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 particular 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).