotes the temporal step size. By using the fully implicit scheme and the standard discrete Laplace operator in the 2D space, Eq. (1) can be discretized as follows:

$$\frac{u_{ij}^{n+1} - u_{ij}^n}{\Delta t} = \frac{1}{h^2}\left(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} - 4u_{ij}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right). \quad (2)$$

In this study, we use Neumann boundary conditions. That is,

$$u_{0,j}^n = u_{1,j}^n,\ u_{N_x+1,j}^n = u_{N_x,j}^n \quad j = 1, 2, \cdots, N_y,$$
$$u_{i,0}^n = u_{i,1}^n,\ u_{i,N_y+1}^n = u_{i,N_y}^n \quad i = 1, 2, \cdots, N_x.$$

Now, we explain a single V-cycle. The schematic of the V-cycle is shown in Fig. 1. For a simple and clear description, we assume that N_x and N_y are both powers of 2, specifically $N_x = N_y = 2^N$. For $k = 1, 2, \ldots, N$, we define discrete coarser domains as follows:

$$\Omega_k = \Big\{(x_i, y_j) : x_i = (i - 0.5)h_k,$$
$$y_j = (j - 0.5)h_k | 1 \le i, j \le 2^{N+1-k},\ \text{and}\ h_k = 2^{k-1}h\Big\},$$

which means that Ω_k represents a finer grid compared to Ω_{k+1} by a factor of 2. The numerical solution $u(x_i, y_j, n\Delta t)$ at time $t = n\Delta t$ on the discrete domain Ω_k is denoted as $u_{i,j,k}^n$. We define the operator $\mathscr{L}_k$ and the source term $f_{i,j,k}$ as follows:

$$\mathscr{L}_k(u_{i,j,k}^{n+1}) = \frac{u_{i,j,k}^{n+1}}{\Delta t} - \frac{1}{h^2}\left(u_{i+1,j,k}^{n+1} + u_{i-1,j,k}^{n+1} + u_{i,j+1,k}^{n+1} + u_{i,j-1,k}^{n+1} - 4u_{i,j,k}^{n+1}\right), \quad (3)$$

$$f_{i,j,k} = \frac{u_{i,j,k}^n}{\Delta t}, \quad (4)$$

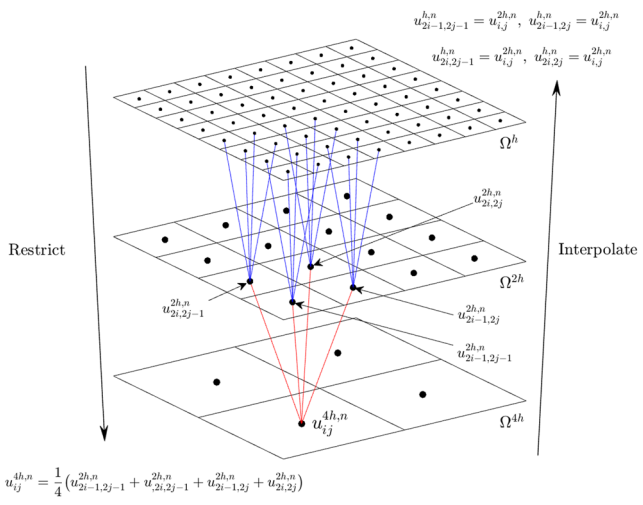

FIG. 1. Schematic diagram of the V-cycle for the one point on Ω^{4h}.

on the discrete domain Ω_k. Let

$$u_k^{n+1,m+1} = MGcycle(k, u_k^{n+1,m}, \mathscr{L}_k, f_k, \nu_1, \nu_2),$$

where ν_1 and ν_2 are the total number of pre- and post-smoothing. The V-cycle is divided into three steps. First step: pre-smoothing, second step: coarse grid correction, and final step: post-smoothing.

- Step 1. Presmoothing
  For $k = 1$, the presmoothing step can be written as

  $$\tilde{u}_k^{n+1,m} = SMOOTH^{\nu_1}(u_k^{n+1,m}, \mathscr{L}_k, f_k).$$

  We use the following Gauss–Seidel iteration method for the presmoothing step:

  $$u_{ijk}^{n+1,m,s+1} = \Big(f_{ij,k} + u_{i-1,j,k}^{n+1,m,s+1} + u_{i+1,j,k}^{n+1,m,s} + u_{i,j-1,k}^{n+1,m,s+1} + u_{i,j+1,k}^{n+1,m,s}\Big)\left(\frac{1}{\Delta t} + \frac{4}{h^2}\right),$$

  where s is the presmoothing iteration step.
- Step 2. Coarse grid correction
  In the coarse grid correction step, we calculate the defect and correct the solution in the fine grid using restricted defects. The defect is calculated as follows: For $k = 1, 2, \ldots, N-1$,

  $$\tilde{d}_k^m = f_k - \mathscr{L}_k(\tilde{u}_k^{n+1,m}).$$

  Subsequently, we apply the restriction operator I_k^{k+1} to restrict the defect $\tilde{d}_k^m$ from the k-level to $(k+1)$-level functions,

  $$d_{k+1}(x_i, y_j) = I_k^{k+1} d_k(x_i, y_j) = \frac{1}{4}\left[d_k\left(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}\right) + d_k\left(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}\right) + d_k\left(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}\right) + d_k\left(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}\right)\right].$$

  Next, an approximate solution $\hat{v}_{k+1}^{n+1,m}$ of the coarse grid equation is computed on Ω_{k+1},

  $$\mathscr{L}_{k+1}(u_{k+1}^{n+1,m}) = \tilde{d}_{k+1}^m.$$

  By employing a relaxation iteration solver, *Relax*, we utilize a zero grid function as the initial approximation:

  $$\hat{v}_{k+1}^{n+1,m} = Relax^{\nu}(0, \mathscr{L}_{k+1}, \tilde{d}_{k+1}^m),\ \text{for}\ k = 1, 2, \ldots, N-1,$$

  where ν is the total number of relaxation iterations. We correct the approximation on Ω_k using an interpolated correction for $k = N-1, N-2, \ldots, 1$,

  $$u_k^{m,\text{after CGC}} = \tilde{u}_k^{n+1,m} + \hat{v}_k^{n+1,m},$$

  where $\hat{v}_k(x_i, y_j) = I_{k+1}^k \hat{v}_{k+1}(x_i, y_j) = \hat{v}_{k+1}(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}})$ for the i and j odd-numbered integers.
- Step 3. Postsmoothing

  $$\tilde{u}_k^{n+1,m+1} = SMOOTH^{\nu_2}(u_k^{m,\text{after CGC}}, \mathscr{L}_k, f_k),$$

where v_2 is the total number of post-smoothing iterations. This completely describes one cycle of the multigrid V-cycle. A single multigrid cycle terminates when the resultant error $\|u^{n+1,m+1} - u^{n+1,m}\|_\infty$ falls below a specified tolerance level denoted as *tol*.

III. NUMERICAL EXPERIMENTS

A. Convergence experiment

We confirm the convergence of the multigrid solver with Δt and h on $\Omega = (0,1) \times (0,1)$. The following initial condition and its analytic solution of the diffusion equation are used,

$$u_{ij}^0 = \cos(\pi x_i)\cos(\pi y_j), \tag{5}$$

$$U_{exact}(x,y,t) = e^{-2\pi^2 t}\cos(\pi x)\cos(\pi y). \tag{6}$$

For some fixed total time $T = N_t \Delta t$, we define l_2-error with the discrete l_2-norm as

$$\|e(N_x,N_y,\Delta t)\|_2 = \sqrt{\frac{1}{N_x N_y}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(u_{ij}^{N_t} - U_{exact}(x_i,y_j,N_t\Delta t)\right)^2}. \tag{7}$$

Then, the temporal and spatial convergence rates are defined as

$$R_t = \log_2\left(\frac{\|e(N_x,N_y,2\Delta t)\|_2}{\|e(N_x,N_y,\Delta t)\|_2}\right)$$

and

$$R_s = \log_2\left(\frac{\|e(N_x,N_y,\Delta t)\|_2}{\|e(2N_x,2N_y,\Delta t)\|_2}\right),$$

respectively. First, we consider the fully implicit scheme in Eq. (2).

For the test outlined in Table I, we set the grid size to a fixed value of $h = 1/256$, and the calculation is executed until reaching a time of $T = 6.4 \times 10^{-4}$. In Table I, we can see the numerical errors and convergence rate in time. In Table II, we fix $\Delta t = 1 \times 10^{-7}$ and calculate up to time $T = 1 \times 10^{-5}$. Table II lists the errors and convergence rate in space for the implicit scheme. From Tables I and II, we can confirm that the fully implicit scheme of the multigrid method has first- and second-order accuracy for time and space, respectively.

Then, we consider the Crank–Nicolson (CN) scheme for the diffusion equation as follows:

$$\frac{u_{ij}^{n+1} - u_{ij}^n}{\Delta t} = \frac{1}{2}\left(\Delta_d u_{ij}^{n+1} + \Delta_d u_{ij}^n\right), \tag{8}$$

$$= \frac{1}{2}\left(\frac{u_{i+1,j}^{n+1} - u_{ij}^{n+1}}{h^2} - \frac{u_{ij}^{n+1} - u_{i-1,j}^{n+1}}{h^2} + \frac{u_{i,j+1}^{n+1} - u_{ij}^{n+1}}{h^2} - \frac{u_{ij}^{n+1} - u_{i,j-1}^{n+1}}{h^2} + \frac{u_{i+1,j}^{n} - u_{ij}^{n}}{h^2} - \frac{u_{ij}^{n} - u_{i-1,j}^{n}}{h^2} + \frac{u_{i,j+1}^{n} - u_{ij}^{n}}{h^2} - \frac{u_{ij}^{n} - u_{i,j-1}^{n}}{h^2}\right). \tag{9}$$

To evaluate the temporal convergence of the CN scheme, we maintain a fixed grid size of $h = 1/256$ and simulate the calculation until a total time of $T = 1.28 \times 10^{-1}$. Table III lists the errors and convergence rate in time for the CN scheme. In Table IV, we use the time step as $\Delta t = 1 \times 10^{-7}$ and fix the total time as $T = 1 \times 10^{-5}$. Table IV lists the CN scheme's errors and convergence rate in space. The outcomes obtained from Tables III and IV illustrate that the Crank–Nicolson (CN) scheme utilized in the multigrid method exhibits second-order accuracy in both time and space.

TABLE I. Errors and rates of the fully implicit scheme for time step Δt.

Δt	1×10^{-5}		2×10^{-5}		4×10^{-5}		8×10^{-5}		1.6×10^{-4}
l_2-error	6.9377×10^{-7}		1.3091×10^{-6}		2.5393×10^{-6}		4.9977×10^{-6}		9.9069×10^{-6}
Rate		0.916		0.956		0.977		0.987	

TABLE II. Errors and rates of the fully implicit scheme for grid size h.

$N_x = N_y$	256		128		64		32		16
l_2-error	1.3360×10^{-10}		5.0517×10^{-10}		1.9913×10^{-9}		7.9342×10^{-9}		3.1677×10^{-8}
Rate		1.919		1.979		1.994		1.997	

TABLE III. Errors and rates of the Crank–Nicolson scheme for time step Δt.

Δt	4×10^{-3}		8×10^{-3}		1.6×10^{-2}		3.2×10^{-2}		6.4×10^{-2}
l_2-error	5.0699×10^{-5}		2.0826×10^{-4}		8.4121×10^{-4}		3.4166×10^{-3}		1.4480×10^{-2}
Rate		2.038		2.014		2.022		2.083	

TABLE IV. Errors and rates of the Crank–Nicolson scheme for grid size h.

$N_x = N_y$	512		256		128		64		32
l_2-error	3.0960×10^{-10}		1.2384×10^{-9}		4.9534×10^{-9}		1.9812×10^{-8}		7.9231×10^{-8}
Rate		2.000		2.000		2.000		2.000	

Now, we consider the multigrid method's CPU times for various grid points. Table V and Fig. 2 list and show, respectively, the CPU time results. Throughout the test, we maintained a fixed value of $\Delta t = 1 \times 10^{-7}$ and executed the program until a total time of $T = 1 \times 10^{-5}$ with $N_t = 100$. Table V displays the CPU time for different grid sizes, specifically, $N_x \times N_y = 16 \times 16$, 32×32, 64×64, 128×128, and 256×256. To calculate the average CPU time, each test was performed five times. Furthermore, Fig. 2 provides a visual representation of the results. It clearly indicates that as we increased the grid size from 16×16 to 256×256, the CPU times linearly increased with respect to grid points.

B. Evolutionary behaviors

We perform the computational simulation using the developed MATLAB implementation of the multigrid method for solving the 2D diffusion equation. Let us consider the randomly initial perturbations on $\Omega = (0,1) \times (0,1)$,

$$u(x,y,0) = 0.5\ \mathrm{rand}(x,y),$$

TABLE V. Average CPU time using different grid sizes for the multigrid method.

$N_x \times N_y$	16×16	32×32	64×64	128×128	256×256
CPU time	0.123 934	0.359 774	1.361 847	5.083 904	20.077 935

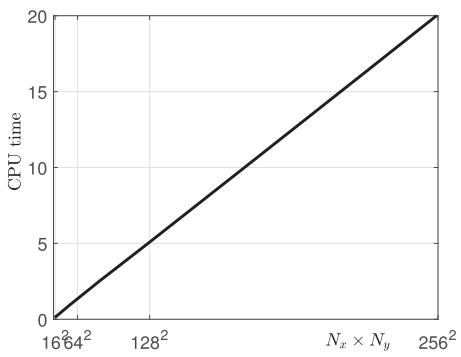

FIG. 2. Average CPU time with respect to different grid sizes $N_x \times N_y = 16 \times 16$, 32×32, 64×64, 128×128, and 256×256 for the multigrid method.

where $\mathrm{rand}(x,y)$ has the random value between -1 and 1, see Fig. 3(a). Here, $N_x = N_y = 128$, $h = 1/128$, and $\Delta t = 0.5h^2$; and the level of relaxation and tolerance are set to $relax = 2$ and $tol = 1.0 \times 10^{-7}$.

Figure 3 illustrates the temporal evolution of the solution $u(x,y,t)$ of a diffusion equation solved using a multigrid method. The four subfigures depict snapshots at different time steps and show how the initial random perturbations dissipate over time due to diffusion. Figure 3(a) shows the initial state, where $u(x,y,0)$ displays random perturbations across the domain $(0,1) \times (0,1)$. The surface appears highly irregular, indicating a high level of spatial variability. Figure 3(b), at $t = 2\Delta t$, shows that as time progresses, the perturbations begin to smooth out slightly, though the solution still exhibits noticeable fluctuations. This suggests the initial phase of diffusion, where high-frequency components are gradually dampened. Figure 3(c), at $t = 5\Delta t$, shows that further into the simulation, the surface becomes increasingly smooth as more of the initial irregularities dissipate. The solution now displays lower amplitude variations, which illustrates the effectiveness of the diffusion process in diminishing the intensity of perturbations. Figure 3(d), at $t = 10\Delta t$, illustrates that at this later time step, the surface is predominantly smooth, with only minimal fluctuations remaining. The diffusion process has largely homogenized the solution, which suggests that the system is approaching a steady-state. The sequence demonstrates the efficacy of the proposed cell structured multigrid method in capturing the diffusion process, with initial random perturbations gradually diminishing over time. The solution approaches a smoother state, characteristic of diffusion-driven systems.

Next, we also conduct the numerical simulation using the proposed MATLAB implementation of the multigrid method to solve the 2D diffusion equation with the sharp initial condition as shown in Fig. 4(a). As illustrated in Figs. 4(b)–4(d), our proposed method demonstrates efficient and robust solutions for the 2D diffusion equation over time, even when dealing with sharp initial conditions of complex shape. The figure shows the temporal evolution of the 2D diffusion equation and highlights the smoothing effect of the diffusion equation at various time steps.

Figure 4(a) displays the initial condition, characterized by a sharp, complex shape resembling a spiral pattern with high peaks and steep gradients, which indicates a high level of initial perturbation. In Fig. 4(b), the sharp edges of the initial shape begin to smooth out, as the diffusion process starts to take effect. Although the peak heights are slightly reduced, prominent features of the original shape remain visible. In Fig. 4(c), as time progresses, the solution continues to smooth, with a notable reduction in the amplitude of peaks and valleys. The initial perturbation fades further, which reflects a transition toward a more homogeneous state. In Fig. 4(d),

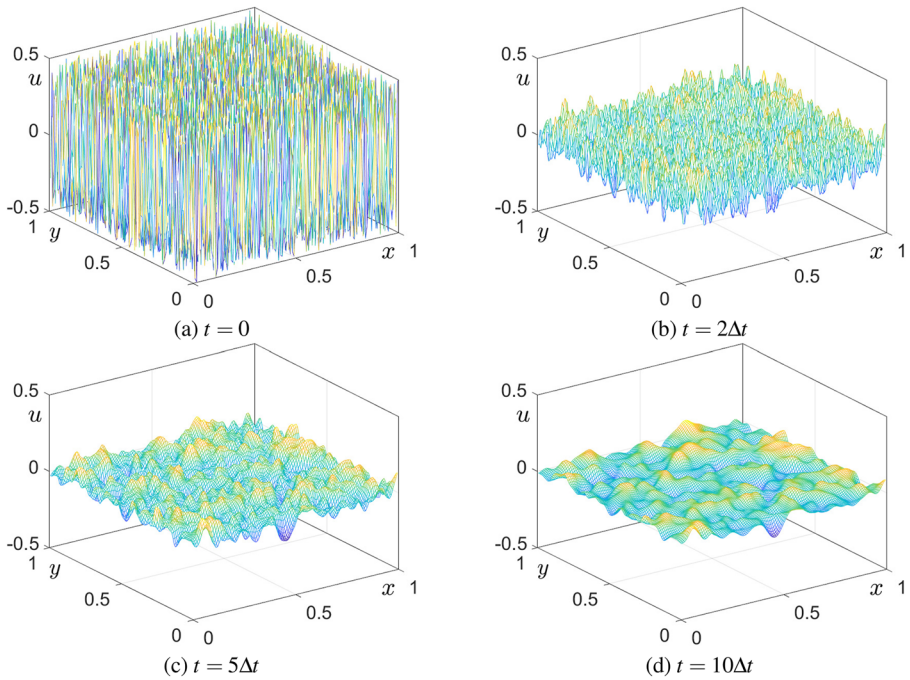

FIG. 3. Temporal evolution of the diffusion equation with random initial perturbations, computed using the proposed cell structure MATLAB program for the multigrid method of the diffusion equation. (a) Initial state at $t = 0$, which shows highly irregular random perturbations. (b) State at $t = 2\Delta t$, where the perturbations begin to smooth out due to diffusion effects. (c) State at $t = 5\Delta t$ shows further smoothing and reduced fluctuations. (d) State at $t = 10\Delta t$ demonstrates near-uniformity as the solution approaches equilibrium.

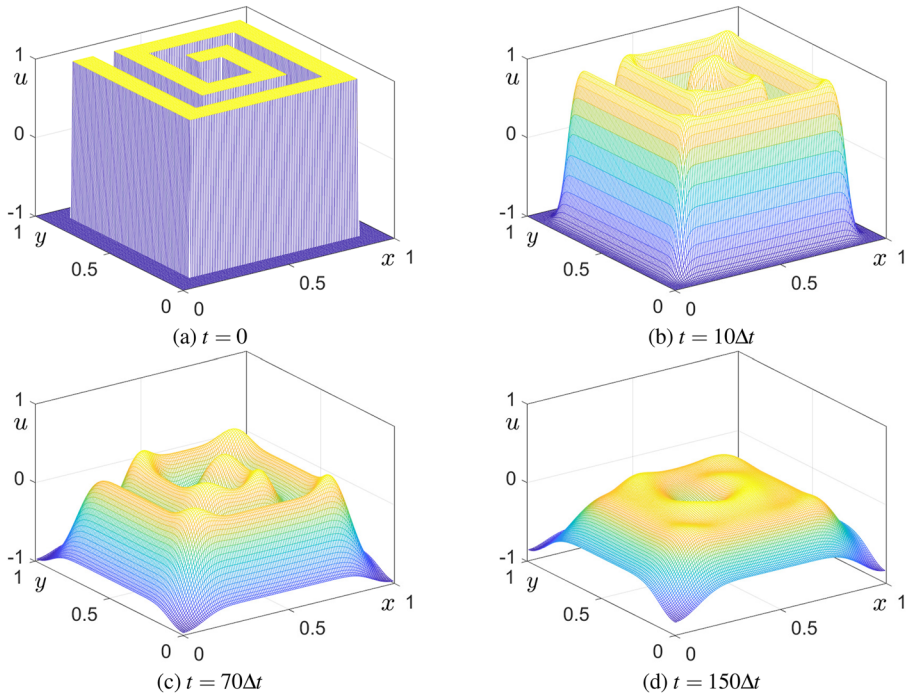

FIG. 4. Temporal evolution of the diffusion equation starting from an initial square spiral shape at $t = 0$. (a) Initial square spiral shape. (b)–(d) Temporal evolution of the solution computed using the proposed cell structure multigrid method at $t = 10\Delta t$, $t = 70\Delta t$, and $t = 150\Delta t$, illustrating the progressive smoothing and dissipation of the perturbation over time.

at this later stage, the surface is predominantly smooth with only minor undulations, which indicates that the diffusion process has nearly homogenized the solution. This suggests that the system is approaching a steady-state with minimal residual fluctuations. This sequence demonstrates the effectiveness of the proposed method in efficiently damping high-frequency components over time, which results in a progressively smoother solution that moves toward equilibrium.

We define

$$u(x,y,0) = \begin{cases} 1, & \text{if } (x+0.1)^2 \le \left(1 - \frac{y^2}{a^2}\right) b^2 \text{ and } x \ge 0.2\cos(2\pi y) - 0.2, \\ -1, & \text{otherwise,} \end{cases}$$

where $a = 0.5$ and $b = 0.8$. The initial condition is illustrated in Fig. 5(a). The parameters used are $N_x = 256$ and $N_y = 256$, $\Delta t = h^2$, $relax = 2$, and $tol = 1.0 \times 10^{-7}$. Figure 5 displays the time progress of the computational solutions. Over the course of time, the initially sharp transition undergoes a transformation and evolves into a more gradual shift, and concentration levels transition into a state characterized by stable, flatter values.

Figure 5(a) shows the initial condition, where the concentration $u(x,y,0)$ has a sharply defined boundary separating regions of $u = 1$ and $u = -1$. The high-gradient boundary reflects a strong initial perturbation, setting up a steep transition between the two regions. In Fig. 5(b), the diffusion process begins to soften the initial sharp edges. The boundary is less defined, indicating that the concentration is gradually spreading out. However, the structure of the initial condition remains visible and shows only minor smoothing effects at this early stage. In Fig. 5(c), as diffusion progresses, the transition becomes increasingly gradual, with the peaks lowering and the high-gradient boundary flattening out. This stage shows a marked reduction in the steepness of the initial transition, which reflects the ongoing spread of concentration levels. In Fig. 5(d), the system has evolved toward a steady state, with a smooth distribution and minimal variation in concentration levels. The previously sharp transition is now diffused, which results in a smoother surface. Overall, the series of images in Fig. 5 effectively demonstrates the diffusion process facilitated by the proposed algorithm, which efficiently smooths out the initially sharp boundary, leading to a gradual and stable flattening of concentration levels over time.

IV. CONCLUSION

In this study, we presented a simple cell structure MATLAB implementation of the multigrid method for solving the 2D diffusion equation using the finite difference method. The diffusion equation is fundamental for modeling various physical

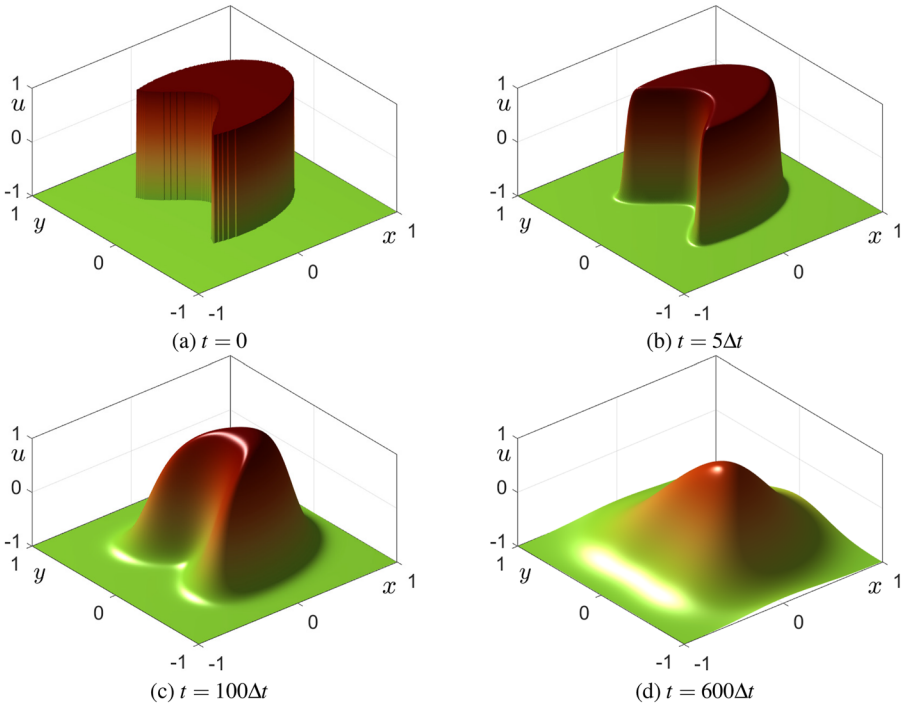

FIG. 5. Temporal evolution of the diffusion equation solution using the proposed cell structure MATLAB program for the multigrid method. (a) Sharp initial profile at $t = 0$. (b) State at $t = 5\Delta t$, where diffusion begins and edges become smooth. (c) State at $t = 100\Delta t$, showing significant spreading and further smoothing of the profile. (d) State at $t = 600\Delta t$, where the solution approaches a near-uniform state due to diffusion.

phenomena, which makes it an essential component in numerous governing equations. These include reaction–diffusion, convection–diffusion, Navier–Stokes, spatial predator-prey, and Allen–Cahn equations, and so on. The multigrid method has emerged as a highly efficient and effective iterative method for solving the diffusion equation. Although the method has been implemented in many programming languages, understanding its implementation can pose challenges, particularly for beginners, due to complex data structures and recursive routines. To resolve these difficulties, we have proposed a straightforward MATLAB implementation of the multigrid method, making it accessible and easily understandable for users at all levels. Using a MATLAB cell structure, we made the code implementation more intuitive and manageable. Moreover, our proposed MATLAB implementation of the multigrid method presented in this paper is concise, consisting of ~90 lines of code. This MATLAB implementation can provide a valuable resource for researchers and practitioners applying the multigrid method to solve the diffusion equation. It offers a simple yet efficient approach that can be easily understood and implemented, facilitating further exploration and experimentation in various scientific and engineering domains. From the numerical tests conducted using the proposed cell structure MATLAB program for the multigrid method, we observed that it produced reasonable results comparable to those obtained with the conventional recursive routine in the multigrid method. In fact, the results should be identical within machine precision because the core algorithm remains the same. However, we achieved the same numerical results using a much simpler and more straightforward cell structure implementation for the multigrid method. The proposed cell structure implementation of the multigrid method can be adapted to solve other types of partial differential equations, such as the Allen–Cahn equation, the Cahn–Hilliard equation,[26,27] the nonlocal Cahn–Hilliard equation,[28] and the Navier–Stokes equations, as examples. The Allen–Cahn equation[29] is defined as follows:

$$\frac{\partial u(x,y,t)}{\partial t} = -\frac{u^3(x,y,t) - u(x,y,t)}{\varepsilon^2} + \Delta u(x,y,t), \tag{10}$$

which includes the diffusion equation. The Cahn–Hilliard equation[30] is defined as follows:

$$\frac{\partial u(x,y,t)}{\partial t} = \Delta\left(\frac{u^3(x,y,t) - u(x,y,t)}{\varepsilon^2} - \Delta u(x,y,t)\right), \tag{11}$$

which includes the biharmonic term, which is a double application of the Laplacian operator. The Navier–Stokes equations[31,32] are defined as follows:

$$\frac{\partial u(x,y,t)}{\partial t} + u(x,y,t)\frac{\partial u(x,y,t)}{\partial x} + v(x,y,t)\frac{\partial u(x,y,t)}{\partial y} = -\frac{\partial p(x,y,t)}{\partial x} + \Delta u(x,y,t), \tag{12}$$

$$\frac{\partial v(x,y,t)}{\partial t} + u(x,y,t)\frac{\partial v(x,y,t)}{\partial x} + v(x,y,t)\frac{\partial v(x,y,t)}{\partial y} = -\frac{\partial p(x,y,t)}{\partial y} + \Delta v(x,y,t), \tag{13}$$

$$\frac{\partial u(x,y,t)}{\partial x} + \frac{\partial v(x,y,t)}{\partial y} = 0, \tag{14}$$

which is typically solved using a projection method that includes the Laplacian operator.

ACKNOWLEDGMENTS

The first author (Y. Choi) was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant No. 2022R1I1A3072824). Jyoti was supported by the Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (Grant No. 2022H1D3A2A02081237). Hyundong Kim was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant No. NRF-2021R1A6A1A03044326). The corresponding author (J. S. Kim) received support from the Brain Korea 21 (BK 21) FOUR program, funded by the Ministry of Education. The authors would like to acknowledge the reviewers for their valuable comments and suggestions, which have greatly improved the quality of this paper.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Yongho Choi: Conceptualization (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Youngjin Hwang**: Investigation (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Soobin Kwak**: Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Seokjun Ham**: Methodology (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jyoti**: Data curation (equal); Investigation (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Hyundong Kim**: Formal analysis (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Junseok Kim**: Conceptualization (equal); Formal analysis (equal); Funding acquisition (equal); Investigation (equal); Methodology (equal); Project administration (equal); Resources (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX: MATLAB CODE FOR THE MULTIGRID METHOD APPLIED TO THE 2D DIFFUSION EQUATION

We present a MATLAB implementation of the multigrid method of the FDM for the 2D diffusion equation.

```
clear all; close all; clc;
global NX NY dt H
% Define domain and grid
xleft=0; xright=1; yleft=0; yright=2; Nx=32*2; h=(xright-xleft)/Nx; Ny=64*2;
x=linspace(xleft+h/2,xright-h/2,Nx); y=linspace(yleft+h/2,yright-h/2,Ny);
% Total levels for multigrid
total_levels=min(log2(Nx), log2(Ny));
% Construct cell structure
for k=1:total_levels
  Cu{k}=zeros(Nx/2^(k-1),Ny/2^(k-1)); f(k)={zeros(Nx/2^(k-1),Ny/2^(k-1))};
  H(k)=2^(k-1)*h; NX(k)=Nx/2^(k-1); NY(k)=Ny/2^(k-1);
end
% Initial condition
for i = 1:NX(1)
  for j = 1:NY(1)
    Cu{1}(i,j) = cos(pi*x(i))*cos(2*pi*y(j));
  end
end
% Multigrid parameters
dt=0.5*H(1)^2; relax=2; tol=1.0e-7; max_it=100;

% Multigrid iteration
for it=1:max_it
f(1)={Cu{1}/dt}; err=2*tol;
% V-cycle
while err>tol
  TCu=Cu{1};
  % Restriction
  for k=1:total_levels-1
    if k>1
      Cu{k}=zeros(NX(k),NY(k));
    end
    Cu{k}=relax_2D(Cu{k},f{k},relax,k); % Pre-smoothing
    d=defect(Cu{k},f{k},k); % Course grid correction
    % Restrict defect
    for i=1:NX(k+1)
      for j=1:NY(k+1)
        f{k+1}(i,j)=(d(2*i,2*j)+d(2*i-1,2*j)+d(2*i,2*j-1)+d(2*i-1,2*j-1))/4;
      end
    end
  end
  k=total_levels; Cu{k}=zeros(NX(k),NY(k));
  % Solve on the coarsest grid
  Cu{k}=relax_2D(Cu{k},f{k},relax,k);
  % Interpolation
  for k=total_levels-1:-1:1
    % Interpolate solution
    for i=1:NX(k+1)
      for j=1:NY(k+1)
        Cu{k}(2*i-1:2*i,2*j-1:2*j) = Cu{k}(2*i-1:2*i,2*j-1:2*j)+Cu{k+1}(i,j);
      end
    end
    Cu{k}=relax_2D(Cu{k},f{k},relax,k); % Post-smoothing
  end
  err=norm(TCu-Cu{1})/sqrt(NX(1)*NY(1));
  end
end
```

```
% Gauss-Seidel smoothing
function Cuk = relax_2D(Cu,f,relax,k)
global NX NY dt H
for iter=1:relax
  for i=1:NX(k)
    for j=1:NY(k)
      sor=f(i,j); coef=1/dt;
      if i>1
        sor=sor+Cu(i-1,j)/H(k)^2; coef=coef+1/H(k)^2;
      end
      if i<NX(k)
        sor=sor+Cu(i+1,j)/H(k)^2; coef=coef+1/H(k)^2;
      end
      if j>1
        sor=sor+Cu(i,j-1)/H(k)^2; coef=coef+1/H(k)^2;
      end
      if j<NY(k)
        sor=sor+Cu(i,j+1)/H(k)^2; coef=coef+1/H(k)^2;
      end
      Cu(i,j) = sor/coef;
    end
  end
end
Cuk=Cu;
end
% Compute defect
function d = defect(Cu,f,k)
global NX NY dt H
for i=1:NX(k)
  for j=1:NY(k)
    Lap=0;
    if i>1
      Lap=Lap+(Cu(i-1,j)-Cu(i,j))/H(k)^2;
    end
    if i<NX(k)
      Lap=Lap+(Cu(i+1,j)-Cu(i,j))/H(k)^2;
    end
    if j>1
      Lap=Lap+(Cu(i,j-1)-Cu(i,j))/H(k)^2;
    end
    if j<NY(k)
      Lap=Lap+(Cu(i,j+1)-Cu(i,j))/H(k)^2;
    end
    d(i,j) = f(i,j)-Cu(i,j)/dt+Lap;
  end
end
end
```

REFERENCES

[1] T. MathWorks, Matlab. The MathWorks, Natick, MA, 2023, p. 176.

[2] J. Kim, S. Kwak, H. G. Lee, Y. Hwang, and S. Ham, "A maximum principle of the Fourier spectral method for the diffusion equation," Electron. Res. Arch. **31**, 5396 (2023).

[3] Y. Li, K. Qin, Q. Xia, and J. Kim, "A second-order unconditionally stable method for the anisotropic dendritic crystal growth model with an orientation-field," Appl. Numer. Math. **184**, 512–526 (2023).

[4] Q. Xia, G. Sun, Q. Yu, J. Kim, and Y. Li, "Thermal-fluid topology optimization with unconditional energy stability and second-order accuracy via phase-field model," Commun. Nonlinear Sci. Numer. Simul. **116**, 106782 (2023).

[5]J. Yang, Y. Li, and J. Kim, "Modified multi-phase diffuse-interface model for compound droplets in contact with solid," J. Comput. Phys. **491**, 112345 (2023).
[6]W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial* (SIAM, 2000).
[7]S. R. Fulton, P. E. Ciesielski, and W. H. Schubert, "Multigrid methods for elliptic problems: A review," Mon. Weather Rev. **114**(5), 943–959 (1986).
[8]M. Griebel, T. Dornseifer, and T. Neunhoeffer, *Numerical Simulation in Fluid Dynamics: A Practical Introduction* (SIAM, 1998).
[9]D. Lee, "Gradient-descent-like scheme for the Allen–Cahn equation," AIP Adv. **13**(8), 085010 (2023).
[10]W. A. Mulder, "Numerical methods, multigrid," in *Encyclopedia of Solid Earth Geophysics* (Springer, 2020), Living Edition, pp. 1–6.
[11]K. U. Rehman, W. Shatanawi, and Z. Mustafa, "Levenberg–Marquardt back-propagation neural networking (LMB-NN) analysis of hydrodynamic forces in fluid flow over multiple cylinders," AIP Adv. **14**(2), 025051 (2024).
[12]U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid* (Academic Press, New York, 2001), p. 316.
[13]T. Ma, L. Zhang, F. Cao, and Y. Ge, "A special multigrid strategy on non-uniform grids for solving 3D convection–diffusion problems with boundary/interior layers," Symmetry **13**(7), 1123 (2021).
[14]Y. Wang and Y. Ge, "High-order compact difference scheme and multigrid method for solving the 2D elliptic problems," Math. Probl. Eng. **2018**, 1.
[15]L. Wu and X. Feng, "A high-order compact (HOC) implicit difference scheme and a multigrid method for solving 3D unsteady reaction diffusion equations," Mathematics **7**(12), 1208 (2019).
[16]E. Deriaz, "High-order Adaptive Mesh Refinement multigrid Poisson solver in any dimension," J. Comput. Phys. **480**, 112012 (2023).
[17]J. Schmitt, S. Kuckuk, and H. Köstler, "EvoStencils: A grammar-based genetic programming approach for constructing efficient geometric multigrid methods," Genet. Program. Evolvable Mach. **22**, 511–537 (2021).
[18]C. Schwarz, "Geometric multigrid for the gyrokinetic Poisson equation from fusion plasma applications," Ph.D. Dissertation (Universität Erlangen-Nrnberg, 2021).
[19]T. Guillet and R. Teyssier, "A simple multigrid scheme for solving the Poisson equation with arbitrary domain boundaries," J. Comput. Phys. **230**(12), 4756–4771 (2011).
[20]M. M. Gupta and J. Zhang, "High accuracy multigrid solution of the 3D convection–diffusion equation," Appl. Math. Comput. **113**(2–3), 249–274 (2000).
[21]H. K. I. Al-Mahdawi, M. Abotaleb, H. Alkattan, A. M. Z. Tareq, A. Badr, and A. Kadi, "Multigrid method for solving inverse problems for heat equation," Mathematics **10**(15), 2802 (2022).
[22]H. K. Al-Mahdawi, A. I. Sidikova, H. Alkattan, M. Abotaleb, A. Kadi, and E. S. M. El-Kenawy, "Parallel multigrid method for solving inverse problems," MethodsX **9**, 101887 (2022).
[23]R. L. Haupt and S. E. Haupt, "Introduction to multigrid using MATLAB," Comput. Appl. Eng. Educ. **1**(5), 421–432 (1993).
[24]L. Zheng, H. Zhang, T. Gerya, M. Knepley, D. A. Yuen, and Y. Shi, "Implementation of a multigrid solver on a GPU for Stokes equations with strongly variable viscosity based on Matlab and CUDA," Int. J. High Perform. Comput. Appl. **28**(1), 50–60 (2014).
[25]A. P. Padhi, S. Chakraborty, A. Chakrabarti, and R. Chowdhury, "Efficient hybrid topology optimization using GPU and homogenization-based multigrid approach," Eng. Comput. **39**, 3593–3615 (2022).
[26]N. O. Rojas, A. Zuñiga, and P. C. Encina, "Pattern formation via cell–cell adhesion and contact inhibition of locomotion in active matter," AIP Adv. **13**(2), 025149 (2023).
[27]W. Zhao and Q. Guan, "Numerical analysis of energy stable weak Galerkin schemes for the Cahn–Hilliard equation," Commun. Nonlinear Sci. Numer. Simul. **118**, 106999 (2023).
[28]C. Lee, S. Kim, S. Kwak, Y. Hwang, S. Ham, S. Kang, and J. Kim, "Semi-automatic fingerprint image restoration algorithm using a partial differential equation," AIMS Math. **8**, 27528 (2023).
[29]Y. Hwang, S. Ham, C. Lee, G. Lee, S. Kang, and J. Kim, "A simple and efficient numerical method for the Allen–Cahn equation on effective symmetric triangular meshes," Electron. Res. Arch. **31**(8), 4557–4578 (2023).
[30]J. Kim, Z. Tan, and J. Yang, "Linear and conservative IMEX Runge–Kutta finite difference schemes with provable energy stability for the Cahn–Hilliard model in arbitrary domains," Comput. Math. Appl. **143**, 133 (2023).
[31]M. Hashemi, S. Shalbaf, M. Jadidi, and A. Dolatabadi, "Effects of gas viscosity and liquid-to-gas density ratio on liquid jet atomization in crossflow," AIP Adv. **13**(3), 035105 (2023).
[32]J. Yang, Z. Tan, J. Wang, and J. Kim, "Modified diffuse interface fluid model and its consistent energy-stable computation in arbitrary domains," J. Comput. Phys. **488**, 112216 (2023).