

Procedures for Calculation Verification

William J. Rider, James R. Kamm, and V. Gregory Weirs
Sandia National Laboratories, Albuquerque NM 87185
January 6, 2011

Overview

Calculation or solution verification is a process by which the discretization or numerical error is estimated in simulations of problems of interest. Calculation verification employs techniques of both error estimation and uncertainty quantification (UQ). There are defined procedures by which numerical error estimates can be converted to numerical uncertainty estimates. Each of these techniques will be discussed in this document. We take the two terms *calculation verification* and *solution verification* to be synonymous. *Code verification* is a related, but distinct process where the correctness of a software implementation of a numerical algorithm is evaluated, typically by comparison against an exact solution.

Numerical methods that are used to obtain approximate numerical solutions of continuum models unavoidably lead to errors in the computed results. These errors are associated with the numerical method *alone* and have nothing to do with any assumptions related to the form of the continuum models (e.g., model-form errors). The challenge of calculation verification is to help provide estimates of such numerical errors. These errors are of four general types: (1) round-off errors, (2) sampling errors, (3) iterative (linear and nonlinear) solver errors, and (4) discretization errors.

Round-off error is an inevitable consequence of finite precision representation of numbers on computers; with high-precision arithmetic, round-off errors typically have smaller effect on computed results than the other primary error sources. Sampling errors occur, e.g., for methods that are inherently stochastic (e.g., Monte Carlo methods) or in sampling a system response quantity of interest. Iterative solver errors are associated with the solution of any linear or nonlinear systems of equations encountered during the course of the computational solution update procedure. In practice, iterative errors may be difficult to reduce to the point where they can be neglected. Discretization errors are a direct consequence of the numerical scheme used to obtain a discrete approximation of continuous equations (e.g., finite difference, finite element, or finite volume methods) and the solution approach used on those discrete equations. For time-dependent problems, both the spatial and temporal discretizations enter into the evaluation of these errors. Many researchers contend that discretization error is often the dominant source of numerical error in scientific computing simulations. This is consistent with much of the authors' experience, although nonlinear solver error can dominate strongly coupled (stiff) problems.

The important concepts to consider regarding iterative solver error are covered by Eça and Hoeckstra [Eça09] and Oberkampf and Roy [Obe10]; we only briefly summarize the issues here. The use of direct solvers for linear systems of equations has become impractical for reasons of efficiency, so that iterative solvers are the usual approach. The iterative solution of linear systems of equations is necessarily inexact and generates a source of error to be estimated. Usually, the iteration will terminate based on criteria not related to the actual error in the solution. This is common if the iterative solver is developed as part of a general-purpose software package. In this case, the estimation of error associated with the iterative solver is essential. Furthermore, the default termination criteria may be unsatisfactory for the application-specific context. In any case the uncertainty contribution must be assessed with care. The iterative solution of nonlinear systems, which typically involves both linear and nonlinear stopping criteria, has the same issues.

Among the most important characteristics of discretization schemes is the order-of-accuracy (also called the convergence rate), which is given by the exponent in the power law relating the numerical truncation error to the value of a parameter associated with the discretization, usually given by the size of the computational cell or time step. The factor multiplying this term gives a measure of the overall error of a given scheme; thus, two different schemes that converge at the same rate may have different (absolute) discretization errors. The standard method by which to estimate this accuracy is systematic mesh refinement (or variation), although there are other, less general approaches [Roy10a]. The results of this approach are combined with error measurement to produce the observed rate-of-convergence, which is compared with the ideal or theoretical rate-of-convergence of the underlying algorithm. In calculation verification, unlike code verification, the use of an analytical or exact solution to a problem is not available as an unambiguous fiducial solution. Instead, the comparisons are made between solutions using different grid resolutions.

To aid analysts in conducting solution verification analyses, the following workflow for calculation verification is proposed.

1. Starting with an algorithm implementation (i.e., code) that has passed the appropriate level of software quality assurance and code verification, choose the software executable to be examined.
2. Provide an analysis of the numerical method as implemented including accuracy and stability properties. (This information should be available from the code verification analysis.)
3. Produce the code input to model the problem(s) of interest.
4. Select the sequence of mesh discretizations to be examined for each problem, and the input necessary to accomplish these calculations.
5. Run the code and provide the means of producing appropriate metrics to evaluate the difference between the computed solutions based on numerical

- parameters within the control of the code user. This can also include the numerical method chosen (order of approximation or scheme).
6. Use the comparison to determine the sequence of estimated errors corresponding to the various discretizations and tolerances.
 7. The error sequence allows the determination of the rate-of-convergence for the method, which is compared to the theoretical rate. For iterative solver errors, the error is a function of the stopping criteria and the discretization.
 8. Using these results, render an assessment of the accuracy (level of error estimated) for the simulation for a given set of numerical settings.
 9. Examine the degree of coverage of features in an implementation by the verification testing.

This document contains detailed descriptions of many aspects of calculation verification, as well as a discussion of the general workflow outlined above. The views contained herein are not meant to be a complete prescription by which to conduct solution verification. Instead, they are intended to guide the conscientious analyst in conducting studies that will contain defensible estimates of the numerical errors inherent in computed solutions for complex, engineering simulations.

What Is Verification And Its Relation To Other Similar Activities?

Numerical simulations are increasingly used to increase understanding, to “solve” problems, to design devices, vehicles and buildings, to “predict” responses of complicated engineering systems, and to inform high-consequence decision makers. Different assessment techniques are employed to address the different ways numerical simulations can fail to provide accurate information. Ultimately, the combined application of all the techniques provides enhanced confidence that the accuracy of the numerical simulation process is adequate for a particular scenario.

The purpose of scientific simulation software differs from that of most commercial software. Verification is needed for scientific simulation codes because this software is designed to produce approximate solutions to mathematical problems for which (i) the exact solution is not known and (ii) knowledge of the error is potentially as valuable as knowledge of the solution, per se. Due to these distinguishing and critical aspects of scientific simulation codes, software quality practices from the broader industry (e.g., regression testing) are necessary but not sufficient for high-consequence scientific simulation codes.

Verification analysis of scientific simulation codes is an example of the assessment of a complex system for which the systematic gathering of appropriate evidence is required. While tests may demonstrate that software is manifestly incorrect, there is no clear-cut procedure with which to “prove” unambiguously that software behavior is, indeed, correct. Thus, the process by which relevant verification evidence is generated and interpreted requires knowledge of the entire simulation and analysis chain. Such knowledge includes understanding of:

- The system being simulated (e.g., the relevant physics, physics models, and these models' representations in mathematical equations);
- The nature of the simulation (including the algorithms used to obtain approximate solutions to the mathematical equations, these algorithms' limitations, the associated numerical analysis, and the software implementation of those algorithms); and
- The process by which the code results are analyzed in the verification process (including, e.g., theory, implementation, and interpretation of convergence analysis).

This body of knowledge is both large and multi-faceted; consequently, the determination of appropriate of verification problems requires guidance from and consensus among experts in each of these fields.

Decision makers and code analysts should bear in mind that simulation software represents intricate numerical algorithms coupled with a complicated hardware/system-software platform. Stated another way, code users and their customers should recognize that simulation software is *not* a “physics engine” that generates instantiations of physical reality. Hence, documented, quantitative verification analysis is a necessary component for developing code confidence and credibility.

At its core, verification of scientific simulation software both quantifies numerical errors and defines a rigorous basis for believing that quantification. Providing an error estimate for complex problems falls under the purview of *solution* (or *calculation*) *verification*, while providing the rigorous basis for such estimates is achieved with *code verification*. The overall activity of verification is the combination of *both* code and solution verification.

More precisely, the different kinds of verification can be defined as follows:

- *Calculation or solution verification*: using the demonstrated convergence properties of the code to estimate numerical errors in solving the model for a problem of interest, involving the evaluation of results of the code alone. The credibility of calculation verification is predicated on producing error estimates from a code that have passed appropriate and relevant code verification analysis.
- *Code verification*: comparing the results of a coded algorithm (i.e., instantiated in software) with an analytical or exact (i.e., “closed form”) solution or highly accurate solution obtained by some other means¹, for the purpose of assessing the code.

¹ For example, for certain problems governed by specific partial differential equations (PDEs), the equation can be reduced to an ordinary differential equation (ODE), the solution of which can often be obtained very accurately, sometimes exactly. A highly accurate

Both code and calculation verification focus on numerical errors. Calculation verification is practiced by code users to estimate the numerical error in their simulations in order to provide a justifiable, best-estimate solution. These estimates are not guaranteed to be rigorous: they, too, are approximations. A key point is that the activity of estimating numerical errors is part of the necessary due diligence for conducting predictive simulations. In contrast, code verification is an activity executed by code developers, mathematicians, and numerical analysts. It compares the behavior of the actual, observed error from numerical simulations to the expected error behavior as derived from rigorous mathematical proofs and estimates.

A common confusion with regard to verification is associated with software quality assurance (SQA), which is a vital, but essentially unrelated activity that comprises an important discipline in its own right. Verification typically flourishes in a culture focused on high quality, but good verification practices are neither necessary nor sufficient for good SQA practices—and vice-versa. Each area of expertise should be independently developed and supported, although the practice of each is mutually supportive.

- *Software verification*: checking for a correct functioning of the software system on a particular platform. This is often used to determine whether the software has properly implemented the defined requirements of the algorithms instantiated in the code.

Such software testing is a critical element of software development. The testing that is closest to code development centers on software engineering techniques, such as unit testing and regression testing, both of which address the correct functioning of software. These types of testing use generic success metrics that apply to almost all classes of software. In contrast, code and solution verification are assessment techniques for software that provides approximate solutions, with metrics specialized to the particular type of algorithm.

There is sometimes confusion of code verification testing with regression testing; however, these are different testing procedures with completely different goals. Regression testing is a software engineering technique that assesses the robustness of software to frequent changes. Regression tests reduce to a (typically large) collection of relatively simple problems that are executed at a regular (typically frequent) time interval. Regression testing seeks principally to reduce the amount of software rework that is created by the introduction of mistakes in software modifications. This reduction is achieved by comparing today's code with yesterday's code via execution of the regression test suite. Thus, regression testing evaluates *software stability*, not *mathematical correctness*.

numerical solution of an ODE could be used as the de facto “exact” solution for purposes of code verification analysis of a PDE solver for that problem.

This mathematical correctness is the purview of code and calculation verification. Rephrasing our earlier definitions, *calculation verification* estimates numerical errors for a problem of interest based on an assumed relationship between numerical error and resolution (as measured by the discretization parameter); code verification tests whether this assumption is satisfied for a known algorithm on a problem with a known solution.

The outcome of code verification analyses should provide defensible evidence of mathematical consistency—or inconsistency—between the mathematical statements of the physics models and their discrete analogues as implemented with numerical algorithms in the simulation codes. The necessity of calculation verification must be emphasized. In the absence of confirmatory verification evidence, “good agreement” of calculations with experimental data could be accidental, i.e., “the right answer for the wrong reasons.”

Another activity that is confused with calculation verification is *mesh sensitivity*. Mesh sensitivity is the process of comparing the solutions computed on two or more grids for the express purpose of determining whether the solution is qualitatively dependent on mesh resolution. The degree of dependence is usually assessed only in a qualitative fashion, although the calculations, if done properly, can serve as the basis for calculation verification. Without the quantitative verification component of this activity, non-convergent calculations may remain unidentified.

Unlike verification, *validation* tests whether a model is a sufficiently accurate representation of the physical processes in a particular problem. For detailed discussions on the relation of verification and validation, we refer the reader to the many published reviews on verification and validation; see, e.g., [AIA98, ASM06, Han01, Joh06, Kam08, Knu03, Nel10, Obe02, Obe04, Obe07, Obe10, Ore94, Rid10, Rid11, Roa98, Roa09, Roy05, Roy10b, Sar98, Sar01, Sch06, Sor07, Ste01, Ste06, Tru03, Tru06] and the references therein. In scientific simulations the *model* refers to the governing equations, which include initial conditions, boundary conditions, constitutive relations, etc. To simulate essentially any nontrivial physical configuration, these equations must be solved computationally, which depends on numerical algorithms, corresponding software implementations, and appropriate use of that software, including suitable assignment of model parameters. Therefore, validation entails comparisons of approximate solutions of the governing equations (which are an imperfect representation of the relevant physical processes) to experimental data (which also contain inaccuracies). Quantitative comparison of experiment (having, e.g., physical and diagnostic uncertainties) with simulations (having concomitant modeling, algorithmic, and solution errors) remains a challenging and strongly problem-dependent undertaking. It is regular experimental practice to provide error bars showing the degree of uncertainty in physical data, and solution verification can help provide estimates of the numerical error in the corresponding simulations.

Every simulation requires a number of inputs to specify the problem to be solved, including the choice among various numerical methods and how they are applied, as well as the assignment of values for parameters in specific component models (e.g., specifying material response). These values may not be known exactly or may represent some average or even ad hoc value that depends on the particular situation. The process of identifying and quantifying the effects of the uncertainty of these simulation inputs on the results of the simulation falls under the rubric of *uncertainty quantification*. The most common way of estimating these effects is to run a (typically large) number of simulations, in which the values of the inputs are taken from user-specified distributions, to determine the corresponding distribution of the results of interest. Unlike code verification, solution verification, or validation, uncertainty quantification deals with the variation of simulation results relative to how the problem is specified, rather than how well that problem is solved.

Code verification is the foundation upon which the other assessment techniques rest. The premise of solution verification is that the code converges at a known rate as the resolution is increased; code verification establishes that this is, in fact, the case. To compare experimental results to code results, both the experimental and numerical errors must be meaningfully quantified, so they can be properly accounted for in the comparison. Validation relies on solution verification to provide estimates of the numerical error, a procedure that, in turn, relies on code verification. The process of uncertainty quantification accepts the model, in this case the code that produces numerical simulations, as an input. Inferences drawn about the system that the model represents are inherently limited by the accuracy of the model. Code verification ensures the model is correctly implemented and underlies solution verification and validation that quantify the accuracy of the model. Code verification is foundational in that without it, the conclusions of the other assessment techniques are suspect, but the converse is not true.

On Defining A Verification Test Problem

To conduct a verification analysis, one must have (i) a clear statement of the problem with sufficient information to run a computer simulation, (ii) an explanation of how the code result and solution (i.e., the benchmark) are to be evaluated, and (iii) a description of the acceptance criteria, including how the calculation errors are to be estimated, for a specific simulation code's results on a particular problem. These concepts are adapted from the notion of a "strong sense verification benchmark," proffered by Oberkampf and Trucano [Obe07], and are intended to enhance the value of verification analyses by reducing the ambiguity of problem statements, evaluation techniques, and their interpretation.

Here, the problem statement should include not only a mathematical description of the problem but also a discussion of the processes modeled (what does this problem test?), the initial and boundary conditions, additional numerical information (what convergence criteria are used?), the principal code features tested, and the nature of

the test. This latter element is addressed in [Obe07], with a set of different categories of benchmarks; Kamm et al. [Kam08] provide examples of such problem statements. Given the complexity of many problems of interest, however, such problem descriptions may still not be definitive, i.e., there may remain unspecified choices in problem set-up that the code analyst must make. Nevertheless, such a description provides a starting point for setting up the problem as well as a touchstone against which one can compare descriptions of the “identical” verification problem, run by different analysts with different simulation codes on different platforms at different times.² In the written description and analysis of verification problems, it is imperative that researchers describe *as thoroughly as possible* the complete specification and set-up of the problem (up to including the code input deck in the written report).

An explanation of the evaluation process by which results are to be evaluated is imperative for both completeness and repeatability. This evaluation process includes, e.g., in code verification what the “exact” solution is and how it is evaluated. Such an explanation may also include how system response quantities (the convergence of which is to be calculated) are evaluated from computed solution variables.

Also required are a description of which errors are to be estimated and an explanation of precisely how those errors are estimated. So that the analysis can be meaningful, acceptance criteria on the error behavior for the quantities of interest on the particular problem are also required, in the spirit of what might be found, say, in a requirements document. In the absence of specific criteria, a codified process for acceptance is necessary

Who Does Calculation Verification? Code Users

Complex simulations cannot be proven to be mathematically correct, or even convergent to the “true” solution. Consequently, the accumulation of quantitative evidence remains the exclusive basis for assessing the mathematical correctness and its relation to physical correctness. The practical view is that this evidence is accumulated over time. This accumulation occurs throughout the on-going processes of code use and the considered analysis of the computational results. Along with the results, the input to the code used to specify the calculation must be carefully examined for correctness. This process is similar to the manner in which software is examined. Code verification is the purview of code developers and algorithm designers, whose responsibilities should include descriptions of verification problems both in documentation and in actual code input. For those

² Analyses containing the possible variations mentioned here differ markedly from “code comparison” exercises, problematic aspects of which are pointed out by Trucano et al. [Tru03]. Verification acceptance criteria should be sufficiently forgiving, however, to allow small variations associated with imperfect specification of verification problems.

using a code to conduct analysis, their responsibility is to act mindfully regarding the quality of the code, and the relevance of the testing to their problems of interest. At best, they should act as advocates for quality control measures such as rigorous and—to the degree possible—extensive verification because it supports confidence building in the calculations for which the code is ultimately to be used.

Who Does Code Verification? *Code Developers, Mathematicians, And Algorithm Engineers.*

Just as complex simulations cannot be proven mathematically correct, likewise complex simulation software cannot be proven to be mathematically correct. Consequently, the accumulation of quantitative evidence remains the exclusive basis for inferring the mathematical correctness; as in calculation verification, this evidence is accumulated over time through the on-going processes of code development and code usage. Thus, the results of code verification analyses are affected by the manner in which software is generated and the proper specification/execution of the verification problems. While the former is the purview of code developers and algorithm designers, responsibility for the latter falls upon those who describe the verification problem both in documentation and in actual code input.

Verification evidence emerging from code development is generated by software engineering processes applied during that development, and by the specific testing practices employed by the development team. Code usage evidence is a more nuanced and diverse body of information that emerges from a heterogeneous group of users. Testing executed under the umbrella of code development is not restricted to the verification approaches discussed here. Unlike other testing procedures applied by code developers (including, e.g., unit tests and the restricted cases applied in regression testing), verification test problems strive also to be relevant to code users.

Verification test suites can be implemented, managed, and applied by code developers in the same manner as regression test suites. The major differences are: (i) the development and execution of verification test suites typically requires more resources (people, computers, time); (ii) the time interval of execution of a verification test suite may be different than for regression testing; and (iii) the direct methods for comparing today's regression test suite results versus yesterday's baseline should be augmented by greater human involvement in judging the quality of the verification tests where possible. On the other hand, subjective judgment is ultimately intrinsically associated with the quality of software at some fundamental level. For example, a verification study rarely produces results that exactly produce the theoretical convergence rate; in fact, the observed rates can vary greatly. The judgment of whether the result is close enough to expectations remains largely a decision for a subject-matter expert matter. Improvement of this state of affairs is an ongoing research area. This increased human element required to assess the execution of verification tests emphasizes that an important value of

verification tests is their use in engaging the user community around a code.

Verification Techniques

The workhorse technique for estimating discretization error is systematic mesh refinement (or de-refinement, i.e., coarsening), while the method for estimating iterative error involves systematic changes in stopping criteria for the iteration. A fundamental expectation for a numerical method is the systematic reduction in solution error as, say, the characteristic length scale associated with the mesh is reduced. By the same token, iterative errors are assumed to be smaller as the stopping criterion is decreased in numerical value. For mesh refinement, in the asymptotic limit where the mesh length scale approaches zero, a correct implementation of a consistent method should approach a rate of convergence equal to that defined by numerical analysis (often obtained with the aid of the Taylor series expansion). In practice, this expectation is not always met, i.e., calculations might not be in the asymptotic range. *This circumstance does not obviate the need for some estimate of the numerical error, however imprecise that estimate may be; in fact the necessity may be increased under these conditions.*

To conduct analysis using this approach, a sequence of grids with different intrinsic mesh scales is used to compute solutions and their associated errors. The combination of errors and mesh scales can then be used to evaluate the observed rate of convergence for the method in the code. In order to estimate the convergence rate, a minimum of two grids is necessary (giving two error estimates, one for each grid). The convergence tolerance for iterative solvers can be investigated by simple changes in the value of the stopping criteria. Assessing iterative convergence is complicated by the fact that the level of error is also related to the mesh through a bounding relation in which the error in the solution is proportional to the condition number of the iteration matrix. Most investigations of iterative solver error only consider the impact of the stopping criteria alone.

Another tool used in verification is error estimation. These methods are commonly derived for the finite element method (FEM) for solving elliptic partial differential equations. The best-known method is the Zienkiewicz-Zhu error estimator [Zie92]. One can use the error estimate constructively to drive adaptive mesh refinement, or to produce an estimate of the error on a given mesh. While the error estimate is produced, this approach does not necessarily produce the sort of evidence basis for calculation convergence, however, because the rate of convergence is not produced in the process. Moreover these tools are more strongly predicated upon the solution being well behaved. Without solution smoothness in the asymptotic range, these tools are highly questionable for precise error estimation. We focus on techniques that can be applied without regard to whether the computations are in the asymptotic range, because it is seldom encountered in practice.

Choice Of Metrics

Several metrics can be used to conduct code or calculation verification. For a mathematically rigorous result, one seeks to evaluate the quantitative difference between two sets of numbers, where each set corresponds to some aspect of the solution computed in a self-consistent manner. For example, integration over the same physically defined region provides a well-defined metric with which to gauge such a difference. Generally speaking, one should employ the metric that follows naturally from the function space in which numerical analysis proofs of convergence are conducted. One often cannot apply this approach in practical circumstances.

For simple scalars (say, the value of some scalar property, e.g., temperature, at a particular location at a specified time), the absolute value of the difference between two values is the obvious choice. This notion generalizes naturally to higher dimensional cases involving the difference between (discrete) function values from the computational mesh. As examples, these values could be, e.g., a time series of temperature at a particular location (the relevant computational mesh being in time) or, say, the pressure field over a specified, fixed three-dimensional volume at a specified time (the relevant computational mesh being in space). In such cases, one often uses the familiar “ p -norm” of functional and numerical analysis. The p -norm of the function g is given by

$$\|g\|_p \equiv \left(\int_a^b |g(x)|^p dx \right)^{1/p} \quad (1)$$

For example, for finite volume methods applied to discontinuous functions, the use of the 1-norm is recommended, while, say, properties of inherently smooth functions are most appropriately measured in the energy or 2-norm. It is can be enlightening to evaluate several norms, e.g., 1-, 2-, and ∞ -norms, where, following from the equation above,

$$\|g\|_\infty \equiv \max_{x \in [a,b]} |g(x)| \quad (2)$$

In the following, we use the double-bar notation “ $\|\cdot\|$ ” without a subscript to denote any appropriate norm. Ideally, the practical results should be connected to rigorous mathematical analysis, but this is often intractable.

In practice, the metrics tested for code and solution verification are generally different. For code verification, numerical analysis of the algorithm defines the metric and the expected convergence rate, and the assumptions under which convergence is expected. For most spatial and temporal discretizations, the metric is an appropriate norm of the error of the solution; that is, the error of a field variable integrated over the computational domain. The numerical analysis specifies how the error is defined. Depending on the properties of the numerical method and the

solution, the theoretical convergence rate may be expected to apply for point values of the solution as well.

For calculation verification, the analyst or code user chooses metrics that are important to the decision maker; these metrics are sometimes called Figures of Merit (FOMs), Quantities of Interest (QOIs), or System Response Quantities (SRQs). These metrics provide the “answer” to the question the analyst is trying to address; determining their values is the reason for running the set of simulations. These metrics may be full field solutions, but more often are values or functionals of the solution at a particular location or time. The convergence of such metrics will depend directly on the mathematical relationship between the FOM and the numerical field solution provided by the simulation. In practice, the quality of FOM error estimates often suffers because the rigorous connection between the numerical field solution and each FOM can be difficult to establish. For example, while very popular as FOMs, point values of the solution are often much more sensitive than integrated quantities and, when reliable error estimates can be obtained for them, the estimated errors are comparatively greater.

Generally, convergence can be proven only under extremely limited circumstances. For example, a discontinuous solution of a hyperbolic system of equations can be proven to converge in the L1 norm of the solution [Maj77]. Other functional relations will not have a precisely defined convergence rate or character. More broadly speaking, the convergence depends upon the smoothness of the solution: the smoother the solution (i.e., having higher derivatives that are well-defined), then the better the chances for convergence in higher norms. The max norm (L_∞) is generally the most difficult norm in which to obtain (or expect) convergence. The max norm is the largest error in the solution and it is defined at a single location. FOMs involving only a single point in the solution often mimic the behavior of the L_∞ norm, thus carrying the same expectations for convergence.

Observed Convergence Behavior

Given a well-defined metric and a series of at least three calculations on successively refined meshes, there are five generic outcomes of a convergence analysis. These possibilities are depicted notionally in Fig. 1. The diagrams in that figure show that the inferred behavior can be: (1) monotonic or oscillatory, and (2) convergent or divergent. The fifth possible behavior, shown in the center, is neither monotonic nor oscillatory, neither convergent nor divergent: instead, this limiting case exhibits a bounded behavior, suggested by the dotted line. Of these possibilities, the monotonically convergent case is the “ideal” case that one desires to occur for a consistent, stable numerical scheme on a problem. The other cases comprise the set of possible non-ideal behaviors, which are sometimes observed in practice.

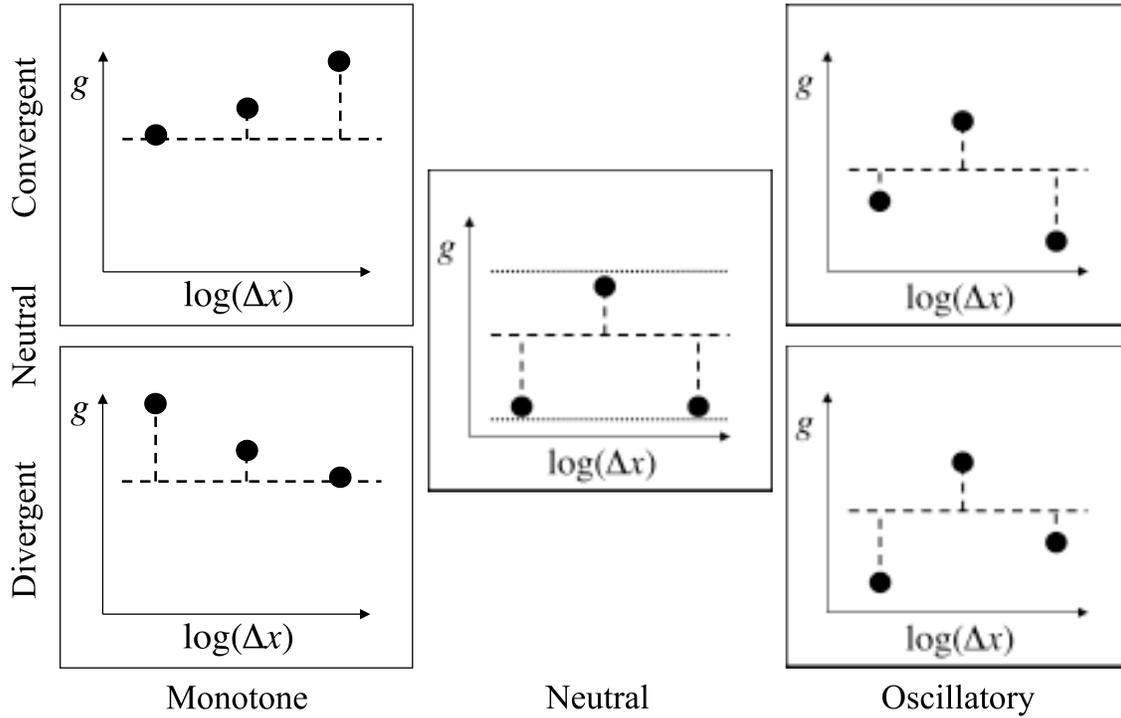


Figure 1. Schematic representations of possible behavior for computed solutions under mesh refinement. The individual plots are notional graphs of computed solution, g , versus the log of the mesh spacing of the calculation, Δx ; the dashed lines are intended only to guide the eye. The vertical axis in the overall figure goes from divergent to convergent, and the horizontal axis from monotone to oscillatory. The ideal case of monotone convergence under mesh refinement is in the upper left of this figure, and the limiting case of neutral behavior is in the center.

Ideal Asymptotic Convergence Analysis

In this section, we examine the case of ideal asymptotic convergence analysis. The axiomatic premise of asymptotic convergence analysis is that the computed difference between the reference and computed solutions can be expanded in a series based on some measure of the discretization of the underlying equations. Taking the spatial mesh as the obvious example, the ansatz for the error in a 1-D simulation is taken to be

$$\|g^f - g^c\| = A_0 + A_1(\Delta x)^\alpha + o((\Delta x)^\alpha) \quad (3)$$

In this relation, g^f is the reference solution, which for calculation verification is computed on a refined mesh, g^c is the computed solution, Δx is some measure of the mesh-cell size, A_0 is the zero-th order error, A_1 is the first order error, and the notation “ $o((\Delta x)^\alpha)$ ” denotes terms that approach zero faster than $(\Delta x)^\alpha$ as $\Delta x \rightarrow 0^+$. For consistent numerical solutions, A_0 should be identically zero; we assume this to be the case in the following discussion. For a consistent solution, the exponent α of Δx is the convergence rate: $\alpha=1$ implies first-order convergence, $\alpha=2$ implies second order convergence, etc.

Assume that the calculation has been run on a “coarse” mesh (subscript c), and a very coarse mesh (subscript vc) characterized by Δx_c (Δx_{vc}), which we hereafter also denote as Δx . The error ansatz implies:

$$\|g^f - g^c\| = A_0 + A_1 (\Delta x)^\alpha + \dots \quad (4)$$

Let us further assume that we have computational results on a “fine” mesh Δx_f (subscript f), where $0 < \Delta x_f < \Delta x_c$ with $\Delta x_c / \Delta x_f \equiv \sigma > 1$. In this case, the error ansatz implies:

$$\|g^f - g^c\| = \sigma^{-\alpha} A_1 (\Delta x)^\alpha + \dots \quad (5)$$

Manipulation of these two equations leads to the following explicit expressions for the quantities α and A_1 :

$$\alpha = \left[\log \|g^f - g^c\| - \log \|g^c - g^{vc}\| \right] / \log \sigma \quad (6)$$

$$A_1 = \|g^f - g^c\| / (\Delta x)^\alpha \quad (7)$$

These two equalities are the workhorse relations that provide a direct approach to convergence analysis as a means to evaluating the order of accuracy for code verification.

For QOIs or FOMs the above development can be utilized without resorting to error norms. The quantity, G , is defined without the use of a norm with the following related error model,

$$\tilde{G} = G^k + A_1 (\Delta x)_k^\alpha + \dots \quad (8)$$

with the remainder of the development proceeding as above provided the approach toward \tilde{G} , the mesh converged solution, is monotonic. In the case where a solution is not monotonically approached, the above error model can still be utilized as long as the error in absolute terms is diminishing monotonically. This recommendation is in some clear opposition to the existing literature although error norms

themselves are typically positive definite quantities. The more general cases are described next.

Non-Ideal Asymptotic Convergence Analysis

The term “non-ideal” refers to situations for which the error does not monotonically decrease with increasing mesh refinement in the simulation: the computed solutions may diverge under mesh refinement, in either a monotonic or non-monotonic fashion, or the computed values may have no overarching characteristics beyond simply being bounded. Some researchers contend that one cannot make further statements about calculations that do not exhibit the ideal asymptotically convergent behavior described in the previous section. Insofar as such non-ideal behavior is often seen in practical engineering simulations, however, this section contains a discussion of the analysis of such non-ideal cases.

Several authors have offered schemes by which the behavior of computed solutions on three or more grids can be categorized. These approaches seek to automatically classify the convergence behavior into one of the categories shown in Fig. 1. The salient parameters in these analyses are the theoretical convergence rate of the underlying numerical scheme, α_{th} , and the observed convergence rate, based on Richardson Extrapolation (see, e.g., [Obe10]), corresponding to the actual simulation results, α_{RE} . A study of this approach is given by Xing and Stern [Xin10], who build upon earlier work of Eça and Hoeckstra [Eça06]. The following discussion is based on these publications; for further background, see the earlier works of Logan and Nitta [Log05] and Stern et al. [Ste01]

For the estimation of convergence properties of scalar quantities, Eça and Hoeckstra [Eça06] approach the problem of estimating the observed convergence rate as a least-squares solution to the governing error ansatz equation,

$$|G^f - G^c| = A_1 (\Delta x)^\alpha + \dots \quad (9)$$

Let p denote the convergence rate corresponding to the least-squares solution for this equation based on the *signed* differences of results computed with adjacent grid spacings, i.e., where both positive and negative values are allowed on the left-hand side of Eq. (9). Similarly, let α^* represent the convergence rate corresponding to the *absolute* differences of results computed with adjacent grid spacings, i.e., where absolute values of the left-hand side of Eq. (9) are used. Then, the behavior of the set of computations is determined from the following algorithm:

```

If  $\alpha > 0$  then monotonic convergence
Else if  $\alpha < 0$  then monotonic divergence
Else if  $\alpha^* < 0$  then oscillatory divergence
Else oscillatory convergence

```

A second way to characterize the nature of the convergence is to examine the ratio of signed error in the solution from one mesh refinement level to the next,

$$R = (G^f - G^c) / (G^c - G^{vc}) = \Delta^{cf} / \Delta^{vc}, \quad (10)$$

If $R < 1$ then monotonic convergence
Else if $R > 1$ then monotonic divergence
Else if $R < -1$ then oscillatory divergence
Else $(-1 < R < 0)$ oscillatory convergence

Estimation Of Associated Numerical Uncertainty

Once the nature of the solution has been properly categorized, the numerical uncertainty can then be estimated as part of the overall uncertainty estimate.³ The Grid Convergence Index (GCI) of Roach (see [Roa98, Roa09]) is perhaps the original attempt to codify the numerical uncertainty associated with inferred convergence parameters. Roache [Roa98] claims that there is evidence for the numerical uncertainty based on the GCI method (with a safety factor of 1.25) to achieve a 95% confidence level. This approach was extended to the Correction Factor (CF) method of Stern et al. [Ste01]. Xing and Stern [Xin10], however, take issue with both of these approaches, stating, "...there is no statistical evidence for what confidence level the GCI and CF methods actually achieve" and, more specifically, that their analyses "...suggest that the use of the GCI₁ method is closer to a 68% than a 95% confidence level." As we describe below, Xing and Stern come to a different conclusion regarding an approach that does meet the 95% confidence level empirically.

Eça and Hoekstra [Eça06] propose heuristics by which to estimate the numerical uncertainty associated with fundamental behavior of a set of computed results. These suggestions appear to be based on the assumption that the underlying numerical scheme has a theoretical convergence rate of two; however, for many multiphysics (and some single-physics) problems, the theoretical convergence rate is unity, for which the specific prescription of [Eça06] should be modified. It is worth noting here that the convergence rate is both a function of the scheme employed and the nature of the solution sought itself. For example, a second-order method applied to a problem with a discontinuous solution cannot product a second-order convergent result. *Hence the expected theoretical convergence rate is to be considered a function of both the method used and the solution sought.*

Xing and Stern [Xin10] take a different approach. To evaluate the numerical uncertainty associated with these solution verification estimates, Xing and Stern performed a statistical analysis of 25 sets of computational data, covering a range of

³ The proceedings of the 1st, 2nd, and 3rd Workshops on CFD Uncertainty Analysis [Eça08] provide an interesting reference on many aspects of uncertainty analysis for CFD.

fluid, thermal, and structural simulations, to arrive at various parameters for their estimations of simulation uncertainty. The parameters obtained by Xing and Stern provide computational uncertainty estimates that demonstrably satisfy the 95% confidence level for the data sets upon which that analysis is based. They suggest that the formula below provides a safety factor with empirical, statistical support. We suggest following this approach whenever the grid sequence provides a convergent sequence.

$$U_{num} = FS|\delta_\alpha| = \begin{cases} (2.45 - 0.85P)|\delta_\alpha|, & \text{if } 0 < P \leq 1 \\ (16.4P - 14.8)|\delta_\alpha|, & \text{if } P > 1 \end{cases} \quad (11)$$

where $P = \alpha_{RE}/\alpha_{th}$ defines whether the solution is asymptotic in observed nature. The numerical error magnitude comes from the Richardson extrapolation toward the monotonically mesh converged solutions as

$$\delta_\alpha = \frac{G^f - G^c}{\sigma^\alpha - 1} \quad (12)$$

or the related error estimate for monotonically decreasing error as

$$\delta_\alpha = \frac{|G^f - G^c|}{\sigma^\alpha - 1} \quad (13)$$

In the case where the solution is not convergent, the numerical uncertainty should nonetheless be estimated, however rough those estimates may be. It is the authors' experience that users of codes will generally move forward with calculations and—without guidance to the contrary—may offer *no* numerical uncertainty estimates whatsoever. We maintain that this practice is potentially more dangerous than providing a weakly justified estimate. We offer the important caveat that this bound is not rigorously justified; instead, it is perhaps appropriately viewed as a heuristic estimate that can be produced given limited information. The simplest approach is to examine the range of solutions produced and multiply this quantity by a generous safety factor,

$$U_{num} = 3(\max G - \min G) \quad (14)$$

The safety factor, set to 3 in (14), might assume different values in different computational science applications. This heuristic approach is similar to that advocated by Eça and Hoekstra [Eça06].

An alternative approach would involve postulating that the numerical error has a portion that vanishes under mesh refinement and a portion that is constant,

$$|G^f - G^c| = \frac{1}{3}U_{num} + A_1 \Delta x \quad (15)$$

the uncertainty is then the constant portion, which for a divergent calculation will be larger than any of the values computed.

The errors associated with iterative (linear and nonlinear) solution of systems of equations can be approached similarly. We note that many investigations have sought to merely render these errors too small to be significant. While such an approach can work, it may not be generally possible in practice. A more complete approach is to quantify these errors in terms of the controllable numerical parameters, usually the stopping criteria for such iterations.

First, consider the dependence of the error in a solution on the stopping criteria,

$$\frac{\|r_k\|}{\|r_0\|} < \text{tol} \quad (16)$$

where $r_k = Ax_k - b$ is the residual, A is the linear system, b is the right hand side and x_k is the solution vector after the k th iteration. The error in the solution is then bounded as

$$\|\tilde{x} - x_k\| \sim \kappa(A) \frac{\|r_k\|}{\|r_0\|} \quad (17)$$

with $\kappa(A)$ being the condition number of A defined by the product of the operator norm of A^{-1} with the operator norm of A , which reduces to ratio of maximum to minimum eigenvalues in the case of matrices [Kel95]. The condition number of a matrix defined by the discretization of a PDE is related to the mesh spacing. For example an elliptic PDE will usually have a matrix representation that scales as $1/(\Delta x)^2$.

Returning to the estimation of error, the simplest numerical parameter to control is the tolerance of the linear solution. We can use the same approach as for mesh spacing and propose that the normed error in the solution be

$$\|g^f - g^c\| = B_1 (\text{tol})^\beta \quad (18)$$

The remaining procedures discussed above can be applied in this case, although we do not have sharp expectations about the power law dependence of the solution on the tolerance. We propose the use of a simple uncertainty estimate of

$$U_{tol} = 1.25 |\delta_\beta| \quad (19)$$

where

$$\delta_\beta = \frac{\|g^f - g^c\|}{q^\beta - 1}; q = \frac{(tol)^c}{(tol)^f} \quad (20)$$

The overall uncertainty from numerical approximation uses the RMS of these two estimates as

$$U_{approx} = \sqrt{U_{num}^2 + U_{tol}^2} \quad (21)$$

Verification Subtleties

There are several subtle but important—and, in some cases, open—issues associated with the estimation of the quantities mentioned above. While the following topics may be considered by some to be arcane, they should be borne in mind by those devising and conducting verification analyses, as well as by code analysts.

- **Nondimensionalization** The above discussion of the error ansatz and the associated convergence parameters contain no assumptions regarding the dimensions of the associated variables. Consequently, parameters in the resulting scaling relations (e.g., Eqs. 3 and 4) may have inconsistent units. One way to avoid this issue is to nondimensionalize all quantities prior to conducting such an analysis. For example, one can choose representative quantities G and X with which to nondimensionalize the computed quantity g and the representative mesh scale Δx :

$$g' = g/G \quad \text{and} \quad \Delta x' = \Delta x/X \quad (22)$$

The nondimensional error ansatz is posited to be

$$\|g'^f - g'^c\| = A_1 (\Delta x')^\alpha + \dots \quad (23)$$

where all terms in this equation are now dimensionless. In this case, care must be taken to nondimensionalize consistently throughout the analysis, and to properly dimensionalize results, e.g., if one were to use this relation to estimate errors at another mesh size.

- **Dimension** For problems in multiple space dimensions (e.g., 2-D Cartesian (x,y)), the spatial convergence analysis described above can be assumed to carry over directly, such that, e.g., the ansatz of Eq. 3 follows identically. That is, one typically does not assume separate convergence rates in separate coordinates. This is a reasonable assumption in almost all cases; the exception is time-convergence, since

the time-integration scheme for a PDE may be of different order than the spatial integrator. For a more thorough discussion and examples of combined space-time convergence, see [Hem05, Tim06].

- **Frame** Spatial convergence analysis is idealized to refer to a fixed mesh, i.e., the Eulerian frame. Approaches have been taken to extend convergence analysis simplistically to the Lagrangian frame (e.g., [Kam03]). More sophisticated approaches, however, are needed. For example, since the fundamental Lagrangian equations are discretized with respect to *mass* and not *space*, an error ansatz analogous to Eq. 3 with the characteristic mesh discretization parameter Δx replaced by a characteristic mass discretization parameter Δm would be more faithful to the underlying formulation.
- **Non-uniform Meshes** The intention behind the expression “ Δx ” in Eq. 3 is that it is a meaningful measure of the characteristic length-scale of mesh cells of the discretized equations. If either adaptive mesh refinement (AMR) or an arbitrary Lagrangian-Eulerian (ALE) approach is used, however, such a quantity—if one exists—is likely to change during the course of a calculation. Again, straightforward approaches for non-uniform and AMR meshes have been examined (e.g., [Li05]), but these are topics of open research.
- **Norm Evaluation** The expression for the norm in Eq. 1 is appropriate, e.g., for Cartesian geometries. This term must be appropriately modified for non-Cartesian geometries. For example, for 1-D spherically symmetric calculations, the integral of the norm is properly expressed as

$$\|g\|_p \equiv \left(\int_a^b |g(x)|^p dx \right)^{1/p} = \left(\int_a^b |g(x)|^p 4\pi r^2 dr \right)^{1/p}. \quad (24)$$

In general, when evaluating the norm one must be mindful of the domain of the integral as well as any symmetries associated with the problem.

- **Norm Evaluation and Interpolation on Different Meshes** The expression for the convergence rate α in the calculation verification (Eq. 6) implies a direct comparison of computed solutions on three different meshes. The analogous expression (Eq. 6) for code verification requires an indirect comparison of computed solutions on two different meshes. To evaluate the differences of two calculations, a common mesh is required; this begs the question: should one extrapolate (“restrict”) fine-mesh values to the coarse mesh, or interpolate (“prolong”) coarse-mesh values onto the fine mesh? Margolin and Shashkov [Mar08] provide a rationale for the former: “...by moving each of the simulation results to the coarsest mesh, we average out the smaller scales and eliminate them as a source of error in studying convergence, thus isolating the discretization error.” The detailed manner by which one should move solutions between different meshes

remains an open research area. Particular attention should be paid to accurately interpolating solutions near discontinuities.

- Tracers as Figures of Merit A tracer is an imaginary particle that does not influence the simulations; it is effectively a point in the computational domain at which the values of the field variables are recorded over time and is used as a diagnostic. Most often tracers are Lagrangian (i.e., they move with the material) or Eulerian (i.e., they are associated with a fixed spatial location.) As FOMs, tracers present several challenges: they record point values, and, as mentioned earlier, point values may not converge at the same rate as the error integrated over the computational domain; they may be more sensitive than integrated quantities; interpolating field variables to tracer locations adds interpolation error (a form of discretization error) to the values recorded at the tracer; and, for tracers that move with respect to the mesh, the location of the tracer is itself subject to discretization error. For these reasons, FOMs associated with tracers can have relatively large numerical uncertainties. Research effort to address these uncertainties is warranted, however, as tracers remain a common type of FOMs.

Example — Direct Numerical Simulation Of Turbulence

In this section we use data from turbulent flow simulations to illustrate a sample calculation verification analysis focused on estimating the numerical uncertainty associated with the estimated discretization error. This analysis is conducted using a series of calculations carried out on a sequence of uniformly refined meshes.

The foundation of this approach is the standard error ansatz utilized in verification as discussed previously. Here, the quantity δ denotes the extrapolated difference in the solution from the fine grid to the desired mesh converged solution *assuming the convergence rate is constant*.

We first consider some idealized examples that illustrate possible outcomes for mesh refinement studies. The figures below provide a schematic representation of experiment and calculation comparison with increasing levels of mesh refinement (Figures 2 through 6). These figures are intended to give a notional representation of current practice and how it can advance toward desired practice. The figures begin with generally accepted current practice in fluid dynamics and show how, with modest additional effort, one can achieve improved analysis results including calculation verification. They do not, however, include a detailed uncertainty assessment, which is also needed.

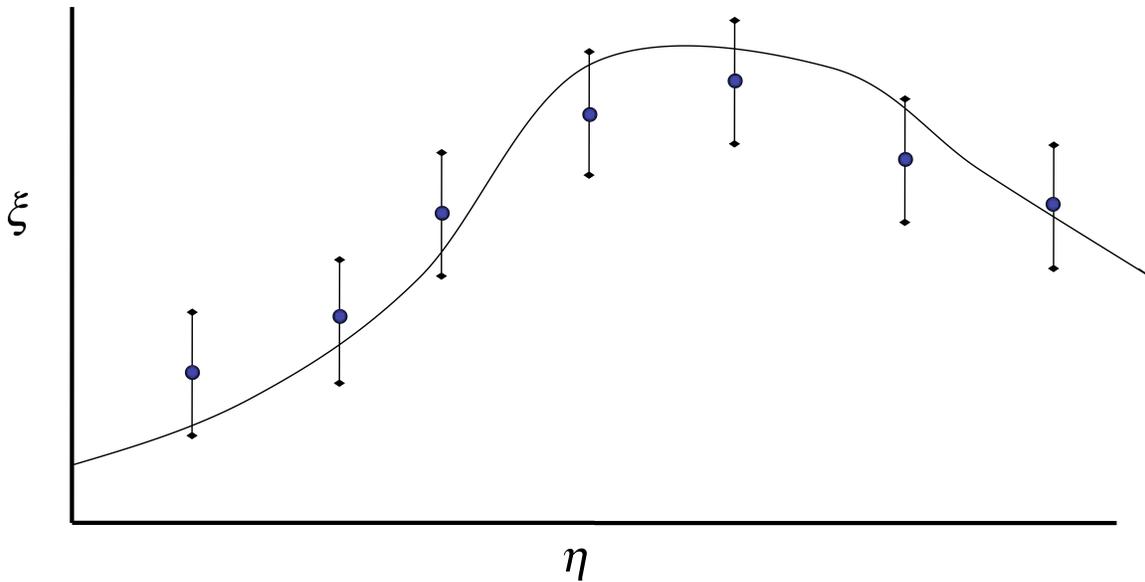


Figure 2. Typical comparison of experiment and calculation. The line denotes the calculation and the symbols denote the data with error bars.

Figure 2 depicts the fundamental comparison of experiment data (the symbols and associated vertical error bars) with computational results (the continuous line). The graphical comparison of these values in this example provides some qualitative evidence that a single calculation and the experiment are approximately congruent.

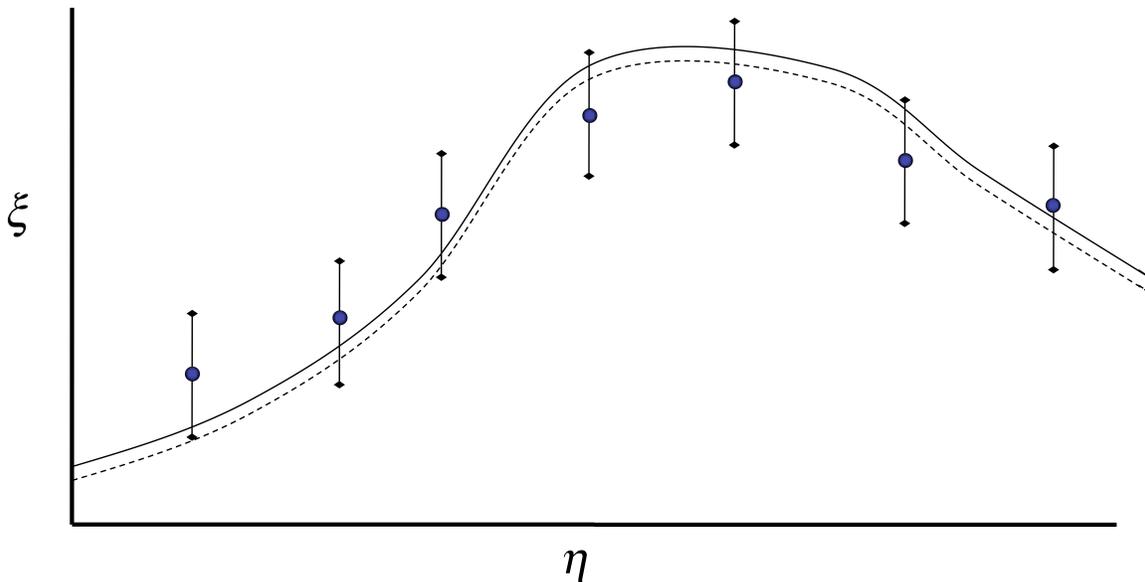


Figure 3. The usual mesh sensitivity representation. The dashed line is a more coarsely discretized calculation, and the “closeness” of the lines is meant to reassure the observer of the accuracy of the calculation. This *does not* constitute verification of the calculation. This is an example of how mesh sensitivity is often presented, that is the two calculations appear to be close, but no convergence is quantitatively assessed.

A useful next step is illustrated in Fig. 3, which adds the results of a comparable *but more coarsely discretized* calculation (dashed line) to the basic experiment-calculation graphic of Fig. 2. The evidence provided by such a plot can provide qualitative hints (i.e., mesh sensitivity) as to whether the computed results might approach or diverge from the experimental data under mesh refinement. Also, the resource requirements of the additional calculation (on a more coarsely discretized mesh) are reduced relative to the original calculation. Such graphical comparisons do not, however, constitute any form of verification of the calculation. The procedures we describe in this document provide the means to produce such a verification analysis.

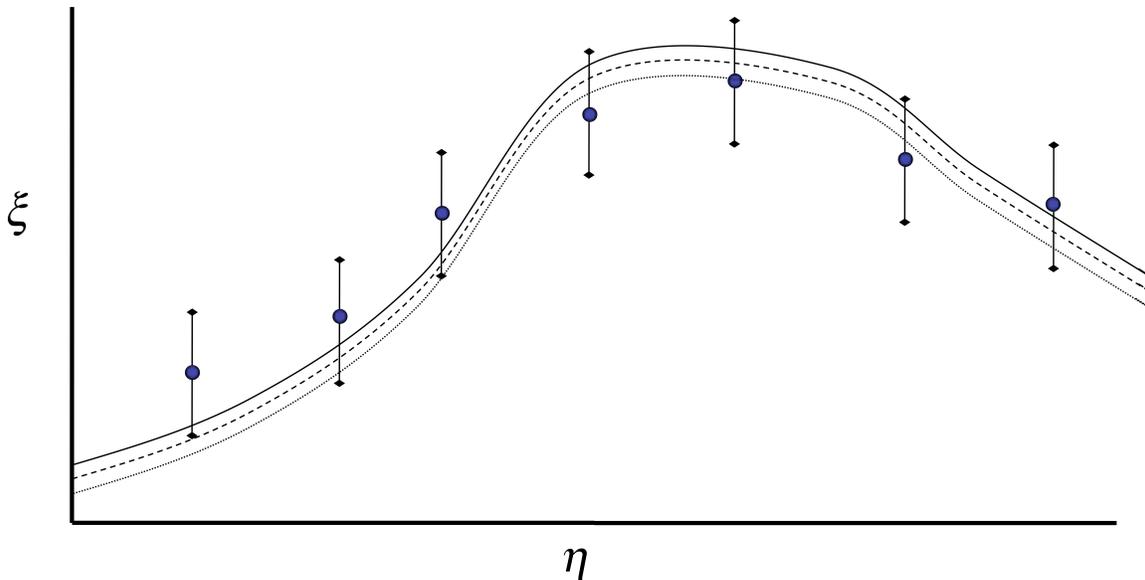


Figure 4. The finest grid used is denoted by the solid line (which is above the dotted line), and the coarsest grid result is given by the dotted line. In the case depicted, the error in the calculations does not shrink as the grid is refined, an indication of non-convergent behavior. The danger of mesh sensitivity is illustrated here with this non-convergent sequence of calculations.

A valuable further step is illustrated in Fig. 4, which adds to Fig. 3 the results of *an even more coarsely discretized calculation* (dotted line), with a mesh resolution ratio equal to that of the results in Fig. 3. In the situation depicted, the difference between any two pairs of nearby calculations *remains approximately constant* under a uniform refinement. Such results imply that the calculations are *not converging*. It is an example of possible monotone, divergent behavior.

Independent of the lack of convergence of the calculations under mesh refinement, the evidence provided in this notional plot provides qualitative evidence suggesting that the computed results diverge from the experimental data under mesh refinement. And, again, the resource requirements for this third calculation are reduced relative to the previous two calculations. This plot alone does not constitute verification; however, with this third calculation, the necessary information is now available to evaluate quantitative error and convergence rate estimates.

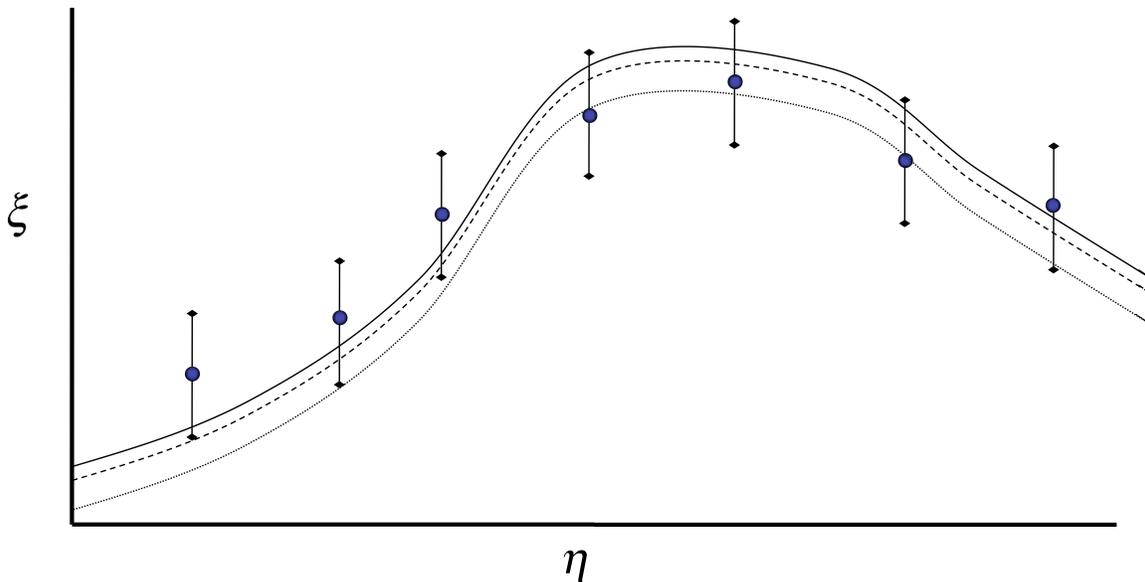


Figure 5. A first order convergent example is shown in schematic form. The finest grid is the black solid line and the dashed lines correspond to two coarser resolutions. For a mesh doubling procedure in which a factor of two, the level of error in the calculation with the refined the mesh spacing also halves as the grid is refined. For example, a second order result would produce an error level that is reduced by a factor of four as the grid is refined.

A different possible outcome of a third calculation is shown in Fig. 5. The most coarsely discretized calculation (dotted line), again with a mesh resolution ratio equal to that of the results in Fig. 3, now suggests that the difference between any two pairs of nearby calculations *decreases under uniform refinement*. That is, the difference between the dotted and dashed line looks greater than the difference between the dashed and solid lines. Thus, this plot provides qualitative evidence that the sequence of computations *appear to be converging* under mesh refinement. This is an example of possible monotone, convergent behavior. Again, this plot alone does not constitute verification, but it contains the necessary information to evaluate quantitative error and convergence rate estimates.

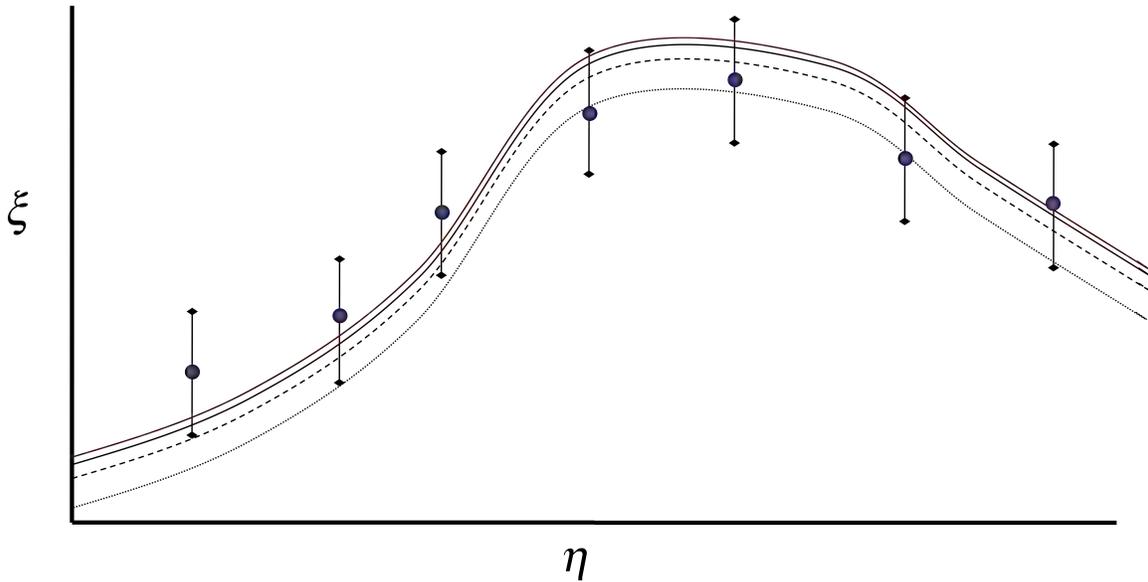


Figure 6. A mesh sequence and extrapolated solution is shown. The solution extrapolated from the finest grid solution is the solid red line. The finest grid is the black solid line and the dashed lines correspond to two coarser resolutions.

Figure 6 contains includes the graphical results of such a quantitative analysis. The red line in this diagram represents the extrapolated solution, which is computed based on the convergence rate estimate obtained from the sequence of three calculations. This extrapolated solution can be compared against the experimental data to provide an estimate of the error between experiment and simulation.

We now turn to a quantitative analysis of such a situation. This analysis is based on the scalar values calculated for turbulent mixing simulations of Donzis [Don07]. In the following, we apply calculation verification to the simulations defined in Table 1 below. The values in this table relate to characteristics of the different simulations. Note that, per the value of grid points (N^3) reported in the second row, there is a uniform mesh refinement ratio between computed results. Also, the value in the fourth row, $k_{max} \eta$, the product of the maximum wavenumber resolvable on the mesh (k_{max}) and the Kolmogorov scale (η), corresponds to the inverse of the mesh size, $1/\Delta x$, so that smaller values of $k_{max} \eta$ represent coarser mesh resolution.

Table 2 contains results of the calculations and simulations for various ensemble-averaged moments of the dissipation ϵ' and enstrophy Ω . The values contained in this table are examples of statistics that one would expect to be well behaved in a direct numerical simulation (DNS). For the $R_\lambda \approx 140$ case, there are four calculations at four different mesh resolutions, with each pair of mesh spacings differing by a factor of two, while for the $R_\lambda \approx 240$ case, there are computed results at three

different mesh resolutions. Recall that smaller values of $k_{max}\eta$ correspond to coarser mesh resolution. We analyze the results in Table 2 using Eqs. (10)–(14).

Table 1. Table 3.1 from [Don07], containing several computed quantities for turbulent mixing simulations conducted on four different meshes. The row starting with N^3 contains the number of mesh points used in the corresponding calculations.

Table 3.1: DNS parameters: Taylor-microscale Reynolds number $R_\lambda = u'\lambda/\nu$, number of grid points N^3 , viscosity ν , resolution measured by $k_{max}\eta$ and $\Delta x/\eta$, number of independent realizations N_r , and length of the simulation T normalized by eddy turnover time $T_E = L/u'$.

R_λ	140	140	140	140	240	240	240
N^3	256^3	512^3	1024^3	2048^3	512^3	1024^3	2048^3
ν	0.0028	0.0028	0.0028	0.0028	0.0011	0.0011	0.0011
$k_{max}\eta$	1.4	2.8	5.7	11.1	1.4	2.8	5.4
$\Delta x/\eta$	2.10	1.05	0.52	0.27	2.08	1.04	0.55
N_r	11	16	18	11	13	12	14
T/T_E	10.0	7.2	8.5	6.0	9.4	5.4	5.4

Table 2. Table 3.2 from [Don07], containing several computed quantities for turbulent mixing simulations conducted on four different meshes for two cases, $R_\lambda \approx 140$ (top) and $R_\lambda \approx 240$ (bottom).

Table 3.2: Ensemble averaged moments of dissipation and enstrophy at $R_\lambda \approx 140$ (top) and 240 (bottom) with 90% confidence intervals.

$R_\lambda \approx 140$

$k_{max}\eta$	1.4	2.8	5.7	11.1
$\langle\langle\epsilon'\rangle^2\rangle$	2.53 ± 0.04	2.85 ± 0.07	2.77 ± 0.06	2.82 ± 0.08
$\langle\langle\epsilon'\rangle^3\rangle$	14.1 ± 0.6	21.5 ± 1.6	19.9 ± 1.4	20.7 ± 2.1
$\langle\langle\epsilon'\rangle^4\rangle$	153 ± 14	388 ± 58	341 ± 48	364 ± 81
$\langle\epsilon^4\rangle/\langle\epsilon^2\rangle^2$	23.9	47.8	44.5	45.8
$\langle\langle\Omega'\rangle^2\rangle$	4.52 ± 0.09	5.19 ± 0.18	5.07 ± 0.19	5.20 ± 0.23
$\langle\langle\Omega'\rangle^3\rangle$	63.0 ± 3.1	100.0 ± 9.3	94.2 ± 9.9	97.6 ± 13.1
$\langle\langle\Omega'\rangle^4\rangle$	2022 ± 179	5315 ± 989	4920 ± 965	4751 ± 1200
$\langle\Omega^4\rangle/\langle\Omega^2\rangle^2$	99.2	197.1	191.3	175.9

$R_\lambda \approx 240$

$k_{max}\eta$	1.4	2.8	5.4
$\langle\langle\epsilon'\rangle^2\rangle$	3.07 ± 0.05	3.17 ± 0.07	3.15 ± 0.06
$\langle\langle\epsilon'\rangle^3\rangle$	25.3 ± 1.3	29.1 ± 1.8	28.8 ± 1.7
$\langle\langle\epsilon'\rangle^4\rangle$	488 ± 53	696 ± 83	697 ± 89
$\langle\epsilon^4\rangle/\langle\epsilon^2\rangle^2$	51.9	69.3	70.4
$\langle\langle\Omega'\rangle^2\rangle$	5.81 ± 0.13	5.99 ± 0.18	5.93 ± 0.12
$\langle\langle\Omega'\rangle^3\rangle$	133 ± 8	150 ± 14	142 ± 9
$\langle\langle\Omega'\rangle^4\rangle$	8364 ± 1017	11222 ± 1869	10211 ± 1503
$\langle\Omega^4\rangle/\langle\Omega^2\rangle^2$	247.7	312.8	290.6

Data Reduction Using Calculation Verification

The results will be displayed below for some representative quantities taken from these tables. For the $R_{\lambda} \approx 140$ case, the first mesh sequencing looks positive, but the last mesh casts doubt on these inferences, as the convergence rates drop in every case. Consequently the numerical uncertainty grows larger and, in some cases, becomes undetermined due to divergence. In the case of these calculations the nature of the flow including the formation of (quasi-)singular structures would indicate the expected convergence rate to be first-order irrespective of the numerical method used.

Before presenting results, some comments are in order. First, all the differences shown use the absolute value of the difference in the solutions; hence, the numerical uncertainty is plus or minus the indicated value. If the solutions are monotonically approaching a value, then the numerical uncertainty would be one sided (for example $\langle(\varepsilon')^2\rangle$), and the appropriate error bar would be asymmetric. The fourth grid is given the subscript of “v” to denote “very fine”. All mesh refinement ratios, r , are two. The large change in the convergence rate may indicate that the solutions might not be converged even at the finest mesh used. A situation that would provide a higher degree of confidence in the solution would show similar convergence rates for quantities across the full sequence of meshes used. Divergence indicates that these values may be unreliable from the perspective of numerical simulation.

The results of our analysis of the $R_{\lambda} \approx 140$ case are given in Table 3. In this table, Δ_{cm} denotes the difference between the coarse and medium solutions (see Eq. (10)), α_{cf} is the inferred convergence rate using the coarse-medium-fine results (see Eq. (10)), and U_{cf} gives an estimate of the associated numerical uncertainty on the finest of the corresponding meshes (see Eqs. (11-14)). The analogous values subscripted mf, fv , and mv correspond to calculated results using the medium, fine, and very fine computational grids.

We evaluate the inferred uncertainty results by comparing the computed values of numerical uncertainty U (in the last two columns of Tables 3 and last column of Table 4) with the corresponding statistical (i.e., ensemble) uncertainty values in the corresponding quantities in Table 2 (i.e., the values immediately after the “ \pm ” sign). Inspection of these values shows that on the coarser sequence of grids, the numerical errors are larger in magnitude than the statistical uncertainty (with similar confidence bands 90% for DNS and 95% for the numerical uncertainty estimators). For example, for the dissipation quantity given by $\langle(\varepsilon')^2\rangle$ in the $R_{\lambda} \approx 140$ case, the numerical uncertainty associated with the discretization error on the coarse-medium-fine grid results, 0.48, is greater than the corresponding ensemble-averaged statistical uncertainty, 0.06. To repeat, the estimated numerical uncertainty associated with the discretization error is greater than—but of the same order-of-magnitude as—the ensemble-averaged statistical uncertainty.

The DNS is computed using a fourth-order Runge-Kutta time integrator and spectral differencing in space, which might lead to the conclusion that a fourth-order convergence should be expected. Moreover, using a fourth order expected convergence rate, the error estimates yield errors not supported by the change in solution to the very fine grid. This lends greater credence to our expected first-order convergence rate. On the finer grid sequence, the numerical errors are much smaller by comparison, now comparable to the statistical errors measured.

Table 3. The results for the $R_\lambda \approx 140$ calculations.

Quantity	Δ_{cm}	Δ_{mf}	Δ_{fv}	α_{cf}	α_{mv}	U_{cf}	U_{mv}
$\langle(\varepsilon')^2\rangle$	0.32	0.08	0.05	2.00	0.68	0.48	0.16
$\langle(\varepsilon')^3\rangle$	7.40	1.60	0.80	2.21	1.00	9.46	1.28
$\langle(\varepsilon')^4\rangle$	235.00	47.00	23.00	2.32	1.03	273.54	46.48
$\langle\varepsilon^4\rangle/\langle\varepsilon^2\rangle^2$	23.90	3.30	1.30	2.86	1.34	16.94	6.12
$\langle(\Omega)^2\rangle$	0.67	0.12	0.13	2.48	-0.12	0.68	0.39
$\langle(\Omega)^3\rangle$	37.00	5.80	3.40	2.67	0.77	31.32	8.65
$\langle(\Omega)^4\rangle$	3293.00	395.00	169.00	3.06	1.22	1904.57	668.18
$\langle\Omega^4\rangle/\langle\Omega^2\rangle^2$	97.90	5.80	15.40	4.08	-1.41	19.02	63.60

*denotes a divergent calculation.

The results of our analysis of the $R_\lambda \approx 240$ case are given in Table 4. In this table, all the quantities are converging at a higher than expected rate. Given the results at the lower Reynolds number, we expect the uncertainty estimates to decrease as the mesh is refined. For most quantities given, the numerical error is larger than the statistical variability measured. For example, for the enstrophy, $\langle(\Omega')^2\rangle$, in the $R_\lambda \approx 240$ case, the numerical uncertainty associated with the discretization error on the coarse-medium-fine grid results, 0.34, is greater than the corresponding ensemble-averaged statistical uncertainty, 0.12.

Table 4. The results for the $R_{\chi} \approx 240$ calculations.

<i>Quantity</i>	Δ_{cm}	Δ_{mf}	α_{cf}	U_{cf}
$\langle(\epsilon')^2\rangle$	0.10	0.02	2.32	0.12
$\langle(\epsilon')^3\rangle$	3.80	0.30	3.66	1.16
$\langle(\epsilon')^4\rangle$	208.00	1.00	7.70	0.54
$\langle\epsilon^4\rangle/\langle\epsilon^2\rangle^2$	17.40	1.10	3.98	3.75
$\langle(\Omega)^2\rangle$	0.18	0.06	1.58	0.34
$\langle(\Omega)^3\rangle$	17.00	8.00	1.09	21.58
$\langle(\Omega)^4\rangle$	2858.00	1011.00	1.50	5416.22
$\langle\Omega^4\rangle/\langle\Omega^2\rangle^2$	65.10	22.20	1.55	122.40

We note that the discretization error is the only type of error estimated in this example. The linear system of equations associated with the solving for the pressure in these calculations is accomplished “exactly” (i.e., the iteration errors are decreased to the roundoff limit) through a spectral transformation, and the time integration is explicit for all nonlinear terms. Nonetheless, even when the mesh resolution is extremely coarse by traditional standards of the turbulent flow community, the numerical uncertainty associated with discretization error is nearly equal or larger than the statistical uncertainty associated with turbulence.

Detailed Workflow

Here, we expand on the details of the proposed workflow. There are other workflow approaches (see, e.g., [Joh06]), and the steps described below are by no means exhaustive. The proposed steps do, however, standardize a calculation verification workflow that can be conducted by a code team (developers and testers) for the purpose of estimating numerical uncertainty. Ideally, the code verification process should be conducted regularly (as well as on demand) so that incorrect implementations impacting mathematical correctness are detected as soon as possible. The general consensus in software development is that the cost of bugs is minimized if they are detected as close as possible to their introduction.

This procedure assumes that the code team is using a well-defined software quality assurance (SQA) process, and the code verification is integrated with this activity. Such SQA includes source code control, regression testing, and documentation, together with other project management activities. For consistency and transparency, we recommend performing the code verification in the same manner and using the same type of tools as other SQA processes.

1. **Starting with an implementation (i.e., code) that has passed the appropriate level of SQA and code verification scrutiny, choose the executable to be examined.** Calculation verification can be a resource-intensive activity involving substantial effort to perform. Calculation verification should be applied to the same version of the code that analysts would use for any important application. The notion that verification and validation should be applied to the same code is important to keep in mind. This process should be applied to the specific version of the code used throughout the entire V&V UQ activity.
2. **Provide an analysis of the numerical method as implemented including accuracy and stability properties.** The analysis should be conducted using any one of a variety of standard approaches. Most commonly, the von Neumann-Fourier method could be employed. For nonlinear systems, the method of modified equation analysis can be used to define the expected rate and form of convergence. The form and nature of the solution being sought can also influence the expected behavior of the numerical solution. For example, if the solution is discontinuous, the numerical solution will not achieve the same order of accuracy as for a smooth solution. Finite element methods can be analyzed via other methods to define the form and nature of the convergence (including the appropriate norm for comparison).
3. **Produce the code input to model the problem(s) for which the code verification will be performed.** Each problem is run using the code's standard modeling interface as for any physical problem that would be modeled. It can be a challenging task to generate code input that correctly specifies a particular problem⁴; e.g., special routines to generate particular initial or boundary conditions that drive the problem may be required, and these routines must be correctly interfaced to the code. It is advisable to consider the complexities and overhead associated with such considerations prior to undertaking such code verification analyses.
4. **Select the sequence of discretizations to be examined so each solution.** Verification necessarily involves convergence testing, which requires that the problem be solved on multiple discrete representations (i.e., grids or meshings). This is consistent with notions associated with h -refinement, although other sorts of discretization modification can be envisioned. The mathematical aspects of verification are typically most conveniently carried out if the discretizations are factors of two apart.
5. **Run the code and provide of means of producing appropriate metrics to compare the numerical solutions.** The solutions to the problem are computed on the discretizations. Most commonly and as discussed above,

⁴ Trucano et al. [Tru06] refer to this concept as the “alignment” between a code and a specific problem (either verification or validation).

these metrics take the form of norms (i.e., p -norms such as the $L2$ or energy norm). The selection of metrics is inherently tied to the mathematics of the problem and its numerical solution. The metrics can be computed over the entire domain, in subsets of the domain, on surfaces, or at specific points. The domain over which the metrics are evaluated and the analysis to be conducted must be free of any spurious solution features (due, e.g., to numerical waves erroneously reflected from computational boundaries).

6. **Use the comparison to determine the sequence of errors in the discretizations.** Using the well-defined metrics for each solution, the error can be computed for each discrete representation. Ideally, there will be a set of metrics available, providing a more complete characterization of the problem and its solution.
7. **The error sequence allows the determination of the rate-of-convergence for the method, which is compared to the theoretical rate.** With a sequence of errors in hand, the demonstrated convergence rate of the code for the problem is estimated. The theoretical convergence rate of a numerical method is a key property. Verification relies upon comparing this rate to the demonstrated rate of convergence. Evidence supporting verification is provided when the demonstrated convergence rate is consistent with the theoretical rate of convergence. This can be a difficult inference to draw, because the theoretical rate of convergence is a limit reached in an asymptotic sense, which cannot be attained for any finite discretization. As a consequence, there are unavoidable deviations from the theoretical rate of convergence, to which judgment must be applied.
8. **Using the results, render an assessment of the method's implementation correctness.** Based on the discrete solutions, errors, and convergence rate, a decision on the correctness of a model can be rendered. This judgment is applied to a code across the full suite of verification test problems.
 - a. The assessment can be positive, that is, the convergence rate is consistent with the method's expected accuracy.
 - b. The assessment can be negative, that is, the convergence rate is inconsistent with the method's expected accuracy.
 - c. The assessment can be inconclusive, that is, one cannot defensibly demonstrate clearly uniform consistency or inconsistency with the method's expected accuracy. For example, the convergence rate is nearly the correct rate, but the differences between the expected rate and the observed rate is uncomfortably large, potentially indicating a problem.

Figures 7a,b show the entire process in diagrams that conceptually expand the line for code verification in Fig. 4. This process should be repeatable and available on

demand. As noted in the introduction to this section, having the code verification integrated with the ongoing SQA activity and tools can greatly facilitate this essential property. The solution verification process is not monolithic, but instead it is flexible and should meet the needs of the specific application. For this reason we include two versions of the flowchart to facilitate this mindset.

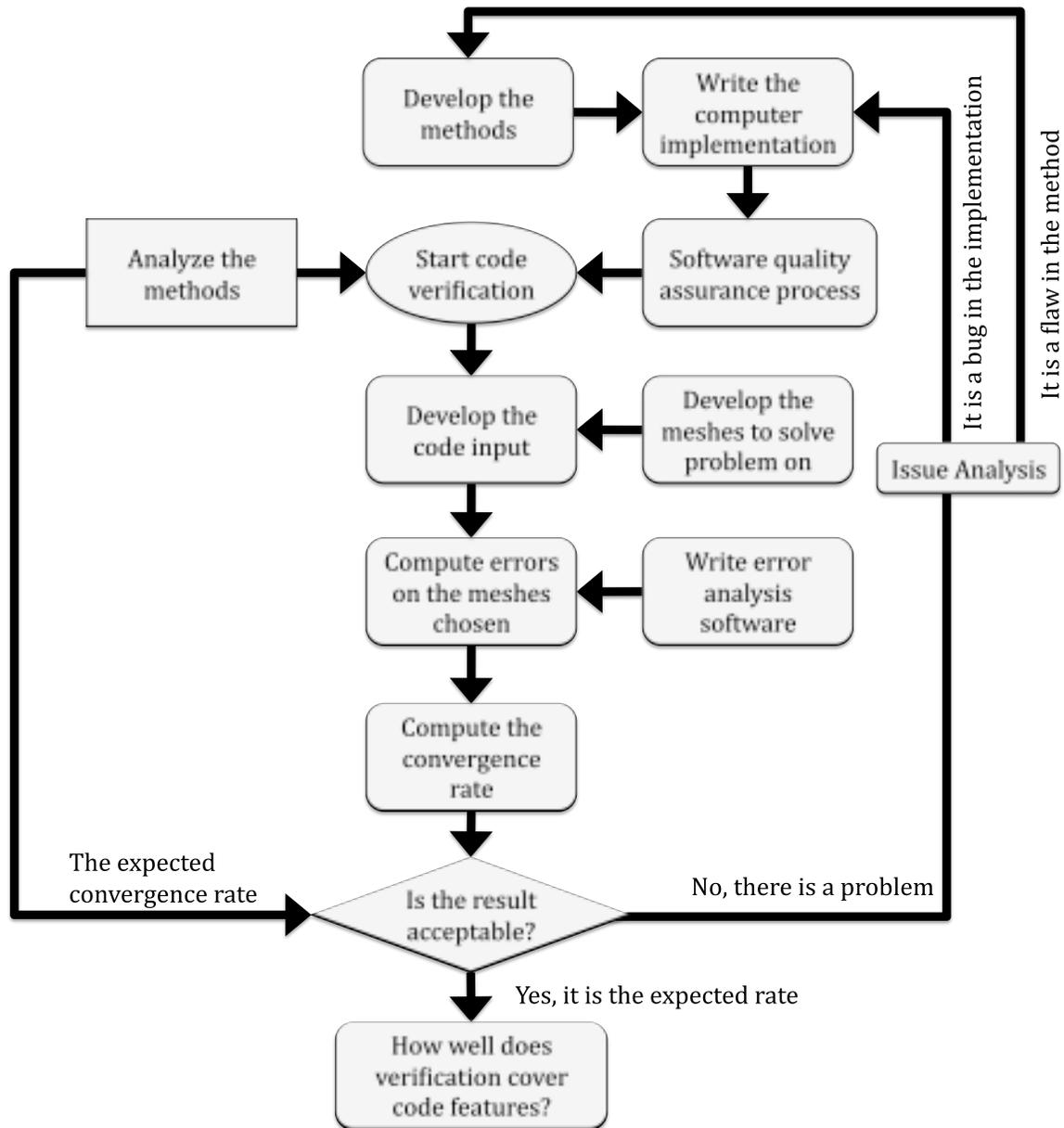


Figure 7(a). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

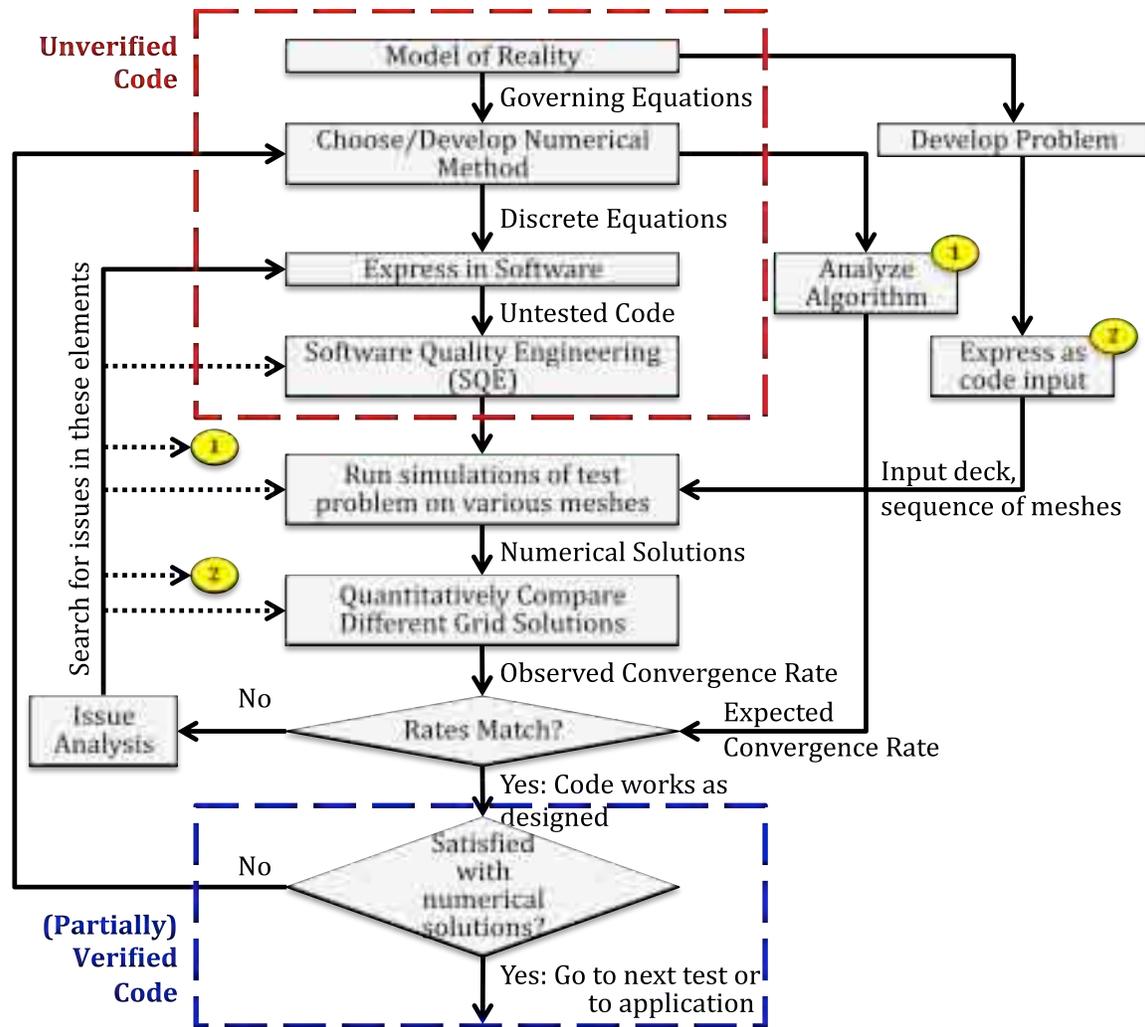


Figure 7(b). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

Conclusions

In this document, we have described the concept of verification, a well-defined process by which the correctness of the implementation of a numerical algorithm in scientific software can be evaluated. There are two complementary activities: (i) code verification, which quantitatively compares the theoretical order of accuracy of a method with the empirical order of accuracy for a problem with a known solution, and (ii) calculation verification. We have provided a detailed workflow for conducting calculation verification.

While approaches to verification are reasonably well codified and fairly widely used, there remain aspects of these analyses that can be difficult to resolve. Most verification cases encountered by the code user will likely be typical; however, difficult cases almost always arise eventually. Unless the analyst has chosen

exceedingly simple problems, particular verification problems may present their own challenges that will require insight, innovation, and determination on the part of the analyst to resolve. Despite these obstacles, verification is a necessary part of the “due diligence” of a scientific code development project and an essential element in producing superior simulation code that can be used for high-consequence analysis and decision-making.

Acknowledgement

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000

References

- [AIA98] AIAA, “Guide for the verification and validation of computational fluid dynamics simulations,” American Institute of Aeronautics and Astronautics, Reston, VA, AIAA-G-077-1998 (1998).
- [ASM06] ASME, *V&V 10 - 2006 Guide for Verification and Validation in Computational Solid Mechanics*, American Society of Mechanical Engineers, New York, NY, (2006).
- [Don07] Donzis, D. A., *Scaling of Turbulence and Turbulent Mixing Using Terascale Numerical Simulations*, Ph.D. Thesis, Georgia Institute of Technology, 2007.
- [Eça06] Eça, L., Hoekstra, M., “Discretization Uncertainty Estimation Based on a Least Squares Version of the Grid Convergence Index,” in *Proceedings of the Second Workshop on CFD Uncertainty Analysis*, Lisbon, Oct. 2006.
- [Eça08] Eça, L., Hoekstra, M., eds., *Proceedings of the 1st, 2nd, and 3rd Workshops on CFD Uncertainty Analysis*,
http://maretec.ist.utl.pt/html_files/CFD_workshops/
- [Eça09] Eça, L., Hoekstra, M., “Evaluation of numerical error estimation based on grid refinement studies with the method of manufactured solutions,” *Computers & Fluids* **38**, 1580–1591 (2009).
- [Han01] Hanson, K. M., Hemez, F. M., “A Framework for Assessing Confidence in Computational Predictions,” *Exp. Tech.* **25**, pp. 50–55 (2001).
- [Hem05] Hemez, F. M., *Non-Linear Error Ansatz Models for Solution Verification in Computational Physics*, Los Alamos National Laboratory report LA-UR-05-8228 (2005).
- [Joh06] Johnson, R. W., Schultz, R. R., Roache, P. J., Celik, I. B., Pointer, W. D., Hassan, Y. A., *Processes and Procedures for Application of CFD to Nuclear Reactor Safety Analysis*, Idaho National Laboratory report INL/EXT-06-11789 (2006).
- [Kam03] Kamm, J., Brock, J., Rousculp, C., Rider, W., *Verification of an ASCI Shavano Project Hydrodynamics Algorithm*, Los Alamos National Laboratory report LA-UR-03-6999 (2003).

- [Kam08] Kamm, J. R., Brock, J. S., Brandon, S. T., Cotrell, D. L., Johnson, B., Knupp, P., Rider, W. J., Trucano, T. G., Weirs, V. G., *Enhanced Verification Test Suite for Physics Simulation Codes*, Los Alamos National Laboratory report LA-14379, Lawrence Livermore National Laboratory report LLNL-TR-411291, Sandia National Laboratories report SAND2008-7813 (2008).
- [Kel95] Kelley, C. T., *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, PA (1995).
- [Knu03] Knupp, P., Salari, K., *Verification of Computer Codes in Computational Science and Engineering*, Chapman & Hall/CRC, Boca Raton, FL (2003).
- [Li05] Li, S., Rider, W. J., Shashkov, M. J., *Two-Dimensional Convergence Study for Problems with Exact Solution: Uniform and Adaptive Grids*, Los Alamos National Laboratory report LA-UR-05-7985 (2005).
- [Mar08] Margolin, L. G., Shashkov, M. J., "Finite volume methods and the equations of finite scale: A mimetic approach," *Int. J. Num. Meth. Fluids* **56**, pp. 991–1002 (2008).
- [Maj77] Majda, A., Osher, S., "Propagation of error into regions of smoothness for accurate difference approximations to hyperbolic equations," *Comm. Pure Appl. Math.* **30**, pp. 671–705 (1977).
- [Nel10] Nelson, R., Unal, C., Stewart, J., Williams, B., *Using Error and Uncertainty Quantification to Verify and Validate Modeling and Simulation*, Los Alamos National Laboratories report LA-UR-10-06125 (2010).
- [Obe02] Oberkampf, W. L., Trucano, T. G., "Verification and Validation in Computational Fluid Dynamics," *Prog. Aerospace Sci.* **38**, pp. 209–272 (2002).
- [Obe04] Oberkampf, W. L., Trucano, T. G., Hirsch C., "Verification, validation, and predictive capability in computational engineering and physics," *Appl. Mech. Rev.* **57**, pp. 345–384 (2004).
- [Obe07] Oberkampf, W. L., Trucano, T. G., "Verification and Validation Benchmarks," *Nuclear Design and Engineering* **23**, pp. 716–743 (2007); also available as Sandia National Laboratories report SAND2007-0853 (2007).
- [Obe10] Oberkampf, W. L., Roy, C. J., *Verification and Validation in Scientific Computing*, Cambridge University Press, New York (2010).
- [Ore94] Oreskes, N., Shrader-Frechette, K., Belitz, K., "Verification, validation, and confirmation of numerical models in the earth sciences," *Science* **263**, pp. 641–646 (1994).
- [Rid10] Rider, W. J., Kamm, J. R., Weirs, V. G., *Code Verification Workflow in CASL*, Sandia National Laboratories report SAND2010-7060P (2010).
- [Rid11] Rider, W. J., Kamm, J. R., Weirs, V. G., *Verification, Validation and Uncertainty Quantification Workflow in CASL*, Sandia National Laboratories report SAND2011-(to be determined)(2011).
- [Roa98] Roache, P., *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque (1998).
- [Roa09] Roache, P., *Fundamentals of Verification and Validation*, Hermosa Publishers, Albuquerque (2009).

- [Roy05] Roy, C. J., "Review of Code and Solution Verification Procedures for Computational Simulation," *J. Comput. Phys.* **205**, pp. 131–156 (2005).
- [Roy10a] Roy, C. J., "Review of Discretization Error Estimators in Scientific Computing," 48th AIAA Aerospace Sciences Meeting, January 2010, Orlando, FL, AIAA 2010-126 (2010).
- [Roy10b] Roy, C. J., Oberkampf, W. L., "A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing," submitted to *Comp. Meth. Appl. Mech. Engrng.* (2010).
- [Sar98] Sargent, R. G., "Verification and Validation of Simulation Models," in *Proceedings of the 1998 Winter Simulation Conference*, Medeiros, D.J., Watson, E.F., Carson, J.S., Manivannan, M.S., eds., pp. 121–130 (1998).
- [Sar01] Sargent, R. G., "Some Approaches and Paradigms for Verifying and Validation Simulation Models," in *Proceedings of the 2001 Winter Simulation Conference*, Peters, B. A., Smith, J. S., Medeiros, D.J., Rohrer, M. W., eds., pp. 106–114 (2001).
- [Sch06] Schwer, L. E. "An Overview of the PTC 60 / V&V 10 Guide for Verification and Validation in Computational Solid Mechanics," reprint by ASME, <http://cstools.asme.org/csconnect/pdf/CommitteeFiles/24816.pdf> (2006).
- [Sor07] Sornette, D., Davis, A. B., Ide, K., Vixie, K. R., Pisarenko, V., Kamm, J. R., "Algorithm for model validation: Theory and applications," *Proc. Nat. Acad. Sci. USA* **104**, pp. 6562–6567 (2007).
- [Ste01] Stern, F., Wilson, R. V., Coleman, H. W., Paterson, E. G., "Comprehensive Approach to Verification and Validation of CFD Simulations—Part 1: Methodology and Procedures," *J. Fluids Engrng* **123**, pp. 793–802 (2001).
- [Ste06] Stern, F., Wilson, R. V., Shao, J., "Quantitative V&V of CFD Simulations and Certification of CFD Codes," *Int. J. Num. Meth. Fluids* **50**, pp. 1335–1355 (2006).
- [Tim06] Timmes, F. X., Fryxell, B., Hrbek, G. M., *Spatial-Temporal Convergence Properties of the Tri-Lab Verification Test Suite in 1D for Code Project A*, Los Alamos National Laboratory report LA-UR-06-6444.
- [Tru03] Trucano, T. G., Pilch, M., Oberkampf, W. L., *On the Role of Code Comparisons in Verification and Validation*, Sandia National Laboratories report SAND2003-2752 (2003).
- [Tru06] Trucano, T. G., Swiler, L. P., Igusa, T., Oberkampf, W. L., Pilch, M., "Calibration, validation, and sensitivity analysis: What's what," *Reliab. Engrng. Syst. Safety* **92**, pp. 1331–1357 (2006).
- [Xin10] Xing, T., Stern, F., "Factors of Safety for Richardson Extrapolation," *J. Fluids Engrng* **132**, pp. 061403-1– 061403-13 (2010).
- [Zie92] Zienkiewicz, O. C., Zhu, J. Z., "The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity," *Int. J. Num. Meth. Eng.* **33**, pp. 1356–1382, (1992).