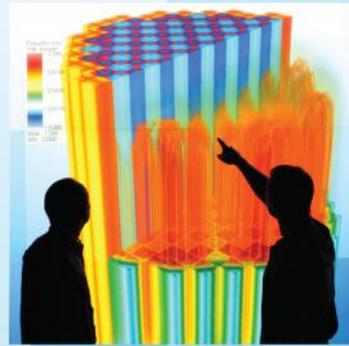




Power uprates
and plant life extension

CASL-U-2012-0132-000



Engineering design
and analysis

L3:VUQ.VVDA.P4.03

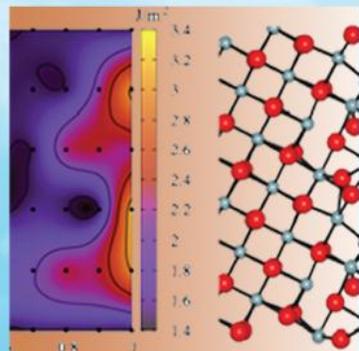
Jim Stewart

SNL

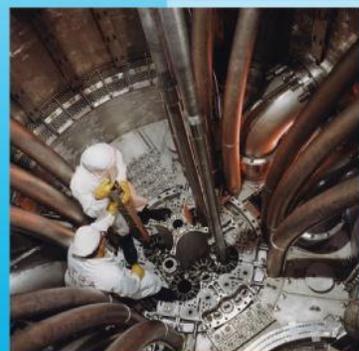
Completed: 8/31/2012



Science-enabling
high performance
computing



Fundamental science



Plant operational data



U.S. DEPARTMENT OF
ENERGY

Nuclear Energy

Advanced Solution Verification of CFD Solutions for LES of Relevance to GTRF Estimates.

William J. Rider and James R. Kamm
Sandia National Laboratories
Albuquerque, NM 87185
August 31, 2012

SAND 2012-7199P

Summary

The purpose of this work is to demonstrate advanced solution verification (i.e., numerical error estimation) techniques on computational fluid dynamics simulations of interest to CASL. The specific case chosen is the GTRF-related fluid response of a detailed subchannel fluid flow computed with the CFD code Drekar. For the full GTRF analysis, the necessary detail includes the time-dependent fluctuating fluid evolution within the section of the rod bundle simulated. We do not examine the structural-vibration response that is the ultimate objective of GTRF simulations. The specific verification analysis will include both standard and recently developed techniques for numerical error estimation, all based on the usual power law error description. In particular, the newer techniques provide a self-contained error and convergence analysis that includes confidence intervals for the results derived. The methodology is described in detail in this report. In addition, we will apply complementary techniques to the related issue of code verification of Drekar.

This work builds upon the basic theory and workflow described earlier by the authors in [Rid10,Rid11].

Introduction

Calculation or solution verification is a class of procedure where the discretization or numerical error is estimated in simulations of problems of interest. Such analyses constitute a specific form of uncertainty quantification. There are a number of defined procedures by which numerical error estimates can be converted to numerical uncertainty estimates. In this report we will review existing procedures and describe a new one. We take the two terms *calculation verification* and *solution verification* to be synonymous. *Code verification* is a related, but distinct process in which the correctness of a software implementation of a numerical algorithm is evaluated, typically by comparison against an exact solution. For the purpose of comparison, the new verification procedure introduced will be applied synergistically to code verification as well.

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 physical correctness of the continuum models (e.g., model-form errors). The process of examining model-form error is known as validation and is distinct from verification. The challenge of solution 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.

Our sole focus in this work will be the last of these, the discretization error. Fully verifying the veracity of our approach would require further study and additional calculations that cannot be justified given the difficulty of obtaining full-scale calculation for estimating discretization error. We fully acknowledge this as a weakness of the present study. In our defense, these errors have been studied by the Drekar team in the context of code verification and judged to be substantially smaller than the discretization error under those (admittedly idealized) circumstances [Shadid].

Discretization errors are a direct consequence of the numerical scheme used to obtain a discrete approximation of the continuous model equations (e.g., finite difference, finite element, or finite volume methods). The solution approach used on those discrete equations *and the nature of the solution itself* determine the expected behavior of the error. For time-dependent problems, both the spatial and temporal discretizations enter into the evaluation of these errors (for radiation transport, energy/spectral and angular discretization must be considered as well). 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. This error is related to how nonlinear aspects of an implicitly defined temporal solution are approximated. Indeed, with respect to the CFD calculations considered here, the large time step RANS modeling would be expected to produce relatively substantial nonlinear solver error. For the LES solutions, attention to the nonlinear error is less important; this is reflected in the ability of semi-implicit methods (explicit advection-implicit pressure) to solve the fluid problem.

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 (for spatial convergence) or time step (for temporal convergence). This is a

standard property of the numerical method; however, it formally applies only when the solution is continuously differentiable. 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 solution 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 under the *a priori* assumption that finer mesh resolution yields more accurate solutions.¹ This assumption is generally regarded to be reasonable, given its fundamental character with regard to numerical analysis.

To aid analysts in conducting solution verification analyses, the following workflow for solution 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 quantities of interest 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.

¹ Implicit in this assumption is the expectation that the quantity being measured is sufficiently well behaved, numerically, that convergence is a sensible concept. For example, in a turbulent flow, the value of the velocity at a particular location in the flow should not be expected to converge, but the (integrated) turbulent kinetic energy of a specified volume of the flow can reasonably be presumed to be convergent.

9. Examine the degree of coverage of features in an implementation by the verification testing.

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 given by numerical analysis (often obtained with the aid of Taylor series expansion). In practice, however, a series of 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 on the given problem. 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.

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

$$\|A_k - A_{k-1}\| = C_0 + C_1 h^p + o(h^p) \quad (3)$$

In this relation, A_k is the reference solution, which for solution verification is computed on a refined mesh, A_{k-1} is the computed solution, h is some measure of the mesh-cell size, C_0 is the zero-th order error, C_1 is the first order error, and the notation " $o((h)^p)$ " denotes terms that approach zero faster than $(h)^p$ as $h \rightarrow 0^+$. For consistent numerical solutions, C_0 should be identically zero; we assume this to be the case in the following discussion. For a consistent solution, the exponent p of h is

the convergence rate: $p = 1$ implies first-order convergence, $p = 2$ implies second order convergence, etc.

The error ansatz implies:

$$\|A_k - A_{k-1}\| = C_0 + C_1 h^p + \dots \quad (4)$$

Let us further assume that we have computational results on a “fine” mesh h_k (subscript k), where $0 < h_k < h_{k-1}$ with $h_{k-1} / h_k \equiv \sigma > 1$. In this case, the error ansatz implies:

$$\|A_k - A_{k-1}\| = \sigma^{-p} C h^p + \dots \quad (5)$$

Manipulation of these two equations leads to the following explicit expressions for the quantities p and C :

$$p = \left[\log \|A_k - A_{k-1}\| - \log \|A_{k-1} - A_{k-2}\| \right] / \log \sigma \quad (6)$$

$$C = \|A_k - A_{k-1}\| / h^p \quad (7)$$

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 quantities of interest (QOIs) or figures of merit (FOMs), the above development can be utilized without resorting to error norms. The quantity, A , is defined without the use of a norm with the following related error model,

$$\tilde{A} = A_k + C h^p + \dots \quad (8)$$

with the remainder of the development proceeding as above, if the approach toward \tilde{A} , 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.

Once the nature of the solution has been properly categorized, the numerical uncertainty can then be estimated as part of the overall uncertainty estimate.² 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

² 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.

GCI and CF methods an 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 technically does meet the 95% confidence level empirically, albeit with respect to a specific ensemble of simulations.

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 produce 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.*

We highlight the heuristic but simple estimation associated with Roache’s procedure as defined by Oberkampf and Roy [Obe10]. The simplicity of this estimate should be held in contrast to the more elaborate procedure described later. For both procedures, the starting point is a regression given the results of the mesh refinement (or coarsening) procedure. This produces a mesh-converged result, \tilde{A} , and convergence rate, p . From these values, one obtains the basic scale for the error estimate,

$$\delta_{\alpha} = \frac{A_k - A_{k-1}}{\sigma^p - 1} = \tilde{A} - A_k. \quad (9)$$

This value is processed with the convergence rate to define a safety factor,

$$U_{num} = F_s |\delta_{\alpha}| = \begin{cases} 1.25 |\delta_{\alpha}| & \text{if } |p - p_{theo}| / p_{theo} < 0.1 \\ 3 |\delta_{\alpha}| & \text{otherwise} \end{cases}. \quad (10)$$

Finally, the grid convergence index (GCI) is the ratio of U_{num}/\tilde{A} expressed as a percentage. The safety factors in (10) were chosen on the basis of expert judgment from extensive CFD experience.

Xing and Stern [Xin10] take a different, more complicated, but nevertheless still empirical 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 fluid, thermal, and structural simulations, to arrive at various parameters for their estimations of simulation uncertainty. The parameter values 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 contend 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 $P = p_{RE}/p_{th}$, the ratio of the Richardson extrapolation based convergence rate (p_{RE}) and the theoretical convergence rate (p_{th}), defines whether the observed solution is asymptotic in nature. The numerical error magnitude comes from the Richardson extrapolation toward the monotonically mesh converged solutions as

$$\delta_\alpha = \frac{A_k - A_{k-1}}{\sigma^p - 1} \quad (12)$$

or the related error estimate for monotonically decreasing error as

$$\delta_\alpha = \frac{A_k - A_{k-1}}{\sigma^p - 1} \quad (13)$$

In the case where the solution is not convergent, the numerical uncertainty should nonetheless be estimated, however imprecise those estimates may be. It is the authors' experience that users of codes will generally move forward with calculations and—absent 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; it is perhaps more appropriately viewed as a heuristic estimate, with documented provenance, that can be readily generated given limited information. The simplest approach is to examine the range of solutions produced and multiply this quantity by a generous safety factor,

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

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].

Detailed Workflow

Here, we reproduce the details of the proposed CASL solution verification workflow, which this work will demonstrate. The proposed steps do, however, standardize a solution 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 that 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. Solution verification can be a resource-intensive activity involving substantial effort to perform. It is important that verification and validation be applied to exactly the same code. Therefore, solution verification should be applied to the same version of the code that analysts would use for any important application. Indeed, 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 problem³; 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

³ Trucano et al. [Tru06] refer to this concept as the "alignment" between a code and a specific problem (either verification or validation).

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 differ by factors of two.

5. Run the code and provide of means of producing appropriate error metrics to compare the numerical solutions. The solutions to the problem are computed on the meshes corresponding to the different discretizations. Most commonly and as discussed above, these metrics take the form of norms (i.e., p -norms such as the L_2 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 is 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 computed solutions. 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 (e.g., L_1 , L_2 , and L_{∞}), 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, i.e., it 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(s), 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 either 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 unacceptably 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. As previously stated, 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, should be flexible and should meet the needs of the specific application. For this reason we include two versions of the flowchart to facilitate this mindset.

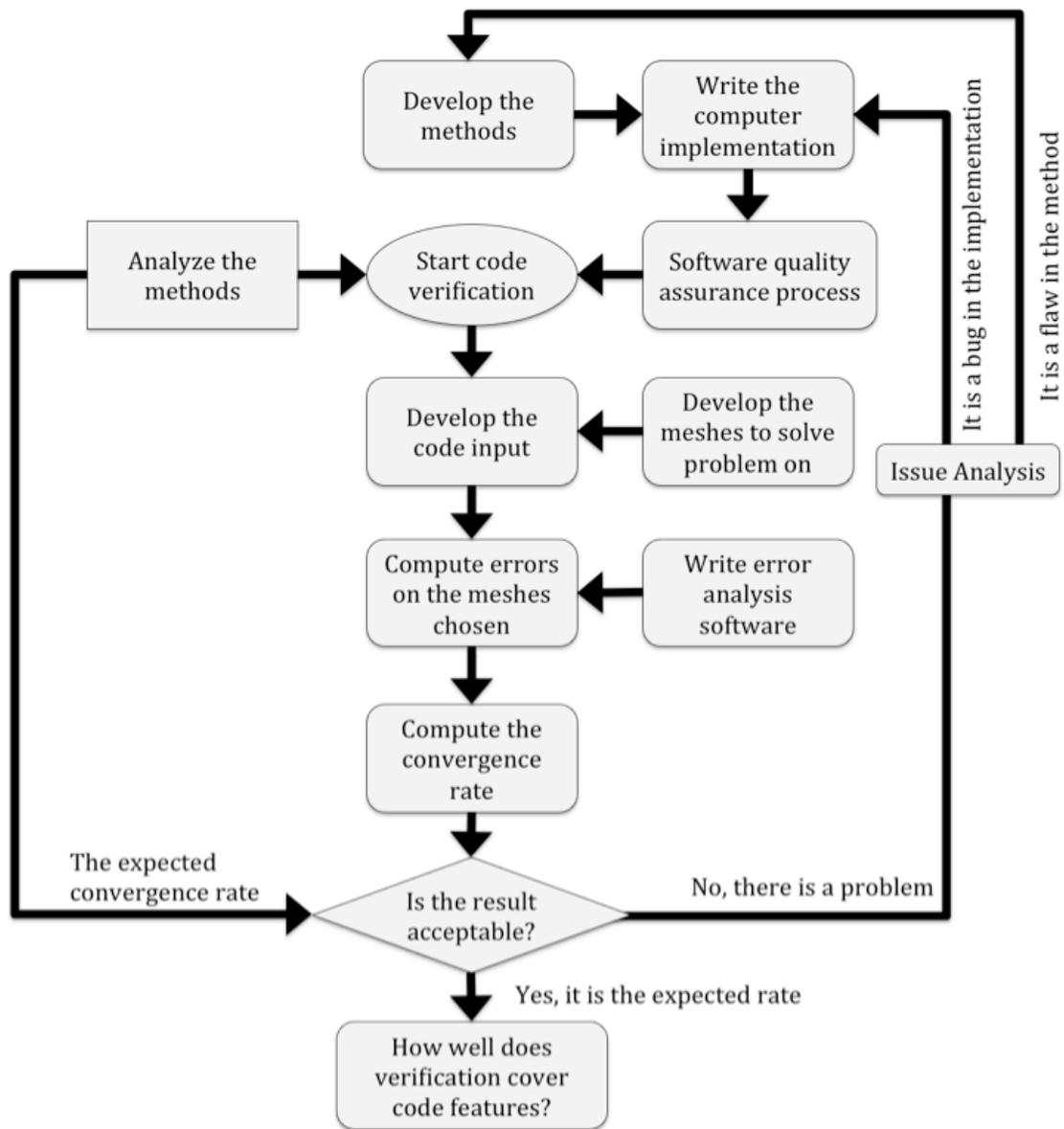


Figure 1(a). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

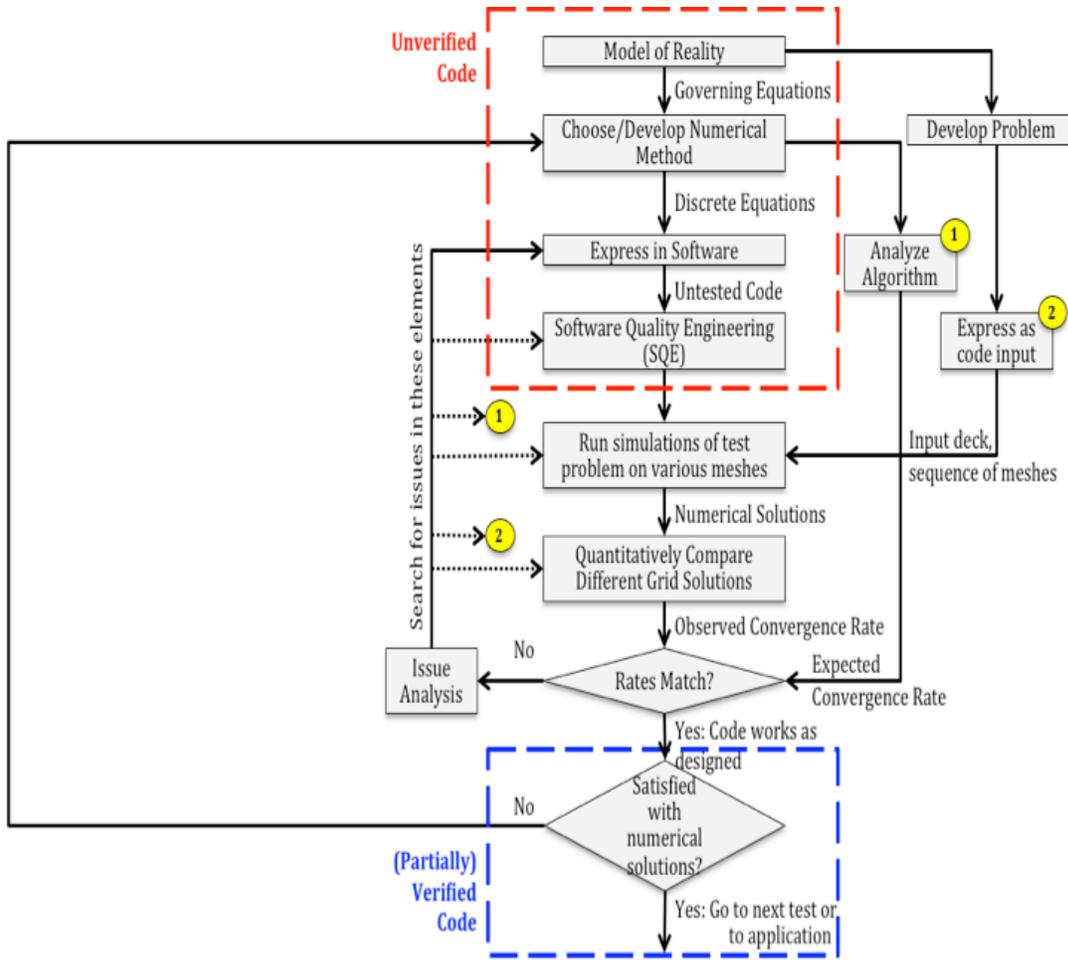


Figure 1(b). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

A New Solution Verification Methodology

The starting point for verification analysis is the definition of a postulated model for the numerical error. The standard model is a power law,

$$A_k = \tilde{A} + Ch_k^p \quad (15)$$

where A_k is the value computed on the k th mesh, \tilde{A} is the (estimate of the) mesh converged solution, C is a proportionality constant, h_k is the mesh length scale (e.g., cell size in 1D), and p is the convergence rate. This ansatz is motivated by conventional analysis (e.g., Richardson extrapolation). One should bear in mind, however, that *any such form is an assumption*. Therefore, one could explore alternative models, but we do not in this work. Often verification (in particular, code verification) focuses on the convergence rate, p as the key result and its congruence with theoretical expectations, p_{theo} . In solution verification, the focus can be

expanded to the overall error term, Ch^p , with specific application to error estimation.

We repeat the important point that the theoretical convergence rate is dependent not only upon the method used for computations, but also upon the nature of the solution itself, the quantity whose convergence is being analyzed, and the metric being considered. For example, a second-order method can be used to compute a result, but the presence of a discontinuity can render the solution only first-order convergent at best [Majda77]. Moreover, under these conditions the first-order result can only be expected for an integrated quantity (e.g., in a hydrocode simulation, the specific internal energy integrated over the domain), and a non-integrated quantity (e.g., the specific internal energy at a point) might be expected to be non-convergent.

This model works well under the proviso that the mesh converged solution is being approached asymptotically in a monotone fashion. Quite often, in practice, this is simply not the case. Under such circumstances the standard model does not work and, instead, we might expect the error to decrease in a power law fashion,

$$|E_k| = Ch_k^p \quad (23)$$

where E_k is the error. Alternatively, one might find that the solution is diverging, which would be characterized by $p < 0$. Such a result is often viewed as a failure, but, in fact, for verification, it is a success: important feedback for a set of calculations has been achieved. We make particular note that the error form in (23) has use in code verification where the errors are computed *a priori* given knowledge of the analytical solution.

In the following development, we will first apply the standard error model in an attempt to achieve a “best-case” result. When this result is available, the error should be defined as the distance between the solution and the best estimate, where the notion of distance will be made precise in the metric used. This is a divergence from the current standard practice for defining “numerical error bars” that are symmetric about the finest mesh used as data. We note that this procedure can only be utilized under the circumstance where the behavior is ideal. Should the data be congruent with the underlying assumptions associated with this model, then this estimate *using the standard error model* will be termed as a “best-estimate” result. If the best estimate is available, then we can also produce an error bound using the second (error) model, which we shall describe. In either case, the error model can be used to bound the error. These estimates provide the foundation by which to define error bars in the currently accepted standard manner, with the error bars associated with the values computed on the finest mesh.

Given a set of metrics computed on a sequence of mesh resolutions, the standard practice is to utilize nonlinear least squares to solve for the parameters in the error model, Eq. (15). Usually this step is completed with little consideration of the

implications of this solution procedure. To help illuminate the significance of this choice, we examine some basic properties of the least squares curve fit. First, the least squares approach is directly associated with normal (Gaussian) statistical assumptions regarding the errors in the parameter values [Bjork]. Specifically, the nonlinear least squares fit is optimal if the errors in the parameters are normally distributed about the optimal values. The least squares formulation has distinct virtues for linear regression problem, because the solution is rendered linear by the minimization of fit residual in the L_2 norm. This property is lost when moving to nonlinear models, such as those we will utilize here. Consequently, we lose little in moving to a more general formalism for the regression.

The field of robust statistics has been developed to reduce the sensitivity of regression procedures to outliers in a given data set. The simplest robust regression approach is to minimize the L_1 norm of the residual. In distinction to the least-squares approach mentioned above, the L_1 regression has a different statistical connection. For L_1 regression, the fit is optimal if the errors are distributed by Laplace's (double-exponential) distribution [Bjork]. The double-exponential distribution is sharply peaked at the mean and has longer tails than the normal distribution. At the other end of spectrum is the minimization of the L_{∞} norm of the residuals (also known as Chebyshev or minimax approximations). Unlike minimization of the L_1 norm, L_{∞} -based regression is *minimally robust* because it can be greatly influenced by outliers; nevertheless, this form of regression is indeed optimal if the errors are distributed uniformly. There are other robust regression procedures, such as least median deviation; we do not utilize such approaches here, but they may prove useful for more general work. More broadly, there is an infinite class of regressions defined by the norm that is chosen for minimization.

For the case we are considering, i.e., a set of metrics computed on a sequence of mesh resolutions, the distribution of errors is unknown and, most likely, does not correspond to any particular analytical probability distribution. There is no reason to favor one distribution over another; that is, that the ensemble of errors should be consistent with some particular distribution is not supported by existing theory or empirical evidence. In particular, there is no reason that the Gaussian distribution associated with standard least-squares regression should be favored, despite its widespread use in applications, including verification.

Finally, we can provide an improvement in the regression via the application of weighting the data. We do have the prior expectation that the results computed on finer grids (i.e., with smaller mesh spacing) are "better." This presumption is essentially a restatement of our belief, ultimately, of convergence under mesh refinement. To reflect this assumption quantitatively, the data can be weighted inversely proportional to the mesh spacing (i.e., by $1/h$)⁴. That is, we judge *a priori* as more "important" the values computed on the finer meshes. This weighting, while

⁴ Of course, this weighting could be modified to be inversely proportional to the mesh spacing to some positive power, i.e., $1/h^q$, where $q > 0$.

usually plausible, is not associated with any particular analytical statistical distribution, but nevertheless provides an alternative, rational approach to data analysis.

Another approach based on prior information would be to utilize the expected (theoretical) convergence rate in the regression. For example, the assumption that the error model for a second-order method is $A_k = \tilde{A} + Ch_k^2$ would produce a linear regression problem. Based on this prior knowledge, the observed convergence rate could reasonably be expected to lie in a certain range, so that a model can be solved using the bounds of this range. Such a line of thought can be extended to the general regression problem by appealing to constrained regression using the above-stated bounds as constraints to the regression problem in the chosen minimization norm.

Robust statistics offer a set of models and regression techniques with which to form estimates of the error and, consequently, of the converged solution. The values of the parameters vary depending on the method used, and the level of variation in the inferred parameters is a direct measure of how the values are distributed. Results may be largely the reflection of outliers in the data set, in which case the parameters themselves may be outliers. The conventional statistics for characterizing a set are the mean and standard deviation, the latter of which is implicitly associated with a Gaussian distribution. These measures are known to be susceptible to the presence of outliers [Huber]; that is, a single outlier can produce a substantial change in these statistics. Of course, the determination of what constitutes an outlier depends upon the statistical assumptions made (often implicitly) in the data analysis.

We contend that such sensitivity is not an appropriate characteristic for a "best estimate" of the result. We make this assertion based on our experience that apparent outliers in the results of numerical calculations of computational science and engineering are far from unknown. To help address this issue, we choose instead the median of our estimates as the measure of central tendency. Unlike the mean, the median of a data set is substantially more robust to outliers [Huber]. The variation in the data can likewise be measured by the median deviation (analogous to the standard deviation), which is the median of the deviation from the median across the ensemble. Our procedure will regress the data using the error model and a number of regression techniques elucidated above, and we will then apply median statistics to identify the best estimate.

Another novel element of our approach it is the ability to examine the results in a manner that does not assume the symmetry of the estimates. The primary analysis is a best estimate of the mesh converged result, \tilde{A} , which should not necessarily be symmetric, but rather potentially have a bias. To accomplish this analysis we first compute the median of \tilde{A} and then divide the list of estimates into two lists of estimates: those less than than the median value and those greater. We subtract the median(\tilde{A}) from each element of these sets and then compute the median deviation for each list. These values are signed, and provide an estimate of the negative or

positive bias in the analysis. On the other hand, the error estimate, $|E|$ is symmetric by construction and should be interpreted as such.

Finally, our approach possesses a number of characteristics of the statistics technique known as bootstrapping. In the bootstrap, small data sets are resampled to provide a better basis for statistical inference. In the case of verification, typically a (very) small number of data points is available. In our analysis, the different regressions provide the set of different statistical views of the data. By using differing regressions and subsets of the data, a bootstrap of a sort is applied. If the data are completely consistent with a certain convergence rate (i.e., the solutions are all in the asymptotic range for the method), then the results of this ensemble will be self-consistent. This will have the effect of producing accurate error estimates with intrinsically small uncertainty. Conversely, if the data are not consistent, then the error estimate will vary significantly, and a large uncertainty will be indicated. Such behavior is ideal for the purposes of solution verification analysis. Our examples will demonstrate this property.

Our New Solution Verification Algorithm

Given this background we will define a sequence of steps to produce our overall error estimates. These estimates will produce a best estimate if the data supports this, and an estimate of the bounds of the error. While the procedure is congruent across the possibilities of under-, exactly- or over-determined regression there are subtle differences that must be acknowledged. At a high level, the overall algorithm is expressed below:

1. Produce an analysis of the numerical method used and the problem solved to establish a theoretical rate of convergence with lower and upper bounds for the convergence rate, p_{lower} and p_{upper} .
2. Screen the data for the basic character (i.e., whether the convergence is monotonically convergent, convergent, or divergent).
3. If the data is monotonically convergent (even weakly, using the end points of the data sequence). Chose a data set starting with the finest mesh values $S_1 = [(h_{N-1}, A_{N-1}), (h_N, A_N)]$, $j=1$.
4. Using the subset of the data, S_1 , produce the following steps.
 - a. Using the data pairs (h_k, A_k) produce a set of constrained regressions using several techniques L_1 , L_2 , $L_{infinity}$, weighted L_2 , p_{theo} L_2 , p_{lower} L_2 , and p_{upper} L_2 .
 - b. Examine the results to see whether the computed estimates of p match either the lower or upper bound. This is a warning sign that probably precludes the completion of a “best estimate” of A .
 - c. Work through the data points from the finest resolution, adding additional (coarser) data points and producing new regression fits for each set of data. This aspect of the procedure is predicated upon the assumption that finer grids produce more accurate results. Thus, for each part of the full data set, one obtains a set of regressions, with the

- results biased toward the finer grids. Return to step 4a until the data is exhausted.
- d. Find the median of the \tilde{A}_{median} estimates, the median deviation, Σ_{median} . The estimate of the mesh converged solution is $\tilde{A}_{median} \pm 3 \Sigma_{median}$. Here, the value $3 \Sigma_{median}$ provides a bound analogous to the 95% confidence interval sought with other solution verification procedures.
 - e. Conduct the asymmetric analysis of the results by separating a sorted list of the \tilde{A} into two equal lists, one with elements less than \tilde{A}_{median} and the other greater than \tilde{A}_{median} . Compute the median of each of these sets and subtract \tilde{A}_{median} , which provides a negative and positive bias, $3 \Sigma^-$ and $3 \Sigma^+$, in \tilde{A}_{median} .
 - f. For all results, one can produce a “GCI-like” result in terms of percentage as $GCI = 3 \Sigma / \tilde{A} * 100$.
(This overall procedure is implemented as a Mathematica script in Appendix A).
5. If the absolute value of the error is monotonically convergent (this includes the monotonically convergent case):
 - a. Compute the absolute difference between the solutions at adjoining meshes, $(h_k, h_{k-1}, |A_k - A_{k-1}|)$ (define $\Delta A_{k,k-1} := |A_k - A_{k-1}|$).
 - b. Produce a set of regressions using the data above $L_1, L_2, L_{infinity}$, weighted $L_2, p_{theo} L_2, p_{lower} L_2$, and $p_{upper} L_2$) for the error model, $C |h_k^p - h_{k-1}^p|$ where the additional constraint that $C > 0$ is used.
 - c. Screen the results of the regression for anomalous behavior in convergence rates. Return to step 5a until the data is exhausted.
 - d. For the best estimate of error, use the median of the error model, $C h_n^p$ regressions evaluated at h_n , where n is the finest grid available. This is the best estimate of the error bar.
 - e. Additionally find the $\max(C h_n^p)$ to produce the bound of the numerical error at the finest grid.
 6. If the errors diverge, compute the rate of divergence and exit.*
 7. If there are unused coarse grid data points $j := j+1$;(if $j < N-1$), $S_j := [(h_{N-j}, A_{N-j}), \dots (h_N, A_N)]$; and return to step 3. This algorithm is given in Appendix A.

* For under-determined (2 grid) cases, this cannot be explicitly determined.

It is worthwhile to make a few comments on the procedure. Expert judgment is added to the process in several key places: the determination of the expected convergence, the screening of the data (with potential rejection of anomalous solutions, and the screening of the regression results). The use of robust statistics can provide some relief from this step, but expert opinion remains a necessary element in this activity. If the data are very well behaved, one produces both a best estimate with a numerical error bar that is not symmetrically placed with regard to the finest solution, and a bounding estimate that is congruent with existing practice. Finally, the procedures eliminate the use of an empirical safety factor, rather instead upon the diversity of estimates and the use of a maximum over those estimates to provide safety in the estimations.

For code verification the basic procedure is the same except no error estimate is needed, as it is explicitly available. We use the same basic error form as before, $E_k = C h_k^p$. There are only two unknowns, the constant and the convergence rate. The procedure we use is otherwise no different than that used for solution verification. The actual script used for the analysis is reproduced in Appendix B.

Example: First-order ODE integration

To clearly demonstrate these techniques we apply both the code and solution verification methodology to a simple problem as an example. To this end, we will solve a classical linear ODE, $dA/dt = -A$, with initial condition $A(0) = 1$, for the solution at time $t = 2$, using the first-order accurate forward Euler method, $A^{n+1} = A^n - h A^n$. The analytical solution is $A(2) = 0.135335$. By utilizing the exact solution, we demonstrate our code verification methodology, and by ignoring the exact solution we demonstrate (and quantify the accuracy of) the solution verification techniques.

Being a simple problem we can compute the results in any number of ways, namely code, Mathematica, Excel, etc.; in this case, we use Excel. We solve the problem at a number of time step sizes as given in Table 1 below. Using the exact solution we can compute errors to enable code verification, and ignoring these results, errors can be estimated.

Time Step Size	Solution at t=2	Exact Error at t=2
0.4	0.0777600000	0.057575283
0.25	0.1001129150	0.035222368
0.2	0.1073741820	0.027961101
0.1	0.1215766550	0.013758628
0.08	0.1243642870	0.010970996
0.04	0.1285121570	0.005449489
0.02	0.1326195560	0.002715727
0.01	0.1339796750	0.001355608
0.008	0.1342511570	0.001084126
0.005	0.1346580430	0.00067724
0.004	0.134793581	0.000541702

Table 1. First order forward Euler solution of an ODE for varying time step sizes.

The code verification results can be computed using the standard techniques with a single linear regression (including a standard deviation computed using Gaussian statistics). In this case, the data in Table 1 gives a convergence rate of 1.03150 ± 0.0029816 . Our new methodology provides very nearly the same result, but, by applying a range of regressions on subsets of the data, uncertainty in the convergence rate is also estimated, with the result: 1.00436 ± 0.003465 , based on 77 different regression fits. Our procedure provides a result that focuses upon the

fine grid results and provides great confidence that the integrator is implemented correctly.

The same sequence of actions can be applied while ignoring the exact solution to produce numerical error estimates. Using the simplest case of Roache’s estimator and standard regression produces the following estimates for the error and convergence rate. In contrast the new procedure provides a more self-contained error estimate with uncertainty together with a convergence rate and uncertainty. Roache’s estimate produces a numerical error of 0.0005178 (GCI $\pm 0.0517851\%$ does not properly bound the error, and neither does the standard deviation of the extrapolated mesh solution ± 0.0000420147), and a rate of 1.0378 (GCI ± 0.00328027). Xing and Stern’s estimator produces a numerical uncertainty of 0.0009263 (CGI $\pm 0.0926339\%$). Our procedure, on the other hand, produces a median convergence rate of 1.0219 ± 0.0154 with a median extrapolated solution of 0.135316 ± 0.000138247 ($\pm 0.102166\%$). Applying the asymmetric analysis to these results reveals more texture. with the bias in the estimated results being large and slightly negative, $3\sum^- = -0.000451116$, $3\sum^+ = +0.0000447998$ (GCI- = -0.333378% , GCI+ = $+0.0331074\%$). Here, we have constrained the convergence rate to lie in the interval $0.5 \leq p \leq 1.5$ in the analysis. The new procedure is clearly more accurate for this well-behaved case, where the older ad hoc approaches are not properly bounding the error in one case. The analysis uses 121 different regression fits to subsets of the data, providing a broad basis for statistical inference utilizing our bootstrap-like approach described above. Figures 1 and 2 provide a snapshot picture of the sample provided by our procedure. On the other hand, our bounding error estimate is 0.0005351 ± 0.0000154 , which is extremely accurate given the exact value for the error given in Table 1.

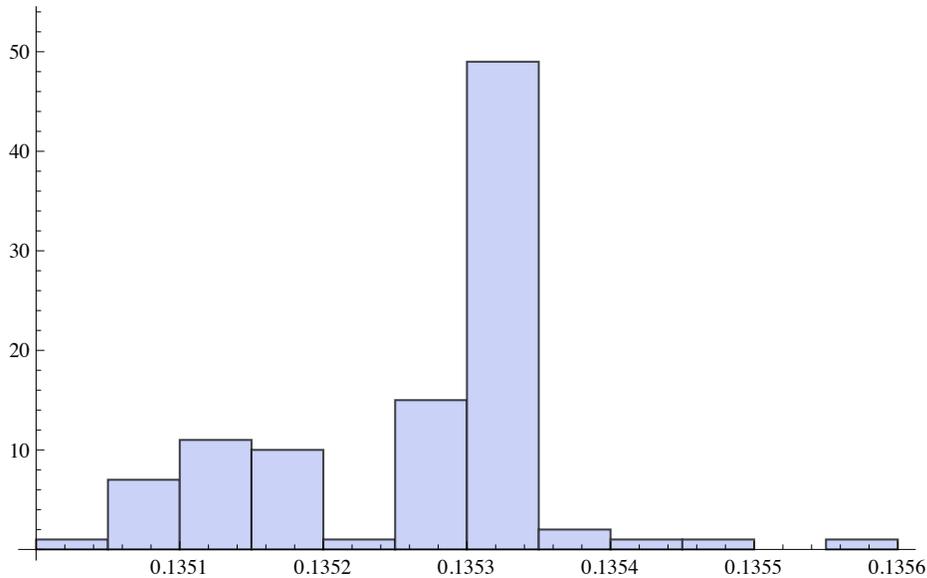


Figure 1. Calculated histogram (i.e., the effective PDF) of the estimated mesh converged result \tilde{A} for the ideal case in our ODE example verification example. Note that the histogram is non-symmetric and biased toward values less than the peak.

The exact solution is contained in the bin associated with the peak of the PDF. This bias is well described by the difference in the computed median deviation values Σ^- and Σ^+ , where the negative deviation is ten times larger than the positive deviation.

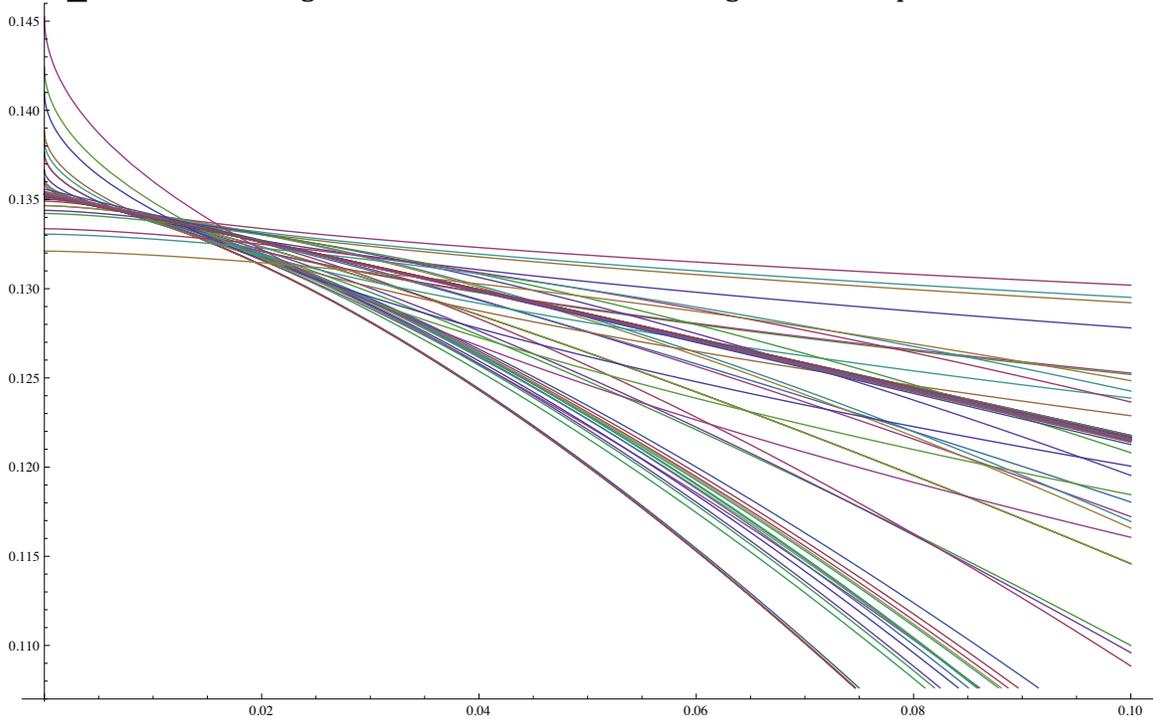


Figure 2. The ensemble of the 99 different solution models derived for the ODE case. Most of the models lie at the upper part of the plot, which strongly influences the estimates and uncertainty. The corresponding histogram (i.e., the effective PDF) is definitely not Gaussian in shape, as shown in Figure 1, the values for which the quantities shown on the ordinate of this plot, i.e., the values for $h(\Delta t)=0$.

This is an almost ideal case that should be a “slam dunk” for almost any reasonable methodology. We can make this problem more realistic—and difficult—by simply analyzing the four coarsest data points given in Table 1. Such a small data set more closely resembles the applied CFD examples in the following sections and situations often encountered in real-world engineering simulations. In the case of code verification, standard regression applied to these four values provides a convergence rate of $1.03851 (\pm 0.00223336)$, while the new procedure gives a similar value of 1.02712 ± 0.00403 . Using the same data in solution verification mode, we produce an estimate of the mesh converged solution $\tilde{A} = 0.13462 \pm 0.001159$ with a convergence rate of $p = 1.06202 \pm 0.009148$. The new estimate captures the exact solution due to the effective bounding procedure, in which the inclusion of the lower and upper bounds on the convergence rate is essential. On the other hand, the bounding error estimate, $U_{num} = 0.001359$, captures the error in the solution nicely. Roache’s estimate is $\tilde{A} = 0.13462 (\pm 0.0000643967)$, with $U_{num} = 0.016304$, and Xing and Stern’s approach gives $\tilde{A} = 0.13462$, with $U_{num} = 0.03411$. Both of these uncertainty estimates are greater than that of the new method by

approximately an order of magnitude, while the estimated converged solutions are nearly identical to several significant figures.

An interesting observation from our example is the significant difference between the standard errors computed from the linear regression in the standard approach augmented by the ad hoc error estimation procedures of either Roache or Xing and Stern. We believe that the differences are most significantly due to the reliance of the standard estimates on a single nonlinear regression as compared with multiple regressions on multiple subsets of the data. For this reason our analysis is more complete (and much more computationally intensive in the analysis phase). The regression standard errors implicitly assume Gaussian statistics and generally under-estimate the actual error, while the numerical uncertainty estimates over-estimate the error. In defense of these estimates, the necessity of over-estimation appears to be a built-in “feature.” The only concern would be the magnitude of the over-estimation of the uncertainty estimates, and its basis in the statistically biased regression that is used to drive the process. We believe that the new approach removes much of the intrinsic bias in regression, replacing it with elements of robust statistics.

Drekar Code Verification

We now apply these techniques to a more complex set of applications specific to the CFD code Drekar. The Drekar team has conducted an extensive set of tests of the basic methods in the code on a set of analytical problems to achieve code verification. These are documented in a separate report [Weber]. We reproduce their results in terms of convergence rates via standard linear regression and augment these findings with an analysis using our methods. The objective is to compare and contrast the approaches and highlight the differences. The overall set of results is shown in Table 2. The raw data is found in the report by Weber [Weber] and broadly confirms the analysis found therein.

We note that the standard convergence rate computed via regression could also produce a convergence rate uncertainty via standard regression uncertainty (with its implicit Gaussian assumption). These results are structurally different from the uncertainty estimated with the multi-regression technique. We have included this uncertainty explicitly, although we note the lack of technical congruence. For Gaussian data, the median deviation is known to be approximately two-thirds the size of a standard deviation [Hoaglin]. The implied errors in the multi-resolution convergence rates are quite different from the standard approach. This might lend credence to a conclusion that the errors are not Gaussian in their distribution.

Problem and Method	Standard Convergence Rate	Multiregression convergence rate and uncertainty
Ethier problem – 1 st order velocity	2.57966 ± (0.0249506)	2.31359 ± 0.200897
Ethier problem – 1 st order pressure	1.44939 ± (0.107549)	2.11666 ± 0.334036
Ethier problem – 2 nd order velocity	2.61285±(0.026739)	2.26233± 0.103723
Ethier problem – 2 nd order pressure	1.41537 ± (0.104975)	2.05999 ± 0.153634
Rayleigh Stokes Velocity 2nd order Space	3.18457 ± (0.0417715)	2.28853 ± 0.09844
Rayleigh Stokes Pressure 2nd order Space	1.85993 ± (0.0178102)	2.14265 ± 0.0351
Rayleigh Stokes Velocity 1st order BDF	0.942179 ± (0.0452008)	1.06296 ± 0.0876819
Rayleigh Stokes Pressure 1st order BDF	0.914665 ± (0.0871837)	0.984202 ± 0.0342315
Rayleigh Stokes Velocity 2nd order BDF	1.81754 ± (0.0681741)	2.22645 ± 0.263049
Rayleigh Stokes Pressure 2nd order BDF	1.41964 ± (0.23145)	2.06041 ± 0.252824
Rayleigh Stokes Velocity 3rd order BDF	3.05312 ± (0.0553349)	3.45639 ± 0.225857
Rayleigh Stokes Pressure 3rd order BDF	2.07431 ± (0.401757)	3.38106 ± 0.237769
Rayleigh Stokes Velocity 4th order BDF	5.07765 ± (0.10723)	4.50769 ± 0.186318
Rayleigh Stokes Pressure 4th order BDF	4.19662 ± (0.526929)	4.47046 ± 0.377723
Rayleigh Stokes Velocity 5th order BDF	5.51404 ± (0.243056)	5.69981 ± 0.180669
Rayleigh Stokes Pressure 5th order BDF	4.58911 ± (1.07952)	4.47046 ± 0.377723

Table 2: Comparison of simple regression-based code verification results (2nd column) and the more complex robust multi-regression approach (3rd column) to the Drekar problems.

The overall character of the convergence rates given by both analyses produces a significant degree of confidence in the numerical methods in Drekar. The convergence rates are consistent with a correct implementation. The new multi-regression technique produces convergence rates that are generally closer to the

design order of accuracy for the methods used to integrate the equations. We assert that our method provides an improved degree of confidence in the solutions.

Drekar Solution Verification

We finish our demonstration with an examination of quantities from the LES simulation of the rod bundle applicable to GTRF. The simulations were computed on grids containing 0.67, 1, 3, 6, and 12 million-mesh cell on both Sandia and Oak Ridge National Lab supercomputers by the Drekar team.

Mesh Elements	Pressure Drop (Pa)	Kinetic Energy (m ² /s ²)	Kinetic Energy Flux (m ⁵ /s ³)
671572	23,4000	0.000674912	0.022413413
1049228	26,7810	0.00079494	0.03032245
2663920	23,8040	0.000821644	0.03014121
5832718	22,0400	0.000822519	0.027973748
12522644	20,7450	0.000823244	0.027696077

Table 3. The raw data obtained from the Drekar Team and used to conduct the analysis of the convergence of the CFD calculations. The calculations used the WALE LES model for all calculation.

One notable feature of the raw data shown in Table 3 is the large change in the results from the coarsest to the second coarsest mesh. For this reason we will examine the convergence including and then excluding the data from the 671572 elements mesh. Often expert judgment typically used in verification analysis would make this choice based on observation of the data. To a very large extent our analysis technique makes this unnecessary. A second preliminary result of our analysis is the inability of standard linear regression to produce sensible results. In every case, the linear regression without explicit constraints produces nonsensical results while the constrained regression does ($0.5 \leq p \leq 2$).

It is notable that our analysis technique provides results that are more consistent between the two data sets although the removal of the coarse grid data does improve the kinetic energy analysis. The same can be said for the mesh converged solution estimate and the convergence rates when comparing the standard solution verification estimates, and our new RMR methodology. Using the new RMR technique the Drekar results are consistently first order accurate (estimated) although the uncertainty in the rate is far too large for comfort. In most case the bounding estimate is larger than the error estimate for the mesh converged estimate (as it should be). The one exception is the pressure drop with the coarse grid removed, which ought to worry anyone examining the data. Nevertheless, the

estimates for the converged mesh solution are well within the uncertainty and quite consistent.

Quantity	Standard Mesh converged estimate	Standard Error estimate	Standard Rate Estimate	New Mesh Converged estimate	New Error Estimate	New rate estimate	New Bounding Estimate
Kinetic Energy X 1000	0.998404	0.530279 (53.1127%)	0.666595	0.845081	0.0620357 (7.3408%)	1.0904 ±1.69147	0.1304874 ±0.118864
Kinetic Energy X 1000 (no coarse)	0.907559	0.255119 (28.1105%)	0.655946	0.828684	0.0138698 (1.67372%)	1.17092 ±1.37471	0.0484734 ±0.0438504
Pressure Drop	23258.4	1636.65 (7.0368%)	0.500142	16496.1	5436.54 (32.9565%)	0.529202 ±0.0876066	11853.3 ±3722.57
Pressure Drop (no coarse)	16469.3	6963.34 (42.2807%)	1.05876	16280.3	7060.3 (43.3671%)	1.08384 ±0.273453	3471.66 ±636.095
Kinetic Energy Flux	0.0308953	0.00226234 (7.32261%)	1.13835	0.026029	0.005745 (22.0715%)	1.31946 ±2.04143	0.00605988 ±0.0017706
Kinetic Energy Flux (no coarse)	0.0218677	0.0183182 (83.7685%)	0.501886	0.0256497	0.00443447 (17.29%)	1.00248 ±1.50744	0.00651942 ±0.0034585

Table 4: Comparison of error estimates and convergence rates for several quantities of interest to LES using Drekar. Comparison of error estimates and convergence rates for several quantities from different data set from Drekar, with the standard results in columns 2–4 and new results in columns 5–7. Note: The kinetic energy has been multiplied by 1000 so allow a more direct comparison with the Hydra results in Table 5.

Overall the Drekar metrics are well behaved although the size of the solution uncertainty and bounding error is worrisome. This certainly provides stern evidence that the mesh is far from converged. The exception is the kinetic energy, which appears to be close to mesh converged, while the pressure drop probably needs at least two halving of the mesh spacing to achieve less than 5% error (this estimated mesh would be on the order of one billion elements using uniform refinement).

Hydra Solution Verification

We have also applied our techniques to previously published results from Hydra [Hydra]. The general assessment of the results was negative in that report and attributed to the quality of the meshes used for the results. Our analysis would seem to support the conclusions drawn by the Hydra team, although we can shed little more light on the reasons for the poor results. The summary of the analysis is provided in Table 3. We analyze three quantities (kinetic energy, pressure drop, and rod surface force) for two of the methods employed by Hydra: the detached

eddy simulation (DES) and the implicit large eddy simulation (ILES). The analysis is completed using the standard GCI methodology and our new approach.

Quantity	Standard Mesh converged estimate	Standard Error estimate	Standard Rate Estimate	New Mesh Converged estimate	New Error Estimate	New rate estimate	New Bounding Estimate
Kinetic Energy (DES)	0.73749	0.03005 (4.074%)	0.500125	0.74188	0.01200 (1.6174%)	0.50042 ±0.00126	0.01560 ±0.02400
Kinetic Energy (ILES)	3.60657	8.50371 (235.78%)	0.00536	0.785852	0.01898 (2.4152%)	1.07051 ±1.71125	0.01163 ±0.02010
Pressure Drop (DES)	9957.35	4882.94 (49.0386%)	0.500131	9979.36	2349.14 (23.54%)	0.5±0.0	955.407 ±1182.28
Pressure Drop (ILES)	192689	554486. (287.76%)	0.003780	10096.2	2686.53 (26.609%)	0.5±0.0	1865.78 ±312.03
Rod Surface Force (DES)	-1.5008	6.74502 (449.42%)	0.003480	0.0001013	0.0003041 (300.%)	1.84908 ± 0.4528	0.006835 ±0.000297
Rod Surface Force (ILES)	1.87773	5.63305 (299.99%)	0.001455	0.09883	0.06793 (68.73%)	0.50002 ±0.000059	0.0006986 ±0.00120

Table 5: Comparison of error estimates and convergence rates for several quantities from different turbulence closure models in Hydra, with the standard results in columns 2–4 and new results in columns 5–7.

Overall, the kinetic energy is the best-behaved quantity, while the surface force is the worst behaved. The kinetic energy is the only quantity where the robustly estimated convergence rate is inside the pre-defined bounds. The other convergence rates are below these bounds for the CGI case or at the bounds for the RMR approach. This result alone casts a great deal of doubt on the quality of the results. Nonetheless, the RMR estimates are reasonable for the pressure drop, although the magnitude of the estimated uncertainty is troubling. More concern is raised by the size of the bounding error estimate, which is smaller than the best-estimate uncertainty. It is a bit more reassuring to observe that adding the uncertainty on the bound does indeed bound the best-estimate uncertainty. Finally, the surface force errors are so large as to completely discount the viability of the calculations for reliably estimating this quantity.

The possibility that the metrics chosen are themselves the problem should not be discounted.

One of the greatest impacts of our analysis is to introduce substantially greater robustness in regression through the use of well-chosen constraints for the regression problem. For example, for positive-definite quantities (e.g., kinetic energy) this property should be maintained in the analysis. The same can be said for the rates of convergence, which are bounded as $0.5 \leq p \leq 2.0$. Furthermore, the

quality of the solutions is poor and the robust regression with constraints provides a set of realizable estimates. Finally, we would comment that the results here probably fall outside the experience of the CFD expertise used in the determination of the safety factors for either the GCI or Xing and Stern's procedure, implicitly calling into question the appropriateness of those approaches for this analysis..

Conclusions

This report has provided both description and basic demonstration of advanced verification techniques applied to GTRF relevant CFD-LES simulations. We reviewed fundamentals of solution verification and the workflow associated with such analyses. The standard approaches to evaluating numerical uncertainty related to discretization error in computed results were discussed. We described a novel method for verification analysis that combines the use of robust statistical methods, asymmetric error estimation, and prior knowledge about the computed results (by weighting fine-mesh results higher and bounding allowed convergence rates). The algorithmic approach to this new method is described in detail. Applying this approach to a simple ODE problem gave results that are comparable to standard approaches of Roache and Xing & Stern. By reducing the number of computed results, i.e., samples, in this case, the new method was shown to have "tighter" uncertainty estimates than the standard techniques.

Analyzing a set of turbulent flow simulations from the CASL-supported Drekar code provides good confidence in the code's result (numerically speaking), and the new solution verification analysis technique (RMR). Further confidence in the analysis technique has been gained by also analyzing the results from Hydra. In both cases the contrast between the standard verification techniques and our RMR approach is rather profound in many cases. Using this approach to analyze turbulent flow properties computed by both codes is an extremely taxing application of verification. The fluctuating time-dependent nature of LES makes the results rough and difficult to analyze.

The results from the two different CFD codes can be contrasted. All-in-all the results are broadly consistent although the convergence rate behavior from Drekar is better, the error estimates from Hydra are smaller with troubling rates of convergence. Because the meshes are the same some degree of conclusion can be drawn; however, the methods in the codes are different as are the subgrid models. This certainly limits the degree to which firmer conclusions may be drawn. The Hydra team has highlighted the poor quality of the meshes generated and used in these studies, and our results do nothing to blunt this criticism. Future studies should hopefully lay these concerns to rest.

We believe that the new techniques have significant advantages as compared with the standard techniques for verification analysis. The first advantage is the more self-contained nature of the methodology. The second advantage is the more

pervasive uncertainty, which in addition utilizes robust statistical techniques to that end.

Acknowledgements

The authors also thank Tim Trucano for many fruitful discussions regarding code verification that have helped inform this work. This effort has benefited from extensive interactions with the entire Drekar team led by John Shadid with Tom Smith and Paula Weber providing calculation results for analysis as well as Erik Cyr and Roger Pawlowski. Additional help was received from Mark Christon and Robert Lowrie of Los Alamos National Laboratory. The development of the verification analysis methodology (RMR) was partially supported by the DOE-ASC V&V program managed by Mary Gonzales, Rich Hills and Walt Witkowski of SNL.

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

- [Bjork] A. Björk, Numerical Methods for Least Squares Problems, SIAM, 1996.
- [Drekar] J. N. Shadid, T. M. Smith, R. P. Pawlowski, E. C. Cyr, “A Summary of SNL Drekar1 Large-scale Parallel CFD Code for use in the Demonstration and Evaluation of Methods for VERA-CFD,” Sandia National Laboratories Report to CASL, 2011.
- [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/
- [Hoaglin] Hoaglin, D. C.; F. Mosteller and J. W. Tukey, Understanding Robust and Exploratory Data Analysis. John Wiley & Sons, 1983.
- [Hydra] M. A. Christon, J. Bakosi, N. Barnett, M. M. Francois, R. B. Lowrie, “Hydra-TH L2 Milestone,” LA-UR-11-07034, CASL Milestone Report.
- [Huber] P. J. Huber, Robust Statistical Procedures, SIAM 1996.
- [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).
- [Obe10] Oberkampf, W. L., Roy, C. J., *Verification and Validation in Scientific Computing*, Cambridge University Press, New York (2010).
- [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).
- [Roy10] Roy, C. J., “Review of Discretization Error Estimators in Scientific Computing,” 48th AIAA Aerospace Sciences Meeting, January 2010, Orlando, FL, AIAA 2010-126 (2010).
- [Shadid] John Shadid, Personal Communication, 2012.
- [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).
- [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).

- [Weber] P. Weber, J. Shadid, E. Cyr, R. Pawlowski, T. Smith, "Initial Drekar: CFD Verification and Validation Study," Sandia National Laboratories, Albuquerque, 2012.
- [Xin10] Xing, T., Stern, F., "Factors of Safety for Richardson Extrapolation," *J. Fluids Engrng* 132, pp. 061403-1– 061403-13 (2010).

Appendix A: Mathematica Script for Code Verification

```
edata = {{0.250000000000, 0.000001808043}, {0.200000000000,
0.000000985836}, {0.175000000000, 0.000001439814}, {0.150000000000,
0.000000444679}, {0.100000000000, 0.000000136837}, {0.075000000000,
0.000000053570}, {0.050000000000, 0.000000009514}, {0.037500000000,
0.000000006435}, {0.025000000000, 0.000000012165}, {0.012500000000,
0.000000014661}, {0.006250000000, 0.000000015001}, {0.003125000000,
0.000000015046}};
```

```
pth = 3.;
emodel = b h^c;
emodel2 = b h^pth;
lcons = b >= 0.0;
m = FindFit[edata, {emodel}, {b, c}, h, Method -> Newton]; emodel /. m
0.0000329286 h^2.07431
h1 =.; h2 =.; mod = {}; rate = {};
```

```
For[k = 1, k < Length[edata], k++, Print[k];
ldata = edata[[Length[edata] - k ;; Length[edata]]];
Print["local data = ", ldata];
```

```
m = Check[FindFit[ldata, {emodel2, lcons}, b, h, Method -> NMinimize],
err];
If[m == err, m = Check[FindFit[ldata, {emodel2, lcons}, b, h], err]];
```

```
If[m == err, Print["Error 0 ", m, " ", ldata], ,
AppendTo[mod, emodel2 /. m]]];
```

```
Print["m0 = ", emodel2 /. m];
b0 = b /. m; c0 = pth;
```

```
m = Check[
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,
NormFunction -> (Norm[#, 2] &)], err];
If[m == err,
m = Check[
FindFit[ldata, {emodel, lcons}, {b, c}, h,
NormFunction -> (Norm[#, 2] &), Method -> NMinimize], err]];
```

```
If[m == err, Print["Error 2 "], ,
AppendTo[mod, emodel /. m];
AppendTo[rate, c /. m]]];
```

```
Print["m2 = ", emodel /. m];
```

```

m = Check[
NonlinearModelFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,
Weights -> (1/# &)], err];
If[m == err,
m = Check[
NonlinearModelFit[ldata, {emodel, lcons}, {b, c}, h, Weights -> (1/# &),
Method -> NMinimize], err]];

If[m == err, Print["Error 2w1"], ,
AppendTo[mod, m["BestFit"]];
AppendTo[rate, c /. m["BestFitParameters"]]];

Print["m2w1 = ", m["BestFit"]];

m = Check[
NonlinearModelFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,
Weights -> (1/#^2 &)], err];
If[m == err,
m = Check[
NonlinearModelFit[ldata, {emodel, lcons}, {b, c}, h,
Weights -> (1/#^2 &), Method -> NMinimize], err]];

If[m == err, Print["Error 2w2"], ,
AppendTo[mod, m["BestFit"]];
AppendTo[rate, c /. m["BestFitParameters"]]];

Print["m2w2 = ", m["BestFit"]];

m = Check[
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,
NormFunction -> (Norm[#, 1024] &)], err];
If[m == err,
m = Check[
FindFit[ldata, {emodel, lcons}, {b, c}, h,
NormFunction -> (Norm[#, 1024] &), Method -> NMinimize], err]];

If[m == err, Print["Error I "], ,
AppendTo[mod, emodel /. m];
AppendTo[rate, c /. m]];

Print["m1024 = ", emodel /. m];

If [k == 0, ,
m = Check[
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,
NormFunction -> (Norm[#, 4] &)], err];

```

```
If[m == err,  
m = Check[  
FindFit[ldata, {emodel, lcons}, {b, c}, h,  
NormFunction -> (Norm[#, 4] &), Method -> NMinimize], err];
```

```
If[m == err, Print["Error 4 "], ,  
AppendTo[mod, emodel /. m];  
AppendTo[rate, c /. m];
```

```
Print["m4 = ", emodel /. m];
```

```
m = Check[  
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,  
NormFunction -> (Norm[#, 16] &)], err];  
If[m == err,  
m = Check[  
FindFit[ldata, {emodel, lcons}, {b, c}, h,  
NormFunction -> (Norm[#, 16] &), Method -> NMinimize], err];
```

```
If[m == err, Print["Error 16 "], ,  
AppendTo[mod, emodel /. m];  
AppendTo[rate, c /. m];
```

```
Print["m16 = ", emodel /. m];
```

```
m = Check[  
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,  
NormFunction -> (Norm[#, 64] &)], err];  
If[m == err,  
m = Check[  
FindFit[ldata, {emodel, lcons}, {b, c}, h,  
NormFunction -> (Norm[#, 64] &), Method -> NMinimize], err];
```

```
If[m == err, Print["Error 64 "], ,  
AppendTo[mod, emodel /. m];  
AppendTo[rate, c /. m];
```

```
Print["m64 = ", emodel /. m];
```

```
m = Check[  
FindFit[ldata, {emodel, lcons}, {{b, b0}, {c, c0}}, h,  
NormFunction -> (Norm[#, 256] &)], err];  
If[m == err,  
m = Check[  
FindFit[ldata, {emodel, lcons}, {b, c}, h,  
NormFunction -> (Norm[#, 256] &), Method -> NMinimize], err];
```

```

If[m == err, Print["Error 256 "], ,
AppendTo[mod, emodel /. m];
AppendTo[rate, c /. m]];

Print["m256 = ", emodel /. m];

];
];

h = 0.02;

mmed = Median[mod]; meddev = MedianDeviation[mod];
Print["median error estimate = ", mmed]; Print["median deviation error \
estimates = ", meddev];
mave = Mean[mod]; msdev = StandardDeviation[mod];
Print["mean error estimate = ", mave]; Print["standard deviation error \
estimates = ", msdev];
rmed = Median[rate]; rdev = MedianDeviation[rate];
Print["median convergence rate = ", rmed]; Print["median deviation \
convergence rates = ", rdev];
rave = Mean[rate]; rsdev = StandardDeviation[rate];
Print["mean convergence rate = ", rave]; Print["standard deviation \
convergence rates = ", rsdev];
Print["length of model vector = ", Length[mod]];
Print["All error estimates ", mod];
Print["rates ", rate]; h =.;
Histogram[mod] Histogram[rate]

```

Appendix B : Mathematica Script for Solution Verification

```
data = {{1.141920333977, 0.762400000000}, {0.984109390844,
0.759500000000}, {0.721372541475, 0.766500000000}, {0.555532758565,
0.772000000000}, {0.430627065018, 0.774800000000}};

edata = {{1.141920333977, 0.984109390844, 0.002900000000}, {0.984109390844,
0.721372541475, 0.007000000000}, {0.721372541475, 0.555532758565,
0.005500000000}, {0.555532758565, 0.430627065018, 0.002800000000}};

pth = 1.; pL = 0.5; pH = 2.0;
model = a + b h^c;
model2 = a + b h^pth;
modelL = a + b h^pL; modelU = a + b h^pH;
cons = {a > 0, pL <= c <= pH};
lcons = {a > 0};
(* GCI estimate *)
m = FindFit[data, {model}, {a, b, c}, h, Method -> Newton];
uf = data[[1, 2]]; uh = a /. m; p = c /. m;
delta = Abs[uf - uh]; Fs = If[Abs[(pth - p)/pth] <= 0.1, 1.25, 3];
Print["model = ", model /. m];
Print["Fs = ", Fs];
Print["Unum = ", Fs delta];
FindFit::cvmit : "Failed to converge to the requested accuracy or precision \
within _100_ iterations. êôðButtonBox["☒",
Appearance->{Automatic, None},
BaseStyle->"Link",
ButtonData->"paclet:ref/message/FindFit/cvmit",
ButtonNote->"FindFit::cvmit"ê"
SequenceForm["model = ", -1.5008407724075412` +
2.256659294898729 h^0.0034866632502479388`]
SequenceForm["Fs = ", 3]
SequenceForm["Unum = ", 6.775922317222624]
(* Stern 1 *)
m = FindFit[data, {model}, {a, b, c}, h, Method -> Newton];
uf = data[[1, 2]]; uh = a /. m; p = c /. m;
h1 = data[[1, 1]]; h2 = data[[2, 1]]; sig = h2/h1;
delta = Abs[uf - uh]; Fs = (sig^p - 1)/(sig^pth - 1);
Print["model = ", model /. m];
Print["Fs = ", Fs]; Print["Unum = ", Fs delta];
FindFit::cvmit : "Failed to converge to the requested accuracy or precision \
within _100_ iterations. êôðButtonBox["☒",
Appearance->{Automatic, None},
BaseStyle->"Link",
ButtonData->"paclet:ref/message/FindFit/cvmit",
ButtonNote->"FindFit::cvmit"ê"
```

```

SequenceForm["model = ", -1.5008407724075412` +
  2.256659294898729 h^0.0034866632502479388`]
SequenceForm["Fs = ", 0.0037514002976742236`]
SequenceForm["Unum = ", 0.00847306566594879]
(* Xing and Stern *)
m = FindFit[data, {model}, {a, b, c}, h, Method -> Newton];
uf = data[[1, 2]]; uh = a /. m; p = c /. m;
theta = p/pth;
delta = Abs[uf - uh]; Fs =
  If[theta < 1, 1.6 theta + 2.45 (1 - theta), 1.6 theta - 14.8 (1 - theta)];
Print["model = ", model /. m];
Print["Fs = ", Fs]; Print["Unum = ", Fs delta];
FindFit::cvmit : "Failed to converge to the requested accuracy or precision \
within _100_ iterations. ãöðButtonBox["☒",
Appearance->{Automatic, None},
BaseStyle->"Link",
ButtonData->"paclet:ref/message/FindFit/cvmit",
ButtonNote->"FindFit::cvmit" ]ê"
SequenceForm["model = ", -1.5008407724075412` +
  2.256659294898729 h^0.0034866632502479388`]
SequenceForm["Fs = ", 2.4470363362372893`]
SequenceForm["Unum = ", 5.526976040588311]
h =.; mod = {}; rate = {}; sol0 = {};

For[k = 0, k < Length[data] - 1, k++, Print[k];
  For[i = 1, i < Length[data] - k, i++, Print[i];
    ldata = data[[i ;; i + 1 + k]];
    Print["local data = ", ldata];

    m = Check[FindFit[ldata, {model2, lcons}, {a, b}, h], err];
    If[m == err,
      m = Check[
        FindFit[ldata, {model2, lcons}, {a, b}, h, Method -> NMinimize], err]];

    If[m == err,
      Print["Error 0 ", m, " ", ldata,
        FindFit[ldata, {model2}, {a, b}, h, Method -> NMinimize]], ,
      AppendTo[mod, model2 /. m];
      AppendTo[sol0, a /. m]];

    Print["m0 = ", model2 /. m];

    m = Check[FindFit[ldata, {modelL, lcons}, {a, b}, h], err];
    If[m == err,
      m = Check[
        FindFit[ldata, {modelL, lcons}, {a, b}, h, Method -> NMinimize], err]];

```

```

If[m == err,
Print["Error L ", m, " ", ldata,
FindFit[ldata, {modelL}, {a, b}, h, Method -> NMinimize]], ,
AppendTo[mod, modelL /. m];
AppendTo[sol0, a /. m]];

Print["mL = ", modelL /. m];

m = Check[FindFit[ldata, {modelU, lcons}, {a, b}, h], err];
If[m == err,
m = Check[
FindFit[ldata, {modelU, lcons}, {a, b}, h, Method -> NMinimize], err]];

If[m == err,
Print["Error U ", m, " ", ldata,
FindFit[ldata, {modelU}, {a, b}, h, Method -> NMinimize]], ,
AppendTo[mod, modelU /. m];
AppendTo[sol0, a /. m]];

Print["mU = ", modelU /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 2] &), Method -> NMinimize], err];
If[m == err,
m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 2] &)], err]];

If[m == err, Print["Error 2c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m2c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 1024] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 1024] &)]];

If[m == err, Print["Error 1c"], ,
AppendTo[mod, model /. m];

```

```

AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["mic = ", model /. m];

If[k == 0, ,
m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 3] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 3] &)]];

If[m == err, Print["Error 4c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m4c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 4] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 4] &)]];

If[m == err, Print["Error 16c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m16c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 5] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 5] &)]];

If[m == err, Print["Error 64c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

```

```

Print["m64c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 6] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 6] &)];

If[m == err, Print["Error 256c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m256c = ", model /. m];

];

]];

mmed = Median[sol0]; meddev = MedianDeviation[sol0];
Print["median solution = ", mmed]; Print["median deviation in solution = ", \
meddev];
rmed = Median[rate]; rdev = MedianDeviation[rate];
Print["median convergence rate = ", rmed];
Print["median deviation in convergence rate = ", rdev];
Print["number of models = ", Length[mod]];
Print["models ", mod];
Histogram[sol0] Histogram[rate]
Show[ListPlot[data], Plot[mod, {h, 0, 0.1}]] Plot[mod, {h, 0, 0.1}]

pL = 0.5; pH = 2.0; pth = 1.0;
model = a + b h^c;
model2 = a + b h^pth;
modelL = a + b h^pL; modelU = a + b h^pH;
cons = {a > 0, pL <= c <= pH};
cons = {a > 0, pL <= c <= pH};
h =.; mod = {}; rate = {}; sol0 = {};

For[k = 1, k < Length[data], k++, Print[k];
ldata = data[[Length[data] - k ;; Length[data]]];
Print["local data = ", ldata];

m = Check[FindFit[ldata, {model2}, {a, b}, h], err];
If[m == err,
m = Check[FindFit[ldata, {model2}, {a, b}, h, Method -> NMinimize], err]];

```

```

If[m == err,
Print["Error 0 ", m, " ", ldata,
FindFit[ldata, {model2}, {a, b}, h, Method -> NMinimize]], ,
AppendTo[mod, model2 /. m];
AppendTo[sol0, a /. m]];

Print["m0 = ", model2 /. m];

m = Check[FindFit[ldata, {modelL}, {a, b}, h], err];
If[m == err,
m = Check[FindFit[ldata, {modelL}, {a, b}, h, Method -> NMinimize], err]];

If[m == err,
Print["Error L ", m, " ", ldata,
FindFit[ldata, {modelL}, {a, b}, h, Method -> NMinimize]], ,
AppendTo[mod, modelL /. m];
AppendTo[sol0, a /. m]];

Print["mL = ", modelL /. m];

m = Check[FindFit[ldata, {modelU}, {a, b}, h], err];
If[m == err,
m = Check[FindFit[ldata, {modelU}, {a, b}, h, Method -> NMinimize], err]];

If[m == err,
Print["Error U ", m, " ", ldata,
FindFit[ldata, {modelU}, {a, b}, h, Method -> NMinimize]], ,
AppendTo[mod, modelU /. m];
AppendTo[sol0, a /. m]];

Print["mU = ", modelU /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 2] &), Method -> NMinimize], err];
If[m == err,
m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 2] &)], err]];

If[m == err, Print["Error 2c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

```

```

Print["m2c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 1024] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 1024] &)]];

If[m == err, Print["Error 1c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["mic = ", model /. m];

If[k == 0, ,
m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 3] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 3] &)]];

If[m == err, Print["Error 4c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m4c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 4] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 4] &)]];

If[m == err, Print["Error 16c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m16c = ", model /. m];

m = Check[

```

```

FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 5] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 5] &)];

If[m == err, Print["Error 64c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m64c = ", model /. m];

m = Check[
FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 6] &), Method -> NMinimize], err];
If[m == err,
m = FindFit[ldata, {model, cons}, {a, b, c}, h,
NormFunction -> (Norm[#, 6] &)];

If[m == err, Print["Error 256c"], ,
AppendTo[mod, model /. m];
AppendTo[sol0, a /. m];
AppendTo[rate, c /. m]];

Print["m256c = ", model /. m];

];

];

mmed = Median[sol0]; meddev = MedianDeviation[sol0];
Print["median solution = ", mmed]; Print["median deviation in solution = ",
3 meddev];
Print["GCI =", 300 meddev/mmed, "%"]
rmed = Median[rate]; rdev = MedianDeviation[rate];
Print["median convergence rate = ", rmed];
Print["median deviation in convergence rate = ", 3 rdev];
Print["GCI convergence rate =", 300 rdev/rmed, "%"]
Print["number of models = ", Length[mod]];
Print["models ", mod];
Histogram[sol0] Histogram[rate]
Show[ListPlot[data], Plot[mod, {h, 0, 0.1}]] Plot[mod, {h, 0, 0.1}]

(* Asymmetric statistics *)
rates = Sort[rate, Less]; sol0s = Sort[sol0, Less];

```

```
Nr = Length[rates]; Nr2 = Round[Nr/2]; Ns = Length[sol0s]; Ns2 = Round[Ns/2];
```

```
If[OddQ[Nr],  
  rmedm = Median[rates[[1 ;; Nr2 + 1]]] - rmed;  
  rmedp = Median[rates[[Nr2 + 1 ;; Nr]]] - rmed,  
  rmedm = Median[rates[[1 ;; Nr2]]] - rmed;  
  rmedp = Median[rates[[Nr2 + 1 ;; Nr]]] - rmed  
];
```

```
Print["Rate Uncertainty Median asymmetric =", 3 rmedm, " +", 3 rmedp];  
Print["Rate Uncertainty Median asymmetric =", 300 rmedm/rmed, "% +",  
  300 rmedp/rmed, "%"];
```

```
If[OddQ[Ns],  
  solmedm = Median[sol0s[[1 ;; Ns2 + 1]]] - mmed;  
  solmedp = Median[sol0s[[Ns2 + 1 ;; Ns]]] - mmed,  
  solmedm = Median[sol0s[[1 ;; Ns2]]] - mmed;  
  solmedp = Median[sol0s[[Ns2 + 1 ;; Ns]]] - mmed  
];
```

```
Print["Solution Uncertainty Median asymmetric =", 3 solmedm, " +",  
  3 solmedp];  
Print["Solution Uncertainty Median asymmetric =", 300 solmedm/mmed, "% +",  
  300 solmedp/mmed, "%"]
```