PS432/PS520 Computational Physics

Miles Turner

Room N105
Ext. 5298
Email: miles.turner@dcu.ie

January 11, 2022

Aims of this lecture

- Recall:

  Verification of Code: Testing for faults in code. Something that developers usually do, and which has to be done only once (unless the code is changed).

  Verification of Calculations: Demonstrating that the effect of numerical parameters has been quantified and is suitably small. An exercise that should be done for every calculation.
- Apply these verification techniques to a simple problem
- Develop the idea that testing against an exact solution is the preferred approach to Verification of Code

Simple Harmonic Motion

- Consider some potential $V(x)$. Near $x = 0$:

$$\begin{aligned} V(x) &\approx V(x=0) + \left.\frac{dV}{dx}\right|_{x=0} x \\ &+ \frac{1}{2}\left.\frac{d^2V}{dx^2}\right|_{x=0} x^2 + \frac{1}{6}\left.\frac{d^3V}{dx^3}\right|_{x=0} x^3 + \cdots \end{aligned}$$

- Suppose there is a stable equilibrium at $x = 0$:

$$\begin{aligned} \left.\frac{dV}{dx}\right|_{x=0} &= 0 \\ \left.\frac{d^2V}{dx^2}\right|_{x=0} &> 0 \end{aligned}$$

Simple Harmonic Motion

- Then:

$$F = -\frac{dV}{dx} \propto -x$$

Hence:

1. For a body moving on any reasonable potential, with “small enough” displacement:

$$F \propto -x$$

2. So Simple Harmonic Motion is a ubiquitous approximation for oscillatory motion near (almost) any stable equilibrium
3. For displacements that are not “small enough”, nonlinear corrections to the force will be needed

Simple Harmonic Motion

- For a linear spring with constant k:

$$F = -kx$$

and

$$m\frac{d^2x}{dt^2} = -kx$$

or

$$\begin{aligned}\frac{d^2x}{dt^2} &= -\frac{k}{m}x \\ &= -\omega_0^2 x\end{aligned}$$

with general solution:

$$\begin{aligned}x(t) &= A\cos\omega_0 t + B\sin\omega_0 t \\ &= C\cos(\omega_0 t + \phi)\end{aligned}$$

with A, B, C and ϕ arbitrary constants

Leapfrog method

The ordinary differential equations:

$$\begin{aligned}\frac{dx}{dt} &= v \\ \frac{dv}{dt} &= -\omega_0^2 x\end{aligned}$$

are replaced by the difference equations:

$$\begin{aligned}\frac{v_{n+1/2} - v_{n-1/2}}{\Delta t} &= -\omega_0^2 x_n \\ \frac{x_{n+1} - x_n}{\Delta t} &= v_{n+1/2}\end{aligned}$$

where

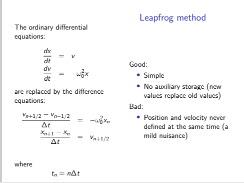
$$t_n = n\Delta t$$

Good:

- Simple
- No auxiliary storage (new values replace old values)

Bad:

- Position and velocity never defined at the same time (a mild nuisance)



Leapfrog method

The ordinary differential equations:

$$\begin{aligned}\frac{dx}{dt} &= v\\ \frac{dv}{dt} &= -\omega_0^2 x\end{aligned}$$

are replaced by the difference equations:

$$\begin{aligned}\frac{v_{n+1/2}-v_{n-1/2}}{\Delta t} &= -\omega_0^2 x_n\\ \frac{x_{n+1}-x_n}{\Delta t} &= v_{n+1/2}\end{aligned}$$

where

$$t_n = n\Delta t$$

Good:

- Simple
- No auxiliary storage (new values replace old values)

Bad:

- Position and velocity never defined at the same time (a mild nuisance)

The “new values replace old values” aspect does not matter much when we are dealing with a single particle, but basically the same method can be used in cases where there are millions or even billions of particles, and then this is an important advantage.

Leapfrog method: Stability and accuracy

- We can write

$$x_{n+1}-2x_n+x_{n-1} = -\Delta t^2\omega_0^2 x_n$$

- We assume:

$$x_n = A\exp(-i\omega n\Delta t)$$

which leads to

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0\Delta t}{2}$$

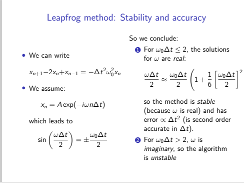
So we conclude:

1. For $\omega_0\Delta t \leq 2$, the solutions for ω are *real*:

$$\frac{\omega\Delta t}{2} \approx \frac{\omega_0\Delta t}{2}\left(1+\frac{1}{6}\left[\frac{\omega_0\Delta t}{2}\right]^2\right.$$

   so the method is *stable* (because ω is real) and has error $\propto \Delta t^2$ (is second order accurate in Δt).

2. For $\omega_0\Delta t > 2$, ω is *imaginary*, so the algorithm is *unstable*



Leapfrog method: Stability and accuracy

- We can write

$$x_{n+1} - 2x_n + x_{n-1} = -\Delta t^2 \omega_0^2 x_n$$

- We assume:

$$x_n = A \exp(-i\omega n \Delta t)$$

which leads to

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0 \Delta t}{2}$$

So we conclude:

1. For $\omega_0 \Delta t \le 2$, the solutions for ω are *real*:

$$\frac{\omega\Delta t}{2} \approx \frac{\omega_0\Delta t}{2}\left(1 + \frac{1}{6}\left[\frac{\omega_0\Delta t}{2}\right]^2\right)$$

so the method is *stable* (because ω is real) and has error $\propto \Delta t^2$ (is second order accurate in Δt).

2. For $\omega_0 \Delta t > 2$, ω is *imaginary*, so the algorithm is *unstable*

We insert $x_n = A \exp(-i\omega n \Delta t)$ into

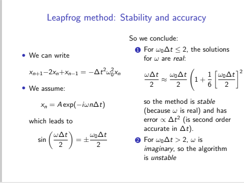
$$x_{n+1} - 2x_n + x_{n-1} = -\omega_0^2 \Delta t^2 x_n$$

to find

$$\begin{aligned}
-\omega_0^2 \Delta t^2 &= \exp(-i\omega\Delta t) + \exp(i\omega\Delta t) - 2 \\
&= 2\cos\omega\Delta t - 2 \\
&= -4\sin^2\left(\frac{\omega\Delta t}{2}\right)
\end{aligned}$$

which implies

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0\Delta t}{2}.$$



Leapfrog method: Stability and accuracy

- We can write

$$x_{n+1} - 2x_n + x_{n-1} = -\Delta t^2 \omega_0^2 x_n$$

- We assume:

$$x_n = A\exp(-i\omega n \Delta t)$$

which leads to

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0\Delta t}{2}$$

So we conclude:

1. For $\omega_0\Delta t \leq 2$, the solutions for ω are *real*:

$$\frac{\omega\Delta t}{2} \approx \frac{\omega_0\Delta t}{2}\left(1 + \frac{1}{6}\left[\frac{\omega_0\Delta t}{2}\right]^2\right.$$

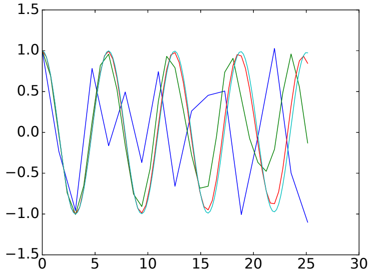
so the method is *stable* (because ω is real) and has error $\propto \Delta t^2$ (is second order accurate in Δt).

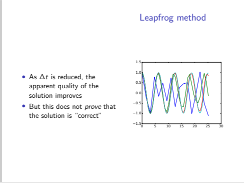
2. For $\omega_0\Delta t > 2$, ω is *imaginary*, so the algorithm is *unstable*

As long as ω is a pure real number, $\exp(-i\omega\Delta n)$ is a pure oscillatory function. So the period might not be correct, but the numerical solution will be oscillatory. If ω is not purely real (because $\omega_0\Delta t > 2$), then part of the solution is a growing exponential (because of the $\pm$). In this case, the solution grows without bound, which is transparently unphysical.

Leapfrog method

- As Δt is reduced, the apparent quality of the solution improves
- But this does not *prove* that the solution is "correct"



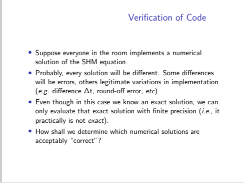

Leapfrog method

- As Δt is reduced, the apparent quality of the solution improves
- But this does not *prove* that the solution is "correct"

For illustrative purposes, some rather coarse time steps are included in the diagram. These are obviously not usefully accurate solutions.

Verification of Code

- Suppose everyone in the room implements a numerical solution of the SHM equation
- Probably, every solution will be different. Some differences will be errors, others legitimate variations in implementation (*e.g.* difference Δt, round-off error, *etc*)
- Even though in this case we know an exact solution, we can only evaluate that exact solution with finite precision (*i.e.*, it practically is not *exact*).
- How shall we determine which numerical solutions are acceptably “correct”?



Verification of Code

- Suppose everyone in the room implements a numerical solution of the SHM equation
- Probably, every solution will be different. Some differences will be errors, others legitimate variations in implementation (*e.g.* difference Δt, round-off error, *etc*)
- Even though in this case we know an exact solution, we can only evaluate that exact solution with finite precision (*i.e.*, it practically is not *exact*).
- How shall we determine which numerical solutions are acceptably "correct"?

An example of nominally identical calculations producing significantly different results was discussed in lecture 1. But this is a common experience whenever independently developed computer codes are compared. Clearly, in most such cases the problem under discussion is significantly more complex than the one we are investigating here.

The practice of comparing codes is often called "benchmarking." Generally speaking, benchmarking is not an acceptable verification procedure (although thinking otherwise is often tempting).

Verification of Code

- We can define a measure of the distance between the exact solution and any proposed numerical solution as:

$$L^2 = \sqrt{\frac{\sum_i \left(f_{\mathrm{numerical}} - f_{\mathrm{exact}}\right)^2}{\sum_i f_{\mathrm{numerical}}^2}},$$

  where the sum is over the points where $f_{\mathrm{numerical}}$ is defined. This is the L^2 **relative error norm**

- For our leapfrog method, we know:

$$f_{\mathrm{numerical}} - f_{\mathrm{exact}} \propto \Delta t^2$$

- Hence:

$$L^2 \propto \Delta t^2$$

Verification

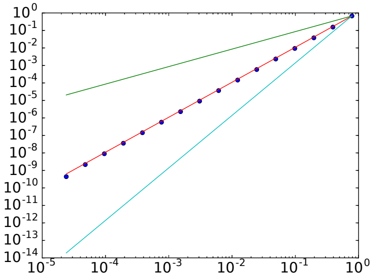
- If we plot L^2 against Δt, we will find

$$L^2 \propto \Delta t^2$$

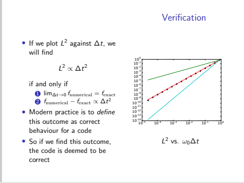
  if and only if

  1. $\lim_{\Delta t \to 0} f_{\mathrm{numerical}} = f_{\mathrm{exact}}$
  2. $f_{\mathrm{numerical}} - f_{\mathrm{exact}} \propto \Delta t^2$

- Modern practice is to *define* this outcome as correct behaviour for a code
- So if we find this outcome, the code is deemed to be correct



L^2 vs. $\omega_0 \Delta t$

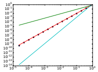


Verification

- If we plot L^2 against Δt, we will find

$$L^2 \propto \Delta t^2$$

  if and only if
  1. $\lim_{\Delta t \to 0} f_{\mathrm{numerical}} = f_{\mathrm{exact}}$
  2. $f_{\mathrm{numerical}} - f_{\mathrm{exact}} \propto \Delta t^2$
- Modern practice is to *define* this outcome as correct behaviour for a code
- So if we find this outcome, the code is deemed to be correct



L^2 vs. $\omega_0 \Delta t$

This is a powerful method of testing a code. What it unfortunately does not offer is any guidance on what mistake has been made, in the (alas common) event that the test fails.

The guidelines in the right hand figure show the expected outcome for solvers with accuracy $O(\Delta t)$, $O(\Delta t^2)$ and $O(\Delta t^3)$. The points come from a numerical solution of SHM equations using the leapfrog method. The expected behaviour, $L^2 \propto \Delta t^2$ is observed.

In practice, this behaviour will usually be observed only for some finite range of $\omega \Delta t$, because for very small values, the accuracy of the solution is limited by round-off error (or finite precision), and for large values, there are higher order contributions to the error.

Finding that the code under test converges faster than expected (*e.g.*, in this case, $L^2 \propto \Delta t^3$) is highly unlikely in practice. The opposite, a slower than expected rate of convergence, (*e.g.*, in this case, $L^2 \propto \Delta t$) is all too common, and indicates the implementation is inaccurate.

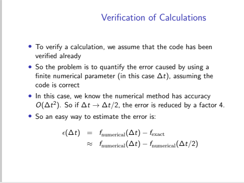
Verification of Code

- Convergence to an exact solution at the expected rate is the strongest possible test of a computer code
- To apply this method requires
  1. Understanding the numerical method (so we know the expected rate of convergence)
  2. Knowing an exact solution
- The second condition looks a big problem, but it need not be.

Verification of Calculations

- To verify a calculation, we assume that the code has been verified already
- So the problem is to quantify the error caused by using a finite numerical parameter (in this case Δt), assuming the code is correct
- In this case, we know the numerical method has accuracy $O(\Delta t^2)$. So if $\Delta t \to \Delta t/2$, the error is reduced by a factor 4.
- So an easy way to estimate the error is:

$$\begin{aligned}\epsilon(\Delta t) &= f_{\mathrm{numerical}}(\Delta t) - f_{\mathrm{exact}} \\ &\approx f_{\mathrm{numerical}}(\Delta t) - f_{\mathrm{numerical}}(\Delta t/2)\end{aligned}$$



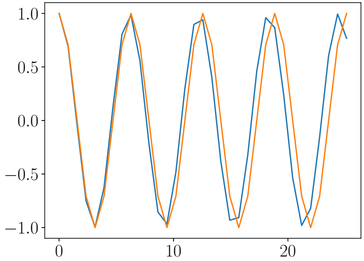
Verification of Calculations

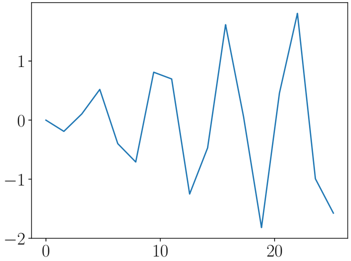
- To verify a calculation, we assume that the code has been verified already
- So the problem is to quantify the error caused by using a finite numerical parameter (in this case Δt), assuming the code is correct
- In this case, we know the numerical method has accuracy $O(\Delta t^2)$. So if $\Delta t \to \Delta t/2$, the error is reduced by a factor 4.
- So an easy way to estimate the error is:

$$\begin{aligned} \epsilon(\Delta t) &= f_{\mathrm{numerical}}(\Delta t) - f_{\mathrm{exact}} \\ &\approx f_{\mathrm{numerical}}(\Delta t) - f_{\mathrm{numerical}}(\Delta t/2) \end{aligned}$$

Clearly, $f_{\mathrm{numerical}}(\Delta t)$ means the numerical solution evaluated using time step Δt. Also clearly, the most accurate solution is $f_{\mathrm{numerical}}(\Delta t/2)$, whereas $f_{\mathrm{numerical}}(\Delta t) - f_{\mathrm{numerical}}(\Delta t/2)$ is an estimate of the error in $f_{\mathrm{numerical}}(\Delta t)$. So the error in $f_{\mathrm{numerical}}(\Delta t/2)$ should be much less. Hence this should be a robust error estimation.

Verification of Calculations



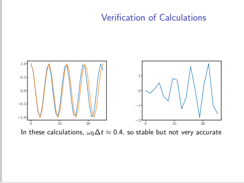


In these calculations, $\omega_0 \Delta t \approx 0.4$, so stable but not very accurate

2022-01-11

PS432/PS520 Computational Physics

└─Verification of Calculations



Verification of Calculations

In these calculations, $\omega_0 \Delta t \approx 0.4$, so stable but not very accurate

One can see here the implication of earlier analysis, that the error in the leapfrog integration method is mostly in phase, rather than amplitude.

Again, a crude calculation has been done for illustrative purposes.

A nonlinear example

- Consider:
$$\frac{d^2x}{dt^2} = -\omega_0^2\left(1+\alpha x\right)x$$
which we can attempt to solve using:
$$\begin{aligned}\frac{v_{n+1/2}-v_{n-1/2}}{\Delta t} &= -\omega_0^2\left(1+\alpha x_n\right)x_n \\ \frac{x_{n+1}-x_n}{\Delta t} &= v_{n+1/2}.\end{aligned}$$

- How will we verify our solver?
- A clever idea here is the **Method of Manufactured Solutions**
- The insight that informs this idea is that *our exact solution need not be physically significant*
- With this notion, we can construct a fake solution to verify against

Manufactured solutions

- Consider again:

$$\frac{d^2x}{dt^2} = -\omega_0^2\,(1+\alpha x)\,x + M(t)$$

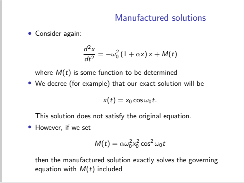
  where $M(t)$ is some function to be determined
- We decree (for example) that our exact solution will be

$$x(t) = x_0 \cos \omega_0 t.$$

  This solution does not satisfy the original equation.
- However, if we set

$$M(t) = \alpha \omega_0^2 x_0^2 \cos^2 \omega_0 t$$

  then the manufactured solution exactly solves the governing equation with $M(t)$ included



Manufactured solutions

- Consider again:

$$\frac{d^2x}{dt^2} = -\omega_0^2\,(1+\alpha x)\,x + M(t)$$

  where $M(t)$ is some function to be determined
- We decree (for example) that our exact solution will be

$$x(t) = x_0 \cos \omega_0 t.$$

  This solution does not satisfy the original equation.
- However, if we set

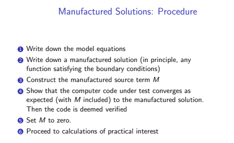
$$M(t) = \alpha \omega_0^2 x_0^2 \cos^2 \omega_0 t$$

  then the manufactured solution exactly solves the governing equation with $M(t)$ included

The Method of Manufactured Solutions was introduced by Roache [2001]. At first sight, this method looks like a sleight of hand, but it has been generally accepted. In general, of course, the manufactured source term can be a function of spatial variables as well as time, and we can conveniently choose any manufactured solution that satisfies the boundary conditions. In this case, for instance, we could sensibly have chosen any particular form of the general solution of the SHM equation. Recommended practice for complex cases involves the use of an algebraic assistant (such as the commercial Mathematica (`www.wolfram.com/mathematica/`) or the free Sage (`http://www.sagemath.org/`) to develop the manufactured source term.

Manufactured Solutions: Procedure

1. Write down the model equations
2. Write down a manufactured solution (in principle, any function satisfying the boundary conditions)
3. Construct the manufactured source term M
4. Show that the computer code under test converges as expected (with M included) to the manufactured solution. Then the code is deemed verified
5. Set M to zero.
6. Proceed to calculations of practical interest



Manufactured Solutions: Procedure

1. Write down the model equations
2. Write down a manufactured solution (in principle, any function satisfying the boundary conditions)
3. Construct the manufactured source term M
4. Show that the computer code under test converges as expected (with M included) to the manufactured solution. Then the code is deemed verified
5. Set M to zero.
6. Proceed to calculations of practical interest

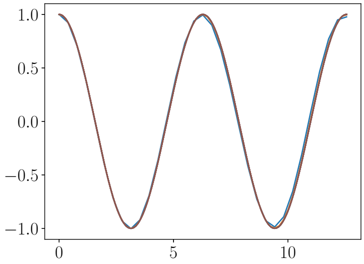
This procedure can look odd. But if the code under test was valid when $M \neq 0$ (which forces the manufactured solution), it cannot be rendered invalid by choosing $M = 0$ (which permits the solutions of practical interest).

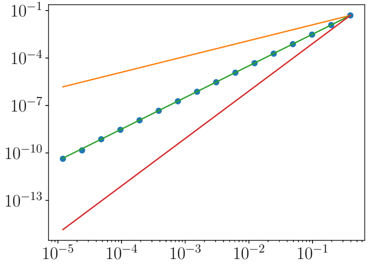
Good practice is to construct the manufactured solution from transcendental functions, as opposed to finite polynomials, because no numerical scheme of finite order can exactly represent a transcendental function. Hence we should always see convergence at the predicted rate (and not faster)

Manufactured Solutions: Summary

- The Method of Manufactured Solutions is a very general and powerful verification technique, adaptable to most (but not all) computer simulation techniques
- This example is simple, but the method can be applied to very complex cases (like Navier-Stokes)
- The main drawback is that one needs to be able to programme the manufactured source term $M(t)$. This involves having access to the source code of the simulation program, which is not always available

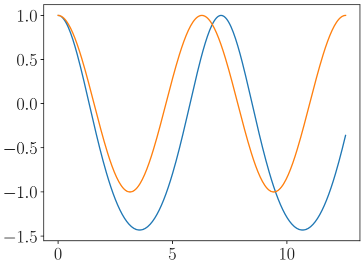
Nonlinear Example: Verification of Code

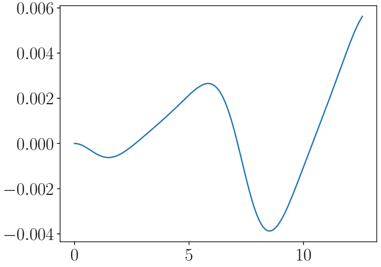




This example is for $\alpha = \frac{1}{2}$, $x_0 = 1$

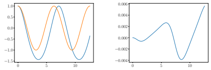
Nonlinear Example: Verification of Calculation





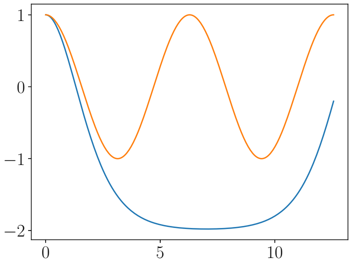
This example is for $\alpha = 0.4$, $x_0 = 1$. The right plot estimates the absolute error. The left plot shows both the nonlinear (blue) and linear solutions (orange).

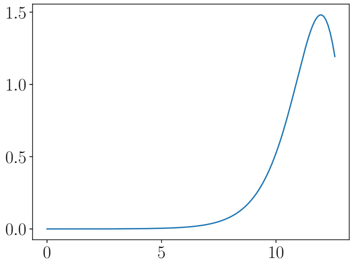
Nonlinear Example: Verification of Calculation


This example is for $\alpha = 0.4$, $x_0 = 1$. The right plot estimates the absolute error. The left plot shows both the nonlinear (blue) and linear solutions (orange).

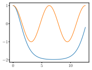
Whether this accuracy is acceptable depends on the aim of the calculation, which we didn't specify. But clearly, it's at least qualitatively correct. The error is estimated by comparing solutions with different time steps.

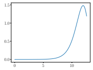
Verification of Calculation





This example is for $\alpha = 0.5$, $x_0 = 1$. The left figure shows the nonlinear oscillator solution (blue) with the linear solution (orange) for comparison.



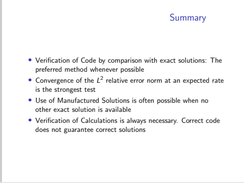


This example is for $\alpha = 0.5$, $x_0 = 1$. The left figure shows the nonlinear oscillator solution (blue) with the linear solution (orange) for comparison.

Something has gone wrong here. The error is large. It's not clear that this computation is fit for any purpose. Why? (The error is, again, estimated by comparing solutions with different time steps.)

Summary

- Verification of Code by comparison with exact solutions: The preferred method whenever possible
- Convergence of the L^2 relative error norm at an expected rate is the strongest test
- Use of Manufactured Solutions is often possible when no other exact solution is available
- Verification of Calculations is always necessary. Correct code does not guarantee correct solutions



- Verification of Code by comparison with exact solutions: The preferred method whenever possible
- Convergence of the L^2 relative error norm at an expected rate is the strongest test
- Use of Manufactured Solutions is often possible when no other exact solution is available
- Verification of Calculations is always necessary. Correct code does not guarantee correct solutions

There is a lot of lengthy and detailed writing on these matters, including two book length treatments [Roache, 2009, Oberkampf and Roy, 2010], and several long articles [Oberkampf and Trucano, 2002, 2008, Roy and Oberkampf, 2011]. The articles are free to access from the DCU network, and much shorter.

References I

W. L. Oberkampf and T. G. Trucano. Verification and validation in computational fluid dynamics. *Prog. Aerosp. Sci.*, 38(3): 209–272, 2002. URL `http://www.sciencedirect.com/science/article/pii/S037604`

William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, October 2010. ISBN 978-1-139-49176-1.

William L. Oberkampf and Timothy G. Trucano. Verification and validation benchmarks. 238(3):716–743, March 2008. ISSN 00295493. doi: 10.1016/j.nucengdes.2007.02.032. URL `http://linkinghub.elsevier.com/retrieve/pii/S002954930700`

References II

Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1): 4–10, November 2001. ISSN 0098-2202. doi: 10.1115/1.1436090. URL `http://dx.doi.org/10.1115/1.1436090`.

Patrick J. Roache. *Fundamentals of Verification and Validation*. Hermosa Publishers, Socorro, NM, September 2009. ISBN 978-0-913478-12-7.

Christopher J. Roy and William L. Oberkampf. A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Comput. Methods in Appl. Mech. Eng.*, 200(25-28):2131–2144, June 2011. ISSN 00457825. doi: 10.1016/j.cma.2011.03.016. URL `http://linkinghub.elsevier.com/retrieve/pii/S00457825110`