Geosci. Model Dev., 4, 435–449, 2011
www.geosci-model-dev.net/4/435/2011/
doi:10.5194/gmd-4-435-2011
© Author(s) 2011. CC Attribution 3.0 License.

Geoscientific Model Development

Automated continuous verification for numerical simulation

P. E. Farrell[1], M. D. Piggott[1,2], G. J. Gorman[1], D. A. Ham[1,2], C. R. Wilson[3], and T. M. Bond[1]

[1]Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, SW7 2AZ, UK
[2]Grantham Institute for Climate Change, Imperial College London, London, SW7 2AZ, UK
[3]Lamont-Doherty Earth Observatory, Columbia University, New York, USA

Received: 23 August 2010 – Published in Geosci. Model Dev. Discuss.: 29 September 2010
Revised: 6 May 2011 – Accepted: 10 May 2011 – Published: 16 May 2011

Abstract. Verification is a process crucially important for the final users of a computational model: code is useless if its results cannot be relied upon. Typically, verification is seen as a discrete event, performed once and for all after development is complete. However, this does not reflect the reality that many geoscientific codes undergo continuous development of the mathematical model, discretisation and software implementation. Therefore, we advocate that in such cases verification must be continuous and happen in parallel with development: the desirability of their automation follows immediately. This paper discusses a framework for automated continuous verification of wide applicability to any kind of numerical simulation. It also documents a range of test cases to show the possibilities of the framework.

1 Introduction

Since the development of the computer, numerical simulation has become an integral part of many scientific and technical enterprises. As computational hardware becomes ever cheaper, numerical simulation becomes more attractive relative to experimentation. Much attention is paid to the development of ever more efficient and powerful algorithms to solve previously intractable problems (Trefethen, 2008). However, in order to be useful, the user of a computational model must have confidence that the results of the numerical simulation are an accurate proxy for reality. A rigorous software quality assurance system is usually a requirement for any deployment to industry, and should be a requirement for any scientific use of a model. Catastrophic accidents such as the Sleipner platform accident, in which an offshore platform collapsed due to failures in finite element modelling, underscore the importance of such efforts. More recently, the controversy surrounding the leaking of emails from the Climate Research Unit at the University of East Anglia underlines the importance of rigorous and auditable testing of scientific software. Failure to properly establish the provenance of simulation results risks undermining public confidence in science.

This paper has two purposes. First, it documents and advocates best practice in the automatic verification of scientific computer models. Second, it documents the particular system in place for Fluidity-ICOM enabling users of published Fluidity-ICOM results to understand the verification process behind that particular model.

1.1 Verification and validation

Verification and validation provide the framework for establishing the usefulness of a computational model for a particular physical situation. Verification assesses the difference between the results produced by the code and the mathematical model. Validation determines if a mathematical model represents the physical situation of interest, i.e. the ability of the model to accurately reproduce experimental data. If the computational model describes the mathematical model well, and the mathematical model relates well to the physical world, then the computational model also relates well to the physical world (Babuška and Oden, 2004). Philosophically, it is impossible to ever assert with absolute certainty that a code will accurately simulate a given physical situation of interest after verifying a finite set of tests (Popper, 1959; Howden, 1976); however, it is clear that a code which is corroborated by having passed the most stringent tests available is surely more useful than a code which has not been scrutinised at all (Babuška and Oden, 2004).

Verification divides into two parts. Code verification is the process of ensuring, to the best degree possible, that there are no coding errors affecting the implementation of the

Correspondence to: D. A. Ham
(david.ham@imperial.ac.uk)

discretisation. Code verification assesses the difference between the code and the discretised model. The other component of model verification is solution verification, assessing the difference between the discretised model and the mathematical model. Code verification deals with software engineering, while solution verification deals with a posteriori error estimation.

An important but subtle distinction between verification and validation is that validity is a property of the algorithm which the model code implements rather than a property of the code itself. To the extent to which validation tests have established the validity of an algorithm, the repetition of those tests does not further establish validity. If a model which has been validated fails to perform as expected due to a change in the code, this is a *verification error*.

Often, undergoing verification is seen as a discrete event, happening after the computational model is completed. Yet many computational models that are used for practical applications are also under active software development; it does not make sense to assert that the code has been verified, when the code has changed since the verification was performed. In principle, whenever the code is changed, verification must be applied to the new revision, for any previous results are irrelevant. We therefore advocate the view that verification must be treated as a continuous process, happening alongside both software development and usage. This is the only way to ensure that the entire modelling effort stays correct as it is developed.

This poses a difficulty. These processes are generally seen to be time-consuming, uninteresting work. Therefore, they should be automated in order to minimise the amount of manual intervention required in the scrutiny of the newly-changed code. In Knupp et al. (2007), it is implicitly argued that code verification takes too much effort to perform on the very latest version, and is thus relegated to release candidates; however, by automating the process, we have achieved great success in applying code verification to a codebase that changes daily. This philosophy is widespread in the software engineering community (Adrion et al., 1982), but is not yet the generally accepted practice among the scientific modelling community. We discuss a framework for automated continuous verification with particular suitability to numerical models. The framework is of wide applicability to any numerical model, although emphasis is placed on modelling in a geoscientific context.

1.2 Other automated verification of geoscientific models

We do not claim that this is the first invention of continuous code verification. Several large, successful geoscientific models undergo similar efforts. The MIT general circulation model (Marshall et al., 1997) has an automated system that runs a verification suite on a variety of different machines. The summary page is available at http://mitgcm.org/public/testing.html. The Unified Model developed by the UK Met Office (Davies et al., 2005) also has an automated nightly verification system, as documented in Easterbrook and Johns (2009). Other research groups have developed suites of tests suitable for use in automated verification; these researchers may well have automated the process, although the authors were unable to find any information about any such automation on their websites. The Modular Ocean Model developed at Princeton (Griffies et al., 2004) documents an extensive collection of verification test cases in their manual (Griffies, 2009). The Regional Ocean Modeling System developed at Rutgers University (Song and Haidvogel, 1994) lists a collection of test cases at https://www.myroms.org/wiki/index.php/Test_Cases. Other large, successful projects do not appear to regard verification as an ongoing process. The contract setting up the consortium to develop the NEMO ocean model (Madec et al., 1998) states that the (quote) "testing and release of new versions" happens "typically once or twice a year" (NEMO Consortium, 2008). The SLIM Ice-Ocean model developed at the Université catholique de Louvain (White et al., 2008) states that (quote) "we ran as few simplistic, highly-idealised test cases as possible. Instead, whenever possible, we tested the model against realistic flows, albeit often simple ones" (Deleersnijder et al., 2010). As realistic simulations are typically too expensive to run continuously, this suggests that no automated continuous verification system is in place.

This brief survey highlights two important points. The first is that even among large and successful modelling efforts, the practice of automated, continuous verification is far from universal. Note that the projects examined all have large development teams and may have professional IT support. This is very atypical of geoscientific modelling in general: the more usual case is of in-house development in a small research group with code passed informally from one generation of PhD students and post-docs to the next.

The second point is that even the large, well-resourced projects do not typically publish their code verification practices in the formal, peer-reviewed literature. As noted above, the authors feel that it is important in maintaining public confidence in simulation results that the provenance, including verification processes, of those results is well established. It is therefore to be hoped that the developers of other models will follow this lead and publish their verification processes.

1.3 Fluidity-ICOM

Although the framework presented here is applicable to any numerical model, it is useful to present particular examples. For this we will use the example of Fluidity-ICOM, the primary software package to which this particular framework has been applied. Fluidity is a finite element, adaptive mesh fluid dynamics simulation package. It is capable of solving the full Navier-Stokes equations in compressible or incompressible form, or the Boussinesq equations. It is equipped with numerous parameterisations for sub-gridscale processes and has embedded models for phenomena as varied as ocean biology, traffic pollution, radiation transport and porous media. Fluidity supports a number of different finite element discretisations and is applied to flow problems in a number of scientific and engineering domains including, in particular, ocean flows. In this last context it is known as the Imperial College Ocean Model (Fluidity-ICOM) and this is the name we will use here. For particular information on Fluidity-ICOM as an ocean model, the reader is referred to Piggott et al. (2008).

While verification is essential for all scientific computer software, the complexity and wide applicability of the Fluidity-ICOM makes this a particularly challenging and critical requirement. The model is developed by several dozen scientists at a number of different institutions and there are approximately 15 commits (changes to the model) on an average work day. In the absence of constant verification, development would be impossible.

2 Verification tests

There are two general strategies to inspect the source code of a computational model. Static analysis involves (usually automated) inspection of the source code itself for issues such as uninitialised variables, mismatched interfaces and off-by-one array indexing errors. Dynamic analysis involves running the software and comparing the output (some functional of the solution variables) to an expected output. The source of the expected output determines the rigour and purpose of the test. Various sources are possible:

- The simplest and least rigorous is to compare the output to previous output; this tests code stability, not code correctness, but can still be useful (Oberkampf and Atrucano, 2002).

- The expected output could be output previously produced by the code that has been examined by an expert in the field. While flawed, asserting the plausibility of the results is better than nothing at all. The test in this case can again be that the output has not changed from previous runs, except where expected.

- The expected output could come from a high-resolution simulation from another verified computational model of the same discretisation. However, analytical solutions are to be preferred as they remove the possibility of common algorithmic error among the implementations.

- The expected output could come from an analytical solution. The test in this case could be the quantification of error in the simulation or numerically computing the rate of convergence to the true solution as some discretisation parameter (h, Δt, ...) tends to 0. Comparing the obtained rate of convergence against the theoretically predicted rate of convergence is generally considered the most powerful test available for ensuring that the discretised model is implemented correctly, as it is very sensitive to coding errors (Roache, 2002; Knupp et al., 2007).

- The analytical solution could come from the method of manufactured solutions (Salari and Knupp, 2000; Roache, 2002). This method involves adding in extra source terms to the governing equations being solved in order to engineer an equation whose solution is known. It is a general and powerful technique for generating analytical solutions for use in error quantification or convergence analyses.

- Once the code verification tests have completed, solution verification for a library of simulations may take place. The functional could be some a posteriori estimate of the discretisation error, and the expected output that it is below a given tolerance. Formally, this solution verification step is only necessary when the discretisation has changed.

- The final source of verification is to re-run previous validation tests. For this purpose, the expected output is derived from a physical experiment. Again, the test could assert that the rate of convergence to the physical result is the same as theoretically predicted. Comparing output to experimental data asserts the applicability of both the computational and mathematical models to the physical world. It is to be emphasised that model verification should happen before model validation, for otherwise the error introduced in the mathematical modelling cannot be distinguished from discretisation or coding errors (Babuška and Oden, 2004). However once a validation test has been passed, the repetition of that test can be used to verify the model after subsequent code changes.

In general, a *test case* is a set of input files, a set of commands to be run on those input files, some functionals of the output of these commands to be computed, and some comparisons against independent data of those functionals. While the purpose and level of rigour of the test changes with the source of the comparison data, this is irrelevant for the execution of the test itself. Indeed, the generality of this view is a great benefit to the design of the framework: code stability tests, code verification and solution verification can all be performed by the same system. Note that this conception of a test case encompasses both static and dynamic analysis: in static analysis, the command to be run is the analysis tool; in dynamic analysis, the command to be run is the model itself.

2.1 Verification as a continuous process

In reality, most computational models are both in production use by end-users and undergoing continual development. Even if the verification procedure has been passed for a previous revision, it does not necessarily mean that the next revision of the computational model will also pass. New development can and does introduce new coding errors. Therefore, verification must be seen as a continuous process: it must happen alongside the software development and deployment. Integrating this process alongside software development greatly eases the burden of deploying stable, working releases to end users.

Regarding verification as a continuous process has many benefits for the parallel process of software development. As new features are added, feedback is immediately available about the impact of these changes to the accuracy of the computational model; since coding errors are detected soon after they are introduced, they are easier to fix as the programmer is still familiar with the newly introduced code. Furthermore, as software development teams become large, it can be difficult to predict the impact of a change to one subroutine to other users of the model; if those other uses of the software are exercised as part of the continuous verification process then unintended side-effects can be detected early and fixed. Since the code is run continuously to check for correctness, other metrics can be obtained at the same time: for example, profiling information can be collected to detect any efficiency changes in the implementation.

2.2 Verification should be automated

Verifying every change to a code base as part of a continuous code verification procedure is laborious and repetitive. It is a therefore natural candidate for automation.

Automating the process of code verification means that checking for correctness can be performed simultaneously on multiple platforms with multiple compilers, platforms to which an individual developer may not have access. It also means that more tests can be run than would be practical for a single human to run; these tests can therefore check more code paths through the software, and can be more pedantic and time-consuming than a human would tolerate. In practice, without automation, the amount of continuous code verification is limited to the problems of immediate interest to the currently active development projects.

2.3 The limitations of testing

It has been noted elsewhere Oreskes et al. (1994) that complex geoscientific models such as GCMs may be formally unverifiable simply because they do not constitute closed mathematical systems. Aspects of these models, particularly parameterisations, may be difficult to formulate analytic solutions for and may not, in fact, converge under mesh refinement. Nonetheless, individual components of models considered in isolation, for example the dynamic core or an individual parameterisation, must have well-defined and testable mathematical behaviour if there is to be any confidence in the model output at all. If there is to be any confidence in the output of a formally unverifiable model, it is surely a necessary condition that each verifiable component passes verification. The methodology explained here is therefore useful in at least this context. The automated verification of code stability (i.e. that the model result does not unexpectedly change) must also be regarded as a key tool in the verification of the most complex models.

3 A framework for automated continuous verification

This section discusses the technical details of the automated continuous verification procedures. The workflow described here is illustrated in Fig. 1 and the individual steps are described in more detail below.

3.1 Commit to source

A canonical copy of the source code is kept in a source code control system. A source code control system is a suite of software for managing the software development process. Developers check out a copy of the source code, make changes locally, and commit them back to the source repository. The source code control system merges changes in the case where another developer commits a change between a checkout and a commit. Source code control systems such as Subversion (Collins-Sussman et al., 2004) are an essential component of modern software development.

When a developer commits to the source code repository, that means that the code has changed, and thus any previous verification is irrelevant to the new version. The source code control system emits a signal to the test framework, notifying that the source has changed and that the code verification process should begin.

3.2 Automated build

The automated verification procedure is managed by buildbot[1]. Buildbot is a software package for the automation of software test cycles. It is designed on the client-server architecture: each machine that builds and tests the newly changed software runs a buildslave client, while the overall process is managed by a buildmaster server. It is the buildmaster that is notified by the source code control system.

When the buildmaster is notified of a software change, it instructs the buildslaves to execute the steps defined in the buildbot configuration. The buildslave updates the copies of the source code it holds, and compiles the source with the compilers specified in the configuration. Any errors in the

[1]http://buildbot.net

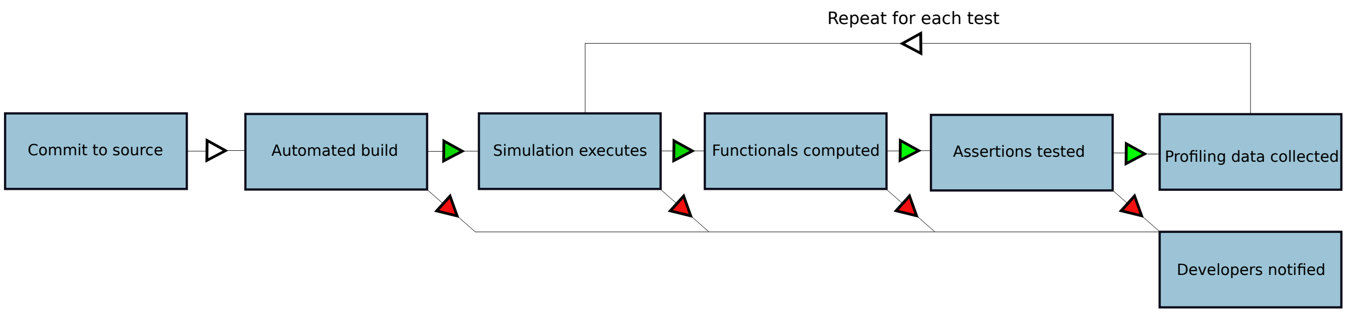

Fig. 1. The workflow for the continuous testing procedure. The test procedure is repeated for each test and the manner of any failure of that test is reported to the developers.

build process halt the code verification process immediately and are reported via email to the developers, who can examine the output of the build process to inspect the error messages. Currently, the Fluidity-ICOM project (Piggott et al., 2008) compiles 32- and 64-bit versions of each revision, in single and double precision and in various build configurations, with GCC and the Intel Compilers.

Assuming the software builds successfully, the test cycle begins. There are two types of tests considered here, unit tests and test cases.

3.3 Unit tests

A unit test operates at the level of an individual unit of source code, for example a subroutine in a procedural language or a class method in an object-oriented language. A unit test passes input to a unit of code and makes assertions about its output. Unit testing is a very powerful and useful technique for programming, as it allows the programmer to write down in an executable manner what is expected of a unit. Examples of unit tests include asserting that the derivative of a constant is zero, asserting that the eigendecomposition of a specified input matrix is correct, or asserting that the residual of a linear solve is less than the specified tolerance. Regular unit testing of individual pieces of code allows them to be relied upon in other, more complex algorithms. This also allows computationally expensive debugging tools such as valgrind[2] and ElectricFence[3] to be applied to individual components rather than to the whole code at once.

With the increasing trend towards common components in software, the possibility of code changes in third party libraries introducing subtle bugs also increases. Unit testing is an excellent way to guard against this possibility, as it defines precisely what the software expects of the libraries it depends upon.

[2]http://valgrind.org

[3]http://perens.com/works/software/

3.4 Test cases

Test cases operate at the level of the entire computational model. The buildbot invokes the test harness, a piece of software which manages the execution of the test case. A test case typically runs the newly built revision on a given simulation and makes assertions comparing the output to some external data source. The purpose and level of rigour of the test, and thus its usefulness, is determined by the reliability of the source of the external data. One advantage of this framework is that different forms of verification may be performed automatically by the same means: code stability tests make assertions against output data obtained from previous runs of the model, code verification tests make assertions against analytical solutions (possibly obtained by the method of manufactured solutions), while solution verification makes assertions about discretisation errors using data from analytic solutions, other models or physical data. Seen abstractly, a test case consists of four things: some input files, a set of commands to be executed, functionals to be computed from the output of those commands, and assertions to be made about the result of those functionals. In order for the test case to be automatable, the test must be completely described in a machine-parsable way: in this framework, a test case is described by an XML document, with functionals of the output and the assertions described in Python code fragments embedded in the XML file. An example XML file is given in Fig. 2.

Each test problem is assigned a name, which is used for printing out status messages. The `<problem_definition>` tag gives information about the expected length of the problem, which is used by the test harness for scheduling decisions, and the number of processors on which the problem is designed to run. The `<command_line>` tag contains the commands to be executed; typically this will be the commands to run the software. Other possible commands might be to run a static analysis tool on the source tree or to run a post-processing tool on software output.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE testproblem SYSTEM "testcase.dtd">

<testproblem>
    <name>Example test</name>
    <problem_definition length="medium" nprocs="1">
        <command_line>run_model example</command_line>
    </problem_definition>
    <variables>
        <variable name="max_val" language="python">
import model_tools output =
model_tools.parse_output("example.out") max_val =
output['max_val']
        </variable>
    </variables>
    <pass_tests>
        <test name="Maximum value equals one" language="python">
assert max_val == 1
        </test>
    </pass_tests>
    <warn_tests>
    </warn_tests>
</testproblem>
```

Fig. 2. Example test case, as described in Sect. 3.4. The test is described by an XML file. The command to be executed is recorded in the `command_line` tag. Functionals of the output to be computed are recorded in `variable` tags. Assertions to be made about the values of the functionals are recorded in `test` tags.

Once the commands to be executed have completed, functionals of the output are computed. A functional is computed by a fragment of Python code in a <`variable`> tag. Making use of a widely available general-purpose scripting language such as Python gives great power and flexibility to the test author. Reference results can be retrieved from a relational database on a networked server, or the computed functional value stored in a database for future reference. Powerful scientific libraries such as SciPy (Jones et al., 2001), VTK (Schroeder et al., 2006) and SAGE (Stein et al., 2007) are made available to the programmer. These can be used for the computation of diagnostics, linear regressions, statistical analyses, image and signal processing, etc.

Once the functionals are computed, the test assertions are made. These also take the form of Python fragments, and typically are composed of one or more `assert` statements. The test is deemed to fail if the code raises a Python exception indicating that an error has occurred (in Python, a failed `assert` statement raises an `AssertionError` exception). If no exception is raised, the test is deemed to have passed. If an individual test is defined in a <`pass_test`> tag, its failure causes the entire test to fail; if the test is defined in a <`warn_test`> tag, a warning is issued. Warning tests are typically used in code stability tests to notify that the results have changed, without necessarily implying that the change is due to an error.

The test harness can integrate with a cluster management system and push the execution of test cases out to a batch queue. In the Fluidity-ICOM test suite, longer test cases are executed in parallel on a dedicated test cluster, while shorter test cases are automatically executed on dedicated workstations.

3.5 Automated profiling

The binaries compiled in Sect. 3.2 are built with the compiler flags necessary to turn on the collection of `gprof` profiling data (Graham et al., 1982). The compiler augments the code with routines to profile the execution of the code. If the test case passes, the profiling data is processed and stored for future reference. This enables developers to inspect the efficiency of each subroutine or class method of the code as a function of time, and correlate any changes in efficiency with code modifications.

The automated collection of other code quality metrics such as cache misses or the presence of memory leaks could also be performed as part of this framework.

4 Fluid dynamics test cases

In this section a selection of the test cases for use in the verification of a computational and geophysical fluid dynamics code are described. To begin, a number of tests which are suitable for use with a standard fluid dynamics model are described; this is followed by a number of tests suitable for models which incorporate buoyancy and Coriolis effects, for example geophysical fluid dynamics codes and ocean models. Note that for the problems presented here the model has been set up so as to optimise the efficiency of the test, i.e. to give rigorous checks on the code in minimal computational time, and not necessarily to optimise the accuracy of the overall calculation or of the particular metric being used.

The test cases presented here are selected to illustrate a range of problem formulations and test statistics. It is not intended to be a comprehensive list of the tests required of a particular class of model: the actual test suite employed by Fluidity-ICOM, for example, contains well in excess of three hundred tests. The first two tests shown here employ the method of manufactured solutions to create new analytic solutions as test comparators. The lid-driven cavity and lock exchange tests exemplify the use of the results of other models run at high resolution as a benchmark while Stommel's western boundary current is an example of a well-known analytic result used as a test case.

The model being tested here uses finite element discretisation methods on tetrahedral or hexahedral elements in three dimensions and triangular or quadrilateral elements in two dimensions. The underlying equations considered in the tests presented here include the advection-diffusion of scalar fields, the Navier-Stokes equations, and the Boussinesq equations with buoyancy and Coriolis terms included. The model has the ability to adapt the mesh dynamically in response to evolving solution fields. For background to the model see Pain et al. (2005); Piggott et al. (2008). For an overview of CFD validation and verification, see Oberkampf et al. (1998); Stern et al. (2001).

4.1 Computational fluid dynamics examples

4.1.1 The method of manufactured solutions: tracer advection

To test the implementation of the advection-diffusion and Navier-Stokes equations spatial convergence tests are performed using the method of manufactured solutions (MMS, Roache, 2002). MMS provides an easy way of generating analytical solutions against which to verify model code. A sufficiently smooth desired analytical solution is designed and a suitable source term added to the right hand side to ensure the validity of the equation. The source is calculated by substituting the desired analytical solution in the underlying differential equation.

The numerical equation is then solved on a sequence of successively finer meshes. The solution on each of these is then compared to the known exact solution and the order of convergence compared to the expected order for that method. When convergence of the solution is not seen, it is an excellent indicator of an error in the model code or the numerical formulation. MMS has been shown to be highly effective at finding such problems (Salari and Knupp, 2000) and continuous monitoring of the results through an automated system allows errors that affect the order of convergence to be immediately noticed.

To test tracer advection-diffusion the desired analytical solution is taken as:

$$T(x,y,t) = \sin(25xy) - 2y/x^{1/2}, \tag{1}$$

while a prescribed velocity field, $\boldsymbol{u} = (u,v)$, is given by

$$u = \sin\Big(5(x^2+y^2)\Big), \quad v = \cos\Big(3(x^2-y^2)\Big). \tag{2}$$

The source term, S, is calculated symbolically using SAGE (Stein et al., 2007) by substituting T and $\boldsymbol{u}$ into the advection-diffusion equation:

$$\begin{aligned} S &= \frac{\partial T}{\partial t} + \boldsymbol{u}\cdot\nabla T - \kappa\nabla^2 T, \\ &= \Big(25y\cos(25xy) + y/x^{3/2}\Big)\sin\Big(5(y^2+x^2)\Big) \\ &\quad + \Big(25x\cos(25xy) - 2/x^{1/2}\Big)\cos\Big(3(x^2-y^2)\Big) \\ &\quad + \kappa\Big(625(x^2+y^2)\sin(25xy) + 3y/(2x^{5/2})\Big). \end{aligned}$$

The computational domain is $0.1 \le x \le 0.6$; $-0.3 \le y \le 0.1$ and is tessellated with a uniform unstructured Delaunay mesh of triangles with characteristic mesh spacing of h in the x and y directions. The analytical solution (Eq. 1) is used to define Dirichlet boundary conditions along the inflowing lower and left boundaries while its derivative is used to define Neumann boundary conditions on the remaining sides. Both the boundary conditions and source term are defined through Python functions defined in the Fluidity-ICOM preprocessor (Ham et al., 2009), where the diffusivity, κ, is taken as 0.7.

As we are performing a spatial convergence test, the desired solution is temporally invariant. However, the equation contains a time derivative and requires an initial condition. This is set to zero everywhere leading to a numerical solution that varies through time. The simulation is terminated once this reaches a steady state (to a tolerance of 10^{-10} in the infinity norm).

Once a steady state has been obtained on all meshes the convergence analysis may be performed. Given the error, E, on two meshes, with characteristic mesh spacing h_1 and h_2 for example:

$$E_{h_1} \approx C h_1^{c_p}, \tag{3}$$

$$E_{h_2} \approx C\left(\frac{h_1}{r}\right)^{c_p}, \tag{4}$$

where C is a constant discretisation specific factor independent of the mesh, c_p is the order of convergence of the method and r is the refinement ratio ($r = 2$ in this case), then the ratio of errors is given by:

$$\frac{E_{h_1}}{E_{h_2}} \approx \left(\frac{C h_1^{c_p}}{C h_1^{c_p}}\right) r^{c_p} = r^{c_p}, \tag{5}$$

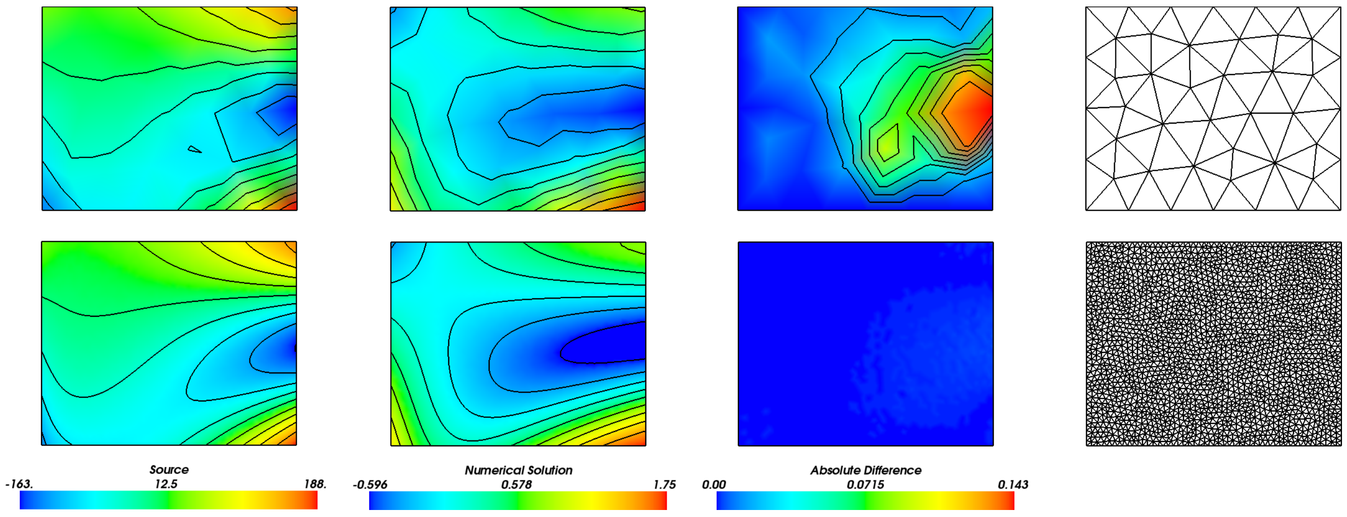

Fig. 3. From left to right: source term (Eq. 3) for the method of manufactured solutions advection-diffusion test case, the numerical solution calculated using a piecewise-linear Galerkin discretisation, the absolute difference between the analytical and numerical solutions, the meshes used to compute the previous images with average mesh spacings, h, of 0.08 (top) and 0.01 (bottom).

and the order of convergence can be calculated as:

$$c_p \approx \log_r \left(\frac{E_{h_1}}{E_{h_2}} \right). \tag{6}$$

This can then be compared to the expected order of convergence for a particular method.

Several model configurations and discretisations are used in the full testing suite. Here, the results of a first-order upwinding control volume (CV) discretisation and a second-order piecewise-linear Galerkin (P1) discretisation are presented. In both cases the Crank-Nicolson method is used to discretise in time. Table 1 demonstrates that the expected order of spatial convergence, or better, is achieved for both discretisations. Figure 3 shows the source term, the numerical solution using the P1 discretisation, the absolute difference between this and the analytical solution at steady state and meshes with average nodal spacings, h, of 0.08 and 0.01.

4.1.2 The method of manufactured solutions: Navier-Stokes equations

The method of manufactured solutions can also be used to test more complicated sets of equations involving multiple coupled prognostic fields, such as the Navier-Stokes equations. Initially an incompressible, smooth and divergence free desired velocity field, $\boldsymbol{u} = (u, v)$ is considered:

$$u = \sin(x)\cos(y), \quad v = -\cos(x)\sin(y), \tag{7}$$

along with a desired pressure, p:

$$p = \cos(x)\cos(y). \tag{8}$$

These are substituted into the momentum equations, with tensor-form viscosity, using SAGE (Stein et al., 2007) to derive the required momentum source:

$$\begin{aligned}\boldsymbol{S} &= \rho \frac{\partial \boldsymbol{u}}{\partial t} + \rho \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \mu \nabla^2 \boldsymbol{u} + \nabla p \\ &= \begin{pmatrix} \rho(\cos(x)\sin(x)\sin^2(y) + \cos(x)\sin(x)\cos^2(y)) \\ \quad +2\mu\sin(x)\cos(y) - \sin(x)\cos(y) \\ \rho(\cos(y)\sin(y)\sin^2(x) + \cos(y)\sin(y)\cos^2(x)) \\ \quad -2\mu\cos(x)\sin(y) - \cos(x)\sin(y) \end{pmatrix}\end{aligned}$$

The incompressible Navier-Stokes equations are then solved in the computational domain $0 \le x \le \pi$; $0 \le y \le \pi$ tessellated using an unstructured Delaunay mesh of triangles. Velocity is discretised using a piecewise-quadratic Galerkin discretisation while pressure uses piecewise-linear elements (the Taylor-Hood element pair). Strong Dirichlet boundary conditions for velocity are provided on all sides of the domain using the desired solution while pressure has natural homogeneous Neumann boundary conditions enforced. Both the strong boundary conditions and source term are defined through Python functions defined in the Fluidity-ICOM preprocessor (Ham et al., 2009), while the density, ρ, and viscosity, μ, are taken as 1.0 and 0.7 respectively.

Table 2 presents the convergence results for velocity and pressure on a series of unstructured meshes with successively smaller average mesh spacings. For both velocity and pressure the expected order of convergence, or better, is observed.

Further variables may be introduced by considering the fully compressible Navier-Stokes equations with a divergent desired velocity field:

Table 1. Spatial order of convergence results for the method of manufactured solutions advection-diffusion test case described in Sect. 4.1.1. The difference between the analytical and numerical solutions using a first-order control volume (CV) discretisation and a second-order piecewise-linear Galerkin (P1) discretisation are calculated in the L_2 norm. The ratio between these on two spatial mesh resolutions, h_1 and h_2, are used to estimate the order of spatial convergence of the model for this problem. The expected order of convergence, or better, is observed for both spatial discretisations.

$h_1 \to h_2$	$0.08 \to 0.04$	$0.04 \to 0.02$	$0.02 \to 0.01$	$0.01 \to 0.005$
c_p (CV)	2.42	2.00	1.43	0.97
c_p (P1)	2.03	1.91	2.08	2.12

Table 2. Spatial order of convergence results for the method of manufactured solutions incompressible Navier-Stokes test case described in Sect. 4.1.2. The difference between the analytical and numerical solutions using a piecewise-quadratic velocity, (u,v), and a piecewise-linear pressure, p, Galerkin discretisation are calculated in the L_2 norm. The ratio between these on two spatial mesh resolutions, h_1 and h_2, are used to estimate the order of spatial convergence of the model for this problem. The expected order of convergence is observed for all variables.

$h_1 \to h_2$	$0.32 \to 0.16$	$0.16 \to 0.08$	$0.08 \to 0.04$
$c_p\ (u)$	3.18	3.03	2.96
$c_p\ (v)$	3.04	2.01	3.04
$c_p\ (p)$	2.27	2.01	1.98

$$u = \sin(x^2+y^2)+1/2, \quad v = \left(\cos(x^2+y^2)+1/2\right)/10, \tag{9}$$

and a spatially varying density field, ρ:

$$\rho = \left(\sin(x^2+y^2)+3/2\right)/2. \tag{10}$$

Assuming, a desired internal energy, e:

$$e = (\cos(x+y)+3/2)/2 \tag{11}$$

it is then possible to define the desired pressure field using a stiffened gas equation of state:

$$p = c_B^2(\rho-\rho_0)+(\gamma-1)\rho e. \tag{12}$$

In this case, coupled momentum, continuity and internal energy equations are solved, each of which require a source term, $\boldsymbol{S_u}$, S_ρ and S_e respectively, to be calculated:

$$\boldsymbol{S_u} = \rho\frac{\partial \boldsymbol{u}}{\partial t}+\rho\boldsymbol{u}\cdot\nabla\boldsymbol{u}-\nabla\cdot\boldsymbol{\tau}+\nabla p, \tag{13}$$

$$S_\rho = \frac{\partial\rho}{\partial t}+\nabla\cdot(\boldsymbol{u}\rho), \tag{14}$$

$$S_\mathrm{e} = \frac{\partial(\rho e)}{\partial t}+\boldsymbol{u}\cdot\nabla e+p\nabla\cdot\boldsymbol{u}, \tag{15}$$

where the deviatoric stress tensor, τ, is linearly related by the viscosity, μ, to the strain-rate tensor, ϵ. The derivation of these sources is omitted here for clarity but as with previous MMS test cases they are easily found using a symbolic mathematics toolkit (e.g. SAGE, Stein et al., 2007).

Table 3. Spatial order of convergence results for the method of manufactured solutions compressible Navier-Stokes test case described in Sect. 4.1.2. The difference between the analytical and numerical solutions using a piecewise-quadratic velocity, (u,v), a piecewise-linear pressure, p, a piecewise-quadratic density, ρ, and a piecewise-linear internal energy, e, Galerkin discretisation are calculated in the L_2 norm. The ratio between these on two spatial mesh resolutions, h_1 and h_2, are used to estimate the order of spatial convergence of the model for this problem. The expected order of convergence is observed for all variables.

$h_1 \to h_2$	$0.1 \to 0.05$	$0.05 \to 0.025$
$c_p\ (u)$	2.45	2.25
$c_p\ (v)$	2.07	2.07
$c_p\ (p)$	2.24	2.15
$c_p\ (\rho)$	2.43	2.15
$c_p\ (e)$	2.14	2.08

The problem is considered in the computational domain $-0.1 \le x \le 0.7$; $0.2 \le y \le 0.8$, which is tessellated using an unstructured mesh of triangles with successively smaller average mesh lengths. As before, a Galerkin discretisation is used for velocity (piecewise-quadratic elements) and pressure (piecewise-linear elements), while a streamline upwind Petrov-Galerkin (SUPG) discretisation is used for the internal energy (piecewise-linear) and density (piecewise-quadratic). The desired velocity is imposed via strong Dirichlet boundary conditions on all sides of the domain while the other variables are prescribed on the lower and left inflowing boundaries. All the sources and boundary conditions are input using Python functions in the Fluidity-ICOM preprocessor, taking the square of the speed of sound, c_B^2, the reference density, ρ_0, the specific heat ratio, γ, and the viscosity, μ, as 0.4, 0.1, 1.4 and 0.7 respectively.

Table 3 presents the order of spatial convergence for all the prognostic variables in the compressible Navier-Stokes test case, all of which demonstrate the expected order of convergence.

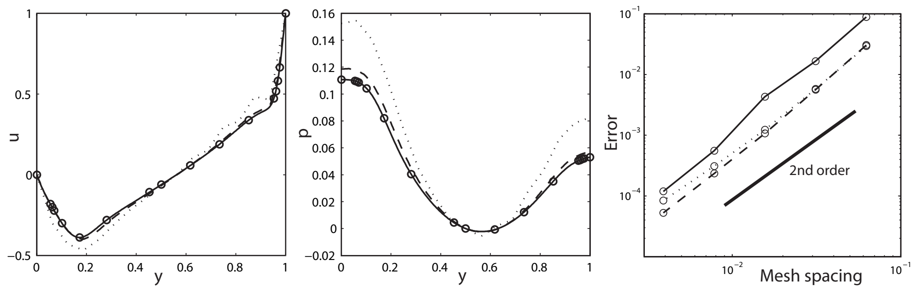

Fig. 4. Left and centre: Numerical approximations to u and p at three resolutions: $\Delta x = 1/16$ (dotted line), $1/32$ (dashed line) and $1/256$ (solid line). The values from Botella and Peyret (1998) are plotted as circles. Right: The root-mean-square errors between the numerical solution and the data from Botella and Peyret (1998) for u (solid line) and p (dashed line), and the absolute value of the difference between the numerical and benchmark kinetic energy (dotted line) taken from Bruneau and Saad (2006). A line indicating second order convergence is also shown.

The method of manufactured solutions is an extremely versatile code verification tool. As well as increasing the complication of the problems considered, as done above, equations may also be simplified. By considering the terms in equation sets individually within an automated testing platform any new coding error introduced may be pinpointed almost instantaneously, even within a large code base. This has motivated the development of over forty MMS test cases in the Fluidity-ICOM verification suite, all of which assert that the expected order of convergence is maintained after each revision of the code.

4.1.3 The lid-driven cavity

The lid-driven cavity is a problem that is often used as part of the verification procedure for CFD codes. The geometry and boundary conditions are simple to prescribe and in two dimensions there are a number of highly accurate numerical benchmark solutions available for a wide range of Reynolds numbers (Botella and Peyret, 1998; Bruneau and Saad, 2006; Erturk et al., 2005). Here the two-dimensional problem at a Reynolds number of 1000 is given as an example.

The unsteady momentum equations with nonlinear advection and viscosity terms are solved in a unit square in the x and y directions along with the continuity equation, which enforces incompressibility. No-slip velocity boundary conditions are imposed on boundaries $x = 0, 1$ and $y = 0$, and the prescribed velocity $u = 1$, $v = 0$ are set on the boundary $y = 1$ (the "lid"). The problem is initialised with a zero velocity field and the solution allowed to converge to steady state via time-stepping. A subset of the benchmark data available from the literature is then used to test for numerical convergence. Here this involves the calculation of the kinetic energy

$$\int_{\Omega} (u^2 + v^2)\, d\Omega, \tag{16}$$

which is compared against the value 0.044503 taken from Bruneau and Saad (2006). In addition, the x-component of velocity and pressure are evaluated at 17 points along the line $x = 0.5$ and compared against the data from Botella and Peyret (1998).

Plots of the solutions and benchmark data are given in Fig. 4. Also shown is a plot of the error convergence with mesh spacing. A regular triangular mesh is used with progressive uniform refinement in the x, y plane. Second order spatial convergence can clearly be seen for the three quantities compared.

The automated assertions in this case are that second order convergence is attained and that the magnitude of errors in the three quantities does not increase with code updates.

4.1.4 Flow past a sphere: drag calculation

In this test, uniform flow past an isolated sphere is simulated and the drag on the sphere is calculated and compared to a curve optimised to fit a large amount of experimental data.

The sphere is of unit diameter centred at the origin. The entire domain is the cuboid defined by $-10 \le x \le 20$, $-10 \le y \le 10$, $-10 \le z \le 10$. The unsteady momentum equations with nonlinear advection and viscous terms along with the incompressibility constraint are solved. Free slip velocity boundary conditions are applied at the four lateral boundaries, $u = 1$ is applied at the inflow boundary $x = -10$, and a free stress boundary condition applied to the outflow at $x = 20$. A series of Reynolds numbers in the range $Re \in [1, 1000]$ are considered. The problem is run for a long enough period that the low Reynolds number simulations reach steady state, and the higher Reynolds number runs long enough that a wake develops behind the sphere and boundary layers on the sphere are formed. This is deemed sufficient for the purposes of this test which is not an in-depth investigation of the physics of this problem, nor an investigation of the optimal set of numerical options to use. Here an unstructured tetrahedral mesh is used along with an adaptive remeshing algorithm (Pain et al., 2001). Figure 5 shows a snapshot of the mesh and velocity vectors taken from a Reynolds number

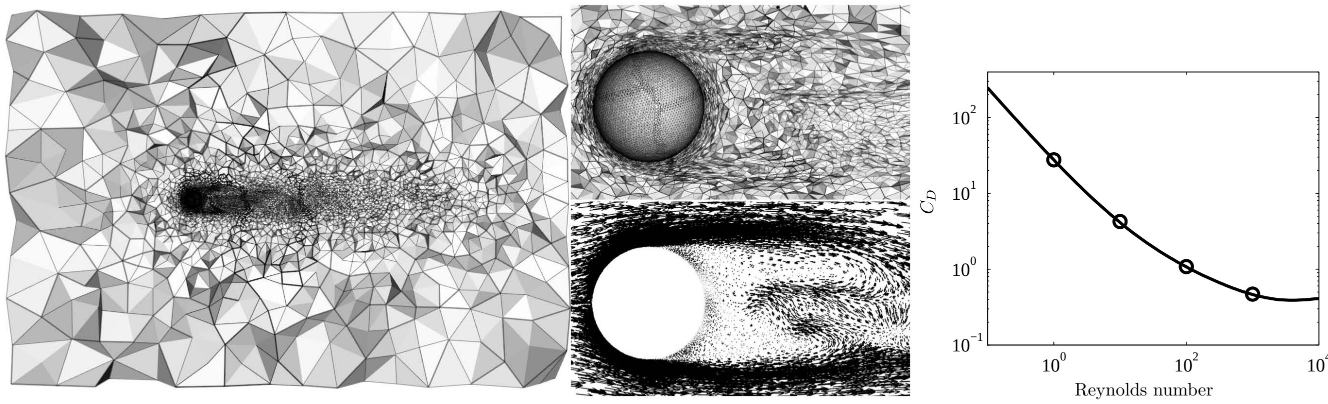

Fig. 5. Left: unstructured adapted mesh from flow past a sphere simulation at $Re = 103$, with half the domain cut away to display refinement close to the sphere and in its wake. Centre: a blow up of the mesh and velocity vectors on a plane through the centre of the domain. Right: comparisons between the computed drag coefficient (circles) and the correlation (solid line) given by expression (18) in the range $Re \in [1, 1000]$.

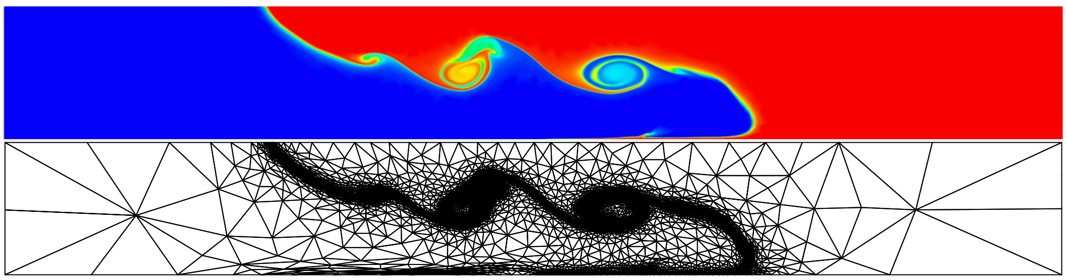

Fig. 6. Top: the temperature field for the lock exchange problem at time 14.2 s. It is apparent that diagnosing the head location from density contours is straightforward. Bottom: the adapted mesh at this time level with enhanced resolution being used to minimise numerical dissipation and maintain a sharp interface between the two density classes in this problem.

1000 simulation. The mesh can be seen to be resolving the wake and the boundary layers on the sphere with enhanced anisotropic resolution. At higher Reynolds numbers the dynamics become more complex and if a full numerical study was being conducted here more care would be taken is the choice of adaptive remeshing parameters and the use of averaged values from simulations allowed to run for longer periods. The drag coefficient is calculated from

$$C_D = \frac{F_x}{\frac{1}{2}\rho u_0^2 A}, \qquad F_x = \int_S (n_x p - n_i \tau_{ix})\, dS, \tag{17}$$

where ρ is the density, taken here to be unity; u_0 is the inflow velocity, here unity; and A is the cross-sectional area of the sphere, here $\pi^2/4$. F_x is the force exerted on the sphere in the free stream direction; S signifies the surface of the sphere; n is the unit outward pointing normal to the sphere (n_x is the x-component and n_i the ith component, here summation over repeated indices is assumed); p is the pressure and τ is the stress tensor; see Panton (1996).

Figure 5 also shows a comparison between the computed drag coefficient with a correlation (to a large amount of laboratory data) taken from Brown and Lawler (2003):

$$C_D = \frac{24}{Re}\left(1 + 0.15 Re^{0.681}\right) + \frac{0.407}{1 + \frac{8710}{Re}}. \tag{18}$$

The assertions tested are that the difference between the computed drag coefficient and values from the correlation (18) at a number of Reynolds numbers are within acceptable bounds. Checks on the number of nodes produced by the adaptive algorithm for given error measure choice and other options are also conducted. While all of these simulations can be run comfortably in serial, the Reynolds number 100 and 1000 cases are performed on 8 cores both to accelerate the tests and as a test of the parallel implementation.

4.2 Geophysical fluid dynamics examples

In this section some of the test cases used for the model in its "oceanographic mode" (i.e. with the incorporation of

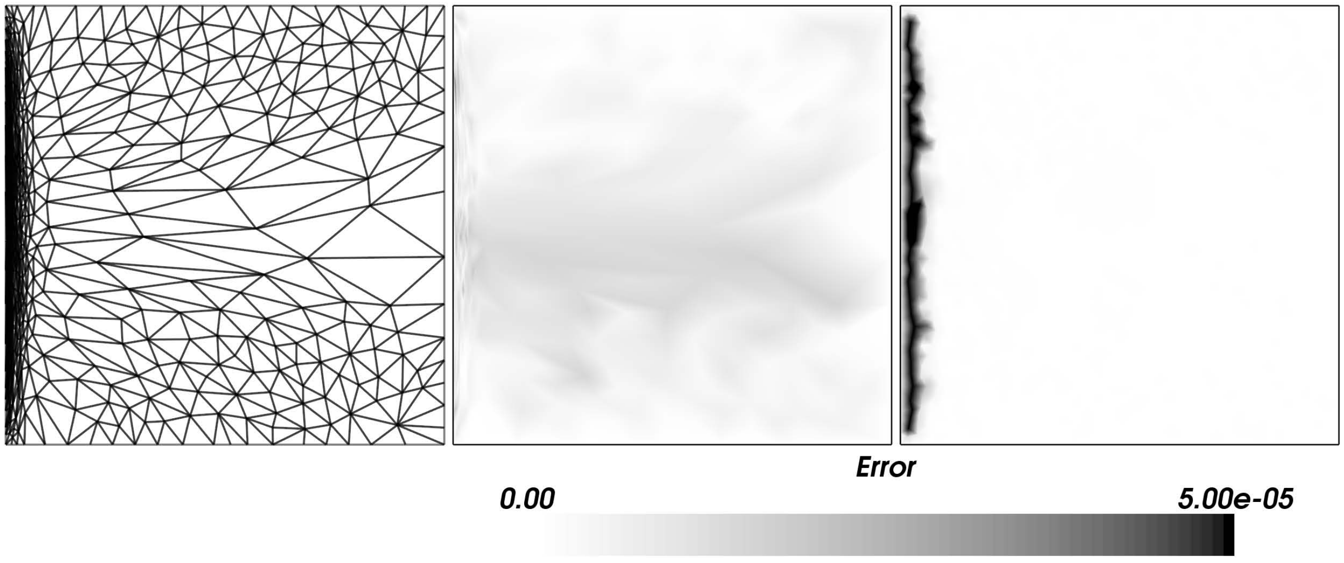

Fig. 7. Left: the adapted mesh in the Stommel western boundary current test. The mesh can be seen to be refined anisotropically in the vicinity of the western boundary. Centre and right: the absolute values of the difference between the numerical and analytical solutions in the case of the adapted (centre) and uniform (right) meshes. The uniform mesh has approximately four times the number of nodes compared to the adapted mesh. Large errors can be seen with the uniform mesh which has insufficient resolution to resolve the rapidly changing streamfunction close to the western boundary.

buoyancy and Coriolis effects) are also presented. Further useful test problems can be found in Haidvogel and Beckmann (1999); Williamson et al. (1992); Ford et al. (2004); Giraldo and Restelli (2008).

4.2.1 Lock exchange

In this problem an advection-diffusion equation for density and the Boussinesq equations for velocity and pressure are used to solve for the evolution of a system where fluid of two densities are initialised next to one another in a rectangular tank in the (x,z) plane. The dense water slumps under gravity and moves under the lighter fluid. A Kelvin-Helmholtz instability causes the generation of overturning billows at the interface between the two densities which contributes to the eventual mixing of the water column (Simpson, 1999). Here a no-slip velocity boundary condition is used at the bottom of the domain with free-slip at the top. The domain is defined by $0 \le x \le 0.8$, $0 \le z \le 0.1$. A constant time step of 0.025 is used and the mesh is adapted every 10 time steps. A full description of the physical parameters used to set-up this problem are given in Fringer et al. (2006); Härtel et al. (2000), and a comprehensive study of this scenario in Fluidity-ICOM is given in Hiester et al. (2011). Here a case with Grashof number of 1.25×10^6 has been used.

The speed of the gravity current head in the horizontal are derived at the upper and lower boundaries by first extracting the maximum and minimum x values of an isosurface of the density field, and then computing the linear growth of each with time after an initial relaxation time, Fig. 6. These values are then compared with the values quoted in Härtel et al. (2000); Fringer et al. (2006), namely -0.012835 for the no-slip boundary and 0.015093 for the free-slip boundary. Härtel et al. (2000) use direct numerical simulation (DNS) to study this problems and hence these metrics of the flow dynamics for this problem are considered as truth.

The automated assertions tested are that the head speeds, computed from a time series of model output via Python script, agree with the DNS values to within an allowed tolerance. Checks on the number of nodes used in the calculation are also performed.

4.2.2 Stommel's western boundary current

This test involves the steady state wind driven barotropic circulation in a rectangular domain, and compares against an analytical solution.

Stommel (1948) was the first to describe why one observes the intensification of boundary currents on the western side of ocean basins, for example the Gulf Stream in the North Atlantic. The streamfunction equation in the domain $0 \le x \le 1$, $0 \le y \le 1$,

$$\nabla^2\Psi + \alpha\frac{\partial \Psi}{\partial x} = \gamma \sin(\pi y), \quad \alpha = \frac{\beta}{R}, \quad \gamma = \frac{F\pi}{R}, \tag{19}$$

with homogeneous Dirichlet boundary conditions is considered, see Hecht et al. (2000). Here $\beta = 50$ is the North-South derivative of the assumed linear Coriolis parameter, $F = 0.1$

is the strength of the wind forcing which takes the form $\tau = -F\cos(\pi y)$, and $R = 1$ is the strength of the assumed linear frictional force. The analytical solution to (Eq. 19) is given by

$$\Psi(x,y) = \gamma\left(\frac{1}{\pi}\right)^2 \sin(\pi y)\left(pe^{Ax} + qe^{Bx} - 1\right), \tag{20}$$

$$p = \frac{1-e^B}{e^A - e^B}, \quad q = 1-p \tag{21}$$

$$A = -\frac{\alpha}{2} + \sqrt{\frac{\alpha^2}{4} + \pi^2}, \tag{22}$$

$$B = -\frac{\alpha}{2} - \sqrt{\frac{\alpha^2}{4} + \pi^2}. \tag{23}$$

Figure 7 shows a comparison of results obtained with uniform and anisotropic adaptive refinement. The form of the streamfunction yields a velocity field with strong shear in the direction normal to the western boundary. The error measure and adaptive remeshing algorithms used here yield a mesh which has long, thin elements aligned with the boundary. The error plots show the high error focused in the western boundary region in the case of the uniform resolution mesh (Fig. 7).

The automatic assertions here involve ensuring errors from uniform and adaptive mesh calculations are within acceptable bounds of the analytical solution. In particular, the L_2 norm of the error obtained with the adapted mesh is checked to be an order of magnitude lower than that with the fixed mesh, with the adapted mesh using approximately one quarter the number of nodes. For further details of this problem solved using Fluidity-ICOM with isotropic and anisotropic mesh adaptivity see Piggott et al. (2009).

5 Conclusions

Automated continuous testing is widely regarded as industry best practice in the software engineering community, but this message has not yet fully penetrated the numerical modelling community. Rigorous verification is necessary for users to have confidence in the model results, and is generally a requirement for deployment to industry. If the model is under active development, these processes must run continuously as the model is changed; it should therefore be automated. This paper has presented an overview of the software infrastructure uses to automate the Fluidity-ICOM test suite, as well as several of the test cases used.

The deployment of the test suite has yielded dramatic improvements in code quality and programmer efficiency. Almost no developer time is wasted investigating the failure of simulations that used to work. Since feedback about a change to the code is given almost immediately, any errors introduced by new code development can be rapidly fixed. As the test suite acts to lock in correct behaviour of the computational model, the computational model becomes provably more efficient and more accurate over time.

As geoscientific simulations become ever more complex, the software complexity of the computational models increases with it; therefore, the standard of software engineering used to write and manage those scientific models must rise also. The widespread deployment of automated frameworks such as that described here is a necessary step if society at large is to trust the results of geoscientific models.

Acknowledgements. The authors would like to thank J. R. Maddison and H. R. Hiester for helpful discussions and input to the work presented here. The authors wish to acknowledge support from the UK Natural Environment Research Council (grants NE/C52101X/1 and NE/C51829X/1) and support from the Imperial College High Performance Computing Service, the Grantham Institute for Climate Change, and the Institute of Shock Physics.

Edited by: R. Redler

References

Adrion, W. R., Branstad, M. A., and Cherniavsky, J. C.: Validation, Verification, and Testing of Computer Software, ACM Computing Surveys, 14, 159–192, doi:10.1145/356876.356879, 1982.

Babuška, I. and Oden, J. T.: Verification and validation in computational engineering and science: basic concepts, Comput. Method Appl. M., 193, 4057–4066, doi:10.1016/j.cma.2004.03.002, 2004.

Botella, O. and Peyret, R.: Benchmark spectral results on the lid-driven cavity flow, Comput. Fluids, 27, 421–433, 1998.

Brown, P. P. and Lawler, D. F.: Sphere Drag and Settling Velocity Revisited, J. Environ. Eng.-ASCE, 129, 222–231, doi:10.1061/(ASCE)0733-9372(2003)129:3(222), 2003.

Bruneau, C. H. and Saad, M.: The 2D lid-driven cavity problem revisited, Comput. Fluids, 35, 326–348, 2006.

Collins-Sussman, B., Fitzpatrick, B. W., and Pilato, C. M.: Version Control with Subversion, O'Reilly & Associates, available at: http://subversion.apache.org, last access: 16 May 2011, 2004.

Davies, T., Cullen, M. J. P., Malcolm, A. J., Mawson, M. H., Staniforth, A., White, A. A., and Wood, N.: A new dynamical core for the Met Office's global and regional modelling of the atmosphere, Q. J. Roy. Meteorol. Soc., 131, 1759–1782, doi:10.1256/qj.04.101, 2005.

Deleersnijder, E., Fichefet, T., Hanert, E., Legat, V., Remacle, J.-F., and Frazao, S. S.: Taking up the challenges of multi-scale marine modelling, available at: http://sites-final.uclouvain.be/slim/assets/files/documents/ProposalWeb_2.pdf, excerpts from a proposal for a Concerted Research Actions (ARC) programme, last access: 16 May 2011, 2010.

Easterbrook, S. M. and Johns, T. C.: Engineering the Software for Understanding Climate Change, Comput. Sci. Eng., 11, 65–74, doi:10.1109/MCSE.2009.193, 2009.

Erturk, E., Corke, T. C., and Gokcol, C.: Numerical solutions of 2-D steady incompressible driven cavity ow at high Reynolds numbers, Int. J. Numer. Meth. Fl., 48, 747–774, doi:10.1002/fld.953, 2005.

Ford, R., Pain, C. C., Piggott, M. D., Goddard, A. J. H., de Oliveira, C. R. E., and Umpleby, A. P.: A Nonhydrostatic Finite-Element Model for Three-Dimensional Stratified Oceanic Flows, Part II: Model Validation, Month. Weather Rev., 132, 2832–2844, 2004.

Fringer, O. B., Gerritsen, M., and Street, R. L.: An unstructured-grid, finite-volume, nonhydrostatic, parallel coastal ocean simulator, Ocean Model., 14, 139–173, 2006.

Giraldo, F. X. and Restelli, M.: A study of spectral element and discontinuous Galerkin methods for the Navier-Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases, J. Comput. Phys., 227, 3849–3877, doi:10.1016/j.jcp.2007.12.009, 2008.

Graham, S. L., Kessler, P. B., and McKusick, M. K.: gprof: a Call Graph Execution Profiler, in: SIGPLAN Symposium on Compiler Construction, 120–126, doi:10.1145/872726.806987, 1982.

Griffies, S. M.: Elements of MOM4p1, available at: http://data1.gfdl.noaa.gov/~arl/pubrel/r/mom4p1/src/mom4p1/doc/guide4p1.pdf, last access: 16 May 2011, 2009.

Griffies, S. M., Harrison, M. J., Pacanowski, R. C., and Rosati, A.: A Technical guide to MOM4, available at: http://data1.gfdl.noaa.gov/~arl/pubrel/o/old/doc/mom4p0_guide.pdf, last access: 16 May 2011, 2004.

Haidvogel, D. B. and Beckmann, A.: Numerical Ocean Circulation Modeling, Imperial College Press, River Edge, NJ, USA, 1999.

Ham, D. A., Farrell, P. E., Gorman, G. J., Maddison, J. R., Wilson, C. R., Kramer, S. C., Shipton, J., Collins, G. S., Cotter, C. J., and Piggott, M. D.: Spud 1.0: generalising and automating the user interfaces of scientific computer models, Geosci. Model Dev., 2, 33–42, doi:10.5194/gmd-2-33-2009, 2009.

Härtel, C., Meiburg, E., and Necker, F.: Analysis and direct numerical simulation of the flow at a gravity-current head, Part I, Flow topology and front speed for slip and no-slip boundaries, J. Fluid Mech., 418, 189–212, 2000.

Hecht, M. W., Wingate, B. A., and Kassis, P.: A better, more discriminating test problem for ocean tracer transport, Ocean Model., 2, 1–15, 2000.

Hiester, H. R., Piggott, M. D., and Allison, P. A.: The impact of mesh adaptivity on the gravity current front speed in a 2D lock-exchange, Ocean Model., 38, 1–2 , 1–21, doi:10.1016/j.ocemod.2011.01.003, 2011

Howden, W. E.: Reliability of the Path Analysis Testing Strategy, IEEE T. Software Eng., SE-2, 208–215, 1976.

Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python, available at: http://www.scipy.org, last access: 16 May 2011, 2001.

Knupp, P., Ober, C. C., and Bond, R. B.: Measuring progress in order-verification within software development projects, Eng. Comput., 23, 271–282, doi:10.1007/s00366-007-0066-x, 2007.

Madec, G., Delecluse, P., and Imbard, M.: OPA 8.1 ocean general circulation model reference manual, Paris VI, France, note 11, 1998.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., and Heisey, C.: A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers, J. Geophys. Res., 102, 5753–5766, doi:10.1029/96JC02775, 1997.

NEMO Consortium: The Nemo Consortium Agreement, availabe at: http://www.nemo-ocean.eu/, last access: 16 May 2011, 2008.

Oberkampf, W. L. and Atrucano, T. G.: Verification and Validation in Computational Fluid Dynamics, Tech. Rep. SAND 2002-0529, Sandia National Laboratories, Albuquerque, NM, 2002.

Oberkampf, W. L., Sindir, M. M., and Conlisk, A. T.: Guide for the Verification and Validation of Computational Fluid Dynamics Simulations, Tech. Rep. G-077-98, American Institute of Aeronautics and Astronautics, 1998.

Oreskes, N., Shrader-Frechette, K., and Belitz, K.: Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences, Science, 263, 641–646, doi:10.1126/science.263.5147.641, 1994.

Pain, C. C., Umpleby, A. P., de Oliveira, C. R. E., and Goddard, A. J. H.: Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations, Comput. Method Appl. M., 190, 3771–3796, doi:10.1016/S0045-7825(00)00294-2, 2001.

Pain, C. C., Piggott, M. D., Goddard, A. J. H., Fang, F., Gorman, G. J., Marshall, D. P., Eaton, M. D., Power, P. W., and de Oliveira, C. R. E.: Three-dimensional unstructured mesh ocean modelling, Ocean Model., 10, 5–33, doi:10.1016/j.ocemod.2004.07.005, 2005.

Panton, R. L.: Incompressible Flow, Wiley-Interscience, 1996.

Piggott, M. D., Gorman, G. J., Pain, C. C., Allison, P. A., Candy, A. S., Martin, B. T., and Wells, M. R.: A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes, Int. J. Numer. Meth. Fl., 56, 1003–1015, doi:10.1002/fld.1663, 2008.

Piggott, M. D., Farrell, P. E., Wilson, C. R., Gorman, G. J., and Pain, C. C.: Anisotropic mesh adaptivity for multi-scale ocean modelling, Philos. T. Roy. Soc. A, 367, 4591–4611, doi:10.1098/rsta.2009.0155, 2009.

Popper, K. R.: The Logic of Scientific Discovery, Routledge, London, 1959.

Roache, P. J.: Code Verification by the Method of Manufactured Solutions, J. Fluid. Eng. T. ASME, 124, 4–10, doi:10.1115/1.1436090, 2002.

Salari, K. and Knupp, P.: Code Verification by the Method of Manufactured Solutions, Tech. Rep. SAND2000-1444, Sandia National Laboratories, Albuquerque, NM, 2000.

Schroeder, W., Martin, K., and Lorensen, B.: The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, KitWare, Inc., 4th Edn., available at: http://www.vtk.org, last access: 16 May 2011, 2006.

Simpson, J. E.: Gravity Currents in the Environment and the Laboratory, Cambridge University Press, 2nd Edn., 1999.

Song, Y. and Haidvogel, D. B.: A Semi-implicit Ocean Circulation Model Using a Generalized Topography-Following Coordinate System, J. Comput. Phys., 115, 228–244, doi:10.1006/jcph.1994.1189, 1994.

Stein, W., Abbott, T., Abshoff, M., et al.: SAGE Mathematics Software, The SAGE Group, available at: http://www.sagemath.org, last access: 16 May 2011, 2007.

Stern, F., Wilson, R. V., Coleman, H. W., and Paterson, E. G.: Comprehensive approach to verification and validation of CFD simulations, 1: Methodology and procedures, J. Fluid. Eng. T. ASME, 123, 793–802, 2001.

Stommel, H.: The westward intensification of wind-driven ocean currents, Transactions of the American Geophysical Union, 29, 202–206, 1948.

Trefethen, L. N.: Numerical Analysis, in: Princeton Companion to Mathematics, edited by: Gowers, T., Princeton University Press, section IV. 21, 604–615, 2008.

White, L., Deleersnijder, E., and Legat, V.: A three-dimensional unstructured mesh finite element shallow-water model, with application to the flows around an island and in a wind-driven, elongated basin, Ocean Model., 22, 26–47, doi:10.1016/j.ocemod.2008.01.001, 2008.

Williamson, D. L., Drake, J., Hack, J., Akob, R., and Swartzrauber, P. N.: A standard test set for numerical approximations to the shallow water equations in spherical geometry, J. Comput. Phys., 102, 211–224, 1992.