Chapter V

Analog Computation

 [1]

A Definition

Although analog computation was eclipsed by digital computation in the second half of the twentieth century, it is returning as an important alternative computing technology. Indeed, as explained in this chapter, theoretical results imply that analog computation can escape from the limitations of digital computation. Furthermore, analog computation has emerged as an important theoretical framework for discussing computation in the brain and other natural systems.

Analog computation gets its name from an *analogy*, or systematic relationship, between the physical processes in the computer and those in the system it is intended to model or simulate (the *primary system*). For example, the electrical quantities voltage, current, and conductance might be used as analogs of the fluid pressure, flow rate, and pipe diameter of a hydrolic system. More specifically, in traditional analog computation, physical quantities in the computation obey the same mathematical laws as physical quantities in the primary system. Thus the computational quantities are proportional

[1] This chapter is based on an unedited draft for an article that appeared in the *Encyclopedia of Complexity and System Science* (Springer, 2008).

to the modeled quantities. This is in contrast to *digital computation*, in which quantities are represented by strings of symbols (e.g., binary digits) that have no direct physical relationship to the modeled quantities. According to the *Oxford English Dictionary* (2nd ed., s.vv. analogue, digital), these usages emerged in the 1940s.

However, in a fundamental sense all computing is based on an analogy, that is, on a systematic relationship between the states and processes in the computer and those in the primary system. In a digital computer, the relationship is more abstract and complex than simple proportionality, but even so simple an analog computer as a slide rule goes beyond strict proportion (i.e., distance on the rule is proportional to the logarithm of the number). In both analog and digital computation—indeed in all computation—the relevant abstract mathematical structure of the problem is realized in the physical states and processes of the computer, but the realization may be more or less direct (MacLennan, 1994a,c, 2004).

Therefore, despite the etymologies of the terms "analog" and "digital," in modern usage the principal distinction between digital and analog computation is that the former operates on discrete representations in discrete steps, while the later operated on continuous representations by means of continuous processes (e.g., MacLennan 2004, Siegelmann 1999, p. 147, Small 2001, p. 30, Weyrick 1969, p. 3). That is, the primary distinction resides in the *topologies* of the states and processes, and it would be more accurate to refer to *discrete* and *continuous computation* (Goldstine, 1972, p. 39). (Consider so-called analog and digital clocks. The principal difference resides in the continuity or discreteness of the representation of time; the motion of the two (or three) hands of an "analog" clock do not mimic the motion of the rotating earth or the position of the sun relative to it.)

B Introduction

B.1 History

B.1.a Pre-electronic analog computation

Just like digital calculation, analog computation was originally performed by hand. Thus we find several analog computational procedures in the "constructions" of Euclidean geometry (Euclid, fl. 300 BCE), which derive from techniques used in ancient surveying and architecture. For example, Problem

II.51 is "to divide a given straight line into two parts, so that the rectangle contained by the whole and one of the parts shall be equal to the square of the other part." Also, Problem VI.13 is "to find a mean proportional between two given straight lines," and VI.30 is "to cut a given straight line in extreme and mean ratio." These procedures do not make use of measurements in terms of any fixed unit or of digital calculation; the lengths and other continuous quantities are manipulated directly (via compass and straightedge). On the other hand, the techniques involve discrete, precise operational steps, and so they can be considered *algorithms*, but over continuous magnitudes rather than discrete numbers.

It is interesting to note that the ancient Greeks distinguished continuous *magnitudes* (Grk., *megethoi*), which have physical dimensions (e.g., length, area, rate), from discrete *numbers* (Grk., *arithmoi*), which do not (Maziarz & Greenwood, 1968). Euclid axiomatizes them separately (magnitudes in Book V, numbers in Book VII), and a mathematical system comprising both discrete and continuous quantities was not achieved until the nineteenth century in the work of Weierstrass and Dedekind.

The earliest known mechanical analog computer is the "Antikythera mechanism," which was found in 1900 in a shipwreck under the sea near the Greek island of Antikythera (between Kythera and Crete). It dates to the second century BCE and appears to be intended for astronomical calculations. The device is sophisticated (at least 70 gears) and well engineered, suggesting that it was not the first of its type, and therefore that other analog computing devices may have been used in the ancient Mediterranean world (Freeth et al., 2006). Indeed, according to Cicero (*Rep.* 22) and other authors, Archimedes (c. 287–c. 212 BCE) and other ancient scientists also built analog computers, such as armillary spheres, for astronomical simulation and computation. Other antique mechanical analog computers include the astrolabe, which is used for the determination of longitude and a variety of other astronomical purposes, and the *torquetum*, which converts astronomical measurements between equatorial, ecliptic, and horizontal coordinates.

A class of special-purpose analog computer, which is simple in conception but may be used for a wide range of purposes, is the *nomograph* (also, *nomogram*, *alignment chart*). In its most common form, it permits the solution of quite arbitrary equations in three real variables, $f(u, v, w) = 0$. The nomograph is a chart or graph with scales for each of the variables; typically these scales are curved and have non-uniform numerical markings. Given values for any two of the variables, a straightedge is laid across their posi-

tions on their scales, and the value of the third variable is read off where the straightedge crosses the third scale. Nomographs were used to solve many problems in engineering and applied mathematics. They improve intuitive understanding by allowing the relationships among the variables to be visualized, and facilitate exploring their variation by moving the straightedge. Lipka (1918) is an example of a course in graphical and mechanical methods of analog computation, including nomographs and slide rules.

Until the introduction of portable electronic calculators in the early 1970s, the *slide rule* was the most familiar analog computing device. Slide rules use logarithms for multiplication and division, and they were invented in the early seventeenth century shortly after John Napier's description of logarithms.

The mid-nineteenth century saw the development of the *field analogy method* by G. Kirchhoff (1824–87) and others (Kirchhoff, 1845). In this approach an electrical field in an electrolytic tank or conductive paper was used to solve two-dimensional boundary problems for temperature distributions and magnetic fields (Small, 2001, p. 34). It is an early example of *analog field computation*, which operates on continuous spatial distributions of quantity (i.e., fields).

In the nineteenth century a number of mechanical analog computers were developed for integration and differentiation (e.g., Lipka 1918, pp. 246–56, Clymer 1993). For example, the *planimeter* measures the area under a curve or within a closed boundary. While the operator moves a pointer along the curve, a rotating wheel accumulates the area. Similarly, the *integraph* is able to draw the integral of a given function as its shape is traced. Other mechanical devices can draw the derivative of a curve or compute a tangent line at a given point.

In the late nineteenth century William Thomson, Lord Kelvin, constructed several analog computers, including a "tide predictor" and a "harmonic analyzer," which computed the Fourier coefficients of a tidal curve (Thomson, 1878, 1938). In 1876 he described how the mechanical integrators invented by his brother could be connected together in a feedback loop in order to solve second and higher order differential equations (Small 2001, pp. 34–5, 42, Thomson 1876). He was unable to construct this *differential analyzer*, which had to await the invention of the torque amplifier in 1927.

The torque amplifier and other technical advancements permitted Vannevar Bush at MIT to construct the first practical differential analyzer in 1930 (Small, 2001, pp. 42–5). It had six integrators and could also do addition, subtraction, multiplication, and division. Input data were entered in

the form of continuous curves, and the machine automatically plotted the output curves continuously as the equations were integrated. Similar differential analyzers were constructed at other laboratories in the US and the UK.

Setting up a problem on the MIT differential analyzer took a long time; gears and rods had to be arranged to define the required dependencies among the variables. Bush later designed a much more sophisticated machine, the Rockefeller Differential Analyzer, which became operational in 1947. With 18 integrators (out of a planned 30), it provided programmatic control of machine setup, and permitted several jobs to be run simultaneously. Mechanical differential analyzers were rapidly supplanted by electronic analog computers in the mid-1950s, and most were disassembled in the 1960s (Bowles 1996, Owens 1986, Small 2001, pp. 50–5).

During World War II, and even later wars, an important application of optical and mechanical analog computation was in "gun directors" and "bomb sights," which performed ballistic computations to accurately target artillery and dropped ordnance.

B.1.b Electronic analog computation in the 20th century

It is commonly supposed that electronic analog computers were superior to mechanical analog computers, and they were in many respects, including speed, cost, ease of construction, size, and portability (Small, 2001, pp. 54–6). On the other hand, mechanical integrators produced higher precision results (0.1%, vs. 1% for early electronic devices) and had greater mathematical flexibility (they were able to integrate with respect to any variable, not just time). However, many important applications did not require high precision and focused on dynamic systems for which time integration was sufficient; for these, electronic analog computers were superior.

Analog computers (non-electronic as well as electronic) can be divided into *active-element* and *passive-element* computers; the former involve some kind of amplification, the latter do not (Truitt & Rogers, 1960, pp. 2-1–4). Passive-element computers included the *network analyzers* that were developed in the 1920s to analyze electric power distribution networks, and which continued in use through the 1950s (Small, 2001, pp. 35–40). They were also applied to problems in thermodynamics, aircraft design, and mechanical engineering. In these systems networks or grids of resistive elements or reactive elements (i.e., involving capacitance and inductance as well as

resistance) were used to model the spatial distribution of physical quantities such as voltage, current, and power (in electric distribution networks), electrical potential in space, stress in solid materials, temperature (in heat diffusion problems), pressure, fluid flow rate, and wave amplitude (Truitt & Rogers, 1960, p. 2-2). That is, network analyzers dealt with partial differential equations (PDEs), whereas active-element computers, such as the differential analyzer and its electronic successors, were restricted to ordinary differential equations (ODEs) in which time was the independent variable. Large network analyzers are early examples of *analog field computers*.

Electronic analog computers became feasible after the invention of the *DC operational amplifier* ("op amp") c. 1940 (Small, 2001, pp. 64, 67–72). Already in the 1930s scientists at Bell Telephone Laboratories (BTL) had developed the DC-coupled feedback-stabilized amplifier, which is the basis of the op amp. In 1940, as the USA prepared to enter World War II, D. L. Parkinson at BTL had a dream in which he saw DC amplifiers being used to control an anti-aircraft gun. As a consequence, with his colleagues C. A. Lovell and B. T. Weber, he wrote a series of papers on "electrical mathematics," which described electrical circuits to "operationalize" addition, subtraction, integration, differentiation, etc. The project to produce an electronic gun-director led to the development and refinement of DC op amps suitable for analog computation.

The war-time work at BTL was focused primarily on control applications of analog devices, such as the gun-director. Other researchers, such as E. Lakatos at BTL, were more interested in applying them to general-purpose analog computation for science and engineering, which resulted in the design of the General Purpose Analog Computer (GPAC), also called "Gypsy," completed in 1949 (Small, 2001, pp. 69–71). Building on the BTL op amp design, fundamental work on electronic analog computation was conducted at Columbia University in the 1940s. In particular, this research showed how analog computation could be applied to the simulation of dynamic systems and to the solution of nonlinear equations.

Commercial general-purpose analog computers (GPACs) emerged in the late 1940s and early 1950s (Small, 2001, pp. 72–3). Typically they provided several dozen integrators, but several GPACs could be connected together to solve larger problems. Later, large-scale GPACs might have up to 500 amplifiers and compute with 0.01%–0.1% precision (Truitt & Rogers, 1960, pp. 2–33).

Besides integrators, typical GPACs provided adders, subtracters, multi-

pliers, fixed function generators (e.g., logarithms, exponentials, trigonometric functions), and variable function generators (for user-defined functions) (Truitt & Rogers, 1960, chs. 1.3, 2.4). A GPAC was programmed by connecting these components together, often by means of a patch panel. In addition, parameters could be set by adjusting potentiometers (attenuators), and arbitrary functions could be entered in the form of graphs (Truitt & Rogers, 1960, pp. 1-72–81, 2-154–156). Output devices plotted data continuously or displayed it numerically (Truitt & Rogers, 1960, pp. 3-1–30).

The most basic way of using a GPAC was in *single-shot mode* (Weyrick, 1969, pp. 168–70). First, parameters and initial values were entered into the potentiometers. Next, putting a master switch in "reset" mode controlled relays to apply the initial values to the integrators. Turning the switch to "operate" or "compute" mode allowed the computation to take place (i.e., the integrators to integrate). Finally, placing the switch in "hold" mode stopped the computation and stabilized the values, allowing them to be read from the computer (e.g., on voltmeters). Although single-shot operation was also called "slow operation" (in comparison to "repetitive operation," discussed next), it was in practice quite fast. Because all of the devices computed in parallel and at electronic speeds, analog computers usually solved problems in real-time but often much faster (Truitt & Rogers 1960, pp. 1-30–32, Small 2001, p. 72).

One common application of GPACs was to explore the effect of one or more parameters on the behavior of a system. To facilitate this exploration of the parameter space, some GPACs provided a *repetitive operation mode*, which worked as follows (Weyrick 1969, p. 170, Small 2001, p. 72). An electronic clock switched the computer between reset and compute modes at an adjustable rate (e.g., 10–1000 cycles per second) (Ashley, 1963, p. 280, n. 1). In effect the simulation was rerun at the clock rate, but if any parameters were adjusted, the simulation results would vary along with them. Therefore, within a few seconds, an entire family of related simulations could be run. More importantly, the operator could acquire an intuitive understanding of the system's dependence on its parameters.

B.1.c The eclipse of analog computing

A common view is that electronic analog computers were a primitive predecessor of the digital computer, and that their use was just a historical episode, or even a digression, in the inevitable triumph of digital technol-

ogy. It is supposed that the current digital hegemony is a simple matter of technological superiority. However, the history is much more complicated, and involves a number of social, economic, historical, pedagogical, and also technical factors, which are outside the scope of this book (see Small 1993 and Small 2001, especially ch. 8, for more information). In any case, beginning after World War II and continuing for twenty-five years, there was lively debate about the relative merits of analog and digital computation.

Speed was an oft-cited advantage of analog computers (Small, 2001, ch. 8). While early digital computers were much faster than mechanical differential analyzers, they were slower (often by several orders of magnitude) than electronic analog computers. Furthermore, although digital computers could perform individual arithmetic operations rapidly, complete problems were solved sequentially, one operation at a time, whereas analog computers operated in parallel. Thus it was argued that increasingly large problems required more time to solve on a digital computer, whereas on an analog computer they might require more hardware but not more time. Even as digital computing speed was improved, analog computing retained its advantage for several decades, but this advantage eroded steadily.

Another important issue was the comparative *precision* of digital and analog computation (Small, 2001, ch. 8). Analog computers typically computed with three or four digits of precision, and it was very expensive to do much better, due to the difficulty of manufacturing the parts and other factors. In contrast, digital computers could perform arithmetic operations with many digits of precision, and the hardware cost was approximately proportional to the number of digits. Against this, analog computing advocates argued that many problems did not require such high precision, because the measurements were known to only a few significant figures and the mathematical models were approximations. Further, they distinguished between precision and *accuracy*, which refers to the conformity of the computation to physical reality, and they argued that digital computation was often less accurate than analog, due to numerical limitations (e.g., truncation, cumulative error in numerical integration). Nevertheless, some important applications, such as the calculation of missile trajectories, required greater precision, and for these, digital computation had the advantage. Indeed, to some extent precision was viewed as inherently desirable, even in applications where it was unimportant, and it was easily mistaken for accuracy. (See Sec. C.4.a for more on precision and accuracy.)

There was even a social factor involved, in that the written programs,

precision, and exactness of digital computation were associated with mathematics and science, but the hands-on operation, parameter variation, and approximate solutions of analog computation were associated with engineering, and so analog computing inherited "the lower status of engineering *vis-à-vis* science" (Small, 2001, p. 251). Thus the status of digital computing was further enhanced as engineering became more mathematical and scientific after World War II (Small, 2001, pp. 247–51).

Already by the mid-1950s the competition between analog and digital had evolved into the idea that they were complementary technologies. This resulted in the development of a variety of *hybrid* analog/digital computing systems (Small, 2001, pp. 251–3, 263–6). In some cases this involved using a digital computer to control an analog computer by using digital logic to connect the analog computing elements, to set parameters, and to gather data. This improved the accessibility and usability of analog computers, but had the disadvantage of distancing the user from the physical analog system. The intercontinental ballistic missile program in the USA stimulated the further development of hybrid computers in the late 1950s and 1960s (Small, 1993). These applications required the speed of analog computation to simulate the closed-loop control systems and the precision of digital computation for accurate computation of trajectories. However, by the early 1970s hybrids were being displaced by all digital systems. Certainly part of the reason was the steady improvement in digital technology, driven by a vibrant digital computer industry, but contemporaries also pointed to an inaccurate perception that analog computing was obsolete and to a lack of education about the advantages and techniques of analog computing.

Another argument made in favor of digital computers was that they were general-purpose, since they could be used in business data processing and other application domains, whereas analog computers were essentially special-purpose, since they were limited to scientific computation (Small, 2001, pp. 248–50). Against this it was argued that *all* computing is essentially computing by analogy, and therefore analog computation was general-purpose because the class of analog computers included digital computers! (See also Sec. A on computing by analogy.) Be that as it may, analog computation, as normally understood, is restricted to continuous variables, and so it was not immediately applicable to discrete data, such as that manipulated in business computing and other nonscientific applications. Therefore business (and eventually consumer) applications motivated the computer industry's investment in digital computer technology at the expense of analog

technology.

Although it is commonly believed that analog computers quickly disappeared after digital computers became available, this is inaccurate, for both general-purpose and special-purpose analog computers have continued to be used in specialized applications to the present time. For example, a general-purpose *electrical* (vs. electronic) analog computer, the Anacom, was still in use in 1991. This is not technological atavism, for "there is no doubt considerable truth in the fact that Anacom continued to be used because it effectively met a need in a historically neglected but nevertheless important computer application area" (Aspray, 1993). As mentioned, the reasons for the eclipse of analog computing were not simply the technological superiority of digital computation; the conditions were much more complex. Therefore a change in conditions has necessitated a reevaluation of analog technology.

B.1.d Analog VLSI

In the mid-1980s, Carver Mead, who already had made important contributions to digital VLSI technology, began to advocate for the development of analog VLSI (Mead, 1987, 1989). His motivation was that "the nervous system of even a very simple animal contains computing paradigms that are orders of magnitude more effective than are those found in systems made by humans" and that they "can be realized in our most commonly available technology—silicon integrated circuits" (Mead, 1989, p. xi). However, he argued, since these natural computation systems are analog and highly nonlinear, progress would require understanding neural information processing in animals and applying it in a new analog VLSI technology.

Because analog computation is closer to the physical laws by which all computation is realized (which are continuous), analog circuits often use fewer devices than corresponding digital circuits. For example, a four-quadrant adder (capable of adding two signed numbers) can be fabricated from four transistors (Mead, 1989, pp. 87–8), and a four-quadrant multiplier from nine to seventeen, depending on the required range of operation (Mead, 1989, pp. 90–6). Intuitions derived from digital logic about what is simple or complex to compute are often misleading when applied to analog computation. For example, two transistors are sufficient to compute the logarithm or exponential, five for the hyperbolic tangent (which is very useful in neural computation), and three for the square root (Mead, 1989, pp. 70–1, 97–9). Thus analog VLSI is an attractive approach to "post-Moore's Law computing" (see Sec.

H, p. 275 below). Mead and his colleagues demonstrated a number of analog VLSI devices inspired by the nervous system, including a "silicon retina" and an "electronic cochlea" (Mead, 1989, chs. 15–16), research that has lead to a renaissance of interest in electronic analog computing.

B.1.e Field-programmable analog arrays

Field Programmable Analog Arrays (FPAAs) permit the programming of analog VLSI systems comparable to Field Programmable Gate Arrays (FPGAs) for digital systems. An FPAA comprises a number of identical Computational Analog Blocks (CABs), each of which contains a small number of analog computing elements. Programmable switching matrices control the interconnections among the elements of a CAB and the interconnections between the CABs. Contemporary FPAAs make use of floating-gate transistors, in which the gate has no DC connection to other circuit elements and thus is able to hold a charge indefinitely **cite**. Therefore the floating gate can be used to store a continuous value that governs the impedance of the transistor by several orders of magnitude. The gate charge can be changed by processes such as electron tunneling, which increases the charge, and hot-electron injection, which decreases it. Digital decoders allow individual floating-gate transistors in the switching matrices to be addressed and programmed. At the extremes of zero and infinite impedance the transistors operate as perfect switches, connecting or disconnecting circuit elements. Programming the connections to these extreme values is time consuming, however, and so in practice some tradeoff is made between programming time and switch impedance. Each CAB contains several Operational Transconductance Amplifiers (OTAs), which are op-amps whose gain is controlled by a bias current. They are the principal analog computing elements, since they can be used for operations such as integration, differentiation, and gain amplification. Other computing elements may include tunable band-pass filters, which can be used for Fourier signal processing, and small matrix-vector multipliers, which can be used to implement linear operators. Current FPAAs can compute with a resolution of 10 bits (precision of 10^{-3}).

B.1.f Non-electronic analog computation

As will be explained later in this chapter, analog computation suggests many opportunities for future computing technologies. Many physical phenomena

are potential media for analog computation provided they have useful mathematical structure (i.e., the mathematical laws describing them are mathematical functions useful for general- or special-purpose computation), and they are sufficiently controllable for practical use.

B.2 Chapter roadmap

The remainder of this chapter will begin by summarizing the fundamentals of analog computing, starting with the continuous state space and the various processes by which analog computation can be organized in time. Next it will discuss analog computation in nature, which provides models and inspiration for many contemporary uses of analog computation, such as neural networks. Then we consider general-purpose analog computing, both from a theoretical perspective and in terms of practical general-purpose analog computers. This leads to a discussion of the theoretical power of analog computation and in particular to the issue of whether analog computing is in some sense more powerful than digital computing. We briefly consider the cognitive aspects of analog computing, and whether it leads to a different approach to computation than does digital computing. Finally, we conclude with some observations on the role of analog computation in "post-Moore's Law computing."

C Fundamentals of analog computing

C.1 Continuous state space

As discussed in Sec. B, the fundamental characteristic that distinguishes analog from digital computation is that the state space is continuous in analog computation and discrete in digital computation. Therefore it might be more accurate to call analog and digital computation *continuous* and *discrete computation*, respectively. Furthermore, since the earliest days there have been *hybrid computers* that combine continuous and discrete state spaces and processes. Thus, there are several respects in which the state space may be continuous.

In the simplest case the state space comprises a finite (generally modest) number of variables, each holding a continuous quantity (e.g., voltage, current, charge). In a traditional GPAC they correspond to the variables in

the ODEs defining the computational process, each typically having some independent meaning in the analysis of the problem. Mathematically, the variables are taken to contain bounded real numbers, although complex-valued variables are also possible (e.g., in AC electronic analog computers). In a practical sense, however, their precision is limited by noise, stability, device tolerance, and other factors (discussed below, Sec. C.4).

In typical analog neural networks the state space is larger in dimension but more structured than in traditional analog computers. The artificial neurons are organized into one or more layers, each composed of a (possibly large) number of artificial neurons. Commonly each layer of neurons is densely connected to the next layer (i.e., each neuron in one layer is connected to every neuron in the next). In general the layers each have some meaning in the problem domain, but the individual neurons constituting them do not (and so, in mathematical descriptions, the neurons are typically numbered rather than named).

The individual artificial neurons usually perform a simple computation such as this:

$$y = \sigma(s), \text{ where } s = b + \sum_{i=1}^{n} w_i x_i,$$

and where y is the activity of the neuron, $x_1, \ldots, x_n$ are the activities of the neurons that provide its inputs, b is a bias term, and $w_1, \ldots, w_n$ are the *weights* or strengths of the connections. Often the *activation function* σ is a real-valued *sigmoid* ("S-shaped") function, such as the *logistic sigmoid*,

$$\sigma(s) = \frac{1}{1 + e^{-s}},$$

in which case the neuron activity y is a real number, but some applications use a discontinuous *threshold function*, such as the *Heaviside function*,

$$U(s) = \begin{cases} +1 & , \quad \text{if } s \geq 0 \\ 0 & , \quad \text{if } s < 0 \end{cases},$$

in which case the activity is a discrete quantity. The *saturated-linear* or *piecewise-linear* sigmoid is also used occasionally:

$$\sigma(s) = \begin{cases} +1 & , \quad \text{if } s > 1 \\ s & , \quad \text{if } 0 \leq s \leq 1 \\ 0 & , \quad \text{if } s < 0 \end{cases}.$$

Regardless of whether the activation function is continuous or discrete, the bias b and connection weights $w_1, \ldots, w_n$ are real numbers, as is the "net input" $s = b + \sum_i w_i x_i$ to the activation function. Analog computation may be used to evaluate the linear combination s and the activation function $\sigma(s)$, if it is real-valued. If it is discrete, analog computation can approximate it with a sufficiently sharp sigmoid. The biases and weights are normally determined by a learning algorithm (e.g., back-propagation), which is also a good candidate for analog implementation.

In summary, the continuous state space of a neural network includes the bias values and net inputs of the neurons and the interconnection strengths between the neurons. It also includes the activity values of the neurons, if the activation function is a real-valued sigmoid function, as is often the case. Often large groups ("layers") of neurons (and the connections between these groups) have some intuitive meaning in the problem domain, but typically the individual neuron activities, bias values, and interconnection weights do not (they are "sub-symbolic").

If we extrapolate the number of neurons in a layer to the continuum limit, we get a *field*, which may be defined as a spatially continuous distribution of continuous quantity. Treating a group of artificial or biological neurons as a continuous mass is a reasonable mathematical approximation if their number is sufficiently large and if their spatial arrangement is significant (as it generally is in the brain). Fields are especially useful in modeling *cortical maps*, in which information is represented by the pattern of activity over a region of neural cortex.

In field computation the state space in continuous in two ways: it is continuous in variation but also in space. Therefore, field computation is especially applicable to solving PDEs and to processing spatially extended information such as visual images. Some early analog computing devices were capable of field computation (Truitt & Rogers, 1960, pp. 1-14–17, 2-2–16). For example, as previously mentioned (Sec. B), large resistor and capacitor networks could be used for solving PDEs such as diffusion problems. In these cases a discrete ensemble of resistors and capacitors was used to approximate a continuous field, while in other cases the computing medium was spatially continuous. The latter made use of conductive sheets (for two-dimensional fields) or electrolytic tanks (for two- or three-dimensional fields). When they were applied to steady-state spatial problems, these analog computers were called *field plotters* or *potential analyzers*.

The ability to fabricate very large arrays of analog computing devices,

combined with the need to exploit massive parallelism in realtime computation and control applications, creates new opportunities for field computation (MacLennan, 1987, 1990, 1999). There is also renewed interest in using physical fields in analog computation. For example, Rubel (1993) defined an abstract *extended analog computer* (EAC), which augments Shannon's (1941) general purpose analog computer with (unspecified) facilities for field computation, such as PDE solvers (see Secs. E.3–E.4 below). J. W. Mills has explored the practical application of these ideas in his *artificial neural field networks* and VLSI EACs, which use the diffusion of electrons in bulk silicon or conductive gels and plastics for 2D and 3D field computation (Mills, 1996; Mills et al., 2006).

C.2 Computational process

We have considered the continuous state space, which is the basis for analog computing, but there are a variety of ways in which analog computers can operate on the state. In particular, the state can change continuously in time or be updated at distinct instants (as in digital computation).

C.2.a Continuous time

Since the laws of physics on which analog computing is based are differential equations, many analog computations proceed in continuous real time. Also, as we have seen, an important application of analog computers in the late 19th and early 20th centuries was the integration of ODEs in which time is the independent variable. A common technique in analog simulation of physical systems is *time scaling*, in which the differential equations are altered systematically so the simulation proceeds either more slowly or more quickly than the primary system (see Sec. C.4 for more on time scaling). On the other hand, because analog computations are close to the physical processes that realize them, analog computing is rapid, which makes it very suitable for real-time control applications.

In principle, any mathematically describable physical process operating on time-varying physical quantities can be used for analog computation. In practice, however, analog computers typically provide familiar operations that scientists and engineers use in differential equations (Rogers & Connolly, 1960; Truitt & Rogers, 1960). These include basic arithmetic operations, such as algebraic sum and difference ($u(t) = v(t) \pm w(t)$), constant

multiplication or scaling ($u(t) = cv(t)$), variable multiplication and division ($u(t) = v(t)w(t)$, $u(t) = v(t)/w(t)$), and inversion ($u(t) = -v(t)$). Transcendental functions may be provided, such as the exponential ($u(t) = \exp v(t)$), logarithm ($u(t) = \ln v(t)$), trigonometric functions ($u(t) = \sin v(t)$, etc.), and *resolvers* for converting between polar and rectangular coordinates. Most important, of course, is definite integration ($u(t) = v_0 + \int_0^t v(\tau)\mathrm{d}\tau$), but differentiation may also be provided ($u(t) = \dot{v}(t)$). Generally, however, direct differentiation is avoided, since noise tends to have a higher frequency than the signal, and therefore differentiation amplifies noise; typically problems are reformulated to avoid direct differentiation (Weyrick, 1969, pp. 26–7). As previously mentioned, many GPACs include *(arbitrary) function generators*, which allow the use of functions defined only by a graph and for which no mathematical definition might be available; in this way empirically defined functions can be used (Rogers & Connolly, 1960, pp. 32–42). Thus, given a graph $(x, f(x))$, or a sufficient set of samples, $(x_k, f(x_k))$, the function generator approximates $u(t) = f(v(t))$. Rather less common are generators for arbitrary functions of two variables, $u(t) = f(v(t), w(t))$, in which the function may be defined by a surface, $(x, y, f(x, y))$, or by sufficient samples from it.

Although analog computing is primarily continuous, there are situations in which discontinuous behavior is required. Therefore some analog computers provide *comparators*, which produce a discontinuous result depending on the relative value of two input values. For example,

$$u = \begin{cases} k & , \quad \text{if } v \geq w, \\ 0 & , \quad \text{if } v < w. \end{cases}$$

Typically, this would be implemented as a Heaviside (unit step) function applied to the difference of the inputs, $u = kU(v - w)$. In addition to allowing the definition of discontinuous functions, comparators provide a primitive decision making ability, and may be used, for example to terminate a computation (switching the computer from "operate" to "hold" mode).

Other operations that have proved useful in analog computation are time delays and noise generators (Howe, 1961, ch. 7). The function of a *time delay* is simply to retard the signal by an adjustable delay $T > 0$: $u(t + T) = v(t)$. One common application is to model delays in the primary system (e.g., human response time).

Typically a *noise generator* produces time-invariant Gaussian-distributed noise with zero mean and a flat power spectrum (over a band compatible with

the analog computing process). The standard deviation can be adjusted by scaling, the mean can be shifted by addition, and the spectrum altered by filtering, as required by the application. Historically noise generators were used to model noise and other random effects in the primary system, to determine, for example, its sensitivity to effects such as turbulence. However, noise can make a positive contribution in some analog computing algorithms (e.g., for symmetry breaking and in simulated annealing, weight perturbation learning, and stochastic resonance).

As already mentioned, some analog computing devices for the direct solution of PDEs have been developed. In general a PDE solver depends on an analogous physical process, that is, on a process obeying the same class of PDEs that it is intended to solve. For example, in Mills' EAC, diffusion of electrons in conductive sheets or solids is used to solve diffusion equations (Mills, 1996; Mills et al., 2006). Historically, PDEs were solved on electronic GPACs by discretizing all but one of the independent variables, thus replacing the differential equations by difference equations (Rogers & Connolly, 1960, pp. 173–93). That is, computation over a field was approximated by computation over a finite real array.

Reaction-diffusion computation is an important example of continuous-time analog computing. The state is represented by a set of time-varying chemical concentration fields, $c_1, \ldots, c_n$. These fields are distributed across a one-, two-, or three-dimensional space Ω, so that, for $\mathbf{x} \in \Omega$, $c_k(\mathbf{x}, t)$ represents the concentration of chemical k at location $\mathbf{x}$ and time t. Computation proceeds in continuous time according to reaction-diffusion equations, which have the form:

$$\partial \mathbf{c}/\partial t = \mathrm{D}\nabla^2 \mathbf{c} + \mathbf{F}(\mathbf{c}),$$

where $\mathbf{c} = (c_1, \ldots, c_n)^{\mathrm{T}}$ is the vector of concentrations, $\mathrm{D} = \mathrm{diag}(d_1, \ldots, d_n)$ is a diagonal matrix of positive diffusion rates, and $\mathbf{F}$ is nonlinear vector function that describes how the chemical reactions affect the concentrations.

Some neural net models operate in continuous time and thus are examples of continuous-time analog computation. For example, Grossberg (Grossberg, 1967, 1973, 1976) defines the activity of a neuron by differential equations such as this:

$$\dot{x}_i = -a_i x_i + \sum_{j=1}^{n} b_{ij} w_{ij}^{(+)} f_j(x_j) - \sum_{j=1}^{n} c_{ij} w_{ij}^{(-)} g_j(x_j) + I_i.$$

This describes the continuous change in the activity of neuron i resulting

from passive decay (first term), positive feedback from other neurons (second term), negative feedback (third term), and input (last term). The f_j and g_j are nonlinear activation functions, and the $w_{ij}^{(+)}$ and $w_{ij}^{(-)}$ are adaptable excitatory and inhibitory connection strengths, respectively.

The continuous Hopfield network is another example of continuous-time analog computation (Hopfield, 1984). The output y_i of a neuron is a nonlinear function of its internal state x_i, $y_i = \sigma(x_i)$, where the hyperbolic tangent is usually used as the activation function, $\sigma(x) = \tanh x$, because its range is $[-1, 1]$. The internal state is defined by a differential equation,

$$\tau_i \dot{x}_i = -a_i x_i + b_i + \sum_{j=1}^{n} w_{ij} y_j,$$

where τ_i is a time constant, a_i is the decay rate, b_i is the bias, and w_{ij} is the connection weight to neuron i from neuron j. In a Hopfield network every neuron is symmetrically connected to every other ($w_{ij} = w_{ji}$) but not to itself ($w_{ii} = 0$).

Of course analog VLSI implementations of neural networks also operate in continuous time (e.g., Mead, 1989; Fakhraie & Smith, 1997)

Concurrent with the resurgence of interest in analog computation have been innovative reconceptualizations of continuous-time computation. For example, Brockett (1988) has shown that dynamical systems can perform a number of problems normally considered to be intrinsically sequential. In particular, a certain system of ODEs (a *nonperiodic finite Toda lattice*) can sort a list of numbers by continuous-time analog computation. The system is started with the vector $\mathbf{x}$ equal to the values to be sorted and a vector $\mathbf{y}$ initialized to small nonzero values; the $\mathbf{y}$ vector converges to a sorted permutation of $\mathbf{x}$.

C.2.b Sequential time

Sequential-time computation refers to computation in which discrete computational operations take place in succession but at no definite interval (van Gelder, 1997). Ordinary digital computer programs take place in sequential time, for the operations occur one after another, but the individual operations are not required to have any specific duration, so long as they take finite time.

One of the oldest examples of sequential analog computation is provided by the compass-and-straightedge constructions of traditional Euclidean geometry (Sec. B). These computations proceed by a sequence of discrete operations, but the individual operations involve continuous representations (e.g., compass settings, straightedge positions) and operate on a continuous state (the figure under construction). Slide rule calculation might seem to be an example of sequential analog computation, but if we look at it, we see that although the operations are performed by an analog device, the intermediate results are recorded digitally (and so this part of the state space is discrete). Thus it is a kind of hybrid computation.

The familiar digital computer automates sequential digital computations that once were performed manually by human "computers." Sequential analog computation can be similarly automated. That is, just as the control unit of an ordinary digital computer sequences digital computations, so a digital control unit can sequence analog computations. In addition to the analog computation devices (adders, multipliers, etc.), such a computer must provide variables and registers capable of holding continuous quantities between the sequential steps of the computation (see also Sec. C.2.c below).

The primitive operations of sequential-time analog computation are typically similar to those in continuous-time computation (e.g., addition, multiplication, transcendental functions), but integration and differentiation with respect to sequential time do not make sense. However, continuous-time integration within a single step, and space-domain integration, as in PDE solvers or field computation devices, are compatible with sequential analog computation.

In general, any model of digital computation can be converted to a similar model of sequential analog computation by changing the discrete state space to a continuum, and making appropriate changes to the rest of the model. For example, we can make an analog Turing machine by allowing it to write a bounded real number (rather than a symbol from a finite alphabet) onto a tape cell. The Turing machine's finite control can be altered to test for tape markings in some specified range.

Similarly, in a series of publications Blum, Shub, and Smale developed a theory of computation over the reals, which is an abstract model of sequential-time analog computation (Blum et al., 1998, 1988). In this "BSS model" programs are represented as flowcharts, but they are able to operate on real-valued variables. Using this model they were able to prove a number of theorems about the complexity of sequential analog algorithms.

The BSS model, and some other sequential analog computation models, assume that it is possible to make exact comparisons between real numbers (analogous to exact comparisons between integers or discrete symbols in digital computation) and to use the result of the comparison to control the path of execution. Comparisons of this kind are problematic because they imply infinite precision in the comparator (which may be defensible in a mathematical model but is impossible in physical analog devices), and because they make the execution path a discontinuous function of the state (whereas analog computation is usually continuous). Indeed, it has been argued that this is not "true" analog computation (Siegelmann, 1999, p. 148).

Many artificial neural network models are examples of sequential-time analog computation. In a simple feed-forward neural network, an input vector is processed by the layers in order, as in a pipeline. That is, the output of layer n becomes the input of layer $n + 1$. Since the model does not make any assumptions about the amount of time it takes a vector to be processed by each layer and to propagate to the next, execution takes place in sequential time. Most *recurrent* neural networks, which have feedback, also operate in sequential time, since the activities of all the neurons are updated synchronously (that is, signals propagate through the layers, or back to earlier layers, in lockstep).

Many artificial neural-net learning algorithms are also sequential-time analog computations. For example, the back-propagation algorithm updates a network's weights, moving sequentially backward through the layers.

In summary, the correctness of sequential time computation (analog or digital) depends on the *order* of operations, not on their *duration*, and similarly the efficiency of sequential computations is evaluated in terms of the *number* of operations, not on their *total duration*.

C.2.c Discrete time

Discrete-time analog computation has similarities to both continuous-time and sequential-time analog computation. Like the latter, it proceeds by a sequence of discrete (analog) computation steps; like the former, these steps occur at a constant rate in real time (e.g., some "frame rate"). If the real-time rate is sufficient for the application, then discrete-time computation can approximate continuous-time computation (including integration and differentiation).

Some electronic GPACs implemented discrete-time analog computation

by a modification of repetitive operation mode, called *iterative analog computation* (Ashley, 1963, ch. 9). Recall (Sec. B.1.b) that in repetitive operation mode a clock rapidly switched the computer between reset and compute modes, thus repeating the same analog computation, but with different parameters (set by the operator). However, each repetition was independent of the others. Iterative operation was different in that analog values computed by one iteration could be used as initial values in the next. This was accomplished by means of an analog memory circuit (based on an op amp) that sampled an analog value at the end of one compute cycle (effectively during hold mode) and used it to initialize an integrator during the following reset cycle. (A modified version of the memory circuit could be used to retain a value over several iterations.) Iterative computation was used for problems such as determining, by iterative search or refinement, the initial conditions that would lead to a desired state at a future time. Since the analog computations were iterated at a fixed clock rate, iterative operation is an example of discrete-time analog computation. However, the clock rate is not directly relevant in some applications (such as the iterative solution of boundary value problems), in which case iterative operation is better characterized as sequential analog computation.

The principal contemporary examples of discrete-time analog computing are in neural network applications to time-series analysis and (discrete-time) control. In each of these cases the input to the neural net is a sequence of discrete-time samples, which propagate through the net and generate discrete-time output signals. Many of these neural nets are recurrent, that is, values from later layers are fed back into earlier layers, which allows the net to remember information from one sample to the next.

C.3 Analog computer programs

The concept of a *program* is central to digital computing, both practically, for it is the means for programming general-purpose digital computers, and theoretically, for it defines the limits of what can be computed by a universal machine, such as a universal Turing machine. Therefore it is important to discuss means for describing or specifying analog computations.

Traditionally, analog computers were used to solve ODEs (and sometimes PDEs), and so in one sense a mathematical differential equation is one way to represent an analog computation. However, since the equations were usually not suitable for direct solution on an analog computer, the process of

programming involved the translation of the equations into a schematic diagram showing how the analog computing devices (integrators etc.) should be connected to solve the problem. These diagrams are the closest analogies to digital computer programs and may be compared to flowcharts, which were once popular in digital computer programming. It is worth noting, however, that flowcharts (and ordinary computer programs) represent sequences among operations, whereas analog computing diagrams represent functional relationships among variables, and therefore a kind of parallel data flow.

Differential equations and schematic diagrams are suitable for continuous-time computation, but for sequential analog computation something more akin to a conventional digital program can be used. Thus, as previously discussed (Sec. C.2.b), the BSS system uses flowcharts to describe sequential computations over the reals. Similarly, Moore (1996) defines recursive functions over the reals by means of a notation similar to a programming language.

In principle any sort of analog computation might involve constants that are arbitrary real numbers, which therefore might not be expressible in finite form (e.g., as a finite string of digits). Although this is of theoretical interest (see Sec. F.3 below), from a practical standpoint these constants could be set with about at most four digits of precision (Rogers & Connolly, 1960, p. 11). Indeed, automatic potentiometer-setting devices were constructed that read a series of decimal numerals from punched paper tape and used them to set the potentiometers for the constants (Truitt & Rogers, 1960, pp. 3-58–60). Nevertheless it is worth observing that analog computers do allow continuous inputs that need not be expressed in digital notation, for example, when the parameters of a simulation are continuously varied by the operator. In principle, therefore, an analog program can incorporate constants that are represented by a real-valued physical quantity (e.g., an angle or a distance), which need not be expressed digitally. Further, as we have seen (Sec. B.1.b), some electronic analog computers could compute a function by means of an arbitrarily drawn curve, that is, not represented by an equation or a finite set of digitized points. Therefore, in the context of analog computing it is natural to expand the concept of a program beyond discrete symbols to include continuous representations (scalar magnitudes, vectors, curves, shapes, surfaces, etc.).

Typically such continuous representations would be used as adjuncts to conventional discrete representations of the analog computational process, such as equations or diagrams. However, in some cases the most natural static

representation of the process is itself continuous, in which case it is more like a "guiding image" than a textual prescription (MacLennan, 1995). A simple example is a potential surface, which defines a continuum of trajectories from initial states (possible inputs) to fixed-point attractors (the results of the computations). Such a "program" may define a deterministic computation (e.g., if the computation proceeds by gradient descent), or it may constrain a nondeterministic computation (e.g., if the computation may proceed by any potential-decreasing trajectory). Thus analog computation suggests a broadened notion of programs and programming.

C.4 Characteristics of analog computation

C.4.a Precision

Analog computation is evaluated in terms of both accuracy and precision, but the two must be distinguished carefully (Ashley 1963, pp. 25–8, Weyrick 1969, pp. 12–13, Small 2001, pp. 257–61). *Accuracy* refers primarily to the relationship between a simulation and the primary system it is simulating or, more generally, to the relationship between the results of a computation and the mathematically correct result. Accuracy is a result of many factors, including the mathematical model chosen, the way it is set up on a computer, and the precision of the analog computing devices. *Precision*, therefore, is a narrower notion, which refers to the quality of a representation or computing device. In analog computing, precision depends on *resolution* (fineness of operation) and *stability* (absence of drift), and may be measured as a fraction of the represented value. Thus a precision of 0.01% means that the representation will stay within 0.01% of the represented value for a reasonable period of time. For purposes of comparing analog devices, the precision is usually expressed as a fraction of *full-scale variation* (i.e., the difference between the maximum and minimum representable values).

It is apparent that the precision of analog computing devices depends on many factors. One is the choice of physical process and the way it is utilized in the device. For example a linear mathematical operation can be realized by using a linear region of a nonlinear physical process, but the realization will be approximate and have some inherent imprecision. Also, associated, unavoidable physical effects (e.g., loading, and leakage and other losses) may prevent precise implementation of an intended mathematical function. Further, there are fundamental physical limitations to resolution

(e.g., quantum effects, diffraction). Noise is inevitable, both intrinsic (e.g., thermal noise) and extrinsic (e.g., ambient radiation). Changes in ambient physical conditions, such as temperature, can affect the physical processes and decrease precision. At slower time scales, materials and components age and their physical characteristics change. In addition, there are always technical and economic limits to the control of components, materials, and processes in analog device fabrication.

The precision of analog and digital computing devices depend on very different factors. The precision of a (binary) digital device depends on the number of bits, which influences the amount of hardware, but not its quality. For example, a 64-bit adder is about twice the size of a 32-bit adder, but can made out of the same components. At worst, the size of a digital device might increase with the square of the number of bits of precision. This is because binary digital devices only need to represent two states, and therefore they can operate in saturation. The fabrication standards sufficient for the first bit of precision are also sufficient for the 64th bit. Analog devices, in contrast, need to be able to represent a continuum of states precisely. Therefore, the fabrication of high-precision analog devices is much more expensive than low-precision devices, since the quality of components, materials, and processes must be much more carefully controlled. Doubling the precision of an analog device may be expensive, whereas the cost of each additional bit of digital precision is incremental; that is, the cost is proportional to the logarithm of the precision expressed as a fraction of full range.

The forgoing considerations might seem to be a convincing argument for the superiority of digital to analog technology, and indeed they were an important factor in the competition between analog and digital computers in the middle of the twentieth century (Small, 2001, pp. 257–61). However, as was argued at that time, many computer applications do not require high precision. Indeed, in many engineering applications, the input data are known to only a few digits, and the equations may be approximate or derived from experiments. In these cases the very high precision of digital computation is unnecessary and may in fact be misleading (e.g., if one displays all 14 digits of a result that is accurate to only three). Furthermore, many applications in image processing and control do not require high precision. More recently, research in artificial neural networks (ANNs) has shown that low-precision analog computation is sufficient for almost all ANN applications. Indeed, neural information processing in the brain seems to operate with very low precision — perhaps less than 10% (McClelland et al., 1986, p. 378) —

for which it compensates with massive parallelism. For example, by *coarse coding* a population of low-precision devices can represent information with relatively high precision (Rumelhart et al. 1986, pp. 91–6, Sanger 1996).

C.4.b Scaling

An important aspect of analog computing is *scaling*, which is used to adjust a problem to an analog computer. First is *time scaling*, which adjusts a problem to the characteristic time scale at which a computer operates, which is a consequence of its design and the physical processes by which it is realized (Peterson 1967, pp. 37–44, Rogers & Connolly 1960, pp. 262–3, Weyrick 1969, pp. 241–3). For example, we might want a simulation to proceed on a very different time scale from the primary system. Thus a weather or economic simulation should proceed faster than real time in order to get useful predictions. Conversely, we might want to slow down a simulation of protein folding so that we can observe the stages in the process. Also, for accurate results it is necessary to avoid exceeding the maximum response rate of the analog devices, which might dictate a slower simulation speed. On the other hand, too slow a computation might be inaccurate as a consequence of instability (e.g., drift and leakage in the integrators).

Time scaling affects only time-dependant operations such as integration. For example, suppose t, time in the primary system or "problem time," is related to τ, time in the computer, by $\tau = \beta t$. Therefore, an integration $u(t) = \int_0^t v(t')\mathrm{d}t'$ in the primary system is replaced by the integration $u(\tau) = \beta^{-1} \int_0^\tau v(\tau')\mathrm{d}\tau'$ on the computer. Thus time scaling may be accomplished simply by decreasing the input gain to the integrator by a factor of β.

Fundamental to analog computation is the representation of a continuous quantity in the primary system by a continuous quantity in the computer. For example, a displacement x in meters might be represented by a potential V in volts. The two are related by an *amplitude* or *magnitude scale factor*, $V = \alpha x$, (with units volts/meter), chosen to meet two criteria (Ashley 1963, pp. 103–6, Peterson 1967, ch. 4, Rogers & Connolly 1960, pp. 127–8, Weyrick 1969, pp. 233–40). On the one hand, α must be sufficiently small so that the range of the problem variable is accommodated within the range of values supported by the computing device. Exceeding the device's intended operating range may lead to inaccurate results (e.g., forcing a linear device into nonlinear behavior). On the other hand, the scale factor should not be too small, or relevant variation in the problem variable will be less than the resolution of

the device, also leading to inaccuracy. (Recall that precision is specified as a fraction of full-range variation.)

In addition to the explicit variables of the primary system, there are implicit variables, such as the time derivatives of the explicit variables, and scale factors must be chosen for them too. For example, in addition to displacement x, a problem might include velocity $\dot{x}$ and acceleration $\ddot{x}$. Therefore, scale factors α, α', and α'' must be chosen so that αx, $\alpha'\dot{x}$, and $\alpha''\ddot{x}$ have an appropriate range of variation (neither too large nor too small).

Once a scale factor has been chosen, the primary system equations are adjusted to obtain the analog computing equations. For example, if we have scaled $u = \alpha x$ and $v = \alpha'\dot{x}$, then the integration $x(t) = \int_0^t \dot{x}(t')\mathrm{d}t'$ would be computed by scaled equation:

$$u(t) = \frac{\alpha}{\alpha'} \int_0^t v(t')\mathrm{d}t'.$$

This is accomplished by simply setting the input gain of the integrator to α/α'.

In practice, time scaling and magnitude scaling are not independent (Rogers & Connolly, 1960, p. 262). For example, if the derivatives of a variable can be large, then the variable can change rapidly, and so it may be necessary to slow down the computation to avoid exceeding the high-frequency response of the computer. Conversely, small derivatives might require the computation to be run faster to avoid integrator leakage etc. Appropriate scale factors are determined by considering both the physics and the mathematics of the problem (Peterson, 1967, pp. 40–4). That is, first, the physics of the primary system may limit the ranges of the variables and their derivatives. Second, analysis of the mathematical equations describing the system can give additional information on the ranges of the variables. For example, in some cases the natural frequency of a system can be estimated from the coefficients of the differential equations; the maximum of the nth derivative is then estimated as the n power of this frequency (Peterson 1967, p. 42, Weyrick 1969, pp. 238–40). In any case, it is not necessary to have accurate values for the ranges; rough estimates giving orders of magnitude are adequate.

It is tempting to think of magnitude scaling as a problem unique to analog computing, but before the invention of floating-point numbers it was also necessary in digital computer programming. In any case it is an essential aspect of analog computing, in which physical processes are more directly used

for computation than they are in digital computing. Although the necessity of scaling has been a source of criticism, advocates for analog computing have argued that it is a blessing in disguise, because it leads to improved understanding of the primary system, which was often the goal of the computation in the first place (Bissell 2004, Small 2001, ch. 8). Practitioners of analog computing are more likely to have an intuitive understanding of both the primary system and its mathematical description (see Sec. G).

D Analog Computation in Nature

Computational processes—that is to say, information processing and control—occur in many living systems, most obviously in nervous systems, but also in the self-organized behavior of groups of organisms. In most cases natural computation is analog, either because it makes use of continuous natural processes, or because it makes use of discrete but stochastic processes. Several examples will be considered briefly.

D.1 Neural computation

In the past neurons were thought of binary computing devices, something like digital logic gates. This was a consequence of the "all or nothing" response of a neuron, which refers to the fact that it does or does not generate an *action potential* (voltage spike) depending, respectively, on whether its total input exceeds a threshold or not (more accurately, it generates an action potential if the membrane depolarization at the axon hillock exceeds the threshold and the neuron is not in its refractory period). Certainly some neurons (e.g., so-called "command neurons") do act something like logic gates. However, most neurons are analyzed better as analog devices, because the *rate* of impulse generation represents significant information. In particular, an *amplitude code*, the membrane potential near the axon hillock (which is a summation of the electrical influences on the neuron), is translated into a *rate code* for more reliable long-distance transmission along the axons. Nevertheless, the code is low precision (about one digit), since information theory shows that it takes at least N milliseconds (and probably more like $5N$ msec.) to discriminate N values (MacLennan, 1991). The rate code is translated back to an amplitude code by the synapses, since successive impulses release neurotransmitter from the axon terminal, which diffuses across the synaptic

cleft to receptors. Thus a synapse acts as a leaky integrator to time-average the impulses.

As previously discussed (Sec. C.1), many artificial neural net models have real-valued neural activities, which correspond to rate-encoded axonal signals of biological neurons. On the other hand, these models typically treat the input connections as simple real-valued weights, which ignores the analog signal processing that takes place in the dendritic trees of biological neurons. The dendritic trees of many neurons are complex structures, which often have tens of thousands of synaptic inputs. The binding of neurotransmitters to receptors causes minute voltage fluctuations, which propagate along the membrane, and ultimately cause voltage fluctuations at the axon hillock, which influence the impulse rate. Since the dendrites have both resistance and capacitance, to a first approximation the signal propagation is described by the "cable equations," which describe passive signal propagation in cables of specified diameter, capacitance, and resistance (Anderson, 1995, ch. 1). Therefore, to a first approximation, a neuron's dendritic net operates as an adaptive linear analog filter with thousands of inputs, and so it is capable of quite complex signal processing. More accurately, however, it must be treated as a *nonlinear* analog filter, since voltage-gated ion channels introduce nonlinear effects. The extent of analog signal processing in dendritic trees is still poorly understood.

In most cases, then, neural information processing is treated best as low-precision analog computation. Although individual neurons have quite broadly tuned responses, accuracy in perception and sensorimotor control is achieved through coarse coding, as already discussed (Sec. C.4). Further, one widely used neural representation is the *cortical map*, in which neurons are systematically arranged in accord with one or more dimensions of their stimulus space, so that stimuli are represented by patterns of activity over the map. (Examples are *tonotopic maps*, in which pitch is mapped to cortical location, and *retinotopic maps*, in which cortical location represents retinal location.) Since neural density in the cortex is at least 146 000 neurons per square millimeter (Changeux, 1985, p. 51), even relatively small cortical maps can be treated as fields and information processing in them as analog field computation. Overall, the brain demonstrates what can be accomplished by massively parallel analog computation, even if the individual devices are comparatively slow and of low precision.

D.2 Adaptive self-organization in social insects

Another example of analog computation in nature is provided by the self-organizing behavior of social insects, microorganisms, and other populations (Camazine et al., 2001). Often such organisms respond to concentrations, or gradients in the concentrations, of chemicals produced by other members of the population. These chemicals may be deposited and diffuse through the environment. In other cases, insects and other organisms communicate by contact, but may maintain estimates of the relative proportions of different kinds of contacts. Because the quantities are effectively continuous, all these are examples of analog control and computation.

Self-organizing populations provide many informative examples of the use of natural processes for analog information processing and control. For example, diffusion of pheromones is a common means of self-organizzation in insect colonies, facilitating the creation of paths to resources, the construction of nests, and many other functions (Camazine et al., 2001). Real diffusion (as opposed to sequential simulations of it) executes, in effect, a massively parallel search of paths from the chemical's source to its recipients and allows the identification of near-optimal paths. Furthermore, if the chemical degrades, as is generally the case, then the system will be adaptive, in effect continually searching out the shortest paths, so long as source continues to function (Camazine et al., 2001). Simulated diffusion has been applied to robot path planning (Khatib, 1986; Rimon & Koditschek, 1989).

D.3 Genetic circuits

Another example of natural analog computing is provided by the *genetic regulatory networks* that control the behavior of cells, in multicellular organisms as well as single-celled ones (Davidson, 2006). These networks are defined by the mutually interdependent regulatory genes, promoters, and repressors that control the internal and external behavior of a cell. The interdependencies are mediated by proteins, the synthesis of which is governed by genes, and which in turn regulate the synthesis of other gene products (or themselves). Since it is the quantities of these substances that is relevant, many of the regulatory motifs can be described in computational terms as adders, subtracters, integrators, etc. Thus the genetic regulatory network implements an analog control system for the cell (Reiner, 1968).

It might be argued that the number of intracellular molecules of a par-

ticular protein is a (relatively small) discrete number, and therefore that it is inaccurate to treat it as a continuous quantity. However, the molecular processes in the cell are stochastic, and so the relevant quantity is the *probability* that a regulatory protein will bind to a regulatory site. Further, the processes take place in continuous real time, and so the rates are generally the significant quantities. Finally, although in some cases gene activity is either on or off (more accurately: very low), in other cases it varies continuously between these extremes (Hartl, 1994, pp. 388–90).

Embryological development combines the analog control of individual cells with the sort of self-organization of populations seen in social insects and other colonial organisms. Locomotion of the cells and the expression of specific genes is controlled by chemical signals, among other mechanisms (Davidson, 2006; Davies, 2005). Thus PDEs have proved useful in explaining some aspects of development; for example *reaction-diffusion equations* have been used to describe the formation of hair-coat patterns and related phenomena (Camazine et al., 2001; Maini & Othmer, 2001; Murray, 1977). Therefore the developmental process is governed by naturally occurring analog computation.

D.4 Is everything a computer?

It might seem that any continuous physical process could be viewed as analog computation, which would make the term almost meaningless. As the question has been put, is it meaningful (or useful) to say that the solar system is *computing* Kepler's laws? In fact, it is possible and worthwhile to make a distinction between computation and other physical processes that happen to be described by mathematical laws (MacLennan, 1994a,c, 2001, 2004).

If we recall the original meaning of analog computation (Sec. A), we see that the computational system is used to solve some mathematical problem with respect to a primary system. What makes this possible is that the computational system and the primary system have the same, or systematically related, abstract (mathematical) structures. Thus the computational system can inform us about the primary system, or be used to control it, etc. Although from a practical standpoint some analogs are better than others, in principle any physical system can be used that obeys the same equations as the primary system.

Based on these considerations we may define computation as a physical process the purpose of which is the abstract manipulation of abstract objects

(i.e., information processing); this definition applies to analog, digital, and hybrid computation (MacLennan, 1994a,c, 2001, 2004). Therefore, to determine if a natural system is computational we need to look to its purpose or function within the context of the living system of which it is a part. One test of whether its function is the abstract manipulation of abstract objects is to ask whether it could still fulfill its function if realized by different physical processes, a property called *multiple realizability*. (Similarly, in artificial systems, a simulation of the economy might be realized equally accurately by a hydraulic analog computer or an electronic analog computer (Bissell, 2004).) By this standard, the majority of the nervous system is purely computational; in principle it could be replaced by electronic devices obeying the same differential equations. In the other cases we have considered (self-organization of living populations, genetic circuits) there are instances of both pure computation and computation mixed with other functions (for example, where the specific substances used have other—e.g. metabolic—roles in the living system).

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

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

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 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 contempo-

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

rary 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 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]

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

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.

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.