**Insights from the Development of an Automated Framework and Test Suite for Verifying a One-dimensional Transport Model**

E. Ateljevich1, K. Zamani2, F. A. Bombardelli2 and J. Anderson1

1Bay-Delta Modeling Branch, California Department of Water Resources (DWR), Sacramento, CA; PH (333) 333-3333; FAX (111) 111-1111; e-mail: eli@water.ca.gov, jamiea@water.ca.gov

2Department of Civil and Environmental Engineering, University of California, Davis, 2001 Ghausi Hall, Davis, CA 95616; PH (530) 752-0949; FAX (530) 752-7872; email: [kzamani@ucdavis.edu](mailto:kzamani@ucdavis.edu), [fabombardelli@ucdavis.edu](mailto:fabombardelli@ucdavis.edu)

**ABSTRACT**

In this paper we describe a framework for software verification of a transport model. The criteria of the framework are driven by the requirements for the model, i.e., by the processes embedded in the model, but are crafted according to principles from both the software and numerical testing fields. Herein, we describe the components and implementation of the suite, emphasizing the incremental nature of the tests, quantitative criteria for testing and the tension between the silent, automatic perspective of software testing and the verbose, graphical requirements required for public reporting of numerical verification results. Our experience might result in a useful starting point for researchers and practitioners wanting to verify codes in similar situations. WE CAN GO AS FAR AS 150 WORDS

**INTRODUCTION**

The California Department of Water Resources maintains Delta Simulation Model 2 (DSM2), a one-dimensional (1D) hydrodynamic and transport model for rapidly simulating flow and water quality in the Sacramento-San Joaquin Delta (Fig. 1). Recently, the authors commenced work on a more flexible and more rigorously verified transport component for this suite. Our target problems include river and estuary advection, 1D approximations of common mixing mechanisms and source terms associated with sediment, radiation and non-conservative water quality kinetics.

When faced with the need of verification and validation of our software developments, we found that rigorous testing frameworks are seldom implemented -- which was surprising considering the many pieces of code of similar characteristics developed in the last 40 years. Therefore, we embarked on the development of a new framework for such purpose.

In this paper, we describe our approach and experiences developing a software verification framework for a transport model. We begin by describing the motivation and requirements for testing. The criteria are driven by the requirements for the model, but are crafted according to principles from both the software and numerical testing fields. We then describe the components and implementation of the suite, emphasizing the incremental nature of the tests, quantitative criteria for testing and the tension between the silent, automatic perspective of software testing and the verbose, graphical requirements required for public reporting of numerical verification results.

**Problem definition, scaling and motivation of the testing.** Both the scaling of the problem and our choice of algorithm influence the components of our test suite. The dynamics are formulated in conservative form as follows:



where *A* is the wetted area, *C* is the scalar concentration, *u* is the flow velocity, *K* is longitudinal dispersion coefficient, and *R* is the source term (deposition, erosion, lateral inflow and other forms of sources and sinks).

The problem domain includes estuaries and river channels and even some open water areas grossly approximated as channels (Fig. 1). The main transport process is advection, and the mixing mechanisms we anticipate are turbulent diffusion, gravitational circulation, and shear dispersion. We anticipate the shear dispersion to dominate over the turbulent diffusion, but we also expect the gravitational circulation to exert an important role in mixing. We additionally contemplate significant, non-linear source terms. While none of the above processes are so quickly varying as to constitute truly stiff reactions, we have CCCC

Our algorithms includes an explicit scheme for advection based on finite-volumes (FVM) and the Lax, two-step method with van-Leer flux limiter; it also includes an implicit, time-centered Crank-Nicolson scheme for dispersion. The advection and reaction solver are coupled as a predictor corrector pair, and diffusion is implemented using operator splitting. Two features of the algorithm are particularly important. First, the scheme requires a flow field flow discharges and flow areas) that preserves mass continuity. In some cases tests from the literature were written in non-conservative or primitive form in terms of a velocity and had to be reworked in conservative form. Second, we employ operator splitting and wanted to exercise the equations with and without known vulnerabilities (such as time-varying boundaries and nonlinear source terms) of this class of algorithm.

The target accuracy is strict second order for individual operators and near second-order for the algorithm as a whole. Second order allows coarser discretization for a modest increase in work. As computer architectures favor multiple operations with minimal movement of data, this advantage of computation over grid density is on the increase. A second order algorithm also gives us a buffer of accuracy as details like networks of channels and coarse boundary data are added. At the time of writing this paper, our splitting is first order Godunov splitting; one goal of our testing is determine how applicable the practical observation that near second-order accuracy can be achieved with first order splitting (e.g. Leveque 1986) is to our algorithm.

**Testing principles**. Flow and transport codes inherently comprise both numerical algorithms and pieces of software. Although some well-developed testing literature exists for both, SANDIA describes some elements of software quality engineering in the context of numerical verification, and notes some cultural reasons why it is seldom implemented. There are myriad approaches to software testing. The principles that we want to emphasize are:

1. Testing should be automatic and continuous.
2. The approach should foster good architecture and exact specification of every unit of code
3. Tests should be designed around the specific ways developers make mistakes.
4. The approach should test ever important special case for sub-processes.
5. Tests need not have diagnostic value.

One goal of tests is that they be a continuous assessment of the code. The tests themselves stay static, and establish a gauntlet of through which future changes must be passed. Such a test is usually called a regression test, and regressions are harvested from past unit tests, system tests and mistakes. One consequence of automation and regression is that test suites must be based on binary *assertions,* true and false statements that can be tested without human intervention and that reveal whether the aspect of the code under consideration is correct. Unit tests exercise single units of code, such as one small subroutine calculating a gradient over a mesh. Properly designed unit tests exercise all the important “corner” cases and special inputs. Examples for a gradient would be:

1. One well-behaved central case.
2. A test of edges of the mesh.
3. Tests that exercise features such as flux limiters with steep or zero gradients in both directions.

Unit tests are fast, and they allow a great number of special cases to be exercised, which might take months when the case is embedded in a larger solver. For instance, in the example of a gradient with limiter, imagine that the limiter contains a bug in one direction. Normal verification based on convergence might miss this in two ways. First, convergence is usually assessed without limiters, as they are order reducing. Second, algorithmic tests are usually constructed to show the algorithm solves a Partial Differential Equation, not that the code does this right in both directions.

\*\*\*\* unit test vs Sandia assertion the convergence tests catch everything.

Incremental 9454948223

above example, it is easy to imagine a full system test

1. Once passed, unit tests act as regression tests.

The only downside of unit tests seem to be unawareness or reluctance to code them. These problems can be exacerbated during parallel computation.

Regression tests: these tests rehearse old mistakes to prevent new coding from re-introducing errors. Regression testing is often the only type of test that can be applied to a legacy code whose architecture is not test friendly.

System or integration tests: System tests include the interaction of components in a meaningful way. A small system might represent a single operator in the algorithm. A large system might represent the whole time update. The behavior tested by a system test will make sense to a user and the criteria are is often reportable artifacts that might be used in a publication. In this regard, we have observed at least one “cultural” issue: the criterion that a test be repeatable and automatic may conflict with the desire to report results.

**NUMERICAL VERIFICATION**

Verification and validation (V&V) are the most important steps to assess the accuracy and reliability of any numerical simulation. In simple words, in the step of verification the correctness of the solution technique used and implementation are evaluated. In turn, validation consists in evaluating whether the model is properly representing the processes involved in the problem. Lax Equivalence Theorem which indicates that for a solver to be stable and consistent it is required to pass the convergence test, is used to assess the consistency and stability of a numerical scheme based on its convergence. As a result, the mesh convergence study became a well-recognized and the standard verification method of CFD codes. The ratio of consecutive error norms is a perfect vehicle to catch any coding error/algorithm problem. The points need to be considered in any mesh-convergence study include:

* For the FDM and FVM descritzations developed under the assumption of smooth function, the discontinuities and jagged initial or boundary conditions can locally or globally decrease the convergence rate. Linfinity[[1]](#footnote-1) should be included as an ultimate diagnostic tool for local errors and worst case scenario. “L2” is more forgiving norm compare to the first error norm “L1”. We recommend L1 as an appropriate global metric of error.[[2]](#footnote-2)
* Convergence ratio in a very coarse grid oscillates around its main value, as the grid size is refined convergence becomes monotone until the mesh size reaches to a point where the machine precision overtakes the truncation error of the numerical scheme, at this point error norms do not change and convergence rate is zero.
* Although the convergence is a reliable warning of a defect, it should not be forgotten that the main goal in practice is a more accurate solver. Therefore the superiority of methods should be assessed both based on convergence and accuracy. Accuracy metrics similarly are error norms as is discussed above, however for evaluating the accuracy error norms should be normalized by an appropriate scale of the solution.
* All of the convergence tests such as MMS, Richardson Extrapolation, could be run by a same driver. The post processing of the convergence test also could carry out with a same code for all the tests.
* Visualization of time evolution of error and results in the solution domain is a decent strategy for debugging in cases the source of inaccuracy is obscure.

Benchmark for calculating error norms

Figure : Stages of CFD Modeling and the

Conceptual Model (Continuum)

Computational Model (Discertized)

Natural Phenomenon

(Reality)

Validation

Qualification

Verification

Programming

Simplification

Simulation

The question here is: if one wants to find accuracy and convergence ratio of a scheme in which the analytical solution is unknown (absence of analytical solution is the main motivator towards all numerical methods), what should be done? It is ideal to test a model’s correctness by comparing its numerical results with analytical solutions; however the difficulty is that there is not a general solution for the non-linear IBVP in hydrodynamics. There are some ways to deal with this problem from the simplest to the most sophisticated:

Comparing with a higher order code/run on dense mesh: (the benchmark solver must be verified beforehand) the other issue here is the circularity in this method, there must be one verified code available at the beginning if not we hit an impasse.

Richardson Extrapolation is the common method for dealing with commercial packages and multidimensional complex systems (Roache and Knupp, 1993), however the drawback is that the method only checks if the solver converges and it is not able to measure where it is converging to.

Difficulties arise in Richardson EXTRAPOLATION???(BC/IC incompatibility?)

Method of Manufactured Solutions (MMS) (Wang and Jia, 2009), and Prescribed Solution Forcing Method (PSF) (Dee and Da Silva, 1986). The basic concept of the MMS and PSF is to compare the correctness of numerical solvers using an arbitrary manufactured function. MMS and PSF are conceptually following the same idea, although the former is more general than the latter. PSF have been used for the verification cases in which the user can not access the source code to define boundary conditions such as some groundwater codes.

The MMS is a general approach to provide a certain analytical solution of the governing equation for the question of model testing and verification of non-linear numerical solvers in rigorous procedure. Since only the numerical method is to be tested (not the physics of the problem) it would be effective if an arbitrarily made non-linear function can be used in model verification. The exact solution which is manufactured in this method does not need necessarily be realistic (Roache 2009, 2002; Wang et al., 2009) but the authors recommend to chose it the reasonable ranges. We want a benchmark solution that is non-trivial but analytical, and that exercises all ordered derivatives in the error expansion and in all terms.

Let the differential equation be expressed as:

 (2)

in which L denotes the differential operators and u is the variable to be solved. When a manufactured function φ is substituted into the differential equation, one would have:

 (3)

Since φ is not the solution of differential equation, the non-zero R is obtained analytically. In the solver, the numerical solution of this equation would be forced to converge to φ with the analytical forcing term R being added to the mathematical equation of the numerical model as the source term. The verification of a numerical model is simple because the solution of equation (1) is known; one needs only compare the difference between the manufactured analytical function, φ, and the numerical solution of equation (1). Although the function φ can be manufactured arbitrarily, it has to be non-trivial for all the terms of the involved mathematic equations to make a meaningful verification tests. MMS does not require the satisfaction of any particular boundary condition other than those defined by φ along the boundaries of computational domain. The difficulties in MMS are the parameters in the equation (1) are need to be checked under the same situation on an estuarine problem because the solver is work in especial ranges of dimensionless numbers also the area and velocity should satisfy the continuity of mass. The following example clarifies the method (Should I provide an example here?):

Scaling of the problem for an estuary

The ADR solver is only working in the feasible ranges of dimensionless numbers () so in case the reaction rate in equation (1) should not exceed a certain limit, and generally speaking the test suit has to be designed within the natural scales of the physical problem. The assumed scales and ranges are as follows: Area~ 1000 [m2], C (0 – 0.05) [vol/vol=1], u (±0.2-2) [m2/s],

**ACKNOWLEDGMENTS**

1. ,,, where *v*= U num - U exact [↑](#footnote-ref-1)
2. It is proven that kL∞ ≤ L2 ≤ L1 ≤ L∞ where k is a constant and 0<k<1, here norms are assumed to be scaled. [↑](#footnote-ref-2)