Imperial College London

Verification and validation of simulation software

Dr David A. Ham

Department of Computing, Imperial College London
Grantham Institute for Climate Change, Imperial College London
david.ham@imperial.ac.uk

With much material from Farrell, P. E., Piggott, M. D., Gorman, G. J., Ham, D. A., Wilson, C. R., Bond, T. M., 2011. Automated continuous verification for numerical simulation. Geoscientific Model Development 4 (2), 435–449

How do we know whether to believe the output of a model?

How do we know whether to believe the output of a model?

This is fundamentally a mixed question of mathematics, software engineering, and application science.

How do we know whether to believe the output of a model?

This is fundamentally a mixed question of mathematics, software engineering, and application science.

It's also a critical question which all computational scientists have to answer if they expect other scientists or society at large to take note of their results.

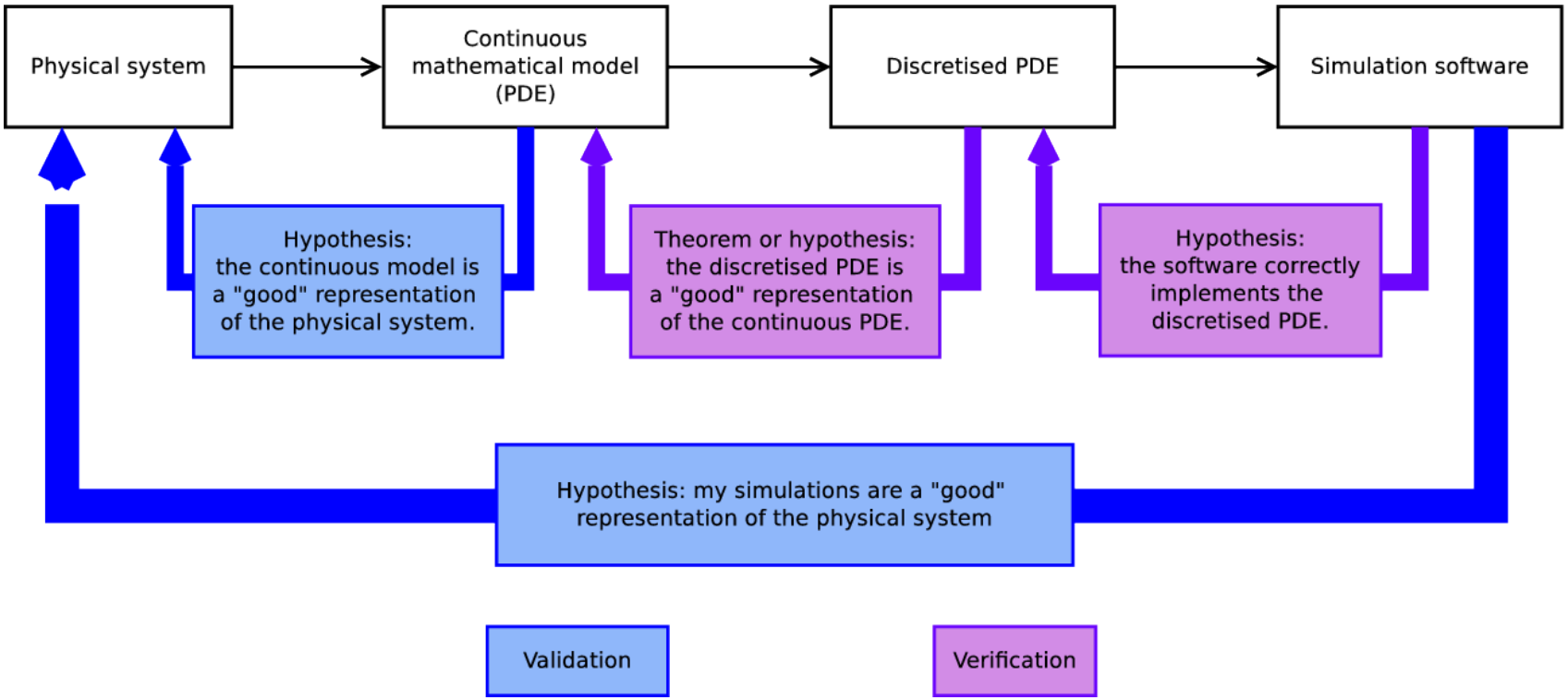
A brief trip back to philosophy of science 101

- Mathematical results are proven. In other words, if the assumptions of a theorem are satisfied, then the result always follows. There is no possibility of another result.
- Science proceeds by hypotheses. These can never be proven in the mathematical sense, but they can be falsified through a contrary observation[1].

So what hypotheses does our software present?

[1] Popper, K. R., 1959. The Logic of Scientific Discovery. Routledge, London, Howden, W. E., 1976. Reliability of the path analysis testing strategy. IEEE Transactions on Software Engineering SE-2 (3), 208–215

Verification and validation operations



Verification and validation operations

- Only the discretisation of the PDE is usually a mathematical operation which can be proven.
- Verification of software by mathematical methods is practiced in some fields of computing, but simulation code is typically far beyond their scope.
- The relationship between the continuous model and the physical system cannot be directly measured, so we must go through all the other steps.

If it's not tested, it's broken.

Unit tests Tests of correct behaviour applied to the smallest possible unit of code.

Analytic solutions Very strong tests of correctness of numerics and implementation, if you can get one!

Method of manufactured solutions A mechanism for generating analytic solutions.

Regression tests Tests against a previously computed result. Only a test of change, not of correctness.

Third party solution Tests against another model. Better than regression tests, but only as good as the other model.

Comparisons to "real" data Essential in validating a model, but of limited use in verification.

Some PDE testing theory

First, know what convergence means for your numerics. Numerical schemes for PDEs usually have convergence behaviour given by an expression such as:

$$E_h = O(h^n) \tag{1}$$

Where E_h is the error in the numerical solution, h is a measure of the mesh spacing and n is the order of convergence of the method. It is important to understand both sides of this expression.

Testing convergence behaviour of numerical schemes is a very sensitive verification test, since almost any error in the numerics will degrade the convergence order.[2]

[2] At least for schemes of greater than first order.

Definition of error

Finite volume and finite element schemes converge in (at least) the space L^2. So the definition:

$$E_h = \left(\int_\Omega (\hat{S} - S)^2 \mathrm{d}V \right)^{\frac{1}{2}} \tag{2}$$

is often appropriate. Here Ω is the solution domain, $\hat{S}$ is the numerical solution and S is the exact solution.

Importantly, this is not the same as pointwise evaluation of the solution.

L^2-norm for low-order finite volume

For low-order finite volume, the L^2-norm is approximated by summing over control volumes:

$$E_h \approx \left(\sum_{v \in \Omega} \left(\mathrm{vol}(v) \left(\hat{S}_v - S(\mathbf{x}_v) \right) \right)^2 \right)^{\frac{1}{2}} \tag{3}$$

where $\mathbf{x}_v$ is the location of the control point. This creates an $O(h^2)$ error in the integration of the analytic solution, so is only acceptable for schemes of second order and below.

General rule: remember to always ensure that the error you make measuring the error is smaller than the error itself!

Evaluating convergence rate

Let's return to:

$$E = O(h^n) \tag{4}$$

and which means:

$$\lim_{h \to 0} E_h \leq Ch^n \tag{5}$$

For some C.

Evaluating convergence rate

Let's return to:

$$E = O(h^n) \tag{4}$$

and which means:

$$\lim_{h \to 0} E_h \leq Ch^n \tag{5}$$

For some C. Further, if n is the optimal convergence rate of the scheme, then for sufficiently small h:

$$E_h \approx Ch^n \tag{6}$$

Evaluating convergence rate

Suppose we run the simulation on two meshes with typical (small) mesh spacing h_1 and h_2:

$$E_{h_1} \approx Ch_1^n, \tag{7}$$

$$E_{h_2} \approx Ch_2^n, \tag{8}$$

Evaluating convergence rate

Suppose we run the simulation on two meshes with typical (small) mesh spacing h_1 and h_2:

$$E_{h_1} \approx Ch_1^n, \tag{7}$$

$$E_{h_2} \approx Ch_2^n, \tag{8}$$

$$\frac{E_{h_1}}{E_{h_2}} \approx \left(\frac{h_1}{h_2}\right)^n \tag{9}$$

Evaluating convergence rate

Suppose we run the simulation on two meshes with typical (small) mesh spacing h_1 and h_2:

$$E_{h_1} \approx Ch_1^n, \tag{7}$$

$$E_{h_2} \approx Ch_2^n, \tag{8}$$

$$\frac{E_{h_1}}{E_{h_2}} \approx \left(\frac{h_1}{h_2}\right)^n \tag{9}$$

$$n \approx \log_{h_1/h_2}\left(\frac{E_{h_1}}{E_{h_2}}\right) \tag{10}$$

Method of manufactured solutions

MMS is a ridiculously simple idea:

- choose the PDE
- choose the function you want as the solution
- calculate what the right hand side source term would have to be.

Method of manufactured solutions

Example: the advection-diffusion equations in 2D.

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T - \kappa \nabla^2 T = 0 \tag{11}$$

Method of manufactured solutions

Example: the advection-diffusion equations in 2D.

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T - \kappa \nabla^2 T = 0 \tag{11}$$

We need a source term, S

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T - \kappa \nabla^2 T = S \tag{12}$$

Method of manufactured solutions

Next, we choose values for the paramters $\mathbf{u}$ and κ, and a solution $T(x, y, t)$.

Method of manufactured solutions

Next, we choose values for the paramters $\mathbf{u}$ and κ, and a solution $T(x, y, t)$. We choose (fairly arbitrarily):

$$\kappa = 0.7 \tag{13}$$

$$\mathbf{u} = \begin{bmatrix} \sin\left(5(x^2 + y^2)\right) \\ \cos\left(3(x^2 - y^2)\right) \end{bmatrix} \tag{14}$$

$$T(x, y, t) = \sin\left(25xy\right) - 2y/x^{1/2} \tag{15}$$

Method of manufactured solutions

Now, it's "just" a matter of substituting into the equations. This is a somewhat technical process which is most easily and reliably achieved using a symbolic algebra package. We used Sage, other examples include Maple and Matlab:

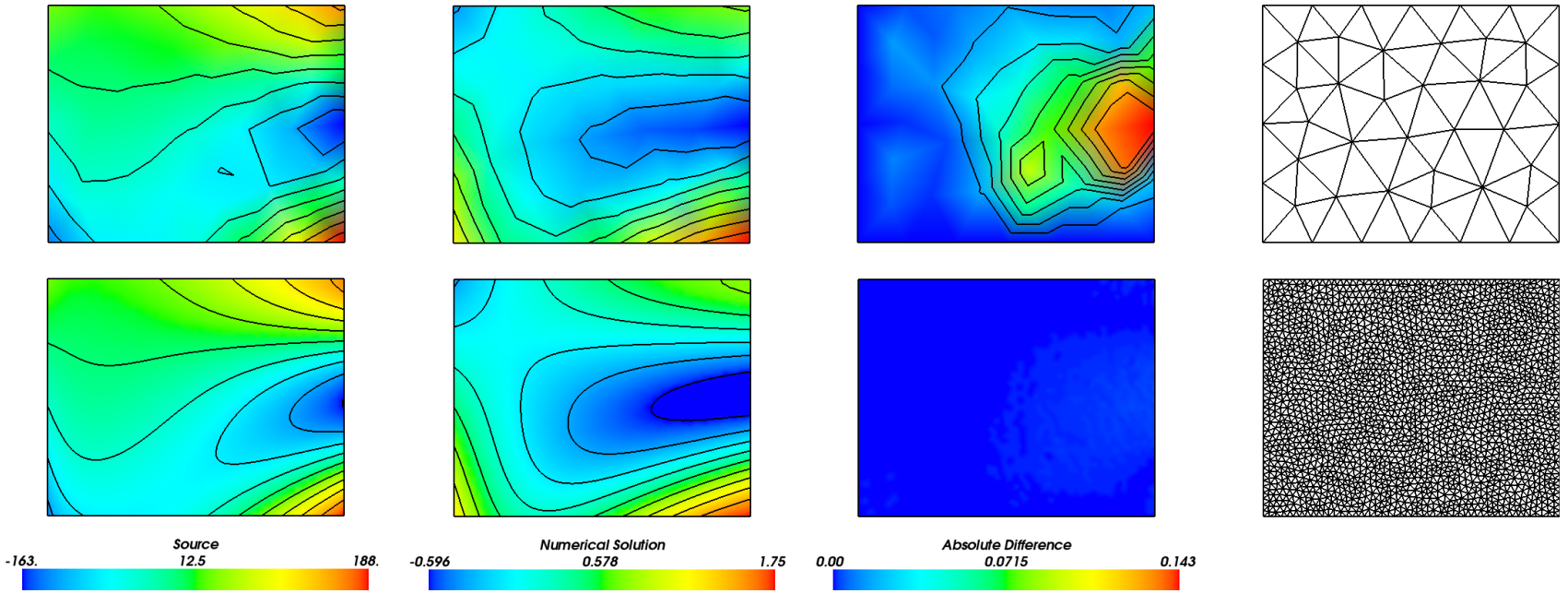
$$
\begin{aligned}
S &= \frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T - \kappa\nabla^2 T, \\
&= \left(25y\cos(25xy) + y/x^{3/2}\right)\sin\left(5(y^2+x^2)\right) \\
&\quad + \left(25x\cos(25xy) - 2/x^{1/2}\right)\cos\left(3(x^2-y^2)\right) \\
&\quad + 0.7\left(625(x^2+y^2)\sin(25xy) + 3y/(2x^{5/2})\right).
\end{aligned}
$$

Method of manufactured solutions

Limitations and issues:

- Be wary of introducing errors in the discretisation of the source term.
- Ditto for any initial or boundary conditions.
- Unfortunate choices of solution may "switch off" terms.
- Only works for convergent schemes!

Method of manufactured solutions



From left to right: source term 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).

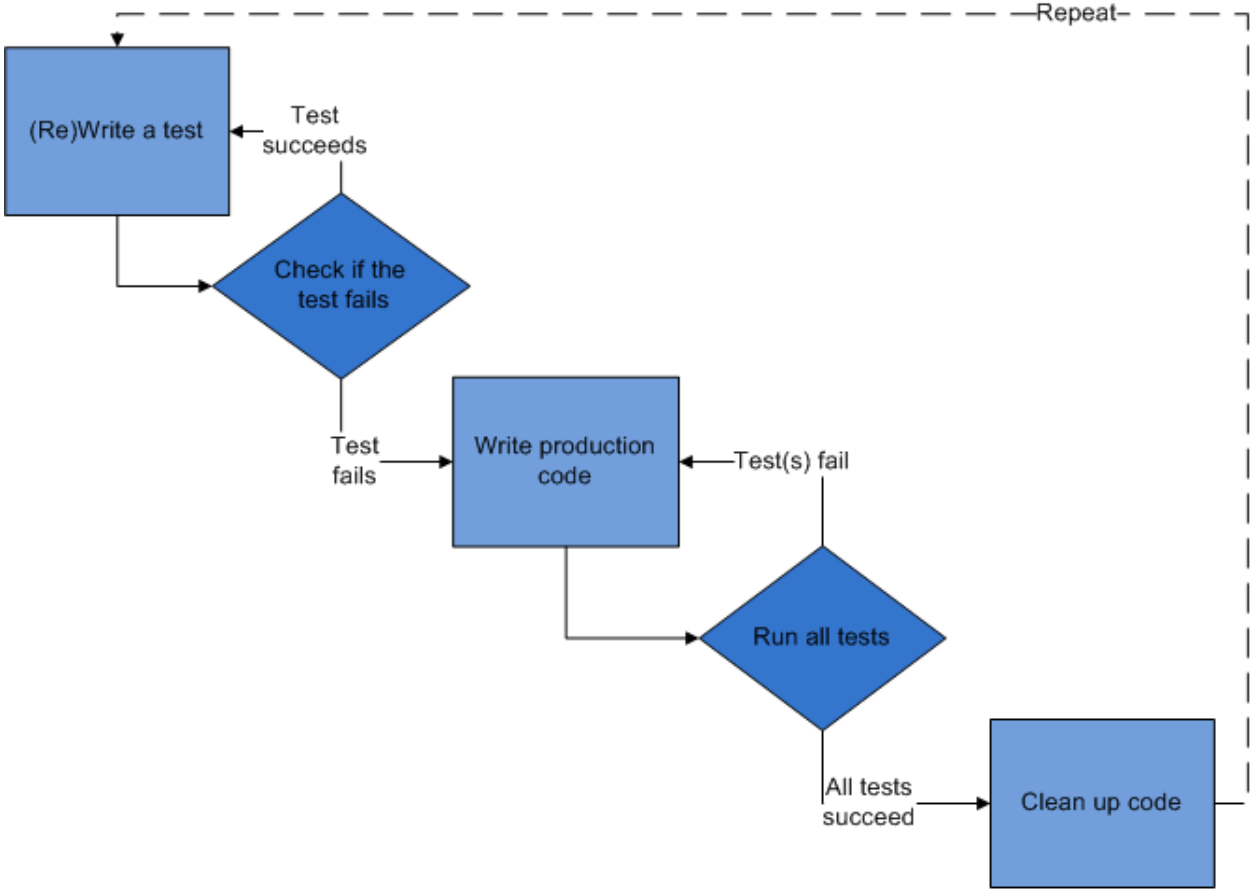
Method of manufactured solution: does our software work?

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

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.

Test driven development

Core commandments of the cult:



Reality check: a test heirarchy

Fluidity at Imperial uses this heirarchy:

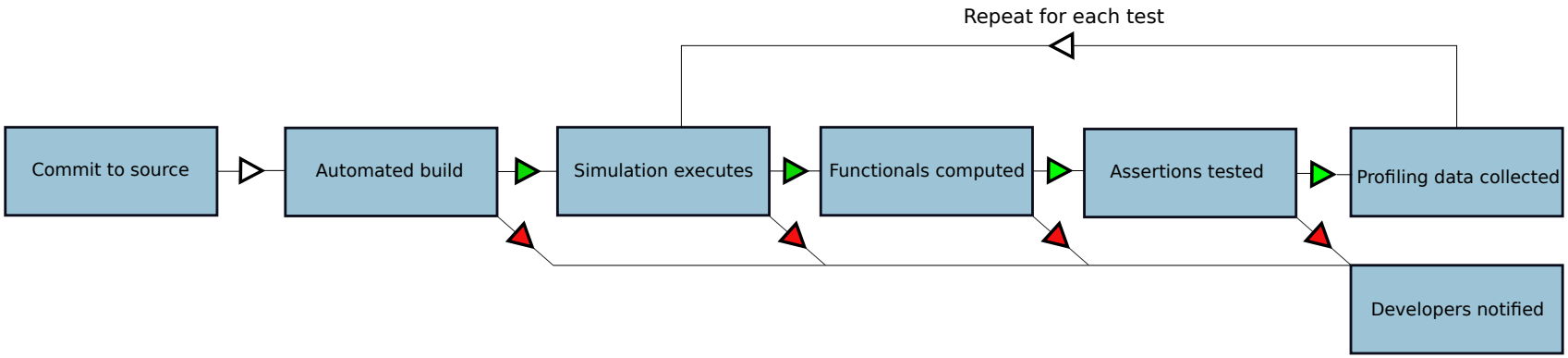
1. Unit tests - very fast
2. Short tests $<$ 20s each
3. Medium tests up to several minutes.
4. Long tests - usually parallel, may take hours.

- Users are expected to run unit tests and short tests locally.
- Everything up to medium is run on every trunk commit and can be requested for any branch commit.
- Long tests run on trunk on a rolling basis.

Even more real check: where to start

1. create a test target in your build system (e.g. `make test`)
2. add unit tests using your favourite unit test framework.
3. add verification tests using your favourite scripting language.
4. `make test`

Buildbot: test automation



Coverage tools: gcov

How do you know how good your tests are? If you're using gcc, then gcov can help!

- gcov (after using the right compiler options) provides information about how many times each file is called.
- lcov provides a graphical representation of this.
- However! Touching each line of code is necessary but not sufficient!