

SIMULATION OF COMPLEX SYSTEMS

**MIGRATING FROM
MATHEMATICAL MODELS
TO
GENERALIZED STATE SPACE**

**By:
William C. Cave**

October 9, 2008

© Copyright 1993 - 2008

Visual Software International

309 Morris Avenue
Spring Lake, NJ 07762

 (732)449-6800
 VSI@VisiSoft.US

 (732)449-0897
 www.VisiSoft.US

TABLE OF CONTENTS

SECTION	TITLE	PAGE
1.	INTRODUCTION	1
2.	ANALYTICAL METHODS	3
3.	NUMERICAL METHODS	11
4.	GENERALIZED STATE SPACE METHODS	15
5.	RABBIT-COYOTE BIOLOGICAL MODEL	23
6.	MODELING DISTRIBUTED RESPONSES TO EVENTS	32
7.	GSS-OIL TANKER DISTRIBUTION-SIMULATION	37
	APPENDIX A - STATE SPACE DEFINITION OF A GSS MODEL	50
	REFERENCES	55

1. INTRODUCTION

For years, scientists and engineers have built models using mathematics. In concert, the field of mathematics has expanded to fulfill the needs for modeling complex physical systems. Fields like classical mechanics, electromagnetic theory, quantum theory, control theory, etc., have made heavy use of systems of equations and vector spaces to explain complex physical behavior. Analog computers were devised to solve complex systems of differential equations by simulating the systems being analyzed using electrical analogs. Most of this work was done before the digital computer became a practical device (circa 1955).

Since that early era, computers have pushed the ability to model complex system behavior to new heights, allowing problems that were heretofore intractable to be solved in a matter of hours. Numerical methods have been built to allow the computer to solve these problems as posed in standard analytical formats, while taking advantage of computer speed and accuracy. However, most of these problems are still formulated using the mathematical approaches devised for analytical (paper and pencil) methods.

To this author's knowledge, the first appearance of the discrete event approach to simulation was published by Geoffrey Gordon in the 1961 Proceedings of the EJCC, [3]. This paper, and others, [4] and [5], describe IBM's General Purpose System Simulator (GPSS). In the later part of the 1960's other products started to appear, e.g., SIMSCRIPT and SLAM, that took advantage of the digital computer's ability to process what have come to be known as discrete events. For most of its history, this "new" approach has been confined mainly to the problems of operations research. A limited counterpart in mathematics that supports control theory includes alphabets as a generalization of numbers, see [10] and [11].

Even today, most modeling is done using an analytically oriented mathematical approach. And that brings us to the purpose of this book. Many classes of problems are best solved using a mathematical approach. However, as we try to understand more complex systems, such as chemical systems, communication systems or human behavior, it is difficult to fit the problem to solution methods when restricted to a mathematical framework. Discrete event simulation is clearly a quantum leap toward solving these otherwise hard-to-define problems.

There are many approaches to implementing a general purpose discrete event simulation environment. The implementation described here, namely the General Simulation System (GSS), is based upon the Generalized State Space framework described in [1]. This makes it is easy to solve complex mathematical problems along with those not easily represented mathematically. We will show that the same solutions can be obtained by applying generalized state vectors and transformations using a set of rules and simple algebra that do not have to conform to a "system of equations" approach.

This book attempts to explain and demonstrate the differences in approaches to simulation with an eye toward showing how the Generalized State Space approach is most general. Having developed simulation environments for the three types of problems - continuous time, discrete time, and discrete event - since the early 1960s, this author has been confronted with the introduction of new approaches for a long time. It is interesting to see the field changing, and it is hoped that this work serves to expose the advantages of each, and especially to introduce Generalized State Space to those of whom it is unfamiliar.

There are many types of systems that can only be analyzed through simulation on a computer. The entities in these systems can be represented in different forms of models. For example, when simulating an electronic circuit, one typically uses *ordinary differential equations* to represent the circuit dynamics, see Appendix A. This is because the elements, such as capacitors, inductors, resistors, transistors, etc. are very conveniently described in that mathematical framework. This is also true for fluid flow problems, except when the system cannot be represented simply as flow between sets of points, and one must use *partial differential equations* to accurately represent the mechanics. Simulation problems using differential equations have been addressed for many decades, and approaches to their solution are well known. Special vector processor computers have been developed to speed up solution of large scale problems of this nature.

In the 1950's, John Von Neumann showed that economic systems could be modeled using *discrete-time difference equations* whose mathematical form is similar to that of ordinary differential equations, except that integration is not necessary. Much of economic theory has evolved around that approach.

The types of problems usually described using a *system of dynamic equations*, i.e., discrete-time difference equations or ordinary differential equations, are presented in Chapters 2 and 3 along with a discussion of typical solution approaches. These approaches fall into the categories of *analytical* and *numerical* methods as described in Chapters 2 and 3 respectively.

Solutions to differential or difference equations are best obtained using general algorithms so that the modeler does not have to rebuild them every time he wants to solve a problem. This approach has formed the basis for many simulation systems that are sold as software packages. However, the statement of the problem must be put into a form required by the approach to solving the mathematical equations. These solution approaches, by their nature, are not conducive to entering English-like decision rules.

When one tries to represent economic or political systems using mathematical equations, great difficulties occur because of the decision processes that must be represented. This is becoming true in engineering, e.g., communications and control systems, where complex decision processes are built into the software. These decision processes are quite obvious to people that understand the application systems. The decision rules could easily be written in plain English. They typically take the form:

IF *THIS HAPPENS*, DO *SOMETHING*, ELSE DO *SOMETHING ELSE*.

This problem is described in Chapters 3 and 4. As indicated above, the author has built three software products for simulation of the three types of problems described above. The most recent product, the General Simulation System (GSS), is based upon Generalized State Space. Chapter 4 describes how GSS can be used to represent systems of dynamic equations as well as rule-oriented decision systems where the time scale is dynamic.

The most important factor to be dealt with when modeling systems containing complex decision processes is their highly nonlinear nature. When decisions take on discrete states, they must be represented accurately. Small inaccuracies can accumulate to render a model of the system to be highly inaccurate. The purpose of this manuscript is to show how important these decision rules are, and how easily they can be modeled by using GSS.

2. ANALYTICAL (PRE-COMPUTER) METHODS

Prior to the advent of high speed computers, analyses of complex systems were performed analytically. In the fields of science and engineering, analysis was generally performed mathematically. Engineers and scientists learned to build mathematical models of the systems they were trying to understand. These mathematical models helped them to analyze the dynamic behavior of complex systems.

As systems became more complex, special methods evolved to accommodate the detailed analyses needed to predict their dynamic behavior under various conditions. The next sections cover some common examples of these analytical methods.

REPRESENTATION OF A LINEAR DISCRETE-TIME SYSTEM

We start with a set of linear *discrete time* equations of the form:

$$\begin{aligned}
 x_1(t+1) &= a_{11} \cdot x_1(t) + a_{12} \cdot x_2(t) + \dots + a_{1n} \cdot x_n(t) + b_1 \cdot u_1(t) \\
 x_2(t+1) &= a_{21} \cdot x_1(t) + a_{22} \cdot x_2(t) + \dots + a_{2n} \cdot x_n(t) + b_2 \cdot u_2(t) \\
 &\vdots \\
 &\vdots \\
 &\vdots \\
 x_n(t+1) &= a_{n1} \cdot x_1(t) + a_{n2} \cdot x_2(t) + \dots + a_{nn} \cdot x_n(t) + b_n \cdot u(t)
 \end{aligned}$$

These equations can be rewritten in matrix form as:

$$\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \\ \vdots \\ \vdots \\ \vdots \\ x_n(t+1) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ \vdots \\ \vdots \\ x_n(t) \end{bmatrix} + \begin{bmatrix} b_1 & 0 & \dots & 0 \\ 0 & b_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_n \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ \vdots \\ \vdots \\ u_n(t) \end{bmatrix}$$

In these equations, $x(t)$ is a *state* variable of the system and $u(t)$ is an external driving force. The vector of state variables is called the *state vector*, and the driving forces the *driving force vector*. This approach is sometimes referred to as the *state space* framework for solving linear systems.

Defining the state vector at t+1 as X(t+1), then

$$X(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \\ \cdot \\ \cdot \\ \cdot \\ x_n(t+1) \end{bmatrix}$$

and the matrix A is:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n1} & \dots & a_{nn} \end{bmatrix}$$

Then with matrix B and vector U defined similarly, the system of equations can be represented as:

$$X(t+1) = A \cdot X(t) + B \cdot U(t)$$

where B is typically a diagonal matrix. If this were a time-variant linear system, then the coefficients, a_{mn} , would be functions of time, i.e., $a_{mn}(\tau)$.

This is the same form as a *continuous-time* system represented by a set of linear first order differential equations.

$$\frac{dX}{dt} \cong \frac{X(t+\Delta t) - X(t)}{\Delta t} = A \cdot X(t) + B \cdot U(t), \text{ as } \Delta t \text{ becomes small enough.}$$

Note that if $\Delta t = 1$ is sufficiently small compared to the system time constants, then A in the discrete-time equation is approximately equal to (A - I) in the continuous-time equation above.

ANALYTICAL SOLUTION OF A LINEAR CONTINUOUS-TIME SYSTEM

To solve the system of linear first order differential equations above using an analytical approach, one first finds the solutions to the *homogenous* part of the system, i.e. that part remaining after dropping the external inputs U. In many cases of interest, the solutions for this part are known to be of the form $C \cdot e^{\lambda t}$ for each x, where C is a constant determined by the initial conditions, and λ is a complex variable determined by solving the system of equations. The meaning of λ is described in many texts in different ways. In general, it is a transformation variable, allowing ease of solution and many insights into complex dynamic systems. It is

known as the LaPlace transform variable, denoted by s , in engineering texts. In mathematics texts, it is called the Eigenvalue.

To derive the solutions posed above, let's assume that we can find a transformation on the homogeneous system from the x variables to y variables such that the form of the equations of state for Y is given as follows:

$$\begin{bmatrix} dy_1/dt \\ dy_2/dt \\ \cdot \\ \cdot \\ \cdot \\ dy_n/dt \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \begin{bmatrix} y_1(t) \\ y_2(t) \\ \cdot \\ \cdot \\ \cdot \\ y_n(t) \end{bmatrix}$$

This system of equations can be represented by the vector equations.

$$\frac{dY}{dt} = \Lambda \cdot Y(t)$$

Now Λ is a diagonal matrix, and the solution for each y_k is simply of the form.

$$Y_k(t) = C_k \cdot e^{\lambda_k t}$$

where C_k is found from the initial conditions, $C_k = y_k(0)$. If we can find a transformation, T , such that

$$X(t) = T \cdot Y(t)$$

then, the solution for X is easily found. This assumes that the inverse, T^{-1} , exists, i.e.,

$$Y(t) = T^{-1} \cdot X(t)$$

Taking the derivative of both sides of the equation,

$$\frac{dY}{dt} = T^{-1} \cdot A \cdot X(t) = T^{-1} \cdot A \cdot T \cdot Y(t)$$

The transformation, $T^{-1} \cdot A \cdot T$, is known as a *similarity transformation* in matrix theory. It transforms the coordinates of a linear system such as to minimize the terms in the matrix, Λ , if not diagonalize it. Remember, if Λ is a diagonal matrix, then the solution for each y_k is simply of the form $C_k \cdot e^{\lambda_k t}$. This is not always the case, and additional terms may have to be added. However, for our examples, we will start by assuming that Λ is a diagonal matrix. Note, finding Λ depends on finding T^{-1} .

Given the above, X is easily found if we find T. From the above equation, we see that,

$$[(T^{-1} \cdot A \cdot T) - \Lambda] \cdot Y(t) = 0$$

The nontrivial solutions ($Y(t) \neq 0$) occur when

$$T^{-1} \cdot A \cdot T = \Lambda$$

and the above transformation is the similarity transformation that, in our case, diagonalizes A. Multiplying both sides by T,

$$A \cdot T = T \cdot \Lambda$$

If Λ is a diagonal matrix, then each column of the resulting matrix, $T \cdot \Lambda$ satisfies the equation

$$A \cdot [T_1 | T_2 | \dots | T_n] = [\lambda_1 \cdot T_1 | \lambda_2 \cdot T_2 | \dots | \lambda_n \cdot T_n]$$

$$A \cdot T_k = \lambda_k \cdot T_k$$

For T^{-1} to exist, the T_k must be *linearly independent* column vectors. These are called the eigenvectors of the system. Restating the problem, each eigenvector is a solution to the system

$$A \cdot X = \lambda \cdot X$$

For each λ_k , one solves the system of n equations in the n unknowns: x_1, x_2, \dots, x_n .

The solutions for λ_k are found by solving the equation

$$[A - \lambda]X = 0$$

yielding the requirement that, for each particular (nontrivial) solution, λ_k , the following must be true.

$$\text{Det}[A - \lambda_k \cdot I] = 0$$

Thus, the λ_k are the roots of the determinant of the matrix $[A - \lambda \cdot I]$. If the λ_k are all distinct, then Λ is diagonal. If there are repeated roots, then Λ will have off-diagonal terms, and the system will have additional solutions of the form $C_k \cdot t^n \cdot e^{\lambda_k t}$.

The general approach above applies whether the system is described as continuous in time, or has discrete sample points in time.

EXAMPLE OF FACTORY PRODUCTION, WAREHOUSE, CONSUMER SYSTEM

Let's assume that consumers buy a given product based on their recent prior consumption, and the amount of stock available on the shelf at the market (as the stock goes up, retailers tend to lower prices). We can write the following simple *discrete time* equation to describe this as follows:

$$\text{PURCHASES (T+1)} = \text{PURCHASES (T)} + K \cdot \text{STOCK (T)}$$

where K is a constant to be determined from actual experience. Let's also assume that the factory produces goods that are put directly into stock. This results in the simple equation as follows:

$$\text{STOCK (T+1)} = \text{STOCK (T)} + \text{PRODUCTION (T)} - \text{PURCHASES (T)}$$

Note, one can interpret PURCHASES(T+1) AND PRODUCTION(T+1) as averages over the time period (T, T+1). The above equations can be written in matrix form as follows

$$\begin{bmatrix} \text{STOCK (T + 1)} \\ \text{PURCHASES (T + 1)} \end{bmatrix} = \begin{bmatrix} \text{STOCK (T)} & - \text{PURCHASES (T)} \\ K \cdot \text{STOCK (T)} + \text{PURCHASES (T)} \end{bmatrix} + \begin{bmatrix} \text{PRODUCTION (T)} \\ 0 \end{bmatrix}$$

This can be written as:

$$\begin{bmatrix} \text{STOCK (T + 1)} \\ \text{PURCHASES (T + 1)} \end{bmatrix} = \begin{bmatrix} 1 & (-1) \\ K & 1 \end{bmatrix} \cdot \begin{bmatrix} \text{STOCK (T)} \\ \text{PURCHASES (T)} \end{bmatrix} + \begin{bmatrix} \text{PRODUCTION (T)} \\ 0 \end{bmatrix}$$

This *discrete time* system is of the form

$$X(T+1) = A \cdot X(T) + B \cdot U(T)$$

To find the eigenvalues for this discrete problem, we replace A with (A + 1) and solve the equation previously derived:

$$\text{Det}[(A + I) - \lambda \cdot I] = 0$$

By setting

$$\text{Det} \begin{bmatrix} (2 - \lambda) & (-1) \\ K & (2 - \lambda) \end{bmatrix} = 0$$

This yields the equation.

$$(2 - \lambda)(2 - \lambda) - (-1)K = 0$$

Rearranging terms, we get:

$$(2 - \lambda)^2 = -K$$

Thus,

$$(2 - \lambda) = \pm [-K]^{1/2}$$

or

$$\lambda = 2 \pm i \cdot [K]^{1/2}$$

where i denotes the imaginary part of the complex number. This system has an inherent frequency of oscillation corresponding to $\omega^2 = K$.

When this model is analyzed, purchases will oscillate with a frequency of approximately

$$f = [K]^{1/2} / 2 \cdot \pi$$

The actual value of this frequency will be adjusted to accommodate the discrete time steps. The time between peaks is $1/f$.

The intent of this example is to demonstrate an approach. The decision to make purchases based upon stock is clearly over simplified. It is possible for purchases and stock to take on large values depending on K and B . This is not realistic, since the consumer will eventually saturate, i.e., consumption will increase to the point where one can only consume so much each month. This leads us to use *nonlinear* models.

NONLINEAR SYSTEMS

The following example illustrates the problems encountered modeling nonlinear systems. Consider a model for predicting sales next month of the form

$$x(T+1) = A \cdot x(T) + B \cdot u(T)$$

where $x(T)$ are sales this month, A is a coefficient on current sales that yields new sales due to word-of-mouth, and B is a coefficient on advertising expenditures this month, $u(T)$. If sales increased linearly with advertising, we should just continue to plow back funds into advertising, as long as we are making a profit, reference Figure 2.1.

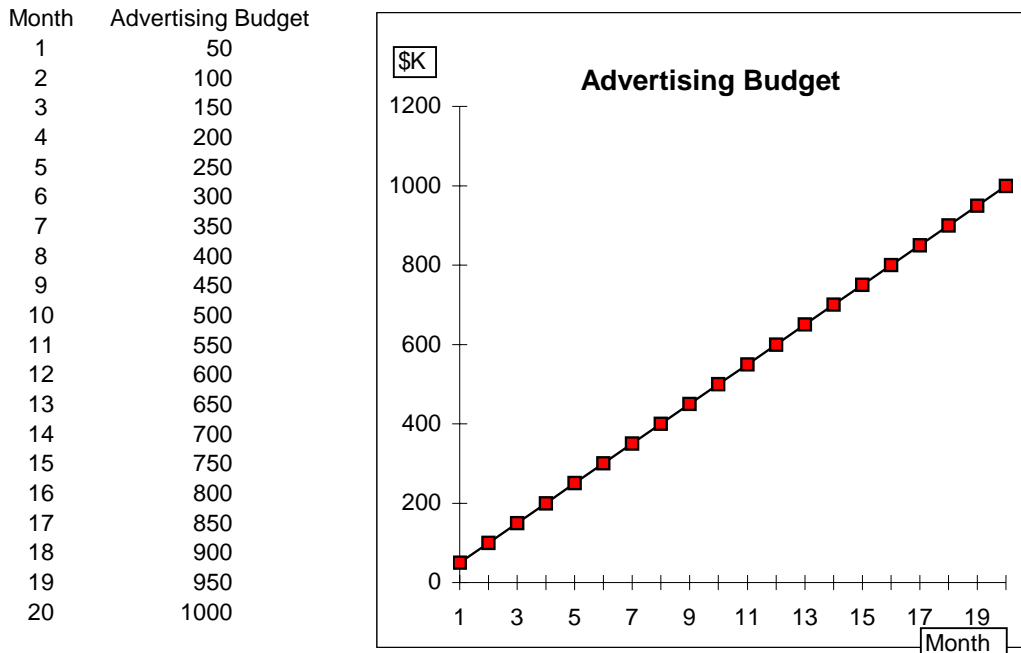


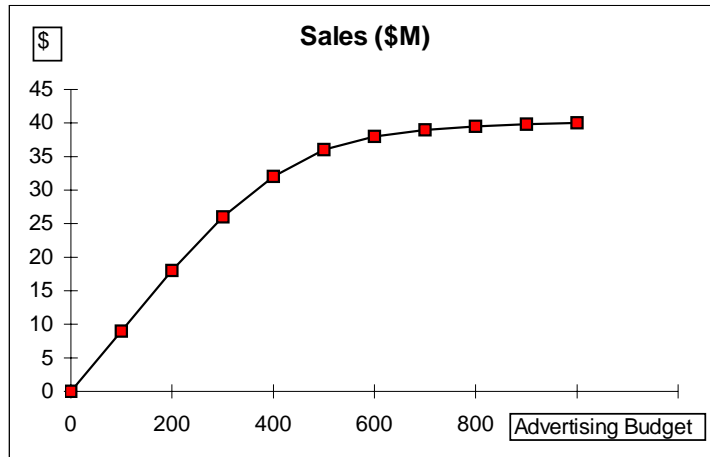
Figure 2.1. Advertising budget as a function of time (months)

Experience indicates that a continual linear increase in sales due to increased advertising is unrealistic due to market saturation. At some point we run out of buyers. Assume that an experiment is run in a small area to determine that relationship, with the results shown in Figure 2.2.

Advertising experiment yields:

U	X
Advertising	Sales (\$M)
0	0
100	9
200	18
300	26
400	32
500	36
600	38
700	39
800	40
900	40
1000	40

$$X(T+1) = f[\text{Advertising}(T)]$$

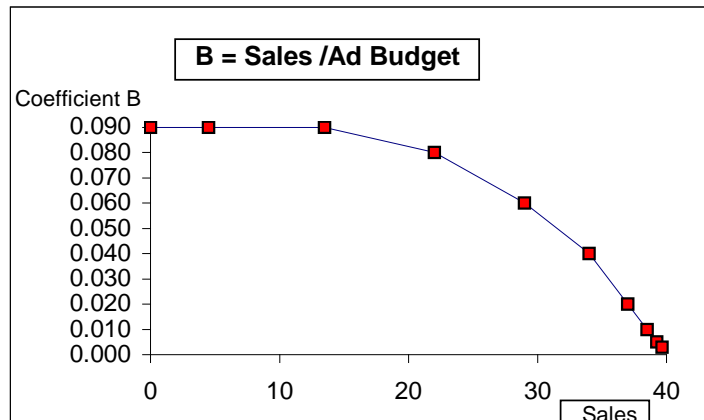


SATURMOD As of 2/25/93

Figure 2.2. SALES as a function of ADVERTISING.

\$	B
Sales	Coefficient B
0	0.090
4.5	0.090
13.5	0.090
22	0.080
29	0.060
34	0.040
37	0.020
38.5	0.010
39.25	0.005
39.65	0.003

$$B = f(\text{Sales}) \text{ Independent of time.}$$



SATURMOD As of 2/25/93

Figure 2.3. Describing function for advertising coefficient - B

Figure 2.3 above shows a potential advertising coefficient, B, for our mode. However, we must factor in the word-of-mouth sales over time as well. This is a nonlinear problem.

3. NUMERICAL METHODS USING THE COMPUTER

The types of nonlinear problems, described in Chapter 2 can be solved using numerical computer methods. There are many ways to do this, generally involving iterative algorithms. However, these approaches all evolve from systems of equations as illustrated above. In general, the users must describe their problem in a framework that is consistent with the mathematical approach underlying the solution. In the case of a nonlinear problem as described above, these algorithms must insure convergence in a reasonable number of iterations at each time step. To insure fast solutions, equations are typically produced by precompilers that generate efficient code for fast algorithms to solve the problem iteratively.

For nonlinear systems, or systems involving a transformation for solution, numerical approaches typically apply fast matrix manipulation methods. In the case of continuous time systems, one may invoke methods for variable time-step integration. These special methods must be tuned to the types of problems being solved, to insure an accurate representation of the system being modeled while meeting requirements for convergence and speed.

The above factory-warehouse-consumer system example can be programmed to simply iterate through the equations numerically, calculating the values for the next time step based on those from the prior time step. There were no nonlinearities in this simple model, and therefore no *algebraic loops* that require an iterative solution at each time step. The computer can simulate what will happen for various values of the parameter K , as well as production and consumption levels. However, unrealistic results can occur, e.g., continued increases in purchases, for certain values of the selected parameters. To insure that models are realistic, one must account for the nonlinearities that occur in real life.

To use a typical numerical algorithm to solve the nonlinear problem, one must go back and *create* a nonlinear function to insure that purchases do not go past saturation. In the engineering literature, nonlinear functions used to solve problems of this type are called *describing functions*. It is most convenient if a describing function can be formed using a sequence of straight line approximations to the nonlinear function. If we look at Figures 2.2 and 2.3, we note that the points of these functions are connected by straight lines. Note also that smooth curve can be approximated as closely as desired with enough straight lines. This makes it possible to develop algorithms that converge rapidly, as well as have a very low probability of divergence with many nonlinear functions in the system.

The important part of this discussion is the following. Independent of the numerical methods used to solve the resulting system of equations, the approach to describing the problem is implicitly constrained to the mathematical framework that is dictated by those numerical methods.

CONSUMER GOODS EXAMPLE

In this example, consumers purchase goods from a store that stocks the goods. During each month, goods are produced and put into the store. Also during the month, goods are purchased by the consumer, put on the shelf at home, and consumed. The storekeeper prices the goods according to the stock on hand at the beginning of the month based on the following rule.

$$[1] \quad \text{PRICE}(T, T+1) = \text{NOMINAL_PRICE} * (\text{NOMINAL_STOCK} / \text{STOCK}(T))$$

The notation $(T, T+1)$ *implies during the period* $(T, T+1)$. This equation states that the price of goods for the month will be set at the beginning of the month, T , based on the stock at the beginning of the month, $\text{STOCK}(T)$, and the NOMINAL_PRICE and NOMINAL_STOCK values that are known pricing constants.

The consumer decides to purchase goods during the period $(T, T+1)$ according to the rule:

$$[2] \quad \text{PURCHASES}(T, T+1) = \text{CONSUMPTION}(T, T+1) - \text{SHELF}(T) \\ + \text{EXCESS} * (\text{NOMINAL_PRICE} / \text{PRICE}(T, T+1))$$

$\text{CONSUMPTION}(T, T+1)$ is the amount consumed during the month, and $\text{SHELF}(T)$ is what is on the shelf at home at the beginning of the month. EXCESS is a constant factor used to determine the excess goods that he will buy, depending on the price of the goods relative to the nominal price.

We can now write an equation for the stock in the store at the end of the month.

[3] $\text{STOCK}(T, T+1)$ is the amount produced for the store during the month. Similarly, we can write an equation for the amount of goods on the consumers shelf at the end of the month as follows:

$$[4] \quad \text{SHELF}(T, T+1) = \text{SHELF}(T) + \text{PURCHASES}(T, T+1) - \text{CONSUMPTION}(T, T+1)$$

Note that equation [1] only depends on stock at the beginning of the month and two constants, so it is easily calculated without any of the other equations. We will assume that consumption is constant here, so that equation [2] can be computed given PRICE . In this simple example, we will assume that production is constant, so that [3] can now be computed directly after [2]. Given [3], [4] can now be computed.

There is a tacit assumption here that the purchases will not cause the stock to go negative. We will address this assumption at the end of this example.

Rewriting these equations such that the notation (T+1) implies *during the period (T, T+1)* and noting that consumption and production have been assumed constant we obtain the following.

$$[5] \quad \text{PRICE}(T+1) = \text{NOMINAL_PRICE} * \text{NOMINAL_STOCK} * (1 / \text{STOCK}(T))$$

$$[6] \quad \text{PURCHASES}(T+1) = \text{CONSUMPTION} - \text{SHELF}(T) + \text{EXCESS} * (\text{NOMINAL_PRICE} / \text{PRICE}(T+1))$$

$$[7] \quad \text{STOCK}(T+1) = \text{STOCK}(T) - \text{PURCHASES}(T+1) + \text{PRODUCTION}$$

$$[8] \quad \text{SHELF}(T+1) = \text{SHELF}(T) + \text{PURCHASES}(T+1) - \text{CONSUMPTION}$$

We note that this is not a linear system of equations, since PRICE depends on the inverse of STOCK(T), and PURCHASES(T+1) appears on the right hand side of the equation as does PRICE(T+1) which is also inverted.

If we replace PRICE(T+1) in equation [6], with [5], we get:

$$[9] \quad \text{PURCHASES}(T+1) = \text{CONSUMPTION} - \text{SHELF}(T) + \text{EXCESS} * (1 / \text{NOMINAL_STOCK}) * \text{STOCK}(T)$$

Similarly, we can substitute [9] for PURCHASES(T+1) in [7] and [8], getting:

$$[10] \quad \text{STOCK}(T+1) = (1 - \text{EXCESS} / \text{NOMINAL_STOCK}) * \text{STOCK}(T) + \text{SHELF}(T) + \text{PRODUCTION} - \text{CONSUMPTION}$$

$$[11] \quad \text{SHELF}(T+1) = (\text{EXCESS} / \text{NOMINAL_STOCK}) * \text{STOCK}(T)$$

Substituting $K = \text{EXCESS} / \text{NOMINAL_STOCK}$, letting $P = \text{PRODUCTION}$ $C = \text{CONSUMPTION}$, we get the following linear system form:

$$\begin{bmatrix} \text{STOCK}(T + 1) \\ \text{SHELF}(T + 1) \end{bmatrix} = \begin{bmatrix} (1 - K) & + 1 \\ K & 0 \end{bmatrix} \begin{bmatrix} \text{STOCK}(T) \\ \text{SHELF}(T) \end{bmatrix} + \begin{bmatrix} P - C \\ 0 \end{bmatrix}$$

To find the eigenvalues of the transition matrix, we set

$$\text{Det} \begin{bmatrix} (2 - K - \lambda) & (-1) \\ K & (1 - \lambda) \end{bmatrix} = 0$$

This yields the equation.

$$(2-K-\lambda)(1-\lambda) + K = 0$$

or

$$\lambda^2 + (K-3)\lambda + 2 = 0$$

Thus,

$$\lambda = 1/2\{-(K-3) \pm [(K-3)^2 - 8]^{1/2}\}$$

K must always be greater than 0. This implies that λ has an imaginary part when $(K-3)^2$ is less than 8. This occurs when $1 < K < 5.83$ which should be most of the time. When K is greater than 3, then λ has a negative real part, implying damping exists when EXCESS starts to grow. When K is between 0 and 1, λ is real and there is no oscillation.

When this system is simulated, and $K = 1$, PURCHASES can take on negative values as the SHELF builds up. This is unrealistic, since we are not accounting for the decision *not to purchase* if PURCHASES go negative. In other words, if the amount on the consumer's shelf is sufficiently large compared to the excess PURCHASES, then the buyer simply would not purchase any goods. This implies a test for purchases going less than 0, in which case they would simply be set back to 0.

Also note that the order in which the equations were written and reduced has been most convenient. If we were not able to achieve the order stated above, or if we could not make the simplifying reductions, then we could not have solved this problem using linear systems theory. We would have had to resort to special nonlinear approaches. Furthermore, imposing the condition that, *If PURCHASES are computed to be negative, set purchases to 0*, cannot be accomplished directly using the linear system format with automatic solutions.

Using the special assumptions made to solve this problem, we can go back to the original equations, [5] through [8], and put a conditional statement in after equation [6] that sets PURCHASES to 0 when it goes negative. This skirts the problem of solving a nonlinear system simply by modeling the decision processes as they are made in real life. However, this approach does not lend itself to a priori mathematical solutions for linear and nonlinear systems. One must build the rules based on a model of the real world system as necessary.

4. GENERALIZED STATE SPACE AND DISCRETE EVENT SIMULATION

The consumer goods example described above is small compared to most models of this type. Large numbers of equations are normally needed to describe the real world problem. Modelers also want to modify these equations often as they try different runs of their simulations. They want to do this without changing the solution approach. This generally leads to a mathematical framework that provides automatic numerical solutions to systems of potentially nonlinear equations similar to those above. To solve such a problem requires a general solution to a nonlinear system, typically involving sparse matrix manipulation methods, and iterative solutions to balance the nonlinear equations. Such a framework precludes the insertion of conditional or "IF" type statements in between the equations.

In most current problems of interest, particularly with computer-based control systems, approaches that use built-in numerical methods impose very severe restrictions on the problem description. As an example of these restrictions, consider that we wanted to impose some complex decision process in the form of IF ... THEN ... ELSE statements to determine the coefficients inside some system of equations. In general, these requirements would be in conflict with preimposed numerical methods. This is where Generalized State Space can be used to build very complex nonlinear models, without worrying about numerical methods, and the corresponding problem setup and solution convergence problems.

DISCRETE EVENT SIMULATION

Although the Generalized State Space approach has been around since the early 1980s, it has not had much coverage in the literature. Mathematics periodicals and texts describe conventional theory, and do not address this more generalized approach. Even the simulation literature is most occupied with mathematical approaches, since they are easier to study from a conventional theoretical standpoint. In the late 1980s, papers started to be published in the control theory literature describing discrete event control systems, see [10] and [11]. Today, more papers populate periodicals with articles on this discrete event mathematics.

Discrete event simulation has been heavily used in industry, particularly in manufacturing, and also in banking. It has also been used heavily in certain areas of military simulation. Discrete event simulation takes a totally different approach to handling the time scale. To describe it in the simple terms of the General Simulation System (GSS), the modeler just *schedules* subsequent processes to run based on the current process. When a process is scheduled, it is put in a time schedule for future processing. This eliminates the need for *looping* through a set of equations.

When combined with Generalized State Space, it provides an implicit solution to the nonlinear convergence problem by direct definition of cause and effect. It is an extremely elegant approach, requiring little understanding of the typical esoteric mathematical techniques required for analytical or numerical solutions. It depends principally upon the ability of the modeler to describe the mechanics of the system being analyzed using English-like rules, and relatively simple algebra. It is directly applicable to very nonlinear problems that result from decision processes. Examples are provided in the next sections.

CONSUMER GOODS EXAMPLE USING DISCRETE EVENT SIMULATION

We will now expand the consumer goods example described above to account for a number of more realistic decision processes. These changes will include the purchaser's decisions based upon his shelf space at home, and the excess amount he wishes to buy based on price of the goods. It will also account for the store manager decisions to price and order more goods to stock his inventory. With the discrete event approach, we can dispense with the dynamic form of the equations in this example, since the values of the state variables all change when specific events occur. We will now describe the decision processes of the store manager and the consumer.

STORE MANAGER'S DECISIONS AND ACTIONS

1. The order has arrived. Put new stock into inventory.

$$\text{STOCK} = \text{STOCK} + \text{ORDER}$$

2. Set price of stock. Having determined that a linear relation between price and stock makes the price too sensitive to changes in stock, a square root relation was introduced.

$$\text{PRICE} = \text{NOMINAL_PRICE} * (\text{NOMINAL_STOCK} / \text{STOCK}) ** (1/2)$$

3. Determine maximum amount of new stock to order based on storage space (MAX-STOCK) and current stock.

$$\text{MAX_ORDER} = \text{MAX_STOCK} - \text{STOCK}$$

4. Determine the order for new inventory based on prior period purchases and minimum stock to be kept on hand after next period purchases, as well as the MAX_ORDER. Note that the model assumes the next period purchases will be the same as the last period. This could easily be adjusted for greater prediction accuracy.

$$\text{ORDER} = \text{PURCHASE} + \text{MIN_STOCK} - \text{STOCK}$$

$$\text{IF ORDER IS GREATER THAN MAX_ORDER,} \\ \text{ORDER} = \text{MAX_ORDER.}$$

$$\text{IF ORDER IS LESS THAN ZERO,} \\ \text{ORDER} = 0$$

CONSUMER'S DECISIONS AND ACTIONS

1. Check shelf space left to determine maximum amount that can be purchased.

$$\text{MAX_PURCHASE} = \text{MAX_SHELF} - \text{AMOUNT_ON_SHELF}$$

2. Determine minimum amount to purchase to cover consumption for the period plus some minimum amount to leave on the shelf.

$$\text{MIN_PURCHASE} = \text{CONSUMPTION} + \text{MIN_SHELF} - \text{AMOUNT_ON_SHELF}$$

3. Make desired purchase decision based on excess purchases to be made as a function of price this period.

$$\text{EXCESS} = \text{NORMAL_EXCESS} * (\text{NOMINAL_PRICE} / \text{PRICE})$$

$$\text{PURCHASE} = \text{MIN_PURCHASE} + \text{EXCESS}$$

$$\text{IF PURCHASE IS GREATER THAN MAX_PURCHASE,} \\ \text{PURCHASE} = \text{MAX_PURCHASE.}$$

4. Go to store and make purchase.

$$\text{IF PURCHASE IS GREATER THAN STOCK,} \\ \text{PURCHASE} = \text{STOCK.}$$

$$\text{STOCK} = \text{STOCK} - \text{PURCHASE}$$

5. Stock shelf and consume

$$\text{AMOUNT_ON_SHELF} = \text{AMOUNT_ON_SHELF} + \text{PURCHASE}$$

$$\text{IF CONSUMPTION IS GREATER THAN AMOUNT_ON_SHELF}$$

$$\text{AMOUNT_ON_SHELF} = 0$$

$$\text{ELSE AMOUNT_ON_SHELF} = \text{AMOUNT_ON_SHELF} - \text{CONSUMPTION.}$$

GRAPHICAL DRAWING OF CONSUMER PURCHASE MODEL

A graphical drawing of the consumer purchase model is shown in Figure 4.1. Using the GSS CAD environment, the ovals represent the data attributes of the models. In the Generalized State Space framework, these are Generalized State Vectors. In the GSS drawing they are called *resources*. The state of the models at any instant in time is determined by the values of the data in the resources. The data contained in these resources are shown on the next page.

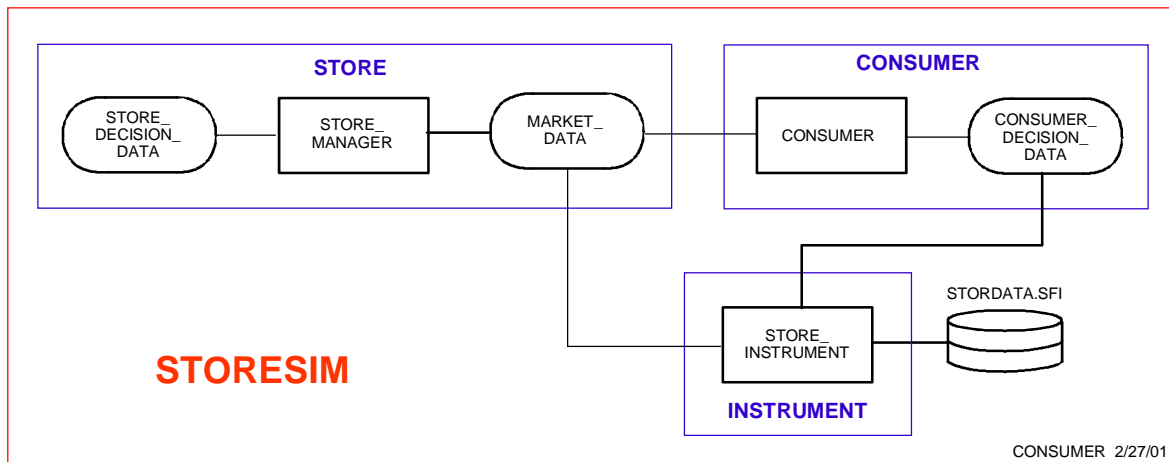


Figure 4.1. Generalized State Space models of store and consumer.

The small boxes represent Generalized State Transformations. In the GSS CAD drawing they called the *processes*. They change the state of the resources. The data and rules in the resources and processes are shown on the next three pages. Note that three asterisks *** signify that a comment follows. The results of the simulation are shown in the two figures that follow.

Figure 4.2 shows the purchases going up and down with price. Note that the cycle for purchases is actually 4 time steps because the lower value of purchase is slightly different every other time. Figure 4.3 shows the shelf values as the number of items purchased goes up and down. Note that the actual number of items consumed is less than the CONSUMPTION CONSTANT (set to 20) when the price was raised.

We also note that a linear analysis of oscillation frequency no longer applies because the model is now nonlinear. Analysis of nonlinear models is best done using parametric and sensitivity analysis. This is easily accomplished using the GSS multiple simulation run facility. This provides for running multiple simulations where each simulation can start with new parameters based upon a process that decides upon the parametric changes.

RESOURCE :	STORE_DECISION_DATA		
STOCK_FACTORS			
1	MAX_STOCK	INTEGER	
1	MIN_STOCK	INTEGER	
PRICE_FACTORS			
1	NOMINAL_PRICE	REAL	
1	NOMINAL_STOCK	INTEGER	
ORDER_DATA			
1	MAX_ORDER	INTEGER	
1	ORDER	INTEGER	

RESOURCE :	MARKET_DATA		
MARKET_ATTRIBUTES			
1	STOCK	INTEGER	
1	PRICE	REAL	
1	PURCHASE	INTEGER	
TIME_PARAMETERS			
1	MONTH	INTEGER	
1	MONTH_30	INTEGER	INITIAL_VALUE 900 ***DAYS
1	DAY_COUNT	INTEGER	

RESOURCE :	CONSUMER_DECISION_DATA		
PURCHASE_FACTORS			
1	MAX_PURCHASE	INTEGER	
1	MIN_PURCHASE	INTEGER	
1	AMOUNT_ON_SHELF	INTEGER	
1	MAX_SHELF	INTEGER	INITIAL_VALUE 35
1	MIN_SHELF	INTEGER	INITIAL_VALUE 5
1	NOMINAL_PRICE	INTEGER	INITIAL_VALUE 1
1	NOMINAL_EXCESS	INTEGER	INITIAL_VALUE 5
1	EXCESS	INTEGER	
CONSUMPTION_DATA			
1	CONSUMPTION	INTEGER	INITIAL-VALUE 20

```

PROCESS:    STORE_MANAGER

TOP_LEVEL_CONTROL
    IF MONTH IS LESS THAN 1
        EXECUTE INITIALIZE.
    DAY_COUNT = CLOCK_TIME
    MONTH = DAY_COUNT / 30
    EXECUTE NEXT_MONTH_STEPS
    SCHEDULE STORE_MANAGER IN 30 DAYS
    STOP AT MONTH_30

INITIALIZE
    MOVE ZEROS TO TIME_PARAMETERS
    NOMINAL_PRICE = 1
    PRICE = NOMINAL_PRICE
    MAX_STOCK = 60
    MIN_STOCK = 10
    NOMINAL_STOCK = 30
    STOCK = NOMINAL_STOCK

NEXT_MONTH_STEPS
    *** UPDATE STOCK WITH NEW ORDER INVENTORY
    STOCK = STOCK + ORDER

    *** SET PRICE OF STOCK
    PRICE = NOMINAL_PRICE * (NOMINAL_STOCK / STOCK)

    *** DETERMINE MAXIMUM ORDER
    MAX_ORDER = MAX_STOCK - STOCK

    *** DETERMINE NEW INVENTORY ORDER
    ORDER = PURCHASES + MIN_STOCK - STOCK
    IF ORDER IS GREATER THAN MAX_ORDER
        ORDER = MAX_ORDER.
    IF ORDER IS LESS THAN 1
        ORDER = 1.

```

```

PROCESS:      CONSUMER

TOP_LEVEL_CONTROL
    EXECUTE NEXT_MONTH_STEPS
    SCHEDULE CONSUMER IN 30 DAYS

NEXT-MONTH_STEPS
    *** DETERMINE MAXIMUM PURCHASE
        MAX_PURCHASE = MAX_SHELF - AMOUNT_ON_SHELF

    *** DETERMINE MINIMUM PURCHASE
        MIN_PURCHASE = CONSUMPTION + MIN_SHELF - AMOUNT_ON_SHELF

    *** DETERMINE DESIRED PURCHASE
        EXCESS = NOMINAL_EXCESS * (NOMINAL_PRICE / PRICE)
        PURCHASE = MIN_PURCHASE + EXCESS
        IF PURCHASE IS LESS THAN ZERO
            PURCHASE = 0.
        IF PURCHASE IS GREATER THAN MAX_PURCHASE
            PURCHASE = MAX_PURCHASE.

    *** MAKE PURCHASE
        IF PURCHASE IS GREATER THAN STOCK
            PURCHASE = STOCK.
        STOCK = STOCK - PURCHASE

    *** STOCK SHELF & CONSUME
        AMOUNT_ON_SHELF = AMOUNT_ON_SHELF + PURCHASE
        IF CONSUMPTION IS GREATER THAN AMOUNT_ON_SHELF
            AMOUNT_ON_SHELF = 0
        ELSE AMOUNT_ON_SHELF = AMOUNT_ON_SHELF - CONSUMPTION.

```

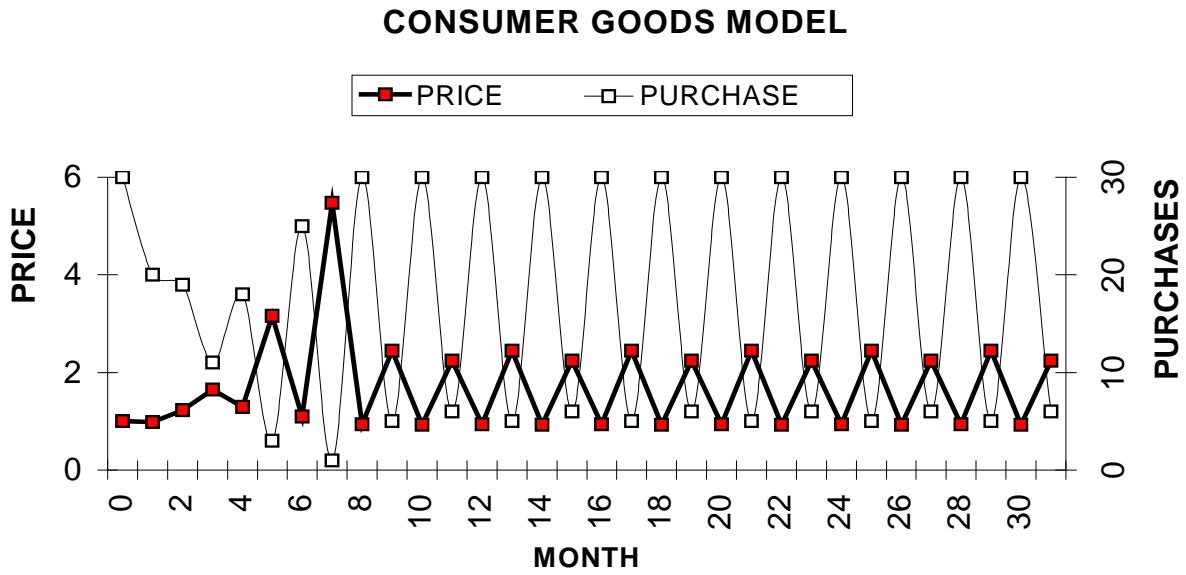


Figure 4.2. Consumer goods model output for purchases and pricing for each month.

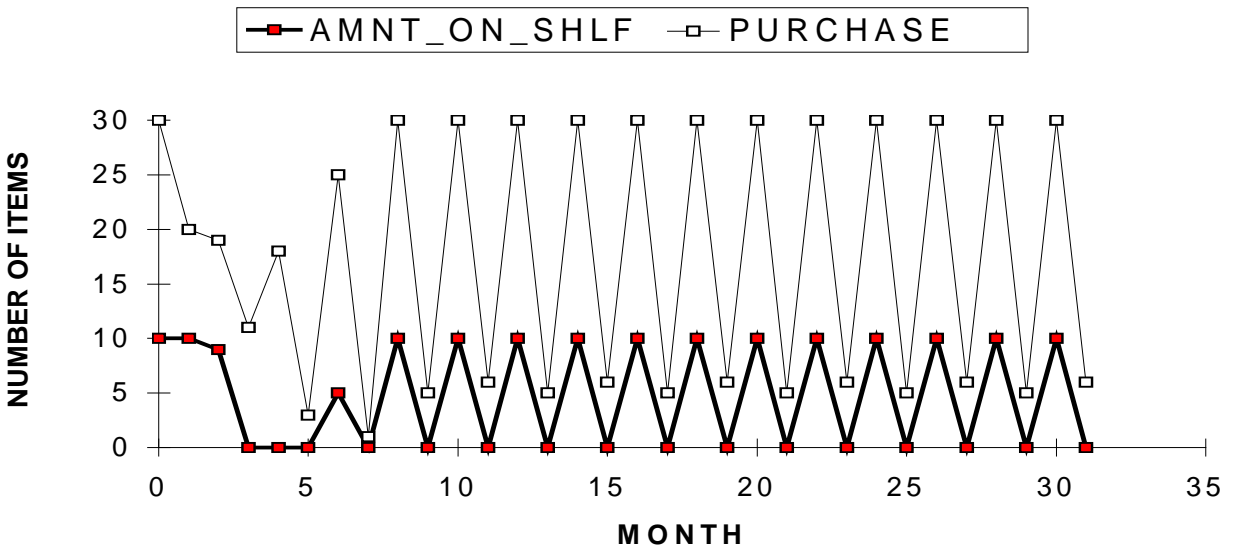


Figure 4.3. Consumer goods model output for purchases and among on shelf for each month.

5. RABBIT - COYOTE BIOLOGICAL MODEL COMPARISON

Let's now compare a continuous-time system model to its Generalized State Space counterpart. We will use the classical example of biological balance between a host and a parasite as provided in many texts, e.g., Gordon, [5], pp. 103. In this example, the dynamics of the interactivity between the rabbit and coyote populations are modeled. In this model, rabbits are the *hosts* (prey), multiplying in large numbers compared to coyotes who are the *parasites* (hunters). The equations, when simplified, take the form described by Gordon as follows.

$$\frac{dr}{dt} = A \cdot r(t) - B \cdot r(t) \cdot c(t)$$

$$\frac{dc}{dt} = K \cdot r(t) \cdot c(t) - D \cdot c(t)$$

The first equation defines the rate of change of the rabbit population, where rabbit births are a fraction, A, of the existing population, and rabbit deaths (due to coyote kills) are a fraction, B, of the product of the rabbit and coyote populations. The coyote population changes similarly, but they are modeled as the birth rate being a fraction, K, of the product of the rabbit and coyote populations, and their death rate is a fraction, D, of their population.

Figure 5.1 illustrates an approach to describing the model graphically using a fairly standard *analog diagram* for the differential equations. This set of equations can be solved using special methods or existing software systems. The analog diagram is easily related to the equations.

Figure 5.2 shows a *stock and flow diagram* for the same system, using slightly different coefficients for the equations. This diagram is somewhat more easily related to the stock and flow of rabbits and coyotes, but is harder to relate to the system of equations.

The classical approach to determining these coefficients for this problem is to assume the solution to be quasi-stable, i.e., oscillatory, with no damping to a stable state. This is justified on the grounds that oscillation is observed in real life. However, as we shall show, this is not a realistic representation of a physical system, since any perturbation will drive the system into an unstable state, causing at least one of the populations to go to infinity or zero. In fact, basically stable systems can be considered to operate in constant oscillation, even though they require continuous perturbation from an external source. One merely has to redefine the external source as part of the overall system. Any form of clock or electronic oscillator is good example. This is fine when using simple mathematical models of oscillators as examples in a classroom environment, where the complexity of nonlinear models need not be described. However, it presents a misleading picture when trying to explain the real biological behavior of interest here.

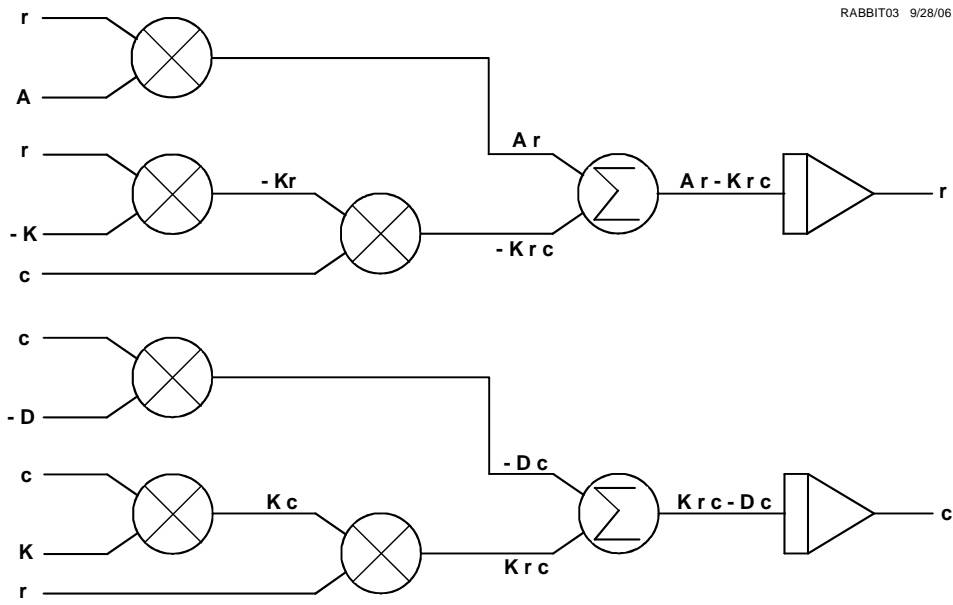


Figure 5.1. Rabbit - coyote biological model using analog symbol diagram.

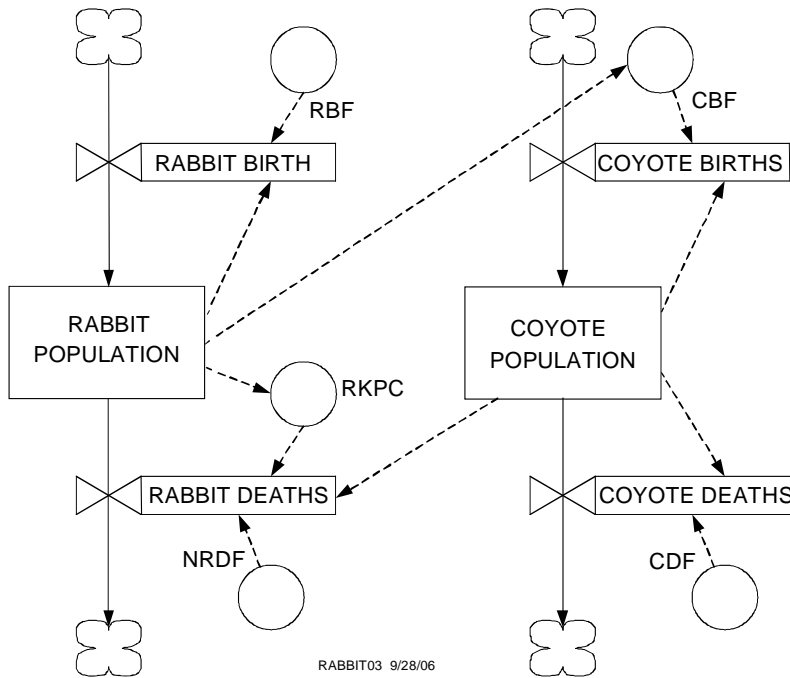


Figure 5.2. Rabbit - coyote biological model using system dynamics symbol diagram.

A MORE REALISTIC MODEL OF THE PHYSICAL PHENOMENON

The theory and design of electronic oscillators has been well researched. Their operational characteristics are governed by nonlinear physical phenomenon, refer to Hafner, [14]. Accurate representation of real physical oscillatory behavior requires nonlinear models. In addition, damping exists in physical systems to some degree, as do external perturbations. When the perturbations are absent, the system will relax, with decreasing oscillatory behavior, to a stable state - normally not oscillatory because of the effects of damping. When a perturbation occurs, the system moves from its stable state. These perturbations can come close enough together to cause superposition of their effects, and give the appearance of continual oscillatory motion. We submit that is the case with the typical biological model.

The model of a system that is less than critically damped will show the same form of oscillatory responses every time it is perturbed. Clearly, biological systems such as rabbits and coyotes are always being perturbed by external factors not modeled here. These can produce what would appear to be continuous oscillation, even though the systems themselves are highly stable. These additional perturbations would hardly change the overall behavior of the system. Depending on how one chooses the coefficients in the nonlinear equations, vastly different results can occur. One must study the effects of perturbations on the populations to gain good agreement with reality.

Given these facts, both of the linear models described above can misrepresent the real physical behavior of the biological system, particularly if one is concerned about studying the survival of the populations. It is more realistic to represent coyote deaths as due to starvation (not enough rabbit kills) and natural causes. Similarly, given reasonable circumstances, the rabbit population in a linear model quickly grows to infinity - an impossible consequence if we are studying a finite geographical area, with its finite food supply.

Rabbits will also starve if they don't have enough food, and can die of natural causes as well. Also, the incubation periods of different animals can be significantly different, affecting the time constants for birth after pregnancy. As we add these more detailed representations to gain accuracy, the model will necessarily become more complex. But, instead of solving a set of equations for fictitious coefficients to land on the single point of oscillation, we are providing real characterizations, observed phenomena and watching the simulated results. Furthermore, both of the prior approaches require a knowledge of how to transform the description of a physical problem into differential equation format. Our Generalized State Space approach dispenses with this requirement. The GSS description appears in a natural language format, requiring only a knowledge of algebra.

A RABBIT-COYOTE BIOLOGICAL MODEL IN GENERALIZED STATE SPACE

Figure 5.3 below shows a GSS version of the rabbit-coyote simulation. It uses the same biological type model, but it is now based upon Generalized State Space. This approach is totally different than that using differential equations described in the prior section.

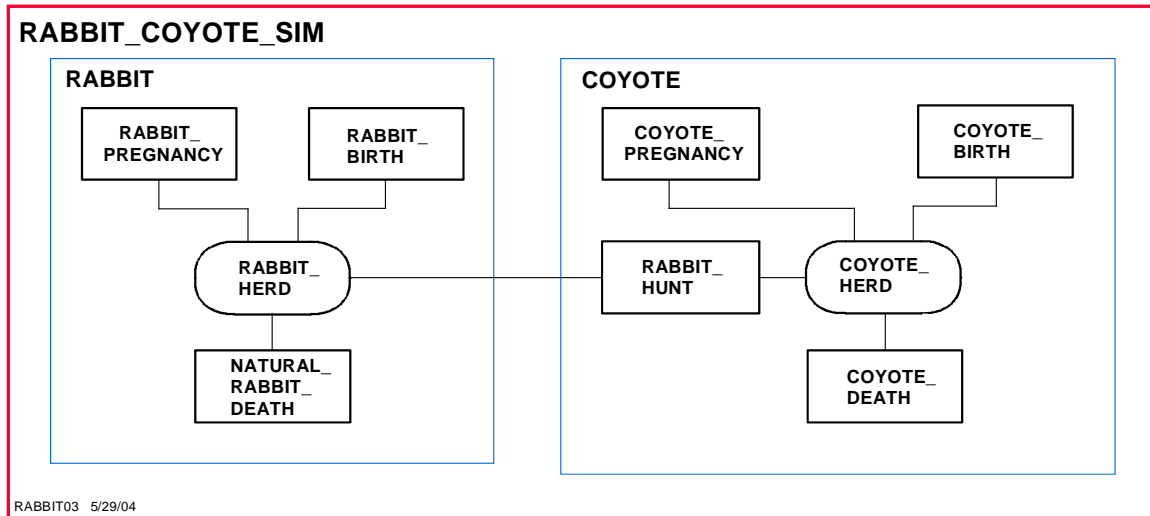


Figure 5.3. Rabbit - coyote biological model using the GSS Model Development Graphics.

The GSS Processes and Resources are shown on the next three pages. Although not many decision processes are represented in these simple models, the approach to the computations is more understandable in terms of real life considerations. One does not have to understand differential equations to represent the physical system. However, the models account for much more than their counterparts in the prior section. Equally important, the counterpart of nonlinear differential equations would be much more complex to write and solve.

Graphs of the dynamic behavior of the possible rabbit - coyote relationship, as represented in GSS, are shown in Figures 5.4 through 5.7. These results were obtained by modifying the birth, death, and hunger submodels for each. It can be seen that a wide range of results can be obtained, depending upon various factors. Figures 5.4 and 5.5 show the result of a single perturbation, at the beginning of the chart, on this very stable system. We note that a sequence of perturbations, *occurring two years apart*, will give the appearance of continuous oscillation. We would expect perturbations to occur at least this often.

RESOURCE: RABBIT_HERD

RABBIT

1	POPULATION	INTEGER	INITIAL_VALUE	10000
1	PREGNANCY_SET	INDEX	INITIAL_VALUE	1
1	PREGNANCIES	QUANTITY(9)	INTEGER	
1	NATURAL_DEATHS	INTEGER		
1	TOTAL_DEATHS	INTEGER		
1	HUNGER_DEATHS	INTEGER		
1	HUNGER_DEATH_FACTOR	REAL		
1	DEATHS_BY_COYOTE	INTEGER		
1	REPRODUCTION_RATE	REAL	INITIAL_VALUE	0.2
1	NATURAL_DEATH_RATE	REAL	INITIAL_VALUE	0.05
1	MEDIAN_POPULATION	INTEGER	INITIAL_VALUE	10000

PROCESS: RABBIT_PREGNANCY

PREGNANCY_CONTROL

```
***      DETERMINE PREGNANCY GROUP (MONTH - INSTANCE)
      IF PREGNANCY_SET IS GREATER THAN 2
          PREGNANCY_SET = 1.
***      COMPUTE NUMBER OF PREGNANCIES FOR THIS PERIOD
      PREGNANCIES (PREGNANCY_SET) = REPRODUCTION_RATE * POPULATION

***      SCHEDULE BIRTH AND PREGNANCY DATES
      SCHEDULE RABBIT_BIRTH IN 60 DAYS USING PREGNANCY_SET
      SCHEDULE RABBIT_PREGNANCY IN 30 DAYS USING PREGNANCY_SET
      INCREMENT PREGNANCY_SET.
```

PROCESS: RABBIT_BIRTH

RABBIT_BIRTH_CONTROL

```
ADD PREGNANCIES(PREGNANCY_SET) TO POPULATION
```

```

PROCESS: NATURAL_RABBIT_DEATH

RABBIT_DEATH_CONTROL
  IF POPULATION IS GREATER THAN ZERO
    HUNGER_DEATH_FACTOR = ((MEDIAN_POPULATION + POPULATION)/
                          MEDIAN_POPULATION)**2
    TOTAL_DEATHS = NATURAL_DEATH_RATE * POPULATION *
                  HUNGER_DEATH_FACTOR
    SUBTRACT TOTAL_DEATHS FROM POPULATION.
  IF POPULATION IS LESS THAN 2
    STOP.
  SCHEDULE NATURAL_RABBIT_DEATH IN 30 DAYS

```

```

RESOURCE: COYOTE_HERD

COYOTE
  1 POPULATION                INTEGER INITIAL_VALUE 250
  1 PREGNANCY_SET              INDEX   INITIAL_VALUE  1
  1 PREGNANCIES QUANTITY(9)    INTEGER *** ALLOW UP TO 9 INSTANCES
  1 NATURAL_DEATHS             INTEGER
  1 HUNGER_DEATHS              INTEGER
  1 TOTAL_DEATHS               INTEGER
  1 RABBIT_KILLS               INTEGER
  1 HUNGER                     REAL
  1 COYOTE_RABBIT_RATIO        REAL
  1 REPRODUCTION_RATE          REAL   INITIAL_VALUE 0.1
  1 NATURAL_DEATH_RATE         REAL   INITIAL_VALUE 0.02
  1 HUNGER_DEATH_RATE          REAL   INITIAL_VALUE 0.02
  1 APPETITE                   INTEGER INITIAL_VALUE 20
  1 PROBABILITY_OF_CATCH       REAL
  1 MEDIAN_RABBIT_CATCH        INTEGER INITIAL_VALUE 10000

TIME_FACTORS
  1 DAY_COUNT                  INTEGER
  1 MONTH                      INTEGER

```

```

PROCESS: COYOTE_PREGNANCY

PREGNANCY_CONTROL

  *** DETERMINE_PREGNANCY SET (MONTH - INSTANCES)
  IF PREGNANCY_SET IS GREATER THAN 3
    PREGNANCY_SET = 1.

  *** SET COYOTE PREGNANCIES AND SCHEDULE BIRTH
  PREGNANCIES(PREGNANCY_SET) = REPRODUCTION_RATE * POPULATION
  SCHEDULE COYOTE_BIRTH IN 90 DAYS USING PREGNANCY_SET
  SCHEDULE COYOTE_PREGNANCY IN 30 DAYS USING PREGNANCY_SET
  INCREMENT PREGNANCY_SET.

```

```
PROCESS: COYOTE_BIRTH

COYOTE_BIRTH_CONTROL
  ADD PREGNANCIES(PREGNANCY_SET) TO POPULATION
```

```
PROCESS: COYOTE_DEATH

COYOTE_DEATH_CONTROL
  IF POPULATION IS GREATER THAN ZERO EXECUTE
    COMPUTE_COYOTE_DEATHS.
  IF POPULATION IS LESS THAN 2
    STOP.
  SCHEDULE COYOTE_DEATH IN 30 DAYS

COMPUTE_COYOTE_DEATHS
  *** DETERMINE COYOTE HUNGER
  COYOTE_HUNGER = 3*(5*COYOTE_POPULATION/RABBIT_KILLS) ** 3
  NATURAL_DEATHS = NATURAL_DEATH_RATE * POPULATION
  HUNGER_DEATHS = COYOTE_HUNGER * HUNGER_DEATH_RATE * POPULATION
  TOTAL_DEATHS = NATURAL_DEATHS + HUNGER_DEATHS
```

```
PROCESS: RABBIT_HUNT

RABBIT_HUNT
  *** DETERMINE RABBIT_KILLS
  PROBABILITY_OF_CATCH = (RABBIT_POPULATION /
    (RABBIT_POPULATION + MEDIAN_RABBIT_CATCH)) ** 2
  RABBIT_KILLS = APPETITE * COYOTE_POPULATION * PROBABILITY_OF_CATCH
  DECREMENT RABBIT_POPULATION BY RABBIT_KILLS
  IF RABBIT_POPULATION IS LESS THAN 2
    STOP.
  IF RABBIT_KILLS ARE LESS THAN ZERO
    RABBIT_KILLS = 1.
  SCHEDULE RABBIT_HUNT IN 30 DAYS
```

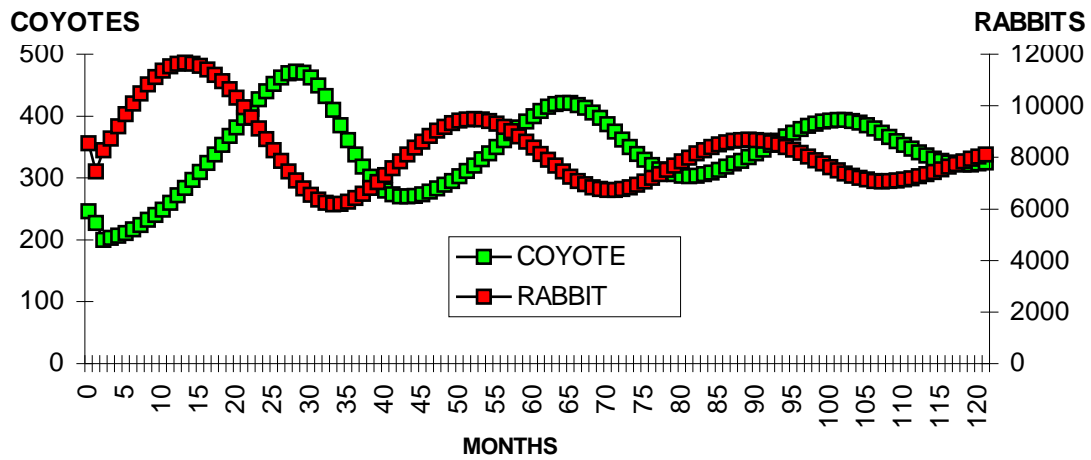


Figure 5.4 Oscillatory relationship when basic model is linear and rabbits are limited by food.

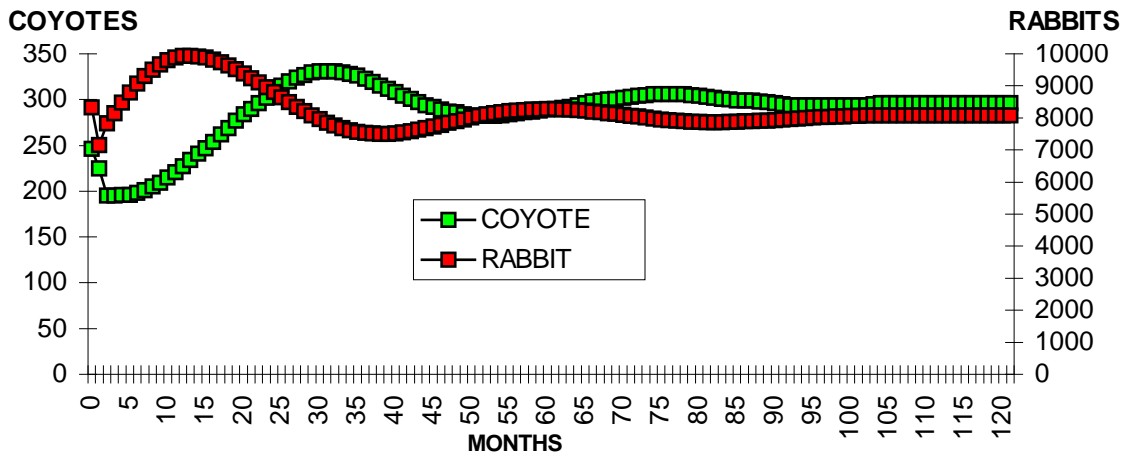


Figure 5.5 Stable relationship when basic model is nonlinear and rabbits are limited by food.

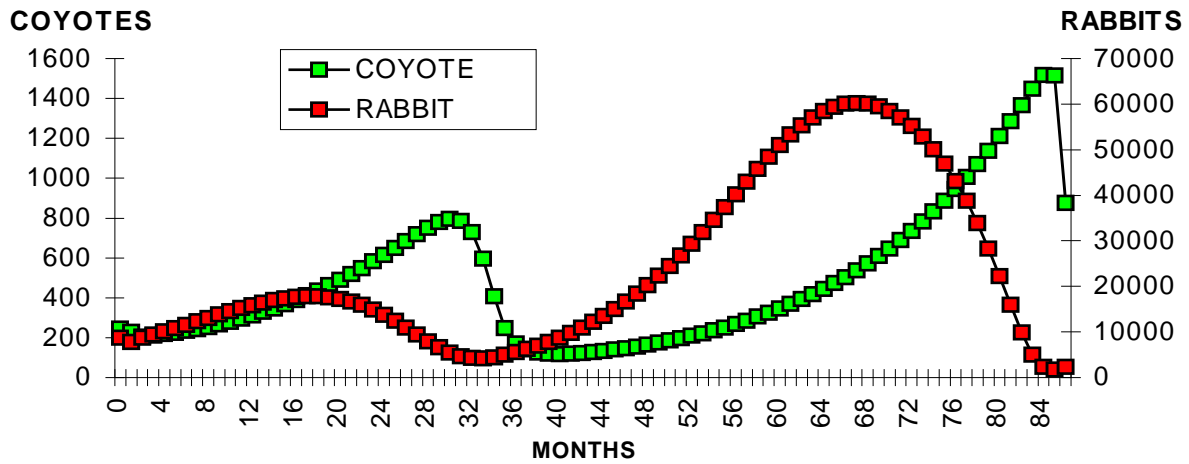


Figure 5.6. Unstable relationship when model is linear and rabbit growth is not limited.

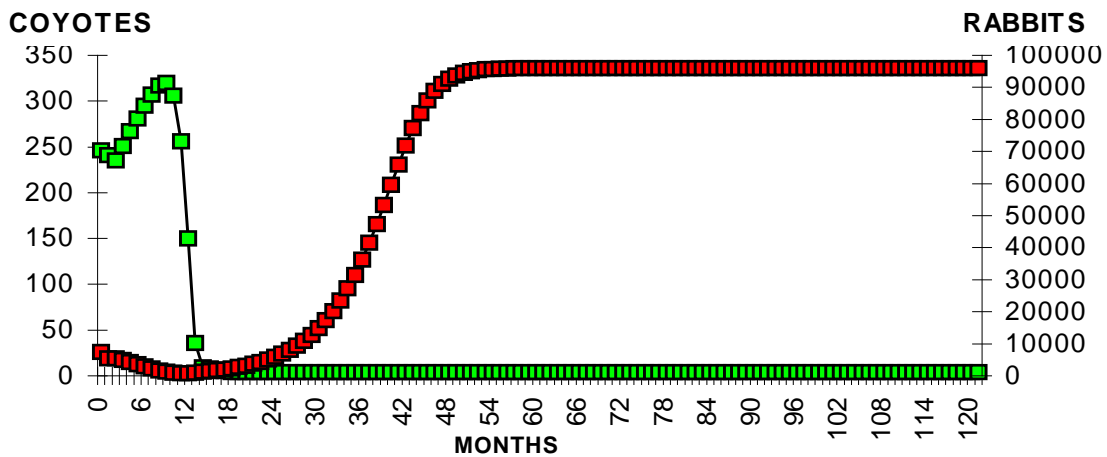


Figure 5.7. Extinction occurs when model is linear and rabbit growth is slow and limited.

6. MODELING DISTRIBUTED RESPONSES TO EVENTS

When modeling populations of elements of nature, one must face the fact that all elements or individuals do not produce the same response to an event, and if they do, it is not produced at the same time. Instead, responses are typically characterized by distributions in time and state space. For example, if the electrical power is lost in a given populated area, large numbers of people will start trying to communicate with police, neighbors, radio stations, relatives, etc. Their response to this well defined event produces a distribution of follow-on events that can be quite varied in their actions as well as the time of occurrence.

It is this very behavior that produces inherent predictability in a system. However, we must be able to model these types of responses accurately as they directly affect the accurate prediction of behavior of such a population. We can do this quite easily using GSS.

We will start with an example of distributed responses to a sequence of events to show how the resulting cumulative response can be modeled quite accurately. This is useful in predicting responses to events that occurred many timesteps in the past. To demonstrate this, we will build a model of housing unit completions as function of the event of taking out a building permit. We will make a number of simplifying assumptions to start, and then build a more accurate model.

HOUSING UNITS COMPLETED AS A FUNCTION OF BUILDING PERMITS

We would like to predict the number of housing completions in a given geographical area months in advance. These predictions could then be used to predict sales of appliances purchased after a house is complete, or phone systems, furniture, carpeting, etc. Actual completions can be measured by certificates of occupancy issued in a given month. Probably the most significant factor in predicting housing completions is building permits. These are normally taken out many months prior to completion. We start with an analysis of one month's worth of building permits to determine the resulting distribution of completions for those permits. Let's assume that our investigation yielded an average distribution that took on a shape as shown in Figure 6.1. Then we could model the resulting distribution as shown where the number of housing units in the distribution equaled the building permits taken out (or some percentage if all did not result in completions).

Figure 6.2 shows the superposition of housing completions due to building permits taken out in months 3 and 10. Figure 6.3 shows the predicted housing completions as a function of building permits starting in month 1. Note the time it takes for a build up of the proper memory of housing completions due to building permits in the past. If this were an actual prediction, then the result would not be valid until the results of any permits in months prior to month 1 were washed out. With the distribution shown, this would take a total of 15 months. The data beyond that point would not be affected by anything before month 1.

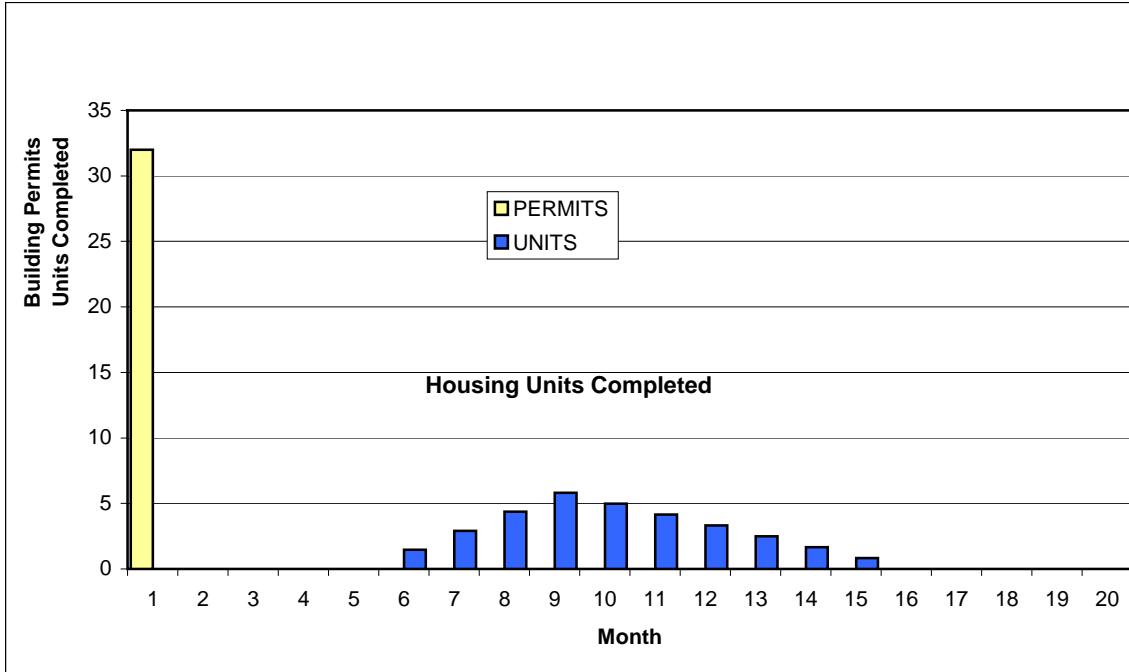


Figure 6.1. Housing units completed in months 6 through 15 as a result of building permits in month 1.

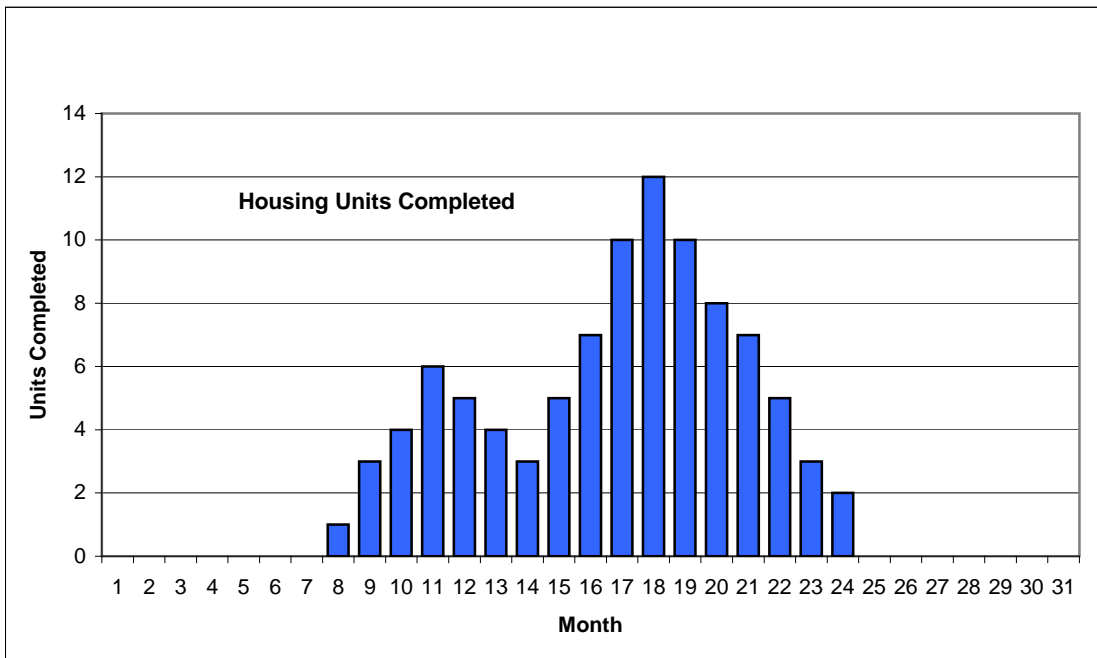


Figure 6.2. Housing units completed in months 8 through 26 as a result of building Permits in taken out in months 2 and 11.

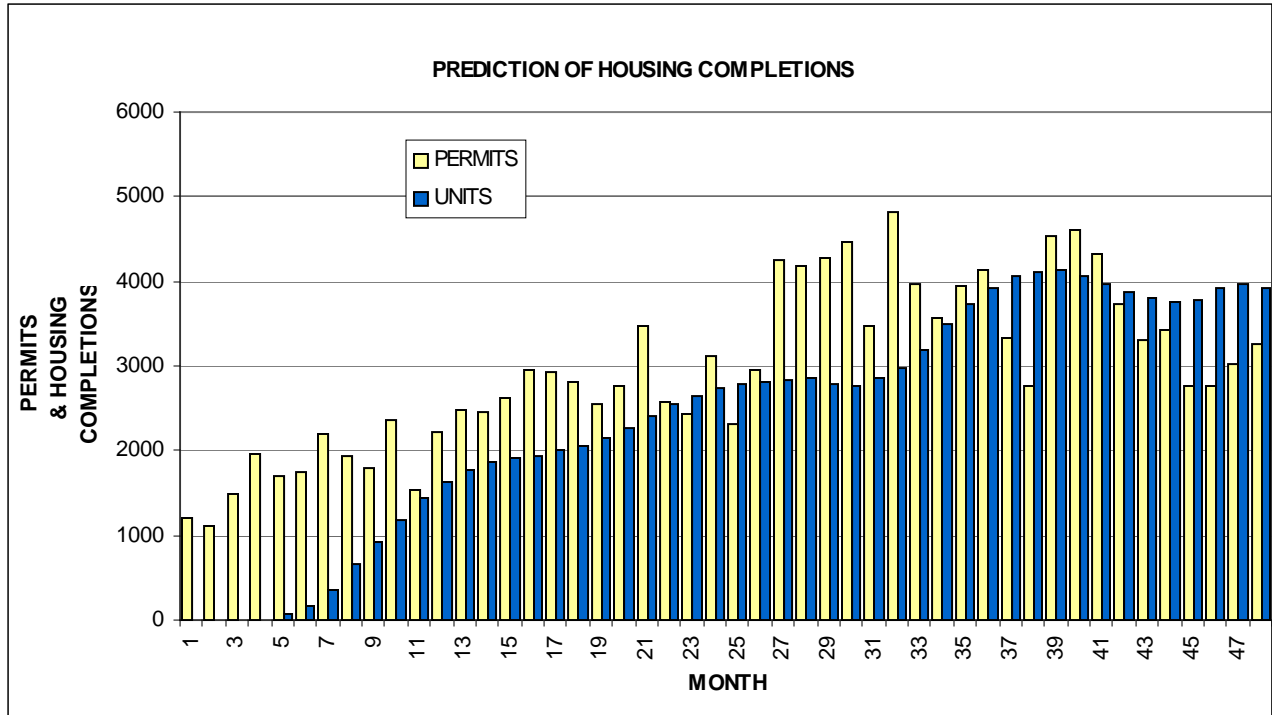


Figure 6.3. Housing units completed a result of building permits starting in month 1.

If one were to plot the correlation between housing completions and building permits as a function of time for this model, it would peak around 10 months. This is because the mean of the distribution falls at about his point. Note also that, if the distribution function accurately represented the actual housing completions, there would by no error in predictions up to five months out because the unknown quantities of building permits taken out after the current time would not affect housing completions during this period. Obviously, there are other factors that would affect housing completions, and we will look to incorporate those into our model.

The GSS model in Figure 6.4 contains simple submodels that produce the prediction of housing unit completion. This model is based on a single external factor, namely building permits, and the distributed response function described above. This information is taken from the input file BLDGPRMT.SFI in Standard File Interface (SFI) format. SFI format is described in Chapter 14 of the GSS Users Reference Manual. The contents of this file are shown in Figure 6.5. The output file HOUSING.SFI is also in SFI format, and shown in Figure 6.6. SFI formatted files can be converted directly to EXCEL for ease of producing the charts shown in Figures 6.1 to 6.3.

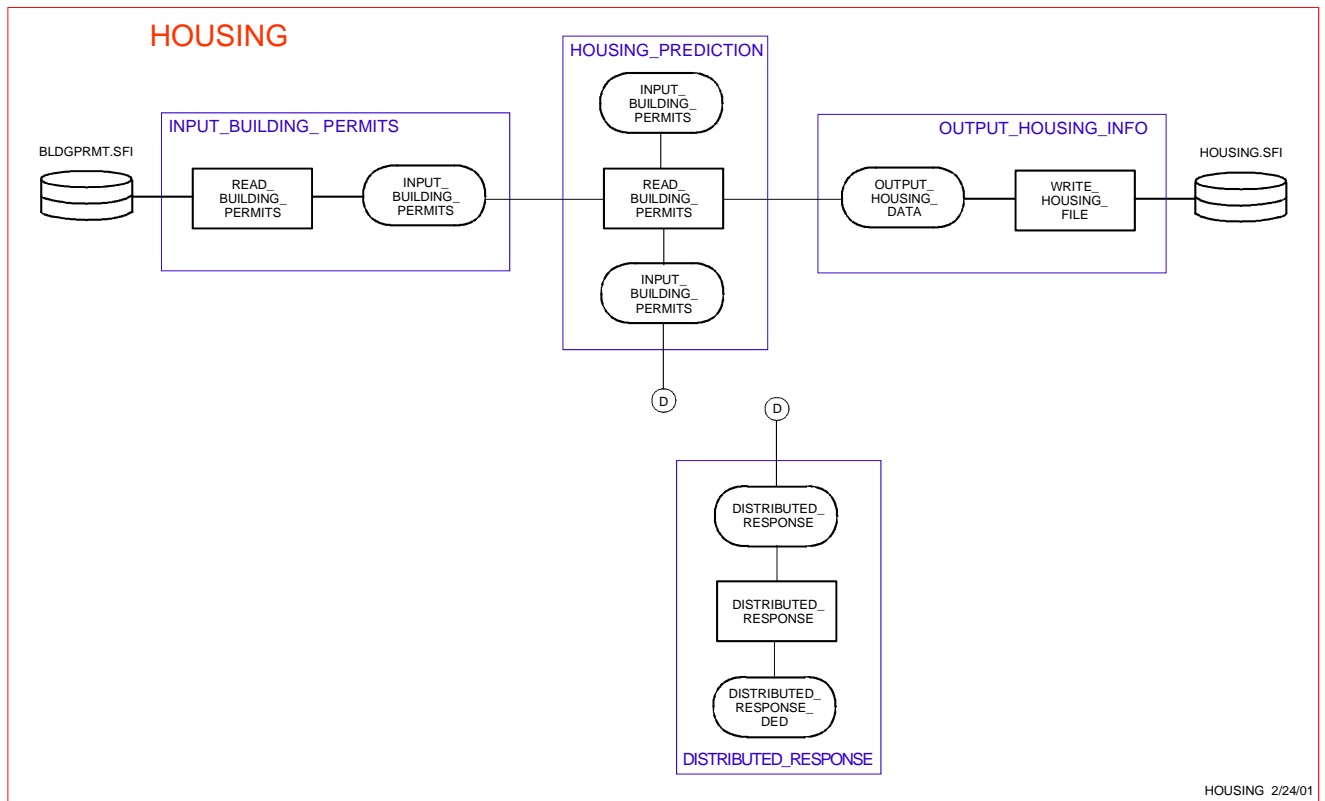


Figure 6.4. Drawing of the HOUSING task and models.

```

* BUILDING PERMIT INPUT FILE
*
*****
*
* SFI HEADER FOLLOWS
TERMINATOR = SPACE
*
*****
* SFI FORMAT RECORDS FOLLOW
NAME = SAMPLE_NUMBER      INTEGER
NAME = BUILDING_PERMITS   INTEGER
*
*****
1      1195
2      1113
3      1483
4      1954
5      1697
6      1752
7      2200
8      1939
9      1804
.
.
.
47     3031
48     3268

```

Figure 6.5. Input file - BLDGPRMT.SFI - for the HOUSING model.

```

* HOUSING OUTPUT FILE
*
*****
*
* SFI HEADER FOLLOWS
TERMINATOR = SPACE, SPACES = 2
*
*****
* SFI FORMAT RECORDS FOLLOW
NAME = SAMPLE_NUMBER      INTEGER 3
NAME = MONTH              INTEGER 3
NAME = BUILDING_PERMITS   INTEGER 6
NAME = HOUSING_UNITS      INTEGER 6
*
*****
1      1      1195      0
2      2      1113      0
3      3      1483      0
4      4      1954      0
5      5      1697      59
6      6      1752      175
7      7      2200      364
8      8      1939      651
9      9      1804      924
.      .      .      .
.      .      .      .
.      .      .      .
47     47     3031     3973
48     48     3268     3935

```

Figure 6.6. Output file - HOUSING.SFI - from the HOUSING model.

7. GSS - OIL TANKER DISTRIBUTION - SIMULATION

OVERVIEW

The Oil Tanker Distribution Simulation provides an example of how the General Simulation System (GSS) can be used to build models of decision systems that have flow controls as submodels. These are called mixed-mode models because, in the past, they have been modeled using both discrete and continuous simulation approaches. The GSS Generalized State Space approach allows the modeler to combine both approaches very easily, without any special considerations.

In this problem, a fleet of tankers loads up with oil at a loading dock in Alaska and travels to Washington to unload at a storage facility. The following basic specifications are provided.

TANKER LOADING FACILITY

All tankers can load simultaneously.

Tanker loading time is uniformly distributed between 2.9 and 3.1 days.

UNLOADING DOCK

Only one tanker can unload at a time, and only during specified hours.

Unloading time starts at 0600 and can run to 2400 hours each day.

STORAGE TANK FACILITY

The storage tank can accept oil from a tanker at the rate of 300,000 bbl per day. This is translated to 3000 bbl/100th of a day.

The storage tank supplies oil to a refinery at the rate of 150,000 bbl per day. This is translated to 1500 bbl/100th of a day.

If the level of oil in the tank falls below 5000 bbl, supply to the refinery is stopped until the tank fills up to 50,000 bbl.

When the level of the tank exceeds a capacity of 2,000,000 bbl, flow from the tanker is halted until the level of the tank falls below 1,600,000.

TANKER FLEET ATTRIBUTES

The fleet size is normally 15 tankers.

Each tanker has a capacity of 150,000 bbl of oil.

When the level of oil in a tanker falls below 7500 bbl, the tanker is considered empty, and flow into the storage facility is shut off.

Travel time from Alaska loaded is normally distributed with a mean of 5 days and standard deviation of 1.5 days.

Travel time from Washington unloaded is normally distributed with a mean of 4 days and standard deviation of 1 day.

INITIAL CONDITIONS

Tankers arrive at the loading facility at half-day intervals starting at time 0.

The storage tank is half full (1,000,000 bbl).

GSS SIMULATION

Figure 7.1 provides the architecture of the GSS model. The GSS Simulation Control Specification, Resources, and Processes are attached. Information above the dashed line in these descriptions pertains to the model architecture. This information is entered in the architecture environment of GSS. This information corresponds to the drawing in Figure 1.

The information below the dashed line is entered in the language environment. This environment provides the simulation control specification language, the resource language, and the process language.

GSS SIMULATION CONTROL SPECIFICATION

A single simulation is called for, although multiple reruns could be done automatically while changing parameters, also automatically, e.g., number of tankers, dock hours, etc. Optimization could also be run, although this problem is quite simple to analyze without it.

Three processes must be started independently at the beginning of the simulation, namely INSTRUMENT, INITIALIZE_TANKER, AND STORAGE_CONTROLLER, in that order.

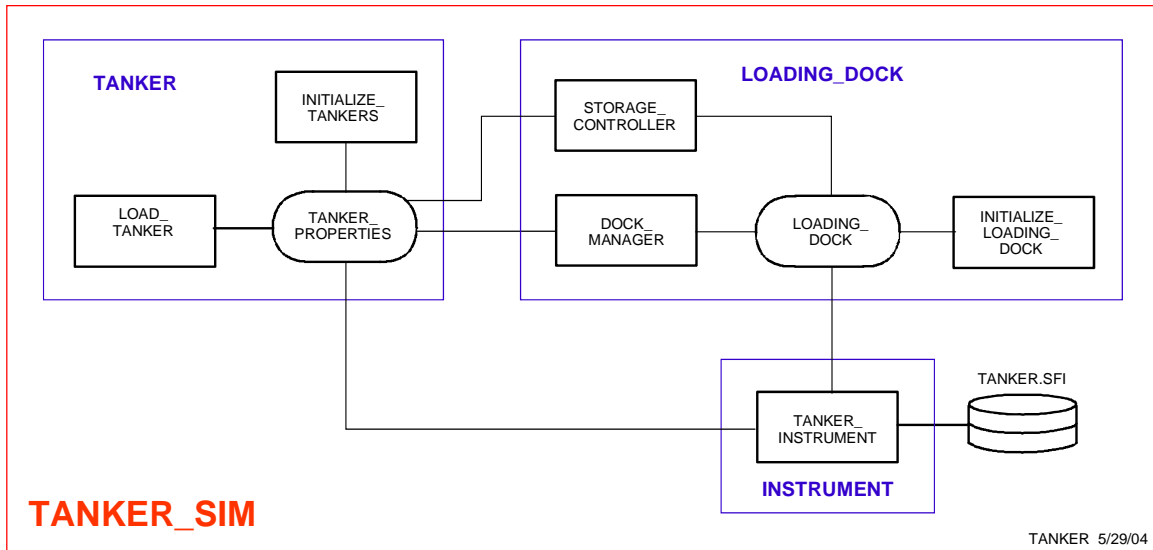


Figure 7-1. GSS Oil tanker distribution model.

GSS MODELS

TANKER MODEL

Resource: TANKER_PROPERTIES

Contains the attributes of the tankers.

Process: INITIALIZE_TANKERS

This process prompts for user inputs, i.e., number of tankers, and number of simulation days. It initially puts the tankers at the loading dock at half hour intervals, starting at time zero as specified.

Process: LOAD_TANKER

This process loads the tankers and schedules them to arrive at the unloading dock based upon the loaded travel time distribution given.

LOADING_DOCK MODEL

Resource: **LOADING_DOCK**

Contains the attributes pertinent to the unloading dock, including the storage properties.

Process: **INITIALIZE_LOADING_DOCK**

This process prompts for the user input - dock closing time. This can be 6:00 pm or 12:00 midnight. It also schedules the storage controller the first time.

Process: **DOCK-MANAGER**

The dock manager provides three functions: (1) starts tankers unloading; (2) sends off tankers that have completed unloading, scheduling them to arrive at the loading dock based upon the loaded travel time distribution given; (3) queues up tankers at the unloading dock while a single tanker unloads. Tankers coming in go to the end of the line, being queued up to unload on a first in-first out basis.

Process: **STORAGE_CONTROLLER**

This process implicitly solves the continuous flow problem using simple sampling rules. The time step can be selected by the user - we chose one-hundredth of a day. This appears to be very fine grained, and could probably be cut by a factor of two or more based upon a twenty-four day fine grain analysis done by PSI. This would cut simulation running time directly, currently at about 20 seconds. When solving differential equations on a digital computer, one must decide whether or not to use variable step size integration. In this problem, the decision processes and other considerations clearly make the use of variable step size integration undesirable, as well as unnecessary. Accuracy of the simulation results clearly depends upon many more important factors involving accurate representation of decision processes and other considerations. For example, one should take into account the time it takes to change tankers. The questions to be answered relative to the cost of running this operation effectively are best done using the rule based approach shown here, where anyone can understand how the models represent the system.

Flow from the tanker or to the refinery is modeled in terms of bbl per 1/100 of a day. This time step is sufficiently small to insure that the oil levels in the tank and the tanker are monitored accurately. One could build in further accuracy by subtracting any excess fractions due to the finite time step exactly. However, this model appears more accurate than is necessary, especially considering other factors that could produce errors that would easily be a factor of ten larger.

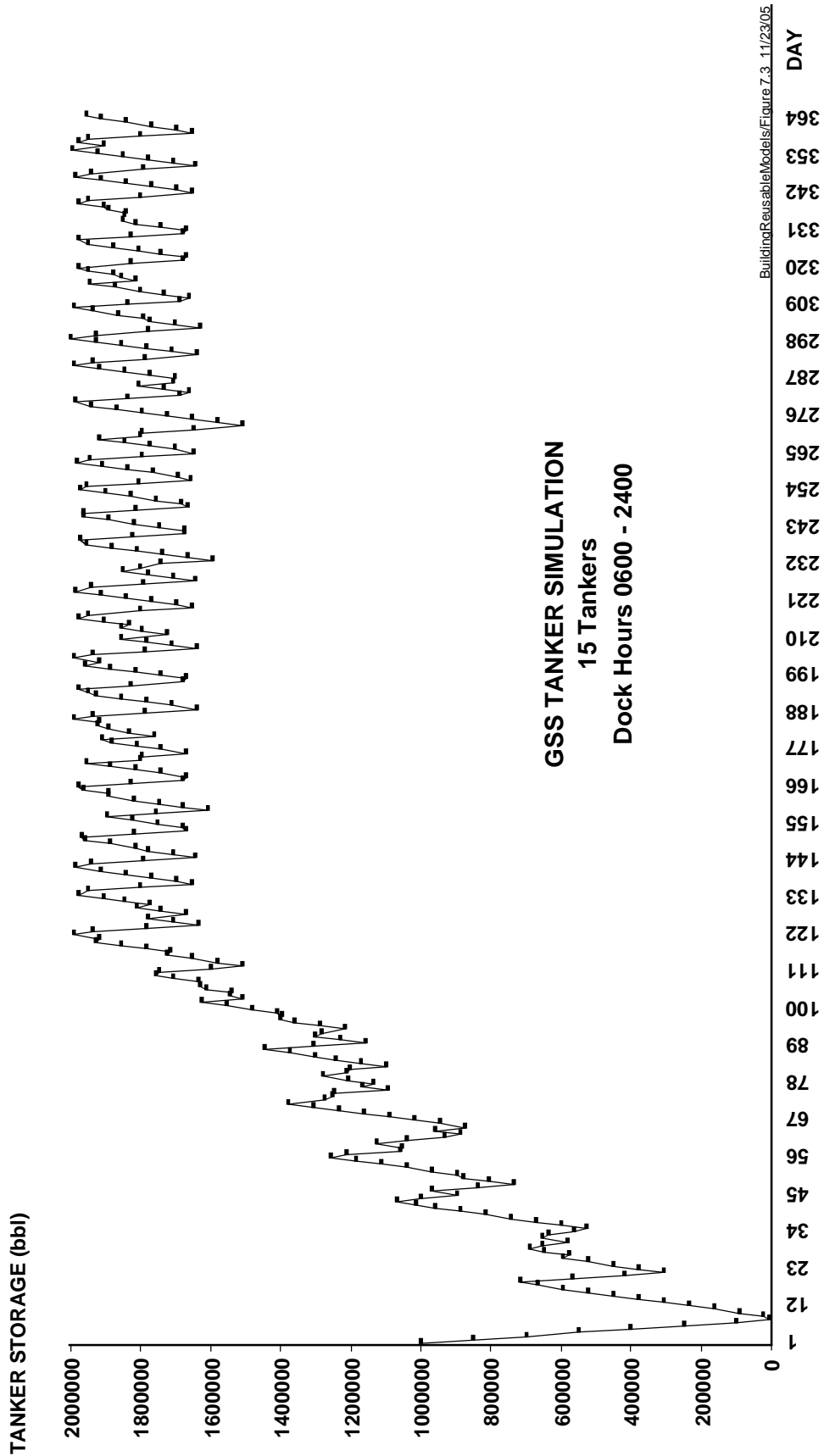
INSTRUMENT MODEL

Process: TANKER_INSTRUMENT

This process collects the desired data and puts it on an SFI formatted file as shown in Figure 7-2 below. This file contains the day number, the amount of oil in the storage tank, the number of tankers in the queue at the unloading dock, and the transit time for both full and empty tankers for each day. This allows users to use EXCEL or LOTUS to plot their results or perform various statistical analyses. The results of using EXCEL are shown in the statistical plots in Figures 7-3 through 7-6.

```
* TANKER.SFI OUTPUT FILE
*
*****
*
* SFI HEADER FOLLOWS
TERMINATOR = NONE
*
*****
* SFI FORMAT RECORDS FOLLOW
NAME = NUMBER_OF_DAYS      INTEGER  5
NAME = AMOUNT_IN_STORAGE   INTEGER  7
NAME = TANKER_COUNT        INTEGER 12
NAME = CROSSING_TIME_FULL  FLOAT 12,2
NAME = CROSSING_TIME_EMPTY FLOAT 12,2
*
*****
* DAYS   STORAGE   TANKER_QUEUE  CROSSING_FULL  CROSSING_EMPTY
0       1000000    0             .00            .00
1       850000    0             4.38           .00
2       700000    0             4.47           .00
3       550000    0             5.08           .00
4       400000    0             5.70           .00
5       250000    0             4.74           .00
6       100000    0             3.91           .00
7        4000     0             1.94           .00
8       22000    0             5.33           .00
9       91000    1             5.33           4.66
10      163000    2             5.33           3.54
.       .         .             .             .
.       .         .             .             .
.       .         .             .             .
360    1699000    2             6.00           3.18
361    1771000    2             6.00           4.22
362    1843000    0             6.00           4.68
363    1915000    0             6.00           2.69
364    1954000    0             7.00           2.81
```

Figure 7-2. Output file - HOUSING.SFI - from the HOUSING model.



BuildingReusableModels\Figure 7.3_11/23/05

Figure 7-3

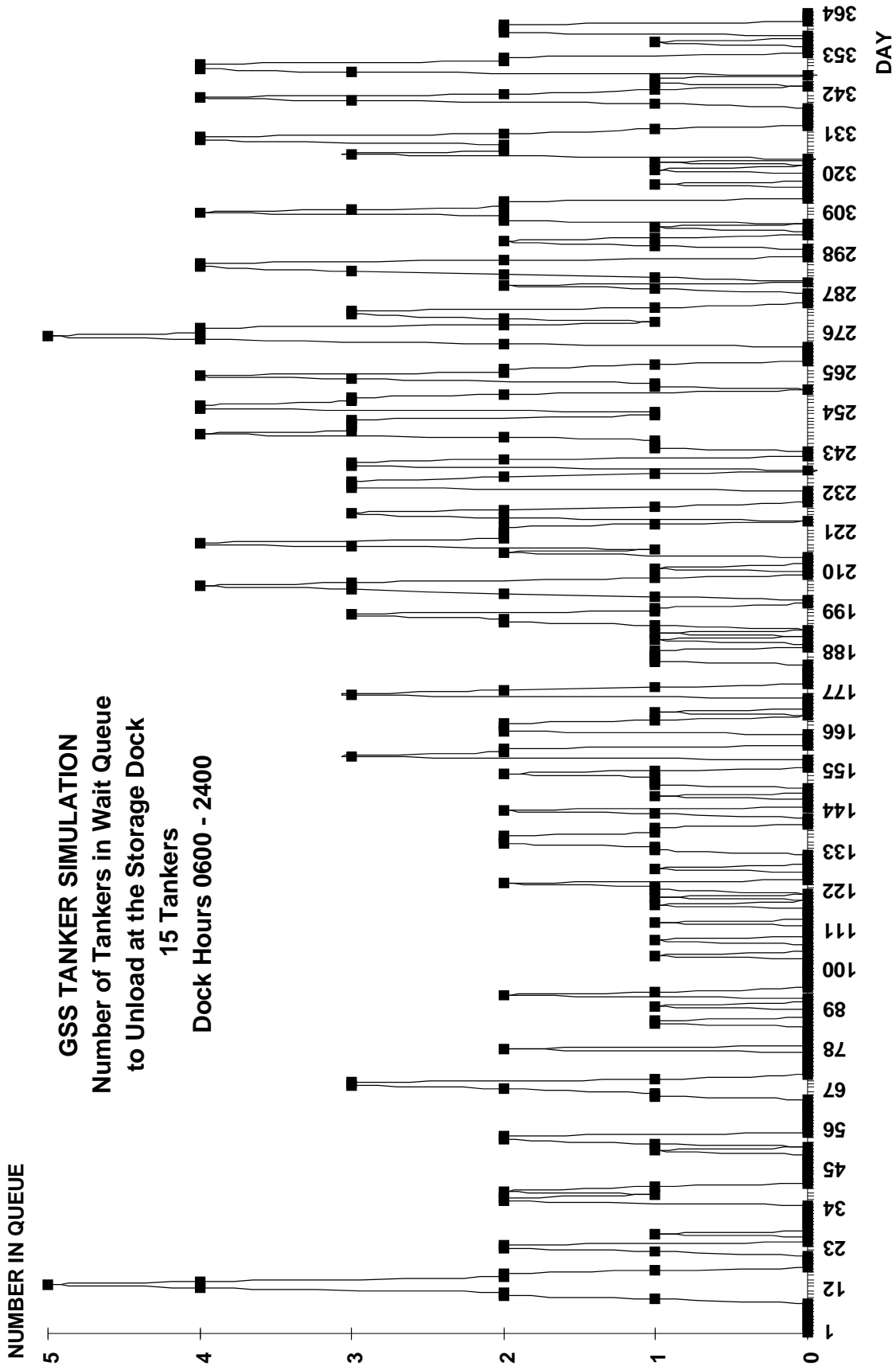


Figure 7-4. Number of tankers in queue at storage dock.

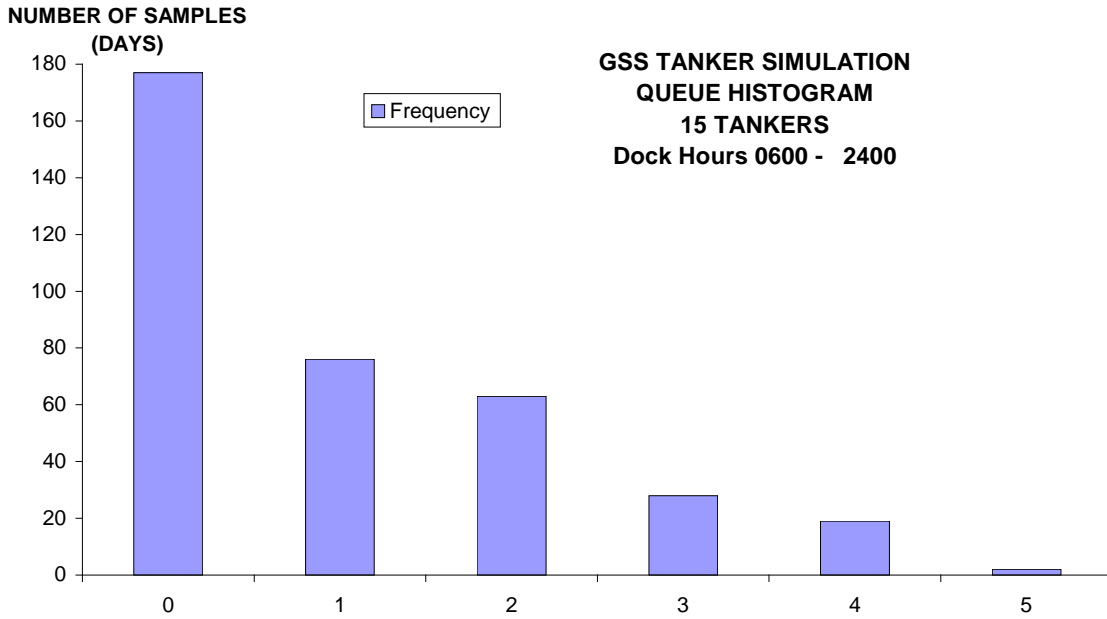


Figure 7-5. Histogram of tanker queue at storage dock.

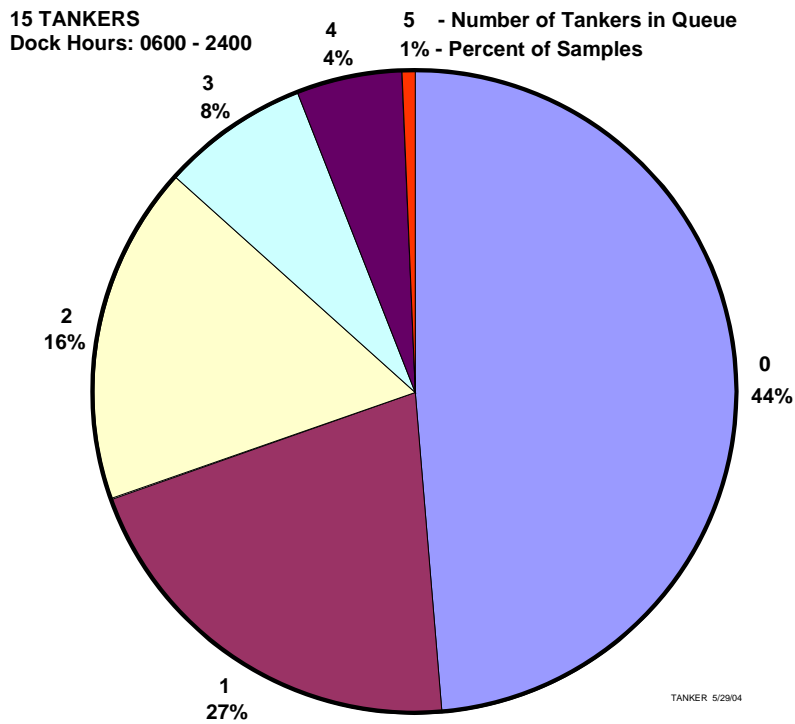


Figure 7-6. Pie chart of percent of time a given number of tankers are in the queue.

RESOURCE:

TANKER_ATTRIBUTES

1	NUMBER_OF_TANKERS	INDEX	INITIAL_VALUE	15		
1	TANKER	INDEX				
1	CAPACITY	REAL	INITIAL_VALUE	150000	***	BBL
1	CROSSING_TIME_FULL	REAL				
1	CROSSING_TIME_EMPTY	REAL				
1	SCHEDULED_TIME	REAL	INITIAL_VALUE	0		
1	TIME_TO_LOAD	REAL				
1	TIME_TO_ARRIVE	REAL				
1	SIMULATION_TIME	INDEX	INITIAL_VALUE	365		
1	DOCKING_REQUEST	QUANTITY (15)	STATUS	NEW_TANKER_ARRIVAL		TANKER_LEAVING

PROCESS INITIALIZE_TANKERS

START_THE_TANKERS

```
DISPLAY 'ENTER NUMBER OF TANEKRS'
ACCEPT NUMBER_OF_TANKERS
DISPLAY 'ENTER DOCK CLOSING TIME ( .75 = 6:00PM, ',
                                     '1.0 = 12:00 MIDNIGHT) '
ACCEPT CLOSE_TIME
DISPLAY 'ENTER SIMULATION TIME IN DAYS'
ACCEPT SIMULATION_TIME
EXECUTE START_TANKER
    INCREMENTING TANKER
    UNTIL TANKER IS GREATER THAN NUMBER_OF_TANKERS.
```

START_TANKER

```
SCHEDULED_TIME = (TANKER - 1)*0.5
SCHEDULE LOAD_TANKER IN SCHEDULED_TIME USING TANKER
```

PROCESS: LOAD_TANKER

LOAD_TANKER

```
TIME_TO_LOAD = UNIFORM (2.9, 3.1)
CROSSING_TIME_FULL = TNORMAL (5.0, 2.25)
TIME_TO_ARRIVE = TIME_TO_LOAD + CROSSING_TIME_FULL
SET DOCKING_REQUEST (TANKER) TO NEW_TANKER_ARRIVING
SCHEDULE DOCK_MANAGER IN TIME _TO _ARRIVE USING TANKER
```

```

RESOURCE:   LOADING_DOCK

STORAGE_TANK_INFORMATION
  1   AMOUNT_IN_STORAGE      REAL INITIAL_VALUE 1000000
  1   FLOW_TO_REFINERY       STATUS CUT_OFF
                               FREE
                               INITIAL_VALUE FREE
  1   STORAGE_SAFETY_VALVE   STATUS ALLOW_FLOW
                               SHUT_OFF
                               INITIAL_VALUE ALLOW_FLOW
  1   FLOW_FROM_TANKER       STATUS CUT_OFF
                               FREE
                               INITIAL_VALUE CUT_OFF

UNLOADING_DOCK_TIMES
  1   OPEN_TIME              REAL INITIAL_VALUE .25 *** 6 AM
  1   CLOSE_TIME             REAL INITIAL_VALUE .50 *** 12 PM
  1   TIME_OF_DAY            REAL
  1   NUMBER_OF_DAYS         INTEGER

TANKER_STATUS
  1   DOCK_SPACE             STATUS OCCUPIED
                               UNOCCUPIED
  1   AMOUNT_IN_TANKER       REAL INITIAL_VALUE UNOCCUPIED
  1   TANKER_FLOW            REAL INITIAL_VALUE 0
  1   TANKER_COUNT           REAL INITIAL_VALUE 3000
  1   TANKER_IN_DOCK         INDEX INITIAL_VALUE 0
  1   POINTER                INDEX
  1   TANKER_POSITION QUANTITY (15) INDEX

```

```

PROCESS   INITIALIZE_LOADING_DOCK

START_THE_TANKERS
  DISPLAY 'ENTER DOCK CLOSING TIME ( .75 = 6:00PM, ',
                               '1.0 = 12:00 MIDNIGHT) '
  ACCEPT CLOSE_TIME
  SCHEDULE STORAGE_CONTROLLER

```

```

PROCESS:      DOCK_MANAGER

DOCK_MANAGER
  IF DOCKING_REQUEST(TANKER) IS NEW_TANKER_ARRIVING
    EXECUTE SCHEDULE_NEW_TANKER
  ELSE IF DOCKING_REQUEST(TANKER) IS TANKER_LEAVING
    EXECUTE SEND_OFF_TANKER
    EXECUTE DOCK_NEXT_TANKER.

*****

SCHEDULE_NEW_TANKER
  IF TANKER_COUNT IS GREATER THAN ZERO
  OR DOCK_SPACE IS OCCUPIED
    EXECUTE QUEUE_UP_TANKER
  ELSE MOVE TANKER TO TANKER_IN _DOCK
    SET DOCK_SPACE TO OCCUPIED
    MOVE CAPACITY TO AMOUNT_IN_TANKER
    SET FLOW_FROM TANKER TO FREE.

QUEUE_UP_TANKER
  INCREMENT TANKER_COUNT
  MOVE TANKER TO TANKER_POSITION (TANKER_COUNT)

*****

SEND_OFF-TANKER
  SET DOCK_SPACE TO UNOCCUPIED
  CROSSING_TIME_EMPTY = TNORMAL(4.0, 1.0)
  SCHEDULE LOAD_TANKER IN CROSSING_TIME_EMPTY USING TANKER

*****

DOCK_NEXT_TANKER
  IF TANKER_COUNT IS GREATER THAN ZERO
    MOVE TANKER_POSITION(1) TO TANKER_IN_DOCK
    SET DOCK_SPACE TO OCCUPIED
    MOVE CAPACITY TO AMOUNT_IN_TANKER
    SET FLOW_FROM_TANKER TO FREE
    DECREMENT TANKER_COUNT
  ELSE SET DOCK_SPACE TO UNOCCPIED.
  IF TANKER_COUNT IS GREATER THAN ZERO
    EXECUTE MOVE_FORWARD
    INCREMENTING POINTER
    UNTIL POINTER IS GREATER THAN TANKER_COUNT.

MOVE_FORWARD
  TANKER_POSITION(POINTER) = TANKER_POSITION (POINTER + 1)

```

```

PROCESS:      STORAGE_CONTROLLER

CONTROL_STORAGE_TANK
  EXECUTE REFINERY_FLOW
  EXECUTE DETERMNE_TIME_OF_DAY
  IF DOCK_SPACE IS OCCUPIED
    AND TIME_OF_DAY IS GREATER THAN OPEN_TIME
    AND TIME_OF_DAY IS NOT GREATER THAN CLOSE_TIME
      EXECUTE TANKER_FLOW.
  SCHEDULE STORAGE_CONTROLLER IN .01 DAYS

DETERMINE_TIME_OF_DAY
  NUMBER_OF_DAYS = CLOCK_TIME
  TIME_OF_DAY = CLOCK_TIME - NUMBER_OF_DAYS

*****

REFINERY_FLOW
  IF FLOW_TO_REFINERY IS NOT CUT_OFF
    IF AMOUNT_IN_STORAGE IS GREATER THAN 5000
      SUBTRACT 1500 FROM AMOUNT_IN_STORAGE
    ELSE SET FLOW_TO_REFINERY TO CUT_OFF
  ELSE IF AMOUNT_IN_STORAGE IS GREATER THAN 50000
    SET FLOW_TO_REFINERY TO FREE
    SUBTRACT 1500 FROM AMOUNT_IN_STORAGE.

*****

TANKER_FLOW *** TANKER AT DOCK AND DOCK IS OPEN
  IF FLOW_FROM_TANKER IS NOT CUT_OFF
    IF STORAGE_SAFETY_VALVE IS NOT SHUT_OFF
      EXECUTE TRANSER_OIL_FROM_TANKER
    ELSE IF AMOUNT_IN_STORAGE IS LESS THAN 1600000
      SET STORAGE_SAFETY_VALVE TO ALLOW_FLOW
      EXECUTE TRANSER_OIL_FROM_TANKER.

TRANSER_OIL_FROM_TANKER
  ADD TANKER_FLOW TO AMOUNT_IN_STORAGE
  SUBTRACT TANKER_FLOW FROM AMOUNT_IN_TANKER
  EXECUTE CHECK_TANKER_LEVEL
  EXECUTE CHECK_STORAGE_LIMIT

CHECK_TANKER_LEVEL
  IF AMOUNT_IN_TANKER IS LESS THAN 7500
    SET FLOW_FROM_TANKER TO CUT_OFF
    SET DOCKING_REQUEST(TANKER_IN_DOCK) TO TANKER_LEAVING
    SCHEDULE DOCK_MANAGER NOW USING TANKER_IN_DOCK.

CHECK_STORAGE_LIMIT
  IF AMOUNT_IN_STORAGE IS NOT LESS THAN 2000000
    SET STORAGE_SAFETY_VALVE TO SHUT_OFF

```

```
PROCESS:      TANKER_INSTRUMENT

INSTRUMENT
  NUMBER_OF_DAYS = CLOCK_TIME
  TIME_OF_DAY = CLOCK_TIME - NUMBER_OF_DAYS
  IF NUMBER_OF_DAYS ARE LESS THAN SIMULATION_TIME
    SCHEDULE TANKER_INSTRUMENT IN 1 DAY WITH PRIORITY 1
  ELSE STOP.
```

SUMMARY AND OBSERVATIONS

This simulation was put together in about two days. It runs in much less than one second on a Pentium laptop under Windows 2000. Hundreds of simulations could be run in less than a minute for optimization purposes.

It is easy to see how one could expand the GSS model to account for many additional factors. It would also be easy to instrument this simulation using the GSS Run-Time Graphics (RTG) system. This would also allow users to interact with the simulation in real-time, making changes or disabling tankers at times considered critical by the analyst watching the simulation graphically.

In this simulation, one could use two different time scales, i.e., hours and days, since this can easily be implemented on a process by process basis in GSS. This would make the simulation more understandable to the reader. We also suggest starting the simulation on day 1 rather than at time zero, again for understandability. Here again, this is easy in GSS. Note also that we used the GSS Truncated Normal Distribution, TNORMAL(MEAN, VARIANCE), to determine the crossing times. This insures that negative times are not chosen, as might be the case for the normal distribution under certain circumstances.

Appendix A - State Space Definition Of A GSS Model

ELECTRICAL NETWORK ANALOGIES

Electrical engineers have evolved a graphical representation for complex networks of electrical elements using interconnected icons of basic element types: resistors, capacitors, inductors, generators, etc. As illustrated in Figure A-1, these elements can be used to build up hierarchical models of higher order elements, e.g., transistors, transmission lines, etc. Such a drawing defines the differential equations that describe the dynamic changes in electrical voltages and currents in the circuit. A mathematical representation is shown in Figure A-2, where X represents a vector of state variables and U represents a vector of external inputs. Given the initial conditions, the state of the circuit is defined for all time thereafter. In other words, the dynamic behavior of the network is defined by the hierarchical symbolic network.

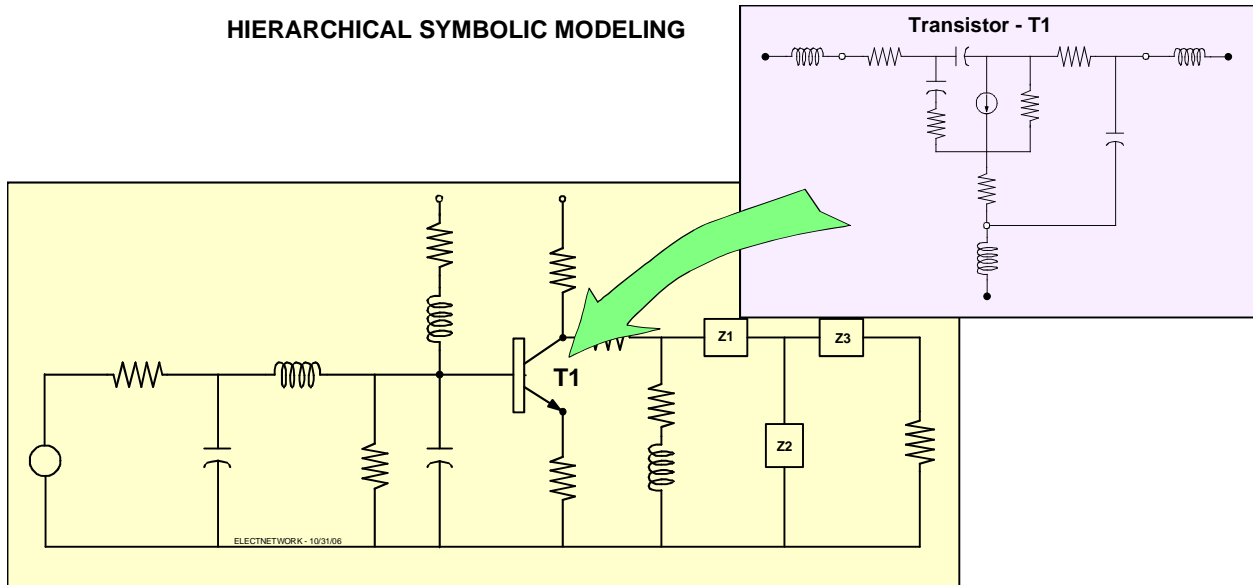


Figure A-1. Iconic representation of an electrical network.

$$\begin{bmatrix} X(T+\tau) \end{bmatrix} = \begin{bmatrix} A(T) \end{bmatrix} \cdot \begin{bmatrix} X(T) \end{bmatrix} + \begin{bmatrix} B(T) \end{bmatrix} \cdot \begin{bmatrix} U(T) \end{bmatrix}$$

Figure A-2. State Space Representation of an electrical network.

Requirement Specification And Design Specification

Designers of electronic circuits start with functional requirement specification documents and produce drawings and design specification documents, see Figure A-3, describing the network to be fabricated in production. Negotiation of changes in functional requirements and testing of proposed fabrication processes for reliability are a significant part of the engineering design process. Engineering judgment supported by mathematical calculations provide a critical part of this process. Testing is used to validate models of the nominal behavior of a network as well as variations in component values due to production and environmental changes. Ensuring reliability requirements are met in anticipated “worst case” environments is a hard constraint.

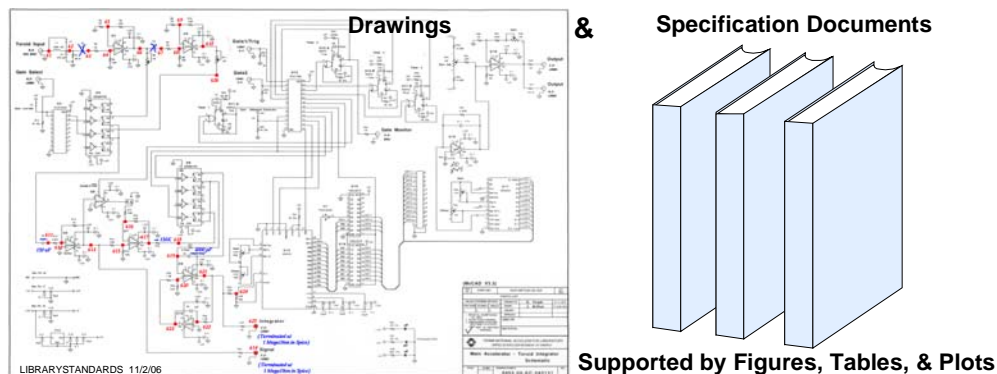


Figure A-3. Requirements and design specification documents.

Use Of Electrical Analogies For Enhanced Productivity

Because of the development of a complete and consistent theory for electrical network design, electrical analogies are used in many other fields, such as those involving mechanical design, fluid flow, etc. Additionally, characterization of nonlinear components and development of fast and accurate solutions of the differential equations representing these networks provides a highly productive facility, minimizing the burden of analysis of different designs. These facilities have been incorporated into numerous Computer-Aided Design (CAD) tools that have reduced the time and resources required to complete a complex design by orders of magnitude. Obtaining fast and accurate solutions to posed design problems provides huge improvements in productivity in the fabrication stage as well as in production of the design specifications.

APPLICATION TO DISCRETE EVENT SIMULATION AND SOFTWARE

There are four reasons why *discrete event* simulation is a good software analogy starting point. First, an electrical network analogy for this type of simulation is easy to derive. Second, it has been shown, see [1], that discrete event simulation can be used to solve nonlinear problems in the time domain that are otherwise intractable using differential or discrete time equations. Third, this same reference provides examples of more easily understood solutions even for linear problems using a discrete event CAD system versus using differential or discrete time equations. This is because of the higher degrees of abstraction imposed when using standard mathematical frameworks. After reviewing sufficient examples, it becomes clear that such abstractions make it much more difficult to model realistic behavior in a way that is easily understood.

Finally, software is a subset of discrete event simulation when using the General Simulation System (GSS). This system, developed in 1982, is based upon the Generalized State Space framework, [1]. We will start with the electrical analogy of discrete event simulation using GSS and State Space as it is used to represent electrical network equations.

Generalized State Space Representation Of GSS

Within a GSS simulation, a model only has access to a subset of the simulation state vector when a process in that model is running. We will call this the model state vector. A subset of the *model state vector* contains those resources that are contained within the model. Because this GSS state vector may contain character as well as numeric data, it is called a *generalized state vector*. The Generalized State Space representation of a GSS model is shown in Figure A-4. The state vector that a model has access to consists of the following items and their corresponding information elements:

- ACCESSIBLE RESOURCES - The information contained in resources to which processes within the model are attached. Note: Shared resources may or may not reside within the model.
- SIMULATION QUEUE - The entries in the queue, including the indices it uses when it's processes are scheduled.
- SIMULATION CLOCK - The time of the simulation clock, including priority, if and when it schedules another process.
- REAL TIME CLOCK - The value of the real-time clock if it is used.
- RANDOM NUMBER GENERATOR - The current value of the random number generator seed if it is used.

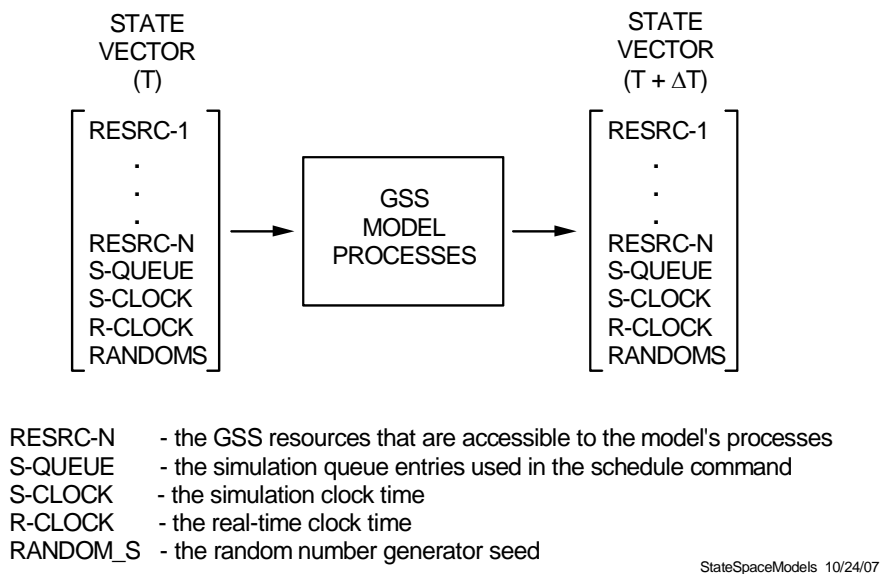


Figure A-4. Generalized State Space representation of GSS.

A GSS process is scheduled based upon its internal logic or other processes. When a GSS process *runs*, it may schedule itself or other processes at specified times in the future, or at the current time. GSS processes run in zero *simulated* time. At any time, the state of a model depends solely upon its state vector. When a process in a model runs, its *terminal state*, i.e., the value of its substate vector - when it passes control back to GSS - depends solely upon its *initial state*, i.e., the initial value of its substate vector, and the rules within the process.

When processes in another model share a part of the state vector of a given model, then any future state of the given model is, in general, dependent upon the rules in the other model, since they can change the given model's state vector.

ANALOGY TO SYMBOLIC MODELS USING STATE SPACE

The Generalized State Space representation of a GSS model, Figure A-3, is analogous to a set of differential equations that represent the state of a dynamic system at any instant in time. All future states are represented by the *equations of motion* in State Space notation, and the initial conditions, reference Schweppe, [SC].

In GSS, the interconnection of resources and processes, see Figures 2 and 4 (main text), is analogous to the electrical circuit drawing in Figure A-1. Each has its corresponding *rules* and *storage* underlying each primitive element. In the case of electrical circuits, there are constituent equations that describe the changes in energy storage in differential form for each primitive icon. Representation of any system element must conform to this form of change.

In the case of GSS, sets of rules operate on sets of attributes (contained in data structures) to define the elementary change relationships in a model. Using GSS, the engineering drawing shown in Figure 4, and the underlying rule and data structures, define the total state of the simulation at any point in time after the initial conditions. This is based upon the *Generalized State Space framework*, see [1].

Choosing the Most Convenient Reference Frame

As implemented in GSS and described above, the Generalized State Space framework supports the representation of discrete event systems as well as discrete and continuous time systems. Figure A-5 illustrates the generalized state space as providing an underlying framework for representing dynamic systems.

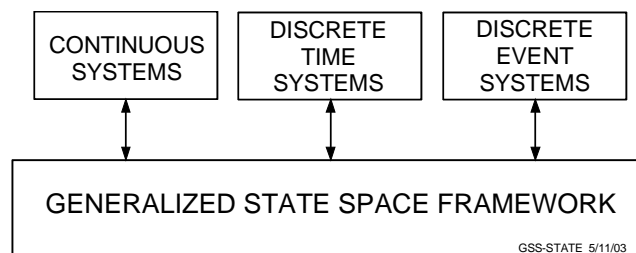


Figure A-5. Generalized State Space:
- an underlying framework for representing dynamic systems.

The difference between representations of a system's dynamics may translate into huge differences in speed and productivity. A particular representation can be selected to make it easier to analyze or predict specific system behavior. If a system is conveniently represented by a set of differential or difference equations, then one of those representations may be best. If the system is more easily described by sets of rules operating on sets of attributes that may contain nonnumeric elements, then that representation should be chosen.

Since the advent of the digital computer, people have moved from analytical methods for integrating differential equations to heuristic algorithmic methods, especially when the systems represented are very nonlinear or nonstationary. Fast numerical algorithms for solving stiff nonlinear systems typically use complex heuristic approaches. All of these approaches can be implemented easily using GSS rule and attribute structures. As computers provide significantly greater memory and speed advantages, the space for solving problems is growing, alleviating restrictions to abstract numerical methods for solution, and allowing rapid movement toward heuristic rule-oriented approaches using complex data structures.

Having selected GSS as the overall framework, the analogy then becomes one of selecting the best set of Generalized State Vectors (GSS Resources) to represent the system attributes. Depending upon how the resources are selected and structured, the Generalized Transformations (GSS Processes) may be much more simple to understand, build, and modify. This is determined by the *independence properties of the architecture*, i.e. the interconnection of resources and processes - *not the code!* See [1].

Mapping Into Software

Since there are only two language statements used by GSS that are specific to simulation, the rest of the language is as general as any software language. It is a very rich language that is used for the GSS counterpart, the Visual Software Environment (VSE). All of the architectural properties available to GSS users also reside in VSE. We note that GSS is written in VSE. Thus, the Generalized State Space framework also provides a theoretical basis for designing and building software.

REFERENCES

- [1] Cave, W.C., *A Generalized State Space Framework For Software*, Visual Software International Technical Report, Spring Lake, NJ, October 2008.
- [2] Gelb, A., Editor, *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.
- [3] Gordon, G., *A General Purpose Systems Simulation Program*, Proc. EJCC, Washington, D.C., pp 87-104., MacMillan Publishing Co., New York, 1961.
- [4] Gordon, G., **The Application of GPSS V to Discrete System Simulation**, Prentice Hall, Englewood Cliffs, NJ, 1975.
- [5] Gordon, J., **System Simulation**, Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [6] *GSS User's Reference Manual*, Version 10.4, Visual Software International, Spring Lake, NJ, 2005.
- [7] Hactel, G.D. and Roher, R.A., *Techniques For The Optimal Design And Synthesis Of Switching Circuits*, Proceedings of the IEEE Special Issue on CAD, Nov 1967, pp 1864.
- [8] Hactel, G.D. et al, *The Sparse Tableau Approach To Network Analysis And Design*, IEEE Transactions on Circuit Theory, Jan 1971, pp 101.
- [9] Yasushi Kambayashi and Henry F. Ledgard, "The Separation Principle - A Programming Paradigm" IEEE Software, March/April 2004
- [10] Ramadge, P.J. and W.M. Wonham, "Supervisory Control of a class of discrete-event processes," SIAM J. Control Optimization, vol 25, no.1, pp 206-230, Jan. 1987.
- [11] Ramadge, P.J. and W.M. Wonham, "The Control Of Discrete-Event Systems," Proc. IEEE, vol 77, no.1, pp 206-230, Jan. 1989.
- [12] Schweppe, F., *Uncertain Dynamic Systems*, Prentice Hall, Englewood Cliffs, NJ, 1978.
- [13] Zadeh, L.A. and Desoer, C.A., *Linear System Theory: The State Space Approach*, McGraw-Hill, NY 1963.
- [14] Hafner, Eric, *Theory and Design of Oscillators*, Proceedings of the IEEE, New York, NY, in two issues, 1976-78.