Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 205 (2005) 131–156
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Review of code and solution verification procedures for computational simulation ☆

Christopher J. Roy *

Department of Aerospace Engineering, Auburn University, 211 Aerospace Engineering Bldg., AL 36849, USA

Received 3 June 2004; received in revised form 24 September 2004; accepted 26 October 2004
Available online 21 December 2004

Abstract

Computational simulation can be defined as any computer application which involves the numerical solution to a system of partial differential equations. In this paper, a broad overview is given of verification procedures for computational simulation. The two aspects of verification examined are code verification and solution verification. Code verification is a set of procedures developed to find coding mistakes that affect the numerical discretization. The method of manufactured solutions combined with order of accuracy verification is recommended for code verification, and this procedure is described in detail. Solution verification is used to estimate the numerical errors that occur in every computational simulation. Both round-off and iterative convergence errors are discussed, and a posteriori methods for estimating the discretization error are examined. Emphasis is placed on discretization error estimation methods based on Richardson extrapolation as they are equally applicable to any numerical method. Additional topics covered include calculating the observed order of accuracy, error bands, and practical aspects of mesh refinement.
© 2004 Elseiver Inc. All rights reserved.

1. Introduction

Due to increases in computer speed and advances in parallel processing algorithms, the cost of computational simulation tends to drop relative to experimentation. While computational simulation will never replace experiments, it will continue to play an increased role in the design, analysis, and optimization of engineering systems. In order for computational simulation to achieve its full potential as a predictive

☆ A preliminary version of this paper was presented at CHT-04: An ICHMT International Symposium on Advances in Computational Heat Transfer, April 2004, G. de Vahl Davis and E. Leonardi (Eds.), CD-ROM Proceedings, ISBN 1-5670-174-2, Begell House, New York, 2004.

* Tel.: +1 334 844 5187; fax: +1 334 844 6803.
E-mail address: cjroy@eng.auburn.edu.

0021-9991/$ - see front matter © 2004 Elseiver Inc. All rights reserved.
doi:10.1016/j.jcp.2004.10.036

tool, engineers must have confidence that the simulation results are an accurate representation of reality. Verification and validation provide a framework for building confidence in computational simulation predictions.

In common usage, the words verification and validation are synonymous; however, in current engineering usage, they have very different meanings. Verification deals with mathematics and addresses the correctness of the numerical solution to a given model. Validation, on the other hand, deals with physics and addresses the appropriateness of the model in reproducing experimental data [1]. Verification can be thought of as solving the chosen equations correctly, while validation is choosing the correct equations in the first place. While many of the examples in the current paper are taken from fluid mechanics and heat transfer, the concepts are equally applicable to any discipline (e.g., structural mechanics, electrodynamics, astrophysics) which involves the numerical solution to partial differential equations, herein referred to as computational simulation.

There are two fundamental aspects to verification: code verification and solution verification. Code verification is the process of ensuring, to the degree possible, that there are no mistakes (bugs) in a computer code or inconsistencies in the solution algorithm. Solution verification is the process of quantifying the numerical errors that occur in every numerical simulation. Examples of these numerical errors include round-off error, iterative error, and discretization error. While code verification is generally performed once and for all for a given set of coding options (at least for smooth problems), solution verification must be performed for each and every simulation that is significantly different than previous simulations.

The purposes of this paper are to (1) provide a broad overview of verification procedures for computational simulation and (2) give details on some of the more practical aspects of verification. The first half of the paper (Section 2) addresses code verification. In this section, issues such as software quality assurance and order of accuracy verification are addressed, and a significant amount of time is spent discussing the method of manufactured solutions. The remainder of the paper (Section 3) deals with solution verification and discusses the different sources of numerical error in computational simulation with an emphasis on estimating the discretization error. This section gives an introduction to Richardson extrapolation and its use as an error estimator as well as some of the more practical aspects of grid refinement.

2. Code verification

Coding mistakes are an often overlooked source of error in computational simulation. Although a formal proof of the "correctness" of a given simulation code is probably not forthcoming [2], code verification provides a framework for developing and maintaining reliable computer codes. This framework includes both software quality assurance and verification of the numerical discretization. There are two books which contain a wealth of knowledge on the topic of code verification. Roache [2] provides an excellent overview of the subject, with emphasis on order of accuracy verification and the method of manufactured solutions, both of which are discussed in this section. The book by Knupp and Salari [3] is entirely dedicated to code verification, and is the most comprehensive and up-to-date reference on the subject.

2.1. Software quality assurance

Software Quality Assurance, or SQA, is a formal set of procedures developed to ensure that software is reliable. It was developed primarily from within the computer science community, and is driven by high-consequence software systems such as aircraft control systems, nuclear power plants, and nuclear weapons systems [3]. When applying SQA procedures to computational simulation, the most important aspects include planning, configuration management, testing, and documentation [4].

Careful planning of a software development project is needed to account for manpower availability, version release and delivery schedules, and budgetary constraints. As part of the planning process, the software requirements should be clearly defined and agreed to by all parties involved including code developers, code users, and any additional code customers.

The overall framework for developing quality computational simulation software is called configuration management, and includes a wide variety of activities. Standardized coding practices such as the use of subroutine templates provide uniformity during code development. Configuration management software such as the freely available concurrent versions system (CVS) [5] is an essential tool when a number of developers are involved on a coding project. This software is able to integrate changes from multiple developers working on the same source file and provides version control which allows access to prior versions of the code by tracking code changes.

SQA utilizes analysis and testing procedures including static analysis, dynamic analysis, and regression testing. Static analysis is any analysis conducted without actually running the code and includes such activities as compiling the code (possibly with different compilers on different platforms) and running external diagnostic software to check variable initialization and consistency of argument lists for subroutines and functions. Dynamic analysis includes any activity which involves running the code. Examples of dynamic analysis include run-time compiler options (such as options for checking array bounds) and external software to find memory leaks. While numerical algorithm testing is technically a form of dynamic testing, it is such an important aspect of code verification for computational simulation that it will be addressed in a separate section. Finally, regression tests involve the comparison of code output to the output from earlier versions of the code, and are designed to find coding mistakes by detecting unintended changes in the code. It is important that the regression test suite be designed to obtain coverage of as much of the code as possible (i.e., all models and coding options). The results of SQA testing should be logged so that failures can be reported and corrected. Finally, code documentation is a critical area and includes documentation of code requirements, the software development plan, the verification and testing plan, governing and auxiliary equations, and available coding options.

2.2. Consistency and convergence

For a numerical scheme to be consistent, the discretized equations must approach the original (continuum) partial differential equations in the limit as the element size (Δx, Δt, etc.) approaches zero. For a stable numerical scheme, the errors must not grow in the marching direction. These errors can be due to any source (round-off error, iterative error, etc). It should be noted that typical stability analyses such as von Neumann's method [6] are valid for linear equations only. Finally, convergence addresses the issue of the solution to the discretized equations approaching the continuum solution to the partial differential equations in the limit of decreasing element size. Convergence is addressed by Lax's equivalence theorem (again valid for linear equations only) which states that given a properly-posed initial value problem and a consistent numerical scheme, stability is the necessary and sufficient condition for convergence. Thus, consistency addresses the equations, while convergence deals with the solution itself. Convergence is measured by evaluating (or estimating) the discretization error. For verification purposes, it is convenient to define the discretization error as the difference between the solution to the discretized equations f_k and the solution to the original partial differential equation f_{exact}

$$\text{DE}_k = f_k - f_{\text{exact}}, \tag{1}$$

where k refers to the mesh level. For the purposes of this paper, the round-off and iterative convergence error are addressed separately, therefore their contributions to the overall discretization error are neglected.

2.3. Code verification evaluation criteria

The criteria for assessing code verification are, in order of increasing rigor [3]:

- expert judgment
- error quantification
- consistency/convergence
- order of accuracy

The least rigorous of these is expert judgment, where an expert is given the code output for a given problem and asked to determine whether or not the results appear correct. The error quantification test involves the quantitative assessment of the error between the numerical solution and an exact or benchmark solution, and the subsequent judgment of whether the error is sufficiently small. Tests for consistency/convergence are used to determine if the discretization error shrinks as the element size is reduced. The most rigorous code verification test is the order of accuracy test, which determines whether of not the discretization error is reduced at the expected rate. This test is equivalent to determining whether the observed order of accuracy matches the formal order of accuracy. For finite-difference and finite-volume methods, the formal order of accuracy is obtained from a truncation error analysis, while for finite-element methods it is found from interpolation theory. Since it is the most difficult test to satisfy (and therefore the most sensitive to coding mistakes), the order of accuracy test is the recommended acceptance test for code verification [3].

2.3.1. Formal order of accuracy

For the simplest case of finite-difference schemes, the formal order of accuracy is determined by the truncation error. One approach to evaluate the truncation error is to start with a Taylor series expansion of the solution variable(s). For example, for the function $T(x)$ expanded about the point x_0, the Taylor series expansion can be written as

$$T(x) = \sum_{k=0}^{\infty} \left. \frac{\partial^k T}{\partial x^k} \right|_{x_0} \frac{(x - x_0)^k}{k!}. \tag{2}$$

Consider the one-dimensional unsteady heat equation given by

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0, \tag{3}$$

where the first term is the unsteady contribution and the second term represents thermal diffusion. This equation can be discretized with finite differences using a forward difference in time and a centered second difference in space, resulting in the simple explicit numerical scheme

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

where the subscripts denote spatial location and the superscripts denote the temporal step. In order to determine the truncation error for this numerical scheme, we can expand each of the above temperature values in terms of the temperature at location i and time step n:

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

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

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

Substituting these expressions into the discretized equation and rearranging yields

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

Eq. (4) represents the actual partial differential equation solved by the discretization scheme. The left-hand side is the original partial differential equation (evaluated at location i and time level n), while the right-hand side represents the difference between the discretized equation and the continuum partial differential equation. This difference is the truncation error. The simple explicit scheme for the unsteady heat equation is consistent since the truncation error (right-hand side of Eq. (4)) goes to zero as Δx and Δt go to zero. The formal order of accuracy of the scheme is first order in time and second order in space since the leading terms in the truncation error contain the factors Δt and $(\Delta x)^2$, respectively.

2.3.2. Observed order of accuracy

The observed order of accuracy is the accuracy that is directly computed from code output for a given simulation or set of simulations. The observed order of accuracy can be adversely affected by mistakes in the computer code, solutions which are not sufficiently smooth, defective numerical algorithms, and numerical solutions which are not in the asymptotic grid convergence range. The asymptotic range is defined as the range of discretization sizes (Δx, Δy, Δt, etc.) where the lowest-order terms in the truncation error dominate.

We will now consider a method for calculating the observed order of accuracy assuming that the exact solution is known. The case when the exact solution is not known will be addressed in the Solution Verification section. Recall that in Eq. (1) we defined the discretization error as the difference between the numerical solution on a given mesh and the exact (continuum) solution to the partial differential equations f_{exact}. Consider a series expansion of the discretization error in terms of h_k, a measure of the element size on mesh level k

$$\mathrm{DE}_k = f_k - f_{\text{exact}} = g_p h_k^p + \mathrm{HOT}, \tag{5}$$

where f_k is the numerical solution on mesh k, g_p is the coefficient of the leading error term, and p is the observed order of accuracy. The main assumption is that the higher-order terms (HOT) are negligible, which is equivalent to saying the solutions are in the asymptotic range. In this case, we can write the discretization error equation for a fine mesh ($k = 1$) and a coarse mesh ($k = 2$) as

$$\mathrm{DE}_1 = f_1 - f_{\text{exact}} = g_p h_1^p,$$
$$\mathrm{DE}_2 = f_2 - f_{\text{exact}} = g_p h_2^p.$$

Since the exact solution is known, the left-hand sides can be evaluated using the numerical solution. Combining these two equations as follows

$$\frac{\mathrm{DE}_2}{\mathrm{DE}_1} = \frac{g_p h_2^p}{g_p h_1^p} = \left(\frac{h_2}{h_1}\right)^p,$$

then taking the natural log of both sides and introducing the grid refinement factor $r = h_2/h_1$ (i.e., the ratio between the coarse and fine mesh element sizes), the observed order of accuracy p becomes

$$p = \frac{\ln\left(\frac{\mathrm{DE}_2}{\mathrm{DE}_1}\right)}{\ln(r)}. \tag{6}$$

Thus, when the exact solution is known, only two solutions are required to obtain the observed order of accuracy. The observed order of accuracy can be evaluated either locally within the solution domain or globally by employing a norm of the discretization error.

When evaluating the observed order of accuracy, round-off and iterative convergence error can adversely affect the results. Round-off error occurs due to finite digit storage on digital computers. Iterative error occurs any time an iterative method is used, as is generally the case for nonlinear systems and large, sparse linear systems. Both of these error sources will be discussed in detail in the Solution Verification section. The discretized form of nonlinear equations can generally be solved to within machine round-off error; however, in practice, the iterative procedure is usually terminated earlier to reduce computational effort. In order to ensure that these sources of error do not adversely impact the order of accuracy calculation, both round-off and iterative error should be at least 100 times smaller than the discretization error (i.e., $< 0.01 \times \mathrm{DE}$).

2.4. Method of exact solutions

Code verification has traditionally been performed by using the method of exact solutions. This approach involves the comparison of a numerical solution to an exact solution to the governing partial differential equations with specified initial and boundary conditions. The main disadvantage of this approach is that there are only a limited number of exact solutions available for complex equations (i.e., those with complex geometries, physics, or nonlinearity). When exact solutions are found for complex equations, they often involve significant simplifications. For example, the flow between parallel plates separated by a small gap with one plate moving is called Couette flow and is described by the Navier–Stokes equations. In Couette flow, the velocity profiles are linear across the gap. This linearity causes the diffusion term, a second derivative of velocity, to be identically zero. In contrast to the method of manufactured solutions discussed in the next sub-section, the method of exact solutions involves the solution to the forward problem. That is, given a partial differential equation, boundary conditions, and initial conditions, the goal is to find the exact solution.

2.5. Method of manufactured solutions

The method of manufactured solutions, or MMS, is a general and very powerful approach to code verification. Rather than trying to find an exact solution to a system of partial differential equations, the goal is to "manufacture" an exact solution to a slightly modified set of equations. For code verification purposes, it is not required (in fact, often not desirable) that the manufactured solution be related to a physically realistic problem; recall that verification deals only with the mathematics of a given problem. The general concept behind MMS is to choose the solution a priori, then operate the governing partial differential equations onto the chosen solution, thereby generating analytical source terms. The chosen (manufactured) solution is then the exact solution to the modified governing equations made up of the original equations plus the analytical source terms. Thus, MMS involves the solution to the backward problem: given an original set of equations and a chosen solution, find a modified set of equations that

the chosen solution will satisfy. The initial and boundary conditions are then determined from the solution.

The use of manufactured solutions and grid convergence studies for the purpose of code verification was first proposed by Roache and Steinberg [7]. They employed symbolic manipulation software to verify a code for generating three-dimensional transformations for elliptic partial differential equations. These concepts were later extended by Roache et al. [8]. The term "manufactured solution" was coined by Oberkampf and Blottner [9] and refers to the fact that the method generates (or manufactures) a related set of governing equations for a chosen analytic solution. An extensive discussion of manufactured solutions for code verification was presented by Salari and Knupp [10], and includes both details of the method as well as application to a variety of partial differential equation sets. This report was later refined and published in book form by Knupp and Salari [3]. A recent review/tutorial was given by Roache [11], and the application of the manufactured solutions procedure for the Euler and Navier–Stokes equations for fluid flow was presented by Roy et al. [12].

The procedure for applying MMS with order of accuracy verification can be summarized in the following six steps:

Step 1. Choose the form of the governing equations
Step 2. Choose the form of the manufactured solution
Step 3. Derive the modified governing equations
Step 4. Solve the discrete form of the modified governing equations on multiple meshes
Step 5. Evaluate the global discretization error in the numerical solution
Step 6. Apply the order of accuracy test to determine if the observed order of accuracy matches the formal order of accuracy

The fourth step, which includes the solution to the modified governing equations, may require code modifications to allow arbitrary source terms, initial conditions, and boundary conditions to be used.

Manufactured solutions should be chosen to be smooth, analytical functions with smooth derivatives. The choice of smooth solutions will allow the formal order of accuracy to be achieved on relatively coarse meshes, and trigonometric and exponential functions are recommended. It is also important to ensure that no derivatives vanish, including cross-derivatives. Care should be taken that one term in the governing equations does not dominate the other terms. For example, when verifying a Navier–Stokes code, the manufactured solution should be chosen to give Reynolds numbers near unity so that convective and diffusive terms are of the same order of magnitude. Finally, realizable solutions should be employed, that is, if the code requires the temperature to be positive (e.g., in the evaluation of the speed of sound which involves the square root of the temperature), then the manufactured solution should be chosen as such.

MMS has been applied to the Euler equations, which govern the flow of an inviscid (frictionless) fluid [12]. The two-dimensional, steady-state form of the Euler equations is given by

$$\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}=f_m,$$
$$\frac{\partial(\rho u^2+p)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}=f_x,$$
$$\frac{\partial(\rho vu)}{\partial x}+\frac{\partial(\rho v^2+p)}{\partial y}=f_y,$$
$$\frac{\partial(\rho ue_t+pu)}{\partial x}+\frac{\partial(\rho ve_t+pv)}{\partial y}=f_e,$$

where arbitrary source terms f are included on the right-hand side, and e_t is the specific total energy, which for a calorically perfect gas is given by

$$e_t = \frac{1}{\gamma - 1} RT + \frac{u^2 + v^2}{2}.$$

The final relation needed to close the set of equations is the equation of state for a calorically perfect gas

$$p = \rho RT.$$

The manufactured solution for this case is chosen as

$$\rho(x,y) = \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{a_{\rho y}\pi y}{L}\right),$$
$$u(x,y) = u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y \cos\left(\frac{a_{uy}\pi y}{L}\right),$$
$$v(x,y) = v_0 + v_x \cos\left(\frac{a_{vx}\pi x}{L}\right) + v_y \sin\left(\frac{a_{vy}\pi y}{L}\right),$$
$$p(x,y) = p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right) + p_y \sin\left(\frac{a_{py}\pi y}{L}\right).$$

The subscripts here refer to constants (not differentiation) with the same units as the variable, and the dimensionless a constants generally vary between 0.5 and 1.5 to provide smooth solutions over an $L \times L$ square domain. For this case, the constants were chosen to give supersonic flow in both the positive x and positive y directions. While not necessary, this choice simplifies the inflow boundary conditions to Dirichlet values at the inflow and Neumann (gradient) values at the outflow. The inflow boundary conditions are determined from the manufactured solution. A plot of the manufactured solution for the mass density is given in Fig. 1.

Substitution of the chosen manufactured solutions into the governing equations allows the analytical determination of the source terms. For this example, the source term for the mass conservation equation is

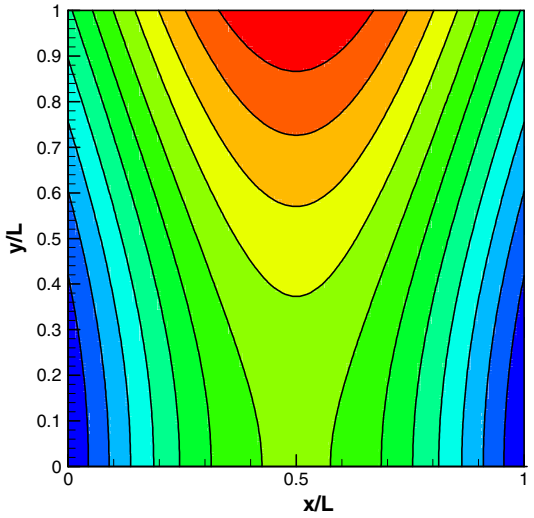

Fig. 1. Manufactured solution for mass density.

$$
\begin{aligned}
f_m = {} & \frac{a_{ux}\pi u_x}{L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left[\rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{a_{\rho y}\pi y}{L}\right)\right] \\
& + \frac{a_{vy}\pi v_y}{L}\cos\left(\frac{a_{vy}\pi y}{L}\right)\left[\rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{a_{\rho y}\pi y}{L}\right)\right] \\
& + \frac{a_{\rho x}\pi \rho_x}{L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left[u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y \cos\left(\frac{a_{uy}\pi y}{L}\right)\right] \\
& + \frac{a_{\rho y}\pi \rho_y}{L}\cos\left(\frac{a_{\rho y}\pi y}{L}\right)\left[v_0 + v_x \sin\left(\frac{a_{vx}\pi x}{L}\right) + v_y \cos\left(\frac{a_{vy}\pi y}{L}\right)\right].
\end{aligned}
$$

As the governing equations get more complex, the use of symbolic manipulation software such as Mathematica™ or Maple™ is strongly recommended. These software packages generally have built-in capabilities to output the source terms as computer code in both FORTRAN and C programming languages. A plot of the above source term for the mass conservation equation is given in Fig. 2.

The governing equations are then discretized and solved on multiple meshes. In this case, two different finite-volume Navier–Stokes codes were employed: Premo, an unstructured grid code, and Wind, a structured grid code (see [12] for details). Both codes utilized the second-order Roe upwind scheme for the convective terms, and second-order central differencing for the diffusive terms. The formal order of accuracy of both codes is thus second order. The five meshes employed are summarized in Table 1.

The coarser meshes were found by successively eliminating every other grid line from the fine mesh (i.e., a grid refinement factor of $r = 2$). Also, the grid spacing is normalized by the fine mesh ($k = 1$) spacing, thus

$$h_k = \frac{\Delta x_k}{\Delta x_1} = \frac{\Delta y_k}{\Delta y_1}.$$

The global discretization error was then examined by employing discrete L_∞ and L_2 norms as follows:

$$
\begin{aligned}
L_{\infty,k} &= \max|f_{k,n} - f_{\text{exact},n}|, \\
L_{2,k} &= \left(\frac{\sum_{n=1}^{N}|f_{k,n} - f_{\text{exact},n}|^2}{N}\right)^{1/2},
\end{aligned}
\tag{7}
$$

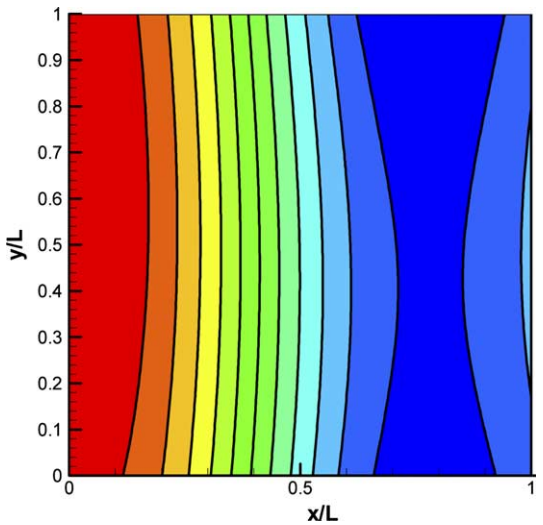

Fig. 2. Analytical source term for the mass conservation equation.

Table 1
Meshes employed in the Euler manufactured solution

Mesh name	Mesh nodes	Grid spacing, h
Mesh 1	129×129	1
Mesh 2	65×65	2
Mesh 3	33×33	4
Mesh 4	17×17	8
Mesh 5	9×9	16

where f_{exact} comes directly from the chosen manufactured solution and n is summed over all of the nodes on the mesh. The behavior of these two discretization error norms as a function of the measure of the element size h is given in Fig. 3. On the logarithmic scale, a first-order scheme will display a slope of unity, while a second-order scheme will give a slope of two. In this case, the observed order of accuracy for both codes is two, thus the formal order of accuracy is recovered, and the two codes are verified for the options examined.

Another method for assessing the observed order of accuracy is to calculate it using the global norms of the discretization error. Since the exact solution is known, the relation for the observed order of accuracy in terms of the error norms is

$$p = \frac{\ln\left(\frac{L_{k+1}}{L_k}\right)}{\ln(r)}, \tag{8}$$

where L_k refers to one of the global error norms given above. A plot of the observed order of accuracy as a function of the element size h is presented in Fig. 4. The Premo code clearly asymptotes to second-order accuracy ($p = 2$), while the Wind code appears to asymptote to an order of accuracy that is slightly higher than two. Further grid refinement would likely provide more definitive results for the Wind code. It is important to note that while the current example was performed on Cartesian meshes for simplicity, a more

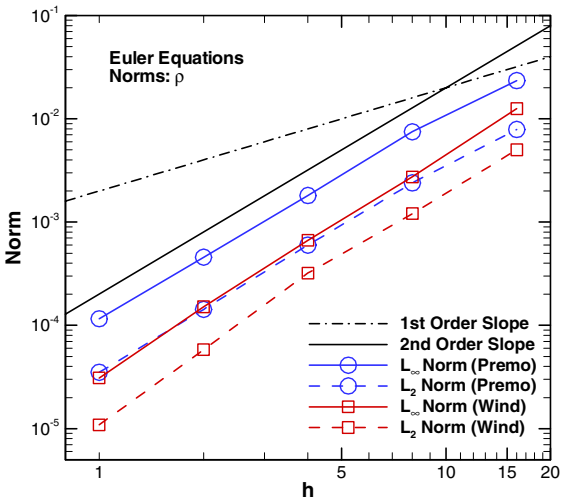

Fig. 3. Global norms of the discretization error in mass density.

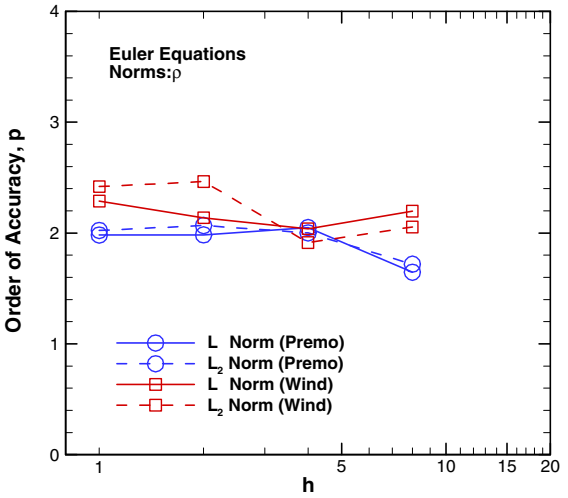

Fig. 4. Observed order of accuracy for the mass density.

general code verification analysis would employ meshes with significant stretching and skewness as well as arbitrary orientations for the domain boundaries.

The use of MMS for code verification has been shown to be remarkably sensitive to mistakes in the discretization (e.g., see Chapter 3.11 of [2]). In addition to the method's sensitivity, there are a number of other advantages to using MMS for code verification. MMS can verify most of the coding options available in codes which numerically solve partial differential equations, and it handles nonlinear and coupled sets of equations without additional difficulty. MMS can also be used to detect flaws in the solution algorithm that lead to inconsistency or non-convergent behavior. Perhaps most importantly, the method works equally well for finite-difference, finite-volume, and finite-element schemes.

There are some disadvantages to using MMS for code verification. MMS requires the ability to include arbitrary source terms, initial conditions, and boundary conditions to a code. Even when the code does provide a framework for including general source terms, the specific form for each manufactured solution must be incorporated into the code. MMS is thus code-intrusive and generally cannot be performed as a black-box analysis. In addition, each model option which changes the governing or auxiliary equations requires new source terms to be generated, which can be time consuming. Since MMS is predicated on having smooth solutions, the analysis of non-smooth solutions (shock-waves, material interfaces, etc.) is an open research issue. Since MMS can only detect problems in the solution itself, it generally cannot be used to detect coding mistakes affecting either the efficiency or robustness of the code.

2.6. Additional approaches for code verification

Although less rigorous than the methods discussed previously, there are two approaches to code verification which can also help build confidence that a code is verified. The first approach is the use of benchmark solutions which have been evaluated or calculated with a demonstrated high-degree of accuracy. One example is the Blasius solution to the laminar boundary layer equations. The Blasius solution employs similarity transformations to reduce the governing partial differential equations to ordinary differential equations, which can then be accurately solved using series solution (the original approach of Blasius) or by numerical solution. The second approach to code verification is code-to-code comparison, where the results from one code are compared to the results from another code. This approach should be used

with caution since a code cannot be verified with confidence by comparison to an unverified code. This method is also susceptible to systematic errors in solution algorithms, a situation that can occur when new codes are developed which are even loosely based on earlier solution algorithms (as is often the case). See [13] for more details on the pitfalls of using code-to-code comparisons for code verification.

3. Solution verification

Solution verification deals with the assessment of the numerical errors which always exist when partial differential equations are solved numerically. Solution verification must necessarily take place after code verification has been conducted; it does not make sense to perform solution verification on a code that contains coding mistakes which affect the discretization error. There are three main aspects to solution verification: verification of input data, numerical error estimation of the solution, and verification of output data. Verification of input data is needed to ensure that the correct input files, grids, tabulated data, etc. are employed. For example, archiving the input file in saved files from a simulation run will allow the analyst to go back and double check that the correct input file was used. Similarly, verification of output data is needed to ensure that the user takes the correct post-processing steps after the simulation is completed. The main aspect of solution verification is the estimation of numerical errors, which is the focus of the current section.

3.1. Sources of numerical error

The three main sources of numerical error in computational simulation are round-off error, iterative convergence error, and discretization error. The latter error source includes both errors in the interior discretization scheme as well as errors in the discretization of the boundary conditions. These error sources are discussed in detail in the following sub-sections.

3.1.1. Round-off error

Round-off errors occur due to the use of finite arithmetic on digital computers. For example, in a single-precision digital computation the follow result is often obtained

$$3.0 * (1.0/3.0) = 0.9999999,$$

while the true answer is of course 1.0. Round-off error can be important for both ill-conditioned systems of equations as well as time-accurate simulations. The adverse effects of round-off error can be mitigated by using more significant digits in the computation. Standard computers employ 32 bits of memory for each storage location. In a double precision calculation, two storage locations are allocated for each number, thus providing 64 bits of memory. Higher-precision storage can be accessed through variable declarations, by using appropriate compiler flags, or by employing one of the recently developed 64-bit computer architectures.

3.1.2. Iterative convergence error

Iterative convergence error arises due to incomplete iterative convergence of a discrete system. Iterative methods are generally required for complex nonlinear systems of equations, but are also the most efficient approach for large, sparse linear systems. The two classes of iterative approaches for linear systems are stationary iterative methods (Jacobi, Gauss–Seidel, line relaxation, multigrid, etc.) and Krylov subspace methods (GMRES, conjugate gradient, Bi-CGSTAB, etc.). Nonlinear systems of equations also employ the above iterative methods, but generally in conjunction with a linearization procedure (e.g., Picard iteration, Newton's method).

As a simple example of an iterative method for nonlinear equations, consider the one-dimensional Burgers equation in dimensionless form:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \frac{1}{Re}\frac{\partial^2 u}{\partial x^2} = 0. \tag{9}$$

This equation contains terms representing unsteadiness, nonlinear convection, and diffusion, and is thus a popular model equation for the Navier–Stokes equations governing viscous fluid flow. Employing a Picard iteration with a simple linearization of the nonlinear term (denoted by the overbar), the resulting linear equation can be discretized using the implicit Euler method to give

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \bar{u}_i^{n+1}\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} - \frac{1}{\mathrm{Re}}\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} = 0, \tag{10}$$

where the convective and diffusive terms are evaluated at the unknown $(n+1)$ time level. This numerical scheme results in a tridiagonal system of algebraic equations which can be solved using the well-known Thomas algorithm. Once the unknowns are solved for at the $(n+1)$ time level, the linearized portion $\bar{u}_i^{n+1}$ is then updated and the linear solution is repeated until the convergence criteria is reached.

The question now arises of what to use for the convergence criteria. Authors often report the convergence of iterative methods in terms of the difference (either absolute or relative) between successive iterates. This approach can be misleading when convergence is slow or stalled. A much better approach is to evaluate the residual at each iterate, where the residual is found by plugging the current numerical solution into the discretized form of the equations. In the above example for Burgers equation, the residual R at location i and time level $(n+1)$ is simply

$$R_i^{n+1} = \frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^{n+1}\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} - \frac{1}{\mathrm{Re}}\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2}. \tag{11}$$

The residual will approach zero (within machine round-off error) as the iterations converge.

Another approach for assessing the iterative convergence of a numerical method is to estimate the actual iterative error in a quantity of interest. Here, we define iterative error as the difference between the current iterative solution to the discretized equations and the exact solution to the discretized equations (note: the latter is not the same as the exact solution to the original governing equation). Roy and Blottner [14,15] utilized knowledge of the convergence behavior of a Navier–Stokes code to develop a method for estimating the iterative error in quantities of interest. This method relies on having solutions at three iteration levels, and was found to provide accurate estimates of the iterative error when compared to solutions that were iteratively converged to machine zero using double-precision computations. Their method assumes that convergence of the iterative method occurs monotonically, while Ferziger and Peric [16] independently developed an iterative error estimator which also addresses oscillatory iterative convergence. The latter study by Roy and Blottner [15], as well as recent work by Roy et al. [12] has shown that the residual reduction tracks extremely well with actual iterative error for a wide range of nonlinear problems in fluid mechanics. This observation supports the common practice of assessing iterative convergence indirectly by examining the residuals. Examples from manufactured solution computations for (a) the Euler equations and (b) the Navier–Stokes equations are presented in Fig. 5. These solutions were obtained with a double precision code (Premo) and with a single precision code (Wind). In all cases, the residual norms closely follow the actual iterative error, which is determined by taking the global L2 norm of the difference between the current iterative solution and the iterative solution converged to machine zero. See [12] for additional details.

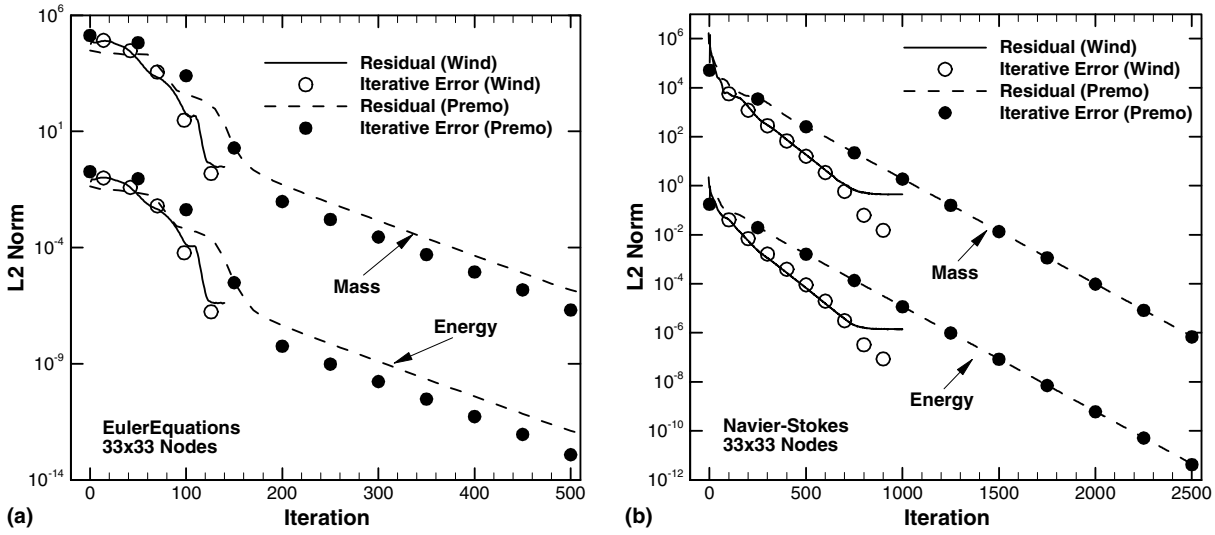

Fig. 5. Global norms of the iterative convergence error and residuals for manufactured solutions of (a) the Euler equations and (b) the Navier–Stokes equations.

3.1.3. Discretization error

The discretization error was defined in Eq. (1) as the difference between a numerical solution and the exact solution to the continuum partial differential equations. It arises due to the conversion of the differential equations into an algebraic system of equations (i.e., the discretization process). This process necessarily introduces discretization parameters such as the element size (Δx, Δy, and Δz) and/or the time step (Δt). The discretization error can be clearly related to the truncation error for linear problems; however, for nonlinear problems, this relationship is not straightforward.

There are two main reasons for evaluating the discretization error. The first reason is to obtain an assessment of the discretization error associated with a given solution, which might be needed during analysis of simulation results or for a model validation study. This error assessment can take three distinct forms: an error estimate (e.g., the most likely value for the error is -5%), an error band (e.g., a confidence level of 95% that the error is within $\pm 8\%$), or an error bound (e.g., the error is guaranteed to be within $\pm 8\%$). The second reason for evaluating the discretization error is to drive a grid adaptation process. Grid adaptation can involve locally adding more elements (h-adaptation), moving points from a region of low error to a region of high error (r-adaptation), or locally increasing the formal order of accuracy (p-adaptation).

3.2. Discretization error estimation

There are a number of methods available for estimating discretization error. These methods can be broadly categorized as a priori methods and a posteriori methods. The a priori methods are those that allow a bound to be placed on the discretization error before the numerical solution is calculated, i.e., find C and p such that $DE < Ch^p$. Here p is simply the formal order of accuracy and can be determined by the methods discussed earlier. The determination of the constant C is challenging and generally problem dependent, and can be very large (and thus not useful) for complex problems. The majority of research today is focused on a posteriori methods for estimating the discretization error. These methods provide an error estimate only after the numerical solution is computed. The a posteriori methods can be further sub-divided into finite-element-based error estimators and extrapolation-based error estimators. Although a brief over-

view of the former is given in the next sub-section, this paper focuses on the latter approach since it is equally applicable to finite-difference, finite-volume, and finite-element methods.

In general, the level of maturity for all of the a posteriori error estimation methods is heavily problem dependent [17]. As a whole, they tend to work well for elliptic problems, but are not as well-developed for parabolic and hyperbolic problems. The level of complexity of the problem is also an important issue. The error estimators work well for smooth, linear problems with simple physics and geometries; however, strong nonlinearities, discontinuities, singularities, and physical and geometric complexity can significantly reduce the reliability and applicability of these methods.

3.2.1. Finite-element-based error estimators

Two fundamentally different types of discretization error estimators have been developed from the finite-element method [17]. The most widely-used are recovery methods, which involve post-processing of the solution gradients (Zienkiewicz-Zhu [18]) or nodal values (Zhang et al. [19]) on patches of neighboring elements. The former approach is often referred to as the ZZ error estimator, while the latter as polynomial preserving recovery (PPR). The basic formulations provide error estimates only in the global energy norm; extensions to quantities of interest must generally be done heuristically (e.g., a 5% error in the global energy norm may correspond to a 10% error in heat flux for a given class of problems). Although difficult to analyze mathematically, recovery-based error estimators do provide ordered error estimates. That is, the error estimate gets better with mesh refinement. Recovery-based methods can be extended to finite-difference and finite-volume schemes, but this process generally requires additional effort.

The second class of error estimators that has arisen from finite elements are residual-based methods. These methods take the form of either explicit residual methods (e.g., Eriksson and Johnson [20]) or implicit residual methods (e.g., Babuska and Miller [21]). These methods were originally formulated to provide error estimates in the global energy norm. Extension of both the explicit and implicit residual methods to provide error estimates in quantities of interest generally requires the solution to the adjoint system (i.e., the dual problem). The explicit method has been extended to finite-volume schemes by Barth and Larson [22]. For more information on residual-based a posteriori error estimators for finite-elements, see [23,24].

3.2.2. Extrapolation-based error estimators

The extrapolation-based error estimators come in two different forms. The most popular approach is based on Richardson extrapolation [25,26] (also referred to as h-extrapolation) and requires numerical solutions on two or more meshes with different levels of refinement. The numerical solutions are then used to obtain a higher-order estimate of the exact solution. This estimate of the exact solution can then be used to estimate the error in the numerical solutions. The second type of extrapolation-based error estimator is order extrapolation (p-extrapolation). In this approach, solutions on the same mesh, but with two different formal orders of accuracy, are used to obtain a higher-order accurate solution, which can again be used to estimate the error. The drawback to order-extrapolation is that it requires advanced solution algorithms to obtain higher-order numerical schemes, which can be difficult to code and expensive to run. The main advantage of the extrapolation-based error estimators is that they can be applied as a post-processing step to any type of discretization, whether it be a finite-difference, finite-volume, or finite-element method.

3.3. Richardson extrapolation

Richardson extrapolation [25,26] is based on the series expansion of the discretization error given in Eq. (5), which can be rewritten as

$$\mathrm{DE}_k = f_k - f_{\mathrm{exact}} = g_1 h_k + g_2 h_k^2 + g_3 h_k^3 + g_4 h_k^4 + \mathrm{HOT}, \tag{12}$$

where g_i is the coefficient of the ith order error term and the exact solution f_{exact} is generally not known. The assumptions that are required for using Richardson extrapolation are that (1) the solutions are smooth, (2) the higher order terms in the discretization error series expansion are small, and (3) uniform meshes are used. The second assumption regarding the higher-order terms is true in the asymptotic range, where h is sufficiently small that the lower-order terms in the expansion dominate. While the last assumption regarding uniform meshes appears to be quite restrictive, transformations (either local or global) can be used if the order of accuracy of the transformation is equal to (or higher than) the order of the numerical scheme. Transformations will be discussed in detail in a later sub-section.

3.3.1. Standard Richardson extrapolation

The standard Richardson extrapolation procedure assumes that the numerical scheme is second-order accurate, and that the mesh is refined or coarsened by a factor of two. Consider a second-order discretization scheme which is used to produce numerical solutions on two meshes: a fine mesh ($h_1 = h$), and a coarse mesh ($h_2 = 2h$). Since the scheme is second-order accurate, the g_1 coefficient is zero, and the discretization error equations on the fine and coarse meshes can be rewritten as

$$f_1 = f_{\text{exact}} + g_2 h^2 + \mathrm{O}\left(h^3\right),$$
$$f_2 = f_{\text{exact}} + g_2 (2h)^2 + \mathrm{O}\left([2h]^3\right).$$

Neglecting higher-order terms, these two equations can be rewritten as

$$f_1 = f_{\text{exact}} + g_2 h^2,$$
$$f_2 = f_{\text{exact}} + g_2 (2h)^2,$$

Solving the first equation for g_2 yields

$$g_2 = \frac{f_1 - f_{\text{exact}}}{h^2}, \tag{13}$$

and solving the second equation for f_{exact} gives

$$f_{\text{exact}} = f_2 - g_2 (2h)^2. \tag{14}$$

Substituting Eq. (13) into Eq. (14) gives

$$f_{\text{exact}} = f_2 - \left[\frac{f_1 - f_{\text{exact}}}{h^2}\right](2h)^2 = f_2 - 4f_1 + 4f_{\text{exact}},$$

or simply

$$f_{\text{exact}} = f_1 + \frac{f_1 - f_2}{3}. \tag{15}$$

Standard Richardson extrapolation thus provides a "correction" to the fine grid solution. This expression for the estimated exact solution f_{exact} is generally third-order accurate. In Richardson's original work [25], he used this extrapolation procedure to obtain a higher-order accurate solution for the stresses in a masonry dam based on two second-order accurate numerical solutions (which were calculated by hand!). In Richardson's case, he employed central differences which cancelled out the odd powers in the truncation error. His estimate for the exact solution was thus fourth-order accurate.

3.3.2. Generalized Richardson extrapolation

Richardson extrapolation can be generalized to pth-order accurate schemes with solutions on a fine mesh (spacing h_1) and a coarse mesh (spacing h_2), which are not necessarily different by a factor of two. Introducing the general grid refinement factor

$$r = \frac{h_2}{h_1}, \tag{16}$$

and setting $h_1 = h$, the discretization error equations can be written as

$$f_1 = f_{\text{exact}} + g_p h^p + \mathrm{O}\left(h^{p+1}\right),$$
$$f_2 = f_{\text{exact}} + g_p (rh)^p + \mathrm{O}\left([rh]^{p+1}\right).$$

Neglecting the higher-order terms, these two equations can be solved for f_{exact} to give

$$f_{\text{exact}} = f_1 + \frac{f_1 - f_2}{r^p - 1}, \tag{17}$$

which is generally a $(p + 1)$-order accurate estimate. Again, it should be emphasized that Richardson extrapolation relies on the assumption that the solutions are asymptotic (i.e., the observed order of accuracy matches the formal order).

3.3.3. Observed order of accuracy

When the exact solution is not known (which is generally the case for solution verification), three numerical solutions on different meshes are required in order to calculate the observed order of accuracy. Consider a pth-order accurate scheme with numerical solutions on a fine mesh (h_1), a medium mesh (h_2), and a coarse mesh (h_3). For the case of a constant grid refinement factor

$$r = h_2/h_1 = h_3/h_2,$$

we can thus write

$$h_1 = h, \quad h_2 = rh, \quad h_3 = r^2 h.$$

The three discretization error equations can be written as

$$f_1 = f_{\text{exact}} + g_p h^p + \mathrm{O}\left(h^{p+1}\right),$$
$$f_2 = f_{\text{exact}} + g_p (rh)^p + \mathrm{O}\left([rh]^{p+1}\right),$$
$$f_3 = f_{\text{exact}} + g_p \left(r^2 h\right)^p + \mathrm{O}\left(\left[r^2 h\right]^{p+1}\right).$$

Neglecting the higher-order terms, these three equations can be used to solve for the observed order of accuracy p to give

$$p = \frac{\ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right)}{\ln(r)}. \tag{18}$$

Note that here the observed order of accuracy is calculated and does not need to be assumed (as with Richardson extrapolation).

For the case of non-constant grid refinement factors

$$r_{12} = h_2/h_1, \quad r_{23} = h_3/h_2,$$

where $r_{12} \neq r_{23}$, the determination of the observed order of accuracy p is more complicated. For this case, the following transcendental equation [2] must be solved

$$\frac{f_3 - f_2}{r_{23}^p - 1} = r_{12}^p \left(\frac{f_2 - f_1}{r_{12}^p - 1} \right). \tag{19}$$

This equation can be easily solved using a simple Picard-type iterative procedure.

3.3.4. Richardson extrapolation as an error estimator

In some cases, researchers mistakenly report discretization error estimates by giving the relative difference between two numerical solutions computed on different meshes, i.e.,

$$\text{Diff} = \frac{f_2 - f_1}{f_1}. \tag{20}$$

This relative difference can be extremely misleading when used as an error estimate. To see why, let us first develop a discretization error estimator using generalized Richardson extrapolation. The relative discretization error (RDE) is simply the difference between the numerical solution and the exact solution, normalized by the exact solution, which for the fine grid ($k = 1$) can be written as

$$\text{RDE}_1 = \frac{f_1 - f_{\text{exact}}}{f_{\text{exact}}}. \tag{21}$$

Substituting the generalized Richardson extrapolation result from Eq. (17) into the numerator gives

$$\text{RDE}_1 = \frac{f_1 - f_{\text{exact}}}{f_{\text{exact}}} = \frac{f_1 - \left[f_1 + \frac{f_1 - f_2}{r^p - 1} \right]}{f_{\text{exact}}} = \frac{f_2 - f_1}{f_{\text{exact}}(r^p - 1)},$$

or simply

$$\text{RDE}_1 = \frac{f_2 - f_1}{f_{\text{exact}}(r^p - 1)}. \tag{22}$$

The reason for leaving f_{exact} in the denominator will be apparent shortly.

Consider two numerical solutions where some quantity of interest has a relative difference (from Eq. (20)) of 5%. For a third-order accurate scheme with $r = 2$, the error estimate based on Richardson extrapolation (Eq. (22)) is 0.71%. However, for a first-order accurate numerical scheme with a grid refinement factor of 1.5, the error estimate based on Richardson extrapolation is 9.1%. Thus, a 5% relative difference in the two solutions can mean very different values for the relative discretization error, depending on the order of accuracy of the scheme and the grid refinement factor. This example illustrates the importance of accounting for the $(r^p - 1)$ factor for obtaining accurate error estimates. This understanding led to the development of Roache's Grid Convergence Index, to be discussed in a later sub-section.

3.3.5. Example: Richardson extrapolation as an error estimator

An example of using Richardson extrapolation as an error estimator was presented by Roy and Blottner [15]. They examined the hypersonic, transitional flow over a sharp cone. The quantity of interest was the heat flux distribution along the surface. The surface heat flux is shown versus the axial coordinate in Fig. 6a for three mesh levels. Also shown are Richardson extrapolation results using the two finest meshes. The sharp rise in heat flux at $x = 0.5$ m is due to the specification of the location for transition from laminar to turbulent flow. In Fig. 6b, the Richardson extrapolation results are used to estimate the relative discretization error (RDE) in each of the numerical solutions. While the coarse mesh shows error estimates as high as 8% in the laminar region, the fine mesh error estimates are much smaller at approximately 0.5%. Note that these error estimates will only have a confidence level of 50%, that is, the true discretization error has

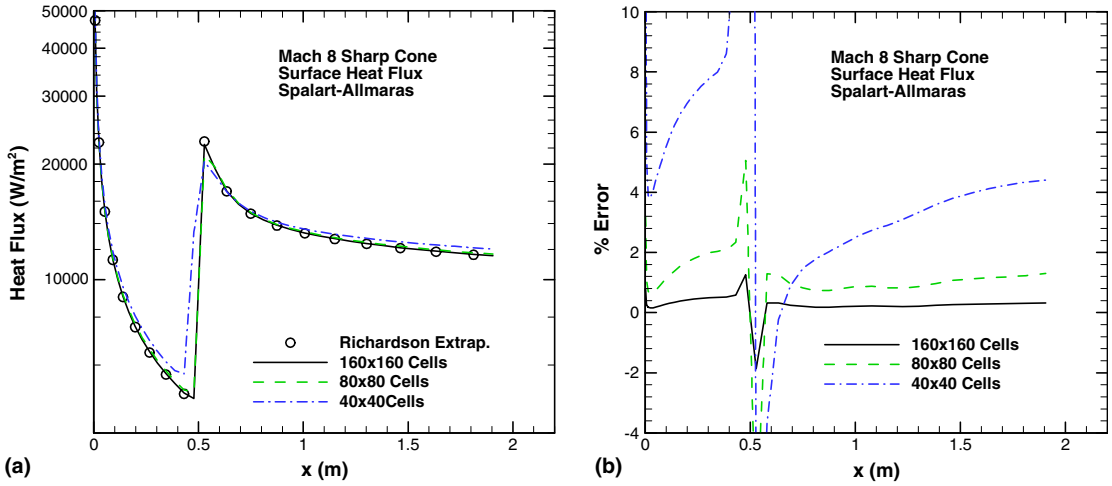

Fig. 6. (a) Surface heat flux and (b) relative discretization error for the transitional sharp cone.

an equal chance of being larger or smaller than the error estimate. See Section 3.4 for a discussion of factors of safety as they relate to confidence level.

In the asymptotic grid convergence region, the discretization error should drop as $1/r^p$. In the current case, the order of accuracy is $p = 2$ and the grid refinement factor is $r = 2$. Thus the RDE estimates should obey the following relationship:

$$\mathrm{RDE}_1 = \frac{\mathrm{RDE}_2}{4} = \frac{\mathrm{RDE}_3}{16}. \tag{23}$$

The use of Richardson extrapolation to estimate the discretization error results in the left equality in Eq. (23) being exactly satisfied. The degree to which the right equality is satisfied provides some indication of whether or not the three solutions are in the asymptotic grid convergence range. The normalized discretization errors from Eq. (23) are presented in Fig. 7. The normalized discretization error values generally agree, suggesting that the observed order of accuracy matches the formal order of accuracy ($p = 2$) everywhere except for the transition location, indicating that the three solutions are indeed asymptotic.

3.3.6. Advantages and disadvantages of extrapolation-based error estimators

Error estimators based on Richardson extrapolation have a number of advantages over other error estimation methods. The primary advantage is that they can be used with any type of discretization scheme, whether it be a finite-difference, finite-volume, or finite-element method. Extrapolation-based error estimation is also fairly easy to perform since it can be used as a post-processing step, and thus does not require modifications to the code. Finally, extrapolation-based error estimators provide estimates of the global discretization error, and do so for any quantity of interest.

There are, however, some disadvantages of using extrapolation-based error estimators. The underlying theory of Richardson extrapolation requires smooth solutions, thus reducing the effectiveness of these error estimators for problems with discontinuities or singularities. Extrapolation-based error estimators also rely on having multiple numerical solutions in the asymptotic grid convergence range, and these solutions can be expensive to compute. In addition, the extrapolation procedure tends to amplify other sources of error such as round-off and iterative convergence error [2].

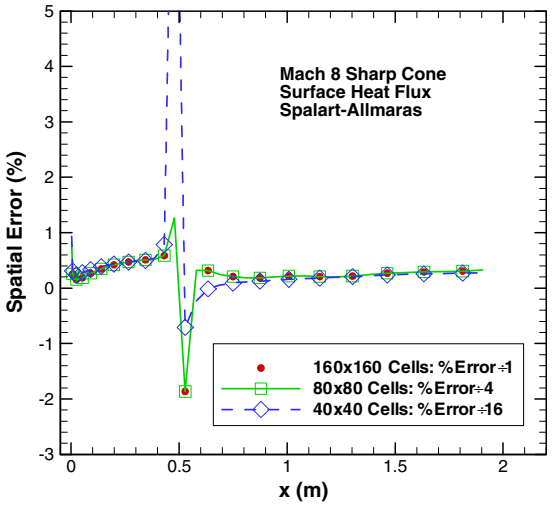

Fig. 7. Normalized relative discretization error in the surface heat flux on the transitional sharp cone.

3.4. Roache's grid convergence index

In 1994, Roache proposed the Grid Convergence Index, or GCI, as a method for uniform reporting of grid refinement studies [27]. The GCI combines the often reported relative difference between solutions (Eq. (20)) with the $(r^p - 1)$ factor from the Richardson extrapolation-based error estimator (Eq. (22)). The GCI also provides an error band rather than an error estimate.

3.4.1. Definition of the GCI

The GCI for the fine grid numerical solution is defined as

$$\mathrm{GCI} = \frac{F_s}{r^p - 1}\left|\frac{f_2 - f_1}{f_1}\right|, \tag{24}$$

where F_s is a factor of safety that is usually set to three ($F_s = 3$). Comparing the GCI to the extrapolation-based RDE estimator given in Eq. (22), we see that the GCI uses a factor of safety F_s, it employs absolute values to provide an error band, and it replaces f_{exact} in the denominator of Eq. (22) with f_1. Most importantly, the GCI correctly accounts for the (assumed) order of accuracy p and the grid refinement factor r.

3.4.2. Relationship between the GCI and a Richardson extrapolation-based error band

The relative discretization error estimate from Eq. (22) can easily be converted to an error band ($\mathrm{RDE}_{\mathrm{band}}$) by taking the absolute value and multiplying by a factor of safety F_s, resulting in

$$\mathrm{RDE}_{\mathrm{band}} = \frac{F_s}{r^p - 1}\left|\frac{f_2 - f_1}{f_{\mathrm{exact}}}\right|. \tag{25}$$

Now, the only difference between the Richardson extrapolation-based error band ($\mathrm{RDE}_{\mathrm{band}}$) and the GCI is the use of f_{exact} in the denominator rather than f_1. Will this make a significant difference? It was shown by Roy [28] that the error in the GCI relative to the $\mathrm{RDE}_{\mathrm{band}}$ is given by

$$\left|\frac{\mathrm{GCI} - \mathrm{RDE}_{\mathrm{band}}}{\mathrm{RDE}_{\mathrm{band}}}\right| = \left|\frac{1}{r^p - 1}\left(\frac{f_2 - f_1}{f_1}\right)\right| = \frac{\mathrm{GCI}}{F_s}.$$

The error in the GCI relative to the RDE_{band} is thus an ordered error, meaning that it is reduced with mesh refinement (i.e., as $h \to 0$). The range of RDE_{band} values for various values of the GCI between 3% and 21% are presented in Table 2. The two values for the RDE_{band} arise depending on whether the quantity $(f_1 - f_2)$ is positive or negative. It is clear that for reasonable values of the GCI (say below 20%), the GCI and the RDE_{band} agree quite well (within 1.5%).

3.4.3. Factor of safety in the GCI

It is important to include the factor of safety in the GCI and the RDE_{band}. Both of these error bands are based on Richardson extrapolation, and we do not know a priori whether the estimated exact solution is above or below the true exact solution to the continuum partial differential equations. Consider Fig. 8, which shows two numerical solutions (f_1 and f_2), the estimated exact solution from Richardson extrapolation (f_{RE}), and the true exact solution (f_{exact}). In general, there is an equal chance that the true exact solution is above or below the estimated value. Thus a factor of safety of $F_s = 1$ centered about the fine grid numerical solution f_1 will only provide 50% confidence that the true error (f_{exact}) is within the error band. Increasing the factor of safety should increase the confidence that the true error is within the error band.

The value for the factor of safety that would provide a 95% confidence band is currently a subject of debate. When only two numerical solutions are performed, the observed order of accuracy cannot be calculated and must be assumed. For this case, Roache [2] recommends $F_s = 3$. When three solutions are performed, the observed order of accuracy can be calculated. If the observed order matches the formal order of accuracy, Roache [2] recommends a smaller factor of safety of $F_s = 1.25$. However, when the solutions are far outside the asymptotic range, the accuracy of the extrapolation procedure is unpredictable and possibly random. In this case, no choice for the factor of safety is sure to be conservative!

Table 2
Values for the RDE_{band} for various GCI values

GCI	RDE_{band}
3%	2.97% or 3.03%
9%	8.73% or 9.27%
15%	14.25% or 15.75%
18%	16.92% or 19.08%
21%	19.53% or 22.47%

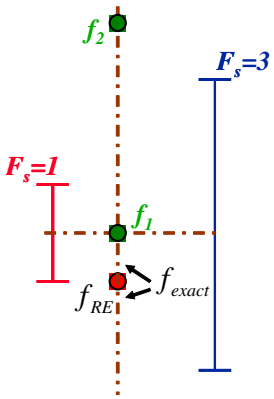

Fig. 8. Factor of safety for extrapolation-based error bands.

3.4.4. Effectiveness of the GCI

Two recent studies have carefully examined the effectiveness of the GCI. Eca and Hoekstra [29] examined a wide range of fluid flow problems using a large number of non-integer refined meshes (at least sixteen different grids for each case). They found that the GCI with a factor of safety of $F_s = 1.25$ worked well on the finer meshes. Cadafalch et al. [30] examined five test cases for fluid flow (some with heat transfer) using four to seven meshes for each case with a grid refinement factor of two. They also found good results for the GCI with a factor of safety of 1.25. In all of these studies, the observed order of accuracy was calculated locally at each point in the domain. See also Chapter 13 of [31] for additional discussion of the effectiveness of the GCI.

3.5. Practical aspects of grid refinement

This sub-section addresses practical aspects of grid refinement for realistic problems of engineering interest. When grid refinement/coarsening is used in conjunction with an extrapolation procedure, the primary requirement that must be met is that the grid refinement factor must be consistent over the entire domain (i.e., it must be constant and not a function of space or time). Topics discussed in this section include: grid refinement versus grid coarsening for both structured and unstructured meshes, non-integer grid refinement, refinement in only one coordinate direction, mesh transformations, and simulations on under-resolved meshes.

3.5.1. Grid refinement versus grid coarsening for structured meshes

In theory, it should not make a difference whether we start with the coarse mesh or the fine mesh. However, in practice, grid coarsening on structured meshes is often easier than grid refinement, especially for complex meshes. Here, complex meshes are defined as those with complex geometries and/or significant grid clustering. For uniform meshes, refinement can be performed by simply averaging neighboring spatial location. For stretched meshes, this type of refinement will lead to discontinuities in the ratio of neighboring element sizes near the original coarse grid nodes. A better strategy for stretched meshes is to use higher-order interpolation to obtain smooth stretching distributions; however, this process can be challenging on highly complex grids. The primary problems that arise during mesh refinement are due to a loss of geometric definition at object surfaces, especially at sharp corners. Furthermore, for structured grid approaches requiring point-to-point match-up at inter-zone boundaries, the refinement strategy must ensure that these points are co-located. Thus for complex, structured meshes, it is often easier to simply start with the fine mesh and successively remove every other point in each of the coordinate directions.

3.5.2. Grid refinement versus grid coarsening for unstructured meshes

For unstructured meshes, it is generally easier to start with the coarse mesh, then refine by sub-dividing the elements. This is due to the difficulties of merging elements in a manner that preserves the element type while enforcing the requirement of a constant grid refinement factor over the entire domain. While refinement for unstructured grid approaches inherits all of the drawbacks of refinement for structured grids discussed in the previous section, there are currently efforts underway to make surface geometry information directly available to mesh refinement routines [32].

The choice of methods for refining the elements will determine the effective grid refinement factor. In two dimensions, triangular elements can easily be refined by connecting the midpoints of the edges as shown in Fig. 9, thereby creating four new triangular elements. This figure also shows a similar refinement strategy for three-dimensional tetrahedra, where the midpoints of each edge are connected, eventually resulting in eight smaller tetrahedra. In both cases shown, the grid refinement factor is two.

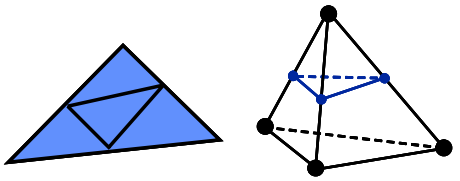

Fig. 9. Refinement strategy for unstructured meshes: (a) 2D triangles and (b) 3D tetrahedra.

3.5.3. Non-integer grid refinement

It is not necessary to use grid refinement factors of two, a process referred to as grid doubling or grid halving (depending on whether one starts with the fine mesh or the coarse mesh). For simple meshes, grid refinement factors as small as $r = 1.1$ can be employed [2]. Using non-integer grid refinement factors may increase the chance of getting all mesh solutions into the asymptotic grid convergence range. However, non-integer grid refinement factors are difficult to apply to complex meshes, especially those involving significant mesh stretching. For simulations on complex, structured meshes, the grid generation can sometimes make up the majority of the overall analysis time. Thus, relying on the original grid generation procedure for grid refinement can be expensive; furthermore, it is difficult to enforce a constant grid refinement factor over the entire domain. Higher-order interpolation can be used for non-integer grid refinement. Here it is again better to start with the fine mesh and then coarsen (at least for structured meshes); however, the same geometry definition problems discussed earlier still exist. When a grid refinement factor of two is employed, there is only significant effort involved in generating the fine mesh; the coarser meshes are found by simply removing every other point. The drawback is not only that the fine mesh may be unnecessarily expensive, but there is also an increased chance that the coarse mesh will be outside the asymptotic grid convergence range.

3.5.4. Independent coordinate refinement

It is sometimes the case that the discretization errors come primarily from just one of the coordinate directions. In such cases, it can be helpful to perform independent refinement in the coordinate directions to determine which one is the primary contributor to the overall discretization error. For independent refinement in x and y, we can write

$$f_k = f_{\text{exact}} + g_x(\Delta x_k)^p + g_y(\Delta y_k)^q + \text{HOT}, \tag{26}$$

where the error terms for each direction are included. In order to keep the analysis general, the order of accuracy in the x direction is p and the order of accuracy in the y direction is q, where the two may or may not be equal. Note that for some numerical schemes, a cross term (e.g., $g_{xy}(\Delta x)^s(\Delta y)^t$) may also be present. As in Richardson extrapolation, assume that p and q are equal to the formal order of accuracy. Consider the case of two solutions ($k = 1$ and $k = 2$) with refinement only in the x direction by a factor of r_x. As the Δx element size is refined, the term $g_y(\Delta y_k)^q$ will be constant. We are now unable to solve for the exact solution f_{exact}, but instead must solve for the quantity

$$f_{\text{exact},x} = f_{\text{exact}} + g_y(\Delta y_k)^q,$$

which includes the error term due to the Δy discretization. Neglecting higher-order terms, the following two equations

$$f_1 = f_{\text{exact},x} + g_x(\Delta x)^p,$$
$$f_2 = f_{\text{exact},x} + g_x(r_x\Delta x)^p,$$

can be solved for $f_{\text{exact},x}$

$$f_{\text{exact},x} = f_1 + \frac{f_1 - f_2}{r_x^p - 1},$$

and the leading x-direction error term

$$g_x(\Delta x)^p = \frac{f_2 - f_1}{r_x^p - 1}. \tag{27}$$

Similarly, introducing a third solution ($k = 3$) with coarsening only in the y direction allows us to solve for the y-direction error term

$$g_y(\Delta y)^q = \frac{f_3 - f_1}{r_y^q - 1}. \tag{28}$$

The size of the two error terms from Eqs. (27) and (28) can then be compared to determine the appropriate direction for further mesh refinement.

3.5.5. Mesh transformations

Most discretization approaches are developed for uniform meshes (Δx, Δy, Δz, Δt = constant). For general geometries, some form of transformation must be applied. In the finite-difference method, global transformations are used which actually modify the governing equations. For finite-element methods, local transformations are employed centered about each element. Finite-volume methods can employ either local or global transformations. In any case, if the transformation is analytical, no additional numerical errors are introduced. However, when discrete transformations are employed, an ordered error is added to the numerical solution. Discrete transformations should be of the same order as (or higher than) the underlying numerical scheme. Mistakes in the transformations will introduce errors into the numerical solutions which can adversely affect the order of accuracy.

3.5.6. Simulations on under-resolved meshes

For computational simulations where the physics and/or geometry are complex (especially time-dependent, three-dimensional cases) the meshes are often under-resolved due to computational resource limitations. By under-resolved we mean that the simulations are not in the asymptotic range where the observed order of accuracy matches the formal order. The results of such simulations should be used with caution since all methods for estimating the discretization error rely on the solutions being asymptotic. As discussed earlier in Section 3.4.3, discretization error estimates for solutions outside the asymptotic range are generally unreliable, and the behavior of the error estimator is possibly random due to the influence of the higher-order terms.

A classic example of an under-resolved mesh from fluid mechanics is a flow containing a shock-wave discontinuity. When the Navier–Stokes equations are used, it is possible to refine the mesh enough to resolve the viscous (i.e., frictional) dissipation in the region of the shock wave and thus reach the asymptotic convergence range. However, such high resolution is rarely practical due to the large number of mesh points required, especially for multi-dimensional flows. In the case where viscous effects are omitted (i.e., the Euler equations), the shock wave will result in discontinuities in the primitive flowfield variables. Recent work by Carpenter and Casper [33] suggests that for the Euler equations on sufficiently refined meshes, the simulation of flows containing shock waves will reduce down to first-order accuracy, regardless of the formal order of the numerical scheme employed.

4. Summary and conclusions

Verification deals with the mathematical correctness of the solution to a given set of equations. The appropriateness of the equations to provide accurate predictions of reality (i.e., experimental data) falls

under the heading of validation and is not addressed in the present work. This paper specifically deals with the numerical solution to partial differential equations, termed computational simulation. While most of the examples given are from fluid dynamics and heat transfer, the concepts can be equally applied to any computational simulation discipline (e.g., structural mechanics, electrodynamics, astrophysics).

The first aspect of verification addressed was code verification, a process designed to find mistakes in the computer code. The most general approach to code verification is the method of manufactured solutions which provides a procedure for generating exact solutions to a modified set of governing equations. The generated exact solution (or manufactured solution) can then be combined with order of accuracy verification, resulting in a highly efficient process for finding coding mistakes.

Solution verification is concerned with assessing the numerical errors that exist in every computational simulation: round-off error, iterative convergence error, and discretization error. Discretization error is generally the dominant source of numerical error in computational simulation, and significant research is ongoing in a posteriori error estimation methods. This paper focuses primarily on the extrapolation-based error estimators as they are the most general error estimation approach, equally applicable to finite-difference, finite-volume, and finite-element methods. The underlying theory for extrapolation-based error estimation is based on the Richardson extrapolation technique for taking numerical solutions on two different meshes and extrapolating to a higher-order accurate solution. The basic concept can be easily extended to allow the calculation of the observed order of accuracy. Further developments, such as Roache's Grid Convergence Index, provide for the estimation of an error band rather than an error estimate. Error bands are an important building block when considering the overall uncertainty (or predictability) of computational simulation results.

Acknowledgements

The author would like to thank William Oberkampf, Frederick Blottner, James Stewart, and Matthew Hopkins of Sandia National Laboratories and Patrick Roache (consultant) for their many invaluable discussions of code and solution verification.

References

[1] Guide for the verification and validation of computational fluid dynamics simulations, American Institute of Aeronautics and Astronautics, AIAA G-077-1998, Reston, VA, 1998.
[2] P.J. Roache, Verification and Validation in Computational Science and Engineering, Hermosa Publishers, New Mexico, 1998.
[3] P. Knupp, K. Salari, in: K.H. Rosen (Ed.), Verification of Computer Codes in Computational Science and Engineering, Chapman and Hall/CRC, Boca Raton, FL, 2003.
[4] M.J. Christensen, R.H. Thayer, The Project Manager's Guide to Software Engineering's Best Practices, IEEE Computer Society, Los Alamitos, CA, 2001.
[5] G.N. Purdy, CVS Pocket Reference, second ed., O'Reilly and Associates, 2003.
[6] D.A. Anderson, J.C. Tannehill, R.H. Pletcher, Computational Fluid Mechanics and Heat Transfer, Hemisphere Publishing Corp, New York, 1984, pp. 70–77.
[7] P.J. Roache, S. Steinberg, Symbolic manipulation and computational fluid dynamics, AIAA J. 22 (10) (1984) 1390–1394.
[8] P.J. Roache, P.M. Knupp, S. Steinberg, R.L. Blaine, Experience with benchmark test cases for groundwater flow, in: I. Celik, C.J. Freitas (Eds.), Benchmark Test Cases for Computational Fluid Dynamics, ASME FED 93, Book No. H00598, 1990, pp. 49–56.
[9] W.L. Oberkampf, F.G. Blottner, Issues in computational fluid dynamics code verification and validation, AIAA J. 36 (5) (1998) 687–695 (see also W.L. Oberkampf, F.G. Blottner, D.P. Aeschliman, Methodology for computational fluid dynamics code verification/validation, AIAA Paper 95-2226, 1995).
[10] K. Salari, P. Knupp, Code Verification by the Method of Manufactured Solutions, SAND 2000-1444, Sandia National Laboratories, Albuquerque, NM, 2000.

[11] P.J. Roache, Code verification by the method of manufactured solutions, J. Fluids Eng. 124 (1) (2002) 4–10.
[12] C.J. Roy, C.C. Nelson, T.M. Smith, C.C. Ober, Verification of Euler/Navier–Stokes codes using the method of manufactured solutions, Int. J. Numer. Meth. Fluids 44 (6) (2004) 599–620.
[13] T.G. Trucano, M.M. Pilch, W.L. Oberkampf, "On the Role of Code Comparisons in Verification and Validation," SAND 2003-2752, Sandia National Laboratories, Albuquerque, NM, 2003.
[14] C.J. Roy, F.G. Blottner, Assessment of one- and two-equation turbulence models for hypersonic transitional flows, J. Spacecraft Rockets 38 (5) (2001) 699–710.
[15] C.J. Roy, F.G. Blottner, Methodology for turbulence model validation: application to hypersonic flows, J. Spacecraft Rockets 40 (3) (2003) 313–325.
[16] J.H. Ferziger, M. Peric, Further discussion of numerical errors in CFD, Int. J. Numer. Meth. Fluids 23 (12) (1996) 1263–1274.
[17] J.R. Stewart, A posteriori error estimation for predictive models, Presentation at the Workshop on the Elements of Predictability, Johns Hopkins University, Baltimore, Maryland, November 13–14, 2003.
[18] O.C. Zienkiewicz, J.Z. Zhu, The superconvergent patch recovery and a posteriori error estimates, part 2: error estimates and adaptivity, Int. J. Numer. Meth. Eng. 33 (1992) 1365–1382.
[19] Z. Zhang, A. Naga, A Meshless Gradient Recovery Method, Part I: Superconvergence Property, Research Report #2, Department of Mathematics, Wayne State University, 2002.
[20] K. Eriksson, C. Johnson, Error estimates and automatic time step control for nonlinear parabolic problems, SIAM J. Numer. Anal. 24 (1) (1987) 12–23.
[21] I. Babuska, A. Miller, Post-processing approach in the finite element method. Part 3: a posteriori error estimates and adaptive mesh selection, Int. J. Numer. Meth. Eng. 20 (12) (1984) 2311–2324.
[22] T.J. Barth, M.G. Larson, A-posteriori error estimation for higher order godunov finite volume methods on unstructured meshes, in: R. Herbin, D. Kroner (Eds.), Finite Volumes for Complex Applications III, HERMES Science Publishing Ltd, London, 2002, pp. 41–63.
[23] M. Ainsworth, J.T. Oden, A Posteriori Error Estimation in Finite Element Analysis, Wiley, New York, 2000.
[24] I. Babuska, Accuracy Estimates and Adaptive Refinements in Finite Element Computations, Wiley, New York, 1986.
[25] L.F. Richardson, The approximate arithmetical solution by finite differences of physical problems involving differential equations with an application to the stresses in a masonry dam, Trans. Royal Society London, Ser. A 210 (1910) 307–357.
[26] L.F. Richardson, The deferred approach to the limit, Trans. Royal Society London, Ser. A 226 (1927) 229–361.
[27] P.J. Roache, Perspective: a method for uniform reporting of grid refinement studies, J. Fluids Eng. 116 (1994) 405–413.
[28] C.J. Roy, Grid convergence error analysis for mixed-order numerical schemes, AIAA Paper 2001-2606, 2001.
[29] L. Eca, M. Hoekstra, An evaluation of verification procedures for CFD applications, in: 24th Symposium on Naval Hydrodynamics, Fukuoka, Japan, July 8–13, 2002.
[30] J. Cadafalch, C.D. Perez-Segarra, R. Consul, A. Oliva, Verification of finite volume computations on steady-state fluid flow and heat transfer, J. Fluids Eng. 24 (2002) 11–21.
[31] D. Pelletier, P.J. Roache, Verification and validation of computational heat transfer (Chapter 13), in: W.J. Minkowycz, E.M. Sparrow, J.Y. Murthy (Eds.), Handbook of Numerical Heat Transfer, 2nd Ed., John Wiley and Sons, Inc., Hoboken, NJ, 2005.
[32] T.J. Tautges, CGM: a geometry interface for mesh generation, analysis and other application, Eng. Computers 17 (3) (2001) 299–314.
[33] M.H. Carpenter, J.H. Casper, Accuracy of shock capturing in two spatial dimensions, AIAA J. 37 (9) (1999) 1072–1079.