E General-purpose analog computation

E.1 The importance of general-purpose computers

Although special-purpose analog and digital computers have been developed, and continue to be developed, for many purposes, the importance of general-purpose computers, which can be adapted easily for a wide variety of purposes, has been recognized since at least the nineteenth century. Babbage's plans for a general-purpose digital computer, his *analytical engine* (1835), are well known, but a general-purpose differential analyzer was advocated by Kelvin (Thomson, 1876). Practical general-purpose analog and digital computers were first developed at about the same time: from the early 1930s through the war years. General-purpose computers of both kinds permit the prototyping of special-purpose computers and, more importantly, permit the flexible reuse of computer hardware for different or evolving purposes.

The concept of a general-purpose computer is useful also for determining the limits of a computing paradigm. If one can design—theoretically or practically—a *universal computer*, that is, a general-purpose computer capable of simulating any computer in a relevant class, then anything uncomputable by the universal computer will also be uncomputable by any computer in that class. This is, of course, the approach used to show that certain functions are uncomputable by any Turing machine because they are uncomputable by a universal Turing machine. For the same reason, the concept of general-purpose analog computers, and in particular of *universal analog computers* are theoretically important for establishing limits to analog computation.

E.2 General-purpose electronic analog computers

Before taking up these theoretical issues, it is worth recalling that a typical electronic GPAC would include linear elements, such as adders, subtracters, constant multipliers, integrators, and differentiators; nonlinear elements, such as variable multipliers and function generators; other computational elements, such as comparators, noise generators, and delay elements (Sec. B.1.b). These are, of course, in addition to input/output devices, which would not affect its computational abilities.

E.3 Shannon's analysis

Claude Shannon did an important analysis of the computational capabilities of the differential analyzer, which applies to many GPACs (Shannon, 1941, 1993). He considered an abstract differential analyzer equipped with an unlimited number of integrators, adders, constant multipliers, and function generators (for functions with only a finite number of finite discontinuities), with at most one source of drive (which limits possible interconnections between units). This was based on prior work that had shown that almost all the generally used elementary functions could be generated with addition and integration. We will summarize informally a few of Shannon's results; for details, please consult the original paper.

First Shannon offers proofs that, by setting up the correct ODEs, a GPAC with the mentioned facilities can generate any function if and only if is not hypertranscendental (Theorem II); thus the GPAC can generate any function that is algebraic transcendental (a very large class), but not, for example, Euler's gamma function and Riemann's zeta function. He also shows that the GPAC can generate functions derived from generable functions, such as the integrals, derivatives, inverses, and compositions of generable functions (Thms. III, IV). These results can be generalized to functions of any number of variables, and to their compositions, partial derivatives, and inverses with respect to any one variable (Thms. VI, VII, IX, X).

Next Shannon shows that a function of any number of variables that is continuous over a closed region of space can be approximated arbitrarily closely over that region with a finite number of adders and integrators (Thms. V, VIII).

Shannon then turns from the generation of functions to the solution of ODEs and shows that the GPAC can solve any system of ODEs defined in terms of non-hypertranscendental functions (Thm. XI).

Finally, Shannon addresses a question that might seem of limited interest, but turns out to be relevant to the computational power of analog computers (see Sec. F below). To understand it we must recall that he was investigating the differential analyzer—a mechanical analog computer—but similar issues arise in other analog computing technologies. The question is whether it is possible to perform an arbitrary constant multiplication, $u = kv$, by means of gear ratios. He show that if we have just two gear ratios a and b ($a, b \neq 0, 1$), such that b is not a rational power of a, then by combinations of these gears we can approximate k arbitrarily closely (Thm. XII). That is, to approximate multiplication by arbitrary real numbers, it is sufficient to be able to multiply by a, b, and their inverses, provided a and b are not related by a rational power.

Shannon mentions an alternative method of constant multiplication, which uses integration, $kv = \int_0^v k \mathrm{d}v$, but this requires setting the integrand to the constant function k. Therefore, multiplying by an arbitrary real number requires the ability to input an arbitrary real as the integrand. The issue of real-valued inputs and outputs to analog computers is relevant both to their theoretical power and to practical matters of their application (see Sec. F.3).

Shannon's proofs, which were incomplete, were eventually refined by Pour-El (1974a) and finally corrected by Lipshitz & Rubel (1987). Rubel (1988) proved that Shannon's GPAC cannot solve the Dirichlet problem for Laplace's equation on the disk; indeed, it is limited to initial-value problems for algebraic ODEs. Specifically, *the Shannon–Pour-El Thesis* is that the outputs of the GPAC are exactly the solutions of the *algebraic differential equations*, that is, equations of the form

$$P[x, y(x), y'(x), y''(x), \ldots, y^{(n)}(x)] = 0,$$

where P is a polynomial that is not identically vanishing in any of its variables (these are the *differentially algebraic* functions) (Rubel, 1985). (For details please consult the cited papers.) The limitations of Shannon's GPAC motivated Rubel's definition of the Extended Analog Computer.

E.4 Rubel's Extended Analog Computer

The combination of Rubel's (1985) conviction that the brain is an analog computer together with the limitations of Shannon's GPAC led him to propose the *Extended Analog Computer* (EAC) (Rubel, 1993).

Like Shannon's GPAC (and the Turing machine), the EAC is a conceptual computer intended to facilitate theoretical investigation of the limits of a class of computers. The EAC extends the GPAC in a number of respects. For example, whereas the GPAC solves equations defined over a single variable (time), the EAC can generate functions over any finite number of real variables. Further, whereas the GPAC is restricted to initial-value problems for ODEs, the EAC solves both initial- and boundary-value problems for a variety of PDEs.

The EAC is structured into a series of levels, each more powerful than the ones below it, from which it accepts inputs. The inputs to the lowest level are a finite number of real variables ("settings"). At this level it operates on real polynomials, from which it is able to generate the differentially algebraic functions. The computation on each level is accomplished by conceptual analog devices, which include constant real-number generators, adders, multipliers, differentiators, "substituters" (for function composition), devices for analytic continuation, and inverters, which solve systems of equations defined over functions generated by the lower levels. Most characteristic of the EAC is the "boundary-value-problem box," which solves systems of PDEs and ODEs subject to boundary conditions and other constraints. The PDEs are defined in terms of functions generated by the lower levels. Such PDE solvers may seem implausible, and so it is important to recall field-computing devices for this purpose were implemented in some practical analog computers (see Sec. B.1) and more recently in Mills' EAC (Mills et al., 2006). As Rubel observed, PDE solvers could be implemented by physical processes that obey the same PDEs (heat equation, wave equation, etc.). (See also Sec. H.1 below.)

Finally, the EAC is required to be "extremely well-posed," which means that each level is relatively insensitive to perturbations in its inputs; thus "all the outputs depend in a strongly deterministic and stable way on the initial settings of the machine" (Rubel, 1993).

Rubel (1993) proves that the EAC can compute everything that the GPAC can compute, but also such functions as the gamma and zeta, and that it can solve the Dirichlet problem for Laplace's equation on the disk, all of which are beyond the GPAC's capabilities. Further, whereas the GPAC can compute differentially algebraic functions of time, the EAC can compute differentially algebraic functions of any finite number of real variables. In fact, Rubel did not find any real-analytic (C^∞) function that is *not* computable on the EAC, but he observes that if the EAC can indeed generate every real-analytic function, it would be too broad to be useful as a model of analog computation.

F Analog computation and the Turing limit

F.1 Introduction

The Church-Turing Thesis asserts that anything that is effectively computable is computable by a Turing machine, but the Turing machine (and equivalent models, such as the lambda calculus) are models of discrete computation, and so it is natural to wonder how analog computing compares in power, and in particular whether it can compute beyond the "Turing limit." Superficial answers are easy to obtain, but the issue is subtle because it depends upon choices among definitions, none of which is obviously correct, it involves the foundations of mathematics and its philosophy, and it raises epistemological issues about the role of models in scientific theories. This is an active research area, but many of the results are apparently inconsistent due to the differing assumptions on which they are based. Therefore this section will be limited to a mention of a few of the interesting results, but without attempting a comprehensive, systematic, or detailed survey; Siegelmann (1999) can serve as an introduction to the literature.

F.2 A sampling of theoretical results

F.2.a Continuous-time models

Orponen's (1997) survey of continuous-time computation theory is a good introduction to the literature as of that time; here we give a sample of these and more recent results.

There are several results showing that—under various assumptions—analog computers have at least the power of Turing machines (TMs). For example, Branicky (1994) showed that a TM could be simulated by ODEs, but he used non-differentiable functions; Bournez et al. (2006) provide an alternative construction using only analytic functions. They also prove that the GPAC computability coincides with (Turing-)computable analysis, which is surprising, since the gamma function is Turing-computable but, as we have seen, the GPAC cannot generate it. The paradox is resolved by a distinction between *generating* a function and *computing* it, with the latter, broader notion permitting convergent computation of the function (that is, as $t \to \infty$). However, the computational power of general ODEs has not been determined in general (Siegelmann, 1999, p. 149). M. B. Pour-El and I. Richards exhibit a Turing-computable ODE that does not have a Turing-computable solution (Pour-El & Richards, 1979, 1982). Stannett (1990) also defined a continuous-time analog computer that could solve the halting problem.

Moore (1996) defines a class of continuous-time recursive functions over the reals, which includes a zero-finding operator μ. Functions can be classified into a hierarchy depending on the number of uses of μ, with the lowest level (no μs) corresponding approximately to Shannon's GPAC. Higher levels can compute non-Turing-computable functions, such as the decision procedure for the halting problem, but he questions whether this result is relevant in the physical world, which is constrained by "noise, quantum effects, finite accuracy, and limited resources." Bournez & Cosnard (1996) have extended these results and shown that many dynamical systems have super-Turing power.

Omohundro (1984) showed that a system of ten coupled nonlinear PDEs could simulate an arbitrary cellular automaton, which implies that PDEs have at least Turing power. Further, D. Wolpert and B. J. MacLennan (Wolpert, 1991; Wolpert & MacLennan, 1993) showed that any TM can be simulated by a field computer with linear dynamics, but the construction uses Dirac delta functions. Pour-El and Richards exhibit a wave equation in three-dimensional space with Turing-computable initial conditions, but for which the unique solution is Turing-uncomputable (Pour-El & Richards, 1981, 1982).

F.2.b Sequential-time models

We will mention a few of the results that have been obtained concerning the power of sequential-time analog computation.

Although the BSS model has been investigated extensively, its power has not been completely determined (Blum et al., 1998, 1988). It is known to depend on whether just rational numbers or arbitrary real numbers are allowed in its programs (Siegelmann, 1999, p. 148).

A *coupled map lattice* (CML) is a cellular automaton with real-valued states; it is a sequential-time analog computer, which can be considered a discrete-space approximation to a simple sequential-time field computer. Orponen & Matamala (1996) showed that a finite CML can simulate a universal Turing machine. However, since a CML can simulate a BSS program or a recurrent neural network (see Sec. F.2.c below), it actually has super-Turing power (Siegelmann, 1999, p. 149).

Recurrent neural networks are some of the most important examples of sequential analog computers, and so the following section is devoted to them.

F.2.c Recurrent neural networks

With the renewed interest in neural networks in the mid-1980s, many investigators wondered if recurrent neural nets have super-Turing power. M. Garzon and S. Franklin showed that a sequential-time net with a countable infinity of neurons could exceed Turing power (Franklin & Garzon, 1990; Garzon & Franklin, 1989, 1990). Indeed, Siegelmann & Sontag (1994b) showed that finite neural nets with real-valued weights have super-Turing power, but Maass & Sontag (1999b) showed that recurrent nets with Gaussian or similar noise had *sub*-Turing power, illustrating again the dependence on these results on assumptions about what is a reasonable mathematical model of analog computing.

For recent results on recurrent neural networks, we will restrict our attention of the work of Siegelmann (1999), who addresses the computational power of these network in terms of the classes of languages they can recognize. Without loss of generality the languages are restricted to sets of binary strings. A string to be tested is fed to the network one bit at a time, along with an input that indicates when the end of the input string has been reached. The network is said to *decide* whether the string is in the language if it correctly indicates whether it is in the set or not, after some finite number of sequential steps since input began.

Siegelmann shows that, if exponential time is allowed for recognition, finite recurrent neural networks with real-valued weights (and saturated-linear activation functions) can compute *all* languages, and thus they are more powerful than Turing machines. Similarly, stochastic networks with rational weights also have super-Turing power, although less power than the deterministic nets with real weights. (Specifically, they compute P/POLY and BPP/log* respectively; see Siegelmann 1999, chs. 4, 9 for details.) She further argues that these neural networks serve as a "standard model" of (sequential) analog computation (comparable to Turing machines in Church-Turing computation), and therefore that the limits and capabilities of these nets apply to sequential analog computation generally.

Siegelmann (1999, p 156) observes that the super-Turing power of recurrent neural networks is a consequence of their use of non-rational real-valued weights. In effect, a real number can contain an infinite number of bits of information. This raises the question of how the non-rational weights of a network can ever be set, since it is not possible to define a physical quantity with infinite precision. However, although non-rational weights may not be able to be set from outside the network, they can be computed within the network by learning algorithms, which are analog computations. Thus, Siegelmann suggests, the fundamental distinction may be between *static computational models*, such as the Turing machine and its equivalents, and *dynamically evolving computational models*, which can tune continuously variable parameters and thereby achieve super-Turing power.

F.2.d Dissipative models

Beyond the issue of the power of analog computing relative to the Turing limit, there are also questions of its relative efficiency. For example, could analog computing solve NP-hard problems in polynomial or even linear time? In traditional computational complexity theory, efficiency issues are addressed in terms of the asymptotic number of computation steps to compute a function as the size of the function's input increases. One way to address corresponding issues in an analog context is by treating an analog computation as a *dissipative system*, which in this context means a system that decreases some quantity (analogous to energy) so that the system state converges to an *point attractor*. From this perspective, the initial state of the system incorporates the input to the computation, and the attractor represents its output. Therefore, H. T. Sieglemann, S. Fishman, and A. Ben-Hur have developed a complexity theory for dissipative systems, in both sequential and continuous time, which addresses the rate of convergence in terms of the underlying rates of the system (Ben-Hur et al., 2002; Siegelmann et al., 1999). The relation between dissipative complexity classes (e.g., P_d, NP_d) and corresponding classical complexity classes (P, NP) remains unclear (Siegelmann, 1999, p. 151).

F.3 Real-valued inputs, output, and constants

A common argument, with relevance to the theoretical power of analog computation, is that an input to an analog computer must be determined by setting a dial to a number or by typing a number into digital-to-analog conversion device, and therefore that the input will be a rational number. The same argument applies to any internal constants in the analog computation. Similarly, it is argued, any output from an analog computer must be measured, and the accuracy of measurement is limited, so that the result will be a rational number. Therefore, it is claimed, real numbers are irrelevant to analog computing, since any practical analog computer computes a function from the rationals to the rationals, and can therefore be simulated by a Turing machine.[2]

There are a number of interrelated issues here, which may be considered briefly. First, the argument is couched in terms of the input or output of *digital representations*, and the numbers so represented are necessarily rational (more generally, computable). This seems natural enough when we think of an analog computer as a calculating device, and in fact many historical analog computers were used in this way and had digital inputs and outputs (since this is our most reliable way of recording and reproducing quantities).

However, in many analog *control systems*, the inputs and outputs are continuous physical quantities that vary continuously in time (also a continuous physical quantity); that is, according to current physical theory, these quantities are real numbers, which vary according to differential equations. It is worth recalling that physical quantities are neither rational nor irrational; they can be so classified only in comparison with each other or with respect to a unit, that is, only if they are measured and digitally represented. Furthermore, physical quantities are neither computable nor uncomputable (in a Church-Turing sense); these terms apply only to discrete representations of these quantities (i.e., to numerals or other digital representations).

Therefore, in accord with ordinary mathematical descriptions of physical processes, analog computations can can be treated as having arbitrary real numbers (in some range) as inputs, outputs, or internal states; like other continuous processes, continuous-time analog computations pass through all the reals in some range, including non-Turing-computable reals. Paradoxically, however, these same physical processes can be simulated on digital computers.

F.4 The issue of simulation by Turing machines and digital computers

Theoretical results about the computational power, relative to Turing machines, of neural networks and other analog models of computation raise difficult issues, some of which are epistemological rather than strictly technical. On the one hand, we have a series of theoretical results proving the super-Turing power of analog computation models of various kinds. On the

[2]See related arguments by Martin Davis (2004, 2006).

other hand, we have the obvious fact that neural nets are routinely simulated on ordinary digital computers, which have at most the power of Turing machines. Furthermore, it is reasonable to suppose that any physical process that might be used to realize analog computation—and certainly the known processes—could be simulated on a digital computer, as is done routinely in computational science. This would seem to be incontrovertible proof that analog computation is no more powerful than Turing machines. The crux of the paradox lies, of course, in the non-Turing-computable reals. These numbers are a familiar, accepted, and necessary part of standard mathematics, in which physical theory is formulated, but from the standpoint of Church-Turing (CT) computation they do not exist. This suggests that the the paradox is not a contradiction, but reflects a divergence between the goals and assumptions of the two models of computation.

F.5 The problem of models of computation

These issues may be put in context by recalling that the Church-Turing (CT) model of computation is in fact a *model*, and therefore that it has the limitations of all models. A model is a cognitive tool that improves our ability to understand some class of phenomena by preserving relevant characteristics of the phenomena while altering other, irrelevant (or less relevant) characteristics. For example, a *scale model* alters the size (taken to be irrelevant) while preserving shape and other characteristics. Often a model achieves its purposes by making *simplifying* or *idealizing assumptions*, which facilitate analysis or simulation of the system. For example, we may use a linear mathematical model of a physical process that is only approximately linear. For a model to be effective it must preserve characteristics and make simplifying assumptions that are appropriate to the domain of questions it is intended to answer, its *frame of relevance* (MacLennan, 2004). If a model is applied to problems outside of its frame of relevance, then it may give answers that are misleading or incorrect, because they depend more on the simplifying assumptions than on the phenomena being modeled. Therefore we must be especially cautious applying a model outside of its frame of relevance, or even at the limits of its frame, where the simplifying assumptions become progressively less appropriate. The problem is aggravated by the fact that often the frame of relevance is not explicitly defined, but resides in a tacit background of practices and skills within some discipline.

Therefore, to determine the applicability of the CT model of computation to analog computing, we must consider the frame of relevance of the CT model. This is easiest if we recall the domain of issues and questions it was originally developed to address: issues of effective calculability and derivability in formalized mathematics. This frame of relevance determines many of the assumptions of the CT model, for example, that information is represented by finite discrete structures of symbols from a finite alphabet, that information processing proceeds by the application of definite formal rules at discrete instants of time, and that a computational or derivational process must be completed in a finite number of these steps.[3] Many of these assumptions are incompatible with analog computing and with the frames of relevance of many models of analog computation.

F.6 Relevant issues for analog computation

Analog computation is often used for control. Historically, analog computers were used in control systems and to simulate control systems, but contemporary analog VLSI is also frequently applied in control. Natural analog computation also frequently serves a control function, for example, sensorimotor control by the nervous system, genetic regulation in cells, and self-organized cooperation in insect colonies. Therefore, control systems delimit one frame of relevance for models of analog computation.

In this frame of relevance real-time response is a critical issue, which models of analog computation, therefore, ought to be able to address. Thus it is necessary to be able to relate the speed and frequency response of analog computation to the rates of the physical processes by which the computation is realized. Traditional methods of algorithm analysis, which are based on sequential time and asymptotic behavior, are inadequate in this frame of relevance. On the one hand, the constants (time scale factors), which reflect the underlying rate of computation are absolutely critical (but ignored in asymptotic analysis); on the other hand, in control applications the asymptotic behavior of algorithm is generally irrelevant, since the inputs are typically fixed in size or of a limited range of sizes.

The CT model of computation is oriented around the idea that the purpose of a computation is to evaluate a mathematical function. Therefore the basic criterion of adequacy for a computation is *correctness*, that is, that

[3]See MacLennan (2003, 2004) for a more detailed discussion of the frame of relevance of the CT model.

given a precise representation of an input to the function, it will produce (after finitely many steps) a precise representation of the corresponding output of the function. In the context of natural computation and control, however, other criteria may be equally or even more relevant. For example, *robustness* is important: how well does the system respond in the presence of noise, uncertainty, imprecision, and error, which are unavoidable in physical natural and artificial control systems, and how well does it respond to defects and damage, which arise in many natural and artificial contexts. Since the real world is unpredictable, *flexibility* is also important: how well does an artificial system respond to inputs for which it was not designed, and how well does a natural system behave in situations outside the range of those to which it is evolutionarily adapted. Therefore, *adaptability* (through learning and other means) is another important issue in this frame of relevance.[4]

F.7 Transcending Turing computability

Thus we see that many applications of analog computation raise different questions from those addressed by the CT model of computation; the most useful models of analog computing will have a different frame of relevance. In order to address traditional questions such as whether analog computers can compute "beyond the Turing limit," or whether they can solve NP-hard problems in polynomial time, it is necessary to construct models of analog computation within the CT frame of relevance. Unfortunately, constructing such models requires making commitments about many issues (such as the representation of reals and the discretization of time), that may affect the answers to these questions, but are fundamentally unimportant in the frame of relevance of the most useful applications of the concept of analog computation. Therefore, being overly focused on traditional problems in the theory of computation (which was formulated for a different frame of relevance) may distract us from formulating models of analog computation that can address important issues in its own frame of relevance.

[4]See MacLennan (2003, 2004) for a more detailed discussion of the frames of relevance of natural computation and control.

G Analog thinking

It will be worthwhile to say a few words about the *cognitive implications* of analog computing, which are a largely forgotten aspect of analog vs. digital debates of the late 20th century. For example, it was argued that analog computing provides a deeper intuitive understanding of a system than the alternatives do (Bissell 2004, Small 2001, ch. 8). On the one hand, analog computers afforded a means of understanding analytically intractable systems by means of "dynamic models." By setting up an analog simulation, it was possible to vary the parameters and explore interactively the behavior of a dynamical system that could not be analyzed mathematically. Digital simulations, in contrast, were orders of magnitude slower and did not permit this kind of interactive investigation. (Performance has improved sufficiently in contemporary digital computers so that in many cases digital simulations can be used as dynamic models, sometimes with an interface that mimics an analog computer; see Bissell 2004.)

Analog computing is also relevant to the cognitive distinction between *knowing how* (*procedural knowledge*) and *knowing that* (*declarative knowledge*) (Small, 2001, ch. 8). The latter ("know-that") is more characteristic of scientific culture, which strives for generality and exactness, often by designing experiments that allow phenomena to be studied in isolation, whereas the former ("know-how") is more characteristic of engineering culture; at least it was so through the first half of the twentieth century, before the development of "engineering science" and the widespread use of analytic techniques in engineering education and practice. Engineers were faced with analytically intractable systems, with inexact measurements, and with empirical relationships (characteristic curves, etc.), all of which made analog computers attractive for solving engineering problems. Furthermore, because analog computing made use of physical phenomena that were mathematically analogous to those in the primary system, the engineer's intuition and understanding of one system could be transferred to the other. Some commentators have mourned the loss of hands-on intuitive understanding resulting from the increasingly scientific orientation of engineering education and the disappearance of analog computers (Bissell, 2004; Lang, 2000; Owens, 1986; Puchta, 1996).

I will mention one last cognitive issue relevant to the differences between analog and digital computing. As already discussed Sec. C.4, it is generally agreed that it is less expensive to achieve high precision with digital technology than with analog technology. Of course, high precision may not be important, for example when the available data are inexact or in natural computation. Further, some advocates of analog computing argue that high precision digital results are often misleading (Small, 2001, p. 261). Precision does not imply accuracy, and the fact that an answer is displayed with 10 digits does not guarantee that it is accurate to 10 digits; in particular, engineering data may be known to only a few significant figures, and the accuracy of digital calculation may be limited by numerical problems. Therefore, on the one hand, users of digital computers might fall into the trap of trusting their apparently exact results, but users of modest-precision analog computers were more inclined to healthy skepticism about their computations. Or so it was claimed.

H Future directions

H.1 Post-Moore's Law computing

Certainly there are many purposes that are best served by digital technology; indeed there is a tendency nowadays to think that everything is done better digitally. Therefore it will be worthwhile to consider whether analog computation should have a role in future computing technologies. I will argue that the approaching end of *Moore's Law* (Moore, 1965), which has predicted exponential growth in digital logic densities, will encourage the development of new analog computing technologies.

Two avenues present themselves as ways toward greater computing power: faster individual computing elements and greater densities of computing elements. Greater density increases power by facilitating parallel computing, and by enabling greater computing power to be put into smaller packages. Other things being equal, the fewer the layers of implementation between the computational operations and the physical processes that realize them, that is to say, the more directly the physical processes implement the computations, the more quickly they will be able to proceed. Since most physical processes are continuous (defined by differential equations), analog computation is generally faster than digital. For example, we may compare analog addition, implemented directly by the additive combination of physical quantities, with the sequential process of digital addition. Similarly, other things being equal, the fewer physical devices required to implement a computational element, the greater will be the density of these elements. Therefore, in general, the closer the computational process is to the physical processes that realize it, the fewer devices will be required, and so the continuity of physical law suggests that analog computation has the potential for greater density than digital. For example, four transistors can realize analog addition, whereas many more are required for digital addition. Both considerations argue for an increasing role of analog computation in post-Moore's Law computing.

From this broad perspective, there are many physical phenomena that are potentially usable for future analog computing technologies. We seek phenomena that can be described by well-known and useful mathematical functions (e.g., addition, multiplication, exponential, logarithm, convolution). These descriptions do not need to be exact for the phenomena to be useful in many applications, for which limited range and precision are adequate. Furthermore, in some applications speed is not an important criterion; for example, in some control applications, small size, low power, robustness, etc. may be more important than speed, so long as the computer responds quickly enough to accomplish the control task. Of course there are many other considerations in determining whether given physical phenomena can be used for practical analog computation in a given application (MacLennan, 2009). These include stability, controllability, manufacturability, and the ease of interfacing with input and output transducers and other devices. Nevertheless, in the post-Moore's Law world, we will have to be willing to consider all physical phenomena as potential computing technologies, and in many cases we will find that analog computing is the most effective way to utilize them.

Natural computation provides many examples of effective analog computation realized by relatively slow, low-precision operations, often through massive parallelism. Therefore, post-Moore's Law computing has much to learn from the natural world.