Chapter 28
Finite Difference Method

Introduction: Numerical Analysis

Numerical analysis is about the use of numerical approximations as opposed to symbolic manipulations for the solution of mathematical problems. This approach originated from the hand calculations with large printed tables. After the birth of computers, automated calculations with computer programs have been promoting this type of analysis into a major approach in scientific studies and engineering applications. The overall goal of numerical analysis is to develop and apply algorithms to provide approximate but acceptable solutions to problems which are difficult to solve with analytical techniques.

Numerical analyses can be categorized into direct and iterative methods based on the control of calculation steps. Direct methods compute the solution to a problem within a finite number of steps, while iterative methods do not necessarily terminate in a finite number of steps. However, for practical purpose, analyses with iterative methods are usually terminated when the residual is smaller than a predefined allowable error.

The field of numerical analysis can be divided into multiple areas depending on the purpose of the analysis. The evaluation of the function value at a given point is one of the simplest applications of numerical analysis. Interpolation and extrapolation aim to find the value of an unknown function at a point with given values of that function in the ranges including and excluding that point, respectively. Regression is similar to interpolation; however, it requires finding out the mathematical formulation of a function instead of function values at given points. Another significant application is to compute the solution to a given equation. Optimization is implemented to search for the point at which a given function is maximized or minimized. Evaluating integrals numerically, which is called numerical integration or numerical quadrature, is the numerical analysis for calculating the value of a definite integral.

Z. Liu, *Multiphysics in Porous Materials*,
https://doi.org/10.1007/978-3-319-93028-2_28

Solving differential equation, especially partial differential equations, is another significant area of numerical analysis, as we know that many physical processes, including most of the monolithic and multiphysical processes introduced in the previous chapters of this book, can be described using partial differential equations. Therefore, solving these equations with initial and boundary conditions and material properties from real examples is like simulating or numerically reproducing the real physical processes. As a result, the area of numerical analysis is also termed numerical simulation. This is especially true when the solution process is enhanced with the computer-aided model establishment and graphical user interface-based postprocessing. These tools make numerical simulation appear as a virtual experiment carried out using computers. Notwithstanding the fancy and vague appearance, we need to bear in mind that those numerical experiments are implemented based on physical and mathematical laws underlying the partial differential equations. In fact, numerical simulation also contains pieces from other numerical analysis areas. For example, numerical integrals are usually needed for the finite element method.

This chapter will introduce one of the most straightforward numerical simulation methods: the finite difference method. We will show how to approximate derivatives using finite differences and discretize the equation and computational domain based on that. The discretization will be discussed for spatial and temporal derivatives sequentially. Then the treatment of the boundary and initial conditions will be explained with an example. A brief introduction to the error analysis in the finite difference method will also be provided. Most of the above introductions will be made in 2D. Therefore, the extension from 2D to 3D will be discussed to bridge the knowledge gap. In the final, MATLAB code for implementing the example in the boundary and initial condition section will be attached.

Following this chapter will be the introduction to the finite volume method, which is similar to the finite difference method in many aspects. The finite element method as a slightly more complicated and more popular type of numerical simulation method will be introduced finally.

Spatial Finite Difference and Derivative

One essential idea behind numerical simulation is discretization. For the purpose, we need to transform a continuous mathematical equation(s) into an algebraic equation. Accordingly, the computational domain will be discretized into a mesh or a grid which consists of multiple subdomains called cells or elements. The size of the matrices in the algebraic equation is determined by the mesh size, the dimensions of the computational domain, and the number of unknowns. Therefore, the discretization of the equations and that of the domains is correlated. Different ways or schemes can be used to approximate the same derivative, leading to different levels of accuracy. In addition, discretization of spatial derivatives and temporal derivatives is also different due to the difference between the spatial and temporal dimensions. For example, for a spatial dimension, we always know the conditions on

the "start" end and need to march toward the "final" end, while for spatial dimensions, boundary conditions could be given on any ends (boundary segments) or even both. This leads to the different terms and ways used for these two types of discretization. The following two sections will be devoted to the finite differences for spatial and temporal dimensions sequentially.

Spatial Discretization

A finite difference is a sequence of the differences between function values as the independent variable changes with finite increments (Fig. 28.1). It usually has the mathematical expression of the form $f(x + h) - f(x)$. A really useful operation is to divide the finite difference by an increment in the independent variable, i.e., h, so that we can obtain the following quotient

$$D(u) = \frac{f(x+h) - f(x)}{h}, \tag{28.1}$$

where $D(u)$ is the finite difference approximation to the derivative.

The idea of the finite difference method is to use the above quotient, $D(u)$, to approach the gradient (derivative) at point x, point $x + h$, or any point between them. The geometric analogy of this approximation is to use a secant line to approximate a tangent line. When the interval gets infinitely small, the error of the above approximation will become infinitely small as well. Therefore, the derivative of a function f at a point x can be defined using the following limit:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{28.2}$$

If h has a non-zero value instead of approaching zero, then the above approximation can be written in a general way as

$$f'(x) = f'(x) + E(h^n), \tag{28.3}$$

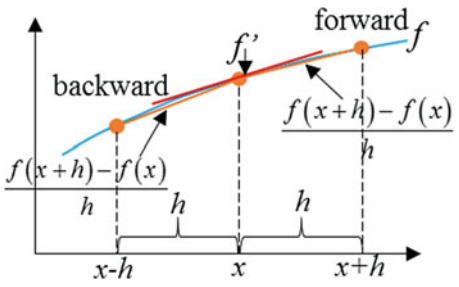

Fig. 28.1 Finite differences as approximations to derivatives

where E is the truncation error, which is a function of the nth power of the increment h. Usually, there are many ways to construct $D(u)$, and these different ways are called schemes in the context of the finite difference method. Equation 28.3 indicates that the scheme has an nth order accuracy. Schemes of different orders of accuracy can also be constructed for higher-order derivatives, which can be formulated in a way similar to Eq. 28.3.

For the first-order derivative, the most common schemes are forward, backward, and central difference schemes. A forward scheme has an expression of the form:

$$f'(x) = \frac{f(x+h) - f(x)}{h}. \tag{28.4}$$

It is called "forward" because we use the function value at a point in front of (along the direction of the corresponding axis) the point at which the derivative is calculated. Likewise, a backward scheme uses the function values at x and $x - h$ instead:

$$f'(x) = \frac{f(x) - f(x-h)}{h}. \tag{28.5}$$

We can also approximate the derivative using the function values at points on the two sides of the point of derivative as:

$$f'(x) = \frac{f(x+h/2) - f(x-h/2)}{h}. \tag{28.6}$$

A second-order derivative can be approximated using first-order derivatives as we can view the second-order derivative as a derivative of first-order derivatives. For example, in order to obtain the second-order derivative at the point x, we can use the derivatives at the points $x - h/2$ and $x + h/2$:

$$f''(x) = \frac{f'(x+h/2) - f'(x-h/2)}{h}. \tag{28.7}$$

The above approximation uses the central difference. We can also use central differences for the first derivative in the above equation, leading to:

$$\begin{aligned} f''(x) &= \frac{\dfrac{f(x+h) - f(x)}{h} - \dfrac{f(x) - f(x-h)}{h}}{h} \\ &= \frac{f(x-h) - 2f(x) + f(x+h)}{h} \end{aligned}. \tag{28.8}$$

For a derivative of any given order, we can construct different difference schemes by employing the function values at points around the point of interest using different weights. The general form of finite difference schemes can be formulated as

$$D(u) = \sum_{i=-m}^{n} a_i(x + ih), \tag{28.9}$$

where *m*and *n* are positive integers. These two integers are the numbers of points before and after the point where the derivative is calculated. The coefficients (weights) a_i and the corresponding values at the chosen points differentiate different schemes. The sum of the coefficients needs to be zero. The finite difference schemes introduced above are the most common ones and can be used to handle many common applications. However, in a few cases, we still want to employ more complicated schemes. This is especially necessary for problems with special features such as high nonlinearity or when there is a specific requirement for the solution such as accuracy. To construct more complicated schemes, we can resort to methods such as the method of undetermined coefficients and polynomial approximation to calculate the coefficients for any designed schemes. However, details for these methods will be excluded here considering that their use is less common in entry-level finite difference applications.

Temporal Finite Difference and Schemes

The finite difference in the time coordinate usually requires a different treatment from that in the spatial coordinates due to the different features of time and space. In the spatial finite difference context, forward and backward methods are usually adopted; by contrast, in the temporal context, we talk more about explicit and implicit methods. To differentiate the finite differences in space and time, subscripts will be used for spatial finite differences, while superscripts will be reserved for the temporal ones. An example is given in this subsection to introduce temporal finite differences where the spatial finite differences are also included for a complete introduction. Let us first reformulate the general transient partial differential equation in the following way

$$u_t = f(u), \tag{28.10}$$

where $f(u)$ in the equation contains all the terms except the temporal derivative term. First, for the temporal difference, let us use the following most common form as our purpose is to calculate the function value at the next time step $n + 1$ based on the known function value at the current time step n

$$\frac{u^{n+1} - u^n}{k} = f(u), \tag{28.11}$$

where k is the time step. An Euler forward scheme is used for the temporal derivative. The word “Euler” differentiates the temporal schemes from the spatial schemes.

The choice of u, i.e., u from which time step, determines the type of the temporal scheme. If all the unknown values are from the current step, then the scheme is explicit:

$$\frac{u^{n+1} - u^n}{k} = f(u^n). \tag{28.12}$$

The following solution can be readily obtained for this scheme:

$$u^{n+1} = u^n + k \cdot f(u^n). \tag{28.13}$$

We can see that an explicit form of the solution can be obtained conveniently. This provides convenience for the development of computer programs. Despite the advantage, the explicit scheme needs to satisfy conditions for the adopted time steps, i.e., k is small enough, to ensure stability. Otherwise, the numerical oscillation in the values of the dependent variables between steps will increase and eventually fail the solution process.

If $f(u)$ depends only on the dependent variables at the next step, then we will have the fully implicit scheme:

$$\frac{u^{n+1} - u^n}{k} = f\left(u^{n+1}\right). \tag{28.14}$$

An explicit mathematical function for the solution to the above equation is not always available. Even if one is available, more effort is usually required to derive the equation and to develop programs based on it. Despite the disadvantage, this fully implicit scheme is unconditionally stable – there is no requirement on the time step k for stability.

The above two types of schemes can also be mixed to construct other schemes

$$\frac{u^{n+1} - u^n}{k} = (1 - \theta) f(u^n) + \theta f\left(u^{n+1}\right), \tag{28.15}$$

where θ is the ratio. θ values of 0 and 1 lead to explicit and fully implicit schemes, respectively, while those values between 0 and 1 produce partially implicit ones. Another common implicit scheme, the Newton-Nicolson scheme will be obtained when θ equals 0.5. Partially implicit schemes are also unconditionally stable.

To gain a better idea of these schemes, let us revisit a simple parabolic PDE which is common for time-dependent processes such as a transient heat transfer process:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}. \tag{28.16}$$

There is only one spatial dimension and one temporal dimension. We discretize the space and time using a mesh, $x_1,\ldots,x_i,\ldots,x_I$, and a "time mesh," $t^0,\ldots,t^n,\ldots,t^N$. A uniform mesh size h and a constant time step k are adopted. Then the function value at any point is:

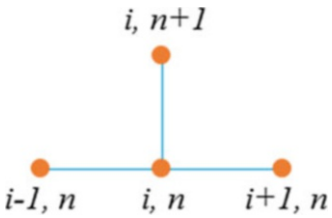

Fig. 28.2 Explicit method stencil

$$u(x_i, t^n) = u_i^n. \tag{28.17}$$

We can discretize the governing equation with the Euler forward scheme for time and the central difference scheme for space. The operation will lead to the following discretized form of the governing equation:

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}. \tag{28.18}$$

The solution to the above equation is

$$u_i^{n+1} = u_i^n + r\left(u_{i-1}^n + u_i^n + u_{i+1}^n\right), \tag{28.19}$$

where $r = k/h^2$.

The stencils to illustrate this method are plotted in Fig. 28.2. As can be seen, the solution at one point at the new (next) step needs the information at this point and its two neighboring points from the current time step.

For the fully implicit method, the discretization will be as follows:

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}. \tag{28.20}$$

The solution is:

$$-ru_{i-1}^{n+1} + (1 + 2r)u_i^{n+1} - ru_{i+1}^{n+1} = u_i^n. \tag{28.21}$$

The stencils for the full implicit method are shown in Fig. 28.3. Therefore, the function values at one point and its two neighboring points at the new step rely on the function value at the point from the current step.

Then a discretization of the above equation in the general form is

$$\frac{u_i^{n+1} - u_i^n}{k} = (1 - \theta)\frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2} + \theta\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}. \tag{28.22}$$

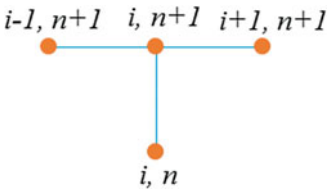

Fig. 28.3 Implicit method stencil

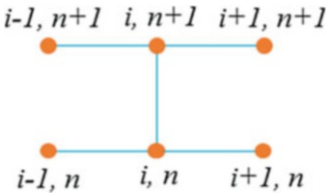

Fig. 28.4 Crank-Nicolson stencil

The solution to the above equation is

$$-\theta r u_{i-1}^{n+1} + (1+2\theta r)u_i^{n+1} - \theta r u_{i+1}^{n+1} = (1-\theta) r u_{i-1}^{n} + [1-2(1-\theta)r]u_i^{n}. + (1-\theta) r u_{i+1}^{n} \tag{28.23}$$

Substituting $\theta = 1/2$ into the above equation will lead to the solution for the Newton-Nicolson method:

$$-\frac{1}{2} r u_{i-1}^{n+1} + (1+r)u_i^{n+1} - \frac{1}{2} r u_{i+1}^{n+1} = \frac{1}{2} r u_{i-1}^{n} + (1-r)u_i^{n} + \frac{1}{2} r u_{i+1}^{n}. \tag{28.24}$$

The relationship between the function values at different steps is shown in Fig. 28.4. The function values at one point and its two neighboring points at the next step rely on the function values at these three points at the current step.

The above introduction is made based on the application of the Euler forward scheme to the temporal derivative on the left-hand side of the equation. In fact, it is also common to use in the temporal derivatives one-step multistage schemes and multistep schemes, which will not be discussed in this book.

Boundary and Initial Conditions

The discretization will convert the mathematical description that is defined in a continuous geometric domain (and possibly time) into an algebraic equation defined at discrete mesh points (and possibly time points) in the domain (and time period). However, the spatial and temporal discretizations alone are not enough to generate such an algebraic equation. Shown in the following are the general forms of the

algebraic equations for equilibrium problems, i.e., boundary value problem (BVP), and transient problems, i.e., initial-boundary value problem (IBVP), respectively

$$\text{BVP} : \mathbf{AU} = \mathbf{F}, \tag{28.25}$$

$$\text{IBVP} : \mathbf{A}^{n+1}\mathbf{U}^{n+1} = \mathbf{A}^{n}\mathbf{U}^{n} + \mathbf{F}, \tag{28.26}$$

where $\mathbf{A}$ is the stiffness matrix; $\mathbf{U}$is the array of unknowns, which are the values of the function at the grid points; $\mathbf{F}$ is the force matrix; the superscripts are the time steps: n is the current time step; and $n + 1$ is the next time step.

Boundary and initial conditions are needed to achieve the above algebraic equation. For boundary conditions, the Dirichlet boundary condition can be applied directly. For example, if the discretized domain starts with Point 1, then function values from neighboring points are needed to calculate the derivatives at this starting point. The Dirichlet boundary condition can then be assigned to "Point 0." "Point 0" is not necessarily explicitly considered in the algebraic equation, and thus its derivative does not need to be represented using finite differences. The application of the Dirichlet boundary condition $u = a$ at the left boundary just requires replacing all the function value at "Point 0," i.e., u_0, with a. The application of the Neumann boundary condition $u^{'} = \alpha$ on the left boundary (1D) involves the substitution of the following equation into the discretization equation of relevant points, e.g., Point 1, to cancel out u_0:

$$\frac{u_1 - u_0}{h} = \alpha. \tag{28.27}$$

The boundary conditions can be applied to other boundaries such as the right boundary in a 1D domain in a similar way.

The treatment of the initial condition is straightforward. We just need to find out the initial values at the grid points according to the initial condition,$\mathbf{U}^0$. Then the solution of the transient problem will involve an iterative process starting from $\mathbf{U}^0$. For example, the solution in the first step is obtained as

$$\mathbf{A}^1\mathbf{U}^1 = \mathbf{A}^0\mathbf{U}^0 + \mathbf{F}, \tag{28.28}$$

where $\mathbf{A}^0$ is calculated with $\mathbf{U}^0$.

To better demonstrate the discretization including boundary and initial conditions, let us recall the general example used in the previous section. A complete description of the problem is as follows:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \tag{28.29}$$

$$u|_{x=0} = a, \tag{28.30}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=L} = \beta, \tag{28.31}$$

$$u|_{t=0} = u^0. \tag{28.32}$$

Let us discretize the domain with a length of L into I points, for which we have $I = \frac{L}{h} + 1$. The physical process to be simulated lasts a time period of Nk. Then the discretization starts with the temporal and spatial discretization at each point:

$$\begin{aligned}
\text{Point1} &: \frac{u_1^{n+1} - u_1^n}{k} = \frac{u_0^n - 2u_1^n + u_2^n}{h^2} \\
\text{Point2} &: \frac{u_2^{n+1} - u_2^n}{k} = \frac{u_1^n - 2u_2^n + u_3^n}{h^2} \\
&\cdots \\
\text{Point}\,i &: \frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2} \\
&\cdots \\
\text{Point}\,I &: \frac{u_I^{n+1} - u_I^n}{k} = \frac{u_{I-1}^n - 2u_I^n + u_{I+1}^n}{h^2}
\end{aligned}$$

If we incorporate the above boundary conditions, we will get:

$$\begin{aligned}
\text{Point1} &: \frac{u_1^{n+1} - u_1^n}{k} = \frac{a - 2u_1^n + u_2^n}{h^2} \\
\text{Point2} &: \frac{u_2^{n+1} - u_2^n}{k} = \frac{u_1^n - 2u_2^n + u_3^n}{h^2} \\
&\cdots \\
\text{Point}\,i &: \frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2} \\
&\cdots \\
\text{Point}\,I &: \frac{u_I^{n+1} - u_I^n}{k} = \frac{u_{I-1}^n - u_I^n + \beta h}{h^2}
\end{aligned}$$

The above series of equations can be represented using the following algebraic equation:

$$\begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_i^{n+1} \\ \vdots \\ u_I^{n+1} \end{bmatrix} = \left(\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} + \frac{k}{h^2} \begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & & \ddots & & & \\ & & 1 & -2 & 1 & \\ & & & & \ddots & \\ & & & & 1 & -1 \end{bmatrix} \right) \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_i^n \\ \vdots \\ u_I^n \end{bmatrix}$$

$$+\frac{k}{h^2}\begin{bmatrix} a \\ 0 \\ 0 \\ \vdots \\ 0 \\ \beta h \end{bmatrix}. \tag{28.33}$$

The initial condition is applied in the first step of the iteration when $n = 1$. The discretized function value for the first step is:

$$U^0 = \begin{bmatrix} u^0 \\ u^0 \\ \vdots \\ u^0 \\ \vdots \\ u^0 \end{bmatrix}. \tag{28.34}$$

Error Analysis

Error analysis is an essential part of numerical simulation methods. But it is also worth pointing out that the error analysis is much less emphasized in engineering applications than in the studies of numerical simulation methods in mathematics. In many engineering applications, the error is examined by comparing the calculated results against analytical or experimental data instead of directly studying the error caused by the numerical schemes. This difference is especially obvious in the finite difference method, where the error associated with numerical schemes has been more extensively studied.

Because the algebraic equation is just an approximation, the numerical solution $\mathbf{U}$ at the spatial grid points and time steps is different from the true solution $\widehat{\mathbf{U}}$. Therefore, in general, $\widehat{\mathbf{U}}$ will not satisfy the above algebraic equation exactly. The discrepancy is the local (truncation) error:

$$\mathbf{A}\widehat{\mathbf{U}} = \mathbf{F} + \boldsymbol{\tau}, \tag{28.35}$$

where $\boldsymbol{\tau}$ is the local error. $\boldsymbol{\tau}$ is an array which has the same size as $\mathbf{U}$ (or $\widehat{\mathbf{U}}$). Therefore, $\boldsymbol{\tau}$ is the ensemble of the errors at each spatial and temporal discretization grid points. The true error is the global error, which is the difference between the numerical solution and the true solution

$$\mathbf{E} = \mathbf{U} - \widehat{\mathbf{U}}. \tag{28.36}$$

The local error and the global error are related via the following equation, provided that $\mathbf{A}$ is invertible:

$$\mathbf{E} = -\mathbf{A}^{-1}\boldsymbol{\tau}. \tag{28.37}$$

The above equation implies that

$$\|\mathbf{E}\| = \left\|-\mathbf{A}^{-1}\boldsymbol{\tau}\right\| \leq \left\|\mathbf{A}^{-1}\right\|\|\boldsymbol{\tau}\|. \tag{28.38}$$

A finite difference method is convergent if $\|\mathbf{E}\| \to 0$ as the mesh size or/and the step size approaches zero. The above three equations correspond to three types of criteria as the mesh size or/and the step size approaches zero:

Convergence: $\|\mathbf{E}\| \to 0$

Consistency: $\|\boldsymbol{\tau}\| \to 0$

Stability: $\|\mathbf{A}^{-1}\| <$ a finite constant

Therefore, the convergence can be ensured by two conditions: stability and consistency. The stability requires that $\|\mathbf{A}^{-1}\|$ is smaller than a constant as the mesh size or/and the step size approaches zero. The consistency requirement states that $\|\boldsymbol{\tau}\| \to 0$ as the mesh size or/and the step size approaches zero. The sign $\|\|$ means the norm of a matrix. For a given finite difference method, $\mathbf{A}$ is known. The stability can be assessed with the second-order norm of a matrix. The method will not be detailed here. Instead, we will investigate the consistency requirement, which is related to the local truncation error. Another reason to look into the local error is that the order of accuracy of finite difference schemes is determined by the local truncation error.

The local truncation error of finite difference schemes can be obtained via the following Taylor series

$$\begin{aligned} f(x+h) = f(x) + \frac{f'(x)}{1!}h + \frac{f^{(2)}(x)}{2!}h^2 + \ldots, \\ + \frac{f^{(n)}(x)}{n!}h^n + O\left(h^{n+1}\right) \end{aligned} \tag{28.39}$$

where $n!$ is the factorial of n and $O(h^{n+1})$ is the difference between the Taylor polynomial of degree n and the original function, which is a function of h^{n+1}. An approximation to the first derivative of the function "f" can be obtained by truncating the Taylor polynomial:

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + \ldots. \tag{28.40}$$

Similarly, we can obtain an approximation to the function at any given point around the point of interest:

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + O\left(h^2\right). \tag{28.41}$$

Based on the above Taylor expansion, we can readily obtain the error for any finite difference schemes such as the forward, backward, and central difference schemes.

Forward:

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{O(h^2)}{h} = f'(x) + O(h) \tag{28.42}$$

Backward:

$$\frac{f(x)-f(x-h)}{h} = f'(x) + O(h) \tag{28.43}$$

Central:

$$\frac{f(x+h)-f(x-h)}{h} = f'(x) + O(h^2) \tag{28.44}$$

Therefore, the forward, backward, and central difference schemes have the first- and second-order accuracy, respectively. The order of accuracy of temporal difference schemes can be obtained in a similar way.

Extension to 2D and 3D Dimensions

The extension of the above discretization to 2D and 3D cases can be implemented in different ways. The major challenge is that it is hard to deal with arrays more than two dimensions. In a 2D domain, the spatial discretization will lead to function values in the form of 2D arrays corresponding to the rows and columns of the points in the discretized 2D domain. We thus need to convert this 2D array into a 1D array, i.e., a column array, so that the stiffness matrices can be simplified from a 4D array to a 2D one.

The above extension can be implemented via both element-wise operations and matrix operations. For element-wise operations, we just need to find out the relationship between the dependent variable and the elements in the stiffness matrix. When conducting the conversion, the stiffness matrix is converted accordingly. This will be shown in the next chapter for the finite volume method. In a typical matrix operation, the conversion can be finished for a 2D domain in the following way

$$\frac{\partial^2 u}{\partial x^2} : (\mathbf{I} \otimes \mathbf{L})\mathbf{U} \tag{28.45}$$

$$\frac{\partial^2 u}{\partial y^2} : (\mathbf{L} \otimes \mathbf{I})\mathbf{U}, \tag{28.46}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{L}$ is the matrix generated by the stiffness matrix excluding any boundary conditions:

$$\mathbf{L} = \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & \vdots & & & & & \\ & & 1 & -2 & 1 & & \\ & & & & \vdots & & \\ & & & & 1 & -1 \end{bmatrix}. \tag{28.47}$$

The boundary conditions can be applied either by modifying the above **L** matrix or the stiffness matrix generated by the above operations specified in Eqs. 28.45 and 28.46.

Practice Problem

Problem Description

Develop FDM code to solve the problem described in Chapter 11. The material constituting the square area (1 m by 1 m) is water. The initial temperature is 273.15 K. The top boundary is set to 293.15 K, while the other boundaries are thermally insulated. Please calculate the temperature distribution at the end of 1 day.

Results

Figure 28.5 illustrates the temperature distribution.

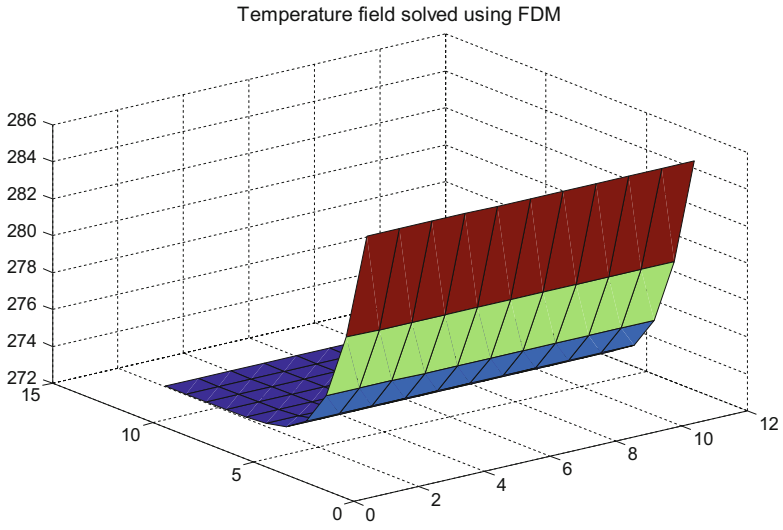

Fig. 28.5 Temperature distribution from the numerical simulation

Appendix: MATLAB Script File

```
clc;clear;
h=0.1;
k=60;
lambda=1;
rho=1000;
C=4200;
a=lambda/rho/C;
b=a*k/h^2;
m=(1/h+1);
U=273.15*ones(m^2,1);

figure(1)

Ky=[-2   1   0   0   0   0   0   0   0   0   0;...
     1  -2   1   0   0   0   0   0   0   0   0;...
     0   1  -2   1   0   0   0   0   0   0   0;...
     0   0   1  -2   1   0   0   0   0   0   0;...
     0   0   0   1  -2   1   0   0   0   0   0;...
     0   0   0   0   1  -2   1   0   0   0   0;...
     0   0   0   0   0   1  -2   1   0   0   0;...
     0   0   0   0   0   0   1  -2   1   0   0;...
     0   0   0   0   0   0   0   1  -2   1   0;...
     0   0   0   0   0   0   0   0   1  -2   1;...
     0   0   0   0   0   0   0   0   0   1  -1];

Kx=[-1   1   0   0   0   0   0   0   0   0   0;...
     1  -2   1   0   0   0   0   0   0   0   0;...
     0   1  -2   1   0   0   0   0   0   0   0;...
     0   0   1  -2   1   0   0   0   0   0   0;...
     0   0   0   1  -2   1   0   0   0   0   0;...
     0   0   0   0   1  -2   1   0   0   0   0;...
     0   0   0   0   0   1  -2   1   0   0   0;...
     0   0   0   0   0   0   1  -2   1   0   0;...
     0   0   0   0   0   0   0   1  -2   1   0;...
     0   0   0   0   0   0   0   0   1  -2   1;...
     0   0   0   0   0   0   0   0   0   1  -1];

F=[293.15;0;0;0;0;0;0;0;0;0;0]*ones(1,m);
%F=[293*ones(m,1);zeros(m^2-2*m,1);273*ones(m,1)]
F=F(:);

K_2D=kron(eye(m),Ky)+kron(Kx,eye(m));

for i=1:60*24
    t=i*k
    U=U+b*K_2D*U+b*F;
    U=reshape(U,m,m);
```

```
    surf(U)
    U=reshape(U,m^2,1);
    drawnow;
    title('Temperature field solved using FDM')
end
```