Synthese (2009) 169:557–574
DOI 10.1007/s11229-008-9430-7

Computer simulations as experiments

Anouk Barberousse · Sara Franceschelli · Cyrille Imbert

Received: 1 November 2006 / Accepted: 4 August 2007 / Published online: 18 November 2008

Abstract Whereas computer simulations involve no direct physical interaction between the machine they are run on and the physical systems they are used to investigate, they are often used as experiments and yield data about these systems. It is commonly argued that they do so because they are implemented on physical machines. We claim that physicality is not necessary for their representational and predictive capacities and that the explanation of why computer simulations generate desired information about their target system is only to be found in the detailed analysis of their semantic levels. We provide such an analysis and we determine the actual consequences of physical implementation for simulations.

Keywords Experiments · Data · Machine implementation · Programs

1 Introduction: the practice of computer simulation

Some computer simulations undoubtedly share some epistemic functions with field experiments: they are run to provide new data about systems that are difficult or impossible to investigate with ordinary instruments. For instance, neutron–matter interaction

A. Barberousse, S. Franceschelli and C. Imbert have contributed equally to this work.

A. Barberousse (✉) · C. Imbert
IHPST, CNRS, Université Paris 1, ENS, Paris, France
e-mail: Anouk.Barberousse@ens.fr

S. Franceschelli
ENS-LSH, Lyon, France

S. Franceschelli
REHSEIS, Paris, France

has been studied intensively by means of molecular dynamics simulations. Owing to these simulations, it is possible to investigate, among other phenomena, the diffusion of neutrons by a phonon, diffraction, anomalous dispersion.

Comparing computer simulations with field experiments is a good way to shed light on many distinctive features of simulations. In both cases, vast amounts of data are first produced and then analyzed with the help of similar techniques. How are these data produced and validated? This question is even more difficult to answer in the case of simulations than in the case of field experiments, for computer simulations do not involve any direct physical interactions with the systems they are used to investigate. We use "direct physical interaction" in the sense of measurement interaction between us and the target system. Even if sometimes simulations are initialized with data obtained by measuring the target system, they never involve any direct physical interaction with it.

This paper aims to give a qualified answer to the above question for the case of computer simulations. We argue that, after its earlier, mostly descriptive phase, the epistemology of computer simulation can now take an explanatory turn (cf. Winsberg 1999, 2001, 2003). Putting simulations and models on the same philosophical footing, we claim in this paper that the best candidates for such an explanation is to be found in a detailed semantic analysis of computer simulations, showing how the physical states of the computer can step after step be interpreted as computational states, as values of variables and finally as representations of the target system's successive states. When performing this semantic analysis, we do not ignore the fact that a decisive component for a good simulation for a simulation to generate desired information about its target system is that it relies on a good model of the target system. This model has to be transformed (sometimes drastically) into algorithms. We carefully analyze some of these transformations, which lead to well-calibrated simulations, obtained by trial-and-error procedures and comparisons with already available results.

A frequently given explanation as to why (some) computer simulations are used as experiments appeals to their being run on physical machines. We call this claim the "physicality claim". In Sect. 2, we present different arguments in the literature that may be used to support the physicality claim. In Sect. 3, we try to answer what we believe to be the best argument in favor of a strong version of the physicality claim. In Sect. 4, we present our own semantic analysis of computer simulations as well as its implications. In Sect. 5, we ground our analysis in a renewed analysis of the actual meaning of machine implementation.

2 Simulations can be used as experiments because they are run on machines

Setting up an epistemology for computer simulations begins with considering the following explanandum E:

> E: Computer simulations, although they are basically computations and do not involve any measurement interactions, nevertheless do generate new data about empirical systems, just as field experiments do.

Explanations of *E* often point out a distinctive relationship between the physicality of computer simulations (i.e. the fact that computer simulation are physical processes) and their being used as data generating experiments. We call the appeal to such a relationship the "physicality claim". In this section, we review different theses relying on the physicality claim. Our aim is to show that this appeal to physicality as an explanatory feature of how computer simulations generate data is left under-analyzed. In particular, what physicality implies is never stated explicitly.

The physicality claim seems to underlie the metaphor of the computer a "a stand in for, or a probe of" the system upon which experiments are made, a metaphor used by Winsberg (2003) as well as Norton and Suppe (2001). This metaphor is central in Norton and Suppe's paper. Since computer simulations allow one to vary initial and boundary conditions, as well as the values of control parameters, in the same way one can in real experiments, and even more so, the methodology of experimentation provides Norton and Suppe with a powerful argument in favor of this thesis (cf. Norton and Suppe 2001, pp. 56–57). In their attempt to explain why this is the case, Norton and Suppe resort to ontological properties of simulations. Simulations, according to them, "embed" theoretical models in real-world *physical* systems (computers). Therefore, even though computers used for simulations do not physically interact with their target systems, they can yield data in the same way field experiments do, namely by probing analogues of their target systems. The "embedding" of theoretical models acts as a reliable substitute for detection and measurement interactions. We shall return to Norton and Suppe's theses in Sect. 3.

Hughes (1999) presents a four-part comparison clarifying the notion of simulation. He first succinctly describes three non-computer simulations of physical processes, namely: a model water-wheel used to understand full-size water-wheels; the electrical model of a damped pendulum; and a cellular automaton, conceived as a physical system whose behavior can be the direct subject of experiment.[1] He then contrasts these examples with a computer simulation of the atomic interactions between a pine and a plate described in Rohrlich (1991). According to him, in the first three examples, contrary to the computer simulation case, there is "an obvious correspondence between the process at work in the simulating device and those at work in the system being modeled". "Nevertheless", he adds, "like the systems examined [the first three simulating systems], the computer is a material system, and in using it to provide answers to our questions *we are relying on the physical processes at work within it*." (p. 139, our emphasis) Here again, physicality seems to be a crucial feature but Hughes provides us with no further details about his own interpretation of the physicality claim.

Finally, Humphreys (1994) argues that when a program runs on a digital computer, the resulting apparatus is a physical system, i.e., any runs of the algorithm are just trials on a physical system. This is a new version of the physicality claim. Humphreys further points out that "there is no empirical content to these simulations in the sense that none of the inputs come from measurements or observations. [...] It is the fact that

[1] Hughes here contrasts CA with computer simulations because, what is characteristic of CA, according to him, is that "the structure of the automaton is designed to match crucial structural properties of the system being modeled, like its topology and its symmetries", not that it can mimic the system's behavior.

they have a dynamical content due to being implemented on a real computer device that sets these models apart from ordinary mathematical models" (1994, p. 112).

The notion of a "dynamical content" is not a standard one and is thus difficult to understand. Even if allusions to the "essentially dynamic nature of simulations" are common-place (as shown in the "Research questions" presenting the "Models and Simulations" Workshop held in June 2006 in Paris), this "dynamic nature" is left unanalyzed. However, it is undoubtedly connected with the fact that computer simulations are, at a basic level, physical processes occurring on concrete devices[2] (cf. also Hartmann 1996). Since they are sequences of physical events, namely physical state transitions, they are particular processes, localized in space and time, exactly as field experiments are. Moreover, the cognitive importance of the fact that, during the performing of (at least some) simulations, something happens before our eyes on the computer's screen so that we can see the evolution of the model we have "put in" the computer, cannot be overestimated in any epistemological description of how we explore computational models. However, Humphreys' claim that "there is no empirical content in these computer simulations" is apparently at odds with the fact that, whereas simulations do not involve measurement interactions, some of them nevertheless contribute like field experiments to providing knowledge about physical systems. This contrast calls for explanation.

Let us introduce an important distinction relative to the notion of "empirical content". The term "data" is commonly used in two senses that are conflated when it is applied to field experiments, but need to be distinguished when applied to simulations. Data may be:

– of empirical origin ("data$_{\mathrm{E}}$"), namely produced by physical interactions with measuring or detecting devices,
– and/or *about* a physical system ("data$_{\mathrm{A}}$").

We claim that data$_{\mathrm{A}}$ may be produced by means of data$_{\mathrm{E}}$, but also via other processes, among them analytical or numerical pen-and-pencil-computed solutions of systems of equations representing the target systems, and computer simulations.

In field experiments, the system investigated produces data in the first sense ("data$_{\mathrm{E}}$"), as well as in the second sense ("data$_{\mathrm{A}}$"). Take a simple optics experiments as an example. Interference fringes are obtained that can be seen on a screen orthogonal to the optical axis of the experiment. In this case, the observed fringes and their measured characteristic distance are the relevant data$_{\mathrm{E}}$produced by the experimental apparatus. These data$_{\mathrm{E}}$ guarantee the data$_{\mathrm{A}}$ we are interested in, namely the presence of interference fringes at such and such distance in such and such optical conditions. It is because our optical system and measuring apparatus produce data$_{\mathrm{E}}$ that we can process data$_{\mathrm{A}}$in order to gain genuine knowledge about it.

Sometimes, the situation is more complicated. Take astronomical observations as a further example. In this case, we record temporal series of light intensities by means of a sensor and a telescope, that we further interpret as indicating stellar positions. The numbers we obtain are data$_{\mathrm{E}}$ produced by our observational apparatus plus the target

[2] Humphreys insists that he is only concerned with computations that have "a dynamic physical implementation" (1994, p. 104).

stars. These numbers are also data$_A$: they are about the light intensities that came across the telescope at such and such times. We may call these data$_A$ "data$_T$", for "data about the telescope". We ordinarily use data$_T$ to obtain other data$_A$, namely data about stellar positions. Let us call these "data$_S$". In the inferential process from data$_T$ to data$_S$, the physical origin of data$_T$ is a necessary condition for our obtaining of any knowledge about stellar positions. For such an inference to be possible, it must be ascertained that data$_T$have been produced by a physical process that has been happening at such and such time and place. A central feature of this physical process is that the target system, namely the stars, are causally connected with the measuring apparatus. It can be checked as follows. For the inference from data data$_T$ to data$_S$ to be satisfactory, data$_T$ have to be analyzed and corrected by dedicated computer programs in order to remove interferences due to the presence of planes or satellites during the recording times.

Let us finally turn to computer simulations. For sure, computer simulations, qua physical systems, produce data$_E$ and data$_A$ about themselves[3] that we may call "data$_C$". We emphasize that data$_C$ are normally interpreted without hesitation as the results R of a computational process. This computational process is meant to solve the equations of the model(s) we use to investigate the phenomena we are interested in. Its very meaning depends on the hypotheses we put in these models. In order to get *the* R we are interested in, namely the numbers that provide desired information about the system, a hard methodological and epistemological work has to be done in terms of calibration, using available results, be they analytical, numerical or empirical, and also in terms of trial and error, exploration of intermediary models, etc. (cf. Winsberg 1999). As Winsberg puts it, many "tricks of the trade" need to be employed, involving simplifying assumptions, removal of degrees of freedom, substitution of simpler empirical relationships for complex ones (ibid., p. S445).

We acknowledge that in order to obtain data$_A$ (about the system that is investigated), we do use these results R. This is why the physicality claim is so attractive: computers being physical systems does matter. So it is of course true that whatever kind of information may be obtained by a simulation, it is provided by a computer, namely by a physical device, as Hughes, Humphreys, Norton and Suppes emphasize. However, this leaves unexplained why and how data$_C$ (about the computer), interpreted as the results of a computation, can be used to produce data$_A$ (about the system that is investigated). And yet, the way from data$_C$ to data$_A$ is precisely the conceptual place grounding the comparison between computer simulations and field experiments. Consequently, the physicality claim does not in itself explain much about E.

We can nevertheless make sense of Humphreys' intuition that "there is no empirical content in computer simulations". In a computer simulation, data$_E$ only allow us to obtain data$_C$ (about the computer), but in no way to understand why and how we may

3 Simulationists do not have access either to data$_E$ or to data$_C$. Data$_C$ are lists of electric intensities, magnetic intensities, and electric potentials at different points of the computer, as well as values of possibly relevant parameters like temperature, plus a precise description of the architecture of the computer, including the materials of the condensors, resistors and the like. What the simulationists work with is only the *interpretation* of data$_C$ as results of a computation. These latter results, taken alone, can hardly be considered as exploitable data about the computer.

obtain data$_{\mathrm{A}}$ (about the target system). As we shall claim, provided that we know which computation we have the results R of, the way the computation is physically carried out becomes irrelevant to interpretating R as a set of data$_{\mathrm{A}}$. Would the results R have been obtained differently, e.g. by analytical or pen-and-pencil-obtained numerical solutions (appropriately visualized with the help of an oft-used data-analysis software), their interpretation as data$_{\mathrm{A}}$ would remain unchanged. As we saw above, this is not true for field experiments.

We now need to understand how data$_{\mathrm{C}}$ (about the computer), namely the results R of a computation, can be transformed into data$_{\mathrm{A}}$ (about the target system). A first option is to claim that the conversion of data$_{\mathrm{C}}$ into data$_{\mathrm{A}}$ is possible because there is a common structure between the target system, the theoretical model of it and the simulation as a physical process. This is what we call the "strong version" of the physicality claim. In the next section, we discuss Norton and Suppe's analysis of computer simulation because we think that it offers the most powerful argument we are aware of in favor of the strong version of the physicality claim. Against the strong version of the physicality claim, we show later that there need not exist any common structure between the computer simulation and the target system. We shall claim instead that only a detailed semantic analysis, explaining how the physical states of the computer can be interpreted as computational states, then as values of variables, and finally as representing the target system's properties, can explain why data$_{\mathrm{c}}$ can be turned into data$_{\mathrm{A}}$. When analyzing the semantic layers constituting a simulation, we do take into account the fact that a decisive component for good a simulation is definitively that it relies on a good model of its target system. This model has to be transformed into algorithms that enable us to get the information we desire about the target system.

3 Toward a semantic analysis of computer simulation

It is unquestionable that the computer running the program is a physical system. However, it is unlikely that the physical properties of this device are the only ones relevant for its scientific use. Whereas the representational properties of a simulation certainly depend on the physical states of the computer on which the corresponding program is run, this dependence is a rather complex one that cannot be flattened by saying that the computer on which the program is run is a stand in for the physical system upon which experiments are made.

Since Norton and Suppe's is the most detailed analysis of the representational capacities of simulations and of the role played by implementation, we examine it in details in this section. Norton and Suppe acknowledge the importance of the semantic properties of simulations, namely of the way physical states of the computer may be interpreted as computational states, themselves interpretable as numerical values representing physical states of the target system. They distinguish two important mappings that are involved in the use of simulations as experiments. (1) The "lumping mapping" connects a base model consisting of sets of simultaneous values for parameters describing the system on to an "embedded lumped model". (2) The "simulation mapping" connects a programmed computer on to the embedded lumped model.

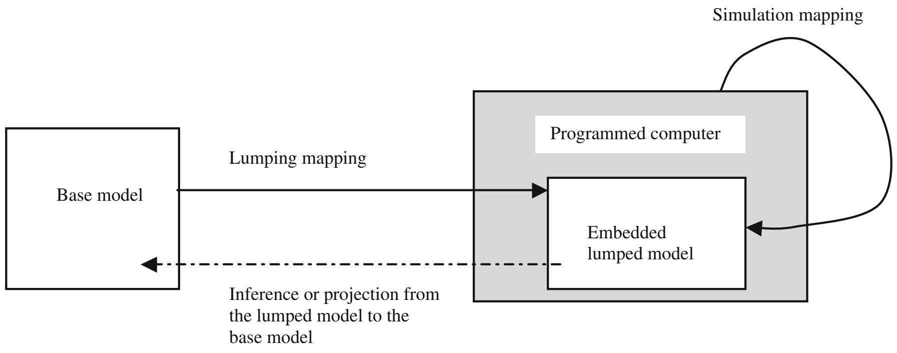

Fig. 1 General simulation modeling situation. Both the base model and the programmed computer realize the lumped model, which is embedded into the computer. (Reproduced from Norton and Suppe 1999, Fig. 3.6, p. 88)

Intuitively, the "embedded lumped model" is a theoretical, mathematical model written in a computer-acceptable language.

In Fig. 1, we have reproduced Norton and Suppe's diagram illustrating the relationships between the base model, the lumped model, and the programmed computer. The lumping mapping as well as the simulation mapping are represented by plain arrows. Let us give Norton and Suppe's own example in order to illustrate their analysis. The simulation's aim is to investigate average annual regional precipitations. The base model is a series of gridded daily precipitation levels. In the lumped model, daily grid-point values are "lumped" together as regional annual averages. Norton and Suppe claim that if the simulation is a good one, lumped-state transition sequences allowed by the lumped model's equations mimic the base model's possible state transitions.

The lumping mapping and the simulation mapping are described with the help of two semantic relations, the "realization" and the "embedding" relations. These relations may be called "semantic" because they both fall within the province of the interpretation either of symbols or of states of a system. With the help of these relations, states of a system are *interpreted as* states of another. According to Norton and Suppe, the existence of the "realization" and the "embedding" relations is instrumental in explaining how computer simulations can be knowledge-creative. The realization relation is defined as follows: "A system S_2 is said to be a *realization* of another system S_1 when there is a many–one behavior preserving mapping from the states of S_2 onto the states of S_1" (Norton and Suppe 2001, note 5). The intuitive definition of the embedding relation is that "a model B of A is *embedded* into A just in case the mapping from A to B partitions complete states of A into equivalence classes that become the complete states of B. Whenever A undergoes a complete state transition from s to $s*$ in B, the equivalence class containing s will be succeeded by the equivalence class containing $s*$" (ibid., note 6).

With the help of these definitions, Norton and Suppe claim that both the physical system represented by the model of data *and* the computer on which the "lumped model" is run are realizations of the lumped model and that this double instantiation of the realization relation is responsible for the experiment-like character of simulations.

First, they claim that "Programmed computers realize lumped models by embedding them into the computer. [The lumped model] embedded into [the programmed computer]—a simulation model M—is a real-world physical system of the lumped model that can be run and observed".

From this, they conclude that "since [the computer] is a real system, observations of it are epistemically creative in the sense that new knowledge of the world can thereby be obtained" (Norton and Suppe 2001, p. 88). Finally, since "what legitimately can be determined about [the target system] from observations of the [implemented model] depends essentially on the extent of behavior preserved under [the two] mappings" (ibid.), Norton and Suppe claim that the following three conditions have to be fulfilled for simulations to yield data or knowledge about the physical systems under investigation.

(i) The lumping and simulation mappings hold;
(ii) Equivalence classes of states of the model of data are mapped onto the states of the lumped model;
(iii) The physical system is isolated from influences not identified in defining the states used in the lumping mapping.

Condition (ii) implies that Norton and Suppe require the lumped model to be embedded both within the computer program *and* within the model of data for the simulation to be informative. This amounts to postulating that the lumped model, the computer program and the model of data share a common structure. Norton and Suppe add that one must in turn do "much experimental work and modeling [in order to determine] whether or not and to what degree, these conditions hold" (ibid., p. 89). The structure identity of the base model, the lumped model and the programmed computer, expressed in condition (ii), is used for: (*a*), justifying that simulations, qua real-world physical systems, can be run, observed, and experimented with; and for (*b*), giving a precise meaning to the intuitive notion of a running program "mimicking" the temporal evolution of a physical system. We shall now show why both claims are unsatisfactory.

We first note that the running computer is not observed qua physical system, but qua computer, i.e., qua *symbolic* system, so that the claim that the computer running the simulation is a stand-in or a probe is at best metaphorical. What is going on on the screen, before our eyes, should not deceive us about the actual nature of the simulating process. Moreover, strictly speaking, the "lumping mapping" holds between the values contained in the data model and the values of the lumped model's solutions, while the "simulation mapping" holds between the values corresponding to the successive interpreted computational states of the computer program, on the one hand, and the values of the lumped model's solutions, on the other. In other words, the computer program as such does not itself "realize" the values of the lumped model; the computer's "current state", regularly updated, does.

Let us now show that in most simulations, the computer's physical states cannot be said to realize the lumped model. It may happen that one and the same physical part of the machine always realizes the memory part of the program, i.e. the list of the values of all variables within the lumped model. It may even happen that the different values of each variable are always stored in the same physical place. However, in most cases, the computer only keeps a record of the addresses where the values of the variables that are used in the program are stored and the storage places can shift. It actually shifts in big simulations when several connected computers are used at the same time, or successively, depending on their availability. In other words, the relation between the programmed computer and the lumped model is a many-many relation: the value of a variable in the model may have many physical locations while a same physical location in the computer may implement the values of several variables at different times.[4] To sum up, if one views the computer as a physical system, there scarcely need be anything like values of physical variables "realizing" the lumped program. Therefore, the lumped program and the programmed computer qua physical system need not share one common structure. We can conclude that, as far as digital simulations are concerned, the fact that simulations yield empirical data cannot be legitimated by a supposedly shared structure between the base model, the lumped model and the programmed computer qua physical system. The appropriate justification is to be found at the computation level or at even higher levels.[5]

Let us further show that a common structure cannot be found between the base model, the programmed computer and the lumped model. We first insist that a more precise expression should be given to this hypothesis: the supposed common structure would exist between the *values of the variables* within the base model, the *values computed by* the programmed computer and the *solutions* of the lumped model.[6] The main reason why this hypothesis is false is that the embedding and realization relations do not have the required properties for the existence of a common structure between these three sets of numbers. Embedding and realization relations are relations between states or between states and equivalence classes of states; in order that they guarantee a common structure, the correspondence they define should be fixed. However, the lumping mapping, for instance, may only hold statistically between the states of the target physical system and the output states.[7] For example, in a computational fluid simulation, the local time-average turbulent energy, or a velocity distribution, or different momenta of a velocity distribution, may be correctly represented in a turbulent fluid simulation, while the exact velocity or vorticity field are not because they depend upon the particular trajectory computed which, contrary to the laminar case, is sensitive to slight perturbations of the velocity field.[8]

[4] Since the physical hardware of the different computers can be different, there is no reason either that the same type of physical state codes for the same computational states. For example, the physical states corresponding to the 0s in one computer may correspond to the 1s in another.

[5] We shall indicate later, in Sect. 5, what about the physicality of the simulation does matter.

[6] Norton and Suppe do not themselves distinguish the various levels at which the computer can be described.

[7] To respond to this concern, one might imagine the following trick. One might carry out the lumping operation again, namely lump together the averaged states. This amounts to making up a super-state of such newly-lumped states. The next step would be to make this super-state into an argument of the mapping. However, different averaged properties can be realized on different and sometimes overlapping intervals of the same temporal series of representational states of the computer. As a result, it is usually possible to find two statistical properties on the same temporal series of representational states of the computer such that they would not be lumped into the same super-state according to this rule. Consequently, no general embedding relation can be built.

[8] Cf. Chassaing (2000, p. 32).

It is now clear that the base model, the lumped model, and the programmed computer scarcely need have the same structure describable in terms of states and state transitions. They usually share some properties corresponding to the information that can be faithfully extracted from the simulation.[9] However, these only properties do not suffice to define a common dynamical structure, i.e. a set of fixed correspondences among states. More details are needed to define such a dynamical structure, for instance, the exact values of the velocity fluid in a turbulence simulation. Since no dynamical structure may be common to the base model, the lumped model, and the programmed computer, a careful analysis of why and how simulations "can be used as instruments for probing or detecting real-world phenomena" is still missing. We provide the main elements of it in the next section.

4 Simulations can be used as experiments because they represent phenomena

We agree with Winsberg, Hughes, Norton and Suppe that an important task for the philosophy of science is to provide a better understanding of the type of knowledge gained through computer simulations. However, since computer simulations do not produce data$_E$ (data from empirical origin) concerning their target systems, contrary to what is the case in field experiments, it cannot be inferred from the fact that they are implemented on concrete machines that they are knowledge-creative. As we have shown, the notion of a computer simulation "mimicking" a physical process is far from clear; even less clear is the idea that the computation's being a physical process itself should explain the "mimicry" relation.

In order to provide a better explanation of the epistemological usefulness of computer simulations, we suggest that their being both computations and representations may be a more fruitful starting point than their being implemented on concrete machines. Since the representational properties of computer simulations are both intricate and layered, the representational analysis must go much deeper than has previously been considered, and a careful analysis of the layers involved is likely to capture what is specific to them and what kind of epistemological benefits they can, or cannot, provide.

4.1 General features of computer simulations

Any general analysis of computer simulations must clearly distinguish between:

(a) an *incomplete* program that includes every single algorithm that is actually used. To be completed, this program only requires that someone enter the values of the relevant variables;
(b) a completed program that only requires someone to press the "run" key to run. This program includes various operations such as the writing of data in a folder, test operations, opening of visual interfaces for visualization. These

[9] Determining which ones they are is what simulationists struggle to do.

operations do not have any representational role. The completed program in turn gives way to:

(c) the running completed program, a process corresponding to the exact running of (b);
(d) the data resulting from (c). These are the data written in the files, i.e. the numbers that can be interpreted as raw data about the represented systems; and finally,
(e) the computer-made analysis of the vast amounts of data (d).

Let us give further details about the items of this "kit form" for simulations.

The computer program in (a) is piecemeal and includes many different subprograms coming from different libraries that usually do not run all together at the same time. Many of them were not typed by the programmers themselves. They include black-boxed software, such as Fluent in computational fluid dynamics. To use them, one need only specify the type of flow (e.g. turbulent or not), the geometry of the configuration, the initial and boundary conditions, etc. (see Lebanon 1996).

Program (b) is especially important in the analysis of computer simulation, since the oft made reference to the physical states of the computer is a reference to the realization of the states of this program. However, program (b) is not what usually seen as the simulation, since only a part of it, that we shall call the "Simulating Portion" (SP), has a representational function or is indispensable to compute these representing parts. There are many other segments of program (b) that do not have any representational function; for instance, subprograms testing that each denominator is different from 0. SP is constituted by the various parts of program (b) that are used in the computation of mathematical functions whose results are interpreted as giving information about the target system. SP is thus designed to do the scientific work required by a simulation, namely passing from one interpreted computational state to another, but it is not the program that actually runs.

Data analysis (e) is the actual aim of a simulation. This analysis is usually sophisticated and involves an important amount of computational resources. It provides the basis for several, hierarchically organized "models of the phenomena" in Winsberg's sense (1999). A "model of the phenomena" is the result of the (partially automated) interpretation of the very large sets of data produced by the simulation: "it can consist of mathematical relations and laws, images (both moving and still), and textual descriptions" (Winsberg 1999, p. 283). It summarizes "the basic robust qualitative features of structurally similar phenomena" (ibid.).

4.2 Running computer programs

In this section and the following one, we shall show that every single semantic step of the hierarchy has to be taken into account in order to describe how simulations can be used in the investigation of empirical systems. The items that are usually associated with simulations are (b) (mostly SP) and (c). However, items (b) and (c) are usually poorly analyzed from a semantical point of view in the philosophical literature. This is why we now turn to the properties of running computer programs, namely of (c).

When the program is running, the memory parts of the computer[10] can themselves be viewed in a variety of ways, each of which must be taken into account if one is to understand how they can provide knowledge about physical systems:

(1) as a succession of physical states of the machine on which the program is run;
(2) as a succession of computational states of the machine;
(3.1) as successive sets of values for mathematical, discrete iterative functions;
(3.2) in some cases, as successive sets of values of solutions of differential equations taken at regular intervals;
(4) as successive numerical representations of physical states of a physical system governed by laws.

We call this list the multi-layered ("lasagna") description of the (memory part of the) running computer program. This list does not provide identity conditions for simulations. It aims at shedding light on the variety of the interpretative mappings that come into play when going from (1) to (4).

Analyzing the relationships between computational states of the computer and numerical representations of states of the physical system under investigation is the best way not to take the representation relation between the simulation and the physical phenomena for granted. Without such an analysis, one cannot assess how scientists select the relevant information within a simulation's output. Our proposal makes that point clear, for it pursues the research that has been done into the representational capacities of models over the past few decades and applies them to simulations.[11]

It has to be emphasized that simulations are usually described the other way round, namely as mainly directed to item (4). This is also the way computational physicists designs their programs. When such descriptions are used, the implementation relation is often taken as a straightforward matter involving no special difficulty. However, since it is almost never obvious how to design a discrete representation of a given continuous function or how to struggle with round off errors, infinite limits, and the like, from the physicist's point of view, the way from (4) to (2) is replete with hard to make decisions.[12] Computational scientists know that there is a gap between what they wish they could implement and what they can implement. They also know that the target properties may not finally all be faithfully represented. Since applied mathematics does not always inform us about this gap, the way the various elements of a computer simulation should be interpreted in order to produce reliable information about the target system has to be established by trial and error, calibration, and the like (see Humphreys 2004, pp. 116–121). This suggests that when the implementation relation is left unanalyzed, one does not exactly know which set of mathematical functions are iteratively computed and, therefore, what physical phenomenon is being simulated.

[10] In a computer, only some (physical) locations are dedicated to the storage of the values of the variables that are used in the running program. We call them the "memory parts" of the computer. They constitute the physical basis of the interpretation cascade we describe.

[11] Cf. for instance, Morgan and Morrison (1999); Hartmann and Frigg (2006).

[12] Clearly listed by Winsberg (1999).

4.3 Implications of the layered analysis of computer simulations

We shall now provide a classification of the various ways of interpreting running computer programs and show that it allows for a better view of the implementation relation than Norton and Suppe's analysis does.[13] A major feature of this relation is that the computer's physical states in (1) can be divided into equivalence classes, each of which represents a computational state in (2), for example 0110...101. In a given computer, only a minuscule subset of physically discernible states are computationally discriminable. This implies that the definition of the computational states has to be relativized to some convenient mapping from physical states of the computer to computational states called Instantiation Function by Pylyshyn.[14] Owing to the Instantiation Function, it is possible to interpret a sequence of nomologically governed physical state changes as a sequence of computational states, namely, to view a physical object as a computer. The Instantiation Function is machine-relative, since there is no uniform way of giving a physical description of the equivalence class of physical states of a machine that correspond to each of its computational states. The final point to emphasize is that all that has been said about the Instantiation Function is not special to simulations but is a feature of computations. The semantic work allowing to validate results of computations as data_A is not done at this step.

Let us proceed further in our hierarchy. Not only do mathematical computations depend on an Instantiation Function, they also require another interpretation function, namely a Semantic Function that maps articulated functional (computational) states of the machine onto some domain of intended interpretation. In our case, the domain of intended interpretation is the set of values of discrete mathematical functions, themselves (sometimes) representing (sometimes approximately) other discrete or continuous functions (see below). The identification of the exact domain of the Semantic Function is required in order to answer question (P): "What computation is being performed?" (Pylyshyn 1984, Chap. 3) In our case, this question turns to question (P'): "What discrete mathematical function is being computed?" To quote Pylyshyn again, "the way the machine must work to be correctly described as following a semantic[15] rule [like computing the values of a given discrete mathematical function] is, on every occasion in which the rule is involved there must be physical properties of the machine's state capable of serving as physical code for that semantic interpretation" (Pylyshyn 1984, p. 68).

Let us take an example. Let us say that the states of the memory registers are designated by such expressions as 0, 1, 01, 10, 11, etc. Let us suppose that when a certain pattern, say "$\bigoplus$", occurs in a portion of the machine called the "instruction register", the machine's memory registers change their states according to a certain

[13] It also gives more detailed reasons as to why the three conditions given by Norton and Suppe are unlikely to be fulfilled.

[14] Pylyshyn's book provides us with a clear and consensus view about the different levels involved in a computer.

[15] It is semantic because it corresponds to the way the syntactic rules between the computational states are interpreted.

rule. For example, when the portion of the machine corresponding to register 1 is in a physical state that maps onto string 101 and the portion of the machine corresponding to register 2 is in a physical state that maps onto string 110, then register 3, whatever its physical state, changes to a physical state mapping onto string 1011. This illustrates what it means for the computer to be governed by a physical regularity that can be interpreted by means of the Instantiation Function as a rule between computational states. At a particular time in the computational process, this rule *might* be used to represent addition of numbers, provided we adopt the appropriate semantic function. In the example, the required semantic function is simply the binary number system, which maps strings of 0s and 1s onto numbers.

As we have shown in Sect. 3, the global time-independent mapping relation from physical states to values of variables representing the physical system is not in most cases an embedding relation because it is many–many. This is particularly obvious in our example. The same registers will be used at different times for different purposes and will therefore be interpreted by different Semantic Functions. Therefore, which physical state corresponds to which representing element at each time is partly fortuitous and keeps changing. This basic fact about computation threatens at its roots the view that the physical process of computation simply mimics the target physical process. The relation between the physical states of the computer and the values of representing variables can be many–one in a classical computer if the Simulating Portion is always located at the same physical place and if each part of this location always codes for the same part of the Simulating Portion. In this case, the physical states of the computer corresponding to this location do form an equivalence class and are indistinguishable from the point of view of their function in the machine's abstract computational description (Pylyshyn 1984, Chap. 3).

All this concerns computation proper, namely the way a Turing machine realizes a discrete mathematical function. Physicists do not have to be involved with these levels, which are the realm of computer scientists. The levels specific to simulations are the upper ones (see Sect. 4.4), where semantic mappings are more complicated.

4.4 The level of computational physics

Throughout the "lasagna" hierarchy, each level shows a particular "semantic sensitivity", namely, each level includes detailed features that may have unexpected semantic effects at the upper level(s). Therefore, each level's representational capacities have to be carefully assessed.

Programmers never deal with the computational states of the machine (2), or with the Semantic Function (see above). Instead, programs already packaged in languages such as C, Fortran, etc. enable them to select the discrete usual mathematical functions that they want to use such as, "+", creations of lists, operations on matrices, etc. Whenever the software writes the corresponding program at level (2), the Semantic Function is black-boxed. In practice, this black-boxing is totally reliable as far as elementary functions are concerned, except that continuous functions and real numbers are replaced by close discrete representations that are easier to compute—close, but different.

This is why when the exact discrete functions aimed at are replaced by easier-to-compute discrete functions,[16] undesirable effects might occur. For example, one may have to use an algorithm that only computes a Fast Fourier Transform within a given margin of error. Another case of semantic sensitivity to the program's details is the representation of real numbers by 32, 64, 80, or even more bytes. Switching from 32 to 64 ,bytes may improve the overall precision of the output, but can also generate huge changes on the final result whenever small scale phenomena govern the overall behavior of the system studied. In such cases, the exact discrete mathematical function that is computed may have important consequences at every higher level. This implies that the details of what is actually computed at each level influence the simulation's overall quality. In order to assess how reliable the discrete functions implemented actually are, one may rely on results from applied mathematics,[17] on successive trial and check methods, or again on calibration techniques when the exact results are know for some values (Humphreys 2004, pp. 116–121).

Semantic sensitivity may also appear at the articulation between discrete and continuous equations. What matters here is having the computed numerical solution converge towards the solution of the continuous equation and having the computation be stable. According to Lax' equivalence theorem, stability is a necessary and sufficient condition for convergence (Farge 1986, p. 164). Von Neumann analysis provides conditions for stability, but only in the case of simple linear problems.[18]

Let us conclude this semantic analysis with the articulation between levels (3) and (4), where even more semantic mappings come into play. Here we find the intricate hierarchies of models mentioned by both Hughes (1999) and Winsberg (1999). Winsberg describes the heart of the practice of computer simulation as a matter of building hierarchies of models: mechanical models in Cartwright's sense,[19] dynamical models, ad hoc models, computational models, and models of phenomena. He insists that on the way from the dynamical models, usually including partial differential equations, to level (3.1), simulationists usually encounter many obstacles, for the intractability of dynamical models may be computational as well as analytical. This means that difficulties do not only appear at the articulation of levels (3.2) and (3.1), but also within level (3.1), when there is not enough computational time and/or power to compute the various functions at this level. In order to obtain a satisfactory description at level (3.1), simplifying assumptions have to be made, degrees of freedom have to be removed, simpler empirical relationships have to be substituted for

[16] These functions can be programmed by the simulationist or are already packaged in if one uses ready-to-use simulation software.

[17] For example, even if the FFT exact algorithms are error free, the use of floating commas with finite precision may be a source of errors. For the ordinary implementation of the ordinary Fourier Transform, the errors are in $O(\varepsilon n^{3/2})$, whereas it is only in $O(\varepsilon \log(n))$ for the Cooley–Tukey algorithm (cf. Cooley and Tukey 1965).

[18] This is the reason why, in computational fluid dynamics simulations, the grid has to be fine-grained enough to have a Reynolds number below 1, i.e., the linear dissipative effects dominate the non-linear convective effects (ibid., pp. 164–165).

[19] "A mechanical model is a bare bones characterization of a physical system that allows us to use the [underlying] theoretical structure to assign a family of equations to the system", for instance, the description of a system as a damped oscillator" (Winsberg 1999, p. 279).

more complex ones (Winsberg 2001, p. S445). This is the price of a not-too-inaccurate representation at level (4). It is paid in a currency involving discrete mathematics as well as various skills transmuting dynamical models into computationally tractable ones.

Paying attention to the details of the computation, and in particular to the algorithms used, is the best way to avoid developing idealized conceptions of simulations. Whereas a quick analysis following the scientist's top-down move leads one to believe that simulations are just a way to find the solutions of theoretical models by implementing them, our analysis shows that the semantics of simulation is not that straightforward and that the assessment of any simulation's representational faithfulness is only the final step in painful semantic work.

5 Machines, after all

A distinctive feature of computer simulations, understood as running computer programs, is that they represent the successive states of a physical system in a way that excludes indeterminable parameters, since all the physical system's quantitative properties that are taken into account in the simulation must have determinate values (this is what we call "semantic saturation"). This is a consequence of the fact that, to be run, the program must be syntactically complete or saturated.

A syntactically incomplete program cannot run since it lacks elements that are necessary for any logical computation to be carried out. This implies that syntactic saturation is necessary for the simulation to run. That syntactic saturation is required also implies that it is usually difficult to obtain exactly the same values for similar simulations on different machines, and *a fortiori* in different labs, because the sub-programs or the random generators used to generate them are usually not all accessible to the simulationists. The different sub-programs or random generators do not bring about exactly the same sequences of 0s and 1s in each case. These sequences cannot generally be accessed; however, the details of these sequences can have different effects at upper levels.

In the conceptual framework we have presented, semantic saturation is a necessary property for a simulation to acquire any meaning at all. The property of semantic saturation makes of program (b) above a "saturated object" in Frege's sense, namely, an object that logically stands on its own. (A logical function, on the contrary, is semantically open: it needs to be completed by a logical constant to make a proposition and thus acquire a propositional meaning). Program (a), as semantically incomplete, cannot be said to represent any particular behavior of a physical system. In order to represent such a behavior, program (b) has to be completed by assigning values to all variables if the program is to run.[20]

[20] Claiming that syntactic and semantic saturation are required for program (b) to run should not be mixed up with (erroneously) assuming that simulations (in the sense of running computer programs) are irreducibly singular sequences of events. As computer programs, they can in principle be run in any place and at any time and still generate the same results. Their being implemented on physically different machines does not affect this property.

One might assume that there is a strong connection between semantic saturation and physical implementation. Both semantic saturation and physical implementation are necessary for any simulation to run. However, they are not necessary in the same sense. In order to understand why, two facts about simulations should be clearly distinguished:

– the fact that they are physical "incarnations" of theoretical models, i. e., that they are implemented on machines,
– and the fact that they are semantically saturated at the level of the mathematical functions they involve.

As we have shown, it is the *second* fact that enables simulations to generate data in the sense of "data about the investigated system". Semantic saturation is indeed a *logical* condition for a simulation to be a simulation, namely an object endowed with representational capacities. Were we to be given analytical solutions or computations made by an immaterial, fictitious being, data (in the sense of "data$_A$") would still be obtained and knowledge gained from these data. It should also be noted that physicists need not (and usually do not) have any idea about which exact physical processes realize the computations. Provided they know which exact mathematical computations have been performed and sent into the output files, they can work on these files, visualize some of the outputs, and interpret the outputs as data.

Why is the physical implementation so important after all? As described above, with the help of the Semantic Function, Turing machines or computers can be interpreted as computing mathematical functions. We can still run a Turing machine ourselves by following its transitions step after step. Physically implementing the Turing machine enables us to make physical systems work for us and do massive number crunching. As a consequence, in order to generate more data and make even bigger computations, one only needs to build bigger computers (up to a certain point). So physical implementation matters only in so far as it enables us to perform huge computations and thereby mechanically unfold what was contained in the discrete dynamical equations.

6 Concluding remark

Let us return to Hughes' claim, also shared by Humphreys and Norton and Suppe, that "the computer is a material system, and in using it to provide answers to our questions we are relying on the physical processes at work within it" (Hughes 1999, p. 139). We have shown that the physicality of the processes at work within the computer cannot explain E, namely that computer simulations, although they are basically computations and do not involve any measurement interactions, nevertheless do generate new data about empirical systems, just as field experiments do.

Acknowledgements We are grateful to Paul Humphreys for his comments on this paper, as well as to anonymous reviewers who helped us greatly improve the paper and make our views more precise. We also thank the participants to the first Models and Simulations meeting; Johannes Lenhard, Alfred Nordmann, Eric Winsberg and the participants to the workshop at the ZiF Bielefeld; as well as François Athané. This research was funded by the CNRS (ATIP "Physique et calcul").

References

Chassaing, P. (2000). *Turbulence en mécanique des fluides, analyse du phénomène en vue de sa modélisation à l'usage de l'ingénieur*. Cepaduès Editions.

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation, 19*, 297–301.

Farge, M. (1986). L'approche numérique en physique. *Philosophia Scientiae, 7*(2), 155–175.

Hartmann, S. (1996). The world as a process. Simulations in the natural and social sciences. In R. Hegselmann et al. (Eds.), *Modelling and simulation in the social sciences from the philosophy of science point of view* (pp. 77–100). Dordrecht: Kluwer.

Hartmann, S., & Frigg, R. (2006). Models in science. In E. N. Zalta (Ed.), *Stanford encyclopedia of philosophy*. Stanford University.

Hegselmann, R., Müller, U., & Troitzsch, K. (Eds.). (1996). *Modelling and simulation in the social sciences from the philosophy of science point of view*. Theory and Decision Library, Dordrecht: Kluwer.

Hughes, R. I. G. (1999). The Ising model, computer simulation, and universal physics. In M. Morgan & M. Morrison (Eds.), *Models as mediators. Perspectives on natural and social science* (pp. 97–145). Cambridge, UK: Cambridge University Press.

Humphreys, P. (1994). Numerical experimentation. In P. Humphreys (Ed.), *Patrick Suppes: Scientist Philosopher. Philosophy of physics, theory structure, and measurement theory* (Vol. 2, 103/121). Dordrecht: Kluwer.

Humphreys, P. (2004). *Extending ourselves. Computer Science, empiricism, and scientific method*. Oxford: Oxford University Press.

Lebanon, N. H. (1996). *Fluent Europe: Computational fluid dynamics software*. Fluent Inc.

Morgan, M., & Morrison, M. (Eds.). (1999). *Models as mediators. Perspectives on natural and social science*. Cambridge, UK: Cambridge University Press.

Norton, S., & Suppe, F. (2001). Why atmospheric modeling is good science. In P. Edwards & C. Miller (Eds.), *Changing the atmosphere* (pp. 67–106). Cambridge, MA: MIT Press.

Pylyshyn, Z. (1984). *Computation and cognition*. Cambridge, MA: MIT Press.

Rohrlich, F. (1991). Computer simulations in the physical sciences. In A. Fine, M. Forbes, & L. Wessels (Eds.), *PSA 1990* (Vol. 2, pp. 507–518). Philosophy of Science Association.

Winsberg, E. (1999). Sanctioning models: The epistemology of simulation. *Science in Context, 12*(2), 275–292.

Winsberg, E. (2001). Simulations, models and theories: Complex physical systems and their representations. *Philosophy of Science (Proceedings), 68*, S442–S454.

Winsberg, E. (2003). Simulated experiments: Methodology for a virtual world. *Philosophy of Science, 70*, 105–125.