Minds and Machines (2019) 29:149–168
https://doi.org/10.1007/s11023-019-09493-8

Verification and Validation of Simulations Against Holism

Julie Jebeile[1] · **Vincent Ardourel**[2]

Received: 14 May 2018 / Accepted: 20 February 2019 / Published online: 22 April 2019

Abstract
It has been argued that the Duhem problem is renewed with computational models since model assumptions having a representational aim and computational assumptions cannot be tested in isolation. In particular, while the Verification and Validation methodology is supposed to prevent such holism, Winsberg (Philos Compass 4:835–845, 2009; Science in the age of computer simulation, University of Chicago Press, Chicago, 2010) argues that verification and validation cannot be separated in practice. Morrison (Reconstructing reality: models, mathematics, and simulations, Oxford University Press, Oxford, 2015) replies that Winsberg overstates the entanglement between the steps. The paper aims at arbitrating these two positions, by stressing their respective validity in relation to domains of application. It importantly argues for an increasing use of formal methods in verification, that makes disentanglement possible.

Keywords Scientific models · Computer simulations · Verification and validation · Duhem problem · Holism · Formal methods

1 Introduction: Duhem Problem

The Duhem problem states that a single theoretical hypothesis cannot be tested empirically in isolation, but is tested with auxiliary hypotheses, e.g., hypotheses about measurement instruments functioning. The model-oriented version of this problem has recently been widely discussed (e.g., Frigg and Reiss 2009; Lenhard

This work was supported by "MOVE-IN Louvain" Incoming Post-doctoral Fellowship, cofunded by the Marie Curie Actions of the European Commission.

✉ Julie Jebeile
julie.jebeile@gmail.com

Vincent Ardourel
vincent.ardourel@gmail.com

[1] Institut supérieur de philosophie, Université catholique de Louvain, Louvain-la-Neuve, Belgium

[2] IHPST, CNRS/Université Paris 1 Panthéon-Sorbonne, Paris, France

and Winsberg 2010; Winsberg 2010; Jebeile and Barberousse 2016; Lenhard 2018). It states that a model's failure to match the available data means that something must be wrong within the model assumptions, but the blameworthy assumption(s) cannot straightforwardly be identified. This is more than a "problem about falsification—about where to assign blame when things go wrong" (Winsberg 2009, p. 839; 2010, p. 24), since, conversely, when a model's outputs agree with data, it might be because adjustments cancel out effects of model uncertainties and numerical errors.

In this paper, we focus on the specificity of the Duhem problem when applied to computational models. In this case, model assumptions are comprehensively tested, including not only the assumptions that have a representational aim, i.e., theoretical principles and simplifying hypotheses, but also the assumptions related to the computational techniques, such as the discretisation of equations. Therefore, "when a computational model fails to account for real data, we do not know whether to blame the underlying model or to blame the modeling assumptions used to transform the underlying model into a computationally tractable algorithm" (Winsberg 2009, p. 839; 2010, p. 24). A specific form of holism thus appears since computational assumptions can interfere with assumptions that have a representational aim.

Yet there is a methodology in computational modeling that is supposed to prevent such holism, viz. Verification and Validation (V&V) (Oberkampf et al. 2002). Verification is a mathematical-oriented step: it ensures that numerical errors do not affect significantly the model outputs. Validation is a physical model-oriented step: it ensures that the model outputs are in agreement with empirical data. However, it has been argued that there is still an entanglement between verification and validation (Winsberg 2010; Lenhard 2018). Scientists indeed would not succeed in keeping separate the two steps. Nevertheless Morrison (2015) disagrees with this view, and claims that Winsberg overstates the entanglement between verification and validation.

The paper aims at arbitrating these two positions, by stressing their respective validity in relation to domains of application. We first introduce the V&V methodology (Sect. 2), and review Winsberg's account for the entanglement of verification and validation (Sect. 3), as well as Morrison's view (Sect. 4). We then endorse Morrison's view by arguing for an increasing use of formal methods in verification since the 1990's, which makes disentanglement possible (Sect. 5). We make it clear that formal methods are mostly used in critical domains such as aerospace systems, defense or domains involving security requirements (Sect. 6). On the other hand, we admit that Winsberg's view applies to a large range of computational models, though not to *all* models. In other words, we suggest that there is a range of possibilities, going from separability to entanglement in V&V, that highly depends on domains of application (Sect. 7).

2 Computational Models and V&V Methodology

This section is devoted to define the notion of models used in the paper, and then to introduce the V&V methodology. First of all, models refer to "analytical models" that are written and solved manually, or to ''computational models'' that are written

for, and solved by, computers. Models contain conceptual components that describe a target system's properties and basic behaviors, as well as simplifying assumptions that make the model equations numerically tractable. In particular, computational models (or *simulation models*) contain the required approximations related to the numerical scheme, i.e., the numerical methods, for the resolution of the equations on the computer.

In discretization-based numerical methods, computational models are discrete versions of the initial continuous model equations. They generate discretization errors, which are produced when continuous variables (such as time and space) are replaced by a discrete set of values.[1] Once implemented on a computer, computer round-off errors are also generated, which are due to the finite memory of computers.[2] Ultimately, scientists want to discern whether the conceptual part of the model is an accurate representation of the target system for the purpose at hand. However, this can only be determined via the computational model entirely. Thus, when they conduct a validation after verification here, they are not strictly validating the model assumptions which are purportedly being tested, but the computational model.

For the computational assumptions not to interfere with the assumptions having a representational aim, sanctioning the model (being either analytical or computational) must proceed in two distinct steps. The first step consists in testing whether the mathematical equations of models are correctly solved: it boils down to checking if the solutions are exact or exact enough. The second step consists in confirming or invalidating the conceptual part of models by verifying that the exact solutions to their equations fit with the experimental data on the natural or social systems under study. If these two steps are not performed distinctively, one after the other, it is difficult to assess the adequacy of a model.

Verification and Validation (V&V) has recently received increasing philosophical scrutiny (Lenhard and Winsberg 2010; Winsberg 2010; Oreskes et al. 1994; Morrison 2014, 2015; Fillion 2017). At first sight, V&V seems to meet this requirement when sanctioning simulation models. First of all, as its name suggests, V&V has two steps, i.e., verification and validation. A standard definition states that "verification [is] the process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model" while validation is defined as "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" (Oberkampf and Trucano 2002, p. 14).

Furthermore verification seems to ensure that numerical errors related to the numerical scheme do not affect significantly the model outputs in the first place, before the model outputs are compared with empirical data in validation. Concretely, the aim of verification is to check whether the computer code works fine in

[1] We can also distinguish ''truncation errors'', which are created by the discretization of equations (when one transforms the differential equations into approximate algebraic equations).

[2] In other numerical methods, such as cellular automata or agent-based models, there are no discretization errors. We will focus on discretization-based numerical methods in this paper.

calculating approximate solutions to the model equations, while the aim of validation is to check whether these solutions match the available experimental data.

The verification step aims to quantify the shift between the computer code and the conceptual part of model of which the code is the implementation. This shift corresponds to discrepancies between the approximate solutions provided by the computer code and the solutions that would have been ideally obtained if one had been able to perform the calculations exactly. More precisely, verification proceeds in two sub-steps:

1. Overall, code verification aims to justify that the code is appropriately implemented within the hardware (i.e., architecture, memory and operating system of the computer) and system software, that all its functions work and that it will not yield wrong predictions for mere computer software reasons (e.g., algorithm error, bug, convergence problem or problem of existence and uniqueness of solutions).
2. Solution verification is about assessing whether the solutions obtained by the simulation are consistent with model assumptions; in other words, whether they are good approximate solutions to the equations.

As for the validation step, it consists mainly in comparing a target set of numerical results, either directly with a database of experimental measurements, or with a set of results obtained with other codes which have already been validated. These latter are known as benchmarks and are useful to overcome the lack of experimental measurements.

For these reasons, V&V could be seen as a way of overcoming the Duhem problem. However, as we will see in the next section, important philosophical arguments have been developed that question V&V's success (Winsberg 2010; Lenhard 2018).

3 Entanglement in V&V

A first set of arguments that supports entanglement in V&V is given by Winsberg (2009, 2010, chapter 2). According to him, entanglement in V&V remains because scientists do not succeed in providing, in the verification step, strong arguments establishing that the obtained simulation outputs approximate the exact solutions to the original differential equations. Therefore, scientists go to the validation step without having completely checked the verification step. Numerical errors can thus be entangled with pure modeling errors.

First of all, because verification is expressed as a mathematical issue or a computer science question, strong arguments are supposed to come from mathematical results and computer science results only. However, Winsberg claims that ''when models are sufficiently complex and non-linear, it is rarely possible to offer mathematical arguments that show, with any degree of force, that verification is being achieved'' (2009, p. 838); and elsewhere, "simulationists are rarely in the position of being able to establish that their results bear some mathematical relationship to an antecedently chosen and theoretically defensible model" (2010,

p. 20). Simulationists thus do not have the mathematical resources to ensure that, for any initial values, any values of parameters, simulation outputs will be close to the exact solutions.

Second, in practice, simulationists use strategies to argue for the reliability of simulation outputs instead of providing strong mathematical arguments:

> What simulationists are forced to do is to focus, instead, on establishing that the combined effect of the models they begin with, and the computational methods they employ, provide results that are reliable enough for the purposes to which they intend to put them. If we are simulating the global climate, it is almost certain that we will not be able to establish that our results bear any mathematical relationship to the ideal model of the climate. (Winsberg 2009, p. 838)

Nonetheless, while such strategies offer grounds for believing a simulation provides reliable information about the target phenomenon, they fail to provide grounds that the computer programme correctly provides solutions to the original equations. They mainly aim at demonstrating that simulations correctly reproduce known analytical results to the original equations (under constrained conditions), other simulation outputs at hand and/or, most importantly, available real-world data (within what Winsberg considers to be "benchmarking"). Other strategies are also based on comparing simulation outputs to background knowledge, e.g., expected responses of the system to changes in parameter values, basic functional relationships, or phenomenological laws.

Thus, according to Winsberg, (1) mathematical arguments that simulationists can offer are very weak, and (2) their strategies aim rather at providing grounds for belief that a simulation provides reliable information about the target phenomenon. It follows that:

> The sanctioning of simulations does not cleanly divide into verification and validation. In fact, simulation results are sanctioned all at once: simulationists try to maximize fidelity to theory, to mathematical rigor, to physical intuition, and to known empirical results. But it is the simultaneous confluence of these efforts, rather than the establishment of each one separately, that ultimately gives us confidence in the results (Winsberg 2010, p. 23).

Winsberg's thesis is supposed to apply to a wide range of scientific practices in which simulations are used, since his work (e.g., 2009, 2010, 2018) is based on practices in climate science, cosmology, computational fluid dynamics used in engineering contexts, multi-scale nanoscience, fluid-dynamical astrophysics. We shall nevertheless discuss the generality of his thesis later on (Sect. 7).

More recently an additional argument has been provided by Lenhard (2018) which complements Winsberg's arguments. Particular emphasis is here put on the adjustable parameters in simulation models. For Lenhard, these parameters cannot be kept separate from the model form. They "also belong to the model form, because without assignment of parameters neither the question about representational adequacy nor the question about behavioral fit can be addressed". In other

words, parameters are part of the representational content in models, so that it makes no sense to consider a simulation model as such without its parameter values being already assigned. And yet it is true that model parameters need to be at least set, i.e., assigned with some numbers, before the verification step be actually performed. As such numbers should preferably be carefully chosen from the start, it follows that adjustment of parameters should precede the verification step. Therefore "It is not possible to first verify that a simulation model is 'right' before tackling the 'external' question whether it is the right model". For Lenhard, this makes the separation between verification and validation impossible in practice.

Yet this situation is representative of the kind of back-and-forth adjustments in the verification and the validation of computational models in numerous scientific domains. It is indeed quite usual for scientists to start again verification after having already conducted a verification and a validation. It may nevertheless not follow from this that the two processes cannot be distinguished at the end of the day, in that the arguments used in the verification step could still be independent from the arguments used in the validation step that are mainly based on empirical comparisons. In the case developed by Lenhard, there is some back-and-forth between the verification and the validation so to establish the adequate values to the parameters, but, once these values are set, this back-and-forth may not be a reason for verification and the validation cannot be performed again in a distinctive manner.

4 Mathematical Arguments in Verification

A claim against Winsberg has been developed by Morrison (2015, chapter 7), which highlights the mathematical arguments that scientists can provide during the verification step so that the two phases of V&V are not doomed to be entangled. We complete her demonstration by providing details about those mathematical arguments.

In the first place, Morrison makes clear that V&V is not thought by practitioners to be foolproof. V&V is actually designed by them specifically as a variety of different techniques used to combat various types of errors and to provide evidence for legitimating simulation results. Moreover practitioners are aware that they cannot rely solely on mathematical arguments, and that the mathematical aspects of verification and the more empirical aspects of validation are not completely independent. Obviously, this dependency lies not only in a validation's reliance on a numerical scheme that was deemed accurate during verification, but also in the verification process's assumption that the model is a correct representation.

First, for Morrison, it seems like an exaggeration to claim that the mathematical arguments provided in verification are very weak. She interprets that the main reasons Winsberg gives are the various failures encountered historically by numerical methods.[3] However, she replies that "The fact that errors have been made does not,

[3] As pointed out by one of the anonymous reviewers, it is not clear that past errors are actually considered by Winsberg as reasons for weakness in V&V. That said, the following objections of Morrison hold insofar as Winsberg does argue that the usual given mathematical arguments are weak, and that strategies for sanctioning a simulation aim at providing grounds for belief that the simulation is reliable.

in itself, mean that numerical methods are inherently problematic" (2015, p. 267). As she later specifies (p. 286), such errors can indeed be incidences of miscalculation, human error and the acceptance of theories, models and methods that later proved inaccurate or unjustified.

Second, she claims that "Winsberg's characterisation ignores crucial aspects of the methodology and in doing so presents a distorted view of its goals and methods" (p. 268). It follows for her that Winsberg's conception of V&V is a piecemeal activity—close to the way experimenters scrutinize experimental setups to uncover possible sources of artifact—made of ad hoc strategies, whose focus is more on successful results rather than on methodological justifications. And yet she argues that V&V has a structured and yet evolving methodology which aims at providing justifications following "well-developed techniques in place that not only address specific types of difficulties but reveal just how important the sequential nature of V&V is" (p. 268).

Morrison claims that "verification requires the identification as well as the demonstration and quantification of errors, together with establishing the robustness, stability, and consistency of the numerical scheme" (p. 263). She introduces the kinds of justifications given in code verification and solution verification.

Code verification concerns detecting possible implementation mistakes and errors in software. This usually consists in comparing computational solutions with exact solutions (i.e., closed forms solutions) to the original equations, or with highly accurate solutions. Since exact solutions are generally difficult to obtain, and are often derived from simplified problems that do not exercise the full functionality of a code, the Method of Manufactured Solutions is often used instead. With this method, a solution is first defined that aims at testing relevant parts of the code, and then a problem is constructed, that is described by a modified version of the original equations that the chosen solution satisfies.

Solution verification is about providing estimations of numerical errors; in cases of highly confident estimations, errors can be removed from the numerical solutions. That said, Morrison concedes that estimations of discretisation errors are particularly complex so that these errors are more commonly associated with epistemic uncertainty.

Morrison does not indicate particular estimation methods nor the way robustness, stability, and consistency of the numerical scheme are established. Therefore, we now complete her argument in explicating the important concepts of consistency and stability. Both are a priori requirements. Satisfying them is a first mandatory step of providing, in verification, a priori justifications for the reliability of numerical methods, as Oberkampf and Trucano argue (2002, p. 32). A priori justifications require no empirical data nor other numerical results to compare; by providing a priori justifications, verification is performed without any appeal to past successes of the code or any agreement during benchmark tests.

Let us call u_{exact} the exact solution of a PDE and $v_{h,tau}$ the discrete solution of the numerical scheme after one step of space h and time tau. The difference $\|u_{exact} - \mathrm{v}_{h,tau}\|$ is called the discretization *error* or the consistency *error*. It is local information about how the discrete solution differs from the exact one after a single discrete space–time step. Convergence error is defined as the difference maximum

between the exact solution and the discrete solution after N space and time steps of computations, i.e., $\max 1 < k, i < N \, ||u_{exact} - v_{h,tau}^{k,i}||$. Unlike discretization error, it is global information since it deals with N steps of computations. A numerical scheme is convergent if and only if $max1 < k, \, i < N \, \|u_{exact} - v_{h,tau}\|$ tends to zero when both h and tau tend to zero.

Consistency is the first a priori property required for a numerical scheme. A numerical scheme is called (strongly) consistent if and only if the discretization error $\|u_{exact} - v_{h,tau}\|$ tends to zero when both h and tau tend to zero."Discretizations that are not strongly consistent will not converge, and the three major questions for empirical performance assessment will not make sense." (Oberkampf and Trucano 2002, p. 34). These three questions are (Oberkampf and Trucano 2002, p. 31):

1. Does the discrete solution converge to the exact solution as the mesh spacing is reduced in real calculations?
2. What is the effective order (the observed values of and) of the discretization that is actually observed in calculations?
3. What is the discretization error that is actually observed for real calculations on finite grids?

Consistency is a necessary but not sufficient condition for solution convergence (Oberkampf and Trucano 2002, p. 35). Stability is the second a priori property required for a numerical scheme. It receives different senses, depending on contexts (in systems sensible to initial conditions, in numerical computations, etc.). Here, it means that the discrete solution $v_{h,tau}^{k,i}$ is bounded by a constant C for any k and i $1 < k, \, i < N$. Unlike consistency, it is not a property related to the exact solution, but a property of the numerical scheme.

If *stability* is met in addition to consistency, then it can be proved, in some cases, that the numerical scheme will be convergent. Furthermore, in numerical analysis, the Lax theorem guarantees that, as soon as the numerical scheme is based on a linear finite difference model, consistency and stability are necessary and sufficient conditions for convergence (Trefethen 1994, chapter 4, p. 153; Oberkampf and Trucano 2002, p. 35, 36).

In a nutshell Morrison argues that Winsberg overstates the entanglement between verification and validation. Scientists look for a priori justifications in verification. They do that either by fixing conditions (consistency, stability) or by developing specific error analyses.[4] Obviously, such methods do not always succeed in all practical cases, but it means at least that holism does not concern all domains of application.

We now argue that formal methods are a recent and promising method for ensuring that there is no uncontrollable numerical errors in a computational model, a method that may well apply to more and more scientific domains. Formal methods in verification have not yet been explored in philosophy of science, and yet constitute serious attempts at overcoming holism.

[4] There is a discussion on a priori arguments and rigorous error analyses of computational methods in Fillion (2017).

5 Verification with Formal Methods

The development of formal methods since the 1960's by mathematicians, logicians and computer scientists, and their wide dissemination in industry strongly support our claim that the disentanglement between verification and validation is a clear (and more and more successful) objective for computer scientists and engineers. These methods are called *formal methods*.

Formal method is a set of verification activities that are therefore independent from any validation activities. In other words, once a model is formally verified, it still needs to be validated, i.e., scientists still need to assess whether its results agree with the available empirical data. We want to show here that formal methods follow the rigorous means of providing strong mathematical arguments for the verification phase.

The research is still ongoing, but what exists already provides strong evidence that formal methods are widely used in practice at the verification step. In a nutshell, formal methods are methods based on logic and discrete mathematics that guarantee that software and hardware designs achieve their intended end. We will notably discuss in Sect. 5.3 a case for which formal methods are used to verify a computational model. Section 6 will then be devoted to discuss the extent of the use of formal methods.

The development and use of formal methods originate from very practical reasons. The lack of verification of a program can lead to costly and dramatic consequences. For example, on 4 June 1996, the flight of the Ariane 5 launch exploded 40 s into flight, incurring a cost of hundreds of millions of dollars (Dowson 1997). The cause has been identified as a software error. At some point, the data conversion from 64-bit floating point to 16-bit signed integer value led to a value greater than what could be represented by a 16-bit integer (Lions 1996). This is the beginning of a chain of events that caused the explosion of Ariane 5. We could also report the dramatic death of 28 servicemen on the 25 February 1991 when a Patriot missile failed to intercept an Iraqi Scud missile. This failure was also due to a software error. It was a roundoff error caused by a fixed-point 24-bit representation of 0.1 in base 2 (Skeel 1992, p. 11). In 2002, the Department of Commerce's National Institute of Standards and Technology (NIST) estimated that software bugs or errors cost $59.5 billion annually to the US economy.[5] As suggested by the NIST, rigorous certification is a key tool for tackling these problems: ''The path to higher software quality is significantly improved software testing. Standardized testing tools, suites, scripts, reference data, reference implementations and metrics *that have undergone a rigorous certification process* would have a large impact on the inadequacies currently plaguing software markets'' (*ibid*., our emphasis).

We first introduce formal methods before discussing their use in practice.

[5] http://www.abeacha.com/NIST_press_release_bugs_cost.htm.

5.1 What are Formal Methods?

Formal methods are a complete research field with a scientific community, scientific books, peer-reviewed journals, international workshops, societies and so on; journals are, for instance, *Formal Aspects of Computing* or *Formal Methods in System Design*.[6] They consist of a large range of rigorous methods based on logic and discrete mathematics. It is hard to define them since they cover different domains, from hardware design to software checking, and are based on different principles, logics, and implemented in different ways, including different programming languages. Let us refer to how R. W. Butler (2001) defines them on NASA's website:

> "Formal Methods" refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. The phrase "mathematically rigorous" means that the specifications used in formal methods are well-formed statements in a mathematical logic and that the formal verifications are rigorous deductions in that logic (i.e. each step follows from a rule of inference and hence can be checked by a mechanical process.) The value of formal methods is that they provide a means to symbolically examine the entire state space of a digital design (whether hardware or software) and establish a correctness or safety property that is true for all possible inputs.[7]

The rigor of formal methods comes from a twofold requirement (see Wiels et al. 2012, p. 2). On the one hand, formal methods use *formal notations*, which are non-ambiguous with logically and mathematically defined syntax and semantics. On the other hand, formal methods allow *automated computation* based on different programming languages.

There are many kinds of formal methods. While impossible to introduce them extensively, we can stress three main families of formal analysis techniques. First, *model checking* consists of exhaustively exploring the possible states of a program to know if it meets a given specification or property (Clarke 2008). For instance, it can be used to guarantee that a program will never meet some given critical states. *Deductive methods*, based for instance on Hoare's logic, or Dijkstra's calculus, allow to automatically establish the correctness of a program or prove some specified properties. Finally, a*bstract interpretation* is a theory that pertains to the semantics of computer programs and which can be interpreted as allowing to extract information about a program without performing all the calculations.

To better grasp what formal methods are, let us introduce an example referring to one the first formal methods developed in the 1960's by Hoare. Let us assume that we want to compute n^m where n and m are non-negative integers (for details, see Rushby 1995, p. 23). We can use the following algorithm based on a 'endwhile' loop: r := 1, i := m; while i =/0 do r := r * n; i: = i − 1; endwhile.

[6] For an overview of the scientific community of formal methods, see http://formal.epfl.ch/.

[7] https://shemesh.larc.nasa.gov/fm/fm-what.html.

At the verification step, a computer scientist might want to question whether this program does actually compute correctly n^m. Even if it seems obvious that this program does compute n^m since it repeats m times the multiplication of n, formal methods aim at proving it sequentially within a framework that can be automatized. Such an approach is very helpful for programs with thousands of code lines. The correctness of programs can be proved within the Hoare logic. It consists of manipulating sentences of the form (P)S(Q) where P and Q are expressions that describe the relationships between the program variables, and S a piece of program text. In our example, (P) can be ($r \times n^i = n^m$ and $i = /0$), (Q) ($r \times n^i = n^m$ and $i = /0$), and S r: + r*n; $i := i - 1$. The interpretation of (P)S(Q) is that if P is true before S, and if S terminates, Q will be true afterwards. There is a series of axioms within Hoare's logics that allow a sequential analysis of the algorithm. Although it is beyond the scope of this paper to present the entire proof of the correctness of this algorithm (for details, see Rushby 1995, p. 23), we want to assert the definitive existence of formal approaches, here based on deductive methods, developed to prove the correctness of algorithms and programs.

5.2 The Increasing Use of Formal Methods

We now turn to the question of the use of formal methods in practice. To begin with, we stress that the wide use of these formal methods is quite recent, which explains why philosophers of science have neglected them so far.

In the 90's, John Rushby, the author of *Formal Methods and their Role in the Certification of Critical Systems*, emphasized that "[r]eaders who found their eyes glazing over at the formulas used to verify the trivial exponentiation program may wonder whether these formal techniques really are practical, and might ask 'how am I going to get my engineers to use this stuff?'. Privately, they may also wonder 'if this stuff is so good, why isn't it used more?'" (1995, p. 22). Although Rushby acknowledges a slow industrial use of formal methods in the 90's, he claims that this is not due to their ineffectiveness but to the field's youthfulness. Because of an understandable conservatism in engineering, formal methods were slowly accepted in industry. Rushby argued in favor of the use of these methods, which has to depend on the reliability requirements of programs. In particular, since "the practicality and cost/benefit of formal methods are heavily dependent on the type of applications considered (…). My opinion is that the greatest benefits are likely to be found when formal methods are applied to the hardest and most difficult problems". In the 1997 Proceedings of the 16th Digital Avionics Systems Conference, Michael Holloway also tried to convince engineers of the interest of these methods in a paper entitled "Why engineers should consider formal methods?". He stressed the gap between the theoretical computer scientists for whom "the efficacity of formal methods is now accepted as proved" and "the attitude of the best minds among practicing engineers (which) has been quite different, with far more rejecting formal methods than embracing them" (p. 1). To sum up, despite theoretical progress, formal methods remained generally unused by the end of the 90's as specialized computer scientists failed to convince the broader community.

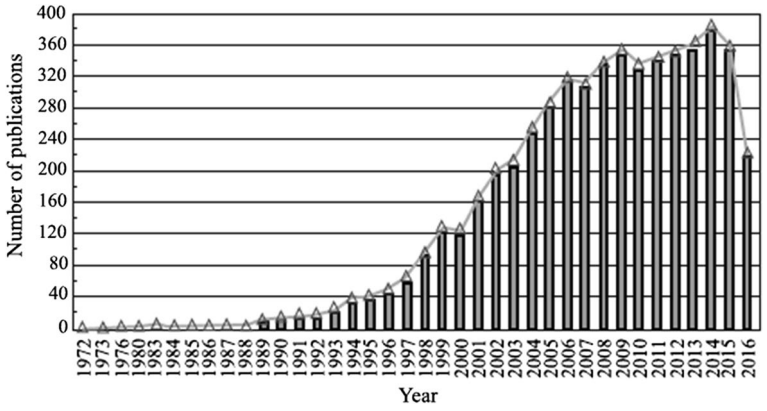

Fig. 1 Figure extracted from (Karna et al. 2018). It represents the number of publications per year. The authors stress that all the publications in 2016 have not yet been recorded, thus explaining its low number

Since the 90's, the situation has changed. In 2007, Rushby touted the rise of applications of formal methods in another paper entitled "Automated Formal Methods Enter the Mainstream". He claims that "[e]ffective formal calculation was not available when formal methods began their evolution[.] (…) Recently, however, a number of developments have combined to make formal methods an attractive technology for many areas of software engineering" (p. 651).

Similarly, Woodcock et al. (2009), who offer a state of the art in the industrial use of formal methods, make the following conclusion:

> There were heroic efforts to use formal methods 20 years ago when few tools were available (to use Bloomeld's phrase [Bloomeld and Craigen 1999]). For example, in the 1980s the application of the Z notation to the IBM CICS transaction processing system was recognised as a major (award-winning) technical achievement [Houston and King 1991], but it is significant that it used only very simple tools: syntax and type-checkers. In the 1990s, the Mondex project (Sect. 4.3) was largely a paper-and-pencil exercise, but it still achieved the highest level of certification. *Our evidence is that times have changed: today many people feel that it would be inconceivable not to use some kind of verification tool*. Whether they are right or not, there has been a sea-change among verification practitioners about what can be achieved: people seem much more determined to verify industrial problems. (p. 30, our emphasis)

In the same line, Karna et al. (2018) offer an extended survey on the role of model checking in software engineering. In particular, it is shown that there is an increasing number of publications related to model checking, which is a specific kind of formal methods (see Fig. 1).

Why are formal methods used more widely in practice? One of the reasons is that progress has been made in the *automation* of formal calculations, which allows to use them for a modest investment. For example, technological progress in the development of satisfiability (SAT) solvers can be applied to formal methods that involve

decision procedure, such as whether the expression q/(z+2) occurring in a program will be divided by zero where z=3*x+6*y−1 for any x and y integers (Rushby 2007).

Nowadays, we could not argue that formal methods are just for theoretical computer scientists. Formal methods are applied in many areas of hardware and software. For example, NASA uses formal techniques in different projects, such as *Airborne Coordinated Conflict Resolution and Detection* (ACCoRD) project, which "is a framework for the formal specification and verification of state-based conflict detection and resolution algorithms".[8] An example of one of the earliest applications of formal methods is Swedish railway signaling (Borälv and Stalmarck 1999, p. 330). In this case, formal methods are used for the development of software-based interlocking systems. Aerospace is one of the main fields in which formal techniques are used. For instance, the French national aerospace research centre (Onera) uses formal methods for the verification of critical aerospace software and strongly argues for its helpfulness (Wiels et al. 2012, p. 2). They emphasize that software verifications performed by means of simulation and testing are "not exhaustive, and still very labor-intensive and costly". Instead, formal methods are "automated and exhaustive". Their use increases since, for instance, "certification credits for the use of formal methods in aeronautics have been obtained by Airbus for the A380 software" (Wiels et al. 2012, p. 2). In particular, the use of Astree is a good example of the increasing popularity of aerospace's applications of formal methods. Astree is a program analyzer that proves the absence of Run Time Errors (RTE) in programs written in the C programming language.[9] This program analyser is based on ''abstraction interpretation'', which is a formal theory applied to the semantics of C language. As an example of the efficiency of Astree, it "was able to prove completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system, a program of 132,000 lines of C".[10]

5.3 Formal Methods in Practice: A Case Study

It is outside the scope of our paper to describe all the possible applications of formal techniques in detail. Nevertheless, in order to argue that formal methods are hardly at a loss to be applied and used in practice, we illustrate, based on a case study, how they can be used in verification. More precisely, this section develops the way an ocean circulation model is verified, at least partially, with formal methods.

The ocean circulation model of interest here, called ADCIRC (ADvanced CIRCulation model), is used, notably by the U.S. Army Corps of Engineers, to study hurricane storm surges. The model describes coastal flooding from tropical storms, and more particularly calculates water surface elevations and velocities from wind velocities, atmospheric pressure, and land and seafloor surfaces. It is based on a

[8] https://shemesh.larc.nasa.gov/people/cam/ACCoRD/.

[9] http://www.astree.ens.fr/.

[10] http://www.astree.ens.fr/, section 'Industrial Applications'. See also Bozzano et al. (2017) for a discussion on formal methods for aerospace systems.

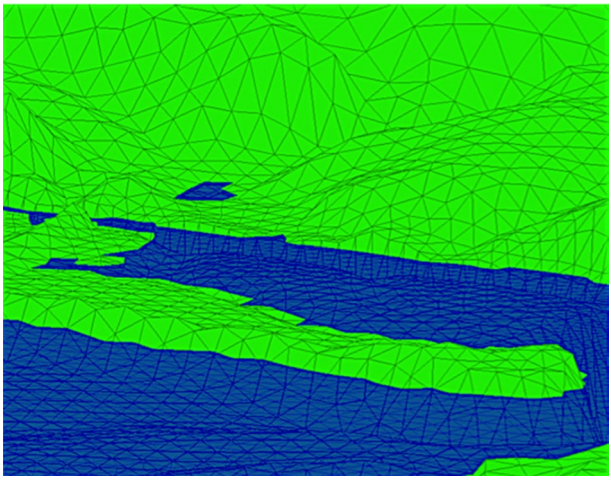

Fig. 2 Figure extracted from Baugh and Altuntas (2018). It depicts a shoreline where the land and seafloor are represented as a collection of elements, viz. contiguous, non-overlapping triangles

spatial mesh space for the dynamics of the ocean (see Fig. 2), that counts 1,224,714 full finite elements. Consequently, the correctness of the computational model importantly depends upon the accuracy of the mesh. Therefore, while verifying the model, Baugh and Altuntas (2018) applies a formal analyzer, called Alloy, to the verification of the mesh.

Baugh and Altuntas emphasize the novelty of this formal approach:

> The tools and techniques most often associated with scientific computing are those of numerical analysis and, for large-scale problems, structured parallelism to improve performance. Beyond those conventional tools, we also see a role for formal methods and present one such application here using the Alloy language and analyzer (Jackson 2012).
> Alloy combines first-order logic with relational calculus and associated quantifiers and operators, along with transitive closure. It offers rich data modeling features based on class-like structures and an automatic form of analysis that is performed within a bounded scope using a SAT solver. For *simulation*, the analyzer can be directed to look for instances satisfying a property of interest. For *checking*, it looks for an instance violating an assertion: a counterexample. The approach is *scope complete* in the sense that all cases are checked within user-specified bounds. Alloy's logic supports three distinct styles of expression, that of predicate calculus, navigation expressions, and relational calculus. The language used for modeling is also used for specifying properties of interest and assertions. (Baugh and Altuntas 2018, p. 101)

Alloy is a declarative modeling language with an automatic form of analysis performed within a SAT solver. It enables users to investigate several properties of the mesh of ADCIRC, including static properties, i.e., whether the mesh is well-formed and satisfies the specification of the algorithms. In particular, the formal analyzer

$$
\begin{aligned}
\text{Mesh} &= \{(m_0)\} \\
\text{Triangle} &= \{(t_0), (t_1), (t_2)\} \\
\text{Vertex} &= \{(v_0), (v_1), (v_2), (v_3), (v_4)\} \\
\text{triangles} &= \{(m_0, t_0), (m_0, t_1), (m_0, t_2)\} \\
\text{adj} &= \{(m_0, t_0, t_1), (m_0, t_1, t_0), \\
&\quad (m_0, t_1, t_2), (m_0, t_2, t_1)\} \\
\text{edges} &= \{(t_0, v_0, v_2), (t_0, v_2, v_1), (t_0, v_1, v_0), \\
&\quad (t_1, v_1, v_2), (t_1, v_2, v_3), (t_1, v_3, v_1), \\
&\quad (t_2, v_3, v_2), (t_2, v_2, v_4), (t_2, v_4, v_3)\}
\end{aligned}
$$

Fig. 8. An instance of a mesh in Alloy.

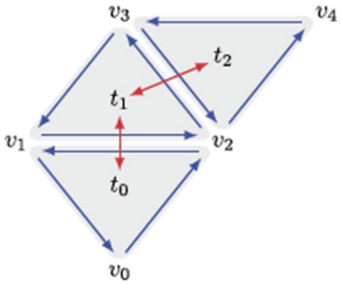

Fig. 9. A planar embedding of the instance as a mesh of triangles (t_0, t_1, and t_2), vertices (v_0, v_1, v_2, v_3, and v_4), and edges (in blue), with triangle adjacency (in red), cf. Figs. 8 and 10.

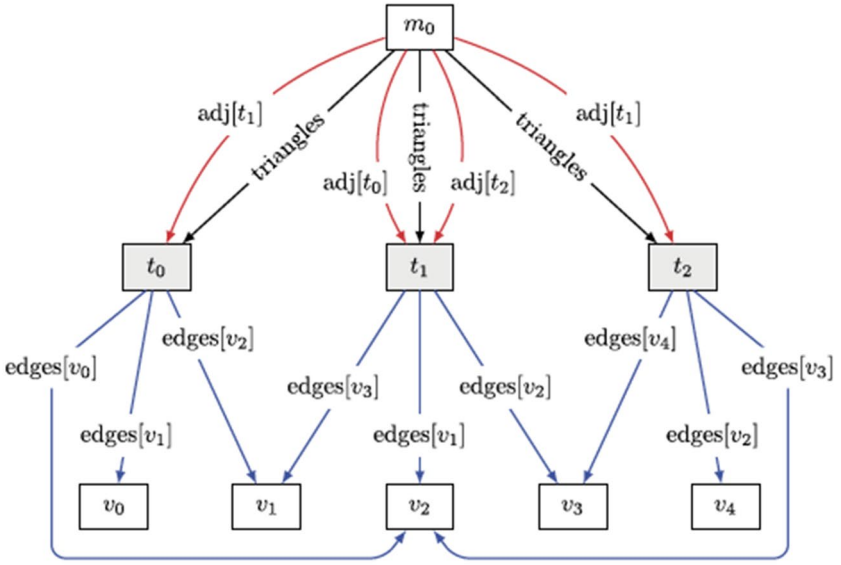

Fig. 10. The same instance displayed as a graph, with arcs that represent relations between atoms.

Fig. 3 Figures extracted from Baugh and Altuntas (2018), numbered as Figs. 8–10 in the original paper

can give a visual representation of meshes, such as, for example, the elementary mesh m_0. This mesh consists of just three triangles t_0, t_1, t_2, and five vertices (v_0, v_1, v_2, v_3, v_4) (see Fig. 3).

With Alloy, users can check topological relations in the mesh, notably Euler formula for graphs. Baugh and Altuntas have verified that, in ADCIRC, there are no cut points, viz., that connectivity is not maintained by a single vertex only in the mesh: "Alloy finds and guarantees that there are no counterexamples within that scope in under a minute on a laptop computer with a 2.8 GHz Intel Core i7" (p. 109). The authors have also verified, based on Alloy, dynamic properties such as the condition

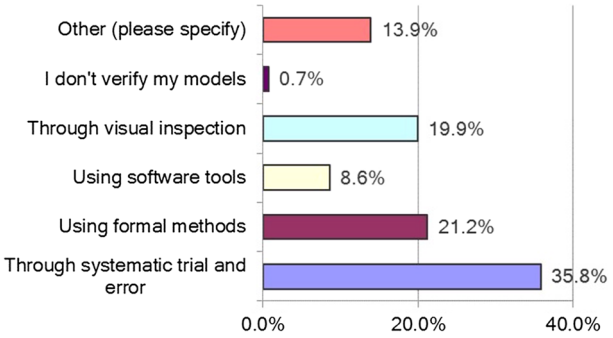

Fig. 4 Figure extracted from Padilla et al. (2017). It represents the common practices for verifying models. Formal methods are the second most often used set of practices for verifying models

under which a triangle in the mesh changes from 'dry' to 'wet'. Moreover, mesh partitioning has been formally verified so to decrease calculation time. In particular, the equivalence between a full run of the program and a partial run has been investigated.

We would like to stress that the ocean circulation model is a complex model composed of non-linear equations. Yet formal methods are used in the verification of this model, suggesting that formal methods can apply in practice in complex and non-linear domains. In the remaining of the paper, we further explore the extent of their use so to shed new light on the debate between Winsberg and Morrison.

6 Formal Methods: The Extent of Their Use

Given the uses of formal methods in science, we have shown that not only verification and validation can be disentangled in principle but also they *are* disentangled in practice in concrete cases. Furthermore, formal methods are more and more used in practice in verification since the 1990's. But to which extent are they currently used? This question is worth addressing since, depending on the provided answer, we are more or less legitimate in arguing for a separation between verification and validation. This section tackles the question by discussing recent overviews on the use of formal methods in science. Thus it provides us with no general argument about the possibility of disentangling verification and validation, but still gives us empirical justifications to the assumption that formal methods are going to broaden their scope of application and thereby that the possibility of disentangling verification and validation is becoming more and more possible and generalizable.

Padilla et al. (2017) made a survey within the modeling and simulation community that helps in estimating how often formal methods are used in verification. The authors recorded the responses of 283 participants, all coming from various educational backgrounds, such as, oceanography, social sciences, and

engineering, all involved in diverse activities of research or business, from academia, industry, government. In this sample group, 151 respondents are model builders.

One topic investigated by the survey is about the common practices in verifying computational models. Those practices range from informal methods, such as visual inspections, to formal methods. The survey clearly shows that formal methods represent the second most frequent kind of verifying practices used by 21.2% of respondents after the systematic trial and error used by 35.8% of them (see Fig. 4).

Although this result should be mitigated by that software tools may be partly included in what they refer as formal methods (cf. Alloy analyzer for the verification of ADCIRC), it still shows that recently, in 2017, formal methods are neither a privileged nor an unusual technique for verifying computational models. Formal methods are definitely used in practice, although not yet systematically. Therefore it remains difficult to draw, from this survey, definitive general conclusions.

Furthermore the survey also shows that the extent to which formal methods are used depends on the characteristics of the models. In particular, it is stressed that ''SD [System Dynamics] models were significantly more likely to have been verified formally (30.5%) than through visual inspection (9.6%) or trial and error (12.9%). On the other side, Agent-based Models were significantly more likely to be verified through visual inspection (38.7%) than formally (13.8%)'' (p. 498).

While this result is already quite compelling, the survey brings an additional interesting element for identifying the domains in which formal methods are more frequently used. In the survey, it is indeed reported that:

> models that were formally verified were significantly more likely to have been submitted for third party accreditation (11% as opposed to 1% of models verified through trial and error) (…)
> In more general terms, modelers in the defense, healthcare, and business industry were significantly more likely to formally verify their models whereas modelers in science and engineering are more likely to use systematic trial and error to verify their models. In addition, modelers who rely on professional organizations and textbooks were significantly likely to formally verify their models while those who rely mostly on technical reports were significantly more likely to verify through visual inspection. (…) Finally, organizations that develop models for sale were also more likely to use formal verification. (p. 498)

Accreditation is a decision usually made by an authority about whether to use a simulation, e.g., when issues of public responsibilities or safety requirements are involved. It is to say that formal methods are overrepresented when accreditation is needed. In the same vein, another survey (Woodcock et al. 2009, p. 5) made within the users of formal methods, reports that formal methods are overrepresented in critical domains, like transport, financial, healthcare, defense, or nuclear.

In summary, formal methods have been identified as the second most often used set of practices for verifying computational models, whatever the domains at stake are (Padilla et al. 2017). Restricted to system dynamics models, formal methods actually become the first most used practice.

7 A Plurality of Practices for V&V

There are diverse methods for verifying computational models, whose relevance depends on scientific domains, and use obviously depends on ethos in the modeling and simulation community. Formal methods are not used in all cases, but they are clearly dominant in some important contexts, including safety and critical domains, and contexts in which certification and accreditation are required.

We are now in the position of arbitrating the debate between Winsberg and Morrison. Their respective theses may not apply on the same kinds of computational models. Winsberg is very significantly interested in models used in climate science (in that sense, his (2018) book dedicated to climate science follows this line), while, elsewhere, e.g., in his (2009) and (2010), he aimed at discussing computer simulations *in general*, not just in climate science. Morrison may have other cases in mind, such as the ones we have stressed which pertain to critical domains.

Winsberg claims that verification is rarely not achieved ‘‘‘when models are sufficiently complex and non-linear’’. He is certainly right for climate science and other scientific domains as well. But, his thesis might not be fully representative of all methods for verifying computational models in science. As we have demonstrated, formal methods are used in complex systems, such as the case of the ocean circulation model ADCIRC or aerospace and aircraft systems. In those cases, verification and validation can be separated in practice. It seems therefore that there is not limitation in the complexity of models up to which formal methods cannot be used in practice. Whether formal methods can be used seems to rather depend on the domain of application, whether it involves critical domains and high security requirements. The ocean circulation model we discussed is used to predict the effects of hurricane storm surge. Such models serve evacuation planning and vulnerability assessment.

Winsberg might agree with us since, in his 2018 book, he concedes a point we argued for:

> In some engineering contexts, […] the V&V framework is actually more or less applicable. Some engineers have made the point to me that in highly sensitive applications, such as simulations used to assure the safety of the nuclear stockpile, it is fundamental and important that the V&V framework apply. (chap 10, p. 162)

The debate regarding V&V’s possible entanglement might thus end in the following terms: there is a plurality of methods for verifying computational models, all depending on the domains of interest, in particular, whether they are critical. Therefore there is a range of possibilities, going from separability to entanglement in V&V.

8 Conclusion

We focused on the debate between Winsberg and Morrison about V&V. This debate comes from the novel way the Duhem problem applies to computational models: model validation comprehensively tests model assumptions, including those with a representational aim and those related to the numerical scheme.

According to Winsberg (2009, 2010, 2018), this problem cannot be overcome by the V&V methodology as models are generally too complex in practice for strong mathematical arguments of verification to be built. Morrison (2015) argued that Winsberg overstates the entanglement between verification and validation.

In this paper, we aimed to arbitrate this debate. We mainly explored the use of formal methods for verifying computational models. We showed that there is an increasing use of formal methods in the modeling and simulation community. These methods are mainly used when models are involved in critical domains with security requirements.

Our conclusion is that the entanglement of V&V has to be mitigated, as it mainly depends on domains of application. That said, formal methods are more and more used, thus making the separability in V&V more and more often possible.

Acknowledgements We thank the guest editors Andreas Kaminski and Michael Resch, as well as to the two anonymous referees for their helpful comments. The paper has also benefited from conversations with audience members at the SPSP Conference in Ghent, and notably with Johannes Lenhard and Nic Fillion.

References

Baugh, J., & Altuntas, A. (2018). Formal methods and finite element analysis of hurricane storm surge: A case study in software verification. *Science of Computer Programming, 158*(15), 100–121.

Borälv, A., & Stalmarck, G. (1999). Formal verification in railways. In M. G. Hinchey & J. P. Bowe (Eds.), *Industrial-strength formal methods in practice*. Berlin: Springer.

Bozzano, M., Bruintjes, H., Cimatti, A., Katoen, J.-P., Noll, T., & Tonetta, S. (2017). Formal methods for aerospace systems—Achievements and challenges. In S. Nakajima, J.-P. Talpin, M. Toyoshima, & H. Yu (Eds.), *Cyber-Physical System Design from an Architecture Analysis Viewpoint* (pp. 133–159). Berlin: Springer.

Butler, R. W. (2001). What is formal methods? Last Updated: April 10, 2016. Consulted in January 2018. https://shemesh.larc.nasa.gov/fm/fm-what.html.

Clarke, E. (2008). *The birth of model checking, 25 years of model checking*., Lecture notes in computer science Berlin: Springer. https://doi.org/10.1007/2F978-3-540-69850-0_1.

Dowson, M. (1997). The ARIANE 5 software failure. *Software Engineering Notes, 22*(2), 84.

Fillion, N. (2017). The vindication of computer simulations. In M. Carrier & J. Lenhard (Eds.), *Mathematics as a tool* (pp. 137–156). Boston: Boston Studies in the Philosophy of Science.

Frigg, R., & Reiss, J. (2009). The philosophy of simulation: Hot new issues or same old stew? *Synthese, 169*(3), 593–613.

Jackson, D. (2012). *Software abstractions: Logic, language, and analysis*. London: MIT Press.

Jebeile, J., & Barberousse, A. (2016). Empirical agreement in model validation. *Studies in History and Philosophy of Science Part A, 56,* 168–174.

Karna, A. K., Chen, Y., Yu, H., Zhong, H., & Zhao, J. (2018). The role of model checking in software engineering. *Frontiers of Computer Science, 12*(4), 642–668.

Lenhard, J. (2018). Holism, or the erosion of modularity: a methodological challenge for validation. *Philosophy of Science, 85*(5) 832–844.
Lenhard, J., & Winsberg, E. (2010). Holism, entrenchment, and the future of climate model pluralism. *Studies in History and Philosophy of Science Part B, 41*(3), 253–262.
Lions, J. L. (1996). Ariane 5 Flight 501 Failure. *Ariane 501 Inquiry Board Report* (p. 4).
Morrison, M. (2014). Values and uncertainty in simulation models. *Erkenntnis*, *79*(S5), 939–959.
Morrison, M. (2015). *Reconstructing reality: Models, mathematics, and simulations*. Oxford: Oxford University Press.
Oberkampf, W. L., & Trucano, T. G. (2002). Verification and validation in computational fluid dynamics. Rapport Sandia, SAND2002-0529.
Oberkampf, W. L., Trucano, T. G., & Hirsch, C. (2002). Verification, validation and predictive capacity in computational engineering and physics. *Applied Mechanics Review, 57*(5), 345.
Oreskes, N., Shrader-Frechette, K., & Belitz, K. (1994). Verification, validation, and confirmation of numerical models in the earth sciences. *Science, 263*(5147), 641–646.
Padilla, J. J., Diallo, S. Y., Lynch, C. J., & Gore, R. (2017). Observations on the practice and profession of modeling and simulation: A survey approach. *Simulation, 94*(6), 493–506.
Rushby, J. (1995). Formal methods and their role in the certification of critical systems. Technical Report CSL-95-1, March 1995.
Rushby, J. (2007). Automated formal methods enter the mainstream. *Communications of the Computer Society of India, Formal Methods Theme Issue, 31*(2), 28–32 **(Archived in Journal of Universal Computer Science vol. 13, No. 5, pp. 650–660)**.
Skeel, R. (1992). *SIAM News* (Vol. 25, No. 4). https://w3.ual.es/~plopez/docencia/itis/patriot.htm. Accessed 20 Nov 2018
Trefethen, L. N. (1994). Finite difference and spectral methods for ordinary and partial differential equations, unpublished text. Available at http://people.maths.ox.ac.uk/trefethen/pdetext.html.
Wiels, V., Delmas, R., Doose, D., Garoche, P. L., & Cazin, J. (2012). Formal verification of critical aerospace software. *AerospaceLab, 4,* 1.
Winsberg, E. (2009). Computer simulation and the philosophy of science. *Philosophy Compass, 4,* 835–845.
Winsberg, E. (2010). *Science in the age of computer simulation*. Chicago: University of Chicago Press.
Winsberg, E. (2018). *Philosophy and climate science*. Cambridge: Cambridge University Press.
Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys, 41*(4), 1–36.