Can Differential Equations Compute?

Technical Report UT-CS-01-459

Bruce J. MacLennan*

Computer Science Department
University of Tennessee, Knoxville
maclennan@cs.utk.edu

May 8, 2001

Abstract

Computationalism is based on the idea that cognition is computation, but connectionism raises the question of whether or not a physical system obeying differential equations is computing. This issue may be addressed by drawing several important distinctions: (1) among various forms of discrete and continuous computational processes, (2) between these abstract processes and their concrete physical realizations, (3) between these abstract and concrete processes and their descriptions (e.g. as equations), and (4) between two senses of "solving" a set of equations. Based on these clarifications, it is argued that whether a biological system is computational depends on its function in a biological context.

Keywords: analog computation, computationalism, differential equation, connectionism, dynamical hypothesis, multiple instantiability, function, symbolic

1 Introduction

Can differential equations (DEs) compute? This issue arises only because of a confluence of two ideas. On the one hand we have computational theories in cognitive science, inspired by the digital computer and Turing machines, and their philosophical consequences such as we find in Searle's Chinese Room Argument. On the other hand we have the rise of connectionist and dynamic-systems theories in cognitive science, which are based on analog computation and differential equations. Analog computation challenges many of the assumptions routinely made about (digital) computation as embodied in cognitive science.

But is analog computation genuine computation? And if so, what distinguishes an analog computer, obeying DEs, from other physical systems obeying DEs? Or as it is sometimes put, must we say that the solar system is computing (viz., Kepler's laws), and if not, why not?

I will give a brief answer to this question here, since I have already discussed it in more detail elsewhere (MacLennan 1994b). In general, I have tried to avoid technical terms (e.g. "symbol") that may be loaded with misleading connotations.

2 Computation

We may define computation as a mathematical process involving mathematical relations among mathematical objects. By "mathematical" I do not intend to restrict computation to numbers or other quantitative objects, but to stress their abstractness and formality. (That is, for Platonists, they exist in the realm of "forms.") For a simple example of a computation we may take a continuous, two-dimensional Fourier transform, such as might be applied to an image in an artificial or natural vision system.

A computational process is defined over a mathematical domain, which may be continuous or discrete, and the process may proceed in discrete or continuous abstract time. (Continuous time corresponds to the ordinary, real continuum; discrete time corresponds to a sequence of isolated instants, for which order is the only significant property; see also van Gelder 1998.) Thus we may distinguish three distinct kinds of processes and hence of computations:

Type **C**: a continuous-time process over a continuous state-space.
Type **CD**: a discrete-time process over a continuous state-space.
Type **D**: a discrete-time process over a discrete state-space.

(The fourth possibility, a continuous-time process over a discrete state-space, does not seem to be a coherent possibility, so far as I can see; apparent examples turn out to be **C** or **D** when analyzed carefully.) We can also have a hybrid computation, but its individual components or phases will be one of these three types. I should add that these types are mathematically defined and exact; it is not an imprecise classification. This classification is based on two common abstractions of time: as a one-dimensional continuum or as a sequence of discrete events. (There are other, less-common abstractions of time, which may be in fact more relevant to psychology.)

Examples: Differential equations define type **C** processes. Conventional algorithms over the integers define type **D** processes. An iterative algorithm over the real numbers (such as Newton's algorithm) defines a type **CD** process.

In passing I should mention that computations are often representational. That is, there is often some systematic mapping from the states, processes, etc. to a "domain of interpretation." However, I have shown in MacLennan (1994b) that such interpretability is neither a necessary nor a sufficient property of computation.

3 Realizations

Next we may consider physical realizations of computational processes; we may call these *computers* for convenience. Here we have a physical process involving physical relationships among physical states (objects, quantities, etc.) proceeding in real (physical) time.

A physical realization of a computation is exact when the physical system is capable of representing all the relationships of the mathematical system (i.e., the relationships relevant to the computation). Technically, there is a homomorphism from the physical system to the mathematical system: the physical system has all the mathematical structure needed for the computation, but may have additional, irrelevant structure. (For more on realizations see MacLennan 1994a, 1994b.)

However, many realizations are only approximate; that is, the physical computer does not realize perfectly the abstract computation. For example, a physical device for computing a Fourier transform may not do it accurately for all possible inputs (it may be possible to overload it, it may introduce noise or error, and so forth).

An approximate physical realization of a computation need not be of the same type as the abstract computation. That is, a continuous physical system may (approximately) realize a discrete computational process or vice versa. (For example, we may use digital computation to realize approximately a continuous 2D Fourier transform.)

However, it is important to be clear that we are considering the discreteness or the continuity of the physical system at the relevant level of analysis (which may not be unique and is often determined pragmatically). That is, while it is absolutely precise whether a computation (qua mathematical system) is continuous or discrete, it is somewhat open whether an approximate realization is continuous or discrete. (It is normally appropriate to treat a digital device as discrete, but at a lower level of analysis it's continuous, and so forth.)

The ability to have alternative physical realizations is what constitutes the "multiple instantiability" of computational processes. On the one hand this is simply a consequence of the fact that computational processes are abstract, formal, mathematical processes, which can be instantiated in differing concrete, physical systems. On the other hand, what makes such physical systems computational is that their sole function is the realization of the abstract system. (Thus, establishing a physical system as computational, i.e., as realizing a computation, requires establishing the system's function, an issue addressed in Section 6.)

4 Descriptions

We must distinguish a computational process from mathematical descriptions of that process. Thus a Fourier transform might be described by a certain integral, and other computations may be described by differential equations (DEs) or finite difference equations (FDEs). Mathematical descriptions are discrete representations, i.e., they are expressed in a language of finite formulas composed of discrete tokens belonging to a finite alphabet of discrete, definite types.

It is *not* true that all computations can be described mathematically — even by a (definite) sequence of approximations. The simplest way to see this is to observe that the set of all possible mathematical descriptions is denumerable, whereas the set of all type **C** or **CD** processes is not denumerable. (To understand why, observe that most real numbers are not describable, even by a computable sequence of approximations, since the set of such approximations is denumerable, whereas the real numbers are nondenumerable.)

Another reason it is important to distinguish a computational process from descriptions of that process is that typically there are multiple descriptions of the same process. For example, the same continuous-time process might be described as (open-form) differential equations or as a closed-form solution to those equations. Also, a computation might be described approximately, if that is more useful (e.g. for mathematical manipulation of the description).

So the ontological position I am implying is that a computation, as realized for example by various physical systems, is prior to any descriptions we may formulate of that computation. This implication is especially important when dealing with computation in a biological context. (I will deal in Section 5 with computations that are generated from a description, e.g. the execution of computer programs.)

Likewise we must distinguish the physical processes realizing a computation from descriptions of those processes (i.e., distinguish the computer from descriptions of the computer). Typically the physical system can be analyzed at many different levels and described in different ways at each of these levels. Thus, at one level a flip-flop in a digital computer changes state discretely; at another it obeys DEs and changes state continuously.

5 Solving Equations

Next I'll say a few words about differential equations (DEs), finite difference equations (FDEs) and their solution. Both DEs and FDEs describe a process in terms of the change of state at a single, arbitrary point in time. For a DE the state is described as changing in continuous time, for an FDE it changes in discrete time. (This of course is the mathematical description; the system itself need not behave the same way and the description can still be approximately correct, as, for example, when a continuous process is described by FDEs.) "Open-form" descriptions of processes (such as DEs and FDEs), which describe state change at a single, arbitrary point in time, may be contrasted with "closed-form" solutions of these equations, which give the state directly, independently of its history.

But we must be careful when we talk about "solving" DEs or FDEs. When a mathematician talks of solving a DE, he or she usually means finding a closed-form solution to the DE. Thus he or she is manipulating one discrete, symbolic structure (the DE) to get a different discrete, symbolic structure (a formula for the closed-form solution). A finite difference equation can be solved in exactly the same way and by exactly parallel techniques (one of the beautiful symmetries in mathematics). (It is no coincidence that the primary tool in each case is a calculus [a type **D** computational device]: the integral/differential calculus for DEs, the summation/difference calculus for FDEs.)

We must distinguish this sense of "solving" a DE (or FDE) from "simulating" the system described by it. In the latter case we construct a system that follows or obeys the equations; i.e., we construct a system for which the equations are a good (perhaps approximate) model. In such a case a mathematician may speak of "integrating" or "numerically solving" DEs, but that is an extended sense of "solving," since it is usually used when solving the equations in the primary sense is impossible. However, it is better to speak of a simulation, since the behavior unfolds in time as described by the DEs or FDEs.

This sort of simulation is what takes place in a general-purpose computer of any kind (analog or digital). Such a computer is given an open-form description of the process (e.g. DEs or FDEs), which it then simulates with more or less fidelity. The description is obeyed by the general-purpose computer, so that for the nonce it becomes a special-purpose computer realizing the desired specific computation.

I should mention in passing that I am restricting my attention to finite, discrete descriptions of processes, since that is what is meant normally by a "description," i.e. a mathematics- or language-like description. On the other hand, it is possible to entertain continuous "descriptions" (representations), and continuous "descriptive images" of this sort may be important in the theory of general-purpose or universal analog computers; see MacLennan (1995).

6 Conclusions

Perhaps now I can bring this discussion back to the question of whether DEs compute and directly address some questions raised in email from Stevan Harnad (1999).

> "What is the difference between differential and difference equations when considering their computational abilities?"

Both DEs and FDEs may be used to describe computational processes. DEs are best for describing type **C** computations; FDEs are best for types **CD** and **D**. (FDEs are traditionally defined over numerical state spaces, but they may be extended to non-numerical domains [MacLennan 1989]. In this extended sense, all conventional [i.e. digital] computer programs are FDEs. Similarly, DEs may be extended to non-numerical domains.)

FDEs may approximate DEs and vice versa. (I do not believe that we have yet the appropriate theoretical framework for addressing differences in the computational power of DEs and FDEs, arguments about Turing Machines approximating DEs notwithstanding.)

"Why/how is solving differential equations not 'computation'?"

There are several senses in which "solving DEs" may be computation. If by "solving" you mean manipulating the mathematical expression of the DEs to get a mathematical formula for the closed-form solution, then this (to the extent it's algorithmic) is computation of the familiar discrete, symbol-manipulating variety. Alternately, we may take "solving" in the extended sense, in which a system is "solving" a system of DEs when its behavior can be described (perhaps approximately) by those DEs. Here, the description of the system by DEs has nothing to do with whether it's computation or not. Many systems may be described by DEs, but are not computing. (And, conversely, computations need not be describable by DEs.)

For example, in an analog computation certain abstract quantities may be represented as well by water pressure and flow rate as by electrical voltage and current. However, your electrical power outlet cannot fulfill its function by delivering a 60 Hz water stream, even though it may be described by the same DEs.

So we have the question of whether a particular physical system, described by DEs, is computing. That is, we must ask whether the function of the physical system is to realize some abstract computation. (E.g. a lens can realize a continuous 2D Fourier transform, but that does not imply that every lens is performing that computation.) This question can be answered empirically, but we must acknowledge that establishing function may be problematic, especially when dealing with biological systems, which often have multiple functions. "Multiple instantiability" provides one operational test, but it will not be possible always to say whether a system is purely computational; there will also be borderline cases. Nevertheless I think we can safely say that many brain systems are computational in the general sense outlined here (i.e., types **C** and **CD** as well as **D**). (See MacLennan 1994b for more on identifying computation in biological systems.)

7 References

1. Harnad, S. (1999) Why are differential equations not algorithms? Letter posted to "cm302 skywriting list," 4 May 1999.

2. MacLennan, B. J. (1989). *The Calculus of Functional Differences and Integrals*, University of Tennessee, Knoxville, Department of Computer Science Technical Report CS-89-80, April 1989.

3. MacLennan, B. J. (1993) Grounding analog computers (commentary on S. Harnad, "Grounding symbols in the analog world with neural nets"). *Think* **2**, June 1993, pp. 48–51. `cogprints.soton.ac.uk/abs/comp/199906003`

4. MacLennan, B. J. (1994a) Continuous computation and the emergence of the discrete. In *Origins: Brain & Self-Organization*, edited by Karl Pribram, Hillsdale, NJ: Lawrence Erlbaum, pp. 121–151. `cogprints.soton.ac.uk/abs/comp/199906001`

5. MacLennan, B. J. (1994b) "Words lie in our way." *Minds and Machines*, special issue on "What is Computation?" Vol. 4, No. 4, pp. 421–437. `cogprints.soton.ac.uk/abs/phil/199906001`

6. MacLennan, B. J. (1995) Continuous formal systems: A unifying model in language and cognition. In *Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems*, August 27–29, 1995, Monterey, CA, pp. 161–172. `cogprints.soton.ac.uk/abs/comp/199906002`

7. Van Gelder, T. J. (1998) The dynamical hypothesis in cognitive science (with commentaries). *Behavioral and Brain Sciences* **21**: 615–65.