**Using Software Assurance and Algorithm Testing to Verify a One-dimensional Transport Model**

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

1Bay-Delta Office, Delta Modeling Section, California Department of Water Resources (DWR), 1416 Ninth Street Sacramento, CA, P.O. BOX 942836; PH (916) 645-5455; FAX (916) 653-6077; 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 framework is 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 outputs 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**

In this paper, we describe our approach and experiences developing a software verification framework for a one dimensional transport model of advection, diffusion and reactions or sources. We begin by describing the motivation and requirements for testing. Our acceptance 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 outputs required for public reporting of numerical verification results.

**Description of requirements and motivation of the testing.**

The California Department of Water Resources maintains the 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. Recently, the authors commenced work on a 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.

The formulation of our problem, scaling of our target modeling applications and choice of algorithm influence the components of our test suite. The model is based on the 1D transport equations in conservative form:



Source/Reaction

Evolution

Advection

Dispersion

where *A* is the wetted area, *C* is the scalar concentration, *u* is the flow velocity, *K* is the longitudinal dispersion coefficient, and *R* is the source term (deposition, erosion, lateral inflow and other forms of sources and sinks). Equation (1) describes the mass conservation of a pollutant in dissolved phase, or suspended sediment away from the streambed.

The problem domain includes estuaries and river channels and even some open water areas grossly approximated as channels. 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 obviously 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, though none of the above processes are so quickly varying as to constitute truly stiff reactions.

Our algorithms include an explicit scheme for advection based on the finite-volumes method (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 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. A second-order algorithm 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. Numerous authors (e.g. Leveque 1986) have observed that near second-order accuracy can be achieved with first order splitting, and the design of the tests probes this point.

**TESTING PRINCIPLES**

Flow and transport codes inherently comprise both numerical algorithms and pieces of software. Well-developed testing literature exists for both. Oberkampf and Trucano (2002) describe some elements of SQE (software quality engineering) in the context of numerical verification, and notes some cultural reasons why it is seldom implemented.

Figure \*\*\* is adapted from this work. We incorporate both numerical and software principles testing in our suite. We regard numerical verification as our key responsibility and the numerical verification toolset as our greatest asset. Nonetheless, we also comment below on how these tools feature as tests; we find that the reporting requirements for verification are in fact sometimes in tension with the principles of good software testing.

**Software Testing Principles.**

Numerical verification is the standard of success of the underlying code, however there are certain software testing principles that we feel help create a framework for the numerical testing. The principles that we want to emphasize are:

1. Testing should be automatic and continuous.
2. The approach should foster exact specification of every unit of code.
3. Testing should provide assurance of whether a set of specifications is met.

One goal of tests is that they be a continuous assessment of the code. The entire suite is a *regression*  suite that establishes a gauntlet through which future changes must be passed. A 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. Convergence criteria are a rigorous basis for assertions, either by requiring strict convergence criteria (“the algorithm is O(2) in time and space”) or a *regression* criterion (“convergence will not get any worse than last time”).

The software testing literature further distinguishes between *unit tests* of atomic routines and *system tests* of larger subtasks. For example, the evaluation of a gradient might be a unit of code and convergence tests can be thought of as system tests.

The software testing point of view is that code must be exercised over a range of inputs that covers every line. For instance, to test a gradient routine with a slope limiter, a developer would want to cover:

1. well-behaved cases in the middle of the mesh.
2. behavior near the edges of the mesh, where one-sided differences may be used instead of central differences.
3. cases that test the limiters with steep or zero gradients in both directions.

Any system test will certainly exercise the central cases, which in any event can seldom be wrong without being obvious. A system test might miss a bug in the limiter for the case of steep decreasing slopes for several reasons. First, convergence is often assessed with limiters turned off, as they are locally order reducing. Second, it is hard to fiddle with the problem in just the right way to make sure the left, right, and center cases of the gradient limiter are all triggered. This is particularly true when trying to exercise all the other units of code the same way – parameter changes made to fully exercise one unit of code may lessen the coverage of another unit.

Overall, we agree with the conclusions of \*\*\*\* that system tests expose bugs well, particularly when an attempt is made to test symmetrically and over special cases. We feel that the hierarchical approach we describe in the next section further helps to isolate problems. Nevertheless, a close reading of \*\*\* does reveal that the convergence tests sometimes initially failed to pick up bugs that are exactly the sorts unit tests might catch (e.g. gaffes in corner cells). We began our coding with near-100% coverage by unit tests and discoveries made in the context of system tests are analyzed and pushed back into unit tests whenever possible.

**Numerical Verification and Algorithmic Testing**

Algorithm tests such as convergence tests serve multiple purposes. They are intended in part to discover bugs (*system test*) and in part to convince ourselves and others of the merit of the algorithm to solve the problems to which it is directed (*acceptance test*).

One of the well-recognized and the standard verification methods of computational-fluid-dynamics (CFD) codes is based on the notion of mesh convergence. Mesh convergence for models that solve partial differential equations is assessed by successively refining the spatial and temporal discretization. As the mesh converges, the error estimates (usually an Ln norm such as sum absolute error or sum squared error) should decrease at a rate that is usually called the *order of convergence*. By checking convergence, we ensure that the model is consistent with an underlying formulation rather than numerical artifacts. Failure to converge usually represents either a bug in the implementation or a difficulty of the algorithm on a class of problem.

The verification toolkit is largely targeted at providing test problems and methods to estimate error in situations where an analytical solution is not available from the literature. When nonlinearity, spatially varying coefficients and other complexities are introduced, tricks must be introduced to obtain good test problems, including the Method of Manufactured Solutions (MMS). MMS can nearly always provide a solution, though ensuring that the MMS problem scaling is reasonable can provide an added challenge.

Depending on the context, error and convergence are usually estimated one of two ways:

* When successive refinements are assessed relative to an analytical solution, we have a direct estimate of error and the ratio allows us to estimate a *convergence rate.*
* When successive grids are compared to one another, we can invoke the concept of Richardson extrapolation and Grid Convergence Index to indirectly estimate error and convergence even when no solution is available.

In practice, we feel that use of Manufactured Solutions (\*\*\*\*) has come to dominate verification problems and Richardson extrapolation is most useful for *in situ* estimates. We will exclusively discuss closed, analytical solutions and manufactured solutions here.

At least in theory, convergence rates can be stipulated as a project requirement and software testing assertion. Convergence rates, not absolute error, are what numerical methods tend to promise. Still, though convergence is a reliable warning of a defect, the main goal in practice is a more accurate solver. Therefore, the superiority of methods should be assessed based on both convergence and accuracy.

Convergence ratios in a very coarse grid oscillates around its main value; as the grid size is refined, convergence becomes monotonic until the mesh size reaches 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. Convergence ratios should be checked for intermediate grid sizes, preferably at the scale of the real phenomenon and discretization used in practice. In the conclusions, we describe some frustration with messages warning of us of test failures from methods that converge, say, at order 1.97.

As acceptance tests, algorithm tests should be conducted over a range of problems that exercise the major physical features that are to be modeled. The community may help with this by providing benchmarks, but we were unable to ascertain that this was the case for our situation. Therefore we collected a series of problems that we think cover most of the cases of interest to us.

As system tests we believe that the tests should be *glass box*, targeting known or discovered vulnerabilities of the algorithm. The layering of the advection, diffusion and reaction components of our convergence tests, for instance, is specifically motivated by our desire to use operator splitting.

Numerical verification In Figure \*\*\*, we have noted that numerical verification and algorithmic tests can be thought of as *system tests*. As such, they tend to have exercises are not necessarily “tests” in the sense conveyed in the previous paragraph at all, but to the extent that they are it is in the role of a *system test*.

* Norms: L∞[[1]](#footnote-1) should be included as an ultimate diagnostic tool for local errors and worst case scenario. L2 is a more forgiving norm compared to the first error norm L1. We recommend L1 as an appropriate global metric of error.[[2]](#footnote-2)
* 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 recommended strategy for debugging in cases where the source of inaccuracy is obscure.

**Case study: Two Wrongs Make a Right in Bidirectional Flow**

**Case study: Two Wrongs Make a Right in Bidirectional Flow**

**Conclusions and lessons learned**

Our test suite succeeds both in finding bugs and in elucidating the strengths and weaknesses of the algorithm on various types of problems. We feel that our test suite is comprehensive for a class of problems that lack benchmarks. We have been able to establish second order convergence on a wide variety of problems despite operator splitting and excellent accuracy on even some of the problems where we miss convergence goals – such as problems with active time-varying boundaries or nonlinearity.

The key issues we have had to sort through are establishment of meaningful test assertions and the tension between test criteria and the need to demonstrate quality. These difficulties mostly lay in the algorithm tests, because the unit test component of the suite does not have a dual job description. The only difficulties with unit tests seem to be culture: generating the will to write them and the skills to write them in a way that covers the unusual cases. There are automatic tools to help discover whether this type of coverage is complete.

When it comes to algorithm tests, nominally we sought a convergence rate O(2) for all components of our algorithm. A convergence criterion seemed in-keeping with the way numerical model accuracy is expressed and less arbitrary than a hard-wired, scale-dependent absolute standard. Early on, however, it was clear that noisy convergence could spoil even a success when expressed as a hard assertion. We were frustrated by numerous test failure messages due to convergence rates such as 1.97 which surely would have passed a graphical acceptance test. We were also occasionally thwarted by sensitivity to problem parameters.

Given a stream of messages, we generally either fixed the code or we searched for bugs until both of the following things happened:

* Convergence properties corresponded well to the expected strengths and limitations of our algorithm; and
* The solution was accurate – qualitatively excellent when compared graphically to solutions and with relative errors of a hundredth of a percent.

We have done our best to support our claims on the first condition. Our hierarchical suite can identify with good precision exactly which added layer of complexity causes a drop in order. The second condition reality check in our requirements for certain types of complex problems. We are in the process of changing our criteria in some cases to an absolute accuracy requirement paired with a regression standard for convergence (“don’t do worse than last time”).

* Lessons/Challenges
  + Expressing numerical tests as assertions
    - converting analytical result comparisons
    - strict O(2) standards trigger too many failures
      * regression (better than before, but less than O(2)?
  + Reporting and the fact that some tests really are for human consumption.
  + The human factors
    - Factors that lead to reluctance
    - Tendency to blame algorithm, precision or smoothness over bugs is ubiquitous
  + “Tests are buggier than the code” phenomenon
  + Tests get changed.
  + FORTRAN compatibility with standard testing tools like continuous integration tools

**TEST SUITE DESIGN**

**General approach.** Our test suite is based on four hierarchical steps, as follows:

1) Testing of the basic components of each physical process (units) for correctness. This involves testing each sub-routine, such as the routine which coarsens the values on the mesh. One of the obvious tests is to check that the coarsen process is correctly undertaken. CCCC

2) System testing of physical processes at level 1: advection, dispersion and reaction.

3) System testing of physical processes at level 2: advection-dispersion, advection-reaction, and dispersion-reaction,

4) System testing of physical processes at level 3: advection-dispersion-reaction.

It is worth mentioning that in each category, we tested two conditions for the boundaries: a) Boundaries close to the main transport events, b) boundaries far from those events.

**Detail of tests.** We detail below the tests developed, and comment on the results.

FABIAN AND KAVEH

**Results.** FABIAN AND KAVEH

**LESSONS LEARNED AND CHALLENGES**

ELI

**CONCLUSIONS AND FINAL REMARKS**

**ACKNOWLEDGMENTS**

Mrs. Tara Smith and Dr. Francis Chung. Support by DWR.

**REFERENCES**

Abbott, M. B., Price, W. A. (Eds.), (1994), “Coastal, Estuarial, and Harbour Engineers’ Reference Book”, Chapman & Hall.

Colella, P., Puckett, E. G., (1998), *“Modern Numerical Methods for Fluid Flow”,* Class Note.

Ferziger, J. H., Peric, M., (2002), “*Computational Methods for Fluid Dynamics”,* Springer.

Fischer, B. H., List, J. E., Koh, R. C., Imberger, J., Brooks, N. H., (1979),  *“Mixing in Inland and Coastal Waters”*, Academic Press, Inc.

Fletcher, C. A. J., (1991), *“Computational Techniques for Fluid Dynamics”*, Springer.

Garcia, M. H., (Ed.), (2008), *“Sediment Engineering, Process, Measurements, Modeling, and Practice”*, Manuals and Reports on Engineering Practice No. 110, ASCE, EWRI.

Leveque, J. R., (1986), *“Intermediate Boundary Conditions for Time-Split Methods Applied to Hyperbolic Partial Differential Equations”*, Mathematics of Computation, Vol. 47 (175), pp. 37-54.

Leveque, J. R., (2002), “Finite Volume Methods for Hyperbolic Problems”, Cambridge University Press.

Oberkampf, W. L., Trucano T. G., (2002), *“Verification and Validation in Computational Fluid Dynamics”*, SANDIA REPORT, No. SAND2002-0529.

Roache, P. J., (2009), *“Fundamentals of Verification and Validation”*, Hermosa Publishers.

Roache, P. J., Knupp, P. M., (1993), *“Complete Richardson Extrapolation”*, Communication in Numerical Method in Eng. Vol. 9, pp. 365-374.

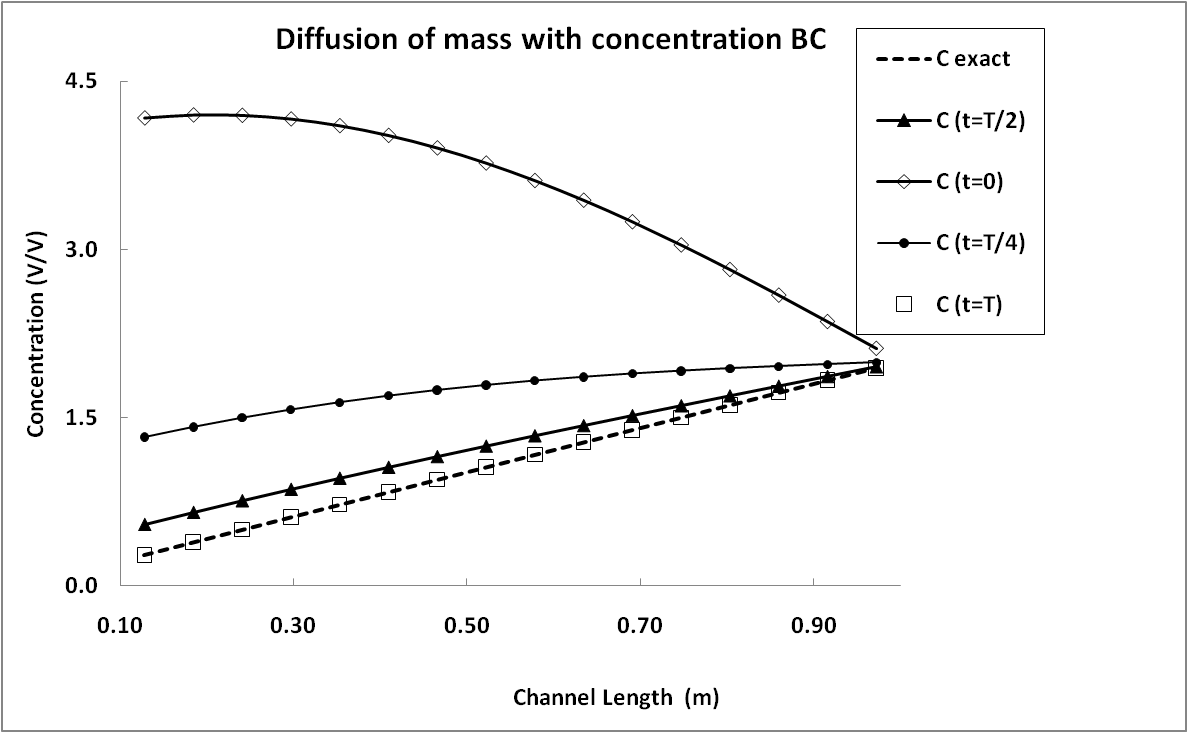
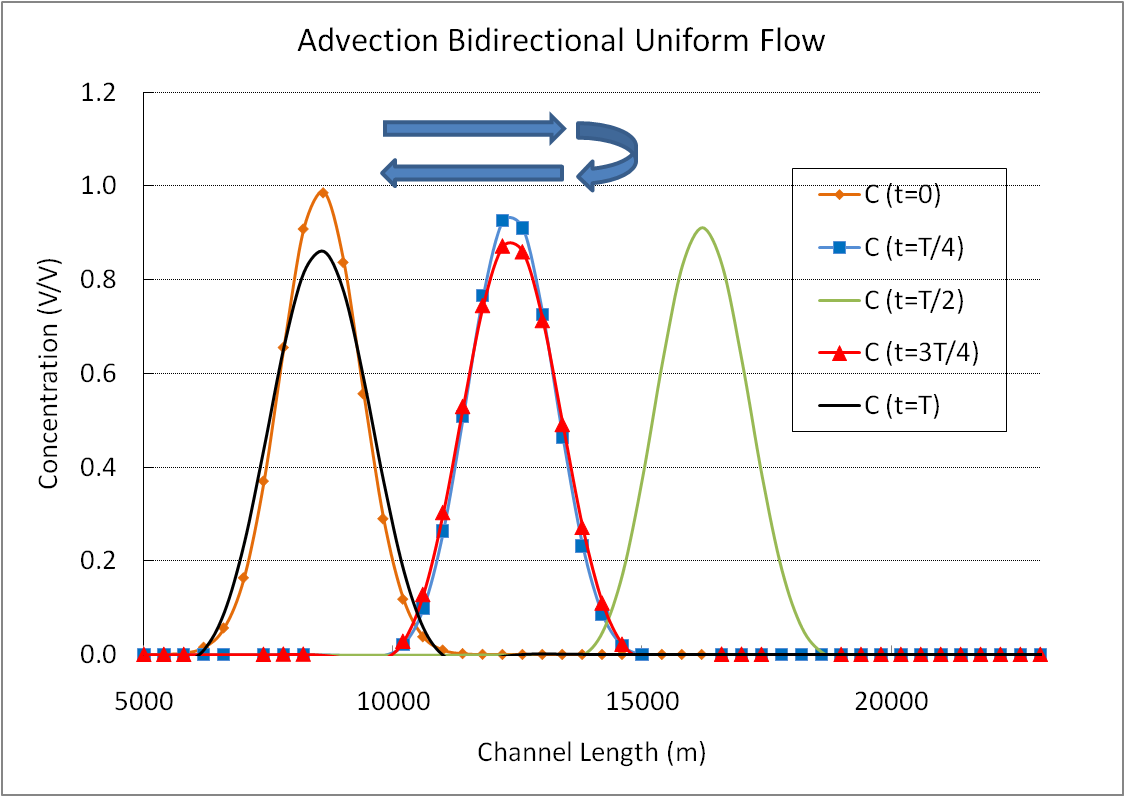
Saltzman, J., (1994), *“An Unsplit 3D Upwind Method for Hyperbolic Conservation Laws”*, J. of Computational Physics, Vol. 115, pp. 153-168.

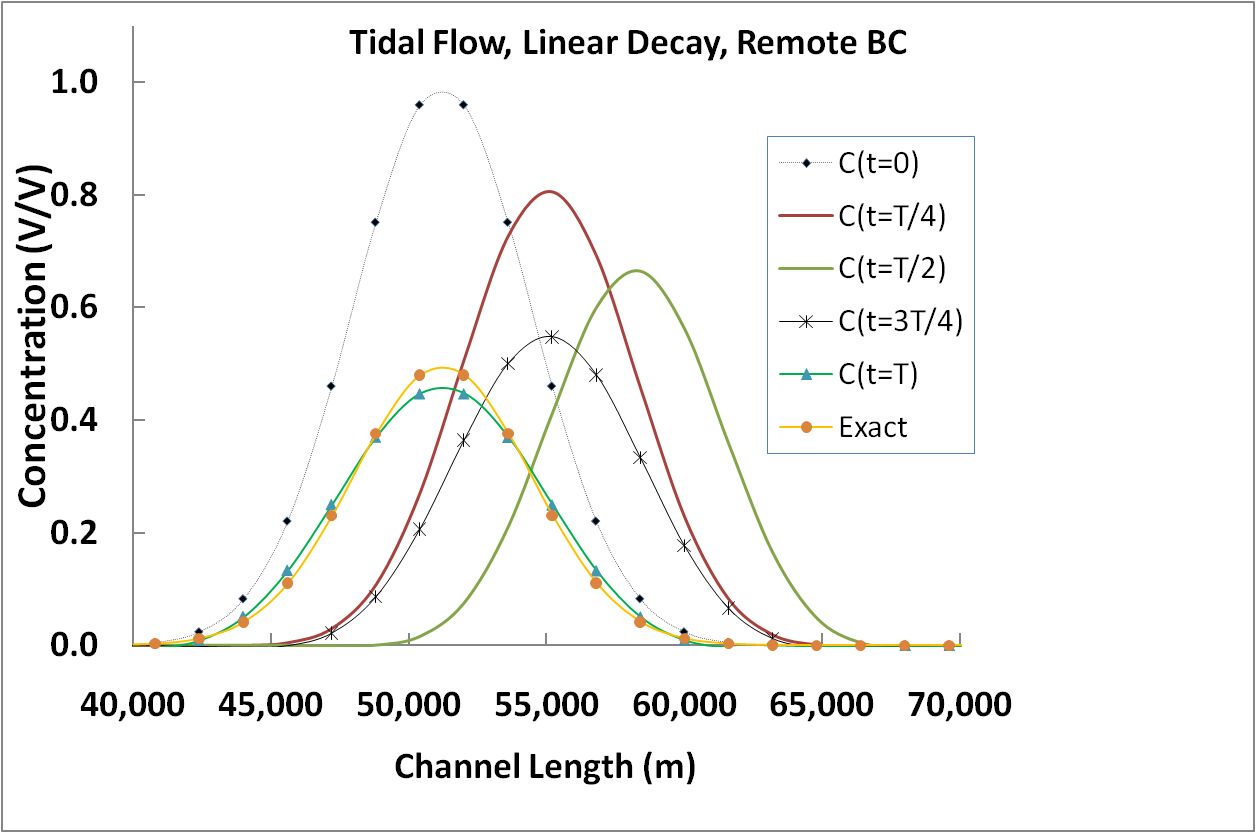
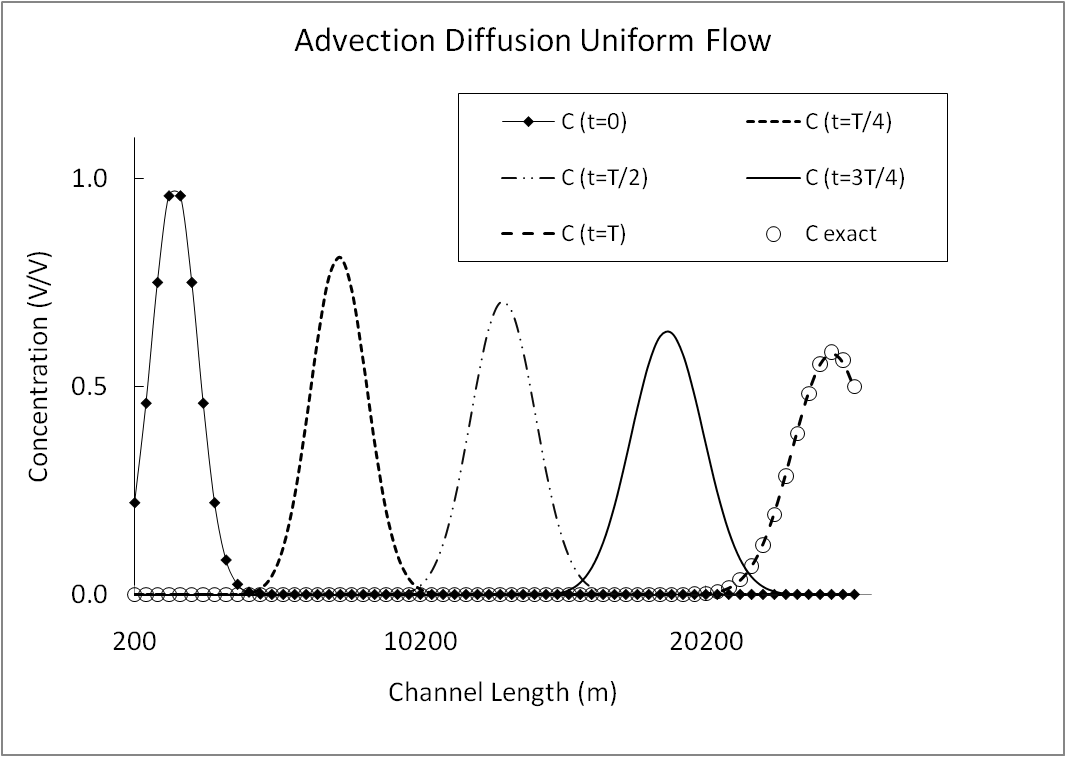
van Rijn, L. C., (1993), “*Principle of Sediment Transport in Rivers, Estuaries, and Coastal Seas”*, AQUA Publications.

Zoppou, C., Knight, J. H., (1997), “*Analytical Solution for Advection and Advection-Diffusion Equation with Spatially Variable Coefficients*”, J. Hydraulic Eng., Vol. 123(2), pp. 144-148.

**SCALING OF THE PROBLEM FOR AN ESTUARY**

The ADR solver is only working in the feasible ranges of dimensionless numbers (Peclet number and Damkohler number,) 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) [m/s], *u*\* isshear velocity is scaled based on where g is gravitational acceleration, *Rh* is hydraulic radius and *S* is bed slope. The longitudinal dispersion coefficient scales with:  Where *B* is width, *H* is depth and is average velocity (Abbott 1993), *K*s ~[2-150 m2/s]. Finally based on the formula suggested by Garcia and Parker (1991) reaction variation range for non-cohesive sediment in an estuary will be ( +1.0×10-3 to -3.7×10-4 [1/s]). That is to mention the length scale in the scaling process assumed the same length of spatial discretization where needed.





1. [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)