Code Verification by the Method of Manufactured Solutions

Patrick J. Roache
Consultant,
P.O. Box 3379,
Los Lunas, NM 87031
e-mail: hermosa@swcp.com

Verification of Calculations involves error estimation, whereas Verification of Codes involves error evaluation, from known benchmark solutions. The best benchmarks are exact analytical solutions with sufficiently complex solution structure; they need not be realistic since Verification is a purely mathematical exercise. The Method of Manufactured Solutions (MMS) provides a straightforward and quite general procedure for generating such solutions. For complex codes, the method utilizes Symbolic Manipulation, but here it is illustrated with simple examples. When used with systematic grid refinement studies, which are remarkably sensitive, MMS produces strong Code Verifications with a theorem-like quality and a clearly defined completion point. [DOI: 10.1115/1.1436090]

Introduction

In the semantic tangle of the subject of ''Quantification of Uncertainty'' (a term which itself generates some disagreement) the three most important terms, and the most universally agreed upon, are Verification of Codes, Verification of Calculations, and Validation. For reasons both logical and practical, these activities must be performed in this order [1,2]. Verification of a Calculation involves error *estimation*, whereas Verification of a Code involves error *evaluation*, from a known solution. Both Verifications are purely mathematical activities, with no concern whatever for the accuracy of physical laws. That is the concern of Validation, i.e., the agreement of the mathematics with science.

Journal Policy Statements on reporting of numerical uncertainty, of which this journal's 1986 statement [3] was the original, refer only to Verification of Calculations; the code used is assumed to be correct. ''Correct'' is perhaps preferable to ''accurate.'' It can be misleading to describe a code as ''accurate,'' because naive users of commercial software may think that, if the code they use is accurate, then their calculation will be accurate. This neglects their own burden to perform systematic discretization convergence tests for their particular calculation, i.e., Verification of a Calculation. Determining the correctness of the code itself can only be done by systematic discretization convergence tests using a known solution or ''benchmark'' (another term with inconsistent connotations). The best benchmark solution or standard of comparison is an exact analytical solution, i.e. a solution expressed in simple primitive functions like sin, exp, tanh, etc. Note that benchmark solutions involving infinite series are not desirable, typically being more numerical trouble to evaluate accurately than the CFD code itself [1]. It is not sufficient that the analytical solution be exact; it is also necessary that the solution structure be sufficiently complex that all terms in the governing equation being tested are exercised. For example, some early and misleading claims of accuracy of commercial codes which used the notoriously inaccurate first-order upstream differencing for advection terms were based on comparisons with Poiseuille, Couette or Rayleigh problems, which do not even ''turn on'' the advection terms.

It has often been stated in research journal articles that general accuracy Verification of Codes for difficult problems, e.g. the full Navier-Stokes equations of fluid dynamics, is not possible because exact solutions exist only for such relatively simple problems that do not fully exercise the code. Many papers and reports approach accuracy Verification of Codes in a haphazard and piecemeal way, comparing single-grid results for a few exact solutions on problems of reduced complexity. In fact, a very general procedure exists for generating analytical solutions for accuracy Verification of Codes. I first presented the method in [4], and later expanded the applications [1,2]. Although a few respected authorities (e.g. [5–9]) have recognized the power of the method, acceptance has been slow and misunderstanding is not uncommon. Based on my experience in many discussions with professional colleagues including teaching short courses with participations by senior researchers, the misunderstanding is due to the deceptive simplicity (elegance?) of the concept. This article is written in an attempt to clarify the concepts with simple examples, to dispel concerns often voiced, to add a few fine points, and to provide some recent references. It is hoped that the reader will bear with the somewhat conversational style, since the paper is part tutorial, part review.

The methodology provides for convincing, rigorous Verification of the numerical accuracy of a code via systematic grid convergence testing. This procedure is straightforward though somewhat tedious to apply, and verifies all accuracy aspects of the code: formulation of the discrete equations (interior and boundary conditions) and their order of accuracy, the accuracy of the solution procedure, and the user instructions.

The Method of Manufactured Solutions

The Method of Manufactured Solutions (MMS) provides a general procedure for generating an analytical solution for code accuracy verification.

The basic idea of the procedure is to simply manufacture an exact solution, without being concerned about its physical realism. (The ''realism'' or lack thereof has nothing to do with the mathematics, and Verification is a purely mathematical exercise.) In the original, most straightforward and most universally applicable version of the method, one simply includes in the code a general source term, $Q(x,y,z,t)$ and uses it to generate a nontrivial but known solution structure. We follow the counsel of G. Polya [10]: *Only a fool starts at the beginning; the wise one starts at the end.*

We first pick a continuum solution. Interestingly enough, we can pick a solution virtually independent of the code or of the hosted equations. That is, we can pick a solution, then use it to verify an incompressible Navier-Stokes code, a Darcy flow in porous media code, a heat conduction code, an electrode design code, a materials code, etc.

We want a solution that is non-trivial but analytic, and that exercises all ordered derivatives in the error expansion and all terms, e.g., cross-derivative terms. For example, chose a solution involving tanh. This solution also defines boundary conditions, to be applied in any (all) forms, i.e., Dirichlet, Neumann, Robin, etc. Then the solution is passed through the governing PDE's to give the production term $Q(x,y,z,t)$ that produces this solution. Since this description sounds circular, we will demonstrate with con-

Contributed by the Fluids Engineering Division for publication in the JOURNAL OF FLUIDS ENGINEERING. Manuscript received by the Fluids Engineering Division July 24, 2001; revised manuscript received November 12, 2001. Associate Editor: G. Karniadakis.

crete examples. In [4] we used Symbolic Manipulation to generate Q, and this is still recommended for complex multidimensional CFD codes. However, for illustration purposes, we can consider simple one-dimensional transient problems and generate the results by hand, in unambiguous steps.

Three Example Problems in MMS. To emphasize the generality of the concept, we pick the first example solution *before we specify the governing equations*. Then we will use this same solution for two different problems, i.e., set of governing PDE's and boundary conditions. The chosen solution $U(t,x)$ is the following.

$$U(t,x)=A+\sin(B), \quad B=x+Ct \tag{1}$$

Example 1. First, let us apply this 1-D transient solution to the nonlinear Burgers equation, often taken as a model for CFD algorithm development [2].

$$u_t=-uu_x+\alpha u_{xx} \tag{2}$$

Incidentally, this specified solution $U(t,x)$ is the exact solution for the constant velocity advection equation with boundary condition of $u(t,0)=A+\sin(Ct)$, so for the high Reynolds number problem (small α) it may look "realistic" in some sense, but it is not a solution to our governing Eq. (2), and its "realism" or lack thereof is irrelevant to the task of Code Verification.

We determine the source term $Q(t,x)$ which, when added to the Burgers equation for $u(t,x)$, produces the solution $u(t,x)=U(t,x)$. We write the Burgers equation as an operator (nonlinear) of u,

$$L(u)\equiv u_t+uu_x-\alpha u_{xx}=0 \tag{3}$$

Then we evaluate the Q that produces U by operating on U with L.

$$Q(t,x)=L(U(t,x))=\partial U/\partial t+U\partial U/\partial x-\alpha\partial^2 U/\partial x^2 \tag{4}$$

By elementary operations on the manufactured solution $U(t,x)$ stated in Eq. (1),

$$Q(t,x)=C\cos(B)+[A+\sin(B)]\cos(B)+\alpha\sin(B) \tag{5}$$

If we now solve the modified equation

$$L(u)\equiv u_t+uu_x-\alpha u_{xx}=Q(t,x) \tag{6}$$

or

$$u_t=-uu_x+\alpha u_{xx}+Q(t,x) \tag{7}$$

with compatible initial and boundary conditions, the exact solution will be $U(t,x)$ given by Eq. (1).

The initial conditions are obviously just $u(0,x)=U(0,x)$ everywhere. The boundary conditions are determined from the manufactured solution $U(t,x)$ of Eq. (1). Note that we have not even specified the domain of the solution as yet. If we want to consider the usual model $0\leqslant x\leqslant 1$ or something like $-10\leqslant x\leqslant 100$, the same solution Eq. (1) applies, but of course the boundary values are determined at the corresponding locations in x. Note also that we have not even specified the *type* of boundary condition as yet. This aspect of the methodology has often caused confusion. Everyone knows that different boundary conditions on a PDE produce different answers; not everyone recognizes immediately that the same solution $U(t,x)$ can be produced by more than one set of boundary condition types. The following combinations of inflow (left boundary, e.g., $x=0$) or outflow (e.g., $x=1$) boundary conditions will produce the same solution $U(t,x)$ over the domain $0\leqslant x\leqslant 1$.

Dirichlet—Dirichlet:

$$u(t,0)=U(t,0)=A+\sin(Ct),u(t,1)=A+\sin(1+Ct) \tag{8}$$

Dirichlet—Outflow Gradient (Neumann):

$$u(t,0)=U(t,0)=A+\sin(Ct),\partial u/\partial x|(t,1)=\cos(1+Ct) \tag{9}$$

Robin (mixed)—Outflow Gradient (Neumann) at $x=\pi$:

$$\begin{aligned} au+bu_x=c \;\; at \;\; (t,0)\rightarrow given \;\; a \;\; and \;\; b, \;\; select \\ =c[A+\sin(Ct)]+b\cos(Ct) \\ \partial u/\partial x|(t,\pi)=\cos(\pi+Ct) \end{aligned} \tag{10}$$

For this time-dependent solution, the boundary values are time-dependent. It also will be possible to manufacture time-dependent solutions with steady boundary values, if required by the code.

Example 2. To further clarify the concept, we now apply the same solution to a different problem, choosing as the new governing PDE a Burgers-like equation that might be a candidate for a 1-D turbulence formulation based on the mixing length concept.

$$\begin{aligned} u_t&=-uu_x+\alpha u_{xx}+\lambda\partial/\partial x[(x\partial u/\partial x)^2] \\ &=-uu_x+\alpha u_{xx}+2\lambda[x(u_x)^2+x^2u_{xx}] \end{aligned} \tag{11}$$

Writing the mixing-length model equation as a nonlinear operator of u,

$$L(u)\equiv u_t+uu_x-\alpha u_{xx}-2\lambda[x(u_x)^2+x^2u_{xx}]=0 \tag{12}$$

we evaluate the Q_m that produces U by operating on U with L_m.

$$\begin{aligned} Q_m(t,x)=L_m(U(t,x))=\partial U/\partial t+U\partial U/\partial x-\alpha\partial^2U/\partial x^2 \\ -2\lambda[x(\partial U/\partial x)^2+x^2\partial^2U/\partial x^2] \end{aligned} \tag{13}$$

By elementary operations on the (same) manufactured solution $U(t,x)$ stated in Eq. (1),

$$\begin{aligned} Q(t,x)=C\cos(B)+[A+\sin(B)]\cos(B)+\alpha\sin(B) \\ -2\lambda[x\cos^2(B)-x^2\sin(B)] \end{aligned} \tag{14}$$

If we now solve the modified model equation

$$L_m(u)\equiv u_t+uu_x-\alpha u_{xx}-2\lambda[x(u_x)^2+x^2u_{xx}]=Q_m(t,x) \tag{15}$$

or

$$u_t=-uu_x+\alpha u_{xx}+2\lambda[x(u_x)^2+x^2u_{xx}]+Q_m(t,x) \tag{16}$$

with compatible initial and boundary conditions, the exact solution for this turbulent problem again will be $U(t,x)$ given by Eq. (1), as it was for the previous laminar problem.

Note: the same initial and boundary conditions and boundary values from the previous problem can apply, since these are determined from the solution, not from the governing PDE, nor from Q or Q_m.

Example 3. We have shown how the same solution can be used as the exact solution to verify two different codes with different governing equations, with different source terms being created to Manufacture the same solution. A third example will demonstrate the arbitrariness of the solution form. Rather than the somewhat "realistic" solution to the constant velocity advection equation given by Eq. (1), let us consider the "unrealistic" but equally valuable solution as follows.

$$U_e(t,x)=\sin(t)e^x \tag{17}$$

Following the same procedure for the Burgers Equation (2), we evaluate the terms in Eq. (4) from the solution U_e of Eq. (17) and obtain

$$Q_e(t,x)=\cos(t)e^x+[\sin(t)e^x]^2-\alpha\sin(t)e^x \tag{18}$$

(arranged for readability rather than compactness). This Q_e, when added to Eq. (2), produces the Manufactured Solution Eq. (17) when compatible initial and boundary conditions are evaluated from Eq. (17).

Application to Verification of Codes. Once a nontrivial exact analytic solution has been generated, by this Method of Manufactured Solutions or perhaps another method, the solution is now used to Verify a Code by performing systematic discretization convergence tests (usually, grid convergence tests) and monitoring the convergence as $\Delta\rightarrow 0$, where Δ is a measure of discretization (e.g., Δx, Δt in a finite difference or finite volume code, and element size in a finite element code, etc.).

The principle definition of ''order of convergence'' is based on behavior of the error of the discrete solution. There are various measures of error, but in some sense we are always referring to the difference between the discrete solution $f(\Delta)$ (or a functional of the solution, such as lift coefficient) and the exact (continuum) solution,

$$E=f(\Delta)-f^{\text{exact}} \tag{19}$$

For an order p method, and for a well-behaved problem (exceptions are discussed in Chapters 6 and 8 of [1]), the error in the solution E asymptotically will be proportional to Δ^p. This terminology applies to every ''consistent'' methodology: finite difference methods (FDM), finite volume methods (FVM), finite element methods (FEM), spectral, pseudo-spectral, vortex-in-cell, etc., regardless of solution smoothness. Thus,

$$E=f(\Delta)-f^{\text{exact}}=C\Delta^p+\text{H.O.T.} \tag{20}$$

where H.O.T. are higher order terms. (For smooth problems, it may be possible in principle to evaluate the coefficient C and the H.O.T. from the continuum solution, but as a practical matter, we do not do this in the accuracy Verification procedure.) We then monitor the numerical error as the grid is systematically refined. Successive grid halving is not required, just refinement. (See [1] for examples, analysis and extensive discussion.) Thorough iterative convergence is required. Theoretically (from Eq. 20), values of $C=error/\Delta^p$ should become constant as the grid is refined for a uniformly p-th order method (''uniformly'' implying at all points for all derivatives). Details and many examples are given in [1].

The following summary points from [1] are worth noting.

The procedure detects all ordered errors. It will not detect coding mistakes that do not affect the answer obtained, e.g. mistakes in an iterative solution routine which affect only the iterative convergence rate. In the present view, these mistakes are not considered as Code Verification issues, since they affect only code efficiency, not accuracy.

The procedure does not evaluate the adequacy of non-ordered approximations, e.g., distance to an outflow boundary, distance to an outer (wind-tunnel wall-like) boundary, use of $\partial p/\partial y=0$ at a wall as a boundary condition (this is not a rigorous physical boundary condition for Navier-Stokes equations). The errors of these approximations do not vanish as $\Delta\rightarrow 0$, hence are ''non-ordered approximations.'' The adequacy of these approximations must be assessed by sensitivity tests which may be described as ''Justification'' exercises [1]; these are similar to Verification of Calculations in that they involve only mathematics, but are simply the results of calculations. If the code manual says it uses a 2nd order accurate discretization of $\partial p/\partial y=0$ at walls and the MMS procedure shows that it does, then the Code is Verified on this point.

When this systematic grid convergence test is verified (for all point-by-point values), we have verified

1. any equation transformations used (e.g., nonorthogonal boundary fitted coordinates),
2. the order of the discretization,
3. the encoding of the discretization, and
4. the matrix solution procedure.

This technique was originally applied in [4] to long Fortran code produced by Artificial Intelligence (Symbol Manipulation) methods. The original 3-D nonorthogonal coordinate code contained about 1800 lines of dense Fortran. It would be impossible to check this by reading the source code, yet the procedure described Verified the code convincingly. Roundoff error was not a problem.

The arbitrary solution, produced inversely by the specification of the source term Q, has been aptly described by Oberkampf et al. [5] and Reed et al. [9] as a ''Manufactured Solution.'' The approach was independently developed and named the ''Prescribed Solution Forcing Method'' by Dee [11]. Others who independently developed the same philosophy and essentially the same methodology are Ethier and Steinman [12] and Powers [13–16]. The first systematic exposition of the method with application to multidimensional nonlinear problems appears to be [4], but in retrospect, it seems that early instances of the use of what we now call the Method of Manufactured Solutions were cited in 1972 (the original version of [2], pp. 363–365). Although the authors did not mention the method they used, it seems clear that they used this approach to generate an *ad hoc* exact solution for time-dependent model equations. Obviously, the simple solution form was chosen first, then passed through the PDE to generate the problem; see the ''Errata and Addenda'' section of the website www.hermosa-pub.com/hermosa for references and details. Undoubtedly, many of the non-infinite-series classical solutions in engineering were obtained this way, i.e. beginning with a solution form. What is strange is that the notion persisted, often repeated, that we did not have any non-trivial solutions to the full nonlinear Navier-Stokes equations, when all we have to do is ''complicate'' the problem a little with the addition of a source term, and we can generate all the solutions we want. The key concept is that, for Verification of Codes, these solutions need not be physically realistic.

The technique is applicable to systems of equations, including full Navier-Stokes in general non-orthogonal coordinates (e.g., see [17,18]), provided that the code is capable (or modifiable) to treat source terms in each PDE.

The technique of Code Verification by monitoring grid convergence is extremely powerful. Upon initial exposure to the technique, engineers are often negative about the method because they intuit that it cannot be sensitive enough to pick up subtle errors. After exposure to numerous examples, if they remain negative it is usually because the method is *excessively* sensitive, revealing minor inconsistencies such as first-order discretizations at a single boundary point in an elliptic problem that effects the size of the error very little (as correctly intuited) but still reduces the rate of convergence to first order for the entire solution. For examples, see [1].

The fact that the Manufactured Solution may bear no relation to any physical problem does not affect the rigor of the accuracy Verification of Codes. The only important point is that the solution (manufactured or otherwise) be non-trivial, i.e., that it exercise all the terms in the error expansion. The algebraic complexity may be something of a difficulty, but is not insurmountable, and in practice has been easily handled using Symbolic Manipulation packages like Macsyma, Mathematica, Maple, etc. Using the source-code (Fortran) writing capability of Macsyma, it is not even necessary for the analyst to look at the form of Q. Rather, the specification of the solution (e.g., tanh function) to the Symbolic Manipulation code results in some complicated analytical expression that can be directly converted by the Symbolic Manipulation code to a Fortran (or Pascal, C, etc.) source code segment, which is then readily emplaced in a source code module (subroutine, function, etc.) that then is called in the accuracy Verification of Code procedure. (This ''emplacement'' can be performed by hand by the analyst, without actually reading the complicated source code expressions, or can itself be automated in the Symbolic Manipulation code.)

The procedure has been applied successfully to nonlinear systems of equations, with separate Q's generated for each equation. Both unsteady and steady solutions are possible. (It may be useful to avoid exponential solution growth in time so as to avoid confusion with instabilities; e.g., see the fully 3D incompressible Navier-Stokes analytical solutions of Ethier and Steinman [12].) Nonlinearity is an issue only because of uniqueness questions;

otherwise, the source term complexity may be worse because of nonlinearity, but that is the job of the Symbolic Manipulation code. Nonuniqueness could be an issue because the code could converge to another legitimate solution other than the Manufactured Solution, producing a false-negative accuracy Verification test for a correct code. However, it would be difficult to contrive a situation in which a false positive accuracy Verification was obtained. In much experience, non-uniqueness has not been an issue. In [4], we applied the procedure to the nonlinear (quasi-linear) PDE's of the elliptic grid generation method for nonorthogonal coordinates. Here, the Manufactured Solution was an analytical 3-D coordinate transformation; see examples in [1].

While the simple example problems herein were chosen for transparency, complex nonlinear systems (like Navier-Stokes equations in non-orthogonal coordinates) benefit from use of computer Symbolic Manipulation routines to perform the differentiation and algebra which generate the source term. As noted above, in this approach it is not necessary to even examine the source term; using the Fortran or C code writing capabilities of software packages like Macsyma, Maple, etc. a subroutine can be produced to generate the pointwise values of the source terms for inclusion in the governing PDE's. For coupled nonlinear PDE's like those of 3-D elliptic grid generation equations, the pointwise evaluation requires simultaneous solution of 3 coupled (non-dimensional) nonlinear equations at each point. We have always used full Newton-Raphson iteration methods, the Jacobians of which are also produced by Symbol Manipulation and Fortran source code writing, so the process remains automated; i.e., one never performs any algebra or calculus manipulations by hand. In fact, for our work in grid generation via variational methods [19–24], we never even looked at the governing PDE's themselves. We considered only the variational principle itself. The Symbolic Manipulation toolkits developed by Prof. Steinberg were used to automatically generate the PDE's by (symbolic) differentiation of the variational equations to produce the Euler-Lagrange equations, to substitute 2nd order difference expressions for the PDE's, to gather terms, to write Fortran subroutines for their evaluation, to generate a specified Manufactured Solution (i.e., ''a continuum grid'' or parameterization which, when discrete values are evaluated, produces a computational grid), to write Fortran code for the source term including Newton-Raphson point solutions, and to perform the entire Code Verification procedure, without the researcher ever having to look at either the continuum or discretized PDE's or source terms.

Note that the Manufactured Solution should be generated in original (''physical space'') coordinates (x,y,z,t). Then the same solution can be used directly with various non-orthogonal grids or coordinate transformations.

The only disadvantage of the procedure is the requirement that the CFD code being Verified must include accurate treatment of a source term and that the code's boundary condition values not be hard-wired. Many codes are built with source terms included, and many algorithms allow trivial extension to include Q's. However, in directionally-split algorithms such as Approximate Factorization [2] the time-accurate treatment of $Q(x,y,z,t)$ involves subtleties and complexities at boundaries, especially for non-orthogonal coordinates [2,18]. Thus, CFD code extensions may be required in order to apply this procedure involving ''Manufactured Solutions'' for Code Verification. Likewise, some groundwater flow codes are built with hard-wired homogeneous Neumann boundary conditions, $\partial f/\partial n=0$. In order to use an arbitrary solution function, nonhomogeneous boundary values like $\partial f/\partial n=g$ would be required. Alternately, one could restrict the choice of Manufactured Solution functions to fit the hard-wired values.

Likewise, to test periodic boundary conditions, one must chose a periodic function for the Manufactured Solution.

The paper by Pelletier and Ignat [8] (see also [1], pp. 162–163) will be of interest to turbulence modelers interested in Code Verification. It provides simple analytical solutions for an incompressible free shear layer applicable to $k-\varepsilon$, $k-\omega$ and $k-\tau$ models.

The practical difficulties arising from large numbers of option combinations are discussed extensively in [1]. Briefly, option combinations are countable, and pessimistic computer science conclusions about complex codes being unverifiable are based on unrealistic conditions like ''arbitrary complexity.'' Furthermore, the number of option combinations required often can be greatly reduced by ''partitioning the option matrix'' [1] based on common sense and knowledge of code structure (a ''glass box'' philosophy [7] as opposed to the more demanding ''black box'' philosophy). Failing this, codes can be Verified only for a subset of option combinations. In any case, these issues are an essential part of Code Verification by any method; they are not unique to the MMS, and in fact the generality of the MMS approach will reduce the difficulties arising from option complexity because less testing will be required for each option combination compared to the usual haphazard and piecemeal approach to Code Verification.

Also see [1] for the following topics: early applications of MMS concepts, discussions and examples of mixed first- and second-order differencing, the small parameter (high Reynolds number) problem, economics of dimensionality, applications of MMS to 3D grid generation codes, effects of strong and inappropriate coordinate stretching, debugging with Manufactured Solutions (when the Code Verification initial result is negative), examples of many manufactured or otherwise contrived analytical solutions in the literature, approximate but highly accurate solutions (often obtained by perturbation methods) that can also be utilized in Code Verification, the possibility of a useful theorem related to MMS, special considerations required for turbulence modeling and other fields with multiple scales, example of MMS Code Verification with a 3-D grid-tracked moving free surface (see [17]), code robustness, examples of the remarkable sensitivity of Code Verification via systematic grid convergence testing, and several methodologies for Verification of Calculations, including the recommended use of the Grid Convergence Index (GCI) for uniform reporting of systematic grid convergence studies.

Recent Work and Further Discussion

Blind Study. Salari and Knupp [17] have exercised the MMS in a blind study, in which one author (Knupp) modified a CFD code previously developed and Verified by the other (Salari), deliberately introducing errors. Then the code author tested the sabotaged code with the MMS. This exercise was not performed on simple model problems, but on a full time-dependent, compressible and incompressible, Navier-Stokes code with plenty of options. In all, 21 cases were studied, including one ''placebo'' (no mistake introduced) and several that involved something other than the solution (e.g., wrong time step, post-processing errors). Several formal mistakes (not order-of-convergence errors) went undetected, as expected.

Two cases showed possible limitations or cautions of MMS. Case E.4 involved an error in a DO loop for updating density arrays. Although MMS was successful, it would not have been if my suggestion (on p. 78 of [1]) had been followed to use exact continuum solutions as the initial conditions to reduce run time. (This is a caution note not just for the MMS but for any Code Verification by systematic grid convergence testing using any benchmark solution.) Also, Case E.12 showed that an error in a convergence test of one variable (a ''.le.'' test replaced with a ''.ge.'' test) could go undetected on a particular problem because the convergence test was successfully implemented for another variable.

All ten of the OAM (Order-of-Accuracy Mistake) errors, i.e., all that could prevent the governing equations from being correctly solved, were successfully detected. In addition, several less serious mistakes were detected using the procedure.

The report also discusses error (and mistake) taxonomies, provides examples and Manufactured Solutions (with source terms) from compressible Navier-Stokes codes as well as heat conduction and 2-D Burgers equation codes in both Cartesian and curvilinear coordinates, and discusses approaches for developing Manufactured Solutions without using source terms.

From the Abstract: "The principle advantage of the MMS procedure over traditional methods of Code Verification is that code capabilities are tested in full generality. The procedure thus results in a high degree of confidence that all coding mistakes which prevent the equations from being solved correctly have been identified."

The understanding and experience of the authors is profound, and the report should be read in its entirety by anyone interested in pursuing the Method of Manufactured Solutions.

Two Multidimensional Features. In the first 1-D example problem above, we noted that the Manufactured Solution, since it is analytic, can be applied over any range of the dependent spatial variable x, e.g. the domain could extend over $x \in [0,1]$ or $x \in [-\pi, +\pi]$ etc. This feature extends to multidimensions, e.g., the same multidimensional analytic solution could be applied to a square driven cavity problem, a rectangular cavity, a backstep, a wing, etc. Also, multidimensional problems might require a little more thought to assure that all terms of the governing equations are exercised. For example, a Manufactured Solution of form $U(t,x,y) = F1(t) \cdot F2(x) \cdot F3(y)$ will not be adequate to exercise governing equations containing cross derivative terms such as $\partial^2 u/\partial x \partial y$ since these are identically zero no matter how complex are the F's.

Mixed Order Methods. Roy [25,26] has shown how to treat mixed-differencing (e.g., first-order upstream differencing for advection and second-order differencing for other terms) in the systematic grid convergence tests. These two papers present the resolution, in an elegant manner, of a long-standing and practical difficulty in grid convergence studies and the GCI (Grid Convergence Index), namely, the treatment of mixed-order convergence. The mixed order behavior can arise either from the explicit use of 1st-order advection discretization and 2nd-order diffusion, or from the 1st-order observed convergence rate of nominally second-order methods caused by shocks. The procedure simply involves another grid level to evaluate the *two* leading coefficients in the error expansion. The analysis includes non-integer grid refinement factors **r**. The papers also demonstrate how nonmonotonic convergence occurs from mixed-order methods in the non-asymptotic range. The method is applicable to both Verification of Codes and Verification of Calculations, and would enable more accurate (less conservative) error estimation by way of the GCI [1] for QUICK and similar methods using 2nd order accurate diffusion terms and 3rd order accurate advection terms.

Radiation Transport Code Including Eigenvalue Problems Pautz [27] presents his experience applying MMS to the radiation transport code ATTILA. The application was inspired by Salari and Knupp [17] and contains some early 1970's references on the basic ideas, but these are " . . . more limited than the more recent and general treatment by Salari and Knupp."

The Code Verification described includes angular flux, scattering cross sections, and spherical harmonics. In the approach used, by choosing the term $f(\mathbf{r}) = 1$ (versus a more general form) in the assumed form of the manufactured Solution, one can isolate spatial differencing terms (they cancel) and Verify whether the code handles angular terms correctly.

The code uses 3-D tetrahedral elements (utilizing linear discontinuous finite element discretization) in space and discrete ordinates in the angular discretization. The study treated the following features: Steady State—Monoenergetic, Steady State—Multigroup, Monoenergetic K-Eigenvalue, Gray Infrared.

The experience with the MMS approach was quite successful. With $f(\mathbf{r}) = 1$, the author discovered coding mistakes in input routines (and a divide by zero for a particular combination of input options). Also, the procedure revealed mistakes in discretization of certain boundary data for the gray infrared problem. The approach Verified 2nd order convergence for norms and 3rd order convergence for average scalar flux. A subtle aspect required for successful application of the MMS procedure was the consistent finite element weighting on the MMS source term. Based on earlier 1-D analysis in the literature, it was expected that all the examined quantities (norms and average scalar flux) would exhibit 3rd order convergence, but the results of the MMS procedure demonstrated only 2nd order convergence for the norms in multi-dimensions. The author concluded that MMS is " . . . a very powerful verification tool" [27]. Further [pers.comm.] the author says "The power and conceptual simplicity of MMS make it an indispensable tool for code development" and recommends that MMS be required in any formal Code Verification system.

Nonhomogeneous Boundary Conditions. An arbitrary Manufactured Solution will not necessarily have homogeneous boundary conditions, e.g., $u \neq 0$ or $\partial u/\partial x \neq 0$. To use such a solution, the code would require this capability. This might be inconvenient, e.g., many CFD codes have hard-wired no-slip conditions at a wall, e.g., $u = 0$. Rather than modify the code, some thought will produce Manufactured Solutions with homogeneous boundary values.

Nonlinear Boundary Conditions. So-called "radiation" outflow conditions are usually linear and are already covered by the previous discussion. Nonlinear boundary conditions, e.g., simple vortex conditions at outflow, or true (physical) heat-transfer radiation boundary conditions, are possible. It may be possible to select a Manufactured Solution that meets the nonlinear boundary condition; otherwise, a source term would have to be added (if it is not already present) in the nonlinear boundary equations to retain the generality of the MMS.

Shocks. Shock solutions are treatable by the MMS, with additional considerations. See pages 89–90 of [1], which include the work of J. Powers and associates [13–16]. The simplest approach may be to Verify the shock capturing algorithms separately on inviscid benchmark problems such as oblique shock solutions, if shock curvature is not viewed as a major question, or if it is, by using attached curved shock solutions obtained by the method of characteristics and/or detached bow shock solutions obtained by the classical inverse method. The benchmark solutions may involve asymptotic approximations in geometry or Mach number, e.g. an analysis [16] neglecting terms of $O(\varepsilon^2)$ where $\varepsilon = 1/M^2$. This approximation can be made very accurate by choosing high M, say $M \sim 20$, for the Code Verification exercise. Note again the distinction of mathematics versus science; it is not a concern that the code being tested might be built on perfect gas assumptions that are not valid at such high M. This does not affect the mathematics of Code Verification; the code would not be applied at such high M when accuracy of the physics becomes important, during Code Validation.

The assumption involved in this approach is that the option matrix of the code may be partitioned (see [1], Chapter 6 for an example). That is, the Verification of the shock-capturing algorithm and coding will not be affected by the later inclusion of viscous terms, boundary conditions, etc. Other option-partitioning assumptions will occur to the reader: separated Verification of a direct banded Gaussian elimination routine in a FEM code; Verification of shock-capturing algorithm separate from non-ideal gas effects; radioactive decay option (which is dimensionless) verified separately from spatial discretization of flow equations. This partitioning approach requires the "black-box" Verification philosophy to be modified to a "glass-box" [7], i.e. some knowledge of code structure is required to justify the approach, and it will be more difficult to convince reviewers, editors, contract monitors, regulators, etc. that the approach is justified. The work savings can be enormous, of course, avoiding the factorial increase of complexity inherent in option combinations.

Another straightforward approach for shocks that does not involve partitioning the option matrix is to generate a Manufactured Solution that is in fact (in the continuum) C^{∞} smooth, but that has such strong gradients in some region that it appears as a shock over the targeted range of grid resolutions. The possible difficulty here is that some shock capturing algorithms are based on the conservation equations without source terms, e.g., Godunov's method and modern variants, and these could conceivably fail where source terms are present. Any shock-capturing algorithm based purely on geometric limiters will be oblivious to the source terms and should work without modification.

Requirement for Source Terms. In the version demonstrated, the MMS requires that the code be capable of treating source terms in each PDE. For some engineering codes, this is always the case, e.g. time-dependent chemistry codes, grid generation codes based on elliptic generation. Another approach to MMS developed by Knupp (see [17] and [1], Chapters 3 and 6) is applicable to variable coefficient problems, e.g., groundwater transport codes or heat conduction codes with variable properties. In this method, a Solution is Manufactured by solving directly for the distribution of variable coefficients that produces it. Generally, for Navier-Stokes codes, distributed source terms are nonphysical and would not have been included, so these codes would have to be modified to use the method. For codes developed ''in house,'' this is very little trouble. The only tricky situation we know of occurs with implicit Approximate Factorization codes built for second-order time accuracy, as noted above and in [1]; in this case, the consistently second-order treatment of source terms is subtle. Still, the trouble of adding source terms is small compared to the alternative of a haphazard and piecemeal approach to Code Verification. It is certainly trivial for FEM code using direct solvers. For general purpose commercial codes, it is not an undue hardship on vendors to be required to include source terms so that users can Verify the Codes themselves. (CFD Software vendors are notoriously uninterested in performing V&V and in sharing results with customers, apparently for good reasons.)

Solution Realism. The MMS as presented generates solutions to PDE's, modified to include source terms for all dependent variables, with no concern for realism of the solution. Thus, acceptance requires that the judge recognize that Code Verification is a purely mathematical exercise. Physical realism and even physical realizability are irrelevant. Actually, there is no *requirement* that the Manufactured Solution look unrealistic, and we can invent appealing solutions if necessary to satisfy managers, regulators, public stakeholders, etc. But it is worthwhile to understand that this ''realism'' is mere window dressing. It is also risky, in that it encourages a dangerous misconception and opens the door to criticism and arguments about adequate ''realism,'' which after all is a qualitative concept and again opens the door to piecemeal and perpetual Code Verification exercises.

Furthermore, ''realistic'' solutions can actually be less desirable. For example, a realistic solution for Navier-Stokes equations would have a boundary-layer behavior at walls. But then the terms neglected in classical boundary layer theory (e.g., $\partial p/\partial y$) will not be strongly exercised in the code, and a minor error might conceivably slip by undetected, e.g., a one-sided difference expression for $\partial p/\partial y$ near the wall thought to be 2nd order but actually 1st order will not affect the answer if $\partial p/\partial y = 0 + O(\Delta^2)$ whereas an ''unrealistic'' solution based on tanh, etc. would exercise these terms.

Code Verification With a Clearly Defined Completion Point As noted earlier, it is now well-recognized that benchmark solutions for Code Verification must exhibit sufficiently complex structure that all terms in the governing equation being tested are exercised. What apparently is not so widely recognized today is that, once a code (or rather, a specific set of code option combinations) has been convincingly Verified on such a solution, it is nearly pointless to continue exercising it on simpler problems. I say ''nearly'' because the exercises have some value, but should not be thought of as Verification, but as Confirmation exercises; see Chapter 1 of [1]. In the present view, Code Verification has a theorem-like quality, and therefore terminates. Like a high-school student plugging numbers into the solution for the quadratic equations, a code user who performs Confirmation exercises gains confidence in the code and in his ability to set up the code and to interpret the results. Such Confirmation exercises are valuable, indeed necessary, as part of user training, but these should not be confused with Code Verification. Similarly, we recognize that simple problems (e.g., 1-D linear wave propagation) are useful in algorithm development, in exploring algorithm and code characteristics, and in comparing performance of different codes, but once again, these comparison exercises should not be confused with Code Verification. For example, one could have three codes of 1st, 2nd, and 4th order accuracy, each of which was rigorously Verified to be so. Then a comparison exercise based on simple wave propagation with a linear advection-diffusion equation would be expected to show increasing accuracy; however, this does not alter the previously determined and completed Code Verifications.

Proof? Does such a Code Verification process deserve the term ''proof''? This is another semantic question whose answer depends on the community context. Logicians, philosophers and pure mathematicians clearly view ''proof'' differently from engineers, with an often other-worldly standard. For example, Fermat's Last Theorem is easily demonstrable; anyone can readily convince themselves of its correctness, and a straightforward computer program can be written to convincingly demonstrate its correctness for systematic millions of cases. No one, not even the philosophers or logicians or pure mathematicians, doubts it. Indeed, if one were to put forward a counter-proof, it would be rejected by all. Yet only recently has a book-length ''proof'' been put forward (and doubts about it remain). Since some philosophers maintain that it is not possible even in principle to prove relativity, or Newton's laws of gravity (which are certainly provable within engineering accuracy) they are not going to accept the notion of proof of correctness of a complex code, i.e., Verification of Code

The notion of proof is at the heart of very important criticisms, not just of the subject MMS, but of the concepts of Code Verification and especially Certification [1] for large public-policy projects. One might agree with some philosophers who maintain it is not possible to prove relativity or Newton's Laws, but would one be willing to cancel a public policy project (e.g., a nuclear waste project) because the modeling used Newton's laws? Presumably not, but stakeholders are willing to cancel such projects under the guise of unprovability of code correctness. The harm is done when these standards for proof of philosophers, mathematicians or logicians are applied to down-to-earth engineering projects. If we accept such out-of-context standards for proof, we cannot do anything, literally. For example, we have no proof of convergence for real systems, because the Lax Equivalence theorem only holds for linear systems. The word ''proof'' is itself a technical term, with different appropriate standards in logic, pure mathematics, applied mathematics, engineering, criminal law vs. torts vs. civil law (e.g., ''beyond a reasonable doubt''), etc. The first definition in one dictionary for ''proof'' is ''The evidence or argument that compels the mind to accept an assertion as true.'' In this sense, if not in a strict mathematical sense, one could claim that the MMS approach can provide proof of Code Verification.

I am unhesitating in claiming ''convincing demonstration'' and ''robust Verification'' for the present MMS approach. A mathematical proof would require the formalism of a theorem; as noted, it would seem that a theorem is possible, for some related problem(s). Further, if one allows the legitimacy of a non-mathematical proof in principle, then I would claim that this method provides it. It is highly unlikely that a code embodying the Burgers equation (2) and passing the Verification test for the solution of the above example could be wrong (without a contrived counter-example). More complex codes with option combinations require more tests, obviously. Computer Scientists like to negate the possibility of complete Code Verification by considering codes of ‘‘arbitrary complexity’’ or ‘‘arbitrary number of options,’’ but in fact real codes have a countable, exercisable number of options; in any case, a code can be conditionally Verified just for those sets of options exercised. Geometry complexity is hard to address in general, but once again the claim of Code Verification can be conditionally stated to restrict geometry families to those tested. In practice, difficulties with complex geometries (e.g., singularities) are often not Code Verification issues at all, but are simply difficulties with Verification of Calculations; i.e., they are not issues of code correctness.

An Alternative View on Code Verification With a Clearly Defined Completion Point. The present view of Code Verification as a theorem-like process with a fixed termination is not universally accepted. In an alternative view [5–7,28] held by respected authorities, Code Verification (and Validation) are ‘‘ongoing activities that do not have a clearly defined completion point’’ [28], more akin to accumulating evidence for a legal case than to proving a theorem [6,7]. Both viewpoints recognize, obviously, that if the code is modified, it is a new code (even if the name of the code remains) and the new code must be re-Verified. Also, both viewpoints recognize that all plausible non-independent combinations of input options must be exercised so that every line of code is executed in order to claim that the entire code is Verified; otherwise, the Verification can be claimed only for the subset of options exercised. And both viewpoints recognize the value of ongoing code exercise by multiple users, both in an evidentiary sense and in user training. In this alternative view these activities could be part of formal Code Verification itself, rather than as Code ‘‘Confirmation’’ as in the present view [1].

The decision whether or not to include these activities under ‘‘Code Verification’’ rather than ‘‘Code Confirmation’’ is semantic but it must be recognized that it has practical and possibly serious consequences. For example, contractual and/or regulatory requirements for delivery or use of a ‘‘Verified Code’’ might be ambiguous in this view, since ‘‘Code Verification’’ by definition is never-ending. Also, any test (even a superficial one) could be claimed as ‘‘partial Verification.’’ However, some advantages exist for this view, e.g., encourages more precision of the meaning of ‘‘Verified Code,’’ and it more explicitly recognizes the value of ongoing code exercise by the user community. Both viewpoints recognize that ongoing code use and exercise can possibly uncover mistakes missed in the Code Verification process (just as a theorem might turn out to have a faulty proof or to have been misinterpreted) but in this alternative view Code Verification cannot be completed, except by specification (perhaps negotiated) of the meaning of ‘‘Verified Code.’’ Verification of individual calculations, and certainly Validations, are still viewed as ongoing processes in both views, of course. [1,2,5–7,28]

Concluding Remarks

The Method of Manufactured Solutions for Code Verification is typically met with skepticism, but in the experience of Oberkampf and Trucano [7] and my own, people who actually try it are enthusiastic. The MMS enables one to produce many exact analytical solutions for use as benchmarks in systematic discretization refinement tests, which tests are remarkably sensitive for Code Verification. The method is straightforward and, when applied to all option combinations in a code, can lead to complete and final Code Verification, with a well-defined completion point. It eliminates the typical haphazard, piecemeal and never-ending approach of partial Code Verifications with various highly simplified problems that still leave the customer unconvinced.

Acknowledgments

I gratefully acknowledge many clarifying discussions with Dr. W. L. Oberkampf.

References

[1] Roache, P. J., 1998, *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque, NM.
[2] Roache, P. J., 1999, *Fundamentals of Computational Fluid Dynamics*, Hermosa Publishers, Albuquerque, NM, Chapter 18.
[3] Roache, P. J., Ghia, K., and White, F., 1986, ‘‘Editorial Policy Statement on the Control of Numerical Accuracy,’’ ASME J. Fluids Eng., **108**, No. 1, Mar., p. 2.
[4] Steinberg, S., and Roache, P. J., 1985, ‘‘Symbolic Manipulation and Computational Fluid Dynamics,’’ J. Comput. Phys., **57**, No. 2, Jan., pp. 251–284.
[5] Oberkampf, W. L., Blottner, F. G., and Aeschliman, D. P., 1995, ‘‘Methodology for Computational Fluid Dynamics Code Verification/Validation,’’ AIAA Paper 95-2226, 26th AIAA Fluid Dynamics Conference, 19–22 June, San Diego, California.
[6] Oberkampf, W. L., and Trucano, T. G., 2000, ‘‘Validation Methodology in Computational Fluid Dynamics,’’ AIAA Paper 2000–2549, 19–22 June, Denver, CO.
[7] Oberkampf, W. L., and Trucano, T. G., 2002, ‘‘Verification and Validation in Computational Fluid Dynamics,’’ AIAA Progress in Aerospace Sciences (to appear).
[8] Pelletier, D., and Ignat, L., 1995, ‘‘On the Accuracy of the Grid Convergence Index and the Zhu-Zienkiewicz Error Estimator,’’ pp. 31–36 in ASME FED Vol. 213, Quantification of Uncertainty in Computational Fluid Dynamics, Aug., R. W. Johnson and E. D. Hughes, eds.
[9] Reed, H. L., Haynes, T. S., and Saric, W. S., 1998, ‘‘CFD Validation Issues in Transition Modeling,’’ AIAA J., **36**, No. 5, May, pp. 742–749. See also AIAA Paper 96-2051, 27th AIAA Fluid Dynamics Conference, 17–20 June 1996, New Orleans, Louisiana.
[10] Polya, G., 1957, *How to Solve It; A New Aspect of Mathematical Method*, Princeton University Press, Princeton, NJ.
[11] Dee, D. P., 1991, ‘‘Prescribed Solution Forcing Method for Model Verification in Hydraulic Engineering,’’ Proc. 1991 National Conference on Hydraulic Engineering, ASCE, Nashville, July 29-August 2.
[12] Ethier, C. R., and Steinman, D. A., 1994, ‘‘Exact Fully 3D Navier-Stokes Solutions for Benchmarking,’’ Int. J. Fract., **19**, pp. 369–375.
[13] Powers, J. M., and Gonthier, K. A., 1992, ‘‘Reaction Zone Structure for Strong, Weak Overdriven and Weak Underdriven Oblique Detonations,’’ Phys. Fluids A, **4**, No. 9, Sept., pp. 2082–2089.
[14] Powers, J. M., and Stewart, D. S., 1992, ‘‘Approximate Solutions for Oblique Detonations in the Hypersonic Limit,’’ AIAA J., **30**, No. 3, Mar., pp. 726–736.
[15] Grismer, M. J., and Powers, J. M., 1992, ‘‘Comparison of Numerical Oblique Detonation Solutions with an Asymptotic Benchmark,’’ AIAA J., **30**, No. 12, Dec., pp. 2985–2987.
[16] Grismer, M. J., and Powers, J. M., 1996, ‘‘Numerical Predictions of Oblique Detonation Stability Boundaries,’’ Shock Waves, **6**, pp. 147–156.
[17] Salari, K., and Knupp, P., 2000, ‘‘Code Verification by the Method of Manufactured Solutions,’’ SAND2000-1444, Sandia National Laboratories, Albuquerque, NM 87185, June.
[18] Salari, K., and Roache, P. J., 1990, ‘‘The Influence of Sweep on Dynamic Stall Produced by a Rapidly Pitching Wing,’’ AIAA Paper 90-0581.
[19] Roache, P. J., and Steinberg, S., 1985, ‘‘A New Approach to Grid Generation Using a Variational Formulation,’’ AIAA 85-1527-CP, *Proceedings AIAA 7th Computational Fluid Dynamics Conference*, 15–17 July, Cincinnati, Ohio.
[20] Steinberg, S., and Roache, P. J., 1986, ‘‘Using MACSYMA to Write Fortran Subroutines,’’ Journal of Symbolic Computation, **2**, pp. 213–216. See also *MACSYMA Newsletter*, Vol. II-2, 1985, pp. 10–12.
[21] Steinberg, S., and Roache, P. J., 1986, ‘‘Variational Grid Generation,’’ Numerical Methods for Partial Differential Equations, **2**, pp. 71–96.
[22] Steinberg, S., and Roache, P. J., 1986, ‘‘A Tool Kit of Symbolic Manipulation Programs for Variational Grid Generation,’’ AIAA Paper No. 86-0241, AIAA 24th Aerospace Sciences Meeting, 6–9 January 1986, Reno, Nevada.
[23] Steinberg, S., and Roache, P. J., 1986, ‘‘Grid Generation: A Variational and Symbolic-Computation Approach,’’ *Proceedings Numerical Grid Generation in Fluid Dynamics Conference*, July, Landshut, W. Germany.
[24] Steinberg, S., and Roache, P. J., 1992, ‘‘Variational Curve and Surface Grid Generation,’’ J. Comput. Phys., **100**, No. 1, pp. 163–178.
[25] Roy, C. J., McWherter-Payne, M. A., and Oberkampf, W. L., 2000, ‘‘Verification and Validation for Laminar Hypersonic Flowfields,’’ AIAA 2000-2550, June, Denver.
[26] Roy, C. J., 2001, ‘‘Grid Convergence Error Analysis for Mixed-Order Numerical Schemes,’’ AIAA Paper 2001–2606, June, Anaheim. See also this issue of ASME JFE.
[27] Pautz, S. D., 2001, ‘‘Verification of Transport Codes by the Method of Manufactured Solutions: the ATTILA Experience,’’ Proc. ANS International Meeting on Mathematical Methods for Nuclear Applications, M&C 2001, Salt Lake City, Utah, Sept.
[28] AIAA, 1998, ‘‘Guide for the Verification and Validation of Computational Fluid Dynamics Simulations,’’ American Institute of Aeronautics and Astronautics, AIAA-G-077-1998, Reston, VA.