

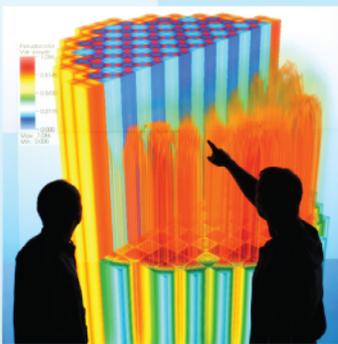
Power uprates
and plant life extension



CASL-U-2013-0069-000



Engineering design
and analysis



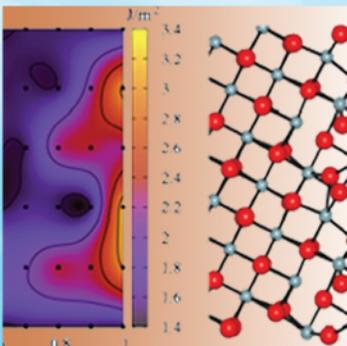
Preliminary Solution Verification of Denovo: Focus on Spatial---Angular Convergence

Science-enabling
high performance
computing



William J. Rider
Sandia National Laboratories
February 28, 2013

Fundamental science



Plant operational data



U.S. DEPARTMENT OF
ENERGY

Nuclear Energy

Preliminary Solution Verification of Denovo: Focus on Spatial-Angular Convergence.

William J. Rider
Sandia National Laboratories
Albuquerque, NM 87185
February 28, 2013

SAND 2013-1421P

Summary

The purpose of this work is to demonstrate solution verification (i.e., numerical error estimation) using advanced techniques on neutronics simulations of interest to CASL. The specific case chosen are simple verification problems computed with the code Denovo developed by ORNL. This effort has been undertaken to understand the basic spatial and angular discretization characteristics. The specific verification analysis will include both standard and recently developed techniques for numerical error estimation, all based on a modified version of the usual power law error description. We will also examine the coupling of spatial-angular discretizations into a single coupling term. One key aspect of the study is the examination of the dependence of the angular error on the discretization. Our proposed error is relatively successful in describing the error. We also exercise the error analysis methodology under these stressing circumstances. 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.

This work builds upon the basic theory, solution verification analysis and workflow described earlier by the authors in [Rid10,Rid11,Rid12].

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.

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). In this case we have to include the angular variable associated with the use of a discrete quadrature used in the discrete ordinates method. This aspect of the work renders our effort far more exploratory in nature. Our finding will be somewhat unique with regard to the published literature. The solution approach used on those discrete equations *and the nature of the solution itself* determine the expected behavior of the error. 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.

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) and in the specific case of our study here the order of the numerical quadrature defined by the number of quadrature points evaluated. 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.
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

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.

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.

Basic Verification Analysis Theory

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 $h_{k-1} / h_k \stackrel{\circ}{=} s > 1$. In this case, the error ansatz implies:

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

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,

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

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

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.

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].

Augmentation of Verification Theory for Neutronics

In order to apply verification to neutronics we must treat the angular variable. The angular quadrature is defined by the order of the quadrature (here our attention is primarily directed toward the level symmetric set used as the default in Denovo). This method is known as the “Sn” method where “n” is the order of the quadrature, and the number of quadrature points is given by $n(n+2)/2$, which scales as n^2 . The angular space is two dimensional thus the analog to h in space is $1/n$ in angle. This is similar to the ansatz made by Jarrell in his thesis [Jar10]. We then take the angular error to be

$$A_k = A + C_n / n^{p_n}, \quad (15)$$

and the coupled angular-space equation with discretization error to be

$$A_k = A + C_h h^{p_h} + C_n / n^{p_n}, \quad (16)$$

or

$$A_k = A + C_h h^{p_h} + C_n / n^{p_n} + C_{hn} \left(\frac{h}{n} \right)^{p_{hn}}. \quad (17)$$

These are the specific forms that we examine from the set of calculations conducted with Denovo. We have tested (16) in comparison to (17) and found that the simpler form of the error model performs better as measured by the requisite uncertainty. Some general conclusions regarding the propriety of these forms will be one of the preliminary conclusions from this study.

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 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_{infinity}), 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

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

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.

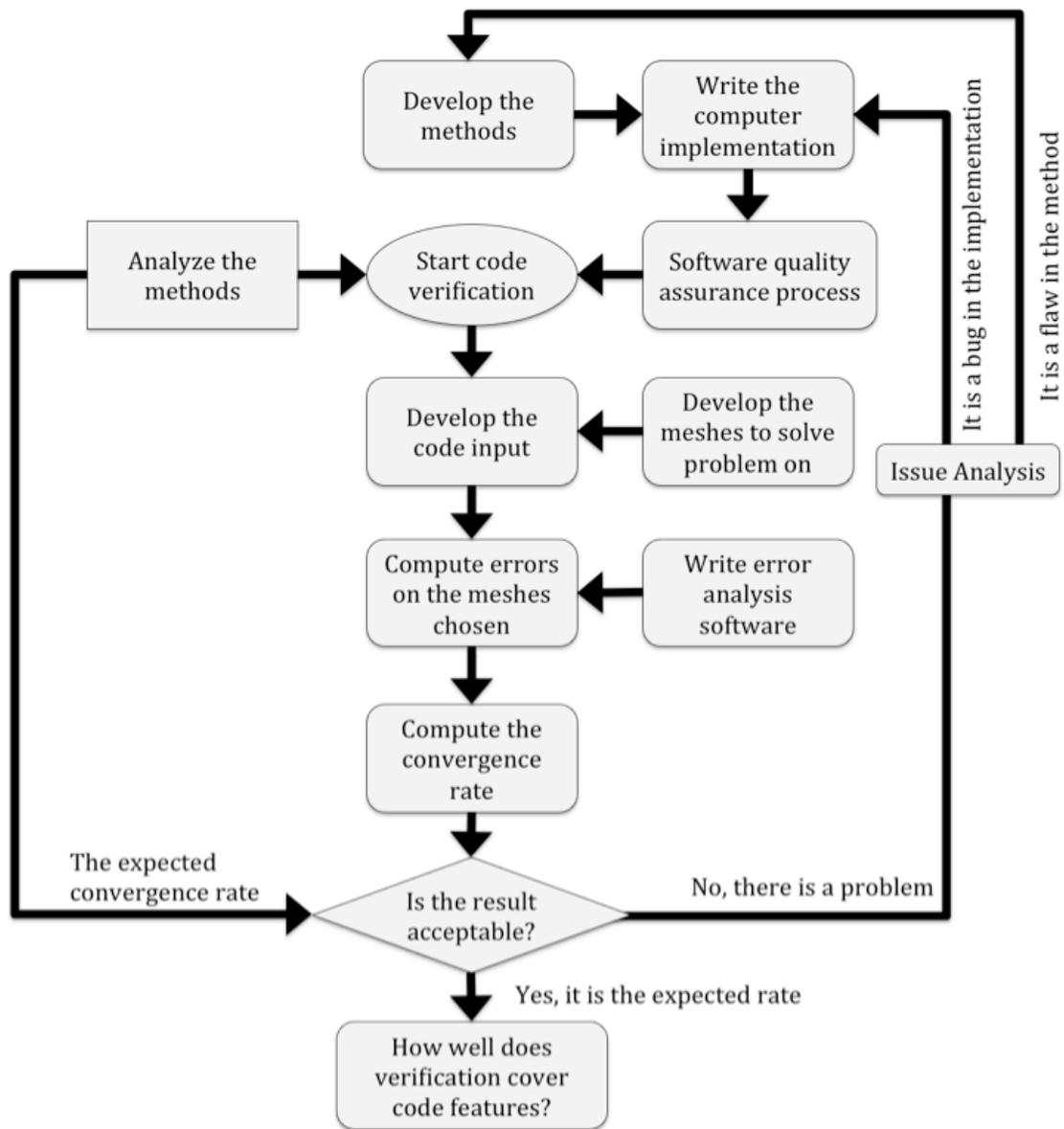


Figure 1(a). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

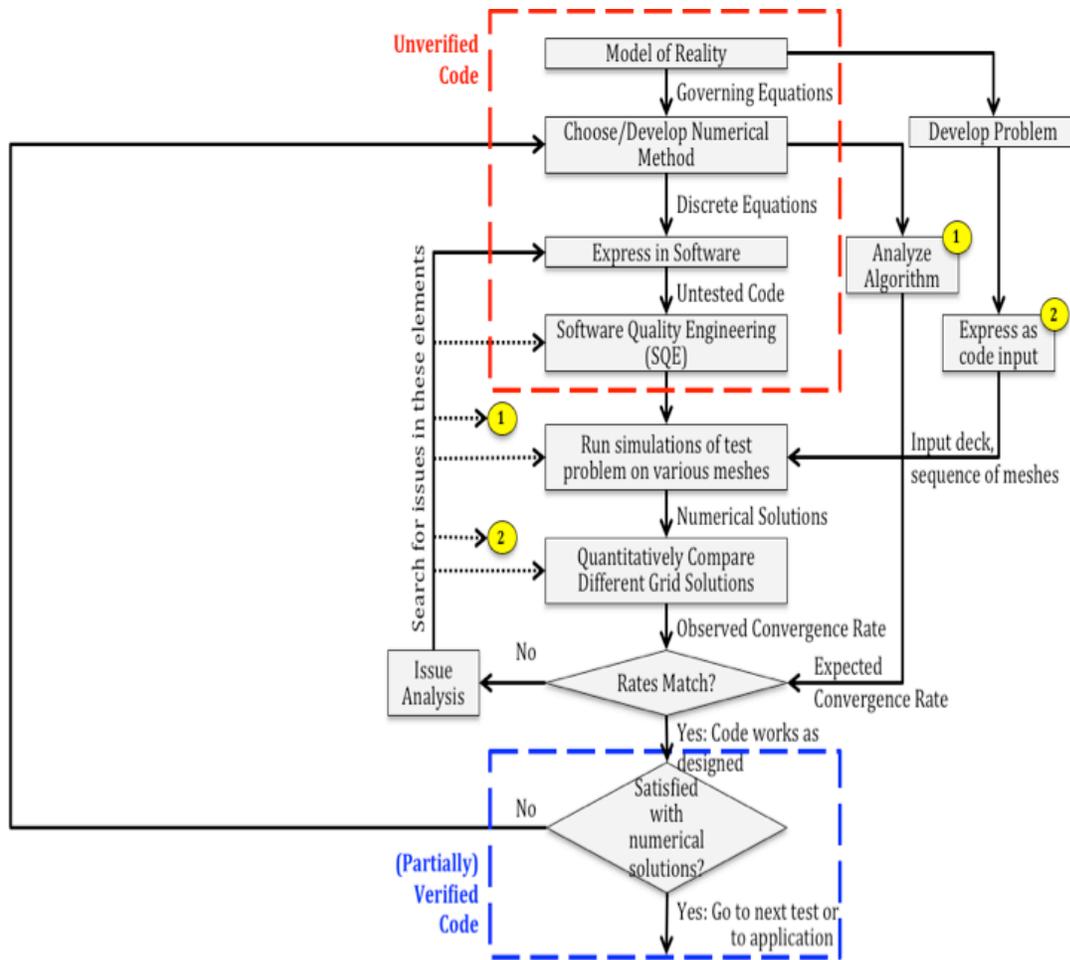


Figure 1(b). The flowchart version of the list of activities is shown for code verification, which can be interpreted as an expansion of the simple expression of this activity.

A Solution Verification Methodology Based On Optimization

The starting point for verification analysis is the definition of a postulated model for the numerical error. The standard model is a power law (which we continue to study albeit in modified form),

$$A_k = \tilde{A} + Ch_k^p \quad (18)$$

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*. Here we also use the set of angular errors forms discussed above (equations 15, 16, 17). 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. We have found that the neutronics literature does not contain significant results regarding the expected rate of convergence for discrete ordinates methods as a function of the angular order of the quadrature.

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,

$$|E_k| = Ch_k^p \quad (19)$$

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 (19) 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. (18). 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 and then implemented via optimization in the work reported here.

We have replaced the regression with the equivalent, but more flexible practice of optimization. This allows us to pose the minimization functional more generally as well as access more robust solution techniques than the general nonlinear regression methods allow. We are not limited by the specific implementation in the regression package in software. In particular robust regression, Tikhonov regularized least squares or LASSO regression can all be easily defined along with a reliable solution to the L_1 , $L_1/2$ and L_{∞} regression problems.

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. In this work we combine the weights for space and angle for a weight, (n/h) . That is, we judge *a priori* as more “important” the values computed on the finer meshes. This weighting, while 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.

Our first effort focused upon the implementation within regression software, but upon the examination of results we found that direct optimization produced better results within our chosen software tool, Mathematica™. Overall, the solution methods used for the minimization are more flexible, reliable and robust. Moreover, significant additional flexibility was gained in defining the functions being used for regression. The allowed a number of robust regression procedures to be utilized in the work including L1, and infinity norms as well as regularized functionals such as Tikhonov and LASSO regularizations. The actual procedures utilized in the results are shown in script form in an appendix to this report.

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

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 $q > 0$. In this effort we also introduce the weighting with respect to angle, i.e., a weight of n/h , or n^q/h^q .

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 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 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 based on Optimization Methods

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 optimization 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} . For the more complex error ansatz with two discrete variables, bounds are entered for all variables. In addition, the basic nature of the solution can be encoded as a constraint (such as positive definiteness, or more specific upper or lower bounds).
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 $S_1 = [(h_{N-1}, n_{N-1}, A_{N-1}), (h_N, n_N, A_N)], j=1$.
4. Using the subset of the data, S_1 , produce the following steps.
 - a. Using the data pairs (h_k, n_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 $p_{upper} L_2$. In addition, L_4, L_8 , Tikhonov, LASSO, and weighted variants of each using (n_k/h_k) .
 - 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 . These estimates will be provided for spatial, or angular errors alone, or their coupled error.
 - 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, \sum_{median} . The estimate of the mesh converged solution is $\tilde{A}_{median} \pm 3 \sum_{median}$. Here, the value $3 \sum_{median}$ 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, $3 \sum^-$ and $3 \sum^+$, in \tilde{A}_{median} .
 - f. For all results, one can produce a “GCI-like” result in terms of percentage as $GCI = 3 \sum / \tilde{A} * 100$.
(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) *Note: this form of analysis is excluded from this study due to the focus on angular discretization:*
 - a. Compute the absolute difference between the solutions at adjoining meshes, $(h_k, n_k, h_{k-1}, |A_k - A_{k-1}|)$ (define $\Delta A_{k,k-1} := |A_k - A_{k-1}|$).
 - b. Produce a set of regressions using the data above $L_1, L_2, L_{\text{infinity}}$, weighted $L_2, p_{\text{theo}} L_2, p_{\text{lower}} L_2$, and $p_{\text{upper}} L_2$) 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. We further note that the error examination has been excluded from this study for brevity.

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, $3\Sigma^- = -0.000451116$, $3\Sigma^+ = +0.0000447998$ ($GCI^- = -0.333378\%$, $GCI^+ = +0.0331074\%$). 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.

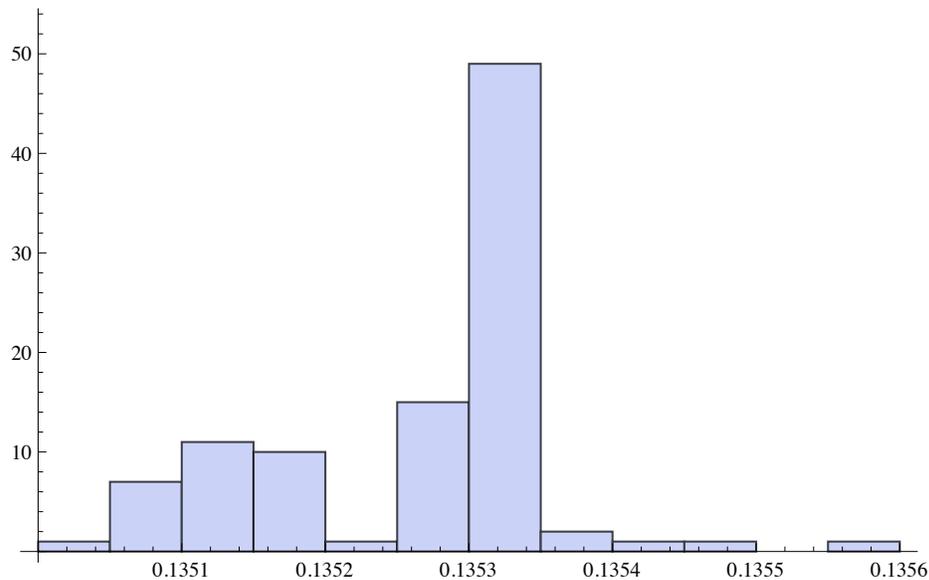


Figure 1. Calculated histogram (i.e., the effective PDF) of the estimated mesh converged result \tilde{A} for the ideal case in our ODE example verification example. Note that the histogram is non-symmetric and biased toward values less than the peak. The exact solution is contained in the bin associated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

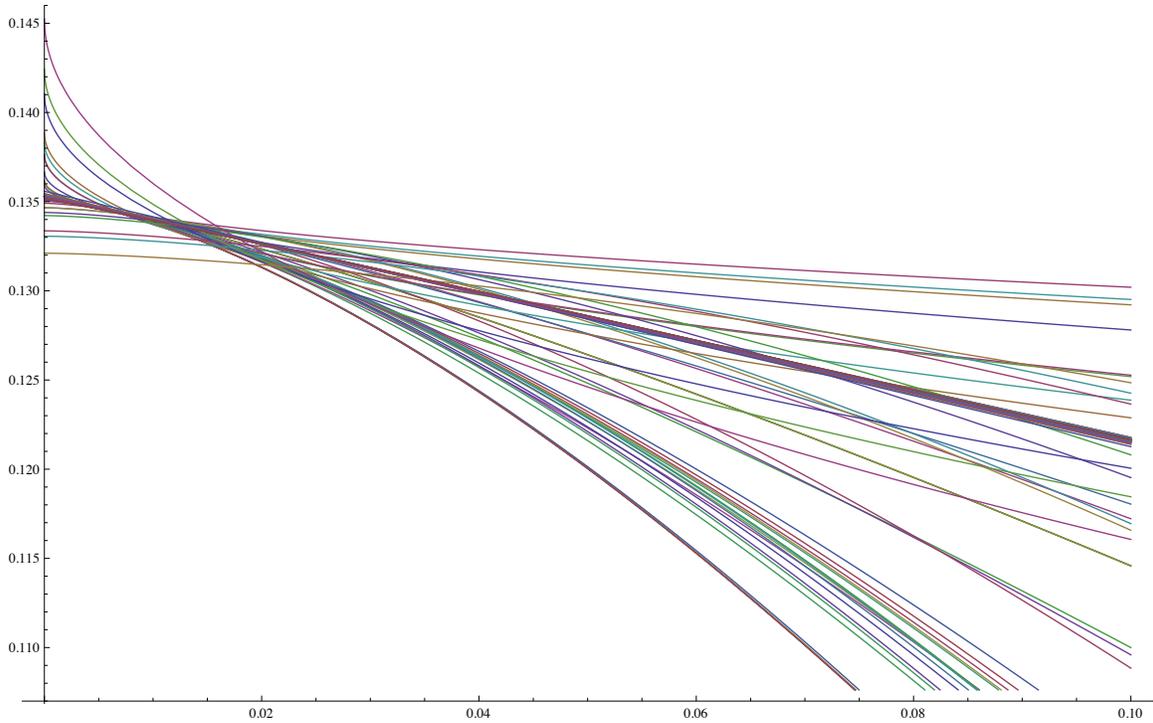


Figure 2. The ensemble of the 99 different solution models derived for the ODE case. Most of the models lie at the upper part of the plot, which strongly influences the estimates and uncertainty. The corresponding histogram (i.e., the effective PDF) is definitely not Gaussian in shape, as shown in Figure 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.

Results For The Neutronics Code, Denovo

Denovo [Eva12] is a discrete ordinates radiation transport method used to solve the Boltzmann transport equation (time-independent),

$$\begin{aligned} \Omega \cdot \nabla \psi(x, \Omega, E) + \sigma(x, E) \psi(x, \Omega, E) = \\ \int dE' \int_{4\pi} d\Omega' \sigma_s(x, \Omega' \cdot \Omega, E' \rightarrow E) \psi(x, \Omega', E') \\ \frac{\chi(E)}{4\pi k} \int dE' \int_{4\pi} d\Omega' \nu \sigma_f(x, \Omega' \cdot \Omega, E' \rightarrow E) \psi(x, \Omega', E') \end{aligned} \quad (19)$$

describing the behavior of neutron populations in space. The methodology used by Denovo is referred to as deterministic transport to distinguish it from statistical methods based on the Monte Carlo method [Met49] although Denovo does contain the capacity to compute Monte Carlo solutions. The code contains the capability to use several different spatial approximations to the streaming term in equation (19) as well as different quadrature formula for the integral term. We will not examine the code's capacity to solve eigenvalue (criticality) problems in this study although future work could include this.

The spatial approximation to the first-order streaming term includes the step characteristic method (i.e., donor cell or upwind in the parlance of hydrodynamics methods), and linear discontinuous (i.e., discontinuous Galerkin), trilinear discontinuous and the classical diamond differencing with negative flux fix-up. These methods each carry certain advantages and have been examined extensively in the literature [Lar82, LM84]. The spatial grid can be 1-, 2-, or 3-dimensional and non-uniform in nature.

The angular discretization includes the classical level symmetric method, which is its default. The order of the quadrature can take even values from 2 to 24. Other available quadratures are Gauss-Legendre, Quadruple Range, Galerkin, and a linear

discontinuous finite element (LDFE) approach. We will examine these quadratures with respect to mesh convergence in the angular variable although expectations for the rate of convergence are sparse in the literature. Denovo employs the Koch-Baker-Alcouffe sweep method for solution along with accelerated source iteration. The code can compute in parallel. Denovo is capable of using the full spectrum of modern computing hardware from laptops to the largest supercomputers. For this study relatively small computers were used (a Macintosh desktop with 2-6 Core 3GHz processors, and up to 24 way parallel computations using MPI on threads). The parallel processing is necessary due to the high-dimensional nature of the transport equation (it is six dimensional in the time independent case). For the study provided in this report, we have solved over one-and-a-half billion unknowns on our most refined grid despite the relative simplicity of the problems chosen.

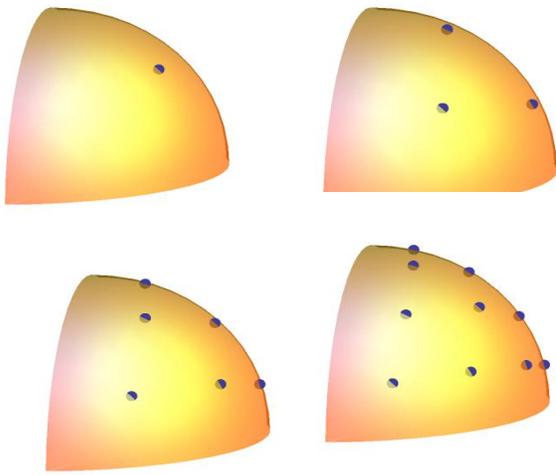


Figure 3. Example of the placement of quadrature points for the level symmetric quadrature for the S2, S4, S6 and S8. A single octant of the solid sphere is displayed with the location of the integration points.

Such flexibility in approximation is not available for the discretization in energy. This reflects the usually complex structure associated with cross section information as a function of neutron energy that does not lend itself to a well-controlled approximation theory. Nonetheless, the behavior of such codes with regard to energy would be a useful study for the future. We note in passing that photon (gamma) transport has been studied with respect to convergence in energy discretization due to its intrinsically better behaved character and second-order convergence has been demonstrated [She10].

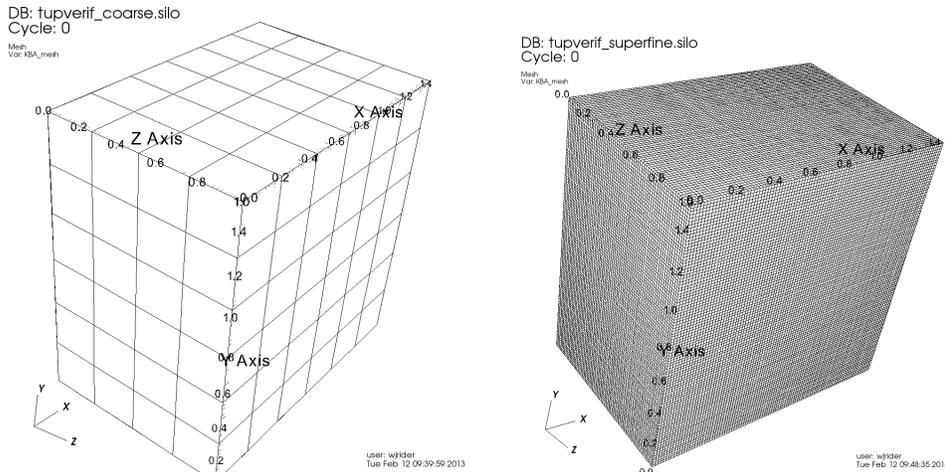


Figure 4. Coarse and fine grids used in our study. These are the coarsest and finest spatial grids utilized for the upscatter problem. The angular grid varied from S2 to S24 level symmetric quadratures.

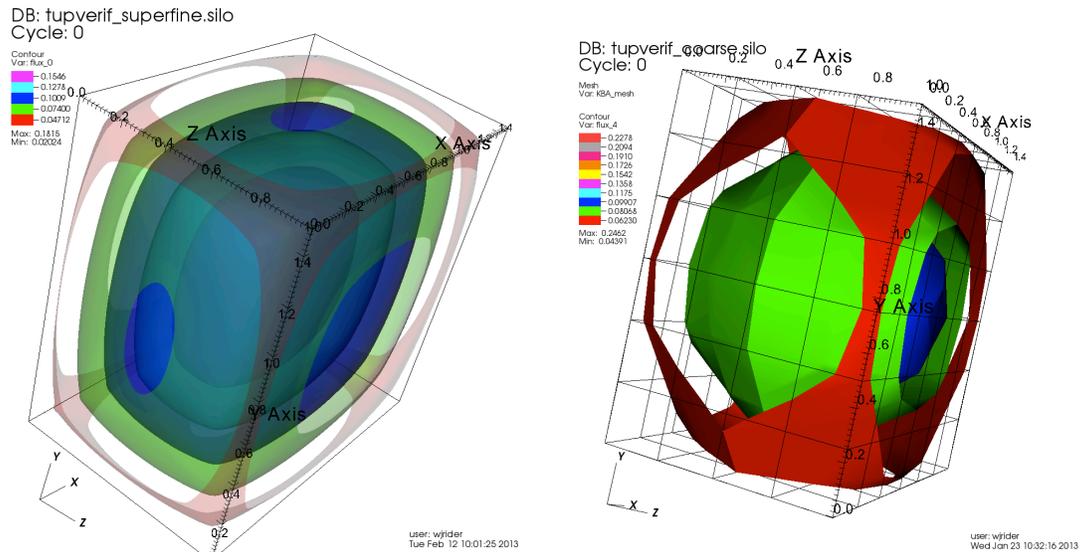


Figure 5. A contour plot of the integrated scalar flux for the energy group 5 on the finest and coarsest meshes for 5 different values of the flux. It is a smooth isotropic problem as shown, which should give relatively well-behaved results in the verification study. Sn6 is used for the angular integration.

We have examined two relatively simple problems used in the Denovo testing suite. Each problem is defined on a coarse grid and low-order angular quadrature, which are systematically refined with a specified number of energy groups. The energy group structure is not modified although as noted this would be an excellent future study. The problems used are focused upon separate physical effects, either upscatter (scattering where the neutron moves to a higher energy), or downscatter (where the neutron moves to lower energy). The upscatter problem uses five energy groups, and the downscatter problem uses two. Each problem has an

isotropic source and amenable to an analytical solution although this solution was not made available to us, it may be applicable to classical code verification.

The basic results for the upscatter problem are summarized in Table 2 for the different spatial and angular “meshes” used. These results will then be analyzed with regard to the first separate spatial and angular convergence, and then again for the coupled error. The results in Table 2 indicate to the casual observer that the solution is quite well “mesh-resolved” in both space and angle for the finest “meshes” chosen. The analysis using the new verification methodology is provided in Table 3. Given the median statistics used in our methodology all of the results for converged fluxes and convergence rates are consistent well within the 95% stated confidence interval. It has been noted that this problem is not useful for measuring spatial convergence because the approximation becomes semi-analytical. This accounts for high rate of convergence for the spatial differencing and the lack of distinction between the step and LD methods. Our proposed error form for the angular quadrature appears to hold up to the examination nominally confirmed approximately a second-order convergence for the level-symmetric approach.

h (Δx)	n	LD int. flux	Step int. flux
0.25000	2	0.16124	0.14930
0.12500	2	0.11277	0.11017
0.06250	2	0.10791	0.10684
0.03125	2	0.10754	0.10703
0.01563	2	0.10752	
0.25000	4	0.16398	0.15387
0.12500	4	0.11685	0.11423
0.06250	4	0.11188	0.11069
0.03125	4	0.11148	0.11087
0.01563	4	0.11145	
0.25000	6	0.16477	0.15555
0.12500	6	0.11762	0.11541
0.06250	6	0.11268	0.11170
0.03125	6	0.11227	0.11177
0.01563	6	0.11224	
0.25000	8	0.16517	0.15599
0.12500	8	0.11800	0.11581
0.06250	8	0.11307	0.11209
0.03125	8	0.11266	0.11216
0.01563	8	0.11262	
0.25000	12	0.16549	0.15647
0.12500	12	0.11829	0.11619
0.06250	12	0.11336	0.11243
0.03125	12	0.11295	0.11247

0.01563	12	0.11292	
0.25000	16	0.16563	0.15670
0.12500	16	0.11841	0.11636
0.06250	16	0.11348	0.11258
0.03125	16	0.11307	0.11261
0.01563	16	0.11304	
0.25000	20	0.16570	
0.12500	20	0.11848	
0.06250	20	0.11355	
0.03125	20	0.11314	
0.01563	20	0.11311	
0.25000	24	0.16575	0.15689
0.12500	24	0.11852	0.11651
0.06250	24	0.11359	0.11271
0.03125	24	0.11318	0.11273
0.01563	24	0.11315	

Table 2. The raw results for the integrated scalar flux from the upscatter problem are shown for space-angle discretizations. The verification analysis is shown in Table 3 using this data. Note: we keep much greater accuracy in the data used in the actual verification analysis.

Case	Converged Flux	\pm flux	Spatial Rate	\pm rate	Angular rate	\pm rate
Step space-angle	0.112767	0.000212	3.57327	0.13475	1.80497	0.85473
LD space	0.113143	0.000126	3.66037	1.05584		
LD angle	0.113615	0.000081			2.23472	0.70415
LD space-angle	0.113222	0.000141	3.28949	0.04058	1.62063	0.54076

Table 3: Convergence results for the downscatter problem, the step method is examined with coupled analysis, while the LD (linear discontinuous) is examined for space and angle alone as well as coupled space-angle for the convergence. We note that the smallest uncertainties are achieved with the coupled method (this is where the different error models (16) and (17) are tested). This problem is noted as being inapplicable to analyzing the spatial operators because of the nature of the transport rendering the step method to be uncharacteristically accurate.

Figures 6, 7 and 8 provide more texture to the results showing the results of the models solved given the above data. The effectiveness of the median statistics

removes irregular results that can result from the roughness of the data or unreliability of the optimization solution. The other primary sources of difficulty are the results obtained with the coarsest angular quadrature, Sn2, which produces results and convergence uncharacteristic of the refined quadratures. We note, again, that the optimization or regression can sometimes spectacularly fail, and these results must be discarded to avoid corrupting results. The box-and-whisker plots are the best depiction of the relative persistence of outliers in the approach.

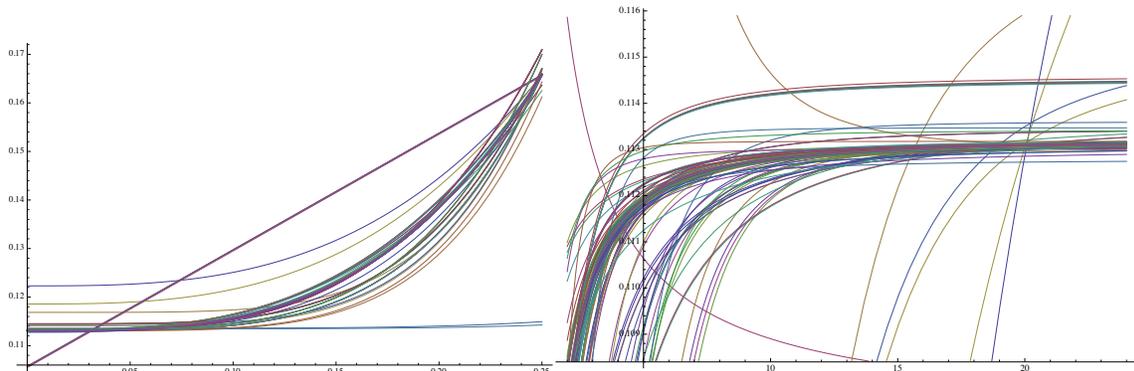


Figure 6. Ensemble of models for space and angular error for the coupled space-angle convergence analysis. Both sets of discretizations appear to be very well behaved despite a handful of outliers (which are removed by the action of the median statistics).

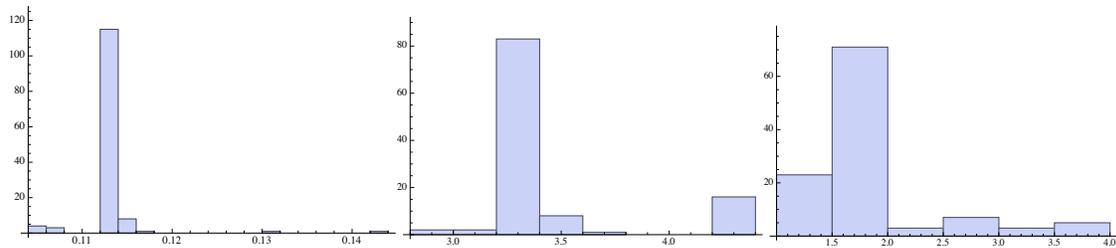


Figure 7. PDFs of the converged solution, spatial convergence rate and angular convergence rate.

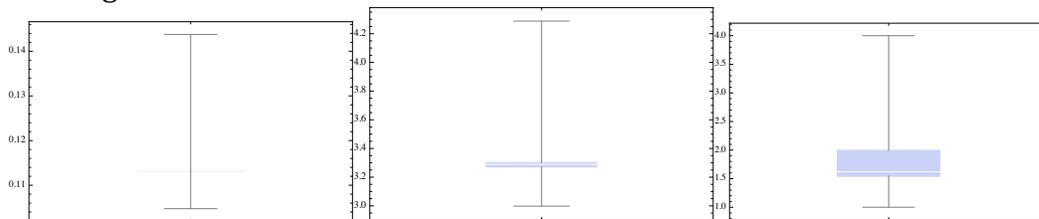


Figure 8. The “box-and-whisker” plots for the same three measures. For the mesh converged result the presence of outliers is undeniable and necessitates robust statistics be used.

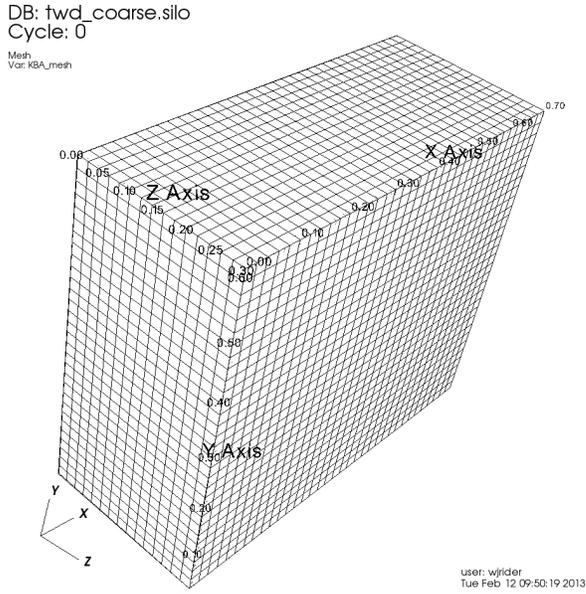


Figure 9. An example of the medium density grid used for the downscatter problem.

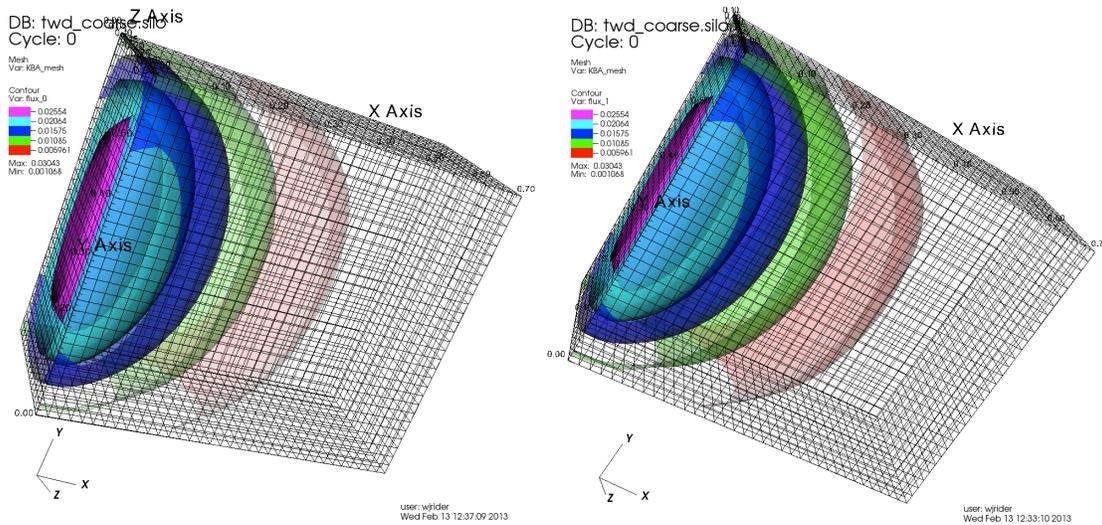


Figure 10. Downscatter results on a medium grid using Sn12 for the integrated scalar flux in the two energy groups used.

The basic data used to examine convergence is given in Tables 4, 6, and 8. This problem is capable of distinguishing between differing spatial differencing approaches. We also examine several of Denovo's available angular quadratures including the method developed by Jarrell (LDFE, [Jar10]) for which convergence results exist. Jarrell found roughly 4th order convergence on simple problems, a property consistent with our results. The analysis is summarized in Tables 5, 7, and 9. We find a distinct difference between the step characteristic and LD schemes with results showing some degree of super-convergence, but the 95% confidence intervals contain the theoretically expected 1st and 2nd order accuracy.

h (Δx)	n	Step flux p0	Step flux p1	LD flux p0	LD flux p1	DD Flux p0	DD flux p1
0.1000	12	0.089181	0.010295	0.088047	0.009856	0.087919	0.009867
0.0500	12	0.088635	0.010086	0.088015	0.009862	0.087855	0.009859
0.0333	12	0.088442	0.010014	0.088010	0.009862	0.087949	0.009866
0.0250	12	0.088341	0.009977	0.088007	0.009862	0.087983	0.009867
0.0200	12	0.088278	0.009955	0.088007	0.009862	0.087982	0.009865
0.0167	12	0.088236	0.009940	0.088007	0.009862	0.087994	0.009865
0.0125	12	0.088181	0.009921	0.088007	0.009862	0.087998	0.009864
0.0250	2			0.076944	0.007179		
0.0250	4			0.086811	0.009460		
0.0250	6			0.087747	0.009723		
0.0250	8			0.087963	0.009809		
0.0250	10			0.087965	0.009837		
0.0250	12			0.088007	0.009862		
0.0250	14			0.088058	0.009880		
0.0250	16			0.088130	0.009898		
0.0250	18			0.088195	0.009911		
0.0250	20			0.088240	0.009921		
0.0250	22			0.088262	0.009927		
0.0250	24			0.088281	0.009933		

Table 4. The raw results for the integrated scalar flux from the downscatter problem are shown for space-angle discretizations. The verification analysis is shown in Table 5 using this data. Note: we keep much greater accuracy in the data used in the actual verification analysis.

One key point to examine from Table 5 are the differences between the classical CGI approach and our new approach. First, the GCI approach provides no uncertainty on the convergence rates despite these rates being undeniably uncertain. In some cases, the solution uncertainty is radically different calling into question whether the CFD experience used is valid for neutronics, and the unreliability of a single regression analysis for ascertaining the converged properties of the scheme and/or calculation. In most cases, the two approaches share consistency in that the CGI solution is within the 95% confidence interval given.

Case	Flux p0	Flux p1	Rate p0 space	Rate p1 space	Rate p0 angle	Rate p1 angle
Step	0.0880849±0.000183	0.00990086±0.000061	1.52187±1.28362	1.76222±1.8659		
Step GCI	0.0879945±0.000559	0.00985842±0.000078	0.889216	0.938243		
LD	0.0880065±0.00000077	0.00986165±0.00000008	3.2795±1.92338	4.21889±1.0533		
LD GCI	0.0880066±0.0000011	0.00986191±0.0000008	2.17422	5.09874		
DD	0.0880019±0.0000062	0.00986435±0.0000029	2.48781±1.8980	3.27643±1.3603		
DD GCI	0.0879984±0.0113321	0.00986454±0.0000004	0.0148108	3.79242		
LD+Sn	0.0882982±0.0001678	0.00993549±0.0000446			2.5473±1.10376	2.6598±0.6907
LD+Sn GCI	0.0881712±0.0003307	0.00991214±0.0000619			3.0029	2.5183

Table 5. Convergence results for separate space and angle for the downscatter problem are shown.

Tables 6 and 7 show the examination of quadrature options in Denovo. Clearly, the standard LS quadrature is less accurate than the other options. Note given here, the other options are much more expensive. Work to be done might include a study of the efficacy/efficiency of each method. This would require the accuracy to be examined in direct relation to the expense. The rates of convergence for GL roughly equals the LS approach, but the QR and LDFE quadratures produce approximately twice the rate of convergence. This may imply that these quadratures would be favored in efficiency for high accuracy demands.

h (Δx)	order	Flux p0 Sn LS	Flux p1 Sn LS	Flux p0 GL	Flux p1 GL	Flux p0 QR	Flux p1 QR
0.1000	2	0.076895	0.007182	0.086357	0.009447	0.090612	0.010341
0.0167	2	0.076944	0.007178	0.086584	0.009496	0.090778	0.010385
0.0250	4	0.086811	0.009460	0.087808	0.009841	0.088517	0.009992
0.0200	6	0.087746	0.009723	0.088130	0.009915	0.088430	0.009974
0.0333	8	0.087962	0.009809	0.088241	0.009939	0.088439	0.009975
0.0500	10	0.087971	0.009837	0.088292	0.009949	0.088401	0.009971
0.0250	12	0.088007	0.009862	0.088330	0.009958	0.088405	0.009972
0.1000	14	0.088078	0.009871	0.088314	0.009946	0.088371	0.009957
0.0167	16	0.088130	0.009898	0.088363	0.009964	0.088406	0.009973

Table 6. Results for couple space-angle convergence analysis for the downscatter problem using different quadratures all using the linear discontinuous spatial discretization.

Case	Flux 0	Flux 1	Rate space 0	Rate space 1	Rate angle 0	Rate angle 1
LD+LS	0.0881292±0.000107	0.00989828±0.000051	3.54647±2.01723	3.83988±1.5191	3.09999±0.493128	2.97501±0.95442
LD+GL	0.0883773±0.000051	0.00996246±0.000019	2.87656±4.6925	4.4019±1.8682	2.23472±0.7042	3.10852±1.4019
LD+QR	0.088405±0.000011	0.00997225±0.000017	3.46771±2.0334	4.23981±1.8452	4.52494±1.5232	3.84999±2.1127

Table 7. Mesh converged solution and convergence rate for the coupled space-angle discretization using different quadratures all using the linear discontinuous spatial discretization.

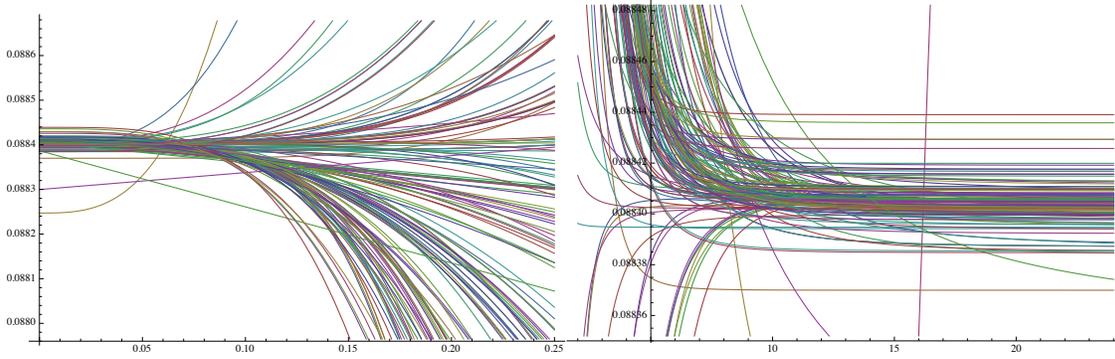


Figure 11. The Spatial and angular variation in models gives a good sense of the need for dismissing outliers from the analysis, which is given further support by the sometimes outlandish results from standard analysis.

Δx	n	LD space flux 0	LD space flux 1
0.0333	2	0.090842	0.010372
0.0333	4	0.088147	0.009937
0.0333	8	0.088387	0.009970
0.0333	16	0.088405	0.009972
0.0333	32	0.088406	0.009972
0.0333	64	0.088406	0.009972

Table 8. Study of the LDFE quadrature method was carried out on a single mesh ($\Delta x=0.033333$) to examine the convergence of the quadrature.

Case	converged flux 0	Converged flux 1	Rate flux 0	Rate flux 1
LD-LDFE	0.0883713±0.0000002	0.0099723±0.0000007	4.9852± 2.70105	4.77744± 1.9618
LD-LDFE CGI	0.088322±0.000148	0.00997226±0.0000002	10.8249	7.24046

Table 9. Results for angle convergence analysis for the downscatter problem using the LDFE quadrature.

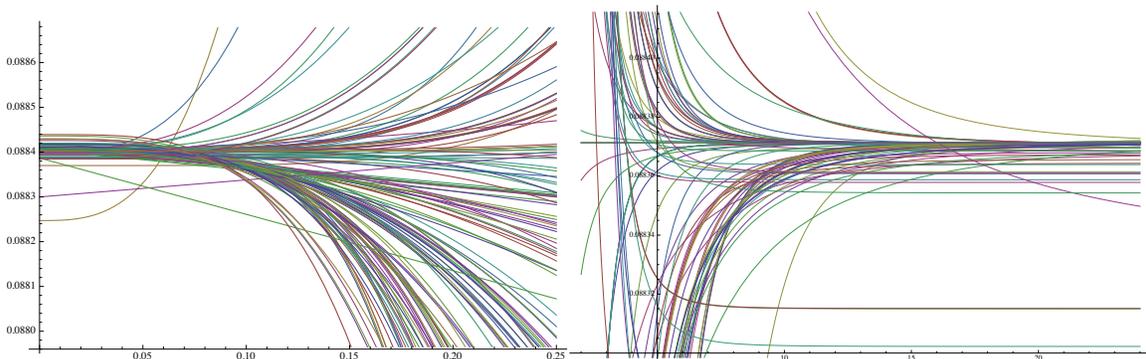


Figure 12. LDFE convergence models for the P0 and P1 integrated flux. The variation in direction is indicative of the non-monotonic convergence of the flux as opposed to any unreliability of the optimization solutions.

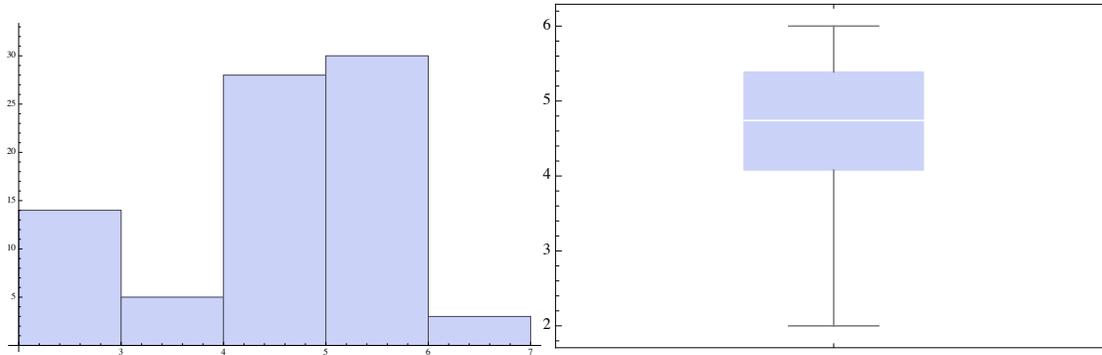


Figure 13. The histogram PDF view of the angular convergence rate and the complimentary box-and-whisker plot showing the relative consistency and spread in estimated convergence rate.

Despite the extreme simplicity of the problems chosen, the verification landscape is enormously complex. We have found a great deal of texture in both the nature of spatial and angular convergence with far too much uncertainty regarding the expected nature. We expect that this nature will rear its head on more applied problems and represents a distinct source of uncertainty moving forward.

We note in closing that each problem has an analytical result, but its closed form was not made available to us. These solutions are evaluated on a reference grid, which was denoted as our “coarse” grid. As such, the comparison with analytical is not code verification, but rather a form of regression testing. True code verification would be a worth future study, and would add significant value to our knowledge of Denovo and the detailed nature of neutron transport in general.

Finally, the impact of the variance of the nonlinear curve fit should be incorporated into the uncertainty in the convergence/error analysis. A preliminary investigation indicates that for many of the estimates, the variance is quite substantial in comparison to the estimated error. Another aspect of the analysis that has not been significantly explored here is the asymmetric estimation of the bias in the solution. Generally, the error is not symmetric with respect to the estimated solution, but rather preferential depending on the nature of the case, resolution and numerical method. This is important is determining whether the solution has a bias toward higher or lower values, which could have profound consequences for applied circumstances.

Acknowledgements

The author thanks Tim Trucano and James Kamm for many fruitful discussions regarding code verification that have helped inform this work. The installation of the working Denovo executable was accomplished through the diligent efforts of Dena Vigil, and the work could not have proceeded without her skill. Tom Evans and the Denovo team have provided support throughout this effort. 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.
- [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/
- [Eva10] T. Evans, *Denovo*, Denovo: A radiation transport code for nuclear applications, SciDAC SciADS 2010, Snowbird Utah, ORNL, 2010.
- [Hoaglin] Hoaglin, D. C.; F. Mosteller and J. W. Tukey, *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, 1983.
- [Huber] P. J. Huber, *Robust Statistical Procedures*, SIAM 1996.
- [Jar10] J. J. Jarrell, *An Adaptive Angular Discretization Method For Neutral-Particle Transport in Three-Dimensional Geomtries*, PhD Dissertation, Texas A&M University, 2010.
- [Lar82] E. W. Larsen and P. Nelson, Finite-difference approximations and superconvergence for the discrete-ordinate equations in slab geometry, *SIAM Journal of Numerical Analysis*, 19(2), pp. 334-349, 1982.
- [LM84] E. E. Lewis and W. P. Miller, *Numerical Methods for Neutron Transport*, Wiley Interscience, 1984.
- [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).
- [Met49] N. Metropolis and S. Ulam, The Monte Carlo Method, *Journal of the American Statistical Association*, 44 (247), pp. 335-341, 1949.
- [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).
- [Rid12] Rider, W. J. and Kamm, J. R., *Advanced Solution Verification of CFD Solutions for LES of Relevance to GTRF Estimates*, Sandia National Laboratory Report SAND 2012-7199P, 2012.
- [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).
- [She10] A. I. Shestakov, R. M. Vignes, J. S. Stolken, *Derivation and Solution of Multifrequency Radiation Diffusion Equations for Homogeneous*

Refractive Lossy Media, LLNL-JRNL-422310, Lawrence Livermore National Laboratory, 2010.

- [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 Solution Verification Coupled Space and Angular

```

psth = 3.; psL = 1; psH = 5.0; ptth = 1.; ptL = 0.5; ptH = 3.0;
model = f0 + bs h^ps + bt /t^pt;
model0 = f0 + bs h^psth + bt /t^ptth;
modelL = f0 + bs h^psL + bt /t^ptL;
modelU = f0 + bs h^psH + bt /t^ptH;
fm[h_, t_, u_, n_] := Abs[ model - u]^n
fm0[h_, t_, u_, n_] := Abs[model0 - u]^n
fmL[h_, t_, u_, n_] := Abs[modelL - u]^n
fmU[h_, t_, u_, n_] := Abs[modelU - u]^n
cons = {psL <= ps <= psH, ptL <= pt <= ptH};
Min2L[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fmL[h, t, u, 2]]; Return[(sum)^0.5])
Min2U[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fmU[h, t, u, 2]]; Return[(sum)^0.5])
Min20[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm0[h, t, u, 2]]; Return[(sum)^0.5])
Min1L[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]]; sum += fm[h, t, u, 1]];
  sum += 0.00001 (Abs[f0/f00] + Abs[bs/bs0] + Abs[ps/ps0] + Abs[bt/bt0] +
    Abs[pt/pt0]); Return[sum])
Min2T[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]]; sum += fm[h, t, u, 2]];
  sum += 0.0000000001 ((f0/f00)^2 + (bs/bs0)^2 + (ps/ps0)^2 + (bt/
    bt0)^2 + (pt/pt0)^2); Return[(sum)^0.5])
Min2[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 2]]; Return[(sum)^0.5])
Min12[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 0.5]]; Return[sum^2])
Min1[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 1]]; Return[sum])
Min4[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 4]]; Return[sum^0.25])
Min8[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 8]]; Return[sum^0.125])
Mini[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum = Max[sum, fm[h, t, u, 1]]; Return[sum])
Min1Lw[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 1] (t/h)];
  sum += 0.00001 (Abs[f0/f00 ] + Abs[bs/bs0] + Abs[ps/ps0] + Abs[bt/bt0] +

```

```

    Abs[pt/pt0]); Return[sum])
Min2Tw[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 2] (t/h)];
  sum += 0.0000001 ((f0/f00)^2 + (bs/bs0)^2 + (ps/ps0)^2 + (bt/bt0)^2 + (pt/
    pt0)^2); Return[(sum)^0.5])
Min2w[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 2] (t/h)]; Return[(sum)^0.5])
Min12w[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 0.5] (t/h)]; Return[sum^2])
Min1w[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 1] (t/h)]; Return[sum])
Min4w[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 4] (t/h)]; Return[sum^0.25])
Min8w[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum += fm[h, t, u, 8] (t/h)]; Return[sum^0.125])
Miniw[dat_] := (sum = 0;
  For[k = 1, k <= Length[dat], k++,
    h = dat[[k, 1]]; t = dat[[k, 2]]; u = dat[[k, 3]];
    sum = Max[sum, fm[h, t, u, 1] (t/h)]]; Return[sum])
data = {{0.25, 2, 0.16124268249677}, {0.25, 4, 0.16397843360126}, {0.25, 6,
  0.16476546214289}, {0.25, 8, 0.16517334950802}, {0.25, 12,
  0.16549090885325}, {0.25, 16, 0.16562667923887}, {0.25, 20,
  0.16570035457266}, {0.25, 24, 0.16574608186218}, {0.125, 2,
  0.11277180736310}, {0.125, 4, 0.11684673059776}, {0.125, 6,
  0.11762107978317}, {0.125, 8, 0.11800166094432}, {0.125, 12,
  0.11829094418725}, {0.125, 16, 0.11841380207920}, {0.125, 20,
  0.11848072931917}, {0.125, 24, 0.11852219490744}, {0.0625, 2,
  0.10790905108521}, {0.0625, 4, 0.11188154163333}, {0.0625, 6,
  0.11267733558198}, {0.0625, 8, 0.11306818708312}, {0.0625, 12,
  0.11336094569033}, {0.0625, 16, 0.11348295719131}, {0.0625, 20,
  0.11354850316339}, {0.0625, 24, 0.11358871942618}, {0.03125, 2,
  0.10754399341971}, {0.03125, 4, 0.11147735795780}, {0.03125, 6,
  0.11226648352730}, {0.03125, 8, 0.11265504708102}, {0.03125, 12,
  0.11294794194459}, {0.03125, 16, 0.11307073474320}, {0.03125, 20,
  0.11313677174308}, {0.03125, 24, 0.11317725200587}, {0.015625, 2,
  0.10751869507894}, {0.015625, 4, 0.11144841371227}, {0.015625, 6,
  0.11223644771749}, {0.015625, 8, 0.11262436432804}, {0.015625, 12,
  0.11291677116577}, {0.015625, 16, 0.11303940127514}, {0.015625, 20,
  0.11310538230224}, {0.015625, 24, 0.11314585451941}};
ti = TimeUsed[]; t0 = ti; h = .; t = .; d = .;
rates = {}; ratet = {}; sol0 = {}; mod = {}; resid = {};

m = NMinimize[{Min20[data]}, {f0, bs, bt}, Method -> "NelderMead",
  MaxIterations -> 1000];
f0 = Abs[f0 /. m[[2]]]; bs0 = Abs[bs /. m[[2]]]; ps0 = psth; bt0 =
  Abs[bt /. m[[2]]]; pt0 = ptth;
r0 = m[[1]];
Print["f0 = ", f0]; Print["bs0 = ", bs0]; Print["bt0 = ", bt0];
Print["r0 = ", r0];

m = NMinimize[{Min2L[data]}, {f0, bs, bt}, Method -> "NelderMead",
  MaxIterations -> 1000];
f0L = f0 /. m[[2]]; bsL = bs /. m[[2]]; btL = bt /. m[[2]];

```

```

m = NMinimize[{Min2U[data]}, {f0, bs, bt}, Method -> "NelderMead",
  MaxIterations -> 1000];
f0U = f0 /. m[[2]]; bsU = bs /. m[[2]]; btU = bt /. m[[2]];

f01 = Min[f0L, f0U]; f02 = Max[f0L, f0U];
bs1 = Min[bsL, bsU]; bs2 = Max[bsL, bsU]; ps1 = psL; ps2 = psH;
bt1 = Min[btL, btU]; bt2 = Max[btL, btU]; pt1 = ptL; pt2 = ptH;

For[j = 0, j < Length[data], j++, Print[j];
  If[j == 0, ldata = data, ldata = Delete[data, j]];
  Print["local data = ", ldata];

  m = NMinimize[{Min20[ldata]}, {f0, bs, bt}, Method -> "NelderMead",
    MaxIterations -> 1000];
  AppendTo[sol0, f0 /. m[[2]]];
  h =.; t =.;
  AppendTo[resid, m[[1]]];
  AppendTo[mod, model0 /. m[[2]]];
  Print["pth model = ", model0 /. m[[2]]];
  Print["resid = ", m[[1]]];

  m = NMinimize[{Min2L[ldata]}, {f0, bs, bt}, Method -> "NelderMead",
    MaxIterations -> 1000];
  AppendTo[sol0, f0 /. m[[2]]];
  h =.; t =.;
  AppendTo[resid, m[[1]]];
  AppendTo[mod, modelL /. m[[2]]];
  Print["pL model = ", modelL /. m[[2]]];
  Print["resid = ", m[[1]]];

  m = NMinimize[{Min2U[ldata]}, {f0, bs, bt}, Method -> "NelderMead",
    MaxIterations -> 1000];
  AppendTo[sol0, f0 /. m[[2]]];
  h =.; t =.;
  AppendTo[resid, m[[1]]];
  AppendTo[mod, modelU /. m[[2]]];
  Print["pH model = ", modelU /. m[[2]]];
  Print["resid = ", m[[1]]];

  t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;

  m = NMinimize[{Min2[ldata],
    cons}, {f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
    bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
    MaxIterations -> 1000];
  AppendTo[sol0, f0 /. m[[2]]];
  AppendTo[rates, ps /. m[[2]]];
  AppendTo[ratet, pt /. m[[2]]];
  AppendTo[resid, m[[1]]];
  h =.; t =.; AppendTo[mod, model /. m[[2]]];
  Print["L2 model = ", model /. m[[2]]];
  t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
  Print["resid = ", m[[1]]];

  m = NMinimize[{Min4[ldata],
    cons}, {f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
    bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
    MaxIterations -> 1000];
  AppendTo[sol0, f0 /. m[[2]]];
  AppendTo[rates, ps /. m[[2]]];
  AppendTo[ratet, pt /. m[[2]]];
  AppendTo[resid, m[[1]]];
  h =.; t =.; AppendTo[mod, model /. m[[2]]];
  Print["L4 model = ", model /. m[[2]]];

```

```

t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min8[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "DifferentialEvolution",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["L8 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min12[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["L 1/2 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min1[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["L1 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Mini[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["Linf model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min1L[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["L1 lasso model = ", model /. m[[2]]];

```

```

t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min2T[lldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["L2 ridge model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min2w[lldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L2 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min4w[lldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L4 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min8w[lldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "DifferentialEvolution",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L8 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min12w[lldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L 1/2 model = ", model /. m[[2]]];

```

```

t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Minlw[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L1 model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Miniw[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted Linf model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{MinlLw[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "RandomSearch",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L1 lasso model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

m = NMinimize[{Min2Tw[ldata],
  cons}, {{f0, f01, f02}, {bs, bs1, bs2}, {ps, ps1, ps2}, {bt, bt1,
  bt2}, {pt, pt1, pt2}}, Method -> "SimulatedAnnealing",
  MaxIterations -> 1000];
AppendTo[sol0, f0 /. m[[2]]];
AppendTo[rates, ps /. m[[2]]];
AppendTo[ratet, pt /. m[[2]]];
AppendTo[resid, m[[1]]];
h =.; t =.; AppendTo[mod, model /. m[[2]]];
Print["weighted L2 ridge model = ", model /. m[[2]]];
t1 = TimeUsed[]; Print["Time solve 1 = ", t1 - t0]; t0 = t1;
Print["resid = ", m[[1]]];

];

Print["Median Residuals =", Median[resid]];
Print["Median Deviation Residuals =", MedianDeviation[resid]];

mmed = Median[sol0]; meddev = MedianDeviation[sol0];
Print["median solution = ", mmed]; Print["median deviation in solution = ", \
meddev];
rsmed = Median[rates]; rsdev = MedianDeviation[rates];
Print["median spatial convergence rate = ", rsmed];

```

```
Print["median deviation in spatial convergence rate = ", rsdev];

rtmed = Median[ratet]; rtdev = MedianDeviation[ratet];
Print["median angular convergence rate = ", rtmed];
Print["median deviation in angular convergence rate = ", rtdev];

Print["number of models = ", Length[sol0]];
Print["Models = ", mod];
Print["constant = ", sol0];
Print["spatial power = ", rates];
tf = TimeUsed[]; Print["total time = ", tf - ti];

Print["Histograms plots"];
Histogram[sol0]
Histogram[rates]
Histogram[ratet]

Print["Model plots"];
t = 24; Show[ListPlot[data], Plot[mod, {h, 0, 0.25}]] Plot[mod, {h, 0, 0.25}]
h = 0.01; Show[ListPlot[data], Plot[mod, {t, 2, 24}]] Plot[mod, {t, 2, 24}]

Print["Box and Whisker plots"];
BoxWhiskerChart[resid, "Median"]
BoxWhiskerChart[rates, "Median"]
BoxWhiskerChart[ratet, "Median"]
BoxWhiskerChart[sol0, "Median"]
```

Appendix B – Python Input For Upscatter

```
#####
## tstupscatter.py
## Thomas M. Evans
## Mon Oct 15 16:34:07 2007
## $Id: tstupscatter.py,v 1.5 2009/02/06 15:32:26 9te Exp $
#####
## Copyright (C) 2007 Oak Ridge National Laboratory, UT-Battelle, LLC.
#####

import sys

#from sc import *
from ld import *
#from tld import *

import tester
from tester import ln

##-----##
## REFERENCE
ref=[
    0.0204834930722, 0.0262652370099, 0.0293849659463, 0.0293849659463,
    0.0262652370099, 0.0204834930722, 0.0262652370099, 0.034848766099,
    0.0414342507292, 0.0414342507292, 0.034848766099, 0.0262652370099,
    0.0293849659463, 0.0414342507292, 0.0574493440752, 0.0574493440752,
    0.0414342507292, 0.0293849659463, 0.0293849659463, 0.0414342507292,
    0.0574493440752, 0.0574493440752, 0.0414342507292, 0.0293849659463,
    0.0262652370099, 0.034848766099, 0.0414342507292, 0.0414342507292,
    0.034848766099, 0.0262652370099, 0.0204834930722, 0.0262652370099,
    0.0293849659463, 0.0293849659463, 0.0262652370099, 0.0204834930722,
    0.0255958929703, 0.0334755672169, 0.0380653157153, 0.0380653157153,
    0.0334755672169, 0.0255958929703, 0.0334755672169, 0.0468502780307,
    0.0635608964631, 0.0635608964631, 0.0468502780307, 0.0334755672169,
    0.0380653157153, 0.0635608964631, 0.142490636319, 0.142490636319,
    0.0635608964631, 0.0380653157153, 0.0380653157153, 0.0380653157153,
    0.0334755672169, 0.0468502780307, 0.0635608964631, 0.0635608964631,
    0.0468502780307, 0.0334755672169, 0.0255958929703, 0.0334755672169,
    0.0380653157153, 0.0380653157153, 0.0334755672169, 0.0255958929703,
    0.0255958929703, 0.0334755672169, 0.0380653157153, 0.0380653157153,
    0.0334755672169, 0.0468502780307, 0.0635608964631, 0.0635608964631,
    0.0468502780307, 0.0334755672169, 0.0255958929703, 0.0334755672169,
    0.0380653157153, 0.0380653157153, 0.0334755672169, 0.0255958929703,
    0.0204834930722, 0.0262652370099, 0.0293849659463, 0.0293849659463,
    0.0262652370099, 0.0204834930722, 0.0262652370099, 0.034848766099,
    0.0414342507292, 0.0414342507292, 0.034848766099, 0.0262652370099,
    0.0293849659463, 0.0414342507292, 0.0574493440752, 0.0574493440752,
    0.0414342507292, 0.0293849659463, 0.0293849659463, 0.0414342507292,
    0.0574493440752, 0.0574493440752, 0.0414342507292, 0.0293849659463,
    0.0262652370099, 0.034848766099, 0.0414342507292, 0.0414342507292,
    0.034848766099, 0.0262652370099, 0.0204834930722, 0.0262652370099,
    0.0293849659463, 0.0293849659463, 0.0262652370099, 0.0204834930722]

##-----##

initialize(sys.argv)
```

```

timer = Timer();
timer.start();

##-----##

db = DB("tstTransport_Solver")

db.insert("num_cells_i", 12)
db.insert("num_cells_j", 12)
db.insert("num_cells_k", 8)

db.insert("num_z_blocks", 1)

db.insert("Pn_order", 0)
db.insert("num_groups", 5)
db.insert("downscatter", 0, 1)

db.insert("tolerance", 1.0e-8)
db.insert("max_itr", 1000)
db.insert("aztec_diag", 0)
db.insert("aztec_output", 0)

db.insert("delta_x", 0.125)
db.insert("delta_y", 0.125)
db.insert("delta_z", 0.125)

db.insert("boundary", "vacuum")

db.insert("upscatter", "gauss_seidel")
db.add_db("upscatter_db", "upscatter");
db.insert("upscatter_db", "upscatter_acceleration", 0, 1)
db.insert("upscatter_db", "up_group_solver", "SI")
db.insert("upscatter_db", "inner_itr", 1)
db.insert("upscatter_db", "tolerance", 1.0e-8)

db.insert("problem_name", "2_ana")

## DECOMPOSITION

if nodes() == 1:

    db.insert("num_blocks_i", 1)
    db.insert("num_blocks_j", 1)

elif nodes() == 2:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 1)

elif nodes() == 4:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 2)

nbnds = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
pbnds = []

db.insert("neutron_bnd", nbnds)
db.insert("photon_bnd", pbnds)

# Angular options
db.add_db("quadrature_db", "quad_options")
db.insert("quadrature_db", "Sn_order", 24)

##-----##

```

```

manager = Manager()
mat      = Mat()
source   = Isotropic_Source()
angles   = Angles()

manager.partition(db, mat, angles)

mapp = manager.get_map()

##-----##
## Material setup

Ng = 5

m00 = [[5.0], [5.0], [5.0], [5.0], [5.0]]
m01 = [[4.0], [6.0], [6.0], [6.0], [6.0]]
m02 = [[3.0], [5.0], [7.0], [7.0], [7.0]]
m03 = [[2.0], [4.0], [6.0], [8.0], [8.0]]
m04 = [[1.0], [3.0], [5.0], [7.0], [9.0]]

m0      = [m00, m01, m02, m03, m04]
sigma_0 = [0.0] * Ng
for g in xrange(Ng):
    sigma_0[g] = 1.0
    for gp in xrange(Ng):
        sigma_0[g] = sigma_0[g] + m0[gp][g][0]

m10 = [[0.5], [0.5], [0.5], [0.5], [0.5]]
m11 = [[0.4], [0.6], [0.6], [0.6], [0.6]]
m12 = [[0.3], [0.5], [0.7], [0.7], [0.7]]
m13 = [[0.2], [0.4], [0.6], [0.8], [0.8]]
m14 = [[0.1], [0.3], [0.5], [0.7], [0.9]]

m1      = [m10, m11, m12, m13, m14]
sigma_1 = [0.0] * Ng
for g in xrange(Ng):
    sigma_1[g] = 1.0
    for gp in xrange(Ng):
        sigma_1[g] = sigma_1[g] + m1[gp][g][0]

mat.set_num(2)

c = [[1,2,3,4], [2, 3, 4], [3, 4], [4], []]

for g in xrange(5):
    mat.assign_upscatter(0, g, sigma_0[g], c[g], m0[g])
    mat.assign_upscatter(1, g, sigma_1[g], c[g], m1[g])

q      = Vec_Dbl(mapp.num_global(), 0.2)
matids = Vec_Int(mapp.num_global(), 1)
srcids = Vec_Int(mapp.num_global(), 0)
indexer = manager.get_indexer()
for k in xrange(1, 3):
    for j in xrange(2, 4):
        for i in xrange(2, 4):
            n = indexer.g2g(i, j, k)
            matids[n] = 0
            q[n]      = 1.0

mat.assign_global_ids(matids)

manager.partition_energy(mat, angles)

```

```

##-----##
## External source setup

manager.setup(source)

shapes = Vec_Dbl(Ng, 1.0)

source.set(1, shapes, srcids, q)

##-----##
## Verify setup

#manager.verify()

##-----##
## Solve

manager.solve(angles)

##-----##
## CHECK OUTPUT

phi = Moments(0)

# get mapping and mesh objects
mapp      = manager.get_map()
indexer   = manager.get_indexer()
mesh      = manager.get_mesh()

# global and local cell numbers
Gx = indexer.num_global(X)
Gy = indexer.num_global(Y)
Gz = mesh.num_cells_dim(Z)
Nx = mesh.num_cells_dim(X)
Ny = mesh.num_cells_dim(Y)
Nz = mesh.num_cells_dim(Z)

if node() == 0:
    print ">>> Partitioned global mesh with %i x %i x %i cells" \
          % (Gx, Gy, Gz)

flux = Vec_Dbl(Gx*Gy*Gz, 0.0)

sum_flux = 0.0
for k in xrange(Nz):
    for j in xrange(Ny):
        for i in xrange(Nx):
            gcell      = indexer.l2g(i,j,k)
            cell       = mesh.convert(i,j,k)
            flux[gcell] = phi.scalar_flux(cell)
            sum_flux += flux[gcell]

sum_flux /= (Gx*Gy*Gz)
print "%i %26.16e " % (Gx*Gy*Gz, sum_flux)

# sum the flux over all processors
total_sum_flux = gsum_double(sum_flux)
print "%i %26.16e " % (Gx*Gy*Gz, total_sum_flux)

#if node() == 0:
#    print "-----manager-----"
#    print dir(manager)
#    print "-----db-----"
#    print dir(db)
#    print "-----mesh-----"

```

```

# print dir(mesh)
# print "-----tester-----"
# print dir(tester)
# print "-----indexer-----"
# print dir(indexer)

mesh = manager.get_mesh()

#for cell in xrange(mesh.num_cells()):
#    n = mapp.l2g(cell)
#    if not tester.soft(phi(cell, 0, 0, 0, EVEN), ref[n], 1.0e-6):
#        tester.fails(ln())

##-----##

timer.stop()
time = timer.wall_clock()

keys = timer_keys()
if len(keys) > 0 and node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)
    print "-----"
    if time > 0.:
        for k in xrange(len(keys)):
            print "%30s : %16.6e" % (keys[k], timer_value(keys[k]) / time)
    print "-----"

##-----##
##-----##
##-----##
## OUTPUT
##-----##

graphics = 1
if graphics == 1:
    # make SILO output
    silo = SILO()
    silo.open("tupverif_medium")

    # output scalar flux by group
    flux = Vec_Dbl(mesh.num_cells(), 0.0)
    #Ng = manager.num_groups()
    Ng = mat.num_groups()

    for g in xrange(Ng):
        phi = Moments(g)
        for cell in xrange(mesh.num_cells()):
            flux[cell] = phi.scalar_flux(cell)
            silo.add("flux_%i" % (g), flux)

    silo.close()

tester.passes("tstupscatter correctly matches reference output.")

##-----##

manager.close()
finalize()

#####
# end of tstupscatter.py

```

#####

Appendix C – Python Input for Downscatter

```
#####
## tsttwd.py
## Thomas M. Evans
## Mon Oct 15 16:34:07 2007
## $Id: tsttwd.py,v 1.6 2008/10/01 19:54:13 9te Exp $
#####
## Copyright (C) 2007 Oak Ridge National Laboratory, UT-Battelle, LLC.
#####

import sys

# does not work in parallel
#print "Space scheme"
#print "1 = step characteristic"
#print "2 = LD"
#print "3 = Trilinear Discont"
#print "4 = weighted diamond differnce with flux fixup"
#print "5 = theta weighted diamond"
#scheme = input('scheme selection # (1-5) =')
#refine = input('refinement amount =')
#order = input('Sn order - even 2-24 =')
scheme = 1
refine = 5
order = 10

if (scheme == 1): from sc import *
if (scheme == 2): from ld import *
if (scheme == 3): from tld import *
if (scheme == 4): from wdd_ff import *
if (scheme == 5): from twd import *

##-----##

initialize(sys.argv)

timer = Timer();
timer.start();

##-----##

db = DB("tsttwd")

db.insert("num_cells_i", 7*refine)
db.insert("num_cells_j", 6*refine)
db.insert("num_cells_k", 3*refine)

db.insert("num_z_blocks", 1)

db.insert("Pn_order", 0)
db.insert("num_groups", 2)
db.insert("downscatter", 1, 1)

db.insert("tolerance", 1.0e-6)
db.insert("max_itr", 1000)
db.insert("aztec_diag", 0)
db.insert("aztec_output", 0)

db.insert("delta_x", 0.1/refine)
db.insert("delta_y", 0.1/refine)
db.insert("delta_z", 0.1/refine)

## DECOMPOSITION
```

```

if nodes() == 1:
    db.insert("num_blocks_i", 1)
    db.insert("num_blocks_j", 1)

elif nodes() == 2:
    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 1)

elif nodes() == 4:
    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 2)

elif nodes() == 8:
    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 2)

elif nodes() == 12:
    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 3)

elif nodes() == 16:
    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 4)

elif nodes() == 24:
    db.insert("num_blocks_i", 6)
    db.insert("num_blocks_j", 4)

## BOUNDARY CONDITIONS

empty = Vec_Dbl(2)
xbnd = Vec_Dbl(2)
xbnd[0] = 1.0

db.insert("boundary", "isotropic")
db.add_db("boundary_db", "bnd_conditions")

db.insert("boundary_db", "plus_x_phi", empty)
db.insert("boundary_db", "plus_y_phi", empty)
db.insert("boundary_db", "plus_z_phi", empty)

db.insert("boundary_db", "minus_x_phi", xbnd)
db.insert("boundary_db", "minus_y_phi", empty)
db.insert("boundary_db", "minus_z_phi", empty)

db.insert("within_group_solver", "SI")

# Angular options
db.add_db("quadrature_db", "quad_options")
db.insert("quadrature_db", "Sn_order", order)

db.output()

##-----##

manager = Manager()
mat = Mat()

```

```

source = General_Source()
angles = Angles()

manager.partition(db, mat, angles)

##-----##
## Material setup

sigma_0 = [[0.4]]
sigma_1 = [[0.6], [0.8]]

cells = []

mat.set_num(1)
mat.assign_id(0, cells)

mat.assign_xs(0, 0, 1.4, sigma_0)
mat.assign_xs(0, 1, 1.5, sigma_1)

manager.partition_energy(mat, angles)

##-----##
## External source setup

manager.setup(source)

source.set_num(1)
source.assign_id(0, cells)

source.assign_isotropic(0, 0, 0.0)
source.assign_isotropic(0, 1, 0.0)

##-----##
## Verify setup

manager.verify()

##-----##
## Solve

manager.solve(angles)

##-----##
## Check output

phi = Moments(0)
phi1 = Moments(1)

# testing precision
tol = 1.0e-8

mapp = manager.get_map()
indexer = manager.get_indexer()
mesh = manager.get_mesh()

# global and local cell numbers
Gx = indexer.num_global(X)
Gy = indexer.num_global(Y)
Gz = mesh.num_cells_dim(Z)
Nx = mesh.num_cells_dim(X)
Ny = mesh.num_cells_dim(Y)
Nz = mesh.num_cells_dim(Z)

if node() == 0:
    print ">>> Partitioned global mesh with %i x %i x %i cells" \

```

```

        % (Gx, Gy, Gz)

flux = Vec_Dbl(Gx*Gy*Gz, 0.0)
flux1 = Vec_Dbl(Gx*Gy*Gz, 0.0)

sum_flux = 0.0
sum_flux1 = 0.0
for k in xrange(Nz):
    for j in xrange(Ny):
        for i in xrange(Nx):
            gcell = indexer.l2g(i,j,k)
            cell = mesh.convert(i,j,k)
            flux[gcell] = phi.scalar_flux(cell)
            sum_flux += flux[gcell]
            flux1[gcell] = phi1.scalar_flux(cell)
            sum_flux1 += flux1[gcell]

sum_flux /= (Gx*Gy*Gz)
sum_flux1 /= (Gx*Gy*Gz)
print "%i %26.16e " % (Gx*Gy*Gz, sum_flux)
print "%i %26.16e " % (Gx*Gy*Gz, sum_flux1)

# sum the flux over all processors
total_sum_flux = gsum_double(sum_flux)
print "flux0 = %i cells %26.16e " % (Gx*Gy*Gz, total_sum_flux)
total_sum_flux1 = gsum_double(sum_flux1)
print "flux1 = %i cells %26.16e " % (Gx*Gy*Gz, total_sum_flux1)

##-----##

timer.stop()
time = timer.wall_clock()

##-----##
##-----##
##-----##
## OUTPUT
##-----##

graphics = 1
if graphics == 1:
    # make SILO output
    silo = SILO()
    silo.open("twd_coarse")

    # output scalar flux by group
    flux = Vec_Dbl(mesh.num_cells(), 0.0)
    #Ng = manager.num_groups()
    Ng = mat.num_groups()

    for g in xrange(Ng):
        phi = Moments(g)
        for cell in xrange(mesh.num_cells()):
            flux[cell] = phi.scalar_flux(cell)
            flux1[cell] = phi1.scalar_flux(cell)
        silo.add("flux_%i" % (g), flux)
        silo.add("flux1_%i" % (g), flux1)

    silo.close()

keys = timer_keys()
if len(keys) > 0 and node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)

```

```
print "-----"
if time > 0.:
    for k in xrange(len(keys)):
        print "%30s : %16.6e" % (keys[k], timer_value(keys[k]) / time)
print "-----"

##-----##

manager.close()
finalize()

#####
#                               end of tsttwd.py
#####
```