The nature of computational models

Konrad Hinsen

Department: Scientific Programming

The nature of computational models

Konrad Hinsen, *Centre de Biophysique Moléculaire, CNRS, Orléans, France*

Abstract—Computational models lie at the heart of computational science. And yet, few scientists have a clear idea of what a computational model actually is. Is it software? Or an algorithm? How does it relate to mathematical models? What are suitable languages or notations for expressing a computational model in the literature? And will AI make computational models obsolete?

Introduction

Any scientific computation involves the application of one or more computational models. Some of these models seem so obvious that we hardly recognize them as such. For example, when we fit a straight line to a set of points, we use a computational model derived from two hypotheses: (1) the dependent quantity is a linear function of the independent quantity, and (2) measurement errors follow a normal distribution, with equal variance for all data points. At the other extreme, some models are so complex that they seem to be all there is to a piece of software. For example, what climate researchers call an Earth system model is really a *simulator* for a model, i.e. software that computes model predictions. There are also branches of computational science, in particular bioinformatics, that emphasize methods for solving problems over models representing the underlying phenomena, and as a consequence hardly discuss models at all. But the models are still there, in the form of assumptions about the systems under study that are implicit in the problem-solving algorithms.

If we condense the process of scientific research to its very basics, it is the creation and iterative improvement of models that describe the observations that we make of natural phenomena. Models and observations are thus the core concepts of science. Two long-standing specializations in many disciplines are the *theoretician*, who comes up with and refines models, and the *experimentalist*, who designs ingenious setups to make good observations.

In computational science, observations are represented as datasets. How about models? A frequent view is that models are represented as code. After all, "code and data" is an association that sounds like the perfect analogy to "theory and experiment". However, this analogy doesn't stand up to critical examination. Code is a technical artifact that involves many choices unrelated to modeling natural phenomena. For example, to write code, I have to choose a programming language, but that choice is completely unrelated to the phenomenon I wish to describe by my model. A computational model can be implemented in any programming language, and should yield identical results independently of that choice.

But then, what *is* a computational model? In the following I will argue that it is a (partial) *specification* of an algorithm or program. As part of my argument, I will explain how specifications, algorithms, and data are analogous to equations, functions, and numbers in mathematics. In mathematics-based science, observations are numbers, relations between observations and parameters are functions, and scientific models are equations. In computational science, observations are data, relations between observations and parameters are algorithms, and computational models are specifications.

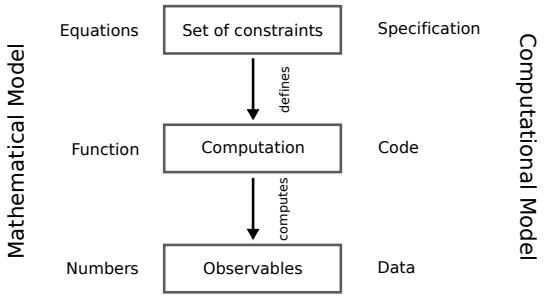

FIGURE 1. Mathematical and computational models in science. Computations can be written down directly as a function or as executable code, but defining them via a set of constraints, i.e. equations or specifications, results in modular models that can be generalized and provide more insight.

Digital Object Identifier 10.1109/XXX.0000.0000000

Mathematical models in science

Quantitative mathematical models play a very important role in scientific disciplines as diverse as physics, biology, and economy. Such models describe relations between observable quantities and system parameters such as time. The simplest form such a relation can take is a function, which expresses one quantity in terms of other quantities and parameters. This approach can easily be generalized to multiple functions, or a single multi-valued functions, for the case of multiple dependent quantities. Models expressed as functions often have adjustable parameters, whose values are determined from observations.

Two historically important examples of function-based models are Ptolemy's and Kepler's models for orbits of celestial bodies. Ptolemy's model describes the orbit of the Sun as seen from the Earth as a circle with superposed epicycles, an idea similar in spirit to Fourier analysis. The radii and relative phases of the cycles are parameters fitted to the observations. Kepler's model states, in modern language, that planets move on elliptical orbits around the Sun, which occupies one of the two focal points. Each ellipse has two adjustable shape parameters, which must be determined from observations of the planet's positions. Kepler's model also makes a quantitative statement about the velocity at which the planets move, accelerating as they approach the sun and slowing down as they move away from it, respecting what in modern language is called the conservation of angular momentum.

Historical discussions describe Kepler's model as superior to Ptolemy's because it chose the Sun, rather then the Earth, as the reference point for celestial orbits. Another major progress is rarely discussed. Ptolemy's model is very generic. Any closed curve can be described as a superposition of circular motions, in the spirit of Fourier analysis. But a good approximation takes many epicycles and thus many fit parameters. Kepler puts the shape of the orbits into the functional form of his model, reducing the number of parameters to two. In general, given two models with adjustable parameters that predict observations equally well, the one with *fewer* parameters is the more powerful one, because it captures more of the regularities of the data. This is a lesson that modern data science is just rediscovering.

A more advanced kind of mathematical model consists of equations, whose solutions are the functions that relate observable quantities and system parameters. Since a function can be regarded as a special case of an equation in which nothing is unknown, equation-based models are a superset of function-based models. In general, equations are *constraints* on functions. Equations are more powerful than functions as scientific models because they can be arbitrarily composed. Functions can of course be composed as well, but not arbitrarily. You can chain together two functions, but the value of one function must then be a valid argument to the second function. Equations can just be lumped together. At worst, the combination has no solution, or too many to be of practical interest. Each equation can focus on one specific aspect of the overall model. The use of equations thus makes mathematical models *modular*. Much of the insight that such models provide comes from the study of its component equations (1).

A good illustration is Newton's model for celestial mechanics, which was a huge improvement on Kepler's ellipses. It consists of the combination of two equations. One of them is a general principle of (classical) mechanics: the force acting on an object is equal to its mass times its acceleration. The other equation is the law of gravitation, which states that two masses attract each other with a force that is proportional to each mass and inversely proportional to the square of the distance between them. It is this separation into a general principle, applicable to many other situations, plus a description of specific interactions, that has turned classical mechanics into one of the most successful and practically important scientific theories of all times.

Kepler's ellipses are a solution of Newton's equations. But Newton's equations go far beyond those ellipses, even if we consider only celestial mechanics. The equations show that ellipses are an exact solution only for one planet orbiting a single star. In our multi-planet solar system, elliptical orbits are only an approximation, useful if the gravitational attraction between planets is small enough to be neglected. Better yet, Newton's equations tell you how to improve the solutions by taking into account these interactions. Moreover, Newton's equations show that there are also non-elliptical solutions: the parabolic orbits of celestial bodies arriving from infinite space and never returning to their place of origin. Finally, Newton's equations permit the derivation of conservation laws, e.g. for energy and angular momentum, that greatly facilitate reasoning about possible solutions even without computing them in detail.

This celestial mechanics example illustrates how function models and equation models tend to play different roles in science. Functions, usually with fitted parameters, are used to summarize observations in a compact form, and separate regularities from the inevitable noise or errors in any measurement. Such

a *descriptive* model (Fig. 2) can be used to compute theoretical values for past and future observations. But it is not generalizable to similar though not identical situations. Equation models are usually *explanatory*: they describe the observations as the outcome of supposedly more fundamental mechanisms, such as kinetics and gravitation, which are applicable in a much wider range of situations (Fig. 3).

Algorithms and specifications

In the realm of computations, the equivalent of a mathematical function is an algorithm. It takes data items as input, and produces one or more data items as output. The inputs thus correspond to a function's arguments, and the outputs to the function's value. This very suggestive correspondence is the reason why many programming languages use the term "function" for code items that implement an algorithm. It is, however, exact only if the algorithm does nothing else than return its outputs: it may not change data elsewhere in memory, nor change what is displayed on a screen, nor send data over a network connection, etc. Under that condition, functions and algorithms can almost be considered synonyms in the context of scientific models, leaving aside mostly theoretical subtleties related to the non-computability of real numbers. Lifting the condition of no effect other than returning outputs, we thus find that algorithms are a generalization of functions.

A commonly held view is that computational models *are* algorithms. Many if not most computational models used today are indeed algorithms, because they are *derived* from mathematical models as solution algorithms. That's a topic I will discuss later. First, I want to address another question: is there something equivalent to equations in the realm of computation? The answer is yes, and that something is called a *specification* [3]. A specification describes what an algorithm or a program is expected to do. Like a set of equations, a specification consists of a set of constraints on an algorithm's or program's behavior. In fact, equations are one possible form of constraint on an algorithm, and therefore a specification is a generalization of a set of equations in the sense that it allows for more types of constraints.

Specifications come in two varieties: *formal* and *informal*. They are the extremes of a scale rather than neatly separate categories. A formal specification is expressed in a formal language, which is a language whose syntax and semantics are precisely defined, making it suitable for processing by computers. An informal specification is plain prose written for human readers, though it must try to be as precise as possible to be of any practical value. In the gray zone between the two extremes, we have semi-formal specifications, a category that includes most of traditional mathematics. Semi-formal specifications are amenable to formal reasoning, such as "multiplying both sides of an equation by the same number yields an equally valid equation", but the set of applicable rules is context-dependent and not itself formalized, as it would have to be to become part of a computer program. This is one reason why the formalization of scientific models often leads to more insight and understanding, in addition to providing a useful computational tool [4].

Algorithms are usually formulated as informal specifications in narratives, or as semi-formal specifications called pseudo-code. When expressed in a programming language, an algorithm becomes executable, and then we call it a program. However, as I explained above, specifications are a more general notion than algorithms. We can thus have formal specifications that are not executable, because they are only constraints on the results produced by a program. This requires particular formal languages, called specification languages, which can express both algorithms and constraints on algorithms,

In software engineering, formal specifications are mainly used for verifying that a program behaves as expected, via formal proofs, tests, or other techniques. The role of the specification is to provide a formulation of the expected behavior of a software system that is simpler and closer to user concerns than an implementation written in a programming language. As an example, consider the task of sorting a list. The specification of this task is: produce a new list whose elements are (1) the same as those of the input list and (2) sorted. The specification does not say *how* this new list should be constructed, and there are indeed many possible solutions, including well-known sorting algorithms such as quicksort or bubble sort. Additional constraints, e.g. on the use of resources (memory, CPU time, ...), can be added to the specification to narrow down the set of acceptable solutions.

Although the principal use of specifications today is the verification of programs, there are also formalisms to *derive* executable code from a formal specification. They are similar in spirit to the techniques that mathematicians have developed to find solutions to equations. One example is the Bird-Meertens formalism [1].

Formal specifications are a good framework for defining computational models for several reasons:

1) They can be processed by computers.
2) They contain mathematical equations, expressed

Mathematical model	Celestial mechanics example	Computational model
function with fitted parameters	Ptolemy: circles with epicycles	machine learning

FIGURE 2. *Descriptive models* summarize a set of observations, separating perceived regularities from noise and measurement errors. Traditionally, they take the form of mathematical functions with fit parameters. Machine learning models are the computational generalization of this idea.

Mathematical model	Celestial mechanics example	Computational model
equations	Newton: (1) $\mathbf{F} = m \cdot \mathbf{a}$, (2) $\mathbf{F}_{ij} = -Gm_i m_j \hat{\mathbf{r}}_{ij} / r_{ij}^2$	specifications
general solution (parametric)	Kepler: elliptical orbits, sun at focal point, constant area speed	algorithm (with input parameters)
specific solution	Earth's orbit around the Sun	computed result

FIGURE 3. *Explanatory models* describe fundamental mechanisms by equations, whose composition defines *general solutions*, in which parameters remain to be fixed in order to obtain *specific solutions*.

in a suitable formal language, as a special case.
3) They contain explicitly formulated algorithms as a special case, and thus also computable mathematical functions.
4) They retain the modularity of sets of equations.

The most important obstacle to the use of formal specifications in computational science is the lack of specification languages suitable for this use case. In contrast to the huge number of programming languages in use today, there are very few specification languages, which moreover tend to have a narrow application focus in software engineering. Today, computational models in science usually have two unconnected representations: an informal and typically incomplete specification as part of a paper or textbook, and an executable implementation as a computational tool, in which the modularity of the specification is lost.

Derived and digital native computational models

Most computational models in use today are derived from mathematical models. In the simplest case, which is a mathematical model already formulated as a computable function, this only involves approximating the real numbers in the mathematical model by floating-point numbers. The linear regression I mentioned in the introduction is a typical example. Another one is the computation of an integral using Simpson's rule. Substituting real numbers by floating-point numbers may seem trivial, and some programming languages support this impression of triviality by calling floating-point numbers "real". However, this substitution is a non-trivial approximation that requires considering the impact of finite precision and non-associative arithmetic [2].

Another frequent kind of derived computational model computes a numerical solution for a set of equations. Whole subfields of computational science and engineering, e.g. computational fluid dynamics, are based on computational models derived from differential equations via some discretization scheme. Deriving algorithms for numerical solutions of mathematical equations is the topic of *numerical analysis*, a field of research that predates computers by many centuries. Indeed, numerical solutions to mathematical equations have been computed by hand almost since the beginning of science, because this was often the only available technique for obtaining numbers that could be compared with observations.

Computers have led to the development of alternatives to numerical analysis for deriving computational models from mathematical equations. The best-known technique of this kind is the *Monte Carlo method* for computing integrals by interpreting them as probability densities. Another example are *lattice gas automata* and *Lattice Boltzmann methods*, which are techniques to solve the equations of fluid dynamics (the Navier-Stokes equations) by simulating a dynamical system that is convenient for computers (moving particles on a lattice by applying simple rules) and whose averages over sufficiently large space and time regions can be shown to behave much like a real fluid. These models were inspired by *cellular automata*, which were perhaps the first "digital native" models, i.e. computational models developed without any reference to traditional mathematical models, based on ideas developed in the 1940s. Chapter 8 of Stephen Wolfram's book "A New

Kind of Science" [5] describes computational models based on cellular automata for various natural phenomena. It contains a good discussion of the differences between digital native models and models based on mathematical equations.

Another class of digital native model consists of *agent-based models*. Like cellular automata, they are explanatory models, used in simulations from which observable quantities are computed in the end. An agent-based model describes the behavior of a system by focusing on its constituents, called agents, and their interactions with other agents, expressed as a set of rules. The term "agents" suggests humans, and agent-based models are indeed widely used to model human behavior, for example in the spread of epidemics. But the agents are not required to exhibit complex or autonomous behavior, and can be as simple as molecules in a model of chemical reactions.

Descriptive digital-native models have attracted a lot of attention recently. Their formulation and exploitation is commonly called *data science*. One popular family of models in this category is the *network model*. It describes some entities as the nodes of a graph, and relations between the entities as the graph's edges. The entities can be physical (molecules, people, cities, ...) or symbolic (e.g. concepts in a knowledge graph, or scientific papers in a citation graph), and the relations can be qualitative or quantitative, described by weighted edges. Network models are studied by techniques from graph theory, and yield mostly statistical inferences about the importance or roles of various entities or groups of entities in the network.

Another very important family of models in data science is collectively called machine learning models. Within this family, the best-known model type is the neural network. It should not be confused with the network model described above: in a network model, the network represents the structure of the data, whereas in a neural network, the network represents the relations between small mathematical functions, called artificial neurons because they were inspired by the behavior of biological neurons.

In principle, a neural network is just a mathematical function with a very large number of adjustable parameters. However, the way neural networks are used is quite different from traditional mathematical function models. In the latter, the functional form encodes the scientific hypotheses of the model, with the parameters serving only for fine-tuning the generic model to a concrete system. In contrast, a neural network can by design represent almost any function. This is known as the universal approximation theorem [6]. By itself, a neural network thus contains no scientific hypothesis at all. Training a network on a dataset of observations is more similar to lossy data compression than to traditional function fitting. Computing predictions from the trained network is then similar to interpolating and extrapolating an approximation to the original dataset.

However, machine learning techniques are evolving quickly, and integrating prior information, such as physical laws, is one of the main strategies for improving neural networks. One way to do this is via the architecture of the network, i.e. the patterns of connections between the neurons. Convolutional networks were the first such problem-specific architecture, designed for image processing. They enforce translational invariance, thus ensuring by construction, rather than as an outcome of the training process, that a cat is recognized in exactly the same way no matter if it's in the top right corner of the image or in the center. Recurrent networks similarly impose a specific architecture for modeling sequential data. A more recent technique, differentiable programming, permits arbitrary combinations of numerical functions expressed as code and numerical functions expressed as neural networks. Large language models, which recently found their way into software for a general public, such as search engines, impose some of the basic structure of human languages on a neural network, allowing it to learn sequences of phrases that contain references to each other.

Another approach to injecting prior knowledge into neural networks is a modification of the *loss function* used in training. The loss function is the quantity that the training process aims to minimize. The main ingredient in constructing the loss function is the deviation of the network's predictions from the training data. Physics-informed neural networks add another term that measures how much the prediction deviates from specified physical laws, which are typically differential equations. This differs from the network architecture approach in that the prior knowledge is used as a soft rather than a hard constraint.

All these techniques are already explored by scientists in many disciplines, with the goal of identifying their strengths and limitations. In parallel, the epistemic status of machine learning models in science is a topic of debate, and likely to occupy philosophers of science for a long time to come. Plain neural networks do not contain scientific hypotheses, meaning that they are not really scientific models. But the more advanced network structures raise the question whether the traditional separation of models and observations still makes sense. It was never a completely accurate perspective on science. Fitted parameters, i.e. hybrid entities depending on both models and observations, have

played an important role in science for centuries. Moreover, the design of any experimental setup depends on input from models. Data thus always depended on models in some way. With machine learning, we get models that strongly depend on data. We can compute predictions using these models, but how to derive scientific insight from them remains an open question. We certainly live in interesting times!

REFERENCES

1. J. Gibbons, “The School of Squiggol: A History of the Bird-Meertens Formalism”, in *Formal Methods, FM 2019 International Workshops*, LNCS Vol. 12233, 2020, pp 35-53.
2. N. Toronto, Jay McCarthy, “Practically Accurate Floating-Point Math”, *Comput. Sci. Eng.*, vol. 16, no. 4, pp. 80–95, 2014.
3. K. Hinsen, “Writing software specifications,”, *Comput. Sci. Eng.*, vol. 17, no. 3, pp. 54–61, 2015.
4. G.J. Sussman, J. Wisdom, “The Role of Programming in the Formulation of Ideas”, AI Memo 2002-018, Artificial Intelligence Laboratory, MIT, https://dspace.mit.edu/handle/1721.1/6707
5. S. Wolfram, “A new kind of science”, Wolfram Media, Champaign, IL, 2002, http://www.wolframscience.com/nks/
6. M. Nielsen, “Neural Networks and Deep Learning”, Chapter 4, Determination Press, 2015, http://neuralnetworksanddeeplearning.com/chap4.html

Konrad Hinsen is a researcher at the Centre de Biophysique Moléculaire in Orléans and at the Synchrotron SOLEIL in Saint Aubin. His research interests include protein structure and dynamics and scientific computing. Hinsen has a PhD in theoretical physics from RWTH Aachen University. Contact him at konrad.hinsen@cnrs.fr.