Contents

Chapter 1

Verification and Validation

Computer simulations are in many engineering applications a cost-efficient way for conducting design and performance optimalization of physical problems. However, thrusting blindly numbers generated from a computer code can prove to be naive. It doesn't take a lot of coding experience before one realizes the many things that can brake down and produce unwanted or unexpected results. Therefore, *credability* of computational results are essential, meaning the simulation is worthy of belief or confidence [2]. *Verification and validation* (VV) is the main approach for assessing and the reliability of computational simulations [9]. The terminology of (VV) have proven unconsistent across differnt engineering disciplines due to the variety of views regarding the fundaments of the method. A thorough review considering the development of (VV) concepts and terminology during the last century are studied in [2], where several attempts of definitions of *verification* and *validation* by different scientific communities are considered. In this thesis, the definitions provided by the *American Society of Mechanical Engineers guide for Verification and Validation in Computational Solid Mechanics* [8] are followed.

Definition 1.1. Verification: The process of determining that a computational model accurately represents the underlying mathematical model and its solution.

Definition 1.2. Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

Simplified *verification* considers if one solves the equations right, while *validation* is checking if one solves the right equations for the given problem. [5]

To test a computational code for all possible parameters, conditions and applications are simply too time consuming. Verification and validation are therefore ongoing processes, with no clear boundary of completeness unless additional requirements are specified [5].

The goal of this chapter is to verify our implementations using the method of manufactured solution (MMS).

1.1 Verification of Code

Verification can be devided into *verification of code* and *verification of calculation* [[4] [2]
Within scientific computing a mathematical model is often the baseline for simulations of a particular problem of interest. For scientists exploring physical phenomena, the mathematical model is often on the form of systems of partial differential equations (PDEt's). A computer program therefore must evaluate mathematical identities such a differential operators and functions in order to produce accurate solutions of the governing PDEt's. Through verification of code, the ultimate goal is to ensure a computer program truly represents the mathematical model, forming a basis for the given problem.
To accumulate sufficient evidence that a mathematical model is solved correctly by a computer code, it must excel within predefined criteria. If the acceptance criterion is not satisfied, a coding mistake is suspected. Should the code pass the preset criteria, the code is considered verified. Different acceptance criteria with increasing rigor are found in [5].

- Simple tests
- Code-to-code comparisons
- Discretization error quantification
- Convergence test
- Order-of-accuracy tests

The two first criteria have the advantage of complying in the absence of exact solutions, however not as rigorous as the three final criteria. *Simple tests* are applied directly on numerical solution, for example evaluating if the code preservers physical properties such as conservation laws. *Code-to-code comparisons* involves comparing the numerical solution of the code to another "reference code". However, the method proves useful only if the same mathematical models are used and the reference code have undergone rigorous verification [5].
The final three criteria *Discretization error quantification*, *Convergence test*, and *Order-of-accuracy tests* are all related to the discretization error E, defined as,

$$E = u_e - u_h$$

where u_e is the exact solution and u_h is the numerical solution at a given mesh and time step. Hence, an exact solution for the given problem is necessarily in order to evaluate the accuracy of the discretization of the mathematical model.
Discretization error quantification evaluates the quantitative error between the numerical solution and the exact solution, for a certain mesh resolution Δx and time step Δt. Even though error quantification surpass the previous criteria presented, the approache pose two major difficulties. First, what degree of spatial and temporal resolution are needed in order to reduce the discretization error E. Second, is

the subjective assessment of how small the error should be in order to give the code credability [5]. Since the previous criteria is, to some degree, bounded by subjective considerations, it lacks independence from the person conducting verification.

Convergence tests assumes the discretization of a PDE to be consistent, if the spatial ∇x and/or temporal ∇x refinement decreases to zero, etc $\nabla x, \nabla t \rightarrow 0.$, so does the discretization error $E \rightarrow 0$ [7]. The method is more consistent in comparison with discretization error quantification critera, as the choice of discretization parameters $\nabla x, \nabla t$ are not limited to one particular resolution. Even though the choice of temporal and spatial refinements are subjective, a reduction of the discretization error is expected for increasing temporal and spatial refinement. The method is considered the minimum criterion for rigorous code verification [5].
The final critera *Order-of-accuracy* is regarded as the most rigorous acceptance criterion for verification [7], [5], [1], which is employed in this thesis. In addition to error estimation and convergence of the numerical solution, the method ensure the discretization error E is reduced in coordinance with the *formal order of accuracy* expected from the numerical scheme. The formal order of accuracy is defined to be the theoretical rate at which the truncation error of a numerical scheme is expected to reduce. The *observed order of accuracy* is the actual rate produced by our code. The order of convergence is calculated xAssuming a PDE of space and time, order-of-accuracy tests are conducted separatly of
By monitoring the dicretization error E by spatial and temporal refinements, one assumes the asymptotic behavior,

$$E = E_x + E_t = u_e - u_h = Chx^p$$

where C is a constant, h represents the spatial or temporal resolution, and p is the convergence rate of the numerical sche me. The error E is assumed to be a sum of the discretization error of space and time.
Refinement is the act of solving the mathematical model for a set of discretization parameters h, such that $h_1 < h_2 < h_3..h_n \rightarrow 0$. Assuming a PDE discretized in space and time, the paramters takes the form $(\Delta x,\ \Delta t)$. When conducting *order-of-accuracy* tests, the convergence rate p is evaluated for temporal and spatial refinements separatly. Therefore, if one consideres spatial refinement study $\Delta x \rightarrow 0$, one must choose Δt small such that the temporal discretization error can be exluded from the total error E. For convergence tests, the code is assumed verified and consistent if the discretization error is proportional to h^p

1.1.1 Method of manufactured solution

For conducting verification of code based on convergence tests, an exact solution of the mathematical model or PDE is needed in order to evaluate the discretization error E. However accurate solutions of PDEt's are limited, and often simplifications of the original problem are needed in order to produce analytically solutions for comparison. *The method of manufactured solutions* provides a simple yet robust

way of making analytic solutions for PDEt́s. The problem is not attacked in a traditional way, by finding an analytic solution by retaining the overall physics defined by the model.
Let a partial differential equation of interest be on the form

$$\mathbf{L}(\mathbf{u}) = \mathbf{f}$$

Here $\mathbf{L}$ is a differential operator, $\mathbf{u}$ is variable the of interest, and $\mathbf{f}$ is some sourceterm.
In the method of manufactured solution one first manufactures a solution $\mathbf{u}$ for the given problem. In general, the choice of $\mathbf{u}$ will not satisfy the governing equations, producing a sourceterm $\mathbf{f}$ after differentiation by $\mathbf{L}$. The produced source term will cancel any imbalance formed by the manufactured solution $\mathbf{u}$ of the original problem. Therefore, the manufactured solution can be constructed without any physical reasoning, proving code verificaion as a purely a mathematical exercise were we are only interested if we are solving our equation right [4]. Even though the manufactured solution $\mathbf{u}$ can be chosen independently, some guidelines have been proposed for rigirous verification([1], [7], [4]).

- The manufactured solution (MS), should be composed of smooth analytic functions such as exponential, trigonometric or polynomials.
- The MS should exercise all terms and derivatives of the PDEt́s.
- The MS should have sufficient number of derivatives

The guidelines presented are not limitation for choosing a MS, but rather improvements for ensuring the representation of the mathematical model is thoroughly tested.
Order-of-accuracy test or *order-of-convergence test* is the most rigorous code I wrote this to check if this works
To deeply verify the robustness of the method of manufactured solution, a report regarding code verification by MMS for CFD was published by Salari and Knupp [7]. This thorough work applied the method for both compressible and incompressible time-dependent Navier-Stokes equation. To prove its robustness the authors deliberate implemented code errors in a verified Navier-Stokes solver by MMS presented in the report. In total 21 blind testcases where implemented, where different approaches of verification frameworks were tested. Of these, 10 coding mistakes that reduces the observed order-of-accuracy was implemented. Here the method of manufactured solution captured all of them.

For construction of the sourceterm $\mathbf{f}$ the Unified Form Language (UFL) [3] provided in FEniCS Project will be used. UFL provides a simple yet powerfull method of declaration for finite element forms. An example will be provided in the Fluid Problem section.

1.1.2 Verification of the fluid-structure interaction solver by MMS

In the last section, the choice MMS for verification of code is stressed to not be limited by the overall physics of the given problem of interest, which is true considering the fluid and solid problems separate. The freedom of choosing a MMS for a monolithic fluid-structure interaction problem is however limited to some extent, due coupling coditions at the interface. Recall from chapter 4,

1. Kinematic boundary condition $\hat{\mathbf{u}}_s = \hat{\mathbf{u}}_f]$, enforced strongly by a continious velocity field in the fluid and solid domain.

2. Dynamic boundary condition $\sigma_s \cdot \mathbf{n} = \sigma_f \cdot \mathbf{n}$, enforced weakly by omitting the boundary integrals from the weak formulation in problem ?.

The construction of a MMS is therefore not obvious, as it must fulfill condition 1 and 2, in addition to the divergence-free condition in the fluid. The struggle is reflected of the abscence of research, regarding MMS for coupled FSI solvers in the litterature. The challenge are often disregarded, such as cite METHOLOGY for COMPARING.., presenting a thorough methodology for comparing fluid-structure interaction solvers based on the verification and validation framework. The verification process is conducted on the fluid and structure solver separatly, taking no consideration to the coupling. Instead, the correctness of the coupling is evaluated by the code validation.
The approach clearly ease the process, assuming verification of each codeblock is "sufficient" to declare the code verified. It must be stressed that solving each problem individually is not true verification, in reference to a monolithic approach where the problems are solved at the same time.
The construction of a MMS for a monolithic FSI problem is therefore out of the scope of this thesis. Conducting verification on the fluid and structure separatly is not , but considered "good enough" to show the mathematical model is discretized accuratly.

1.2 Validation

Through *verification*, one can assure that a scientific code implements a mathematical model correctly. However, c orrectness is unnecessarly, if the model fails to serve as an accurate representation of the physical problem of interest. By definition 1.2, *Validation* is the act of demonstrating that a mathematical model is applicable for its intended use with a certain degree of accuracy. This demonstration is not intended to portray the model as an absolute truth, nor the best model available [6]. The acceptence criteria of validation is based on the numerical solution produced, by comparison with existing experiment data. The dependency of thrusting experiments, makes *validation* assess a wide range of issuses [9]

- The relevance of mathematical model compared to the experiment.

- Assurence of that the experiments was conducted correctly, in accordinance with prescribed parameters, initial and boundary conditions e.t
- Uncertainty of experimental measurements

Comparing numerical results with existing experiments,raise some issues in the validation process. First, reproducing of experimental resuts...

Bibliography

[1] Stéphane Étienne, D Tremblay, and Dominique Pelletier. Code Verification and the Method of Manufactured Solutions for Fluid-Structure Interaction Problems. *36th AIAA Fluid Dynamics Conference and Exhibit*, (June):1–11, 2006.

[2] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing.* Cambridge University Press, Cambridge, 2010.

[3] Fenics Project. Unified Form Language (UFL) Documentation. 2016.

[4] Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1):4, 2002.

[5] P.J. Roache. *Verification and Validation in Computational Science and Engineering.* Computing in Science Engineering, Hermosa Publishers, 1998, 8-9, 1998.

[6] Edward J. Rykiel. Testing ecological models: The meaning of validation. *Ecological Modelling*, 90(3):229–244, 1996.

[7] Kambiz Salari and Patrick Knupp. Code Verification by the Method of Manufactured Solution. Technical report, Sandia National Laboratories, 2000.

[8] LE Schwer. Guide for verification and validation in computational solid mechanics. *American Society of Mechanical Engineers*, PTC 60(V&V 10):1–15, 2006.

[9] Ian Sommerville. Verification and Validation. Technical Report February, 2006.