SIMULATION SOFTWARE AND NUMERICAL ISSUES

Alexander L. Pugh
Massachusetts Institute of Technology, Cambridge, USA

Keywords: Adams integration method, adaptive integration methods, Euler integration method, numerical integration, predictor-corrector integration method, rectangular integration method, reference mode, Runge-Kutta integration method, simulation error, simulation software, stiff systems

Contents

Summary

Practically all system dynamicists build simulation models using differential equations. The numerical solution of these equations on digital computers requires some sort of approximation such as offered by the methods of Euler or Runge-Kutta. The most sophisticated methods even allow the user to specify an error that should not be exceeded. Unfortunately, these methods can exhibit instability when they are used to approximate systems with both large and small time constants (stiff systems). Excellent software incorporating both of these methods exists to greatly simplify the construction of system dynamics (SD) models and to build easy-to-use interfaces, so that individuals unfamiliar with the model, and perhaps unfamiliar with SD, can carry out simulations of their own design.

1. Design Considerations

At the heart of the SD methodology is the construction and exercise of computer based models. (See *System Dynamics: Systemic Feedback Modeling for Policy Analysis*.) We build models:

- to implement our mental model in a precise form,

- to discover whether our mental model is complete,
- to communicate our model to others in an unambiguous way, thus permitting others to understand and participate in the construction of our model,
- to test whether this model is sufficient to reproduce the real world behavior we expect (the reference mode), and
- to build a vehicle to test ways of reducing the problem without the risk, elapsed time, and great expense that would be necessary to test these ideas in the real world.

There are three techniques for constructing such models:

- discrete-event models that reflect the world at the micro level of individual items and events, which typically are built using tools such as GPSS, Modsim, Promodel Simscript, Simula, SLAM, SIMAN, or Taylor;
- discrete-time difference equation models that relate the situation at one instant of time to an earlier one through subscripted variables (e.g. X_i, Y_{i-1}); or
- continuous, differential equation models (actually integral equation models) that are based on calculus.

We choose to abstract the real world into a set of differential equations because we are interested in the big picture. Discrete-event models invite a focus on detail, for example, the selection of a probability distribution to be used for a particular process. If the problem is at a very detailed level, these models are appropriate. But if the problem is at a larger level—a company interacting with its market, or a country addressing economic or environmental issues—a differential equation model is easier to develop and use. Most models have one or more intangible variables such as quality or reputation that cannot be measured precisely, but must be included if the model is to be realistic. Including these variables as real numbers in a set of differential equations is the easier way to proceed. Difference equations can also be used to build macro-level "big picture" models, but impose the restriction of requiring an early commitment to a discrete time step that may prove to be a poor choice as the model develops.

All digital differential equation solvers or simulators ultimately use difference equations to approximate integration. The first SD simulation language, Dynamo, explicitly displayed the difference equations it was using, because many early users of SD were not familiar with calculus and the difference equations were simple and nonthreatening. Unfortunately, this led some users to think in difference equation terms in all their generality, rather than the very restricted subset associated with Euler integration (described below).

A number of software tools exist to simplify the construction of difference equation models. General purpose languages such as those used in Excel and other spreadsheets, Basic, C, and Fortran can build the difference equations that approximate the differential equations, but leave much work for the builder to complete the task. Special purpose languages such as Powersim®, Stella®/ithink®, and Vensim® allow (or require) the model builder to diagram the model as (s)he enters the equations. (Stella and ithink are practically identical languages but are marketed to different audiences.)

They order the equations for proper computation of results, and provide easy to use methods to plot and tabulate these results. Dynamo, the first special-purpose language for SD, does not offer a diagramming facility, but does include all the other features of the graphical languages. These graphical languages enforce consistency between the equations and the diagram, which is an important plus. But medium-sized models can lead to very large, incomprehensible diagrams (sometimes referred to as spaghetti diagrams), and large models may be nearly impossible to diagram. Some languages offer means to reduce this problem, but none totally solve the problem for large models (over 1000 equations and constants).

One method for reducing complexity is to use arrays when there are several very similar structures, such as the manufacture of several products, or grouping persons or objects into age classes. While diagramming similar structures is fairly easy to understand, diagramming an aging chain expressed as an array does not give good insight into the true structure.

Another method used to simplify model construction is to use macros to repeat structure. Macros look like functions (have the same syntax as functions), but rather than involving specific machine language unique to themselves, as do the SIN and COS functions, they use the normal code generation facilities of the compiler. For example, all the languages offer a third-order delay that requires three levels and rates. But by using a macro, one only needs to specify the input and the delay to completely specify (and diagram) the delay. (Not all languages allow the user to design his/her own macros.)

2. Major Numerical Methods

All the continuous simulation languages solve or simulate a set of simultaneous integral equations (also called ordinary differential equations: functions of a single variable—generally time—in contrast to partial differential equations, which typically are functions of space and time) by approximating the integration process with a set of difference equations. (Time is broken into finite steps rather than being truly continuous.)

2.1. Rectangular (Euler) Integration Method

Most system dynamicists use the rectangular (Euler) integration method, which is the default in all the SD simulation languages. In the Euler method, flows into and out of all the levels (stocks) are computed for the beginning of the interval and then assumed constant for the balance of the interval. Recalling that integration is computing the area under a curve (variable versus time), we approximate that area by a series of rectangles whose height is the value of the curve at the beginning of the time step, and width is the time step (referred to as DT for delta time).

As a very simple example let us integrate TIME squared:

$$X = \int_2^4 TIME^2 dt$$

Or

X = INTEGRAL(TIME^2)

We will use a time step of one unit of TIME. Starting at TIME = 2, the integrand (thing being integrated) is 4. Multiplying that by the step size (1.0) we get 4.0. At TIME = 3 we repeat the process and get 9.0. (See Table 1 and Figure 1.)

TIME	$TIME^2$	Integral
2	4	4
3	9	9
Total =		13

Table 1. DT = 1

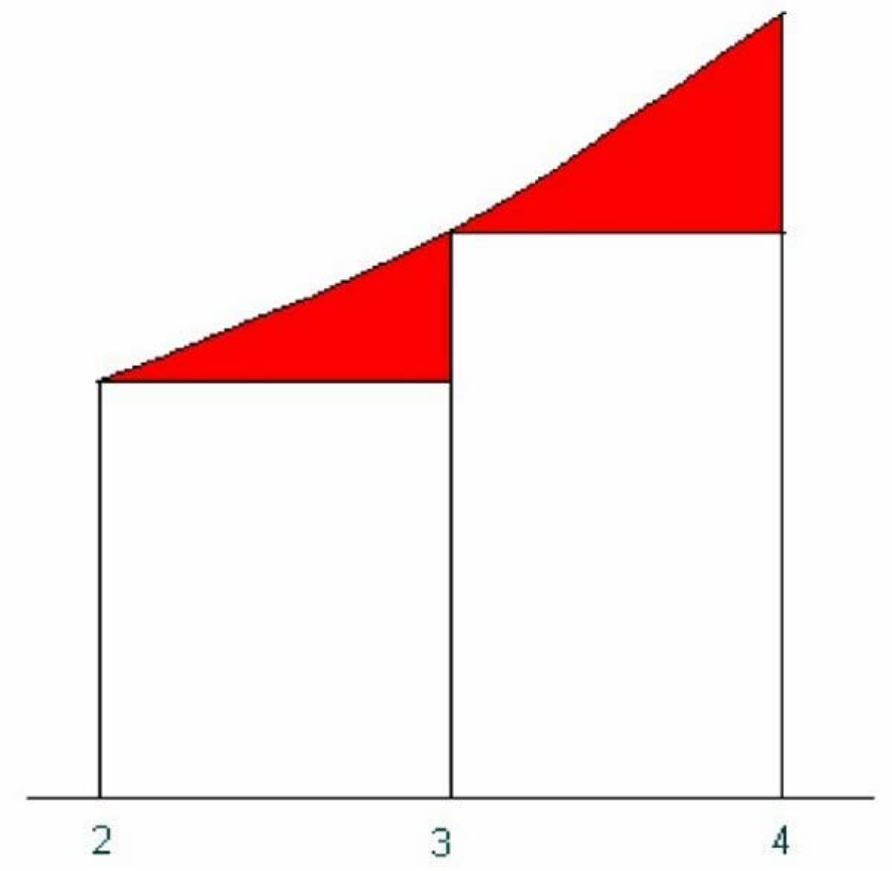

Figure 1. Simple integration (error shown in red)

This is simultaneously very simple and very crude. Obviously, reducing the size of the time step will reduce the error. We will return to error considerations later.

-
-
-

Bibliography

Peterson S. and Richmond B. (1997). *ithink Technical Documentation*, 394 pp. Hanover, NH: High Performance Systems, Inc. [Similar documentation for Stella is also available. Both manuals are updated regularly.]

Powersim Corp. (1996). *Powersim 2.5 Reference Manual*, 425 pp. Herndon, VA: Powersim Corp. [Later versions of the software are available along with updated manuals.]

Pugh A.L. (1983). *Dynamo User's Manual*, 121 pp. Massachusetts, USA: Pegasus Communications Inc. [Describes the integration methods used in Dynamo.]

Shampine L.F. and Gordon M.K. (1975). *Computer Solutions of Ordinary Differential Equations, The Initial Value Problem*, 318 pp. San Francisco: W. H. Freeman and Co. [A thorough discussion of the Adams method of integration.]

Sterman J.D. (2000). *Business Dynamics, Systems Thinking and Modeling for a Complex World*, 982 pp. Boston: Irwin/McGraw-Hill. [See Appendix A for a discussion for numerical integration.]

Ventana Systems, Inc. (1997). *Vensim® Personal Learning Edition User's Guide*, 168 pp. Harvard, MA: Ventana Systems, Inc. [Later versions of the software are available along with updated manuals.]

Biographical Sketch

Alexander L. Pugh (also known as Jack Pugh) was born in 1930 in Philadelphia, Pennsylvania. With degrees from University of Pennsylvania and Massachusetts Institute of Technology, he joined Jay Forrester in 1957 to help form the field of system dynamics. In 1959 he took over the development of the Dynamo simulation language at MIT. In 1963 he founded Pugh-Roberts Associates, Inc. with Edward B. Roberts, and was Treasurer and Director until the firm was sold to PA Consulting Group in January of 1991. He both consulted in SD and continued the development of Dynamo there. He retired in 1995.

Along with George P. Richardson he wrote *Introduction to System Dynamics Modeling with Dynamo*. He was one of the founders of the System Dynamics Society and its treasurer for 15 years.