Verification of Scientific Codes

Åsmund Ødegård

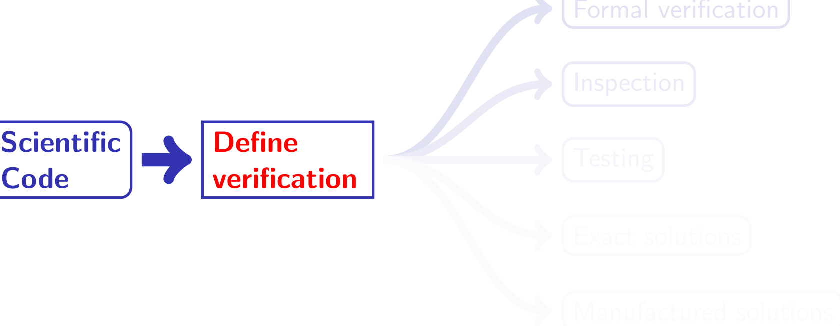
Simula Research Laboratory AS

30. March, 2006

Overview of talk: Verification of Scientific Codes

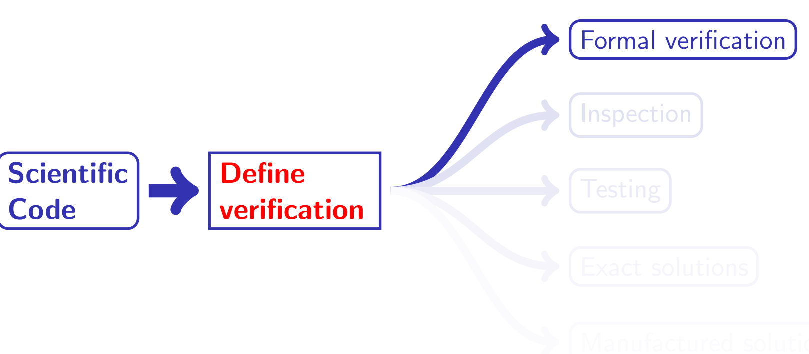
And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



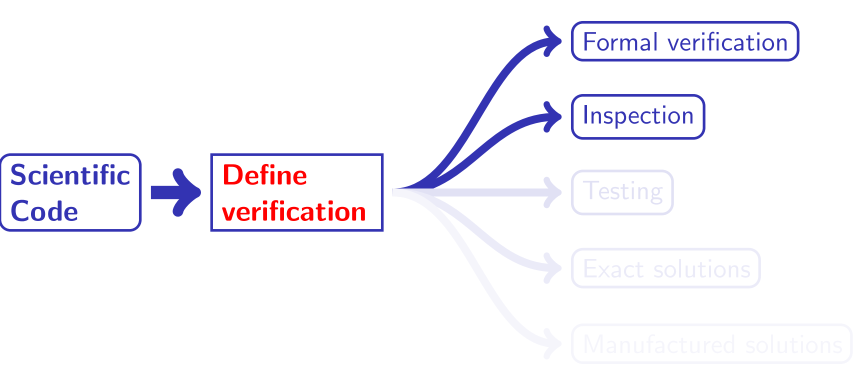
And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



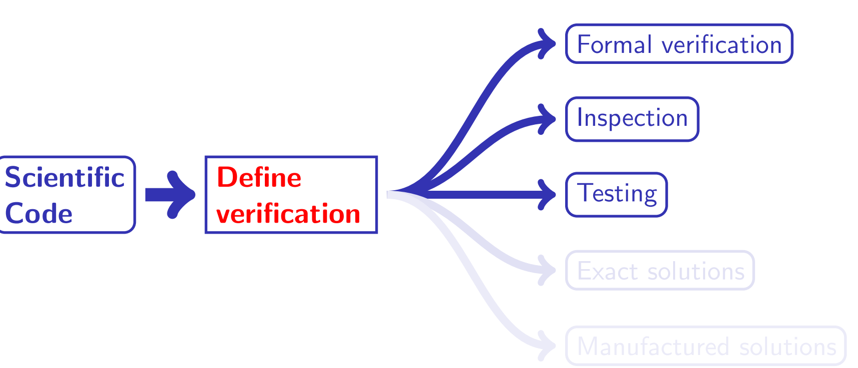
And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



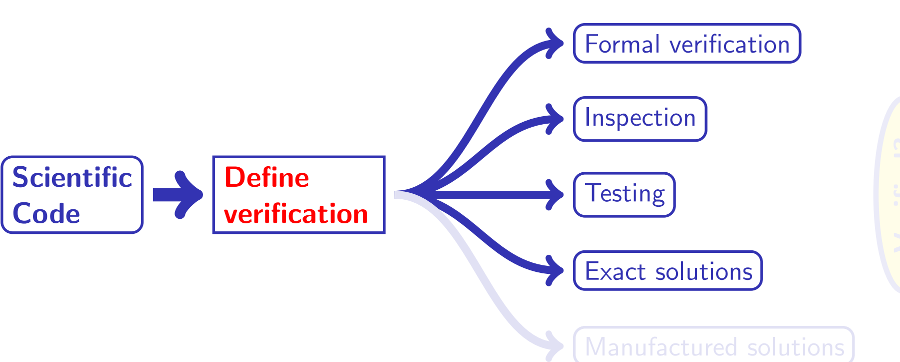
And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes

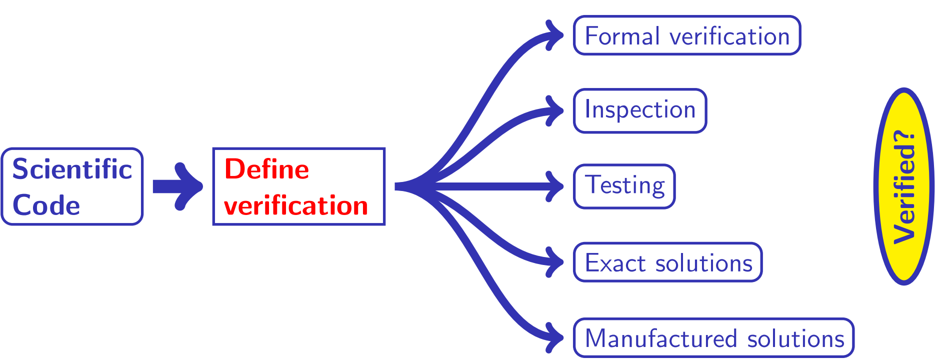
Scientific Code → **Define verification**

- Formal verification
- Inspection
- Testing
- Exact solutions
- Manufactured solutions

Verified?

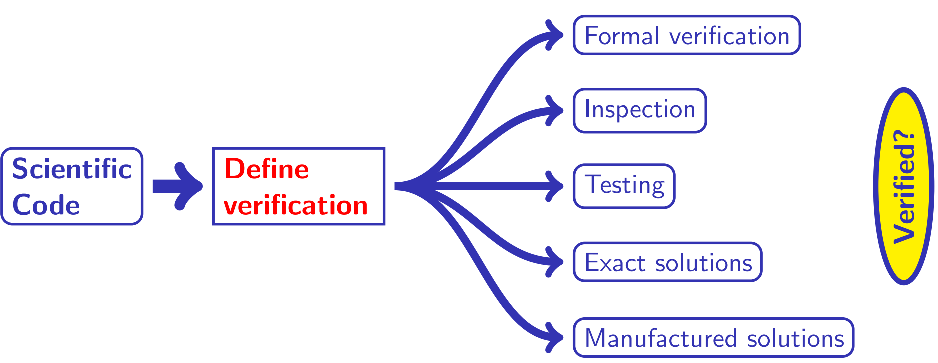
And then what, beyond code verification!

Overview of talk: Verification of Scientific Codes



And then what, beyond code verification!

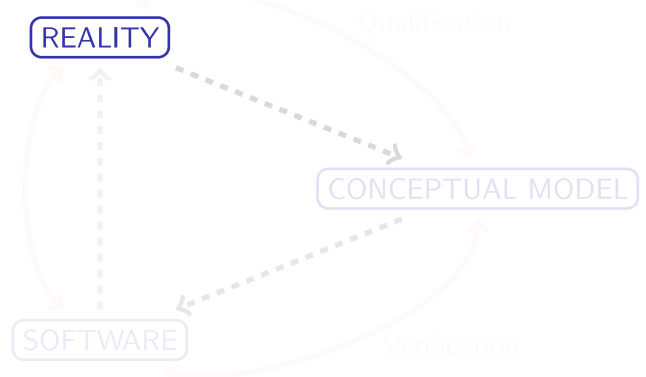
Overview of talk: Verification of Scientific Codes



And then what, beyond code verification!

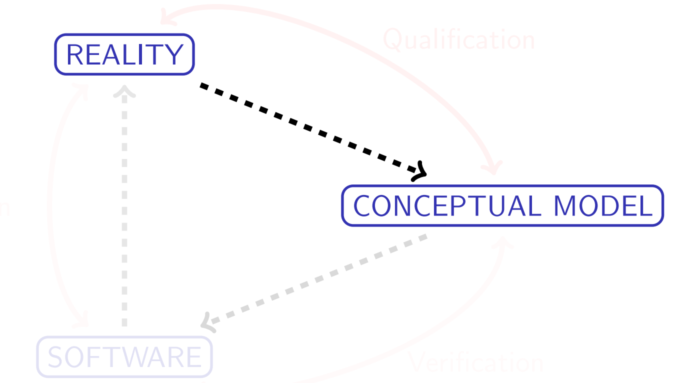
Setting the scene Oberkampf et. al.[1]

Consider an application or problem of interest!



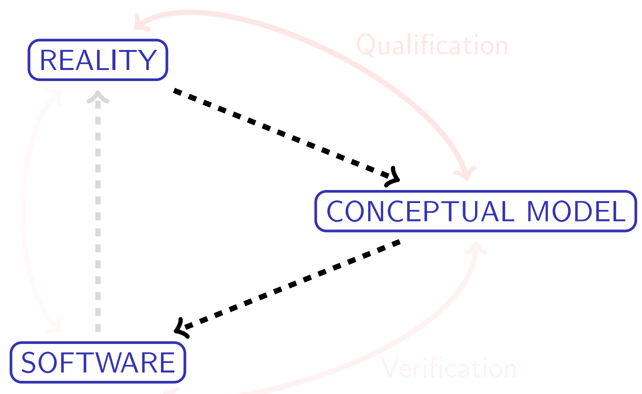
Setting the scene Oberkampf et. al.[1]

Create a conceptual model for the problem.



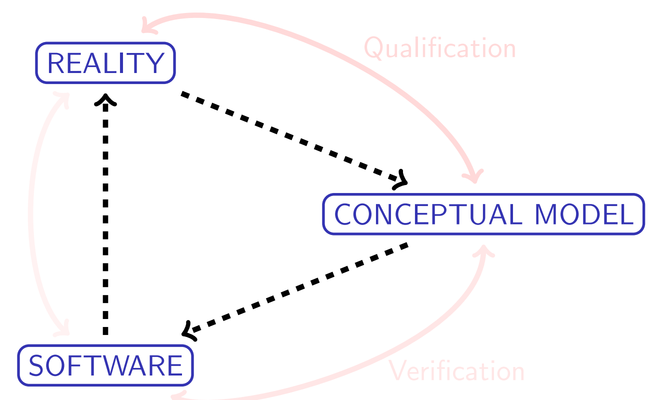
Setting the scene Oberkampf et. al.[1]

Implement a computer code for solving the model.



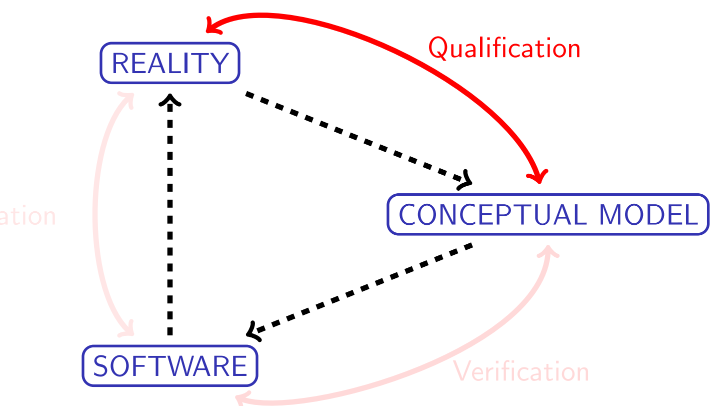
Setting the scene — Oberkampf et. al.[1]

Use the program to simulate the original problem!



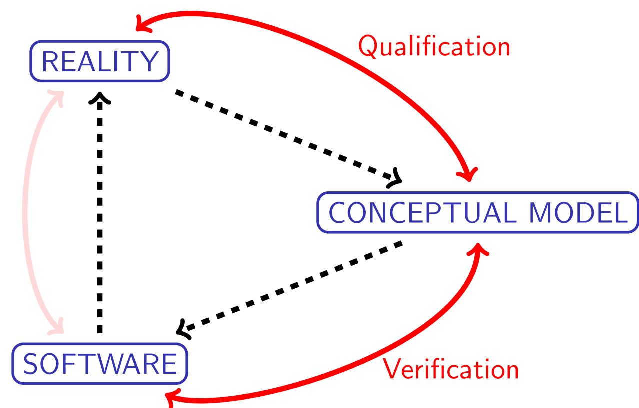
Setting the scene Oberkampf et. al.[1]

Is the conceptual model adequate for the intended application?



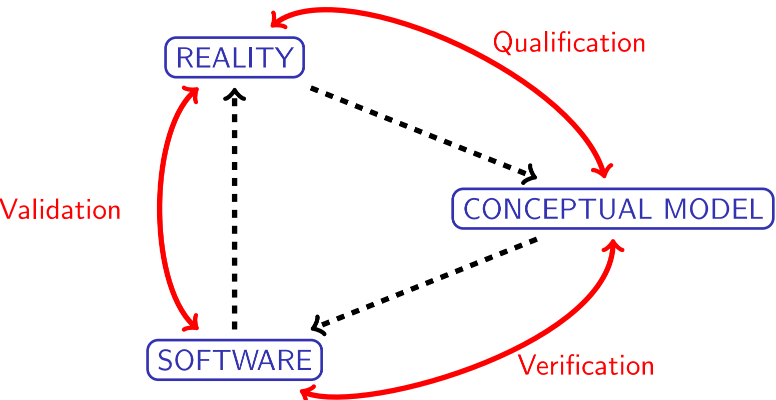
Setting the scene Oberkampf et. al.[1]

Do we solve the equations right?

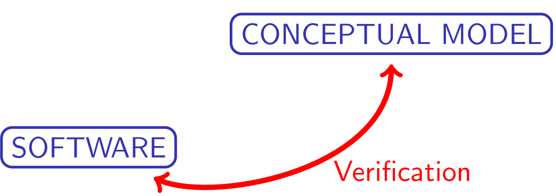


Setting the scene Oberkampf et. al.[1]

Do we solve the right equations?

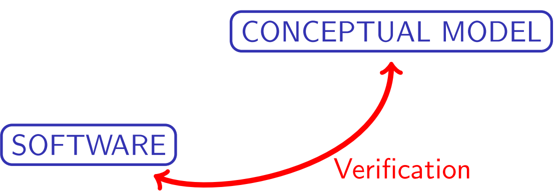


We focus on the verification part



- Assume that the Conceptual model is given, usually as a system of Partial Differential Equations (PDEs). The term "mathematical model" will also be used.
- Further, assume that software for solving the model is implemented.

We focus on the verification part



- Assume that the Conceptual model is given, usually as a system of Partial Differential Equations (PDEs). The term “mathematical model” will also be used.
- Further, assume that software for solving the model is implemented.

Definitions of code verification

Verification is confirmation of truth

Computer science definitions:

Are we building the product right?
Verification – solving the equations right

Some other definitions:

Formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property, using formal methods.

Verification is to convincingly demonstrate that the code yields a solution to the conceptual model when some discretization parameter h tends to zero.

The goals for code verification.

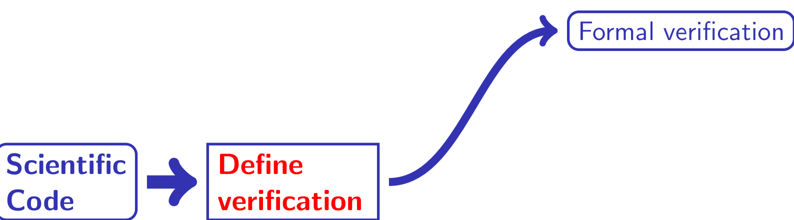
- Make sure that there are no coding mistakes ("bugs") in the code.
- Make sure that there are no mistakes in the numerical method.
- And in the end, make sure that any solution found by the code actually is an approximative solution of the conceptual model.

It is debatable whether such goals are reachable for complex codes. A more pragmatic view is therefore that verification is about minimizing the risk for mistakes in the code and in particular calculations.

What code verification is not about.

- We do not want to say anything about robustness of the code.
- We do not want to compare with reality, as that is part of validation.
- Quantification of uncertainties in calculation is usually considered to be part of soultion verification, and will not be discussed today.

Overview of talk: Verification of Scientific Codes



Formal verification related to scientific codes

The act of proving or disproving the correctness of a system with respect to a certain formal specification or property

- Create an abstract mathematical model of the code.
- Correspondance between code and the mathematical model known by construction.
- Establish a formal proof for the mathematical model, where the theorm is the formal specification.

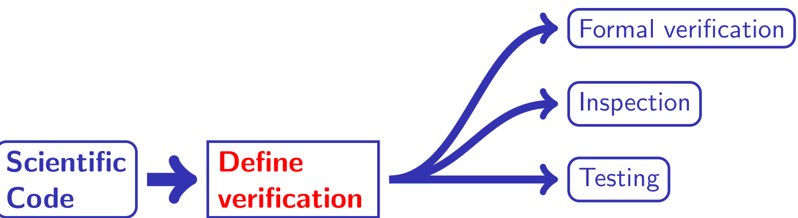
- Some software for automatically checking the proof exist.
- Most available software depends on the use of special purpose languages, although some preliminary results are reported for general purpose languages as well, c.f.[11].

Possibilities for formal verification in the future?

- There is reported some efforts on semantic analysis of code, to detect the numerics implemented by the code, c.f.[10].
- Such information could be used to check that implemented code agree with the problem specified by the user, that is what he wants to implement.
- This is also preliminary work.

The formal verification approach will be interesting to watch, but is not feasible today.

Overview of talk: Verification of Scientific Codes



A test problem

Consider Poisson's equation on the unit square:

$$\begin{aligned} -\nabla^2 u = f \quad & \text{in} \quad \Omega, \\ u = 0 \quad & \text{in} \quad \partial\Omega, \end{aligned}$$

where $\Omega = [0,1] \times [0,1]$ and $\partial\Omega$ the boundary of Ω.

We have implemented a solver for this problem with

$$f = sin(2\pi x) sin(2\pi y).$$

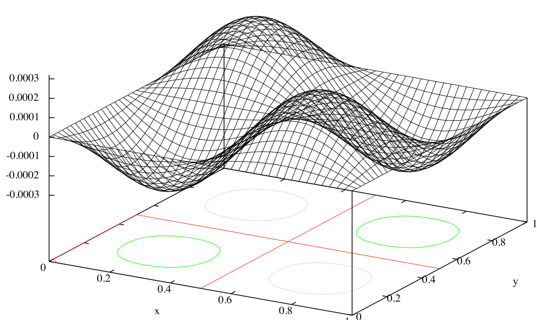
The solver is implemented in Python, using the Finite Difference Method.

We will use the general expression $Lu = f$ for the exact problem, and $L_h u_h = f$ for the discrete problem.

Inspection as part of verification

- Does the code compile?
- Does it run?
- Does the results seems reasonable?
- Use knowledge about the model, check if expected features are reproduced.
  - If there are symmetries in the model, are these present in the numerical solution
  - The steady–state solution of for instance heat conduction is expected to be smooth, is this reproduced?

Solution of our test-problem



Seems reasonable - a nice plot, and we recognize the well known sine-shape present in the source as well.

Static and dynamic tests.

In addition to make sure that the code compiles, there are a few more things we can do:

- Using tools like *lint*, we can find variables that are used before initialization.
- Using other tools we can check the program dynamically for out–of–bounds indexing of arrays.
- It is also possible to detect unreachable code segments, which probably will indicate a coding mistake.

Convergence

Convergency is a basic requirement for a numerical algorithm. That is, when the grid discretization parameter, h, tend to zero, the numerical should approach to something.

- Assume u_h solves the numerical problem $L_h u_h = f$ and let $R_h = L_h u_h - f$ be the residual. Then we should have

$$||R_h||_\Delta \to 0 \text{ when } h \to 0.$$

- It is dangerous to check with the discrete L_h present in the code.
- We can check $||u_{h^i}||_\Delta$ for decreasing values of h^i.
- Or we can check whether $||u_{h^i} - u_{h^{i+1}}||_\Delta$ approaches 0.

Run our testcase for various values of h

h	norm of R	norm of solution
0.2	4.269e-16	0.02618
0.1	2.419e-16	0.004045
0.05	2.291e-16	0.0006928
0.025	5.332e-16	0.0001602
0.0125	5.92e-16	3.93e-05
0.00625	3.504e-16	9.781e-06

- While the norm of R is small, it does not converge, Also, the norm of the solution does not converge – obviously the solution move closer to zero each time h is reduced.
- There must be mistake in the code

The center of the stencil had -2 instead of -4.

Comparison tests

If there exists established codes that solves the same problem, solutions generated with the new and the old codes can be compared.

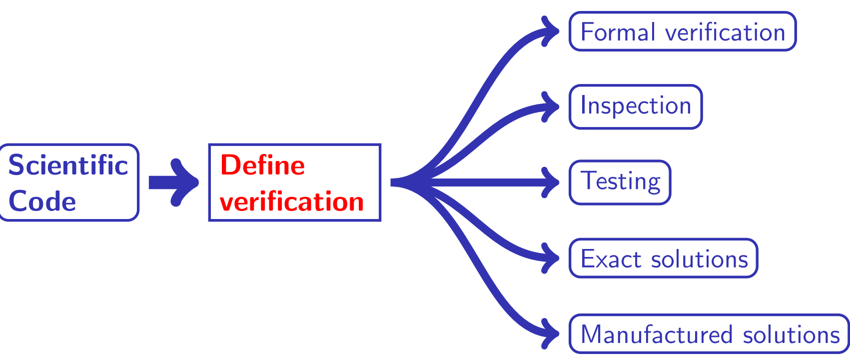
- If the maximum difference between solutions is within some given tolerance, the code is likely free of mistakes.
- No exact solution is required to perform this test.
- A grid refinement procedure should be used to check that results are in the asymptotic range.
- Asymptotic range is defined as the range of discretization parameters where the lowest–order terms in the truncation error dominate.

Comparison tests continued...

There are some drawbacks to this approach:

- We usually don't have different codes for the same problem.
- If different meshes are used, we need to introduce approximations in input/output in order to compare.
- If the codes targets slightly different applications, we may only be able to do the comparison for some simplified cases.
- The confidence in the new code will depend heavily on the confidence in the old one. There is always the possibility for doing the same mistake twice!.

Overview of talk: Verification of Scientific Codes



Verification with Method of Exact Solutions (MES)

If an exact solution to the mathematical model can be found, the numerical solution can be compared to the exact solution.

- Exact solutions are generally not available!.
- Sometimes exact solutions can be found for simplified models
  - Approximate a nonlinear model with a linear model.
  - Simplify the geometry.
- If such simplifications are done, we can not consider the code verified for the full model.

MES continued...

If an exact solution for a simplified problem can be found:

- Stepwise add features back in the model
- For each step, compare the new solution with the previous one
- If the new solution seems to agree with the feature added, continue.

In this way, we can conclude that the numerical solution for the full model seems reasonable. Still, this is only based on (expert) judgement, not quantitative measure.

Verification with Method of Manufactured Solutions (MMS)

Consider the testproblem again:

$$-\nabla^2 u = f.$$

Instead of seeking some exact solution, we “manufacture” a solution:

$$u(x, y) = x(1-x)y(1-y)e^{x+y},$$

and calculate f such that u is a solution:

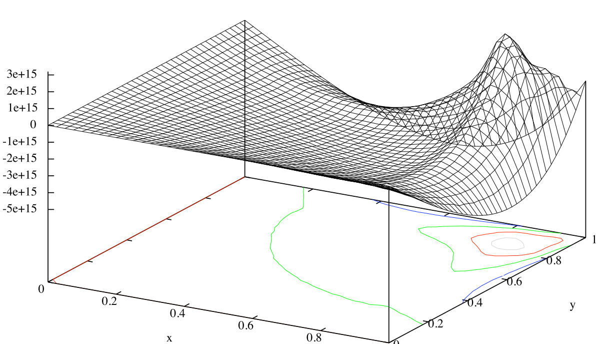
$$f = \left[(3x + x^2)y(1-y) + (3y + y^2)x(1-x)\right] e^{x+y}.$$

Here, the solution match our boundary condition. Generally, we have to adjust the boundary and initial conditions.
(This example is taken from Tveito and Winther[8]).

MMS continued...

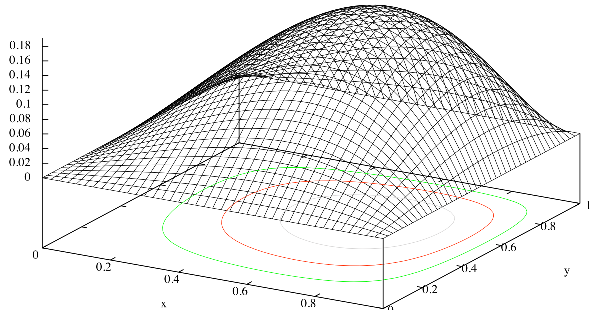
- Run the code using the computed source function f and the corresponding initial and boundary conditions.
- Compare the numerical solution with the now known analytical solution.
- If the numerical solution agree with the analytical solution, the code is verified for this solution.

Plot of solution from testcase:



Doesn't look very good! Boundary conditions are not correct.

Plot of solution from testcase:



Plot after correction.

Some issues with MMS

We will now consider the following topics related to MMS:

- In what sense should the numerical solution agree with the analytical solution.
- Is it possible to conclude based on only one manufactured solution? If not, how many solution do we have to manufacture?
- Practical notes on the implementation of MMS.

Acceptance criteria for MES and MMS

Assume u is the analytical solution, u_h the numerical solution for some grid parameter h and $e_h = ||u - u_h||_\Delta$ the error.

Consistency:

- Perform more tests with different values for the grid discretization parameter h.
- The numerical solution is consistent if

$$lim_{h \rightarrow 0} e_h = 0.$$

Consistency - Convergence order

Acceptance criteria for MES and MMS

Assume u is the analytical solution, u_h the numerical solution for some grid parameter h and $e_h = ||u - u_h||_\Delta$ the error.

Convergence Order check (also named order of accuracy):

- For a consistent method, we should have:

$$e_h = Ch^p + O(h^{p+1}),$$

  p theoretical convergence order, C constant independent of h.

- Observed convergence order: Assume that a numerical solution is found for h and h/r:

$$\tilde{p} \approx log(\frac{e_h}{e_{h/r}})/log(r).$$

Consistency - Convergence order

Acceptance critera for MES and MMS

- For any of the acceptance critera to be meaningfull, the discretization parameters (h, Δt, etc.) should be in the asymptotic range.
- Theoretical convergence order is obtained from truncation error analysis (FDM/FVM) or interpolation theory (FEM).
- Round–off/iterative errors should be $< 10^{-2} \times e_h$, the discretization error.

Convergence order for the testcase.

h	error norm	observed order
0.2	0.00193	
0.1	0.000485	1.99
0.05	0.000121	2.0
0.025	3.04e-05	2.0
0.0125	7.59e-06	2.0
0.00625	1.9e-06	2.0

- It can be showed that the theoretical order is 2.
- Hence, our code has passed the verification test!

Comments on the MMS

MMS is considerd to be the strongest verification method in the literature. Salari and Knupp[2, 3] propose a 12 step procedure for verification. Some of the main points:

- Design a suite of coverage tests: The manufactured solutions should cover all options in the code – especially all possible boundary conditions.
- Perform a full grid refinement test and compare observed and theoretical convergence order.

According to these authors, the code is verified by definition if the code pass the complete procedure.

Some objection to the definition of verified

- The solutions used in MMS will usually be, and some authors, e.g. Roy[7] states should be, smooth functions, while solutions may be non-smooth.
- One may argue that a few isolated test, even when they exercise all parts of the code, still just verify that the code is correct for these isolated cases.
- For instance, solving the problem on different geometries may introduce new problems – remark that geometry/mesh often are read from file.
- It is not clear whether this is a problem with desiging coverage tests, or with the method itself.

Some ideas regarding FEM

- It may be possible to implement MMS in the weak sense, i.e., manufacture solutions u such that $a(u, v) = L(v)$.
- Also, as the numerical solutions in FEM is a linear combination of basis functions,

$$u_h = \sum_i \alpha_i \varphi_i,$$

  it may be possible to verify the basis functions φ_i, by using them as manufactured solutions. Remark that the basis functions are usually not smooth, and hence must be interpreted in the weak sense.
- Whether we can prove verification for all solutions is an open question.

Practical implementation of MMS

In order to be able to use MMS, there are some requirements for the code we are testing:

- It must be possible to replace the source function with a new function.
- Also, we need to specify tailored initial and boundary conditions.
- As the expression for the source function can be complicated, it will be especially beneficial if we can couple the code directly with some symbolic math tool.

Fully Automatic Method of Manufactured Solutions

An interesting approach is taken by Skavhaug et.al[9]:

- famms is a framework for MMS implemented in Python.
- Code implemented in Fortran and C/C++ can be coupled with a symbolic engine through a Python layer.
- In this way the MMS can be automated.

Ready to move on

- The next step is to quantify uncertainties in the numerical solution.
- Richardsons extrapolation, based on $e_h = Ch^p + O(h^{p+1})$, is a standard procedure for obtaining uncertainty bounds without knowledge of the analytical solution.
- Further on validation against the original reality must be done.
- If the solution does not match physical evidence, within the uncertainty bound, we must now assume that the conceptual model is wrong.

Conclusions

- Based on the litterateur on the subject, we have defined verification.
- In the absence of formal verification method, a practical procedure towards verification is presented.
- An example show how mistakes can be discoverd using the procedure.
- Although it is an open question whether verification must be an ongoing process, the procedure based on MMS provide substantial evidence for correctness of the code.

William L. Oberkampf, and Timothy G. Trucano.
Verification and Validation in Computational Fluid Dynamics.
Sandia Report 2002-0529, Sandia National Laboratories, 2002.

Patrick Knupp, and Kambiz Salari, in: K.H. Rosen (Ed.).
Verification of Computer Codes in Computational Science and Engineering.
Chapman and Hall/CRC, Boka Raton, FL, 2003.

Kambiz Salari, and Patrick Knupp.
Code Verification by the Method of Manufactured Solutions.
Sandia Report 2000-1444, Sandia National Laboratories, 2000.

Patrick J. Roache.
Verification and Validation in Computational Science and Engineering.
Hermosa Publishers, Albuquerque NM, 1998.

- Reid Simmons, Charles Pecheur, and Grama Srinivasan.
  Towards Automatic Verification of Autonomous Systems.
  Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2000.

- Hans Petter Langtangen and Aslak Tveito.
  Numeriske metoder i kontinuumsmekanikk
  Sintef Rapport 91689, Oslo, 1991

- Christopher J. Roy.
  Review of code and solution verification procedures for computational simulation.
  Journal of Computational Physics, vol. 205, pp. 131-156, 2005.

- Aslak Tveito and Ragnar Winther.
  Introduction to Partial Differential Equations, A Computational Approach.
  Texts in Applied Mathematics vol 29, Springer–Verlag, 1998.

- Ola Skavhaug, Kent-Andre Mardal, and Hans Petter Langtangen.
  A Python Framework for Verifying Numerical Solutions of Partial Differential Equations
  To appear.
- Mark E. M. Stewart.
  An Experiment In Scientific Program Understanding.
  Proceedings of IEEE Conference on Automated Software Engineering, pp. 281-284, 2000.
- Changqing Wang and David R. Musser.
  Dynamic Verification of C++ Generic Algorithms.
  Software Engineering, vol 23, no. 5, pp. 314-327, 1997.