Modeling & Simulation of Dynamic Systems – a Tutorial[1]

James H. Taylor
Department of Electrical & Computer Engineering
University of New Brunswick
Fredericton, New Brunswick CANADA E3B 5A3
email: jtaylor@unb.ca web-site: www.ece.unb.ca/jtaylor/

Abstract: A brief overview of modeling and simulation (M & S) technology is presented, describing the development of M & S in a variety of industrial contexts, such as aerospace, manufacturing, and process industries. Typically this occurs in phases: introduction of M & S, shaping M & S to meet an industry's needs, and maturation. It is demonstrated that in industry after industry M & S has become increasingly sophisticated, increasingly utilized, and increasingly important – to the point that in industries where M & S is mature it is one of the keys to product development, product improvement, and, in the end, increased competitive position and profit margin.

The technical side of M & S is also discussed, in detail. Important points include the selection of modeling and simulation methods for various application areas, and the state-of-the-art in simulating dynamic systems. Emphasis is placed on rigorous techniques and selecting the most appropriate method for a given problem. The final goal is to aid in making M & S an effective tool for achieving the benefits mentioned above for agricultural and bio-industrial systems.

Key Words: Dynamic systems, modeling, simulation, numerical integration, discontinuities, hybrid systems.

1. Introduction

Computer-based M & S has been under steady development for about 50 years now. In one form or another, this methodology has become ever more important in an ever broadening arena of applications. Here we overview a part of this wide field, the modeling and simulation of dynamical systems, specifically, systems that can be appropriately modeled by ordinary differential equations (ODEs), partial differential equations (PDEs), differential algebraic equations (DAEs), and ODEs interfaced with discrete-time algorithms (DTAs). Some aspects of the technical discussion of modeling is, of necessity, be quite sketchy, and the more comprehensive presentation of simulation methods focusses primarily on ODE systems, with additional pointers to techniques for dealing with DAEs, PDEs, and such systems interfaced with DTAs.

Significant utilization of dynamic M & S has started at different times and progressed at different rates in various application areas. As a generalization, the early years of M & S were focussed primarily on aerospace and military applications – i.e., high-tech and expensive technological systems, where the engineering analysis and design

[1]Based on a paper presented at **M^{2}SABI'01 Conference**, Haifa, Israel, June 2001 titled "Modeling & Simulation of Agricultural and Bio-Industry Systems – What, Me Worry?".

effort was substantial and the M & S effort could be "afforded". Then M & S spread to smaller high-tech and civilian arenas – robotics and transportation systems coming immediately to mind. At the present, it is quite safe to state that M & S, in one form or another, is applied to almost every area of human endeavor, from agriculture to biology to economics to manufacturing ... to social systems to zoology – although the approaches, the degree of maturity and the pervasiveness varies greatly from area to area. Regarding "affordability", in many applications today, most users feel that they cannot afford **not** to perform M & S.

A scan of contemporary literature in the field reveals that the motivation and goals of M & S are essentially simple and universal: M & S is utilized to understand the behavior and to improve the behavior of dynamic systems. However, the specific focus and approach differs significantly from application to application. This is due to (1) the different types of models required for studying different types of systems, and (2) the different types of investigations that are undertaken. Therefore, in virtually every area, M & S has been refined, extended and customized to meet the specific needs of that discipline. By now, these contributions make up a corpus of techniques and tools/algorithms that is truly vast.

Finally, there are numerous driving forces for the rapidly increasing necessity and popularity of M & S. Five general, dominant factors are:

- the continuing need to achieve better process or product performance;
- the increasing complexity of advanced technological systems;
- the growing need for competitive advantage, e.g., efficiency, economy;
- the phenomenal increase in available computer power and decrease in cost; and
- the coupling of M & S with other powerful computer-based methods, e.g., optimization.

For a graphic and well-known illustration of these trends, we have only to look at the current standard practices in the automotive industry: The performance (not only power and handling, but emission control, safety, comfort, etc.) has increased greatly in the last 25 years, the complexity of the entire package (engine/powertrain/chassis/features) has increased at least 10-fold in that same time-frame (if you still do all your own car maintenance, please raise your hand), and the fuel economy *vs* performance ratio has (by mandate) also increased significantly. This evolution is due, to a considerable degree, to a hundred-fold increase in M & S, including large-scale optimization.

2. Modeling Overview

Modeling is not the primary focus of this presentation, so what follows is, of necessity, merely an informal review of basic ideas. This will provide the context for the discussion of simulation methods to follow. First, some terminology:

1. *"Hard" modeling* – using scientific principles (Newton's law, Kirchhov's laws, laws of thermodynamics, reaction kinetics, etc.) to derive an analytical model. The feasibility of doing so varies a great deal from one discipline to another. This may be said to be "easy" for electro-mechanical systems (e.g., robotics) and "very difficult or impossible" in some biological application areas. In fact, these lines are not entirely clear, since some biological phenomena can be modeled readily from first principles, and some effects in electro-mechanical systems cannot (friction being a notorious example). Of course, someone who has spent a year or more developing and validating a realistic, detailed model of a robot might disagree with the characterization "easy", but by that is meant only that the physical principles and approach are well established.

2. *"Soft" modeling* – using formal or informal fitting techniques to match a mathematical model's behavior to empirical data/observations. The Lotka-Volterra competition equations [11] for population dynamics are a well-known "classical" example of this process, and Forrester's world models [7] represent a much more ambitious – but controversial – illustration.

3. *Model identification* – A large number of methods and software packages exist for model identification, informally described as determining model structure (e.g., order) and parameters based on time-series data of an object's or process' input/output behavior. These include frequency response (nonparametric) modeling, regression, least squares techniques, maximum likelihood, instrumental variables – for a comprehensive coverage, see Ljung [12].

Developing a system model may well involve a combination of the above approaches. "Hard modeling" produces a so-called "white box" model, and a model based solely on determining the parameters of a model of assumed (nonphysical) structure is called a "black box" model. Obviously, a combination of hard and soft modeling or model identification produces a "grey box" model – this is done quite commonly.

The modeling process outlined above produces the set of ODEs, DAEs or PDEs that characterize the continuous-time part of the system (process) plus perhaps DTAs describing the behavior of digital components interacting with the process. Since DTAs can be faithfully emulated in a digital simulation (including effects due to word length, execution time etc.) in a relatively straightforward manner we focus primarily on handling the continuous-time part. After a few general comments regarding DAEs and PDEs, we further restrict our attention to ODEs.

First, we distinguish between ODEs and DAEs as follows: A generic form of a set of DAEs and output equations may be expressed as:

$$
\begin{aligned}
0 &= F_c(x_c, \dot{x_c}, u_c, u_k, t) &(1)\\
y_c(t) &= H_c(x_c, \dot{x_c}, u_c, u_k, t) &(2)
\end{aligned}
$$

where x_c is the state vector, y_c is the output vector, u_c and u_k are the input signals (continuous- and discrete-time, respectively – the latter due to interfaced DTAs, if any), and t is time; in general u_c and u_k are vectors. Note that there are implicit "zero-order holds" operating on the elements of u_k, i.e., these inputs remain constant between those times when they change instantaneously. We observe that the Jacobian $F_{\dot{x_c}} \triangleq \partial F_c / \partial \dot{x_c}$ is usually identically singular; otherwise the system in (1) can be treated as an ODE set [3].

The form in (1) is called a *fully implicit* DAE [3]; without imposing additional conditions or constraints, it generally cannot be solved by any existing numerical code. In fact, determining if such a model is solvable [3, 17] and arriving at consistent initial conditions [3, 16] is a complicated matter. To achieve a practical definition of the class of continuous-time models to be treated, we need to specialize the form in (1) in some manner:

- Most simply, we may replace the DAE form in Eqns. (1,2) with the following ordinary differential equation set:

  $$
  \begin{aligned}
  \dot{x_c}(t) &= f_c(x_c, u_c, u_k, t) &(3)\\
  y_c(t) &= h_c(x_c, u_c, u_k, t) &(4)
  \end{aligned}
  $$

  Such models have been the focus of most commercial modeling and simulation environments in the last few decades [1, 5, 19, 20].

- Next most simply, we may replace the form in (1) with the following *constrained* ODE set:

  $$
  \begin{aligned}
  \dot{x_c}(t) &= f_c(x_c, z_c, u_c, u_k, b_i, m_j, t) &(5)\\
  0 &= g_c(x_c, z_c, u_c, u_k, b_i, m_j, t) &(6)\\
  y_c(t) &= h_c(x_c, z_c, u_c, u_k, b_i, m_j, t) &(7)
  \end{aligned}
  $$

  where constraint variables z_c have been added along with constraint equations (6). Models of this form are called *semi-explicit* DAE*s*. It has been shown that ODE solvers can be used for simulating this simplest class of DAEs [9, 10]

It is important to note that the choice of continuous-time model form is not hard-and-fast; rather, it is a matter of judgement about what is required to attain suitable "realism". Very basic decisions regarding realism include not only the model type, but also what order of model is needed (how many state variables) and what nonlinearities need to be included. Such decisions may also effect solvability; for example, it is usually not a good practice to develop an ODE model that includes both very fast and very slow dynamics (called a "stiff" model [8]), since they are hard to simulate, with some solvers taking a lot of computer time, other solvers failing completely.

One way to avoid a stiff model is to convert the "very fast dynamics" into algebraic constraints, thereby converting stiff equations (3) into DAEs (5, 6). In some cases this may be trivial (e.g., neglecting the fast dynamics associated with a servomotor's inductive lag is simply done by setting the inductance to 0 and eliminating the corresponding current as a state variable) or easy – for example, if the states are separable into fast and slow states, $x_{fast}\,, x_{slow}$ then

$$\begin{aligned} x &= [\,x_{fast}\ x_{slow}\,]^T \\ \dot{x}_{fast} &= f_{fast}(x_{fast}, x_{slow}, t) \\ \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \end{aligned} \tag{8}$$

yields the DAE set

$$\begin{aligned} \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \\ 0 &= f_{fast}(x_{fast}, x_{slow}, t) \end{aligned} \tag{9}$$

Where such a direct state decomposition is not feasible, the conversion process is not as simple but still worthwhile and done routinely in some applications, such as modeling aircraft engines. Some stiff system integration algorithms perform such a decomposition automatically [8].

Another problem that gives rise to DAEs or that requires some sort of work-around is the existence of "algebraic loops". A simple example of this problem – and a common source of difficulty – is the standard control system where the open loop transfer function is not strictly proper (i.e., the numerator order is the same as the denominator order). Consider the open-loop transfer function $G(s) = (s+1)/(s+10)$ with input u and output y in a unity feedback system with command input r; a model for the closed-loop system is:

$$\begin{aligned} \dot{x} &= -10x + u \\ y &= -9x + u \\ u &= r - y \end{aligned} \tag{10}$$

Evidently there is a circularity in these equations, as one must know u to evaluate y and *vice versa*. In many environments (e.g., MATLAB, Simnon) you are not allowed to program an ODE model with algebraic loops; however, Eqn. 10 may be treated as a DAE, or the work-around is to include some additional strictly proper dynamics in the loop, such as $100/(s + 100)$.

The fundamental distinction between PDEs and ODEs is that the former represent variation over both time and space (in one, two or three dimensions) rather than time alone – hence the need for partial derivatives, $\partial/\partial t$ as well as $\partial/\partial x$ and perhaps $\partial/\partial y$ and $\partial/\partial z$. An alternative nomenclature is *distributed-parameter* model for a set of PDEs and *lumped-parameter* model for a set of ODEs – in other words, the physical phenomena take place over a spatial distribution in a PDE but are "lumped" into single points in space in an ODE.

The digital simulation of a PDE set is itself a vast discipline, far beyond the scope of this presentation. Over the past 50 years very powerful techniques and algorithms have been developed for the solution of this problem, finite element methods and software being the best known and most utilized [18]. Here we merely observe that, as in the case of contrasting ODEs and DAEs, the decision to model using PDEs is again not firm. Instead, the requirement for realism for the problem being studied is the basis for selecting the modeling approach. For example, if one is modeling a drive system containing a flexible shaft, one may assume that the shaft is so stiff that flexibility may be ignored, or one may represent the flexibility using one lumped-parameter spring, or several such springs, or one may use a PDE representation. Considering the n lumped-parameter spring model, $n = 0, 1, 2, \ldots$, the $n = 0$ option corresponds to a rigid shaft, $n = 1$ accounts for the first resonant peak in a lightly damped assembly, etc. The basis for choosing n is the required bandwidth (or fast transient response) needed in the user's studies to be performed using the model. For ultimate fidelity one would have to choose n quite large – or use a PDE representation.

3. Simulation Overview

There are in fact two main considerations regarding the appropriate approach to be taken in simulating a dynamic system on a digital computer: the first is what language to employ in representing the model, and the second is what algorithm(s) to use to solve it. By "solve", in the context of ODEs, we informally mean take an initial condition, (x_0, t_0), and an input signal (or vector of signals), $u(t)$ defined for $[t_0\,, t_f]$ where t_f is the desired solution final time, and generate the solution, which we denote $x(t; x_0, t_0, u(\tau \in [t_0\,, t]))$ for $t \in [t_0\,, t_f]$. For DAEs and PDEs the term solve has a directly analogous meaning; hereafter, we will deal only with ODEs.

3.1. Simulation Languages

Most users take advantage of an off-the-shelf modeling and simulation environment, in which case the language is given. Well-known ground-breaking languages for modeling ODEs include the SCi Continuous System Simulation Language (CSSL [2]), the Advanced Continuous Simulation Language (ACSL [1]) and Simnon [5]. In the 1980s and 1990s the number of languages and environments proliferated, including extensions along traditional lines, e.g., the Hybrid System Modeling Language [21] and more innovative approaches, including object-oriented ones such as OMOLA [13] and DYMOLA [6]. In this same period several competing graphical modeling systems were developed and marketed, most notably MathWorks' SimuLink [19] and Integrated Systems' SystemBuild [20], where one assembles the model using interconnected blocks, picked, placed and connected graphically. Still more recently, at least in many disciplines, MathWorks' products have become dominant, with

MATLAB serving as a strong formulaic language (for which one writes the ODE set as a function in an `m-file` and uses one of the numerical integrators such as `ode45`), and SimuLink. The choice of a formulaic or a graphical modeling language is largely a matter of personal preference, although many feel that writing the ODEs directly permits more control and assurance of the outcome.

3.2. Simulation Algorithms

Turning to the question of algorithms, which in terms of quality simulation is **the** most important question, one must consider the type of system being studied. To make the issues clear, we continue to focus entirely on the simulation of ODEs. Several important categories should be noted:

1. *Continuous system models,* $\mathcal{M}_c$ – ideally, the nonlinearities and the input signals in f_c (Eqn. 3) should be continuous and differentiable so that the solutions $x(t; x_0, u(\tau \in [t_0\,, t]))$ can be fit accurately with high-order polynomials.

2. *System models with discontinuities,* $\mathcal{M}_d$ – either the nonlinearities or the input signals (or both) may make instantaneous changes in value (examples: a signum function or "ideal relay" nonlinearity, a square-wave input).

3. *Hybrid system models,* $\mathcal{M}_h$ – different researchers use this term in different ways; here we call any system where discontinuous effects and/or DTAs require the use of "modes" (below) to rigorously model its behavior a hybrid system.

There are several major families of integration algorithms, and their strengths and limitations make them more or less suitable for the categories of system models defined above.

1. *Continuous system models* – for models in $\mathcal{M}_c$ the best choices are the predictor / corrector methods [15]. They were derived to provide the best fit of an n^{th}-order polynomial to a number of past solution points, as well as to achieve desired stability properties (stability in the sense of integration errors not growing as more integration steps are taken). The general form: given $x_{k-m}, \ldots x_{k-1}, x_k$ and corresponding past derivatives, we first predict to obtain $x_{p,k+1}$, then calculate $\dot{x}_{k+1}$, and finally determine the corrected point:

$$\begin{aligned} x_{p,k+1} &= \sum_{i=0}^{m}(a_i x_{k-i} + h b_i \dot{x}_{k-i}) \\ \dot{x}_{k+1} &= f(x_{p,k+1}, t_{k+1}) \\ x_{c,k+1} &= \sum_{i=-1}^{m}(c_i x_{k-i} + h d_i \dot{x}_{k-i}) \end{aligned} \tag{11}$$

   Letting $m = 4$ and solving for the coefficients a_i, b_i, c_i, d_i to achieve stability plus minimum truncation error (making the algorithm exact for $x = t^4$) gives rise to a number of algorithms, e.g.:

$$\begin{aligned} x_{p,k+1} &= x_k + \frac{h}{24}(55\dot{x}_k - 59\dot{x}_{k-1} + 37\dot{x}_{k-2} - 9\dot{x}_{k-3}) \\ x_{c,k+1} &= x_k + \frac{h}{24}(9\dot{x}_{k+1} + 19\dot{x}_k - 5\dot{x}_{k-1} + \dot{x}_{k-2}) \end{aligned} \tag{12}$$

This algorithm is called Adams-Bashford / Adams-Moultin, the first pair of contributors being associated with the first or predictor formula, the second pair being responsible for the corrector. There are many algorithms of this type, differing in order as well as stability behavior (and hence in coefficient values). The strength of this family of algorithms is that error analysis is straightforward and rigorous, based on Taylor series expansions. This, in turn, makes automatic step-size control straightforward and rigorous. Automatic step-size control is incorporated in many integration routines, and it is very useful and effective in most cases (saving the user from having to select and perhaps iterate to find an appropriate step); however, this feature is not always effective and sometimes contraproductive (see discussion of discontinuities and state-event handling below). One limitation is clearly evident: one must start the solution some other way (e.g., by a Runge-Kutta method, see below), to obtain a sufficient number of points (four, in the above case) for the algorithm to start working. A second, and more important problem exists as well: whenever $\dot{x}$ is discontinuous the previous derivatives are meaningless and polynomial fitting is a mistake. With discrete-time subsystems, you should restart every sample time, which is easy, but with discontinuous nonlinearities it is hard to detect when such fitting is inappropriate.

2. *Discontinuous system models* – for models in $\mathcal{M}_d$ the best choices are the Runge-Kutta methods. These do not use past points and polynomial fitting, rather they "explore" the derivative vector field taking several fractional and whole steps from t_k to t_{k+1}, then they combine the resulting derivatives to generate the final result. The following is a simple 4th-order Runge-Kutta algorithm [15]: Given x_k and thus $\dot{x}_k$ we execute three explorations (two with half steps, one with a full step) and then combine the resulting derivatives to obtain the final result:

$$\begin{aligned} x^{(1)}_{k+\frac{1}{2}} = x_k + \frac{h}{2}\dot{x}_k \quad &\rightarrow \quad \dot{x}^{(1)}_{k+\frac{1}{2}} = f(x^{(1)}_{k+\frac{1}{2}}, t_k + \frac{1}{2}h) \\ x^{(2)}_{k+\frac{1}{2}} = x_k + \frac{h}{2}\dot{x}^{(1)}_{k+\frac{1}{2}} \quad &\rightarrow \quad \dot{x}^{(2)}_{k+\frac{1}{2}} = f(x^{(2)}_{k+\frac{1}{2}}, t_k + \frac{1}{2}h) \\ x^{(1)}_{k+1} = x_k + h\dot{x}^{(2)}_{k+\frac{1}{2}} \quad &\rightarrow \quad \dot{x}_{k+1} = f(x^{(1)}_{k+1}, t_k + h) \end{aligned}$$

$$\rightarrow x_{k+1} = x_k + \frac{h}{6}(\dot{x}_k + 2\dot{x}^{(1)}_{k+\frac{1}{2}} + 2\dot{x}^{(2)}_{k+\frac{1}{2}} + \dot{x}_{k+1}) \tag{13}$$

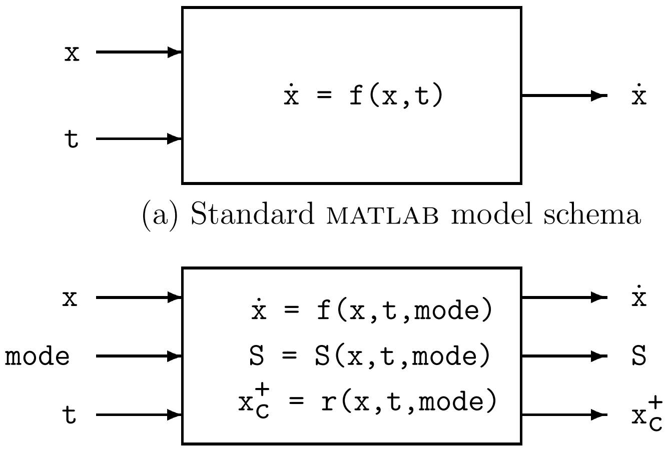
Here, too, error estimates and step-size adjustment can be made based on this exploration. The clear advantages are that these algorithms are self-starting, and they do not attempt to fit a high-order polynomial to a solution with discontinuous derivatives. It used to be said that the error and step-size control mechanisms were not on as firm a foundation as in predictor / corrector methods, but modern Runge-Kutta algorithms such as those found in MATLAB are excellent. There is only one serious problem (which also occurs with predictor / corrector methods): for certain discontinuous effects the nature of the solution causes the algorithm to decrease the integration step drastically, and the solution creeps along very slowly (in terms of computer time expended). This deficiency is eliminated by the hybrid system approach described next.

3. *Hybrid system models* – informally stated, models in $\mathcal{M}_h$ include those with serious discontinuities, even multi-valued nonlinearities (which may include effects such as hysteresis, for example) and perhaps interfaced DTAs. The trajectories of such systems are nonsmooth and, in general, require the use multiple models (depending on the mode(s) of the system) in order to achieve accurate and efficient simulation. Dynamic systems with these effects are best treated using state-event handling [22], where each change of mode (and thus model) is signaled by a switching function, and the model is changed accordingly. Most importantly, the algorithm should never blindly integrate past the mode change. To support this, we first extend the ODE part of a hybrid system to encompass the concept of modes:

$$\begin{aligned} \dot{x_c} &= f_c(x_c, u_c, u_d, m, t) \\ y_c &= h_c(x_c, u_c, u_d, m, t) \end{aligned}$$

   where m (generally a vector) keeps track of the current model to be used. For example, if a system includes a relay with deadzone, i.e., $f(v) = +F\,,\ v > v_s\,;\ 0\,,\ -v_s < v < v_s\,;\ -F\,,\ v < -v_s$, then the associated mode would take on the values 1, 0, -1 respectively; if mode = 1 then the model $f(v) = +F$ is used, and so on. The switching function for the transition from 1 to 0 is $S(v) = v - v_s$, and a state-event handling algorithm detects a corresponding zero-crossing, iterates to find the exact instant of switching, and changes the mode so that $f(v)$ in the model used after the event contains $f(v) = 0$. The simulation (numerical integration) of the system between mode changes is carried out by a suitable algorithm, usually a member of the Runge-Kutta family such as ode45 in MATLAB. It is particularly important to note that with this approach the simulator is always integrating a continuous model, so rigor and efficiency are maintained throughout.

To be more complete and specific: A model that supports mode changes may be structured as shown in Fig. 1. Compared with the standard MATLAB schema (a) where each model has the input variables t and x and output variable $\dot{x}$, the state-event handling approach (b) requires one additional input, the mode m, and two additional outputs, the switching (mode-changing) function $S(x, t, mode)$ and a state-reset, $r(x, t, mode)$. Again, note that S and m (mode) may be vectors, to



(a) Standard MATLAB model schema

(b) Extended MATLAB model schema

Figure 1: MATLAB model input/output structure

support multiple state events and switching conditions. Also, note that this schema incorporates another feature required to support multiple models and modes with generality: state variable reset. While standard simulation routines do not allow the user's model to modify the state, this assumes a fixed model, not one with changing modes in which, for example, it might be necessary to reset states after two bodies join (a state event) to conserve momentum.

Properly constructed, a model created using modes and the state-event handling approach promotes both rigor and efficiency. Rigor is achieved by providing mechanisms such that changes in the model occur predictably and at exactly the right time, and efficiency is accomplished by avoiding the "creeping simulation" problem mentioned above, where the numerical integration software tries to deal with any switching / discontinuity by dramatically reducing the step size until the event is located and past. For further information, see [23, 24] and / or visit the author's web-site www.ece.unb.ca/jtaylor/ where software for hybrid systems simulation and a number of tutorial examples are available for distribution.

4. Summary and Conclusion

We have briefly considered the development and importance of M & S in a wide range of applications, to show that this field of endeavor is becoming increasingly powerful and increasingly popular across virtually all area of technology. Then, we considered a variety of modeling approaches and models, to establish a context for discussing simulation methods and the close interrelations between the type of model and requirements for simulating them accurately and efficiently. The discourse on simulation methods was concluded with an exposition on hybrid systems, state-events and algorithms for dealing with systems that require multiple models depending on the mode of the system. This last topic illustrates the importance of awareness of issues and care in choosing M & S techniques and algorithms.

5. References

[1] *Advanced Continuous Simulation Language (ACSL), Reference Manual.* Mitchell & Gauthier Associates, Concord, MA 01742.

[2] Augustin, D. C., Strauss, J. C., Fineberg, M. S., Johnson, B. B., Linebarger, R. N., and Sansom, F. J., "The SCi Continuous System Simulation Language (CSSL)", *Simulation*, Vol. 9, No. 6, December 1967.

[3] Brenan, K. E., Campbell, S. L. and Petzold, L. R., *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North Holland, 1989.

[4] Campbell, S. L. and Griepentrog, E., "Solvability of General Differential Algebraic Equations", *SIAM J. of Scientific Computation*, Vol. 16, pp. 257-270, 1995.

[5] Elmqvist, H., "SIMNON - An Interactive Simulation Program for Non-Linear Systems", in *Proc. of Simulation '77*, Montreux, France, 1977.

[6] Elmqvist, H., Cellier, F. E. and Otter, M., "Object-Oriented Modeling of Power-Electronic Circuits Using Dymola", *Proc. CISS'94* (First Joint Conference of International Simulation Societies), Zurich, Switzerland, August 1994.

[7] Forrester, J., *World Dynamics*, Wright-Allen Press, Cambridge, MA, 1971.

[8] Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[9] Gear, C. W., "The Simultaneous Numerical Solution of Differential-Algebraic Equations", *IEEE Transactions on Circuit Theory*, Vol. TC-18, pp. 89-95, 1971.

[10] Gear, C. W. and Petzold, L. R., "ODE Methods for the Solution of Differential/Algebraic Systems", *SIAM J. of Numerical Analysis*, Vol. 21, pp. 367-384, 1984.

[11] Lotka, A. J., *Elements of Physical Biology*, Williams & Wilkins Co., Baltimore, 1925; Volterra, V. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. Mem. R. Accad. Naz. dei Lincei. Ser. VI, Vol. 21926; presented in many books, cf. pages 409-448 in Chapman, R. N., *Animal Ecology*, McGraw-Hill, New York, 1931.

[12] Ljung, L., *System Identification - Theory for the User*, Prentice Hall, Saddle River, NJ, 1999 (second edition).

[13] Mattsson, S. E. and Andersson, M., "The Ideas Behind Omola", *Proc. CACSD'92, IEEE Computer-Aided Control Systems Design Conference,* Napa, CA, pp. 218–224, March 1992.

[14] Petzold, L. R., "A Description of DASSL: A Differential/Algebraic System Solver", *Proc. 10th IMACS World Congress*, Montreal, August 1982.

[15] Pizer, S. M., *Numerical Computing and Mathematical Analysis*, Science Research Associates Inc., Chicago, 1975.

[16] Potra, F. A. and Rheinboldt, W. C., "Differential-Geometric Techniques for Solving Differential Algebraic Equations", in *Real-Time Integration Methods for Mechanical System Simulation*, Ed. by E. J. Haug and R. C. Deyo, Springer-Verlag Computer & Systems Sciences Series, Vol. 69, pp. 155-191, 1991.

[17] Rabier, P. J. and Rheinboldt, W. C., "A General Existence and Uniqueness Theorem for Implicit Differential Algebraic Equations", *Diff. Int. Eqns.*, Vol. 4, pp. 563-582, 1991.

[18] Reddy, J. N., *An Introduction to the Finite Element Method*, McGraw-Hill, New York, 1984.

[19] *SimuLink User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[20] *SystemBuild User's Guide*, Integrated Systems, Inc., Santa Clara, CA 95054.

[21] Taylor, J. H., "A Modeling Language for Hybrid Systems", *Proc. CACSD94* (IEEE/IFAC Symposium on Computer-Aided Control System Design), Tucson, AZ, March 1994.

[22] J. H. Taylor, "Rigorous Handling of State Events in MATLAB", *Proc. IEEE Conference on Control Applications,* Albany, NY, pp. 156-161, September 1995.

[23] J. H. Taylor and D. Kebede, "Modeling and Simulation of Hybrid Systems", *Proc. IEEE Conference on Decision and Control,* New Orleans, LA, pp. 2685-2687, December 1995.

[24] J. H. Taylor and D. Kebede, "Rigorous Hybrid Systems Simulation of an Electromechanical Pointing System with Discrete-time Control", *Proc. American Control Conference,* Albuquerque, NM, pp. 2786-2789, June 1997.