MODELING AND SIMULATION

MODELS AND MODEL THEORY

A *model* is an entity that is used to represent some other entity for some well-defined purpose. In this most general sense, examples of models include:

- *Mental* models, such as the internalized conception of a person's relationships with the environment, used to guide everyday behavior
- *Iconic* models, such as (a) a circuit diagram used to represent the functional interconnection of electronic components or (b) a map used to record geographical, geological, or meteorological data
- *Linguistic* models, such as (a) a verbal protocol for a biological experiment, or (b) a written specification defining the purpose, requirements, and operation of a software program
- *Physical* models, such as (a) a scale mock-up of an airfoil used in wind-tunnel testing for a new aircraft design or (b) an analog circuit developed to replicate the neural activity of the brain
- *Mathematical* and *computational* models, such as (a) the set of mass- and energy-balance equations that predict the end products of a chemical reaction or (b) the mathematical and logical relations embodied in a computer program that simulates the behavior of an electromechanical device

Models are a mainstay of every scientific and engineering discipline. Social and management scientists also make extensive use of models. The specific models adopted in different disciplines differ in subject, form, and intended use, and every discipline tends to develop its own approach and techniques for studying models. However, basic concepts such as model description, simplification, solution, and validation are universal across disciplines.

Model theory (1) seeks a logical and axiomatic understanding of these common underlying concepts, independently of their particular expression and any modeling endeavor. At its core, the theory of models and modeling cannot be divorced from broader philosophical issues that concern the origins, nature, methods, and limits of human knowledge (epistemology) and the means of rational inquiry (logic and the scientific method). Philosophical notions of correlation and causality are central to model theory.

For example, a department store may have data that show that more umbrellas are sold on days when it rains and fewer umbrellas are sold on days when the sun shines. A positive *correlation* between the average number of umbrellas sold each month and the average amount of rain that falls in that month seems entirely plausible. A model which captures this relationship might be used to predict umbrella sales from a record of the amount of precipitation received. Moreover, since it is easy to imagine that increased precipitation *causes* customers to buy umbrellas, this model also might be used to show how many more umbrellas the store might sell if somehow customers could be induced to believe that it would rain. The *correlation* also holds in the opposite direction—we might well use the inverse model to predict the amount of precipitation received based on a record of umbrella sales. However, it would be difficult to support the idea that umbrella sales *cause* it to rain. A model that shows how much more it would rain if we could somehow increase umbrella sales clearly would not be credible.

Models are pervasive in fields of inquiry simply because a good model is more accessible to study than the actual entity the model represents. Models typically are less costly and less time-consuming to build and analyze. Variants in the parameters and structure of a model are easier to implement, and the resulting changes in model behavior are easier to isolate, understand, and communicate to others. A model can be used to achieve insight when direct experimentation with the actual system is too demanding, disruptive, or dangerous to undertake. Indeed, a model can be developed to answer questions about a system that has not yet been observed or constructed, or even one that cannot be measured with current instrumentation or realized with current technologies.

MATHEMATICAL SYSTEMS MODELS

Mathematical and computational models are particularly useful because of the rich body of theory and wide range of quantitative and computational techniques that exist for the study of logical expressions and the solution of equations. The availability and power of digital computers have increased the use and importance of mathematical models and computer simulation in all modern disciplines. A great variety of programming languages and applications software is now available for modeling, data reduction and model calibration, computational analysis, system simulation, and model visualization. Icon-based, drag-and-drop programming environments have virtually eliminated the need for writing code and have automated many of the other routine tasks formerly associated with developing and analyzing models and simulations. Given these powerful tools, many applications that formerly relied on other types of models now also use mathematical models and computer implementations extensively.

Much of the current thinking about models and modeling is intimately tied to mathematical system theory. Broadly, a system is a collection of elements that interact to achieve some purpose. The boundary of a system separates the elements internal to the system from the environment external to the system.

The key attributes of a system are represented by a set of *parameters,* with essentially fixed values, and three sets of *system variables,* which may assume different values at different times and/or at different locations in space. The action of the environment on the system is defined by a set of *input* or predictor variables. Inputs typically are represented by the input vector $\boldsymbol{u} \in \boldsymbol{U}$, where the set $\boldsymbol{U}$ of all values that can be assumed by the input is called the input space. The internal condition of the system is defined

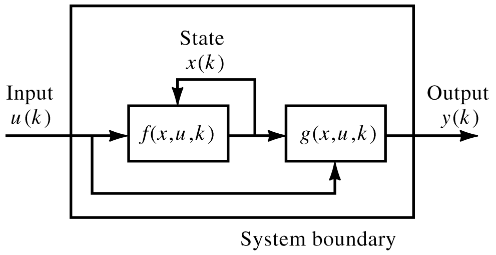

Figure 1. Mathematical systems model showing state and output equations and the relationships among input, state, and output variables. Arrows indicate direction of causal relationships.

by a set of *state variables* represented by the state vector $\boldsymbol{x} \in \boldsymbol{X}$, where $\boldsymbol{X}$ is the state space. The action of the system on the environment is defined by a set of *output* or predicted variables represented by the output vector $\boldsymbol{y} \in \boldsymbol{Y}$, where $\boldsymbol{Y}$ is the output space. Variations in the system variables over time and/or space are ordered by a set of *index variables* represented by the vector $\boldsymbol{k} \in \boldsymbol{K}$, defined on the index set $\boldsymbol{K}$.

The relationship between system inputs and outputs is determined by the parameters and structure of the system, as mediated by the independent variables and system state. In other words, the input alters the internal state of the system and the altered state is observed in turn through changes in the output for various values of the independent variables. These relationships are shown in Fig. 1. The state equation

$$\boldsymbol{f} : (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{k}) \rightarrow \boldsymbol{x}$$

describes how the new state is determined from the inputs and current state. The output equation

$$\boldsymbol{g} : (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{k}) \rightarrow \boldsymbol{y}$$

describes how the current output is determined from the current input and state.

The pattern of changes exhibited by the system output in response to any particular series of inputs is called an output *trajectory,* $(\boldsymbol{u}_i, \boldsymbol{y}_i)$. The collection of all possible trajectories $\{(\boldsymbol{u}_i, \boldsymbol{y}_i), \forall i\}$ represents the *behavior* of the system. Clearly, this behavior is generated by the functions $\boldsymbol{f}$ and $\boldsymbol{g}$ and is the fundamental interest in model-based studies.

In this formalism, a system is completely determined by the algebraic structure $\{\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{f}, \boldsymbol{g}, \boldsymbol{K}\}$. If a system S_o has the structure $\{\boldsymbol{U}_o, \boldsymbol{Y}_o, \boldsymbol{X}_o, \boldsymbol{f}_o, \boldsymbol{g}_o, \boldsymbol{K}_o\}$, then a model of S_o is just some other system S_m with structure $\{\boldsymbol{U}_m, \boldsymbol{Y}_m, \boldsymbol{X}_m, \boldsymbol{f}_m, \boldsymbol{g}_m, \boldsymbol{K}_m\}$. The model system S_m is used as a proxy for object system S_o for some well-defined purpose. Clearly, there can be many alternative models $S_{m(i)}$ for S_o, $i = 1, 2, \ldots$. Model theory deals in a fundamental way with the nature and adequacy of the correspondence of these alternative systems, depending on the intended purpose of the modeling exercise.

CLASSIFICATION OF SYSTEM MODELS

Mathematical models are distinguished by criteria that describe the fundamental properties of model variables and equations. These criteria in turn prescribe the appropriate theory and mathematical techniques that can be used to study and or solve alternative models. These criteria and the classification of models based on these criteria are given in the following subsections.

Number of System Variables

Models that include a single input, state, and output are called *scalar* models, because the system-dependent variables assume scalar values. Models that include multiple system variables are called *multivariate models* or *MIMO* (*multiple-input, multiple-output*) models.

Continuity of the System Variables

Continuous-state models are those in which the system variables are continuously variable over a (finite or infinite) range of permissible values; that is, $\boldsymbol{U}$, $\boldsymbol{Y}$, and $\boldsymbol{X}$ are continuous vector spaces for real-valued vectors. Continuous-state models are typical of physical systems at macroscopic resolution, where key dependent variables might include flows (currents, forces, fluid flows, and heat fluxes) and potentials (voltages, velocities, pressures, and temperatures). *Discrete-state* models are those in which the system variables assume only a countable number of different values; that is, $\boldsymbol{U}$, $\boldsymbol{Y}$, and $\boldsymbol{X}$ are countably finite or countably infinite vector spaces. Discrete-state models are typical of sample-data systems, including computer control and communication systems, as well as systems in which variables are measured naturally in terms of item counts (e.g., jobs, packets, parts, or people).

Number of Index Variables

A fundamental distinction can be made between mathematical models based on the number of index variables incorporated within the model. *Static models* relate the values of the input to the corresponding values of the output, without explicit reference to the index set $\boldsymbol{K}$. In many cases, static models are used to organize and summarize experimental data, derived from empirical tests or generated by computer simulations. In contrast to static models, *dynamic* models seek to explain the behavior of a system in response to changes over time and/or space. *Lumped-parameter* dynamic models involve a single index variable, most often time. Lumped-parameter models are common in circuit design and control engineering. *Distributed-parameter* dynamic models involve multiple index variables, most often time and one or more spatial coordinates. Distributed-parameter models are common in the study of structures and of mass and heat transport.

Continuity of the Index Variable

In *continuous-time* dynamic models, the system variables are defined over a continuous range of the index variables, even though the dependence is not necessarily described by a mathematically continuous function. Continuous-state, continuous-time models (sometimes called *analog models*) are the staple of classical engineering science. In *discrete-time* dynamic models, the system variables are defined only at distinct instants of time. Discrete-time models are typical of digital and computer-based systems. In *discrete-event* dynamic models, the index variable is a discrete index which references a set of events. The events are instantaneous (they take no time to occur), strictly sequential (no two events occur at exactly the same instant of time), and asynchronous (the time between events can vary). The occurrence of events typically is determined by the internal logic of the system and the system state changes only in response to these occurrences. Discrete-event models are typical of a wide range of man-made systems, where queuing for shared resources is a key determinant in the behavior and performance of the system. These include models typically developed for the operation of manufacturing, distribution, transportation, computer and communications networks, and service systems.

Linearity and Superposition

Consider a system with arbitrary trajectories $(\boldsymbol{u}_i, \boldsymbol{y}_i)$ and $(\boldsymbol{u}_j, \boldsymbol{y}_j)$. If the output to a linear combination of the inputs is simply a linear combination of the independent outputs [i.e., if for some constant vectors $\boldsymbol{a}_i$, $\boldsymbol{a}_j$, $\boldsymbol{b}_i$, and $\boldsymbol{b}_j$ the pair $(\boldsymbol{a}_i\boldsymbol{u}_i + \boldsymbol{a}_j\boldsymbol{u}_j, \boldsymbol{b}_i\boldsymbol{y}_i + \boldsymbol{b}_j\boldsymbol{y}_j)$ also is a valid trajectory], then the *principle of superposition* applies for these inputs. Systems for which superposition applies over the range of inputs of interest are represented by *linear* models over this range. *Nonlinear* models, for which superposition does not apply, typically are significantly more difficult or even impossible to solve analytically. For this reason, nonlinear models often are approximated by linear models for analytical ease.

Treatment of Uncertainty

Models in which the parameters and variables can be known with a high degree of certainty are called *deterministic* models. While the values of system attributes are never known with infinite precision, deterministic models are a common and useful approximation when actual uncertainties are small over the range of values of interest for the purpose of the modeling exercise. *Probabilistic* models are used when significant uncertainty exists in the values of some parameters and/or variables. Model parameters and variables are expressed as random numbers or processes and are characterized by the parameters of probability distributions. *Stochastic* models are those which are both probabilistic and dynamic.

Mixed Classifications

Models that include combinations of continuous-time, discrete-time, and/or discrete-event subsystems are called *hybrid models.* Hybrid models are most common in computer control and communication systems, where physical devices represented by continuous-time, continuous-state models and computer control device are represented by discrete-time models. Continuous variables are sampled and quantized using A/D (analog-to-digital) conversion, control action is determined digitally, and control signals are reconstructed using D/A (digital-to-analog) conversion. Hybrid models sometimes are approximated as entirely continuous or entirely discrete.

SOME COMMON MODEL FORMS

Response Surfaces

Static models do not make explicit reference to index variables. If explicit reference to the state variables also is suppressed, then the resulting static model reduces to a set of coupled, input–output equations of the form

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{u})$$

Experimenters frequently employ models of this form to organize field or simulation data, to explore the corresponding correlation between system inputs and outputs, to hypothesize causal relationships for the underlying system, or to summarize data for efficient storage, search, or optimization. When used for this purpose, models of this form are called *response surfaces* (2). A response surface can be determined by regression over a sample of n observations of input–output pairs $(\boldsymbol{u}_j, \boldsymbol{y}_j)$, $j = 1, \boldsymbol{n}$. The input and observed output values can be scaled using alternative transformations $\alpha(\boldsymbol{u}_j)$ and $\beta(\boldsymbol{y}_j)$ and alternative functional forms $\boldsymbol{f}$ for the surface

$$\beta(\boldsymbol{y}_j) = \boldsymbol{f}(\alpha(\boldsymbol{u}_j))$$

can tested. The objective is to find a surface that provides a good fit between the data the input–output values predicted by the model. Response surface methodology (RSM) has been developed in recent years to build empirical models and to apply these models in model-based optimization studies.

For example, Fig. 2 shows the discrete-time trajectories of umbrella sales and precipitation each month for one calendar year. The scatter plot in Fig. 3 illustrates the correlation between sales and precipitation. The straight line in Fig. 3 is the response surface $y = 70.1u$, where monthly precipitation (in inches) is the input u and umbrella sales (in units) is the output y.

Analog State-Variable Models

Continuous-state, continuous-time, lumped-parameter models are sometimes called *analog models.* Analog models are natural representations for a great many physical systems, including systems with electrical, mechanical, fluid, and/or thermal subsystems. As such, analog models are a staple of classical engineering science. The solution of an analog model relates the current value of the system output (at some arbitrary time t) to the current value of the system state through the output equation

$$\boldsymbol{y}(t) = \boldsymbol{g}(\boldsymbol{x}(t))$$

The current value of the state is determined from the known initial value of the state (at some earlier time t_0) based on the history of inputs and states over the interval from t_0 to t:

$$\boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau)\,d\tau$$

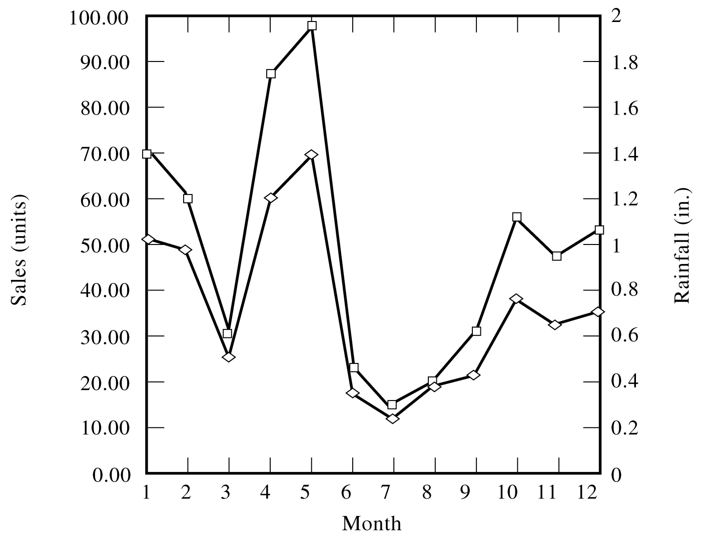

Figure 2. Discrete-time trajectories of umbrella sales and precipitation each month for one calendar year.

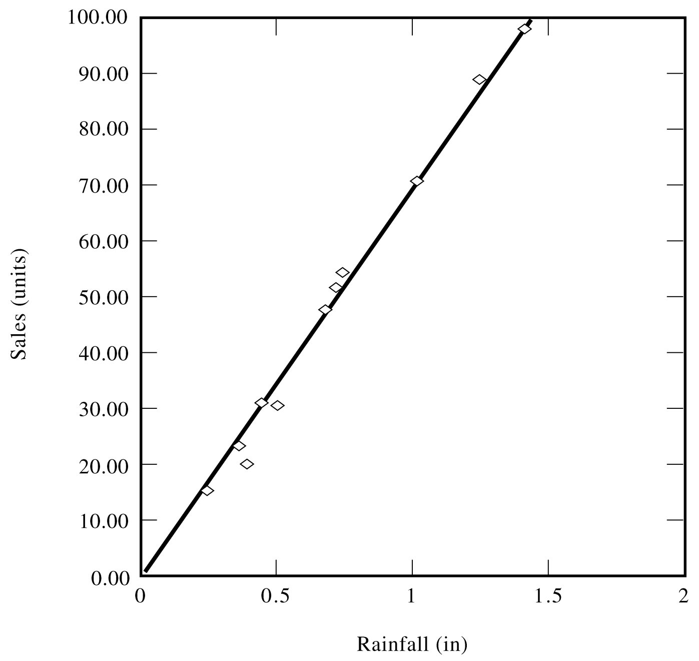

Figure 3. Scatter plot illustrating the correlation between monthly sales and precipitation. The straight line is the response surface $y = 70.1u$, where monthly precipitation (in inches) is the input u and umbrella sales (in units) is the output y.

This equation is the solution to the *state-variable* model, represented by the vector differential equation

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{1}$$

For linear, time-invariant systems, the state-variable model has the form

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$$

that is readily solved by standard techniques. State-variable methods of this form are the basis for modern control theory.

Finite-Element Models

Continuous-time, continuous-state, distributed-parameter models commonly arise in the study of electrical transmission, mass and heat transport, and the mechanics of complex structures and structural components. These models are described by partial differential equations, containing partial derivatives with respect to each of the index variables. These equations typically are so complex that direct solutions are difficult or impossible to obtain. To circumvent this difficulty, distributed-parameter models can be approximated by a finite number of spatial "lumps," each characterized by some average value of the state within the lump. By eliminating the independent spatial coordinates,

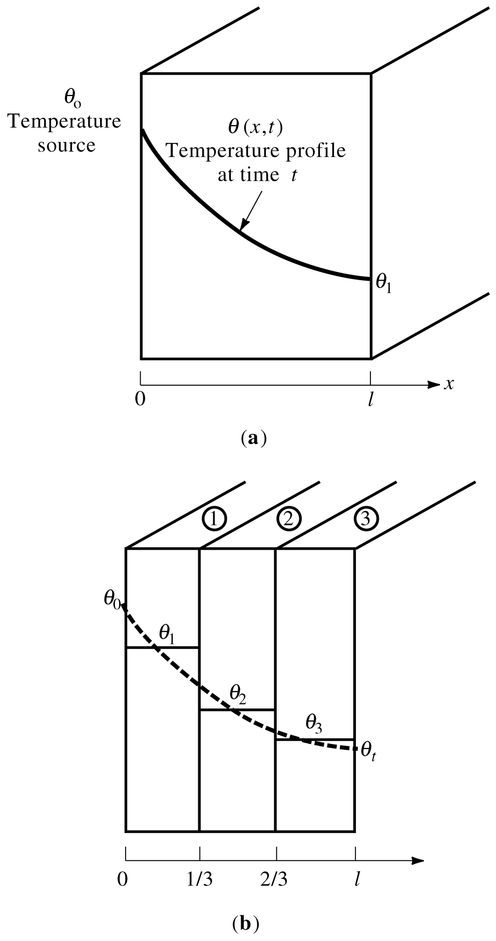

Figure 4. Heat transfer through a wall: (a) Distributed parameter model and, (b) approximating lumped-parameter model for three state variables.

the result is an analog model of the form previously described. If a sufficiently fine-grained representation of the lumped microstructure can be achieved, an analog model can be derived that will approximate the distributed model to any desired degree of accuracy. Increasing the granularity of the approximation requires increasing in the dimensions of the analog model, however, with a resulting compromise between model accuracy and precision and the difficulty of solving higher-order differential equations. Figure 4 illustrates the approximation of a distributed parameter model for temperature in a one-dimensional wall, where $\theta(x, t)$ is the temperature depth x and time t, with a lumped-parameter model for three state variables, where $\theta_i(t)$ is the temperature at depth x_i, $i = 1, 2, 3$, also at time t.

Discrete-Time State-Variable Models

Discrete-time models are natural representations for computer-based systems, in which operations are synchronized by an internal digital clock, as well as for many social, economic, and management systems, in which data are sampled and recorded at intervals according to a survey schedule. The solution of a discrete-time model relates the current value of the system output (at some arbitrary instant t_k) to the current value of the system state

$$\boldsymbol{y}(k) = \boldsymbol{g}(\boldsymbol{x}(k))$$

The current value of the state is determined from the known initial value of the state (at some earlier instant t_0) based on the history of inputs and states over the k steps from t_0 to t_k

$$\boldsymbol{x}(k) = \boldsymbol{f}^{(k)}(\boldsymbol{x}(0), \boldsymbol{u}(0), t_0)$$

The k-step transition function $\boldsymbol{f}^{(k)}$ is the solution to the state-variable model represented by the first-order vector difference equation

$$\boldsymbol{x}(k+1) = \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k), k)$$

For linear time-invariant systems, the discrete-time state-variable model has the form

$$x(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k)$$

which is readily solved by standard techniques, very similar to those for differential equations. Perhaps more importantly, nonlinear discrete-time state variables can be solved iteratively. Beginning with the known initial state, successive values of the state can be computed from the preceding (computed) value of the state and the corresponding, known value of the input as

$$\begin{aligned}
\boldsymbol{x}(1) &= \boldsymbol{f}(\boldsymbol{x}(0), \boldsymbol{u}(0), 0)\\
\boldsymbol{x}(2) &= \boldsymbol{f}(\boldsymbol{x}(1), \boldsymbol{u}(1), 1)\\
&\vdots\\
\boldsymbol{x}(k) &= \boldsymbol{f}(\boldsymbol{x}(k-1), \boldsymbol{u}(k-1), k-1)\\
\boldsymbol{x}(k+1) &= \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k), k)
\end{aligned}$$

Implementation of this *iterative* or *numerical solution* strategy on a digital computer is straightforward, allowing extraordinarily complicated and otherwise difficult difference equations to be solved quickly and easily for specific inputs, initial conditions, and parameter values. Moreover, differential equations that are difficult or impossible to solve analytically can be approximated by difference equations, that in turn can be solved numerically. This computational approach to solving state-variable models is the basis for continuous system simulation, described in the following.

SIMULATION

Simulation is a model-based approach to the design, analysis, and control of systems which is fundamentally *experimental.* In principle, computer simulation is much like running laboratory or field tests, except that the physical system is replaced by a computational model. Broadly speaking, simulation involves creating a model which imitates the behavior of interest; running the model to generate observations of this behavior; analyzing the observations to understand and summarize this behavior; testing and comparing alternative designs and controls to improve system performance; and validating, explaining, and supporting the simulation outcomes and recommendations.

A *simulation run* or *replication* is a controlled experiment in which a specific realization of the model is manipulated in order to determine the response associated with that realization. A simulation study always comprises *multiple runs.* For deterministic models, one run must be completed for each different combination of model parameters and/or initial conditions under consideration. The generalized solution of the model then must be inferred from a finite number of such runs.

For stochastic models, in which inputs and outputs are realizations of random variables, the situation is even more complicated. Multiple runs, each using different input random number streams, must be completed for *each* combination of model parameters and/or initial conditions. The response for this combination must be inferred statistically from the set of runs or sample paths. The generalized solution in turn must be inferred, again statistically, from multiple sets of multiple runs.

Simulation stands in contrast to analytical approaches to the solution of models. In an analytical approach, the model is expressed as a set of equations that describe how the state changes over time. We solve these equations using standard mathematical methods—algebra, calculus, or numerical analysis—to determine the distribution of the state at any particular time. The result is a general, closed-form solution, which gives the state at any time as a function of the initial state, the input, and the model parameters. Because of the generality of closed-form solutions, when models readily can be solved analytically, this is always the preferred approach.

Simulation is used widely instead of analytical approaches because closed-form solutions for nonlinear, time-varying, and discrete-event systems are rarely available. In addition, while explicit closed-form solutions for time-invariant linear systems can always be found, this is sometimes impractical if the systems are very large. Moreover, simulation models can incorporate necessary procedural information that describes the process through which the state changes over time—information that often cannot be expressed in terms of equations alone. While simulation suffers all of the disadvantages of experimentalism, it is highly versatile and frequently the only practical means of analyzing complex models.

CONTINUOUS SYSTEM SIMULATION

Digital *continuous-system simulation* involves the approximate solution of an analog state-variable model over successive time steps. Consider the general state-variable equation [Eq. (1)] to be simulated over the time interval $t_0 \le t \le t_K$. The solution to this problem is based on the repeated solution of the single-variable, single-step subproblem. The subproblem may be stated formally as follows:

Given:

1. $\Delta t(k) = t_k - t_{k-1}$, the length of the kth time step
2. $dx_i/dt = f_i[\boldsymbol{x}(t), \boldsymbol{u}(t), t]$ for $t_0 \le t \le t_k$, the ith equation of state defined for the state variable $x_i(t)$ over the kth time step.
3. $\boldsymbol{u}(t)$ for $t_0 \le t \le t_k$, the input vector defined for the kth time step.
4. $\tilde{\boldsymbol{x}}(k-1) \cong \boldsymbol{x}(t_{k-1})$, an initial approximation for the state vector at the beginning of the time step.

Find:

5. $\tilde{x}_i(k) \cong x_i(t_k)$, a final approximation for the state variable $x_i(t)$ at the end of the kth time step.

Solving this single-variable, single-step subproblem for each of the state variables $x_i(t_k), i = 1, 2, \ldots, n$, yields a final approximation for the state vector $\tilde{\boldsymbol{x}}(k) \cong \boldsymbol{x}(t_k)$ at the end of the kth time step. Solving the complete single-step problem K times over K time steps, beginning with the initial condition $\tilde{\boldsymbol{x}}(0) \cong \boldsymbol{x}(t_0)$ and using the final value of $\tilde{\boldsymbol{x}}(k)$ from the kth time step as the initial value of the state for the $(k+1)$st time step, yields a discrete succession of approximations $\tilde{\boldsymbol{x}}(1) \cong \boldsymbol{x}(t_1), \tilde{\boldsymbol{x}}(2) \cong \boldsymbol{x}(t_2), \ldots, \tilde{\boldsymbol{x}}(K) \cong \boldsymbol{x}(t_K)$ spanning the solution time interval.

The basic procedure for completing the single-variable, single-step problem is the same regardless of the particular integration method chosen. The procedure consists of two parts:

1. Calculation of the average value of the ith derivative over the time step as

$$\frac{dx_i}{dt} = f_i[\boldsymbol{x}(t^*), \boldsymbol{u}(t^*), t^*] = \frac{\Delta x_i(k)}{\Delta t(k)} \cong \tilde{f}_i(k)$$

2. Calculation of the final value of the simulated variable at the end of the time step as

$$\tilde{x}_i(k) = \tilde{x}_i(k-1) + \Delta x_i(k) \cong \tilde{x}_i(k-1) + \Delta \tilde{f}_i(k)$$

If the function f_i is continuous, then t^* is guaranteed to be on the time step; that is, $t_0 \le t^* \le t_k$. Since the value of t^* is otherwise unknown, however, the value of the derivative can only be approximated as $\tilde{f}_i(k)$.

Different *numerical integration* methods are distinguished by the means used to calculate the approximation $\tilde{f}_i(k)$. A wide variety of such methods is available for digital simulation of dynamic systems. The choice of a particular method depends on the nature of the model being simulated, the accuracy required in the simulated data, and the computing effort available for the simulation study. Several popular classes of integration methods are outlined in the following subsections.

Euler Method

The simplest procedure for numerical integration is the *Euler* or *rectangular method.* As illustrated in Fig. 5, the standard Euler method approximates the average value of the ith derivative over the kth time step using the derivative evaluated at the beginning of the time step; that is,

$$\tilde{f}_i(k) = f_i[\tilde{\boldsymbol{x}}(k-1), \tilde{\boldsymbol{u}}(k-1), k-1]$$

A modification of this method uses the newly calculated state variables in the derivative calculation as these new

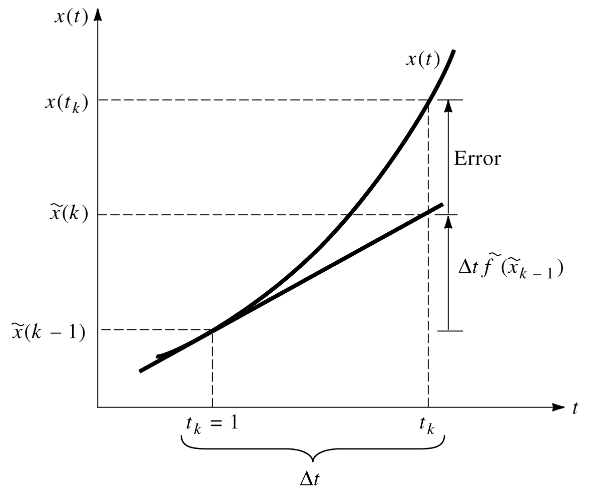

Figure 5. Geometric interpretation of the Euler method for numerical integration.

values become available. Assuming that the state variables are computed in numerical order according to the subscripts, this implies

$$\tilde{f}_i(k) \;=\; f_i[\{x_1(k), \ldots, x_{i-1}(k), x_i(k-1), \ldots, x_n(k-1)\}^T, \tilde{\boldsymbol{u}}(k-1), k-1]$$

The modified Euler method is modestly more efficient than the standard procedure and, frequently, is more accurate. In addition, since the input vector $\boldsymbol{u}(t)$ is usually known for the entire time step, using an average value of the input such as

$$\boldsymbol{u}(k) = \frac{1}{\Delta t(k)} \int_{t_{k-1}}^{t_k} \boldsymbol{u}(\tau) d\tau$$

frequently leads to a superior approximation of $\tilde{f}_i(k)$.

The Euler method requires the least amount of computational effort per time step of any numerical integration scheme. Local truncation error is proportional to Δt^2, however, which means that the error within each time step is highly sensitive to step size. Because the accuracy of the method demands very small time steps, the number of time steps required to implement the method successfully can be large relative to other methods. This can imply a large computational overhead and can lead to inaccuracies through the accumulation of roundoff error at each step.

Runge–Kutta Methods

Runge–Kutta methods precompute two or more values of the derivative in the time step $t_0 \le t \le t_k$ and use some weighted average of these values to calculate $\tilde{f}_i(k)$. The *order* of a Runge–Kutta method refers to the number of derivative terms (or *derivative calls*) used in the scalar single-step calculation. A Runge–Kutta routine of order N therefore uses the approximation

$$\tilde{f}_i(k) = \sum_{j=1}^{N} w_i f_{ij}(k)$$

where the N approximations to the derivative are

$$f_{i1}(k) = f_i[\tilde{\boldsymbol{x}}(k-1), \tilde{\boldsymbol{u}}(k-1), k-1]$$

(the Euler approximation) and

$$f_{ij} = f_i \left[\tilde{\boldsymbol{x}}(k-1) + \Delta t \sum_{t=1}^{j-1} \boldsymbol{I} b_{jt} f_{it}, u \left(t_{k-1} + \Delta t) \sum_{t=1}^{j-1} b_{jt} \right) \right]$$

where $\boldsymbol{I}$ is the identity matrix. The weighting coefficients w_i and b_{jt} are not unique, but are selected such that the error in the approximation is zero when $x_i(t)$ is some specified Nth-degree polynomial in t.

Because Runge–Kutta formulas are designed to be exact for a polynomial of order N, local truncation error is of the order Δt^{N+1}. This considerable improvement over the Euler method means that comparable accuracy can be achieved for larger step sizes. The penalty is that N derivative calls are required for each scalar evaluation within each time step.

Euler and Runge–Kutta methods are examples of *single-step methods* for numerical integration, so-called because the approximate state $\tilde{\boldsymbol{x}}(k)$ is calculated from knowledge of the approximate state $\tilde{\boldsymbol{x}}(k-1)$ without requiring knowledge of the state at any time prior to the beginning of the current time step. These methods are also referred to as *self-starting methods,* since calculations may proceed from any known state.

Multistep Methods

Multistep methods differ from the single-step methods in that multistep methods use the stored values of two or more previously computed states and/or derivatives in order to compute the derivative approximation $\tilde{f}_i(k)$ for the current time step. The advantage of multistep methods over Runge–Kutta methods is that these require only one derivative call for each state variable at each time step for comparable accuracy. The disadvantage is that multistep methods are not self-starting, since calculations cannot proceed from the initial state alone. Multistep methods must be started, or restarted in the case of discontinuous derivatives, using a single-step method to calculate the first several steps.

The most popular of the multistep methods are the *Adams–Bashforth predictor methods* and the *Adams–Moulton corrector methods.* These methods use the derivative approximation

$$\tilde{f}_i(k) = \sum_{j=0}^{N} b_i f_i[\tilde{\boldsymbol{x}}(k-j), \boldsymbol{u}(k-j), k-j]$$

where the b_i are weighting coefficients. These coefficients are selected such that the error in the approximation is zero when $x_i(t)$ is a specified polynomial. Note that the predictor methods employ an *open* or *explicit rule,* since for these methods $b_0 = 0$ and a prior estimate of $\tilde{x}_i(k)$ is not required. The corrector methods used a *closed* or *implicit rule,* since for these methods $b_0 \neq 0$ and a prior estimate of $\tilde{x}_i(k)$ is required.

Predictor–Corrector Methods

Predictor–corrector methods use one of the multistep predictor equations to provide an initial estimate (or "prediction") of $x_i(t)$. This initial estimate is then used with one of the multistep corrector equations to provide a second and improved (or "corrected") estimate of $x_i(t)$ before proceeding to the next step. A popular choice is the four-point Adams–Bashforth predictor together with the four-point Adams–Moulton corrector, resulting in a prediction of

$$\begin{aligned}\tilde{x}_i(k) &= \tilde{x}_i(k-1) + \frac{\Delta t}{24}[55\tilde{f}_i(k-1) - 59\tilde{f}_i(k-2)\\ &\quad + 37\tilde{f}_i(k-3) - 9\tilde{f}_i(k-4)]\end{aligned}$$

(for $i = 1, 2, \ldots, n$) and a correction of

$$\begin{aligned}\tilde{x}_i(k) &= \tilde{x}_i(k-1) + \frac{\Delta t}{24}[9\tilde{f}_i[\tilde{\mathbf{x}}(k), \mathbf{u}(k), k]\\ &\quad + 19\tilde{f}_i(k-2) - 5\tilde{f}_i(k-3) + \tilde{f}_i(k-4)]\end{aligned}$$

Predictor–corrector methods generally incorporate a strategy for increasing or decreasing the size of the time step depending on the difference between the predicted and corrected $x(k)$ values. Such *variable time-step methods* are particularly useful if the simulated system possesses local time constants that differ by several orders of magnitude, or if there is little a priori knowledge about the system response.

Numerical Integration Errors

An inherent characteristic of digital simulation is that the discrete data points generated by the simulation $\tilde{x}_i(k)$ are only approximations to the exact solution $x_i(t)$ at the corresponding point in time. This results from two types of errors that are unavoidable in the numerical solutions.

Round-off errors occur because numbers stored in a digital computer have finite word length (i.e., a finite number of bits per word) and therefore limited precision. Because the results of calculations cannot be stored exactly, round-off error tends to increase with the number of calculations performed. Therefore, for a given total solution interval, round-off error tends to increase (i) with increasing integration-rule order (since more calculations must be performed at each time step) and (ii) with decreasing step size Δt (since more time steps are required).

Truncation errors or *numerical approximation errors* occur because of the inherent limitations in the numerical integration methods themselves. Such errors would arise even if the digital computer had infinite precision. *Local* or *per-step truncation error* is defined as

$$\mathbf{e}(k) = \mathbf{x}(k) - \mathbf{x}(t_k)$$

given that $\mathbf{x}(k-1) - \mathbf{x}(t_{k-1})$ and that the calculation at the kth time step is infinitely precise. For many integration methods, local truncation errors can be approximated at each step. *Global* or *total truncation error* is defined as

$$\mathbf{e}(K) = \mathbf{x}(K) - \mathbf{x}(t_K)$$

given that $\mathbf{x}(0) - \mathbf{x}(t_0)$ and the calculations for all K time steps are infinitely precise. Global truncation error usually cannot be estimated, nor can efforts to reduce local truncation errors be guaranteed to yield acceptable global errors. In general, however, truncation errors can be decreased by using more sophisticated integration methods and by decreasing the step size Δt.

Time Constants and Time Steps

As a general rule, the step size Δt for simulation must be less than the smallest local time constant of the model simulated. This can be illustrated by considering the simple first-order system

$$\frac{dx}{dt} = \lambda x(t)$$

and the difference equation defining the corresponding Euler integration:

$$\begin{aligned}x(k+1) &= x(k) + \Delta t \lambda x(k)\\ &= (1 + \Delta t \lambda)x(k)\end{aligned}$$

The continuous system is stable for $\lambda < 0$, while the discrete approximation is stable for $|1 + \lambda\ \Delta t| < 1$. Therefore, if the original system is stable, the simulated response will be stable only for

$$\Delta t \le |\lambda^{-1}|$$

where the equality defines the *critical step size.* For larger step sizes, the simulation will exhibit *numerical instability.* In general, while higher-order integration methods will provide greater per-step accuracy, the critical step size itself will not be greatly reduced.

A major problem arises when the simulated model has one or more time constants $|\lambda^{-1}|$ that are small when compared to the total solution time interval $t_0 \le t \le t_k$. Numerical stability will then require very small Δt, even though the transient response associated with these higher-frequency subsystems may contribute little to the particular solution. Such problems can be addressed either by neglecting the higher-frequency components where appropriate or by adopting special numerical integration methods for *stiff systems.*

Selecting an Integration Method

The best numerical integration method for a specific simulation is the method that yields an acceptable global approximation error with the minimum amount of round-off error and computing effort. No single method is best for all applications. The selection of an integration method depends on the model simulated, the purpose of the simulation study, and the availability of computing hardware and software.

In general, for well-behaved problems with continuous derivatives and no stiffness, a lower-order Adams predictor is often a good choice. Multistep methods also facilitate estimating local truncation error. Multistep methods should be avoided for systems with discontinuities, however, because of the need for frequent restarts. Runge–Kutta methods have the advantage that these are self-starting and provide fair stability. For stiff systems where high-frequency modes have little influence on the global response, special stiff-system methods enable the use of economically large step sizes. Variable-step rules are useful when little is known *a priori* about solutions. Variable-step rules often make a good choice as general-purpose integration methods.

Round-off error usually is not a major concern in the selection of an integration method, since the goal of minimizing computing effort typically obviates attendant problems. Double-precision simulation can be used where round-off is a potential concern. An upper bound on step size often exists because of discontinuities in derivative functions, or because of the need for response output at closely spaced time intervals.

Digital simulation can be implemented for a specific model in any high-level language such as FORTRAN or C. In addition, many special-purpose continuous system simulation languages are commonly widely available across a wide range of platforms. Such languages greatly simplify programming tasks and typically provide friendly user interfaces and good graphical output.

DISCRETE-EVENT SIMULATION

In *discrete-event* dynamic models, the independent variable is a discrete index which references a set of events. The events are instantaneous, strictly sequential, and asynchronous, as previously defined. The occurrence of events is determined by inputs and by the internal logic of the system. The system state changes in response to the occurrence of events.

For example, consider a simple *queuing system* comprising a server (called a *permanent entity*) and a set of jobs (called *temporary entities*) to be processed by the server. If the server is idle when a new job arrives for processing, then the job typically begins processing immediately. However, if the server is busy when a new job arrives, then the arriving job typically must wait for some or all of the prior jobs to complete processing before the new job can access the server and begin processing.

The state of the queuing system is the total number of jobs in the system $x(t)$ at any time t, including any job in process and all jobs waiting. Outputs are performance measures derived by knowing the trajectory of the state over time, including such measures as average cycle time for jobs, the average queue length and waiting time, and the average throughput for the system. Clearly, the state of the system changes in response to two types of events. An *arrival event* increases the number in system by one job. A *departure event* (or *service completion event*) decreases the number in system by one job.

Random Number Generation

The discrete-event systems of greatest interest are stochastic, with events occurring randomly over time as constrained by the process logic. Inputs to a simulation are random numbers, and the theory of discrete-event simulation is intimately bound to the rich and evolving theory of generating *pseudorandom numbers* and *variates* on a computer. (Pseudorandom numbers are real values generated deterministically by a computer algorithm. While completely deterministic, these numbers satisfy standard tests for statistical independence and randomness). In the queuing example, arrival events typically are determined by generating random variates that define the time between the arrivals of successive jobs. Departure events are determined in part by generating random numbers which define the time required to complete each job, once the job begins processing.

Time Advance Mechanism

In contrast to continuous systems simulations, which use some form of *fixed-increment time advance* to determine the next value of the independent variable after each iteration, almost all discrete-event simulations employ a *next-event time-advance* approach. With the next-event approach, at each time step the state of the system is updated to account for the fact that an event has occurred. Then the times of occurrence of future events are determined. The value of simulated time is then advanced to the time of occurrence of the first or *most imminent* of these future events. This process advances the simulation time from one event time to the next and is continued until some prescribed stopping event occurs. The next-event approach clearly is more efficient than the fixed increment approach, since computation time is not wasted during periods of inactivity between events when by definition the state cannot change.

Logical Components

The next-event time-advance approach relies on searching and manipulating data structures that are *lists* or *chains* of current and future events. At each time step, the current event list or *calendar* is scanned to determine the next event to be processed. The processing of an event typically involves linking and unlinking existing and/or newly created events to the lists. The logical components shared by most discrete-event simulations using the next-event time-advance approach implement these list processing, event processing, accounting, and reporting requirements. These components include the following (3):

System Image or State. The collection of state variables necessary to describe the system at a particular time.

Simulation Clock. A variable giving the current value of simulated time.

Event List or Calendar. A list containing the next time when each type of event will occur.

Statistical Counters. Variables used for storing statistical information about system performance.

Initialization Routine. A subprogram to initialize the simulation model at time zero.

Timing Routine. A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur.

Event Routines. Subprograms that update the system state when a particular type of event occurs (there is one event routine for each event type).

Library Routines. A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model.

Report Generator. A subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends.

Main Program. A subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.

MODELING AND SIMULATION ISSUES

Ziegler (4) proposed a formal theory of modeling and simulation that builds on the ideas of mathematical systems models. Ziegler's theory encompasses five basic elements:

1. *Real Model.* The system modeled. It is simply a source of observable data, in the form of input–output pairs $(\boldsymbol{u}_j, \boldsymbol{y}_j)$. Typically, there are no other clues available to determine its structure.
2. *Base Model.* The investigator's image or mental model of the real system. It is a system that is capable (at least hypothetically) of accounting for the complete behavior of the real system. If the real system is highly complex, the base model also is highly complex. The cost, time, and difficulty of realizing the base model explicitly most often are prohibitive, unwarranted, or impossible. Therefore, as a practical matter, the structure of the base model is, at best, partially known to the investigator.
3. *Experiment Frame.* The set of limited circumstances under which the real system is to be observed and understood for the purpose of the modeling exercise. It is a restricted subset of the observed output behaviors.
4. *Lumped Model.* The concept most often associated with the term model. It is a system which is capable of accounting for the output behavior of the real system, under the experiment frame of interest. It is an explicit simplification and partial realization of the base model, with its structure completely known to the investigator.
5. *Computer.* The means by which the behavior of the lumped model is generated. The computer, in the sense intended by Ziegler, is not necessarily a digital computer. For simple models, it may represent the explicit analytical solution to the model equations, worked out by hand. For more complex models, however, the computer may need to generate individual trajectories step by step, based on instructions provided by the model. This step-by-step process is what is most often associated with the concept of simulation and is usually conducted by using a digital computer.

Model theory illuminates the fundamental relationships among these modeling elements. Three of the most important modeling relationships are briefly described below.

Validation concerns the relationship between models and the real system. The objective of validation is to ensure that a model matches the system modeled, so that the conclusions drawn about the model are reasonable conclusions about the real system as well. A base model is valid to the extent that it faithfully reproduces the behavior of the real system in all experiment frames. On the other hand, a lumped model is valid to the extent that it faithfully matches the real system under the experimental frame for which it is defined. There can be many different lumped models that are valid, and a lumped model can be valid in one experiment frame and not another. Model validation is a deep and difficult issue, and there are many different levels and interpretations of validity.

Simplification concerns the relationships between a base model and its associated lumped models. The objective of simplification is to achieve the most efficient and effective lumped model that is valid within the experiment frame for which it is defined. Simplification can be achieved in many ways. These include dropping relatively insignificant system variables and associated structures, replacing deterministic variables and structures with random variables and associated generating functions, coarsening the range set of system variables, and aggregating system variables and structures into larger blocks. Many of the formal ideas associated with simplification, such as isomorphism and homomorphism, concern the preservation of structural similarities between mathematical systems.

Simulation, in the sense intended by Ziegler, concerns the relationship between models and the computer. The objective of simulation is to ensure that the computer faithfully reproduces the behavior implied by the model. The behavior of a lumped model must be distinguished from the correctness of its computer implementations or solutions, in the same way that the behavior of a real system must be distinguished from the validity of its models. While a valid model may have been developed, it is also necessary to have a correct simulation. Otherwise, the model solution cannot be used to draw conclusions about the real system. Formal ideas associated with simulation include the completeness, consistency, and ambiguity of the computer implementation. The process of matching a simulation to its lumped model sometimes is known as *verification.* Many of the same techniques used to validate models are also used to verify simulations.

MODEL VALIDATION

One of the most important and most difficult issues in modeling is that of establishing the level of credibility that should be given to a model and, as a consequence, the level of confidence a decision maker should have in conclusions derived from model results. As introduced above, validation is the process of determining whether or not a model is adequate for the specific tasks to which it will be applied. Validation tests the agreement between the behavior of the model and the behavior of the real-world system which is being modeled (5). Validation is the process of bringing to an acceptable level the user's confidence that any inference about the real-world system derived from the model is correct (6). Validation is not a general seal of approval, but is instead an indication of a level of confidence in the model's behavior under limited conditions and for specific purposes—that is, a check on its operational agreement with the real-world system (7).

Verification, Validation, Problem Analysis, and Model Assessment

A number of concepts have evolved in the literature on model validation which provide useful distinctions between the various related activities involved in evaluating a model for use in support of decision-making. Fishman and Kiviat (5) introduced the now standard division of evaluation activities into three categories: *verification,* the process of ensuring that a model behaves as the modeler intends it should behave; *validation,* the process of demonstrating agreement between the behavior of a model and the behavior of the system the model is intended to represent; and *problem analysis,* the process of interpreting the data and results generated by application of a model. More recently, a number of investigators (7–10) have extended the domain of evaluation activities to include the more general and dynamic notion of *model assessment.* Model assessment includes not only the activities of verification, validation, and problem analysis, but also *model maintenance and quality control,* to ensure the continued usability of the model and its readiness for use, and *model understanding,* to determine the assumptions and limitations of the model, the appropriate and inappropriate uses of the model, and the reasons a model generates the results that it does.

Verification is concerned with the internal consistency of a model and the degree to which the model embodies the intent of the modeler. A model that is fully verified is not necessarily an accurate representation of the system modeled, but is instead a precise interpretation of the modeler's description of the system as he or she intends to represent it. Verification involves tests to ensure that model equations and logic are accurately stated, that the logic and order of model computations are accurately carried out, and that the data and input to the model are correctly interpreted and applied during computations.

For computer-based models, verification is closely associated with computer programming and with the techniques of software engineering used to develop readable, reliable computer programs. The techniques of structured programming, in general, are invaluable in the verification of large and complex models. In addition, Law (3) describes several techniques for verification that are perhaps unique to computer simulation modeling.

Validation is concerned with the accuracy of a model as a representation of the system modeled. How to measure the validity of a model is problematic, however, and certainly depends upon the intended use of the model. Indeed, no model can ever be entirely valid in the sense of being supported by objective truth, since models are, by nature and design, simplifications of the systems they are intended to represent. As Greenberger et al. (7) suggest, "useful," "illuminating," "convincing," and "inspiring confidence" are more reasonable descriptors of models than is "valid." Validation issues, tests, and philosophy are the central concerns of this article and are considered in greater detail in the sections which follow.

Problem analysis, or output analysis, is concerned with determining the true parameters of a model and with correctly interpreting data generated by solving the model. As with model verification, problem analysis is largely a model-based activity that says nothing directly about the true parameters or characteristic behavior of the system modeled. Inferences about the behavior of the real system apply only to the extent that the model is valid for the system under study and for the particular analysis applied. Output analysis is a specialized technical subject.

Assessment is concerned with determining whether or not a model can be used with confidence for a particular decision problem within a particular decision-making environment. The basic idea behind model assessment, as distinct from the more limited idea of model validation *per se,* is the accumulation of evidence by independent and dispassionate investigators regarding the credibility and applicability of a model. Model assessment serves many purposes: education, model, development and improvement, theoretical analysis, understanding the model and the process being studied, obtaining insights to aid decision-making, ensuring the reproducibility of results, improving model documentation, making the model more accessible, and determining the utility of the model for a particular decision-making situation. Model assessment is typically discussed in the context of models developed (and intended to be institutionalized) for policy analysis, but the evaluation procedure applies equally well to models which are to be used regularly within any decision-making environment. Assessment is important because the decision maker typically has had little or no involvement in the modeling process and therefore requires an independent basis for deciding when to accept and when to reject model results. This basis is difficult to develop without independent evaluation of the impact on model structure and behavior of the assumptions of the model, the availability of data used to calibrate the model, and the other elements of the process implicit in model development.

Direct and Indirect Approaches to Model Validation

Validation represents a collection of activities aimed at deducing just how well a model captures those behavioral characteristics of the system modeled which are essential for understanding a given decision-making situation. If a good deal is known about the past behavior of the system modeled, or if the system modeled is accessible to study or at least some limited experimentation, then comparison of the behavior of the model with the behavior of the modeled system provides a direct means for model validation.

A model is said to be *replicatively valid* if the data generated by the model correlate (within tolerances established by the investigator) with data collected from the real system, where the data from the real system have been collected prior to development and calibration of the model. A model is said to be *predictively valid* if data generated by the model correlate with data collected from the real system, where the data from the real system have been collected after the model has been developed and run. Of the two validity tests, predictive validity is the stronger. Replicative validity is typically an objective of the modeling exercise, rather than a test of model validity per se since every modeler almost certainly endeavors to modify and refine a model until its conformity with known behavioral data is achieved.

A model is said to be *structurally valid* if it not only replicates the data generated by the real system, but does so for the same reasons and according to the same causal mechanisms as the real system. A model which can be shown to be both predictively and structurally valid obviously is highly desirable.

In a great many situations, direct approaches to model validation are not available. These are common in future-oriented decision situations in volatile environments, where the potential risks and rewards are greatest. In these situations the real-world system modeled is not well understood, or does not yet actually exist, or is otherwise inaccessible to study or experimentation. The past behavior of the real-world system is either unknown, or provides a poor guide to the likely future behavior of that system. The future behavior of the real-world system cannot be known a priori with certainty. In these cases, the modeler or decision maker must rely upon indirect tests of a model's credibility.

Face validity is the primary objective of the first phase of model development. A model with face validity is one which appears to be reasonable (or does not appear to be unreasonable) to people who are knowledgeable about the system under study. In general, even the most complex models are constructed from simpler primitives. Complexity results from the large number of hypotheses used in constructing the model and from the myriad interactions that can occur between individual model components, rather than from any inherent complexity in the hypotheses as such. Face validity ensures that the individual relationships and hypotheses built into a model are consistent with what is understood or assumed to be true about the real-world system by experts. Face validity also ensures that any relationship that can be rejected based upon prior knowledge and experience will not be incorporated within the model.

Face validity is achieved by using all existing information about the system modeled during model development and initial testing. Information sources include observation of the system elements or subsystems, existing theory, conversations with experts, general knowledge, and even the intuition of the modeler. Early involvement of the ultimate user or decision maker in the initial formulation of a model, if possible, along with continued close involvement of the user as the model develops, tends to promote "ownership" of the model on the part of the user. This is the most highly desired form of face validity, since decision makers are far more likely to accept as valid and to use models which they intimately understand and which they have actively helped to develop.

Sensitivity analysis, or *variable-parameter validity,* is a set of quantitative procedures for testing the validity or credibility of assumptions that were made during the initial stages of model development and that have survived the test of face validity. Sensitivity analysis seeks to assess the amount of change in the model state, output, or other critical variables that result from changes in selected model parameters or inputs. One use of sensitivity analysis is to test the sense and magnitude of the impact of one variable upon another, in order to ensure that the changes induced are intuitive (or, if counterintuitive, to determine the reasons for such changes from the underlying causal structure of the model).

A second use of sensitivity analysis is to isolate pairs of inputs and outputs where small changes in the input result in large changes in the corresponding output. Since the model is particularly sensitive to the relationships that couple these pairs, sensitivity analysis can be used in this way to determine those assumptions and hypotheses upon which the model most critically depends. Relationships to which the model is highly sensitive can be singled out for further critical evaluation. In a similar fashion, sensitivity analysis can be used to determine relationships to which the model is insensitive. This introduces the possibility of simplifying the model by reducing the level of detail with which insensitive subsystems are represented.

Sensitivity analysis is one way to compensate for uncertainty in a model, but it typically is a difficult, technically demanding, and potentially expensive and time-consuming activity. *Monte Carlo* techniques can be used to explore formally the distribution of model outcomes resulting from the distributions of uncertainty in model parameters and inputs. Other advanced techniques of statistical sensitivity analysis, such as response surface methods, can also be used to great advantage, when time and budget permit and when the importance of the decision consequences demands.

A Framework for Validation Activities

Schellenberger (11) developed a general validation framework which is particularly useful in organizing related validation tasks and objectives. This three-part framework includes the ideas of technical validity, operational validity, and dynamic validity.

Technical validity concerns the assumptions and data used to formulate a model and is the aggregate of model validity, data validity, logical/mathematical validity, and predictive validity. Model validity conforms to the standard definition of validity and seeks to determine the degree to which the model is an accurate representation of the system modeled. The activities of model validation include identifying and criticizing all the assumptions of a model (stated and implied), such as content assumptions concerning the scope and definition of variables, causal assumptions concerning the nature and extent of cross impacts among variables, and mathematical assumptions concerning the exact form and continuity of model relationships. *Data validity* addresses the adequacy of raw and processed data. The activities of data validation include determining the accuracy, impartiality, and representativeness of raw data, as well as the effects and potential biases introduced in reducing raw data to the structured form actually used in the model. *Logical/mathematical validity* conforms to the standard definition of model verification and seeks to determine errors in translating the model into an accurate computer code. The activities of logical/mathematical validation include determining whether computations are accurate and precise, whether the flows of data and intermediate calculations are correct, and whether all of the necessary variables and relationships have been included within the computer program. *Predictive validity* addresses the correlation between behavioral data generated by the model and the corresponding data from the system modeled. The activities of predictive validation include statistical tests, analyses of time series data, and even intuitive comparisons of behavioral trends.

Operational validity assesses the meaning and importance of technical errors in a model and focuses on differences between the model and the system modeled as determined through technical validation. The fundamental question here is one of degree. Are the differences between the model and the real world sufficient to draw into question the results of the model? Are the insights gained from the model sufficient to overcome concerns about inaccuracies in the specific numerical data generated by the model? Is the model sufficiently robust to yield consistent conclusions for reasonable ranges of parameter variations? Clearly, sensitivity analysis is one means of exploring these questions and, for this reason, is a key element in determining operational validity. Also included under the heading of operational validity activities is the notion of *implementation validity*. The idea here is to determine the extent to which recommended actions, derived from a model-based study, will have the intended effect, when implemented in the real-world system.

Dynamic validity concerns maintaining a model over time in order to extend the usefulness of the model for decision-making. Dynamic validation requires both updating and review. Updating refers to the process through which the need for incremental changes in the model database or model structure is identified and through which the necessary changes are implemented. On a broader scale, review refers to the process through which the success or failure of the model is regularly gauged and through which major changes in, or revalidation of, the model is triggered.

Philosophical Perspectives

The subject of model validation cannot be divorced from the broader philosophical issue of how, in general, we may know the truth. Epistemology is the branch of philosophy which is concerned with the origins, nature, methods, and limits of knowledge; it is also concerned with what we can know, how we can know it, and how much faith we can have in the validity of our knowledge. One epistemological theory (and most likely the predominant theory among scientists, engineers, and those others who build and use mathematical models) is that all human knowledge that is strictly of a rational nature is fundamentally, inescapably model-based. Our knowledge of any aspect of the real world in this view constitutes an internalized "mental model" of that world, namely, Ziegler's base model.

Because our organized knowledge of the real world is internalized in mental models, such knowledge must be based upon perceptual information passed through our senses. We develop and refine our mental models according to primary sensory information (data), secondary information given us by others (data, theory, and opinion), and reason and logic (a form of verification for mental models that itself is based upon internalized models). We know that our senses are selective and can be deceived, and therefore we can never have guarantees that our mental models are based upon information that is either complete or entirely correct. We know that information from secondary sources is not immutable, and therefore we cannot rely upon authority for absolute substantiation of our mental models. We know that any logical construct must begin with some fundamental predicate assumption that cannot be tested, and therefore an appeal to pure reason is insufficient to validate our mental models.

In summary, it is unlikely that we can know anything with absolute certainty. All knowledge is to a greater or lesser degree personal and subjective, and it is tentative and subject to future revision or rejection (12). This is not to say that necessarily there are no truths to be known, but simply to recognize that our means of knowing these (if these exist) are inherently imperfect. It is within this context that the issue of model validation must be viewed. We can and should test and refine models in an effort to develop a high degree of confidence in their usefulness, but validation in an absolute sense is most likely a quest which belies the fundamental limits of human understanding. Validation in an absolute sense is neither possible nor necessary.

Validation Checklist

By far the most important test for the validity of a model rests with the question, "Does it make sense?" If the results of a modeling exercise defy the intuition of those most intimately familiar with the real-world system, then it is unlikely that any amount of explanation will ever persuade. By the same token, if the results are reasonable—if these conform to prior experiences and, best of all, offer insight to match our intuitions—then the issue of validation will undoubtedly be resolved favorably. In summary, validation is a continuous process by which confidence and credibility are developed for a model. Shannon (6) leaves us with the following checklist of actions that will ensure that the greatest possible validity is achieved:

- Use common sense and logic.
- Take maximum advantage of the knowledge and insight of those most familiar with the system under study.
- Test empirically all of the assumptions of the model, whenever possible, using the appropriate statistical techniques.
- Pay close attention to details, and check and recheck each step of the model building process.
- Use test data and all available means during debugging to ensure that the model behaves as intended.
- Compare the input–output transformation of the model with that of the real system, whenever possible, using the appropriate statistical tests.
- Run field tests and conduct peripheral research where possible.

- Undertake sensitivity analyses of model inputs and parameters.
- Check carefully the predictions of the model and the actual results achieved in the real-world system.

BIBLIOGRAPHY

1. K. P. White, Jr. Model theory, in *Encyclopedia of Science and Technology*, New York: McGraw-Hill, 2002.
2. G. E. P. Box and N. R. Draper, *Response Surfaces, Mixtures, and Ridge Analyses*, 2nd ed., New York: Wiley, 2007.
3. A. M. Law, *Simulation Modeling and Analysis*, 4th ed., New York: McGraw-Hill, 2007.
4. B. P. Ziegler *Theory of Modelling and Simulation*, New York: Wiley, 1976.
5. G. S. Fishman P. J. Kiviat The statistics of discrete-event simulation, *Simulation*, **10**: 185–195, 1968.
6. R. E. Shannon *Systems Simulation: The Art and Science*, Englewood Cliffs, NJ: Prentice-Hall, 1975.
7. M. Greenberger M. A. Crenson B. L. Crissey *Models in the Policy Process: Public Decision Making in the Computer Era*, New York: Russell Sage Foundation, 1976.
8. S. I. Gass Evaluation of complex models, *Comput. Oper. Res.*, **4**: 27–35, 1977.
9. S I. Gass Decision-aiding models: Validation, assessment, and related issues for policy analysis, *Oper. Res.*, **31**, 603–631, 1983.
10. *US Government Accounting Office*,1979 *Guidelines for Model Evaluation*, Report No. PAD-79-17, Washington, DC: US Government Accounting Office, 1979.
11. R. E. Schellenberger Criteria for assessing model validity for managerial purposes, *Decis. Sci.*, **5**: 644–653, 1974.
12. M. Polanyi *Personal Knowledge: Toward a Post-Critical Philosophy*, New York: Harper and Row, 1958.

Reading List

J. Banks, B. L. Nelson, J. S. Carson, D. M. Nicol, *Discrete-Event System Simulation*, Englewood Cliffs, NJ: Prentice Hall, 2004.

J. Banks R. R. Gibson Selecting simulation software, *IIE Solutions*, **29** (5): 30–32, 1997.

B. S. Bennett *Simulation Fundamentals*, Upper Saddle River, NJ: Prentice-Hall, 1996.

G. Gordon *System Simulation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1978.

C. Harrell K. Tumay *Simulation Made Easy: A Manager's Guide*, Norcross, GA: IIE Press, 1995.

W. D. Kelton, R. P. Sadowski, D. T. Sturrock, *Simulation with Arena*, 4th ed., New York: McGraw-Hill, 2007.

N. A. Kheir (ed.) *Systems Modeling and Computer Simulation*, 2nd ed., New York: Marcel Dekker, 1996.

D. G. Luenberger *Introduction to Dynamic Systems: Theory, Models, and Applications*, New York: Wiley, 1978.

R. H. Myers and D. C. Montgomery, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 2nd ed., New York: Wiley, 2002.

A. P. Sage C. C. White *Optimum Systems Control*, 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1977.

T. J. Schribner D. T. Brunner Inside simulation software: How it works and why it matters, *Proc. 2006 Winter Simulation Conf.*, 2006.

W. Thissen Investigations into the World 3 model: Lessons for understanding complicated models, *IEEE Trans. Syst., Man Cybern.*, **8**: 183–193, 1978.

K. P. White, Jr. Modeling and simulation, inM. Kutz (ed.), *Handbook of Mechanical Engineering*, 3rd ed., New York: Wiley, 2005.

K. PRESTON WHITE Jr.
Professor of Systems and Information Engineering,
University of Virginia,
Charlottesville, VA