5
Code verification

In scientific computing, the goal of code verification is to ensure that the code is a faithful representation of the underlying mathematical model. This mathematical model generally takes the form of partial differential or integral equations along with associated initial condition, boundary conditions, and auxiliary relationships. Code verification thus addresses both the correctness of the chosen numerical algorithm and the correctness of the instantiation of that algorithm into written source code, i.e., ensuring there are no coding mistakes or "bugs."

A computer program, referred to here simply as a code, is a collection of instructions for a computer written in a programming language. As discussed in Chapter 4, in the software engineering community code verification is called software verification and is comprised of software tests which ensure that the software meets the stated requirements. When conducting system-level testing of non-scientific software, in many cases it is possible to exactly determine the correct code output for a set of given code inputs. However, in scientific computing, the code output depends on the numerical algorithm, the spatial mesh, the time step, the iterative tolerance, and the number of digits of precision used in the computations. Due to these factors, it is not possible to know the correct code output (i.e., numerical solution) *a priori*. The developer of scientific computing software is thus faced with the difficult challenge of determining appropriate system-level software tests.

This chapter discusses various procedures for verifying scientific computing codes. Although a formal proof of the "correctness" of a complex scientific computing code is probably not possible (Roache, 1998), code testing using the order verification procedures discussed in this chapter can provide a high degree of confidence that the code will produce the correct solution. An integral part of these procedures is the use of systematic mesh and time step refinement. For rigorous code verification, an exact solution to the underlying governing equations (i.e., the mathematical model) is required. We defer the difficult issue of how to obtain these exact solutions until Chapter 6, and for now simply assume that an exact solution to the mathematical model is available. Finally, unless otherwise noted, the code verification procedures discussed in this chapter do not depend on the discretization approach, thus we will defer our discussion of the different discretization methods (finite difference, finite volume, finite element, etc.) until Chapter 8.

5.1 Code verification criteria

Before choosing a criterion for code verification, one must first select the code outputs to be tested. The first code outputs to be considered are the dependent variables in the mathematical model. For all but the simplest code verification criteria, we will compare the solution to a reference solution, ideally an exact solution to the mathematical model. In this case, we can convert the difference between the code output and the reference solution over the entire domain into a single, scalar error measure using a norm.

If a continuous representation of the numerical solution is available (e.g., from the finite element method), then a continuous norm can be used. For example, the L_1 norm of the solution error over the domain is given by

$$\|u - u_{\text{ref}}\|_1 = \frac{1}{\Omega} \int_{\Omega} |u - u_{\text{ref}}| \, \mathrm{d}\omega, \tag{5.1}$$

where u is the numerical solution, u_{ref} the reference solution, and Ω is the domain of interest. The L_1 norm is the most appropriate norm to use when discontinuities or singularities exist in the solution (Rider, 2009). If instead a discrete representation of the numerical solution is available (e.g., from a finite difference or finite volume method), then a discrete norm of the error can be used. The discrete L_1 norm provides a measure of the average absolute error over the domain and can be defined as

$$\|u - u_{\text{ref}}\|_1 = \frac{1}{\Omega} \sum_{n=1}^{N} \omega_n \left| u_n - u_{\text{ref},n} \right|, \tag{5.2}$$

where the subscript n refers to a summation over all N cells of size ω_n in both space and time. Note that for uniform meshes (i.e., those with constant cell spacing in all directions), the cell sizes cancel resulting in simply:

$$\|u - u_{\text{ref}}\|_1 = \frac{1}{N} \sum_{n=1}^{N} \left| u_n - u_{\text{ref},n} \right|. \tag{5.3}$$

Another commonly used norm for evaluating the discretization error is the L_2 (or Euclidean) norm, which effectively provides the root mean square of the error. For a uniform mesh, the discrete L_2 norm is given by

$$\|u - u_{\text{ref}}\|_2 = \left(\frac{1}{N} \sum_{n=1}^{N} \left| u_n - u_{\text{ref},n} \right|^2 \right)^{1/2}. \tag{5.4}$$

The max (or infinity) norm returns the maximum absolute error over the entire domain, and is generally the most sensitive measure of error:

$$\|u - u_{\text{ref}}\|_\infty = \max \left| u_n - u_{\text{ref},n} \right|, \quad n = 1 \text{ to } N. \tag{5.5}$$

In addition to the dependent variables, one should also examine any system response quantities that may be of interest to the code user. These quantities can take the form of derivatives (e.g., local heat flux, local material stress), integrals (e.g., drag on an object, net heat flux through a surface), or other functionals of the solution variables (e.g., natural frequency, maximum deflection, maximum temperature). All system response quantities that may potentially be of interest to the user should be included as part of the code verification process to ensure that both the dependent variables and the procedures for obtaining the system response quantities are verified. For example, a numerical solution for the dependent variables may be verified, but if the subsequent numerical integration used for a system response quantity contains a mistake, then incorrect values of that quantity will be produced.

There are a number of different criteria that can be used for verifying a scientific computing code. In order of increasing rigor, these criteria are:

1 simple tests,
2 code-to-code comparisons,
3 discretization error quantification,
4 convergence tests, and
5 order-of-accuracy tests.

The first two, simple tests and code-to-code comparisons, are the least rigorous but have the advantage that they can be performed for cases where an exact solution to the mathematical model is not available. The remaining criteria require that an exact solution to the mathematical model be available, or at the very least a demonstrably accurate surrogate solution. These five criteria are discussed in more detail below.

5.1.1 Simple tests

The following simple tests, while not a replacement for rigorous code verification studies, can be used as part of the code verification process. They have the advantage that they can be applied even when an exact solution to the mathematical model is not available since they are applied directly to the numerical solution.

5.1.1.1 Symmetry tests

In most cases, when a code is provided with a symmetric geometry, initial conditions, and boundary conditions, it will produce a symmetric solution. In some cases, physical instabilities can lead to solutions that are asymmetric at any given point in time, but may still be symmetric in a statistical sense. One example is the laminar, viscous flow past a circular cylinder, which will be symmetric for Reynolds numbers below 40, but will generate a von Karman vortex street at higher Reynolds numbers (Panton, 2005). Note that this test should not be used near a bifurcation point (i.e., near a set of conditions where the solution can rapidly change its basic character).

5.1.1.2 Conservation tests

In many scientific computing applications, the mathematical model will be based on the conservation of certain properties such as mass, momentum, and energy. In a discrete sense, different numerical approaches will handle conservation differently. In the finite-difference method, conservation is only assured in the limit as the mesh and/or time step are refined. In the finite element method, conservation is strictly enforced over the global domain boundaries, but locally only in the limiting sense. For finite volume discretizations, conservation is explicitly enforced at each cell face, and thus this approach should satisfy the conservation requirement even on very coarse meshes. An example conservation test for steady-state heat conduction is to ensure that the net energy flux into the domain minus the net energy flux out of the domain equals zero, either within round-off error (finite element and finite volume methods) or in the limit as the mesh is refined (finite difference method). See Chapter 8 for a more detailed discussion of the differences between these discretization approaches.

5.1.1.3 Galilean invariance tests

Most scientific computing disciplines have their foundations in Newtonian (or classical) mechanics. As such, solutions to both the mathematical model and the discrete equations should obey the principle of Galilean invariance, which states that the laws of physics are valid for all inertial reference frames. Inertial reference frames are allowed to undergo linear translation, but not acceleration or rotation. Two common Galilean invariance tests are to allow the coordinate system to move at a fixed linear velocity or to simply exchange the direction of the coordinate axes (e.g., instead of having a 2-D cantilevered beam extend in the x-direction and deflect in the y-direction, have it extend in the y-direction and deflect in the x-direction). In addition, for structured grid codes that employ a global transformation from physical space (x, y, z) to computational space (ξ, η, ζ), certain mistakes in the global mesh transformations can be found by simply re-running a problem with the computational coordinates reoriented in different directions; again this procedure should have no effect on the final numerical solution.

5.1.2 Code-to-code comparisons

Code-to-code comparisons are among the most common approaches used to assess code correctness. A *code-to-code comparison* occurs when the output (numerical solution or system response quantity) from one code is compared to the output from another code. Following Trucano *et al*. (2003), code-to-code comparisons are only useful when (1) the two codes employ the same mathematical models and (2) the "reference" code has undergone rigorous code verification assessment or some other acceptable type of code verification. Even when these two conditions are met, code-to-code comparisons should be used with caution.

If the same models are not used in the two codes, then differences in the code output could be due to model differences and not coding mistakes. Likewise, agreement between the two codes could occur due to the serendipitous cancellation of errors due to coding mistakes and differences due to the model. A common mistake made while performing code-to-code comparisons with codes that employ different numerical schemes (i.e., discrete equations) is to assume that the codes should produce the same (or very similar) output for the same problem with the same spatial mesh and/or time step. On the contrary, the code outputs will only be the same if exactly the same algorithm is employed, and even subtle algorithm differences can produce different outputs for the same mesh and time step.

For the case where the reference code has not itself been verified, agreement between the two codes *does not* imply correctness for either code. The fortuitous agreement (i.e., a false positive for the test) can occur due to the same algorithm deficiency being present in both codes. Even when the above two requirements have been met, "code comparisons do not provide substantive evidence that software is functioning correctly" (Trucano *et al.*, 2003). Thus code-to-code comparisons should not be used as a substitute for rigorous code verification assessments. See Trucano *et al.* (2003) for a more detailed discussion of the proper usage of code-to-code comparisons.

5.1.3 Discretization error evaluation

Discretization error evaluation is the traditional method for code verification that can be used when an exact solution to the mathematical model is available. This test involves the quantitative assessment of the error between the numerical solution (i.e., the code output) and an exact solution to the mathematical model using a single mesh and/or time step. The main drawback to this test is that once the discretization error has been evaluated, it requires a subjective judgment of whether or not the error is sufficiently small. See Chapter 6 for an extensive discussion of methods for obtaining exact solutions to the mathematical model.

5.1.4 Convergence tests

A *convergence test* is performed to assess whether the error in the discrete solution relative to the exact solution to the mathematical model (i.e., the discretization error) reduces as the mesh and time step are refined. (The formal definition of convergence will be given in Section 5.2.3.) As was the case for discretization error evaluation, the convergence test also requires an exact solution to the mathematical model. However, in this case, it is not just the magnitude of the discretization error that is assessed, but whether or not that error reduces with increasing mesh and time step refinement. The convergence test is the minimum criterion that should be used for rigorous code verification.

5.1.5 Order-of-accuracy tests

The most rigorous code verification criterion is the *order-of-accuracy test*, which examines not only the convergence of the numerical solution, but also whether or not the discretization error is reduced at the theoretical rate as the mesh and/or time step are refined. This theoretical rate is called the *formal order of accuracy* and it is usually found by performing a truncation error analysis of the numerical scheme (see Section 5.3.1). The actual rate at which the discretization error is reduced is called the *observed order of accuracy*, and its calculation requires two systematically refined meshes and/or time steps when the exact solution to the mathematical model is available. The procedures for computing the observed order of accuracy in such cases are presented in Section 5.3.2.

The order-of-accuracy test is the most difficult test to satisfy; therefore it is the most rigorous of the code verification criteria. It is extremely sensitive to even small mistakes in the code and deficiencies in the numerical algorithm. The order-of-accuracy test is the most reliable code verification criterion for finding coding mistakes and algorithm deficiencies which affect the order of accuracy of the computed solutions. Such order-of-accuracy problems can arise from many common coding mistakes including implementation of boundary conditions, transformations, operator splitting, etc. For these reasons, the order-of-accuracy test is the recommended criterion for code verification.

5.2 Definitions

The definitions presented in this section follow the standard definitions from the numerical analysis of partial differential and integral equations (e.g., Richtmyer and Morton, 1967). A firm grasp of these definitions is needed before moving on to the concepts behind the formal and observed order of accuracy used in the order-verification procedures.

5.2.1 Truncation error

The *truncation error* is the difference between the discretized equations and the original partial differential (or integral) equations. It is *not* the difference between a real number and its finite representation for storage in computer memory; this "digit truncation" is called *round-off error* and is discussed in Chapter 7. Truncation error necessarily occurs whenever a mathematical model is approximated by a discretization method. The form of the truncation error can usually be found by performing Taylor series expansions of the dependent variables and then inserting these expansions into the discrete equations. Recall the general Taylor series representation for the smooth function $u(x)$ expanded about the point x_0:

$$u(x) = u(x_0) + \left.\frac{\partial u}{\partial x}\right|_{x_0} (x - x_0) + \left.\frac{\partial^2 u}{\partial x^2}\right|_{x_0} \frac{(x - x_0)^2}{2} + \left.\frac{\partial^3 u}{\partial x^3}\right|_{x_0} \frac{(x - x_0)^3}{6} + O\left[(x - x_0)^4\right],$$

where the $O[(x - x_0)^4]$ term denotes that the leading term that is omitted is on the order of $(x - x_0)$ to the fourth power. This expansion can be represented more compactly as:

$$u(x) = \sum_{k=0}^{\infty} \left.\frac{\partial^k u}{\partial x^k}\right|_{x_0} \frac{(x - x_0)^k}{k!}.$$

5.2.1.1 Example: truncation error analysis

For a simple example of truncation error analysis, consider the following mathematical model for the 1-D unsteady heat equation:

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0, \tag{5.6}$$

where the first term is the unsteady contribution and the second term represents thermal diffusion with a constant diffusivity α. Let $L(T)$ represent this partial differential operator and let $\tilde{T}$ be the exact solution to this mathematical model assuming appropriate initial and boundary conditions. Thus we have

$$L(\tilde{T}) = 0. \tag{5.7}$$

For completeness, this mathematical model operator $L(\cdot)$ should be formulated as a vector containing the partial differential equation along with appropriate initial and boundary conditions. For simplicity, we will omit the initial and boundary conditions from the following discussion.

Equation (5.6) can be discretized with a finite difference method using a forward difference in time and a centered second difference in space, resulting in the simple explicit numerical scheme

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} - \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} = 0, \tag{5.8}$$

where the i subscripts denote spatial location, the n superscripts denote the temporal step, Δx is the constant spatial distance between nodes, and Δt is the time step. We can represent this discrete equation compactly using the discrete operator $L_h(T)$ which is solved exactly by the numerical solution T_h, i.e., we have

$$L_h(T_h) = 0. \tag{5.9}$$

The variable h is a single parameter that is used to denote systematic mesh refinement, i.e., refinement over the entire spatial domain, in all spatial coordinate directions, and in time (for unsteady problems). For the current example, this parameter is given by

$$h = \frac{\Delta x}{\Delta x_{\text{ref}}} = \frac{\Delta t}{\Delta t_{\text{ref}}}, \tag{5.10}$$

where Δx_{ref} and Δt_{ref} refer to some arbitrary reference spatial node spacing and time step, respectively. Later, the h parameter will be extended to consider different refinement ratios in time and even in the different coordinate directions. For the current purposes, the important point is that when h goes to zero, it implies that Δx and Δt also go to zero at the same rate. Note that, for this finite difference discretization, T_h represents a vector of temperature values defined at each node and time step.

In order to find the truncation error for the numerical scheme given in Eq. (5.8), we can expand the above temperature values in a Taylor series about the temperature at spatial location i and time step n (assuming sufficient differentiability of T):

$$T_i^{n+1} = T_i^n + \left.\frac{\partial T}{\partial t}\right|_i^n \frac{\Delta t}{1!} + \left.\frac{\partial^2 T}{\partial t^2}\right|_i^n \frac{(\Delta t)^2}{2!} + \left.\frac{\partial^3 T}{\partial t^3}\right|_i^n \frac{(\Delta t)^3}{3!} + O\left(\Delta t^4\right),$$

$$T_{i+1}^n = T_i^n + \left.\frac{\partial T}{\partial x}\right|_i^n \frac{\Delta x}{1!} + \left.\frac{\partial^2 T}{\partial x^2}\right|_i^n \frac{(\Delta x)^2}{2!} + \left.\frac{\partial^3 T}{\partial x^3}\right|_i^n \frac{(\Delta x)^3}{3!} + O\left(\Delta x^4\right),$$

$$T_{i-1}^n = T_i^n + \left.\frac{\partial T}{\partial x}\right|_i^n \frac{(-\Delta x)}{1!} + \left.\frac{\partial^2 T}{\partial x^2}\right|_i^n \frac{(-\Delta x)^2}{2!} + \left.\frac{\partial^3 T}{\partial x^3}\right|_i^n \frac{(-\Delta x)^3}{3!} + O\left(\Delta x^4\right).$$

Substituting these expressions into the discrete equation and rearranging yields

$$\underbrace{\frac{T_i^{n+1} - T_i^n}{\Delta t} - \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2}}_{L_h(T)}$$
$$= \underbrace{\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2}}_{L(T)} + \underbrace{\left[\frac{1}{2}\frac{\partial^2 T}{\partial t^2}\right]\Delta t + \left[-\frac{\alpha}{12}\frac{\partial^4 T}{\partial x^4}\right](\Delta x)^2 + O\left(\Delta t^2, \Delta x^4\right)}_{\text{truncation error: } TE_h(T)}. \quad (5.11)$$

Thus we have the general relationship that the discrete equation equals the mathematical model plus the truncation error. In order for this equality to make sense, it is implied that either (1) the continuous derivatives in $L(T)$ and $TE_h(T)$ are restricted to the nodal points or (2) the discrete operator $L_h(T)$ is mapped onto a continuous space.

5.2.1.2 Generalized truncation error expression (GTEE)

Using the operator notation discussed earlier and substituting in the generic (sufficiently smooth) dependent variable u into Eq. (5.11) yields

$$L_h(u) = L(u) + TE_h(u), \quad (5.12)$$

where we again assume an appropriate mapping of the operators onto either a continuous or discrete space. We refer to Eq. (5.12) as the *generalized truncation error expression (GTEE)* and it is used extensively in this chapter as well as in Chapters 8 and 9. It relates the discrete equations to the mathematical model in a very general manner and is one of the most important equations in the evaluation, estimation, and reduction of discretization errors in scientific computing. When set to zero, the right hand side of the GTEE can be thought of as the actual mathematical model that is solved by the discretization scheme $L_h(u_h) = 0$. The GTEE is the starting point for determining the *consistency* and the *formal order of accuracy* of the numerical method. While the GTEE can be derived even for nonlinear mathematical models, for linear (or linearized) mathematical models, it also explicitly shows the relationship between the truncation error and the discretization error. As will be shown in Chapter 8, the GTEE can also be used to provide estimates of the truncation error. Finally, this equation provides a general relationship between the discrete equation and the (possibly nonlinear) mathematical model since we have not specified what the function u is, only that it satisfies certain differentiability constraints. It is relatively straightforward (although somewhat tedious) to show that the general polynomial function

$$u(x,t) = \sum_{i=0}^{N_x} a_i x^i + \sum_{j=0}^{N_t} b_j t^j$$

will satisfy Eq. (5.12) exactly for the example problem of 1-D unsteady heat conduction given above (hint: choose N_x and N_t small enough such that the higher-order terms are zero).

Most authors (e.g., Richtmyer and Morton, 1967; Ferziger and Peric, 2002) formally define the truncation error only when the exact solution to the mathematical model is inserted into Eq. (5.12), thus resulting in

$$L_h(\tilde{u}) = TE_h(\tilde{u}) \tag{5.13}$$

since $L(\tilde{u}) = 0$. For our specific finite difference example for the 1D unsteady heat equation given above, we have

$$\begin{aligned} &\frac{\tilde{T}_i^{n+1} - \tilde{T}_i^n}{\Delta t} - \alpha \frac{\tilde{T}_{i+1}^n - 2\tilde{T}_i^n + \tilde{T}_{i-1}^n}{(\Delta x)^2} \\ &= \left[\frac{1}{2}\frac{\partial^2 \tilde{T}}{\partial t^2}\right]\Delta t + \left[-\frac{\alpha}{12}\frac{\partial^4 \tilde{T}}{\partial x^4}\right](\Delta x)^2 + O\left(\Delta t^2, \Delta x^4\right) = TE_h\left(\tilde{T}\right), \end{aligned} \tag{5.14}$$

where the notation $\tilde{T}_i^n$ implies that the exact solution is restricted to spatial location i and temporal location n. For the purposes of this book, we will employ the GTEE found from Eq. (5.12) since it will provide more flexibility in how the truncation error is used.

5.2.2 *Discretization error*

Discretization error is formally defined as the difference between the exact solution to the discrete equations and the exact solution to the mathematical model. Using our earlier notation, we can thus write the discretization error for the general dependent variable u as

$$\varepsilon_h = u_h - \tilde{u}, \tag{5.15}$$

where again the h subscript denotes the exact solution to the discrete equations and the overtilde denotes the exact solution to the mathematical model.

5.2.3 *Consistency*

For a numerical scheme to be *consistent*, the discretized equations $L_h(\cdot)$ must approach the mathematical model equations $L(\cdot)$ in the limit as the discretization parameters (Δx, Δy, Δz, Δt, denoted collectively by the parameter h) approach zero. In terms of the truncation error discussion above, a consistent numerical scheme can be defined as one in which the truncation error vanishes in the limit as $h \to 0$. Not all numerical schemes are consistent, and one notable example is the DuFort–Frankel finite difference method applied to the unsteady heat conduction equation, which has a leading truncation error term proportional to $(\Delta t/\Delta x)^2$ (Tannehill *et al.*, 1997). This scheme is only consistent under the restriction that Δt approach zero at a faster rate than Δx.

5.2.4 Stability

For initial value (i.e., hyperbolic and parabolic) problems, a discretization scheme is said to be *stable* if numerical errors do not grow unbounded in the marching direction. The numerical errors are typically considered to come from computer round-off (Ferziger and Peric, 2002), but in fact can come from any source. The idea of numerical stability originally derives from initial value problems for hyperbolic and parabolic partial differential equations (Crank and Nicolson, 1947), but the concepts can also be applied to relaxation methods for elliptic problems (e.g., see Hirsch, 2007). It is important to note that the concept of numerical stability applies only to the discrete equations (Hirsch, 2007), and should not be confused with natural instabilities that can arise in the mathematical model itself.

Most approaches for analyzing numerical stability apply only to linear partial differential equations with constant coefficients. The most popular approach for determining stability is von Neumann stability analysis (Hirsch, 2007). Also referred to as Fourier stability analysis, von Neumann's method employs a Fourier decomposition of the numerical error and neglects the boundary conditions by assuming these error components are periodic. The fact that von Neumann's method neglects the boundary conditions is not overly restrictive in practice and results in a fairly straightforward stability analysis (e.g., see Richtmyer and Morton, 1967; Hirsch, 2007). However, the restriction to linear differential equations with constant coefficients is significant. Time and time again, we will find that many of our tools for analyzing the behavior of numerical schemes are only applicable to linear equations. We are thus left in the uncomfortable situation of hoping that we can simply extend the results of these methods to the complicated, nonlinear mathematical models of interest, and this fact should not be forgotten. As a practical matter when dealing with nonlinear problems, a stability analysis should be performed for the linearized problem to provide initial guidance on the stability limits; then numerical tests should be performed to confirm the stability restrictions for the nonlinear problem.

5.2.5 Convergence

Convergence addresses whether or not the exact solution to the discrete equations approaches the exact solution to the mathematical model in the limit of decreasing mesh spacing and time step size. Whereas convergence and consistency both address the limiting behavior of the discrete method relative to the mathematical model, convergence deals with the solution while consistency deals with the equations. This definition of convergence should not be confused with convergence of an iterative method (see Chapter 7), which we will refer to as iterative convergence.

For marching problems, convergence is determined by Lax's equivalence theorem, which is again valid only for linear equations. Lax's theorem states that, given a well-posed initial value problem and a consistent numerical scheme, stability is the necessary and sufficient condition for convergence (Richtmyer and Morton, 1967). When used for code verification purposes, convergence is demonstrated by examining the actual behavior of the discretization error ε_h as $h \to 0$.

Recent work with finite volume methods (Despres, 2004) suggests that some modifications to (or perhaps clarifications of) Lax's equivalence theorem may be needed. In his work, Despres claims that the finite volume method is formally inconsistent for 2-D triangular meshes, but found it to be convergent assuming certain solution regularity constraints. It is interesting to note that while Despres does provide theoretical developments, numerical examples are not included. Given the work of Despres (2004) and the references cited therein, it is possible that Lax's theorem should be augmented with the additional assumption of systematic mesh refinement (discussed in Section 5.4) along with mesh topology restrictions. Additional work is required to understand these mesh quality and topology issues as they relate to the consistency and convergence of discretization schemes.

5.3 Order of accuracy

The term *order of accuracy* refers to the rate at which the discrete solution approaches the exact solution to the mathematical model in the limit as the discretization parameters go to zero. The order of accuracy can be addressed in either a theoretical sense (i.e., the order of accuracy of a given numerical scheme assuming it has been implemented correctly) or in a more empirical manner (i.e., the actual order of accuracy of discrete solutions). The former is called the *formal order of accuracy*, while the latter is the *observed order of accuracy*. These two terms are discussed in detail below.

5.3.1 Formal order of accuracy

The *formal order of accuracy* is the theoretical rate of convergence of the discrete solution u_h to the exact solution to the mathematical model $\tilde{u}$. This theoretical rate is defined only in an asymptotic sense as the discretization parameters (Δx, Δy, Δt, etc., currently represented by the single parameter h) go to zero in a systematic manner. It will be shown next that the formal order of accuracy can be related back to the truncation error; however, we are limited to linear (or linearized) equations to show this relationship.

The key relationship between the discrete equation and the mathematical model is the GTEE given by Eq. (5.12), which is repeated here for convenience:

$$L_h(u) = L(u) + TE_h(u). \tag{5.12}$$

Inserting the exact solution to the discrete equation u_h into Eq. (5.12), then subtracting the original mathematical model equation $L(\tilde{u}) = 0$, yields

$$L(u_h) - L(\tilde{u}) + TE_h(u_h) = 0.$$

If the mathematical operator $L(\cdot)$ is linear (or linearized), then $L(u_h) - L(\tilde{u}) = L(u_h - \tilde{u})$. The difference between the discrete solution u_h and the exact solution to the mathematical model $\tilde{u}$ is simply the discretization error ε_h defined by Eq. (5.15), thus we find that the discretization error and the truncation error are related by

$$L(\varepsilon_h) = -TE_h(u_h). \tag{5.16}$$

Equation (5.16) governs the transport of the discretization error and is called the *continuous discretization error transport equation* since it employs the continuous mathematical operator (Roy, 2009). According to this equation, the discretization error is propagated in the same manner as the original solution u. For example, if the original mathematical model contains terms governing the convection and diffusion of u, then the discretization error ε_h will also be convected and diffused. More importantly in the context of our present discussion, Eq. (5.16) also shows that the truncation error serves as the local source for the discretization error (Ferziger and Peric, 2002); thus the rate of reduction of the local truncation error with mesh refinement will produce corresponding reductions in the discretization error. Applying this continuous error transport equation to the 1-D unsteady heat conduction example from Section 5.2.1 results in:

$$\frac{\partial \varepsilon_h}{\partial t} - \alpha \frac{\partial^2 \varepsilon_h}{\partial x^2} = -\left[\frac{1}{2}\frac{\partial^2 T_h}{\partial t^2}\right]\Delta t - \left[-\frac{\alpha}{12}\frac{\partial^4 T_h}{\partial x^4}\right](\Delta x)^2 + O\left(\Delta t^2, \Delta x^4\right).$$

Having tied the rate of reduction of the discretization error to the truncation error, we are now in the position to define the formal order of accuracy of a numerical scheme as the smallest exponent which acts upon a discretization parameter in the truncation error, since this will dominate the limiting behavior as $h \to 0$. For problems in space and time, it is sometimes helpful to refer to the formal order of accuracy in time separately from the formal order of accuracy in space. For the 1-D unsteady heat conduction example above, the simple explicit finite difference discretization is formally first-order accurate in time and second-order accurate in space. While it is not uncommon to use different order-of-accuracy discretizations for different terms in the equations (e.g., third-order convection and second-order diffusion), the formal order of accuracy of such a mixed-order scheme is simply equal to the lowest order accurate discretization employed.

The truncation error can usually be derived for even complicated, nonlinear discretization methods (e.g., see Grinstein *et al.*, 2007). For cases where the formal order of accuracy has not been determined from a truncation error analysis, there are three approaches that can be used to estimate the formal order of accuracy (note that Knupp (2009) refers to this as the "expected" order of accuracy). The first approach is to approximate the truncation error by inserting the exact solution to the mathematical model into the discrete equations.

Since the exact solution to the mathematical model will not satisfy the discrete equation, the remainder (i.e., the discrete residual) will approximate the truncation error as shown in Eq. (5.13). By evaluating the discrete residual on successively finer meshes (i.e., as $h \to 0$), the rate of reduction of the truncation error can be estimated, thus producing the formal order of accuracy of the discretization scheme (assuming no coding mistakes are present). This first approach is called the residual method and is discussed in detail in Section 5.5.6.1. The second approach is to compute the observed order of accuracy for a series of meshes as $h \to 0$, and this approach is addressed in the next section. In this case, if the observed order of accuracy is found to be two, then one is safe to assume that the formal order of accuracy is *at least* second order. The final approach is to simply assume the expected order of accuracy from the quadrature employed in the discretization. For example, a linear basis function used with the finite element method generally results in a second-order accurate scheme for the dependent variables, as does linear interpolation/extrapolation when used to determine the interfacial fluxes in the finite volume method (a process sometimes referred to as flux quadrature).

In the development of the truncation error, a certain degree of solution smoothness was assumed. As such, the formal order of accuracy can be reduced in the presence of discontinuities and singularities in the solution. For example, the observed order of accuracy for inviscid gas dynamics problems containing shock waves has been shown to reduce to first order for a wide range of numerical discretization approaches (e.g., Engquist and Sjogreen, 1998; Carpenter and Casper, 1999; Roy, 2003; Banks *et al.*, 2008), regardless of the formal order of accuracy of the scheme on smooth problems. Furthermore, for linear discontinuities (e.g., contact discontinuities and slip lines in inviscid gas dynamics), the formal order generally reduces to $p/(p+1)$ (i.e., below one) for methods which have a formal order p for smooth problems (Banks *et al.*, 2008). In some situations, the formal order of accuracy of a numerical method may be difficult to determine since it depends on the nature and strength of the discontinuities/singularities present in the solution.

5.3.2 Observed order of accuracy

As discussed above, the *observed order of accuracy* is the actual order of accuracy obtained on a series of systematically-refined meshes. For now, we only consider the case where the exact solution to the mathematical model is known. For this case, the discretization error can be evaluated exactly (or at least within round-off and iterative error) and only two mesh levels are required to compute the observed order of accuracy. The more difficult case of computing the observed order of accuracy when the exact solution to the mathematical model is not known is deferred to Chapter 8.

Consider a series expansion of the solution to the discrete equations u_h in terms of the mesh spacing h in the limit as $h \to 0$:

$$u_h = u_{h=0} + \left.\frac{\partial u}{\partial h}\right|_{h=0} h + \left.\frac{\partial^2 u}{\partial h^2}\right|_{h=0} \frac{h^2}{2} + \left.\frac{\partial^3 u}{\partial h^3}\right|_{h=0} \frac{h^3}{6} + O(h^4). \tag{5.17}$$

If a convergent numerical scheme is employed (i.e., if it is consistent and stable), then we have $u_{h=0} = \tilde{u}$. Furthermore, for a formally second-order accurate scheme, by definition we will have $\frac{\partial u}{\partial h}\big|_{h=0} = 0$ since terms of order h do not appear in the truncation error (recall Eq. (5.16)). Employing the definition of the discretization error from Eq. (5.15) we find that for a general second-order accurate numerical scheme

$$\varepsilon_h = g_2 h^2 + O(h^3), \tag{5.18}$$

where the coefficient $g_2 = g_2(x, y, z, t)$ only and is thus independent of h (Ferziger and Peric, 2002). Note that for discretizations that exclusively employ second-order accurate central-type differencing, the truncation error only contains even powers of h, and thus the higher order terms in Eq. (5.18) would be $O(h^4)$. For a more general pth-order accurate scheme, we have

$$\varepsilon_h = g_p h^p + O(h^{p+1}) \tag{5.19}$$

unless again central-type differencing is used, whereupon the higher order terms will be $O(h^{p+2})$.

Equation (5.19) provides an appropriate theoretical starting point for computing the observed order of accuracy. In the limit as $h \to 0$, the higher-order terms in Eq. (5.19) will become small relative to the leading term and can be neglected. Consider now two discrete solutions, one computed on a fine mesh with spacing h and another computed on a coarse mesh with spacing $2h$ found by eliminating every other cell or node from the fine mesh. Neglecting the higher-order terms (whether they are small or not), Eq. (5.19) can be written for the two solutions as

$$\begin{aligned} \varepsilon_{2h} &= g_p (2h)^{\hat{p}}, \\ \varepsilon_h &= g_p h^{\hat{p}}. \end{aligned}$$

Dividing the first equation by the second one, then taking the natural log, we can solve for the observed order of accuracy to give

$$\hat{p} = \frac{\ln\left(\frac{\varepsilon_{2h}}{\varepsilon_h}\right)}{\ln(2)}. \tag{5.20}$$

Here the "ˆ" is used to differentiate this *observed order of accuracy* from the formal order of accuracy of the method. The observed order can be computed regardless of whether or not the higher-order terms in Eq. (5.19) are small; however, this observed order of accuracy $\hat{p}$ will only match the formal order of accuracy when the higher-order terms are in fact small, i.e., in the limiting sense as $h \to 0$.

A more general expression for the observed order of accuracy can be found that applies to meshes that are systematically refined by an arbitrary factor. Introducing the grid refinement factor r which is defined as the ratio of coarse to fine grid mesh spacing,

$$r \equiv \frac{h_{\text{coarse}}}{h_{\text{fine}}}, \tag{5.21}$$

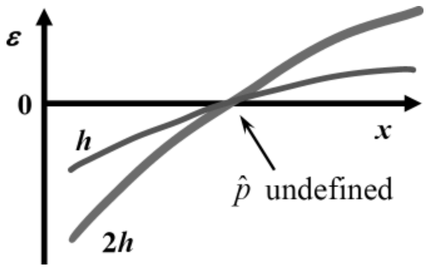

Figure 5.1 Qualitative plot of local discretization error on a coarse mesh ($2h$) and a fine mesh (h) showing that the observed order of accuracy from Eq. (5.20) can be undefined when examined locally.

where we require $r > 1$, the discretization error expansion for the two mesh levels becomes

$$\varepsilon_{rh} = g_p (rh)^{\hat{p}},$$
$$\varepsilon_h = g_p h^{\hat{p}}.$$

Again dividing the first equation by the second and taking the natural log, we find the more general expression for the observed order of accuracy

$$\hat{p} = \frac{\ln\left(\frac{\varepsilon_{rh}}{\varepsilon_h}\right)}{\ln(r)}. \tag{5.22}$$

At this point, it is important to mention that the exact solution to the discretized equations u_h is generally unknown due to the presence of round-off and iterative convergence errors in the numerical solutions. While round-off and iterative errors are discussed in detail in Chapter 7, their impact on the observed order of accuracy will be addressed in Section 5.3.2.2.

The observed order of accuracy can be evaluated using the discretization error in the dependent variables, norms of those errors, or the discretization error in any quantities that can be derived from the solution. When applied to norms of the discretization error, the relationship for the observed order of accuracy becomes

$$\hat{p} = \frac{\ln\left(\frac{\|\varepsilon_{rh}\|}{\|\varepsilon_h\|}\right)}{\ln(r)}, \tag{5.23}$$

where any of the norms discussed in Section 5.1 can be used. Care must be taken when computing the observed order of accuracy locally since unrealistic orders can be produced. For example, when the discrete solutions approach the exact solution to the mathematical model from below in some region and from above in another, the observed order of accuracy will be undefined at the crossover point. Figure 5.1 gives an example of just such a case and shows the discretization error versus a spatial coordinate x. Applying Eq. (5.22) would likely produce $\hat{p} \approx 1$ almost everywhere except at the crossover point, where if $\varepsilon_h = \varepsilon_{2h} = 0$, the observed order of accuracy from Eq. (5.22) is undefined (Potter *et al.*, 2005). For this reason, global quantities are recommended rather than local quantities for code verification purposes.

The observed order of accuracy can fail to match the nominal formal order due to mistakes in the computer code, discrete solutions which are not in the asymptotic range (i.e., when the higher-order terms in the truncation error are not small), and the presence of round-off and iterative error. The latter two issues are discussed in the following sections.

5.3.2.1 Asymptotic range

The *asymptotic range* is defined as the range of discretization sizes (Δx, Δy, Δt, etc., denoted here collectively by the parameter h) where the lowest-order terms in the truncation error and discretization error expansions dominate. It is only in the asymptotic range that the limiting behavior of these errors can be observed. For code verification purposes, the order of accuracy test will only be successful when the solutions are in this asymptotic range. In our experience, the *asymptotic range is surprisingly difficult to identify and achieve* for all but the simplest scientific computing applications. Even an experienced code user with a good intuition on the mesh resolution required to obtain a "good" solution will generally underestimate the resolution required to obtain the asymptotic range. As we shall see later in Chapter 8, all approaches for estimating discretization error also rely on the solution(s) being asymptotic.

5.3.2.2 Effects of iterative and round-off error

Recall that the underlying theory used to develop the general observed order-of-accuracy expression given in Eq. (5.22) made use of the exact solution to the discrete equation u_h. In practice, u_h is only known within some tolerance determined by the number of digits used in the computations (i.e., round-off error) and the criterion used for iterative convergence. The discrete equations can generally be iteratively solved to within machine round-off error; however, in practice, the iterative procedure is usually terminated earlier to reduce computational effort. Round-off and iterative error are discussed in detail in Chapter 7. To ensure that the computed solutions are accurate approximations of the exact solution to the discrete equations u_h, both round-off and iterative error should be at least 100 times smaller than the discretization error on the finest mesh employed (i.e., $\leq 0.01 \times \varepsilon_h$) (Roy, 2005).

5.4 Systematic mesh refinement

Up to this point, the asymptotic behavior of the truncation and discretization error (and thus the formal and observed orders of accuracy) has been addressed by the somewhat vague notion of taking the limit as the discretization parameters (Δx, Δy, Δt, etc.) go to zero. For time-dependent problems, refinement in time is straightforward since the time step is fixed over the spatial domain and can be coarsened or refined by an arbitrary factor, subject of course to stability constraints. For problems involving the discretization of a spatial domain, the refinement process is more challenging since the spatial mesh resolution and quality can vary significantly over the domain depending on the geometric complexity. In this section, we introduce the concept of systematic mesh refinement which requires uniformity of the refinement over the spatial domain and consistency of the refinement as $h \to 0$. These two requirements are discussed in detail, as well as additional issues related to the use of local and/or global mesh transformations and mesh topology.

5.4.1 Uniform mesh refinement

Local refinement of the mesh in selected regions of interest, while often useful for reducing the discretization error, is not appropriate for assessing the asymptotic behavior of discrete solutions. The reason is that the series expansion for the discretization error given in Eq. (5.17) is in terms of a single parameter h which is assumed to apply over the entire domain. When the mesh is refined locally in one region but not in another, then the refinement can no longer be described by a single parameter. This same concept holds for mesh refinement in only one coordinate direction, which requires special procedures when used for evaluating the observed order of accuracy (see Section 5.5.3) or for estimating the discretization error. The requirement that the mesh refinement be uniform is not the same as requiring that the mesh itself be uniform, only that it be refined in a uniform manner. Note that this requirement of uniform refinement is not restricted to integer refinement factors between meshes. Assuming that a coarse and fine mesh are related through *uniform refinement*, the grid refinement factor can be computed as

$$r_{12} = \left(\frac{N_1}{N_2}\right)^{1/d}, \tag{5.24}$$

where N_1 and N_2 are the number of nodes/cells/elements on the fine and coarse meshes, respectively, and d is the number of spatial dimensions.

An example of uniform and nonuniform mesh refinement is presented in Figure 5.2. The initial coarse mesh (Figure 5.2a) has 4×4 cells, and when this mesh is refined by a factor of two in each direction, the resulting uniformly refined mesh (Figure 5.2b) has 8×8 cells. The mesh shown in Figure 5.2c also has 8×8 cells, but has selectively refined to the x-axis and in the middle of the two bounding arcs. While the average cell length scale is refined by a factor of two, the local cell length scale varies over the domain, thus this mesh has not been uniformly refined.

5.4.2 Consistent mesh refinement

In general, one should not expect to obtain convergent solutions with mesh refinement when poor quality meshes are used. This is certainly true for the extreme example of a mesh with degenerate cells, such as those involving mesh crossover (see Figure 5.3), that persist with uniform refinement. We now introduce the concept of *consistent mesh refinement* which requires that mesh quality must either stay constant or improve in the limit as $h \to 0$.

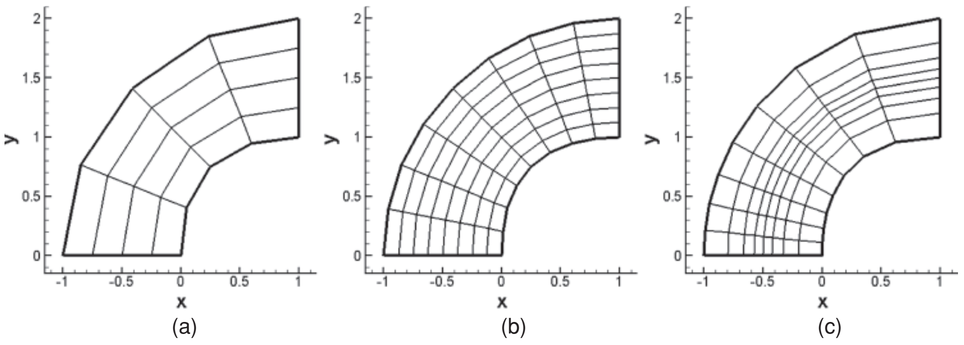

Figure 5.2 Example of uniform and nonuniform mesh refinement: (a) coarse mesh with 4×4 cells, (b) uniformly refined mesh with 8×8 cells, and (c) nonuniformly refined mesh with 8×8 cells.

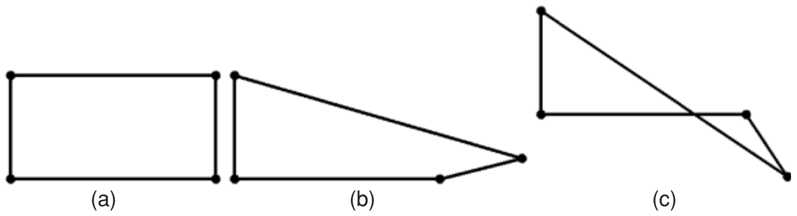

Figure 5.3 Example of a degenerate cell due to mesh crossover: (a) initial quadrilateral cell, (b) intermediate skewing of the cell, and (c) final skewing resulting in mesh crossover.

Examples of mesh quality metrics include cell aspect ratio, skewness, and stretching rate (i.e., the rate at which the mesh transitions from coarse to fine spacing).

To further illustrate the concept of consistent mesh refinement, consider the simple 2-D triangular mesh over the square domain given in Figure 5.4a. While this initial coarse mesh certainly has poor quality, it is the approach used for refining this mesh that will determine the consistency. Consider now three cases where this initial coarse mesh is uniformly refined. In the first case, the midpoints of each edge are connected so that each coarse mesh cell is decomposed into four finer cells with similar shape, as shown in Figure 5.4b. Clearly, if this refinement procedure is performed repeatedly, then in the limit as $h \to 0$ even very fine meshes will retain the same mesh qualities related to cell skewness, cell volume variation, and cell stretching (i.e., the change in cell size from one region to another). Another refinement approach might allow for different connectivity of the cells and also provide more flexibility in choosing new node locations (i.e., not necessarily at edge midpoints) as shown in Figure 5.4c. As this refinement approach is applied in the limit as $h \to 0$, the mesh quality will generally improve. A third refinement strategy might employ arbitrary edge node placement while not allowing changes in the cell-to-cell connectivity (Figure 5.4d). Considering Figure 5.4, refinement strategy (b) employs fixed quality meshes and strategy (c) employs meshes with improving quality as $h \to 0$, thus both are considered

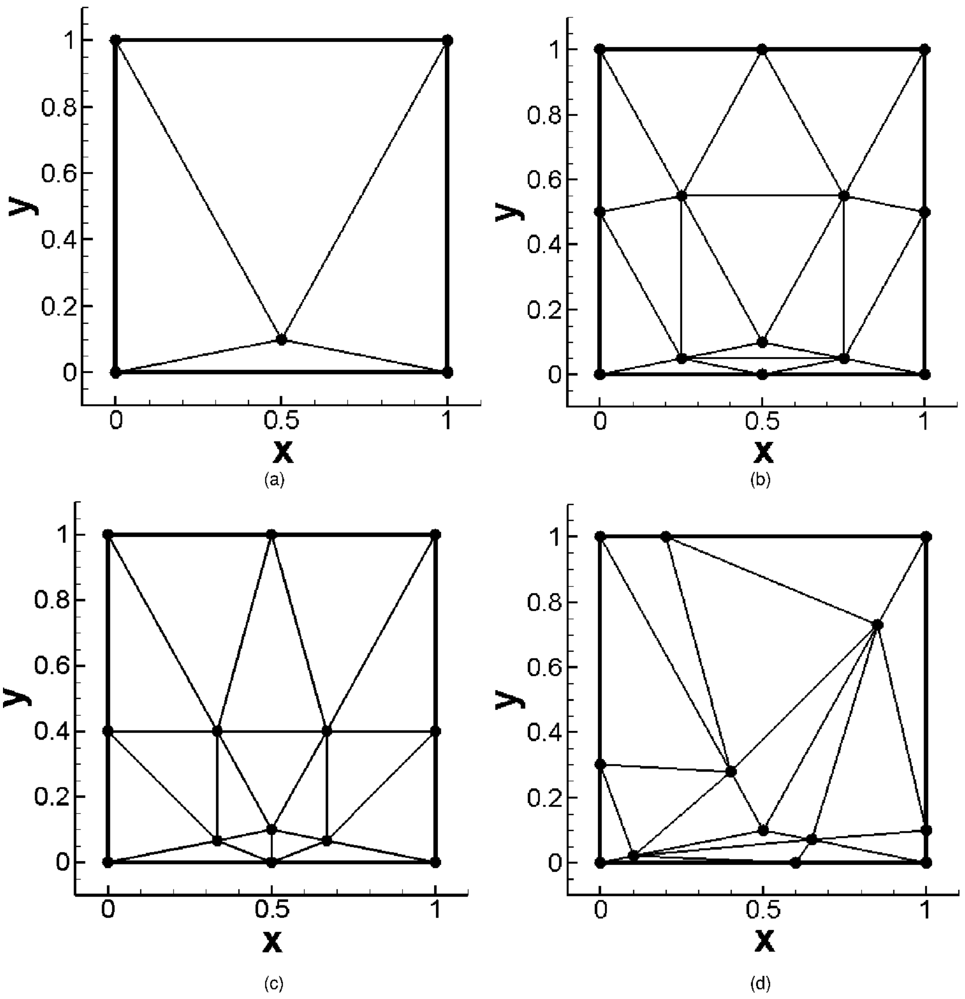

Figure 5.4 Example showing consistent and inconsistent mesh refinement: (a) poor quality coarse mesh with four unstructured triangular cells, (b) uniformly refined mesh that retains a fixed mesh quality, (c) uniformly refined mesh with improved mesh quality, and (d) uniformly refined mesh with inconsistent refinement.

consistent. Strategy (d) is an inconsistent mesh refinement approach since the quality of the mesh degrades with refinement.

Borrowing concepts from Knupp (2003), we assume the existence of a global mesh quality metric σ which varies between 0 and 1, with $\sigma = 1$ denoting an isotropic mesh, i.e., one with "ideal" mesh quality (square quadrilaterals, cubic hexahedrals, equilateral triangles, etc.). Smaller values of σ would denote anisotropic meshes with lower quality (skewness, stretching, curvature, etc). Consistent mesh refinement can thus be defined by requiring that $\sigma_{\text{fine}} \geq \sigma_{\text{coarse}}$ during refinement. Consistent refinement with $\sigma \to 1$ as $h \to 0$

can place significant burdens on the mesh generation and refinement procedure (especially for unstructured meshes), but provides the easiest criteria for discretization schemes to satisfy since meshes become more isotropic with refinement (e.g., they become Cartesian for quadrilateral and hexahedral meshes). A more difficult mesh quality requirement to satisfy from the code verification point of view is to require convergence of the numerical solutions for meshes with a fixed quality measure σ as $h \to 0$. Such nuances relating the numerical scheme behavior to the mesh quality fall under the heading of solution verification and are addressed in more detail in Chapters 8 and 9. For code verification purposes, it is important to document the asymptotic behavior of the quality of the meshes used for the code verification study.

5.4.3 Mesh transformations

Inherent in some discretization methods is an assumption that the mesh employs uniform spacing (i.e., Δx, Δy, and Δz are constant). When applied blindly to the cases with nonuniform meshes, schemes that are formally second-order accurate on uniform meshes will often reduce to first-order accuracy on nonuniform meshes. While Ferziger and Peric (1996) argue that these first-order errors will either be limited to small fractions of the domain or even vanish as the mesh is refined, this may result in extremely fine meshes to obtain the asymptotic range. The key point is that additional discretization errors will be introduced by nonuniform meshes.

In order to mitigate these additional errors, mesh transformations are sometimes used to handle complex geometries and to allow local mesh refinement. For discretization methods employing body-fitted structured (i.e., curvilinear) meshes, these transformations often take the form of global transformations of the governing equations. For finite-volume and finite-element methods on structured or unstructured meshes, these transformations usually take the form of local mesh transformations centered about each cell or element. An example of a global transformation for a body-fitted structured grid in 2-D is presented in Figure 5.5. The transformation must ensure a one-to-one mapping between the grid line intersections in physical space (a) and those in computational coordinates (b).

Consider the 2-D steady transformation from physical space (x, y) to a uniform computational space (ξ, η) given by:

$$\begin{aligned} \xi &= \xi(x, y), \\ \eta &= \eta(x, y). \end{aligned} \tag{5.25}$$

Using the chain rule, it can be shown that derivatives in physical space can be converted to derivatives in the uniform computational space (Thompson *et al.*, 1985), e.g.,

$$\frac{\partial u}{\partial x} = \frac{y_\eta}{J}\frac{\partial u}{\partial \xi} - \frac{y_\xi}{J}\frac{\partial u}{\partial \eta}, \tag{5.26}$$

where y_η and y_ξ are metrics of the transformation and J is the Jacobian of the transformation defined as $J = x_\xi y_\eta - x_\eta y_\xi$. The accuracy of the discrete approximation of the solution

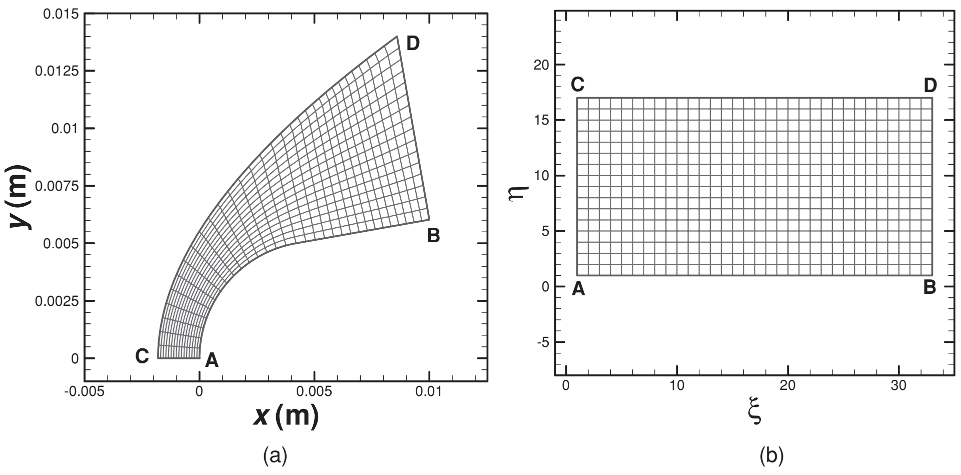

Figure 5.5 Example of a global transformation of a 2-D body-fitted structured mesh: (a) mesh in physical (x, y) coordinates and (b) mesh in the transformed computational (ξ, η) coordinates.

derivatives will depend on the chosen discretization scheme, the mesh resolution, the mesh quality, and the solution behavior (Roy, 2009).

The transformation itself can either be analytic or discrete in nature. Thompson *et al.* (1985) point out that using the same discrete approximation for the metrics that is used for solution derivatives can often result in smaller numerical errors compared to the case where purely analytic metrics are used. This surprising result occurs due to error cancellation and can be easily shown by examining the truncation error of the first derivative in one dimension (Mastin, 1999). As an example of a discrete approximation of the metrics, consider the metric term x_ξ which can be approximated using central differences to second-order accuracy as

$$x_\xi = \frac{x_{i+1} - x_{i-1}}{2\Delta\xi} + O(\Delta\xi^2).$$

When discrete transformations are used, they should be of the same order as, or possibly higher-order than, the underlying discretization scheme to ensure that the formal order of accuracy is not reduced. While mistakes in the discrete transformations can adversely impact the numerical solutions, these mistakes can be detected during the code verification process assuming sufficiently general mesh topologies are employed.

5.4.4 Mesh topology issues

There are many different mesh topologies that can be used in scientific computing. When conducting code verification studies, it is recommended that the most general mesh topology that will be employed for solving the problems of interest be used for the code verification studies. For example, if simulations will only be performed on Cartesian meshes, then

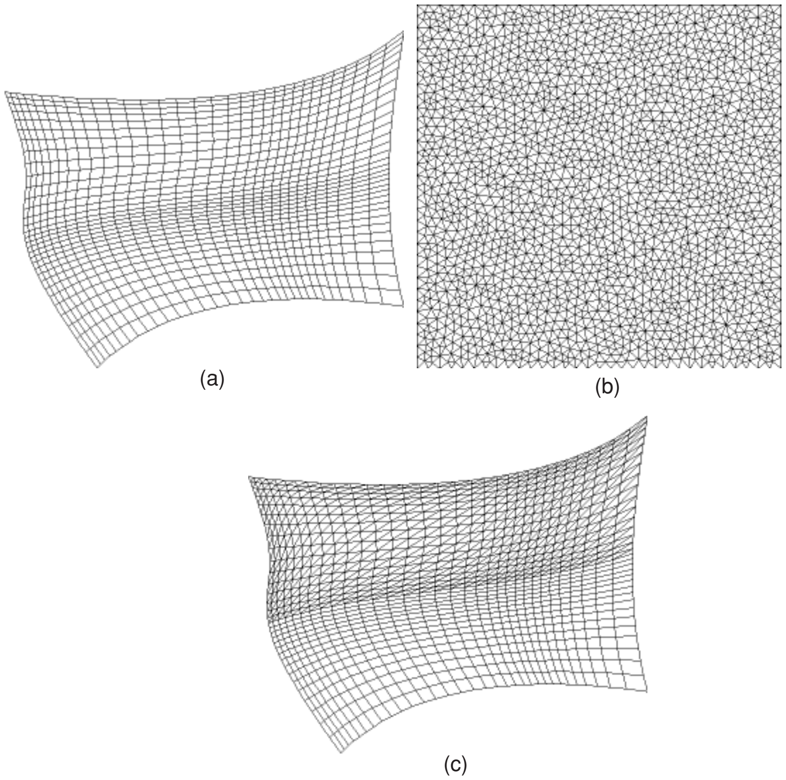

Figure 5.6 Example mesh topologies in 2-D: (a) structured curvilinear, (b) unstructured triangles, and (c) hybrid structured/unstructured curvilinear (adapted from Veluri *et al.*, 2008).

it is sufficient to conduct the code verification studies on Cartesian meshes. However, if simulations will be performed on non-isotropic (i.e., nonideal) meshes consisting of a combination of hexahedral, prismatic, and tetrahedral cells, then those mesh topologies should be used during code verification.

Meshes in 1-D consist of an order set of nodes or cells that may either be uniformly or nonuniformly distributed. For 2-D meshes, the nodes/cells may either be structured quadrilaterals, unstructured triangles, unstructured polygons with an arbitrary number of sides, or some hybrid combination of these. Examples of a hierarchy of 2-D mesh topologies appropriate for code verification are given in Figure 5.6 (Veluri *et al.*, 2008).

In 3-D, structured meshes can either be Cartesian, stretched Cartesian, or curvilinear (i.e., body fitted). 3-D unstructured meshes can contain cells that are tetrahedral (four-side pyramids), pyramidal (five-sided pyramids), prismatic (any 2-D cell type extruded in the third direction), hexahedral, polyhedral, or hybrid combinations of these. An example of a general 3-D hybrid mesh topology that has been employed for performing code verification on a scientific computing code with general unstructured mesh capabilities is given in Figure 5.7. This mesh consists of hexahedral and prismatic triangular cells extruded from

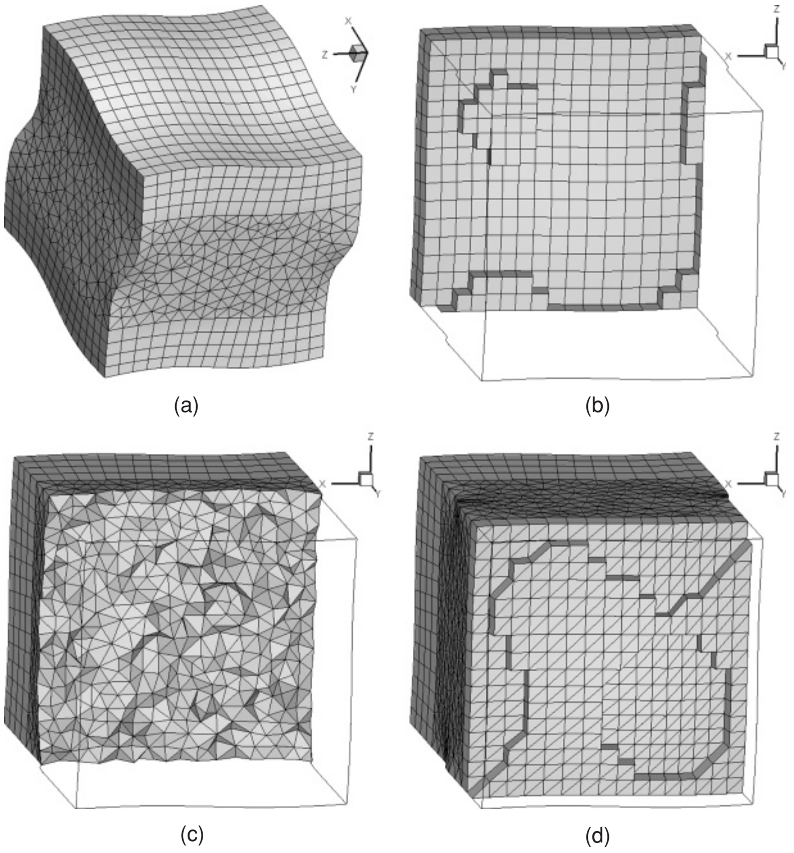

Figure 5.7 A general hybrid mesh topology in 3-D: (a) full 3-D mesh, (b) internal view showing hexahedral cells, (c) internal view showing tetrahedral cells, and (d) internal view showing prismatic cells.

the curved $y_{\min}$ and $y_{\max}$ boundaries joined together with a region of tetrahedral cells in the middle.

5.5 Order verification procedures

There are two books which deal with the subject of order-of-accuracy verification. Roache (1998) provides an overview of the subject, with emphasis on order verification using the method of manufactured solutions (discussed in Chapter 6). The book by Knupp and Salari (2003) is entirely dedicated to code order verification and is one of the most comprehensive references on the subject. Although Knupp and Salari prefer the terminology "code order verification," here we will simply use "order verification" to refer to order-of-accuracy verification of a scientific computing code. More recent reviews of order verification procedures are provided by Roy (2005) and Knupp *et al*. (2007).

Order verification entails a comparison between the limiting behavior of the observed order of accuracy and the formal order. Once an order verification test has been passed, then the code is considered verified for the code options (submodels, numerical algorithms, boundary conditions, etc.) exercised in the verification test. Any further order verification performed for those code options is simply considered confirmation of code correctness (Roache, 1998).

This section addresses the order verification procedures applicable for discretizations in space and/or time. These procedures can be invaluable for identifying the presence of coding mistakes (i.e., bugs) and problems with the numerical algorithms. Techniques are also discussed to aid in the debugging process once a coding mistake is found to exist. Limitations of the order verification procedure are then described, as well as different variants of the standard order verification procedure. This section concludes with a discussion of who bears the responsibility for code verification.

5.5.1 Spatial discretization

This section describes the order verification procedure for steady-state problems, i.e., those that do not have time as an independent variable. The order verification procedure discussed here is adapted from the procedure recommended by Knupp and Salari (2003). In brief, this procedure is used to determine whether or not the code output (numerical solution and other system response quantities) converges to the exact solution to the mathematical model at the formal rate with systematic mesh refinement. If the formal order of accuracy is observed in an asymptotic sense, then the code is considered verified for the coding options exercised. Failure to achieve the formal order of accuracy indicates the presence of a coding mistake or a problem with the numerical algorithm. The steps in the order verification procedure for steady-state problems are presented in Figure 5.8, and these steps are discussed in detail below.

1 Define mathematical model

The governing equations (i.e., the mathematical model) generally occur in partial differential or integral form and must be specified unambiguously along with any initial conditions, boundary conditions, and auxiliary equations. Small errors in defining the mathematical model can easily cause the order verification test to fail. For example, an error in the fourth significant digit of the thermal conductivity caused an order verification test to fail for a computational fluid dynamics code used to solve the Navier–Stokes equations (Roy *et al.*, 2007).

2 Choose numerical algorithm

A discretization scheme, or numerical algorithm, must be chosen. This includes both the general discretization approach (finite difference, finite volume, finite element, etc.) and the specific approaches to spatial quadrature. Discretization of any boundary or initial conditions involving spatial derivatives (e.g., Neumann-type boundary conditions) must

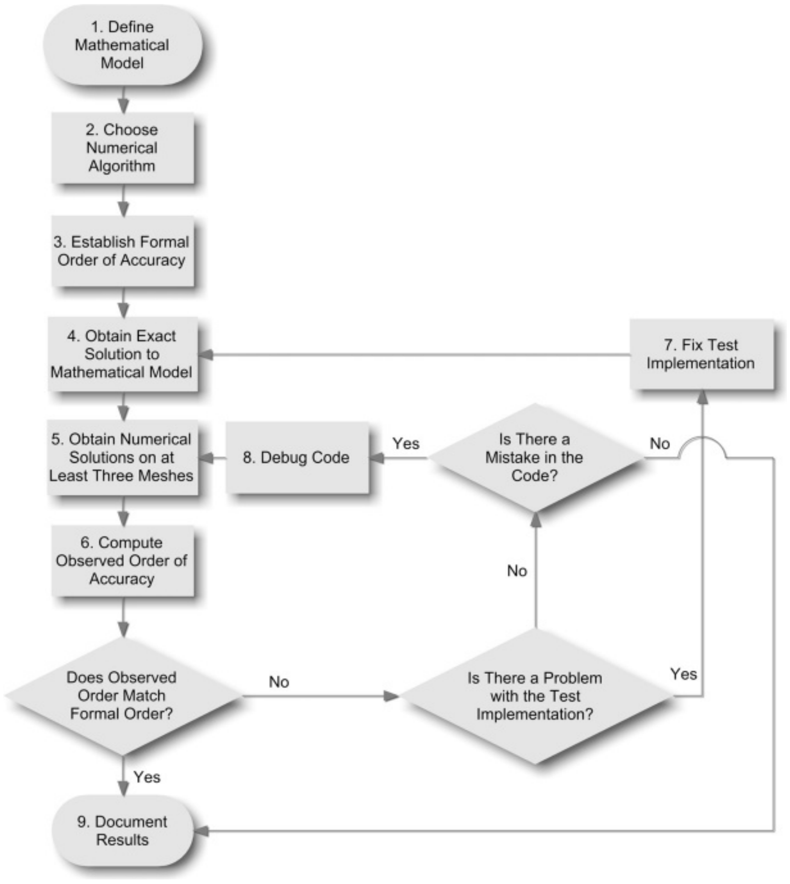
1. Define Mathematical Model
2. Choose Numerical Algorithm
3. Establish Formal Order of Accuracy
4. Obtain Exact Solution to Mathematical Model
7. Fix Test Implementation
5. Obtain Numerical Solutions on at Least Three Meshes
8. Debug Code
Yes
Is There a Mistake in the Code?
No
6. Compute Observed Order of Accuracy
No
Does Observed Order Match Formal Order?
No
Is There a Problem with the Test Implementation?
Yes
Yes
9. Document Results

Figure 5.8 Flowchart showing the order verification procedure (adapted from Knupp and Salari, 2003).

also be considered. Note that different iterative solvers can be tested by simply starting the iterations with the final solution values from an iterative solver that has been used in the verification test, thus testing alternative iterative solvers does not require additional code verification tests (Roache, 1998).

3 Establish formal order of accuracy

The formal order of accuracy of the numerical scheme should be established, ideally by an analysis of the truncation error. As discussed in Section 5.3.1, for cases where a truncation error analysis is not available, either the residual method (Section 5.5.6.1), the order verification procedure itself (i.e., the observed order of accuracy), or the expected order of accuracy can be substituted.

4 Obtain exact solution to mathematical model

The exact solution to the mathematical model must be obtained, including both the solution (i.e., the dependent variables) and the system response quantities. New exact solutions are needed any time the governing equations are changed (e.g., when a new model is examined). The same exact solution can be reused if the only changes in the code relate to the numerical scheme (e.g., a different flux function is employed). A key point is that actual numerical values for this exact solution must be computed, which may reduce the utility of series solution as they are no longer exact once the series is truncated. See Chapter 6 for a description of various methods for obtaining exact solutions to mathematical models for scientific computing applications.

5 Obtain numerical solutions on at least four meshes

While only two mesh levels are required to compute the observed order of accuracy when the exact solution to the mathematical model is known, it is strongly recommended that at least four mesh levels be used to demonstrate that the observed order of accuracy is asymptotically approaching the formal order as the mesh discretization parameters (e.g., Δx, Δy, Δz) approach zero. The mesh refinement must be performed in a systematic manner as discussed in Section 5.4. If only one grid topology is to be tested, then the most general type of grid that will be run with the code should be used.

6 Compute observed order of accuracy

With numerical values from both the numerical solution and the exact solution to the mathematical model now available, the discretization error can be evaluated. Global norms of the solution discretization error should be computed as opposed to examining local values. In addition to the error norms for the solution, discretization errors in all system response quantities of interest should also be examined. Recall that the iterative and round-off errors must be small in order to use the numerical solution as a surrogate for the exact solution to the discrete equations when computing the discretization error. For highly-refined spatial meshes and/or small time steps, the discretization error can be small and thus round-off error can adversely impact the order-of-accuracy test. The observed order of accuracy can be computed from Eq. (5.22) for the system response quantities and from Eq. (5.23) for the norms of the discretization error in the solution.

Note that only for the simplest scientific computing cases (e.g., linear elliptic problems) will the observed order of accuracy match the formal order to more than approximately two significant figures during a successful order verification test. For complex scientific computing codes, it is more common that the observed order of accuracy will approach the formal order with increasing mesh refinement. Thus, it is the asymptotic behavior of the observed order of accuracy that is of interest. In addition, observed orders of accuracy that converge to a value higher than the formal order can either indicate the presence of unforeseen error cancellation (which should not be cause for concern) or mistakes in establishing the formal order of accuracy.

If the observed order of accuracy does not match the formal order in an asymptotic sense, then one should first troubleshoot the test implementation (see Step 7) and then debug the code (Step 8) if necessary. If the observed order does match the formal order, then the verification test is considered successful, and one should proceed to Step 9 to document the test results.

7 Fix test implementation

When the observed order of accuracy does not match the formal order of accuracy, the first step is to make sure that the test was implemented correctly. Common examples of incorrect test implementations include mistakes in constructing or evaluating the exact solution to the mathematical model and mistakes made during the comparison between the numerical and exact solutions. If the test implementation is found to be flawed, then the test should be fixed and then repeated. If the test was correctly implemented, then a coding mistake or algorithm inconsistency is likely present and one should proceed to Step 8 to debug the code.

8 Debug the code

When an order-of-accuracy test fails, it indicates either a mistake in the programming of the discrete algorithm, or worse, an inconsistency in the discrete algorithm itself. See Section 5.5.4 for a discussion of approaches to aid in debugging the scientific computing code.

9 Document results

All code verification results should be documented so that a subsequent user understands the code's verification status and does not duplicate the effort. In addition to documenting the observed order of accuracy, the magnitude of the discretization error should also be reported for both system response quantities and the numerical solution (i.e., the norms of the discretization error). It is further recommended that the meshes and solutions used in the code verification test be added to one of the less-frequently run dynamic software test suites (e.g., a monthly test suite), thus allowing the order-of-accuracy verification test to be repeated on a regular basis. Coarse grid cases, which can typically be executed rapidly, can be added as system-level regression tests to a test suite that is run more frequently, as discussed in Chapter 4.

5.5.2 Temporal discretization

The order verification procedure for temporal problems with no spatial dependency is essentially the same as for spatial problems described above. The only difference is that the time step Δt is refined rather than a spatial mesh, thus mesh quality is not a concern. For unsteady problems, the temporal discretization error can sometimes be quite small.

Therefore the round-off error should be examined carefully to ensure that it does not adversely impact the observed order-of-accuracy computation.

5.5.3 Spatial and temporal discretization

It is more difficult to apply the order verification procedure to problems that involve both spatial and temporal discretization, especially for the case where the spatial order of accuracy is different from the temporal order. In addition, temporal discretization errors can in some cases be much smaller than spatial errors (especially when explicit time marching schemes are used), thus making it more difficult to verify the temporal order of accuracy.

For numerical schemes involving spatial and temporal discretization, it is helpful to rewrite the discretization error expansion given in Eq. (5.19) by separating out the spatial and temporal terms as

$$\varepsilon_{h_x}^{h_t} = g_x h_x^p + g_t h_t^q + O(h_x^{p+1}) + O(h_t^{q+1}), \tag{5.27}$$

where h_x denotes the spatial discretization (i.e., $h_x = \Delta x/\Delta x_{\text{ref}} = \Delta y/\Delta y_{\text{ref}}$, etc.), h_t the temporal discretization (i.e., $h_t = \Delta t/\Delta t_{\text{ref}}$), p is the spatial order of accuracy, and q is the temporal order of accuracy. If adaptive time-stepping algorithms are employed, the adaptive algorithm should be disabled to provide explicit control over the size of the time step. Procedures similar to those described in this section can be used for independent refinement in the spatial coordinates, e.g., by introducing $h_x = \frac{\Delta x}{\Delta x_{\text{ref}}}$ and $h_y = \frac{\Delta y}{\Delta y_{\text{ref}}}$, etc. This section will discuss different spatial and temporal order verification procedures that can be either conducted separately or in a combined manner.

5.5.3.1 Separate order analysis

The simplest approach for performing order verification on a code with both spatial and temporal discretization is to first verify the spatial discretization on a steady-state problem. Once the spatial order of accuracy has been verified, then the temporal order can be investigated separately. The temporal order verification can employ a problem with no spatial discretization such as an unsteady zero-dimensional case or a case with linear spatial variations which can generally be resolved by second-order methods to within round-off error. Alternatively, a problem can be chosen which includes spatial discretization errors, but on a highly-refined spatial mesh (Knupp and Salari, 2003). In the latter approach, the use of a highly-refined spatial mesh is often required to reduce the spatial errors to negligible levels, thus allowing the spatial discretization error term $g_x h_x^p$ term to be neglected in Eq. (5.27) relative to the temporal error term $g_t h_t^q$. In practice, this can be difficult to achieve due to stability constraints (especially for explicit methods) or the expense of computing solutions on the highly-refined spatial meshes. An alternative is to reduce the spatial dimensionality of the problem in order to allow a highly-refined spatial mesh to be used. The drawback to using separate order analysis is that it will not uncover issues related to the interaction between the spatial and temporal discretization.

Table 5.1 *Mesh levels used in the spatial and temporal code verification study of Kamm* et al. *(2003).*

Spatial step, h_x	Temporal step, h_t
Δx	Δt
$\Delta x / r_x$	Δt
$\Delta x / r_x^2$	Δt
Δx	$\Delta t / r_t$
Δx	$\Delta t / r_t^2$
$\Delta x / r_x$	$\Delta t / r_t$
$\Delta x / r_x$	$\Delta t / {r_t}^2$

5.5.3.2 Combined order analysis

A combined spatial and temporal order verification method has been developed by Kamm *et al.* (2003). Their procedure begins with a general formulation of the discretization error, which for a global system response quantity can be written in the form

$$\varepsilon_{h_x}^{h_t} = g_x h_x^{\hat{p}} + g_t h_t^{\hat{q}} + g_{xt} h_x^{\hat{r}} h_t^{\hat{s}}, \tag{5.28}$$

where the observed orders of accuracy $\hat{p}, \hat{q}, \hat{r}$, and $\hat{s}$ are to be solved for along with the three coefficients g_x, g_t, and g_{xt}. In addition, for the norms of the discretization error, a similar expansion is employed,

$$\left\| \varepsilon_{h_x}^{h_t} \right\| = g_x h_x^{\hat{p}} + g_t h_t^{\hat{q}} + g_{xt} h_x^{\hat{r}} h_t^{\hat{s}}, \tag{5.29}$$

where of course the observed orders and coefficients may be different. In order to solve for these seven unknowns, seven independent levels of spatial and/or temporal refinement are required. Beginning with an initial mesh with Δx and Δt Kamm *et al.* (2003) alternately refined in space (by r_x) and time (by r_t) to obtain the seven mesh levels shown in Table 5.1. By computing these seven different numerical solutions, the discretization error expressions for all seven mesh levels result in a coupled, nonlinear set of algebraic equations. The authors solved this nonlinear system of equations using a Newton-type iterative procedure. An advantage of this approach is that it does not require that the three terms in the discretization error expression be the same order of magnitude. The main drawback is the computational expense since a single calculation of the observed orders of accuracy requires seven different numerical solutions. Additional solutions should also be computed to ensure the asymptotic behavior of the observed order of accuracy has been achieved.

Kamm *et al.* (2003) included the mixed spatial/temporal term because their application employed the explicit Lax–Wendroff temporal integration scheme combined with a Godunov-type spatial discretization, which has formal orders of accuracy of $p = 2$, $q = 2$,

Table 5.2 *Mesh levels recommended for the simpler error expansion of Eqs. (5.30) and (5.31) which omit the mixed spatial/ temporal term.*

Spatial step, h_x	Temporal step, h_t
Δx	Δt
$\Delta x/r_x$	Δt
Δx	$\Delta t/r_t$
$\Delta x/r_x$	$\Delta t/r_t$

and $r = s = 1$ (i.e., it is formally second-order accurate). For many spatial/temporal discretization approaches, the mixed spatial/temporal term can be omitted because it does not appear in the truncation error, thereby reducing the unknowns and the required number of independent mesh levels down to four. The resulting error expansion for global system response quantities becomes

$$\varepsilon_{h_x}^{h_t} = g_x h_x^{\hat{p}} + g_t h_t^{\hat{q}}, \tag{5.30}$$

while the expansion for discretization error norms becomes

$$\left\| \varepsilon_{h_x}^{h_t} \right\| = g_x h_x^{\hat{p}} + g_t h_t^{\hat{q}}. \tag{5.31}$$

Although not unique, recommended mesh levels to employ for the simpler error expansions given by Eqs. (5.30) and (5.31) are given in Table 5.2. The resulting four nonlinear algebraic equations have no closed form solution and thus must be solved numerically (e.g., using Newton's method) for the orders of accuracy $\hat{p}$ and $\hat{q}$ and the coefficients g_x and g_t.

An alternative based on the discretization error expansions of Eqs. (5.30) and (5.31) that does not require the solution to a system of nonlinear algebraic equations can be summarized briefly as follows. First, a spatial mesh refinement study using three meshes is performed with a fixed time step to obtain $\hat{p}$ and g_x. Then a temporal refinement study is performed using three different time steps to obtain $\hat{q}$ and g_t. Once these four unknowns have been estimated, the spatial step size h_x and the temporal step size h_t can be chosen such that the spatial discretization error term $(g_x h_x^p)$ has the same order of magnitude as the temporal error term $(g_t h_t^q)$. Once these two terms are approximately the same order of magnitude, a combined spatial and temporal order verification is conducted by choosing the temporal refinement factor such that the temporal error term drops by the same factor as the spatial term with refinement. This procedure is explained in detail below.

In order to estimate $\hat{p}$ and g_x, a spatial mesh refinement study is performed with a fixed time step. Note that this will introduce a fixed temporal discretization error (i.e., $g_t h_t^q$) for all computations, thus the standard observed order-of-accuracy relationship from Eq. (5.22) cannot be used. Considering only the discretization error norms for now (the same analysis will also apply to the discretization error of the system response quantities), Eq. (5.31) can be rewritten as

$$\left\|\varepsilon_{h_x}^{h_t}\right\| = \phi + g_x h_x^{\hat{p}}, \tag{5.32}$$

where $\phi = g_t h_t^{\hat{q}}$ is the fixed temporal error term. For this case, the observed order-of-accuracy expression from Chapter 8 given by Eq. (8.70) can be used, which requires three mesh solutions, e.g., coarse ($r_x^2 h_x$), medium ($r_x h_x$), and fine (h_x):

$$\hat{p} = \frac{\ln\left(\dfrac{\left\|\varepsilon_{r_x^2 h_x}^{h_t}\right\| - \left\|\varepsilon_{r_x h_x}^{h_t}\right\|}{\left\|\varepsilon_{r_x h_x}^{h_t}\right\| - \left\|\varepsilon_{h_x}^{h_t}\right\|}\right)}{\ln\left(r_x\right)} \tag{5.33}$$

By calculating $\hat{p}$ in this manner, the constant temporal error term ϕ will cancel out. The spatial discretization error coefficient g_x can then be found from

$$g_x = \frac{\left\|\varepsilon_{r_x h_x}^{h_t}\right\| - \left\|\varepsilon_{h_x}^{h_t}\right\|}{h_x^{\hat{p}}\left(r_x^{\hat{p}} - 1\right)}. \tag{5.34}$$

A similar analysis is then performed by refining the time step with a fixed spatial mesh to obtain $\hat{q}$ and g_t. Once these orders of accuracy and coefficients have been estimated, then the leading spatial and temporal discretization error terms can be adjusted to approximately the same order of magnitude by coarsening or refining in time and/or space (subject to numerical stability restrictions). If the discretization error terms are of drastically different magnitude, then extremely refined meshes and/or time steps may be needed to detect coding mistakes.

At this point, if the formal spatial and temporal orders of accuracy are the same (i.e., if $p = q$), then the standard order verification procedure with only two mesh levels can be applied using Eqs. (5.22) or (5.23) since $r_x = r_t$. For the more complicated case when $p \neq q$, temporal refinement can be conducted by choosing the temporal refinement factor r_t according to Eq. (5.35) following Richards (1997):

$$r_t = \left(r_x\right)^{p/q}. \tag{5.35}$$

This choice for r_t will ensure that the spatial and temporal discretization error terms are reduced by the same factor with refinement when the solutions are in the asymptotic range. Some recommended values of r_t for $r_x = 2$ are given in Table 5.3 for various formal spatial and temporal orders of accuracy. The observed orders of accuracy in space and time are then computed using two mesh levels according to

$$\hat{p} = \frac{\ln\left(\dfrac{\left\|\varepsilon_{r_x h_x}^{r_t h_t}\right\|}{\left\|\varepsilon_{h_x}^{h_t}\right\|}\right)}{\ln\left(r_x\right)} \quad \text{and} \quad \hat{q} = \frac{\ln\left(\dfrac{\left\|\varepsilon_{r_x h_x}^{r_t h_t}\right\|}{\left\|\varepsilon_{h_x}^{h_t}\right\|}\right)}{\ln\left(r_t\right)}. \tag{5.36}$$

Table 5.3 *Temporal refinement factors required to conduct combined spatial and temporal order verification using only two numerical solutions.*

Spatial order, p	Temporal order, q	Spatial refinement factor, r_x	Temporal refinement factor, r_t	Expected error reduction ratio (coarse/fine)
1	1	2	2	2
1	2	2	$\sqrt{2}$	2
1	3	2	$\sqrt[3]{2}$	2
1	4	2	$\sqrt[4]{2}$	2
2	1	2	4	4
2	2	2	2	4
2	3	2	$\sqrt[3]{4}$	4
2	4	2	$\sqrt[4]{4}$	4
3	1	2	8	8
3	2	2	$\sqrt{8}$	8
3	3	2	2	8
3	4	2	$\sqrt[4]{8}$	8

The analysis using system response quantities from Eq. (5.30) is exactly the same as given above in Eq. (5.36), only with the norm notation omitted.

5.5.4 Recommendations for debugging

Order verification provides a highly-sensitive test as to whether there are mistakes in the computer code and/or inconsistencies in the discrete algorithm. In addition, aspects of the order verification procedure can be extremely useful for tracking down the mistakes (i.e., debugging the code) once they have been found to exist. Once an order verification test fails, and assuming the test was properly implemented, the local variation of discretization error in the domain should be examined. Accumulation of error near a boundary or a corner cell generally indicates that the errors are in the boundary conditions. Errors in regions of mild grid clustering or skewness can indicate mistakes in the mesh transformations or spatial quadratures. Mesh quality problems could be due to the use of a poor quality mesh, or worse, due to a discrete algorithm that is overly-sensitive to mesh irregularities. The effects of mesh quality on the discretization error are examined in detail in Chapter 9.

5.5.5 Limitations of order verification

A significant limitation of the order verification process is that the formal order of accuracy can change due to the level of smoothness of the solution, as discussed in Section 5.3.1. In addition, since order verification can only detect problems in the solution itself, it generally cannot be used to detect coding mistakes affecting the efficiency of the code. For example, a mistake that causes an iterative scheme to converge in 500 iterations when it should converge in ten iterations will not be detected by order verification. Note that this type of mistake would be found with an appropriate component-level test fixture for the iterative scheme (as discussed in Chapter 4). Similarly, mistakes which affect the robustness of the numerical algorithm will not be detected, since these mistakes also do not affect the final numerical solution.

Finally, the standard order verification procedure does not verify individual terms in the discretization. Thus a numerical scheme which is formally first-order accurate for convection and second-order accurate for diffusion results in a numerical scheme with first-order accuracy. Mistakes reducing the order of accuracy of the diffusion term to first order would therefore not be detected. This limitation can be addressed by selectively turning terms on and off, which is most easily accomplished when the method of manufactured solutions is employed (see Chapter 6).

5.5.6 Alternative approaches for order verification

In recent years, a number of variants of the order verification procedure have been proposed. Most of these alternative approaches were developed in order to avoid the high cost of generating and computing numerical solutions on highly-refined 3-D meshes. All of the approaches discussed here do indeed reduce the cost of conducting the order verification test relative to the standard approach of systematic mesh refinement; however, each approach is also accompanied by drawbacks which are also discussed. These alternative approaches are presented in order of increasing reliability, with a summary of the strengths and weaknesses of each approach presented at the end of this section.

5.5.6.1 Residual method

In general, when the discrete operator $L_h(\cdot)$ operates on anything but the exact solution to the discrete equations u_h, the nonzero result is referred to as the *discrete residual*, or simply the *residual*. This discrete residual is not to be confused with the iterative residual which is found by inserting an approximate iterative solution into the discrete equations (see Chapter 7). Recall that the truncation error can be evaluated by inserting the exact solution to the mathematical model $\tilde{u}$ into the discrete operator as given previously in Eq. (5.13). Assuming an exact solution to the mathematical model is available, the truncation error (i.e., the discrete residual) can be evaluated directly on a given mesh without actually solving for the numerical solution on this grid. Since no iterations are needed, this truncation error evaluation is usually very inexpensive. The truncation error found by inserting the exact solution to the mathematical model into the discrete operator can be evaluated on a series of systematically-refined meshes. The observed order of accuracy is then computed by Eq. (5.23), but using norms of the truncation error rather than the discretization error.

There are a number of drawbacks to the residual form of order verification that are related to the fact that the residual does not incorporate all aspects of the code (Ober, 2004). Specifically, the residual method does not test:

- boundary conditions that do not contribute to the residual (e.g., Dirichlet boundary conditions),
- system response quantities such as lift, drag, combustion efficiency, maximum heat flux, maximum stress, oscillation frequency, etc.,
- numerical algorithms where the governing equations are solved in a segregated or decoupled manner (e.g., the SIMPLE algorithm (Patankar, 1980) for incompressible fluids problems where the momentum equations are solved, followed by a pressure projection step to satisfy the mass conservation equation), and
- explicit multi-stage schemes such as Runge-Kutta.

In addition, it has been observed that the truncation error can converge at a lower rate than the discretization error for finite-volume schemes on certain unstructured mesh topologies (Despres, 2004; Thomas *et al*., 2008). In these cases, it is possible that the residual method might exhibit a lower order of accuracy than a traditional order-of-accuracy test applied to the discretization error. For examples of the residual method applied for code verification see Burg and Murali (2004) and Thomas *et al*. (2008).

5.5.6.2 Statistical method

The *statistical form of order verification* was proposed by Hebert and Luke (2005) and uses only a single mesh, which is successively scaled down and used to sample over the chosen domain. The sampling is performed randomly, and norms of the volume-weighted discretization error are examined. The main advantage of the statistical method is that it is relatively inexpensive because it does not require refinement of the mesh. There are, however, a number of disadvantages. First, since the domain is sampled statistically, convergence of the statistical method must be ensured. Second, it tends to weight the boundary points more heavily as the grids are shrunk relative to traditional order verification since the ratio of boundary points to interior points is fixed rather than reducing with mesh refinement. Finally, this approach assumes that the discretization errors are independent random variables, thus neglecting the transported component of error into the refined domains. (See Chapter 8 for a discussion of the difference between transported and locally-generated components of the discretization error.) Due to these issues, it is possible to pass a statistical order verification test for a case that would fail a traditional order verification test based on systematic mesh refinement.

5.5.6.3 Downscaling method

The *downscaling approach* to order verification (Diskin and Thomas, 2007; Thomas *et al*., 2008) shares a number of attributes with the statistical method described above. The major difference is that instead of statistically sampling the smaller meshes in the domain, the mesh is scaled down about a single point in the domain, which eliminates the statistical convergence issues associated with statistical order verification. The focal point to which

Table 5.4 *Comparison of different order verification approaches showing the cost and the type of order-of-accuracy estimate produced (adapted from Thomas* et al.*, 2008).*

Verification method	Cost	Type of order estimate
Standard order verification with systematic mesh refinement	High	Precise order of accuracy
Downscaling method	Moderate to low	Admits false positives
Statistical method	Moderate (due to sampling)	Admits false positives
Residual method	Very low	Admits false positives and false negatives

the grids are scaled can be chosen to emphasize the internal discretization, the boundary discretization, or singularities (Thomas *et al*., 2008). A major benefit of the downscaling method is that it allows for boundary condition verification in the presence of complex geometries. The mesh scaling can be performed in a very simple manner when examining the interior discretization or straight boundaries, but must be modified to ensure the proper scaling of a mesh around a curved boundary. The downscaling method also neglects the possibility of discretization error transport into the scaled-down domain, and thus can provide overly optimistic estimates of the actual convergence rate.

5.5.6.4 Summary of order verification approaches

In order to summarize the characteristics of the different order verification approaches, it is first helpful to categorize the results of an order verification test. Here we will define a positive result as one in which the observed order of accuracy is found to match the formal order in an asymptotic sense, while a negative result is one where the observed order is less than the formal order. We now define a false positive as a case where a less-rigorous order verification test achieves a positive result, but the more rigorous order verification procedure with systematic mesh refinement produces a negative result. Similarly, a false negative occurs when the test result is negative, but standard order verification with systematic mesh refinement is positive. The characteristics of the four approaches for order-of-accuracy verification are given in Table 5.4. As one might expect, the cost of conducting and order verification study varies proportionately with the reliability of the observed order of accuracy estimate.

5.6 Responsibility for code verification

The ultimate responsibility for ensuring that rigorous code verification has been performed lies with the user of the scientific computing code. This holds true whether the code was developed by the user, by someone else in the user's organization, or by an independent organization (government laboratory, commercial software company, etc.). It is not sufficient for the code user to simply assume that code verification studies have been successfully performed.

In the ideal case, code verification should be performed during, and as an integrated part of, the software development process. While code verification studies are most often conducted by the code developers, a higher level of independence can be achieved when they are conducted by a separate group, by the code customer, or even by an independent regulatory agency (recall the discussion of independence of the verification and validation process in Chapter 2). While there are often fewer coding mistakes to find in a scientific computing code that has a prior usage history, performing code verification studies for a mature code can be expensive and challenging if the code was not designed with code verification testing in mind.

Commercial companies that produce scientific computing software rarely perform rigorous code verification studies, or if they do, they do not make the results public. Most code verification efforts that are documented for commercial codes seem to be limited to simple benchmark examples that demonstrate "engineering accuracy" rather than verifying the order of accuracy of the code (Oberkampf and Trucano, 2008). Recently, Abanto *et al*. (2005) performed order verification studies on three different commercial computational fluid dynamics codes which were formally at least second-order accurate. Most tests resulted in either first-order accuracy or nonconvergent behavior with mesh refinement. It is our opinion that code users should be aware that commercial software companies are unlikely to perform rigorous code verification studies unless users request it.

In the absence of rigorous, documented code verification evidence, there are code verification activities that can be performed by the user. In addition to the simple code verification activities discussed in Section 5.1, order verification tests can also be conducted when an exact solution (or a verifiably accurate surrogate solution) to the mathematical model is available. While the traditional approach for finding exact solutions can be used, the more general method of manufactured solutions procedure requires the ability to employ user-defined boundary conditions, initial conditions, and source terms, and thus can be difficult to implement for a user who does not have access to source code. The next chapter focuses on different approaches for obtaining exact solutions to the mathematical model including the method of manufactured solutions.

5.7 References

Abanto, J., D. Pelletier, A. Garon, J-Y. Trepanier, and M. Reggio (2005). *Verication of some Commercial CFD Codes on Atypical CFD Problems*, AIAA Paper 2005–682.

Banks, J. W., T. Aslam, and W. J. Rider (2008). On sub-linear convergence for linearly degenerate waves in capturing schemes, *Journal of Computational Physics*. **227**, 6985–7002.

Burg, C. and V. Murali (2004). *Efficient Code Verification Using the Residual Formulation of the Method of Manufactured Solutions*, AIAA Paper 2004–2628.

Carpenter, M. H. and J. H. Casper (1999). Accuracy of shock capturing in two spatial dimensions, *AIAA Journal*. **37**(9), 1072–1079.

Crank, J. and P. A. Nicolson (1947). Practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, *Proceedings of the Cambridge Philosophical Society*. **43**, 50–67.

Despres, B. (2004). Lax theorem and finite volume schemes, *Mathematics of Computation*. **73**(247), 1203–1234.

Diskin, B. and J. L. Thomas (2007). *Accuracy Analysis for Mixed-Element Finite Volume Discretization Schemes*, Technical Report TR 2007–8, Hampton, VA, National Institute of Aerospace.

Engquist, B. and B. Sjogreen (1998). The convergence rate of finite difference schemes in the presence of shocks, *SIAM Journal of Numerical Analysis*. **35**(6), 2464–2485.

Ferziger, J. H. and M. Peric (1996). Further discussion of numerical errors in CFD, *International Journal for Numerical Methods in Fluids*. **23**(12), 1263–1274.

Ferziger, J. H. and M. Peric (2002). *Computational Methods for Fluid Dynamics*, 3rd edn., Berlin, Springer-Verlag.

Grinstein, F. F., L. G. Margolin, and W. J. Rider (2007). *Implicit Large Eddy Simulation: Computing Turbulent Fluid Dynamics*, Cambridge, UK, Cambridge University Press.

Hebert, S. and E. A. Luke (2005). *Honey, I Shrunk the Grids! A New Approach to CFD Verification Studies*, AIAA Paper 2005–685.

Hirsch, C. (2007). *Numerical Computation of Internal and External Flows (Vol. 1)*, 2nd edn., Burlington, MA, Elsevier.

Kamm, J. R., W. J. Rider, and J. S. Brock (2003). *Combined Space and Time Convergence Analyses of a Compressible Flow Algorithm*, AIAA Paper 2003–4241.

Knupp, P. M. (2003). Algebraic mesh quality metrics for unstructured initial meshes, *Finite Elements in Analysis and Design*. **39**(3), 217–241.

Knupp, P. M. (2009). Private communication, March 9, 2009.

Knupp, P. M. and K. Salari (2003). *Verification of Computer Codes in Computational Science and Engineering*, K. H. Rosen (ed.), Boca Raton, FL, Chapman and Hall/CRC.

Knupp, P., C. Ober, and R. Bond (2007). Measuring progress in order-verification within software development projects, *Engineering with Computers*. **23**, 271–282.

Mastin, C. W. (1999). Truncation Error on Structured Grids, in *Handbook of Grid Generation*, J. F. Thompson, B. K. Soni, and N. P. Weatherill, (eds.), Boca Raton, CRC Press.

Ober, C. C. (2004). Private communication, August 19, 2004.

Oberkampf, W. L. and T. G. Trucano (2008). Verification and validation benchmarks, *Nuclear Engineering and Design*. **238**(3), 716–743.

Panton, R. L. (2005). *Incompressible Flow*, 3rd edn., Hoboken, NJ, John Wiley and Sons.

Patankar, S. V. (1980). *Numerical Heat Transfer and Fluid Flow*, New York, Hemisphere Publishing Corp.

Potter, D. L., F. G. Blottner, A. R. Black, C. J. Roy, and B. L. Bainbridge (2005). *Visualization of Instrumental Verification Information Details (VIVID): Code Development, Description, and Usage*, SAND2005–1485, Albuquerque, NM, Sandia National Laboratories.

Richards, S. A. (1997). Completed Richardson extrapolation in space and time, *Communications in Numerical Methods in Engineering*. **13**, 573–582.

Richtmyer, R. D. and K. W. Morton (1967). *Difference Methods for Initial-value Problems*, 2nd edn., New York, John Wiley and Sons.

Rider, W. J. (2009). Private communication, March 27, 2009.

Roache, P. J. (1998). *Verification and Validation in Computational Science and Engineering*, Albuquerque, NM, Hermosa Publishers.

Roy, C. J. (2003). Grid convergence error analysis for mixed-order numerical schemes, *AIAA Journal*. **41**(4), 595–604.

Roy, C. J. (2005). Review of code and solution verification procedures for computational simulation, *Journal of Computational Physics*. **205**(1), 131–156.

Roy, C. J. (2009). *Strategies for Driving Mesh Adaptation in CFD*, AIAA Paper 2009–1302.

Roy, C. J., E. Tendean, S. P. Veluri, R. Rifki, E. A. Luke, and S. Hebert (2007). *Verification of RANS Turbulence Models in Loci-CHEM using the Method of Manufactured Solutions*, AIAA Paper 2007–4203.

Tannehill, J. C., D. A. Anderson, and R. H. Pletcher (1997). *Computational Fluid Mechanics and Heat Transfer*, 2nd edn., Philadelphia, PA, Taylor and Francis.

Thomas, J. L., B. Diskin, and C. L. Rumsey (2008). Toward verification of unstructured-grid solvers, *AIAA Journal*. **46**(12), 3070–3079.

Thompson, J. F., Z. U. A. Warsi, and C. W. Mastin (1985). *Numerical Grid Generation: Foundations and Applications*, New York, Elsevier. (www.erc.msstate.edu/publications/gridbook).

Trucano, T. G., M. M. Pilch, and W. L. Oberkampf (2003). *On the Role of Code Comparisons in Verification and Validation*, SAND 2003–2752, Albuquerque, NM, Sandia National Laboratories.

Veluri, S., C. J. Roy, S. Hebert, and E. A. Luke (2008). *Verification of the Loci-CHEM CFD Code Using the Method of Manufactured Solutions*, AIAA Paper 2008–661.