9

Computer Methods in Systems Studies

9.1 Introduction

In the preceding eight chapters, we have given an introduction to the study of discrete systems. It has been continually demonstrated that the electronic computer is integrally involved in this study. Not only is the computer required as an essential tool in the analysis, simulation, and design of discrete systems, but we also see the computer assume a significant role in the control and general operation of discrete systems and mixed continuous-discrete systems. Throughout the material presented in the eight chapters, computer techniques are used. Mention has been made of the existence of computer routines to carry out a number of computational tasks. Several computer techniques have been recommended for the analysis of sampled-data systems and for the optimum design of discrete systems.

Because of the rather important role played by the modern computer in the analysis, design, development, and control of a great variety of systems, we consider it highly appropriate to introduce the reader to some of the detailed aspects of computer methods. We will concern ourselves primarily with the programming and application of computers, not with their construction. Furthermore, attention will be given to a discussion of a selected number of numerical techniques that are useful in system simulations.

Historically, it has been customary to divide computers into analog and digital. In recent years, a third important kind of computer has been added to the computer family—the *hybrid computer*. This particular variety of computer has found extensive use in systems studies of all kinds (particularly simulation, optimum design, high-speed iterative solutions, etc.). We will assume a background in basic analog and digital programming (FORTRAN) for the purpose of studying the material presented in this chapter. This will be sufficient to follow the development on hybrid programming.

To a large extent computer programs are prepared by expert programmers to handle a great many different tasks. However, it is often very helpful for the systems designer, the user of these programs, to have an elementary understanding of some of the techniques upon which these programs are based. It is for this purpose that this chapter has been prepared.

9.2 Numerical Methods of Simulating System Dynamics

In almost every digital simulation of dynamic systems the programming of the solution of differential and/or difference equations is required. These equations characterize the dynamics of the system under study. The digital machine performs its basic operations involving arithmetic, memory, and logic operations, in terms of variables which are always represented in discrete form. All continuous mathematical operations must be converted into a corresponding discrete form before they can be processed by the computer. Subject to this requirement are the frequently occurring operations such as integration and differentiation, for example. Also affected, although somewhat differently, are the computations of basically continuous response curves such as root-locus and frequency response.

To handle these and a great many other tasks, suitable digital programs, called *subroutines*, are available for use by the systems designer. In this section we will examine a number of routines which are frequently used in the solution of differential equations. We will direct our attention to a presentation of the discrete approximation technique and errors that are incurred by it. Briefly, two basic types of errors are introduced: the truncation error, which is due to finite approximations of infinite expansions, and the round-off error, which is due to the finite bit capacity of the digital computer. Furthermore, consideration has to be given to the possibility of an unstable discrete representation.

In the solution of differential equations by a digital computer it is assumed

that the system to be simulated is characterized in first-order differential equation form, or state variable equations, i.e.,

$$\frac{d}{dt}x_i = f_i(x_1, x_2, \ldots, x_n, t) \qquad i = 1, 2, \ldots, n \tag{9.2-1}$$

or, in the case of a linear system,

$$\dot{\mathbf{x}} = \mathbf{Fx} + \mathbf{Gr} \tag{9.2-2}$$

Techniques dealing with the derivation of these equations have been presented in numerous texts on modern system theory.

Numerical integration of these equations is based upon a recursive evaluation of their discrete equivalent, which is essentially of the form

$$x_i(k+1) = g_i(x_1(k), x_2(k), \ldots, x_n(k), kT) \quad i = 1, 2, \ldots, n \tag{9.2-3}$$

where $x_i(k)$ is the ith state variable of the simulated system at time $t = kT$. We shall now present four methods of generating a digital simulation model and discuss their relative merits.

9.2-1 Euler Method

Probably the simplest approximation to (9.2-1) is obtained from a first-order Taylor series expansion of the state variables $x_i(t)$. Such an expansion yields

$$x_i(t+T) = x_i(t) + T\dot{x}_i(t) \tag{9.2-4}$$

If we let

$$x_i(t+T) = x_i(k+1) \quad \text{and} \quad x_i(t) = x_i(k)$$

where k denotes the kth evaluation, corresponding to $t = kT$, and also substitute (9.2-1) into (9.2-4), we obtain

$$x_i(k+1) = x_i(k) + Tf_i(x_1, x_2, \ldots, x_n, k) \tag{9.2-5}$$

This approximation is generally referred to as the Euler approximation. It is simple and can be programmed in a straightforward fashion. As an illustration, consider the following differential equation.

EXAMPLE 9-2.1

Determine the Euler approximation of the differential equation

$$\ddot{x} + a\dot{x} + bx = 1 \quad t \geq 0 \tag{9.2-6}$$

First, a set of state equations is given by

$$\begin{aligned} \dot{x}_1 &= -ax_1 - bx_2 + 1 \\ \dot{x}_2 &= x_1 \end{aligned} \tag{9.2-7}$$

Consequently, the Euler approximation is

$$\begin{aligned} x_1(k+1) &= x_1(k) + T(-ax_1(k) - bx_2(k) + 1) \\ x_2(k+1) &= x_2(k) + Tx_1(k) \end{aligned} \tag{9.2-8}$$

where T is the incremental time used in the numerical integration of (9.2-6).

The Euler approximation works both for linear and nonlinear system models. Despite these attractive features, however, it suffers serious shortcomings in accuracy and stability and is therefore rarely used. Discussion of these drawbacks will serve well to point out some of the important considerations that go into the selection of a numerical technique in general.

Truncation Error

The series expansion (9.2-4) is truncated after the first derivative term. The error in the solution which is due to this effect is called the *truncation error*.

The significance of the truncation error can be further emphasized by the fact that it tends to accumulate with time. However, differential equations which characterize stable dynamic systems contain built-in feedback which is preserved in modified form in their discrete approximation. This feedback acts in support of the accuracy of the solution. In addition, the numerical approximation solution will seek the same steady-state as the original continuous model, provided, of course, that the stability of the equivalent discrete model is not adversely affected. Let us illustrate this problem by reconsidering Example 9.2-1.

EXAMPLE 9.2-2

Investigate the steady-state and stability characteristics of the Euler approximation of Example 9.2-1.

It is quite clear that the steady-state of (9.2-6)—i.e., $\ddot{x} = \dot{x} = 0$, is $1/b$ with no problems of stability, provided both a and b are positive. From (9.2-8) it can be seen as $k \longrightarrow \infty$ that $x_1(k) \longrightarrow 0$ and $x_2(k) \longrightarrow 1/b$. Thus, the discrete approximation has the same steady-state value.

We next examine the stability of the discrete approximation. For this purpose we determine eigenvalues of the system matrix of (9.2-8). In matrix format we have

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 - aT & -bT \\ T & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} T \\ 0 \end{bmatrix} \tag{9.2-9}$$

The eigenvalues are

$$\lambda_{1,2} = 1 - \frac{aT}{2} \pm \sqrt{T\left(\frac{a^2}{4} - b\right)}$$

If, for instance, $a = 2$ and $b = 10$, then the eigenvalues are given by $\lambda_1 = 1 - 4T$ and $\lambda_2 = 1 + 2T$. It is seen that the Euler approximation is unstable for any choice of T. If, however, $a = .2$ and $b = .1$, then $\lambda_{1,2} = 1 - .1T \pm .3jT$. For $T = .1$, $\lambda_{1,2} = .99 \pm j.03$. For this choice of a, b and T we have $|\lambda_{1,2}| < 1$; hence, the Euler approximation is stable. However, if T is increased to 1, one may readily verify that the discrete simulation would be unstable.

The effect of the truncation error may be partially countered by making the time interval T sufficiently small. This is accompanied, however, by an increase in the number of steps in the solution requiring more computer time and eventually making another type of error, round-off error, prominent. Similarly, the stability of the solution may be controlled by the selection of the step size.

Round-off Error

Round-off error is the second basic error encountered in digital computation. It results as a consequence of the finite number of digits with which a digital computer can carry out arithmetic operations. This number may range from four digits for digital machines with 12-bit words to 15 digits for digital machines with 60-bit words. Round-off affects the last digit of a given digital word. For instance, in a word with eight digits the first seven digits are exact, while the last digit is rounded off. By this process it is possible to represent a real number by an eight-digit word that conceivably could have an infinite number of digits for exact definition. The first seven digits match, while the eighth digit is adjusted up if the ninth digit is larger than .5; otherwise it stays the same.

Round-off error affects every digital computation. In numerical integration it assumes particular importance when the integration step has to be selected as very small. Under this condition the significant changes in the values of dynamic systems variables occur more and more in the last few digits of the computer words. An illustration of the influence of round-off error is given by the next example.

EXAMPLE 9.2-3

Consider the numerical solution of the differential equation

$$\ddot{x} + \omega^2 x = 0, \qquad x(0) = A, \quad \dot{x}(0) = B \tag{9.2-10}$$

This is the equation of a simple harmonic oscillator with resonant frequency ω. When $x(t)$ and $\dot{x}(t)$ are computed and plotted against one another in $(x, \dot{x})$ space, the resulting curve, for instance, is a circle with radius $R = \sqrt{A^2 + B^2}$ when $\omega = 1$. Our objective here is to use an Euler approximation to obtain $x(t)$ and $\dot{x}(t)$ and to determine how close the result agrees with a circle.

Results of this experiment are shown at the end of this section and compared with other methods.

9.2-2 Tustin Method

A technique that offers some appeal in the simulation of linear systems is the Tustin method. It is basically an approximation of differentiation by a difference equation. Consider the following relationships.

The definition of the discrete transform variable z is given by

$$z = e^{sT} \tag{9.2-11}$$

Solving for s yields

$$s = \frac{1}{T} \ln z \tag{9.2-12}$$

The logarithmic term may be approximated by the series

$$\ln z = 2(u + \tfrac{1}{3}u^3 + \tfrac{1}{5}u^5 + \ldots) \tag{9.2-13}$$

where

$$u = \frac{1 - z^{-1}}{1 + z^{-1}}$$

Truncating the series after the first term and substituting into (9.2-12), we obtain the so-called Tustin approximation for the derivative operator s.

$$s \approx \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{9.2-14}$$

This relationship is used in converting the continuous-time models of linear systems into difference equations. This difference equation may be readily solved recursively on a digital computer, which yields a fairly accurate digital simulation of the linear system.

The substitution (9.2-14) may be applied to an nth-order linear differential equation, to n first-order linear differential equations, or to an nth-order transfer function. The result can usually be arranged into one of two forms, (1) n first-order difference equations or (2) one nth-order difference equation. This may best be illustrated by two examples.

EXAMPLE 9.2-4

Derive the difference equation that represents a Tustin simulation of the transfer function

$$G(s) = \frac{as + b}{s(s + c)}$$

Upon substitution of the approximation given by equation (9.2-14) we obtain

$$G(z) = \frac{a\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} + b}{\left(\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}\right)^2 + c\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}}$$

which is simplified to

$$G(z) = \frac{2aT + bT^2 - 2bT^2z^{-1} + (bT^2 - 2aT)z^{-2}}{4 + 2cT - 8z^{-1} + (4 - 2cT)z^{-2}}$$

This second-order digital transfer function may be readily programmed for computer execution.

EXAMPLE 9.2-5

Derive a difference equation for the harmonic oscillator of Example 9.2-3, using the Tustin approximation.

The differential equation of the harmonic oscillator is given by (9.2-10), or in state variable representation

$$\dot{x}_1 = -\omega^2 x_2$$
$$\dot{x}_2 = x_1$$

Using (9.2-14), we have

$$\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}X_1(z) = -\omega^2 X_2(z)$$

and

$$\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}X_2(z) = X_1(z)$$

Converting into difference equations, we obtain

$$x_1(k) = x_1(k - 1) - \frac{\omega^2 T}{2}[x_2(k) + x_2(k - 1)] \qquad (9.2\text{-}15)$$

$$x_2(k) = x_2(k - 1) + \frac{T}{2}[x_1(k) + x_1(k - 1)] \qquad (9.2\text{-}16)$$

In order to evaluate these difference equations recursively, we must eliminate $x_2(k)$ from the first equation. This yields

$$x_1(k) = \left(\frac{4 - \omega^2T^2}{4 + \omega^2T^2}\right)x_1(k - 1) + 2\left(\frac{2 - \omega^2T}{4 + \omega^2T^2}\right)x_2(k - 1) \qquad (9.2\text{-}17)$$

The digital simulation of the harmonic oscillator now involves (9.2-17) and (9.2-16), to be recursively evaluated, in that order.

This simulation is carried out for various values of T and is compared with the results of the Euler and Runge-Kutta methods. The results of this comparison are summarized at the end of this section.

The Tustin method obviously requires a considerable amount of manipulation to make the substitution (9.2-14) and to rearrange the result into a suitable form. This job can be easily handled by a digital computer subroutine. For instance, one program may start with the description of the system by a transfer function and generate a digital transfer function of the form

$$D(z) = \frac{b_0 + b_1z^{-1} + \ldots + b_nz^{-n}}{1 + a_1z^{-1} + \ldots + a_nz^{-n}}$$

This may then be easily programmed as a digital recursion equation.

9.2-3 Runge-Kutta Method

The Runge-Kutta method and its various modifications are the most widely employed of the single-step methods. These are methods of a class for which $x_i(k + 1)$ can be obtained from $x_i(k)$ alone and the differential equations. The methods are self-starting and are not difficult to program. The basic Runge-Kutta method is partially an extension of the Euler method, which, as was mentioned, is based upon the first-order Taylor series (9.2-4). This series expansion suggests inclusion of higher-order terms to reduce the truncation error. Although the first derivative is available from the differential equation, higher-order derivatives must be evaluated separately, possibly by difference methods. This would result in an effective method, except that the evaluation of the higher-order derivatives can be very tedious. The Runge-Kutta method makes use of the higher-order Taylor approximations indirectly so as to avoid this problem.

The Runge-Kutta methods used in practice are based on fourth-order Taylor approximations. Although these are simple in their use, their deriva-

tions entail complicated developments.* It will suffice here to present only the computational algorithm.

Let the system of equations to be solved be given in the familiar form

$$\begin{aligned} \dot{x}_i &= f_i(x_1, x_2, \ldots, x_n, t) = f_i(\mathbf{x}, t) \\ x_i(t_0) &= x_{i0} \quad i = 1, 2, \ldots, n \end{aligned} \tag{9.2-18}$$

Let $x_i(k)$ be the value of x_i at $t = t_k$ and $f_i(\mathbf{x}(k), t_k)$ the derivative of x_i at $t = t_k$. If T is the increment (step-size) of the time variable t, the Runge-Kutta fourth-order method uses the formulas

$$\begin{aligned} K_{1i} &= Tf_i(\mathbf{x}(k), t_k) \\ K_{2i} &= Tf_i[\mathbf{x}(k) + .5\mathbf{K}_1, t_k + .5T] \\ K_{3i} &= Tf_i[\mathbf{x}(k) + .5\mathbf{K}_2, t_k + .5T] \\ K_{4i} &= Tf_i[\mathbf{x}(k) + \mathbf{K}_3, t_k + T] \\ x_i(k+1) &= x_i(k) + \tfrac{1}{6}[K_{1i} + 2K_{2i} + 2K_{3i} + K_{4i}] \quad i = 1, 2, \ldots, n \end{aligned} \tag{9.2-19}$$

where $\mathbf{K}_1$, $\mathbf{K}_2$, $\mathbf{K}_3$, and $\mathbf{K}_4$ are $n \times 1$ vectors determined by (9.2-19).

For each step of integration in (9.2-19) the Runge-Kutta method requires four evaluations of the functions $f_i(x_1, x_2, \ldots, x_n, t)$, thus requiring considerable amount of machine time. The Runge-Kutta method is characterized by a high degree of accuracy, which compares well with analytical methods of solution.

EXAMPLE 9.2-6

Consider a Runge-Kutta solution to the problem of the harmonic oscillator presented in Example 9.2-3. The results are shown for various step sizes at the end of this section.

Selection of Step Size and Control of Truncation Error

Essentially, the only factor left to the user's discretion is the time increment T. For obvious reasons this should be selected as large as possible to keep the machine running time as small as possible. Because the truncation error is kept small, in most cases the only risk incurred by making the step size too large is the possibility of numerical instability. For most applications numerical instability is a more predominant problem than inaccuracy caused by truncation error. The analytical complexity of the method prohibits a direct stability analysis, even for the simplest case. A good rule of thumb is to keep the step size approximately at a tenth of the value of the smallest time constant of the differential equations to be solved.

*See, for example, Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, 1958.

The typical dynamic transient is associated with large values of derivatives during the early stages, which rapidly decrease during the final stages. Considerable computer time may be saved if the step is adjusted to be small when the derivatives are large and vice versa.

The step size may be adjusted automatically during a calculation provided there exists an explicit expression for the upper bound of the truncation error.

With an explicit knowledge of the instantaneous, or local, truncation error one can always select that step size which keeps the truncation error just below an acceptable upper bound. Although it is known that the local truncation error is roughly proportional to the fifth power of the step size for a fourth-order Runge-Kutta integration method, it is not possible to obtain a running estimate of the truncation error during the course of an integration.

A modification of the basic fourth-order Runge-Kutta method which has a slightly smaller truncation error and provides an estimate of the truncation error is the Runge-Kutta-Merson method. It requires five iterations per step size and step size adjustments may be made during the course of an integration. The Runge-Kutta-Merson* algorithm is

$$\begin{aligned} K_{1i} &= \tfrac{1}{3}Tf_i[\mathbf{x}(k), t_k] \\ K_{2i} &= \tfrac{1}{3}Tf_i\left[\mathbf{x}(k) + \mathbf{K}_1, t_k + \frac{T}{3}\right] \\ K_{3i} &= \tfrac{1}{3}Tf_i\left[\mathbf{x}(k) + .5\mathbf{K}_1 + .5\mathbf{K}_2, t_k + \frac{T}{3}\right] \\ K_{4i} &= \tfrac{1}{3}Tf_i(\mathbf{x}(k) + \tfrac{3}{8}\mathbf{K}_1 + \tfrac{9}{8}\mathbf{K}_3, t_k + .5T) \\ K_{5i} &= \tfrac{1}{3}Tf_i(\mathbf{x}(k) + \tfrac{3}{2}\mathbf{K}_1 - \tfrac{9}{2}\mathbf{K}_3 + 6\mathbf{K}_4, t_k + T) \end{aligned} \tag{9.2-20}$$

$$x_i(k+1) = x_i(k) + .5(K_{1i} + 4K_{4i} + K_{5i}) \quad i = 1, 2, \ldots, n \tag{9.2-21}$$

The estimate of the error is given by

$$\text{truncation error} = \max\left(K_{1i} - \tfrac{9}{2}K_{3i} + 4K_{4i} - \tfrac{1}{2}K_{5i}\right) \tag{9.2-22}$$

The operation of automatic step size adjustment in the Runge-Kutta-Merson algorithm would be to maximize the step size while the truncation is kept within a specified bound. A computer program called subroutine INTFUN utilizing this algorithm in contained in Appendix 9A.

9.2-4 Adams-Moulton Predictor-corrector Method

A third important numerical technique for the integration of differential equations is the Adams-Moulton method. It employs a predictor-corrector principle and uses the following recursive algorithm:

*Lance, G. N., *Numerical Methods for Highspeed Computers*, Iliffe, London, 1960.

$$x_i^p(k+1) = x_i(k) + \frac{T}{24}[55f_i(k) - 59f_i(k-1) + 37f_i(k-2) - 9f_i(k-3)] \tag{9.2-23}$$

$$x_i(k+1) = x_i(k) + \frac{T}{24}[9f_i^p(k+1) + 19f_i(k) - 5f_i(k-1) + f_i(k-2)] \quad i = 1, 2, \ldots, n \tag{9.2-24}$$

The method requires two evaluations of the differential equations for each step. One involves $f_i(k)$ in the predictor equation (9.2-23), while the other is $f_i^p(k+1)$ in the corrector equations (9.2-24). The principle of the Adams-Moulton method is based on the determination of a first estimate $x_i^p(k+1)$ by (9.2-23) using values of the derivatives at four successive time instants, with a subsequent correction by (9.2-24) using values of the derivatives at three successive time instants and $f_i^p(k+1)$, the derivative at the first estimate.

The Adams-Moulton method has a time-saving advantage over the Runge-Kutta method. However, it is not self-starting. For the first three time intervals, a one-step method like the Runge-Kutta method is used to obtain the needed starting values. Thus a combination of the Runge-Kutta and Adams-Moulton methods is employed.

Control of Step Size and Truncation Error

The Adams-Moulton is a fourth-order method, and hence the truncation error is of the order of T^5. An explicit expression for an estimate of the truncation error is available, thus making automatic step size adjustments possible. It is given by

$$\text{truncation error} = \max_i \frac{|x_i^p(k+1) - x_i(k+1)|}{14D_i} \tag{9.2-25}$$

where

$$D_i = \max_i \{x_i^p(k+1), \alpha\} \quad i = 1, 2, \ldots, n$$

and where α is a positive constant used to prevent unnecessary reductions in T. It is usually set equal to 1.

As is the case with the various Runge-Kutta methods, the Adams-Moulton method may be used in the integration of differential equations with arbitrary expressions for $f_i(\mathbf{x}, t)$. In fact, the execution of the program is totally indifferent to the nature of $f_i(x_1, x_2, \ldots, x_n, t)$.

In choosing between a Runge-Kutta method or the Adams-Moulton method, obviously time is of prime consideration. An important additional criterion affecting a choice is the presence of discontinuities in the right-hand side functions. If the derivatives contain no discontinuities, a predictor-corrector method using only two function evaluations per step is likely to be faster than a one-step method. If, however, they do have discontinuities, as is so frequently the case in engineering problems, a one-step method is more accurate and more efficient.

The remaining three techniques of simulation to be presented in this section are restricted to linear systems, or at least piecewise linear systems with constant coefficients.

9.2-5 State Transition Method

A numerical technique of simulating linear systems that is rapidly gaining widespread acceptance by systems designers is based upon state variable techniques. A linear system may be described by the equations

$$\frac{d}{dt}\mathbf{x} = \mathbf{F}\mathbf{x} + \mathbf{G}u \tag{9.2-26}$$

$$y = \mathbf{C}\mathbf{x} + du \tag{9.2-27}$$

When the input $u(t)$ can be adequately represented by a piecewise constant equivalent

$$u(t): \quad u(kT + \tau) = u(kT) \quad 0 \leq \tau < T \tag{9.2-28}$$

it is possible to determine $y(t)$ at the discrete times $t = 0, T, 2T, \ldots$ by means of the discrete state equations

$$\mathbf{x}[(k+1)T] = e^{\mathbf{F}T}\mathbf{x}(kT) + \int_0^T e^{\mathbf{F}\tau}d\tau\mathbf{G}u(kT) \tag{9.2-29}$$

$$y(kT) = \mathbf{C}\mathbf{x}(kT) + du(kT) \tag{9.2-30}$$

These relations were derived in Chapter 2 with the following notation:

$$\mathbf{A}(T) = e^{\mathbf{F}T} \quad \text{and} \quad \mathbf{B} = \int_0^T e^{\mathbf{F}\tau}d\tau\mathbf{G} \tag{9.2-31}$$

The matrices **A** and **B** have to be evaluated only once for any given time interval T.

The most remarkable feature of the state transition method is that it offers a discrete simulation of a continuous-time system which is almost completely exact. There are only two sources of error. One is introduced by sampling the input. The other one is caused by the iterative procedure for evaluating the matrices **A** and **B**. But no error is introduced by the actual discrete model. It is exact.

Evaluation of **A** *and* **B**

We shall present two procedures for approximating **A** and **B** by series techniques. Because of the relatively recent development of these techniques an extensive development on the truncation error will be given.

The first of the techniques is due to Liou.* The transition matrix **A** is expressed by the infinite series

$$\mathbf{A} = \sum_{i=0}^{\infty} \frac{\mathbf{F}^i T^i}{i!}, \qquad \mathbf{F}^0 = \mathbf{I} \tag{9.2-32}$$

This series is uniformly convergent in a finite interval. It is, therefore, possible to evaluate **A** within prescribed accuracy. If the series is truncated at $i = L$, then we may write

$$\mathbf{A} = \sum_{i=0}^{L} \frac{\mathbf{F}^i T^i}{i!} + \sum_{i=L+1}^{\infty} \frac{\mathbf{F}^i T^i}{i!} = \mathbf{M} + \mathbf{R} \tag{9.2-33}$$

The first term in (9.2-33) represents the series approximation, while the second term corresponds to the remainder term.

If each element in **A** is required to be within an accuracy of at least d significant figures, then

$$|r_{ij}| \leq 10^{-d} |m_{ij}| \tag{9.2-34}$$

where r_{ij} and m_{ij} correspond to the elements of **R** and **M**, respectively. Let the norm of **F** be

$$\|\mathbf{F}\| = \max_i \left(\sum_{j=1}^{n} |a_{ij}| \right)$$

Then, it can be shown that

$$\|\mathbf{F}^k\| \leq \|\mathbf{F}\|^k \qquad k = 1, 2, \cdots$$

Hence each element of the matrix $\mathbf{F}^k$ is less than or equal to $\|\mathbf{F}\|^k$. It follows that

$$|r_{ij}| \leq \sum_{i=L+1}^{\infty} \frac{\|\mathbf{F}\|^i T^i}{i!} \tag{9.2-35}$$

Let the ratio of the second term to the first term of the series (9.2-35) be ϵ, that is

$$\epsilon = \frac{\|\mathbf{F}\| T}{L+2} \tag{9.2-36}$$

from which we conclude that

$$\frac{\|\mathbf{F}\| T}{2} \leq \epsilon$$

Substituting the last relation into (9.2-35) we have

*M. L. Liou, "A novel method of evaluating transient responses," *Proceedings of IEEE*, Vol. 54, No. 1, January 1966, pp. 20–23.

$$|r_{ij}| \leq \frac{\|\mathbf{F}\|^{L+1} T^{L+1}}{(L+1)!}(1 + \epsilon + \epsilon^2 + \epsilon^3 + \cdots)$$

$$= \frac{\|\mathbf{F}\|^{L+1} T^{L+1}}{(L+1)!} \frac{1}{1-\epsilon} \tag{9.2-37}$$

Thus the matrix **A** can be evaluated approximately according to the following iterative procedure:

1. Choose an initial value of L.
2. Evaluate m_{ij} by (9.2-33).
3. Determine ϵ by (9.2-36).
4. Find the upper bound of $|r_{ij}|$ by (9.2-37).
5. Compare each element of M obtained from (2) with the upper bound of $|r_{ij}|$ obtained from (4); if (9.2-34) is not satisfied, increase L and repeat the iteration; otherwise, the iteration is complete.

The matrix **B** may be evaluated in a similar manner. From the properties of exponential matrices it can be shown that

$$\mathbf{B} = (e^{\mathbf{F}T} - \mathbf{I})\mathbf{F}^{-1}\mathbf{G} \tag{9.2-38}$$

Thus, using (9.2-30),

$$\mathbf{B} = T \sum_{j=0}^{\infty} \frac{\mathbf{F}^j T^j}{(j+1)!} \mathbf{G} \tag{9.2-39}$$

Since the series expression for **B** converges faster than (9.2-32), it suffices to determine L for $e^{\mathbf{F}T}$ as outlined above and apply the same value for **B**.

A second method of evaluating the matrices **A** and **B** also uses the truncated series approximation (9.2-33);* however, it differs in the method of evaluating the series and in the manner in which the series is terminated. The finite power series **M** is related to the identity

$$\mathbf{M} = \sum_{i=0}^{L} \frac{(\mathbf{F}T)^i}{i!}$$

$$= \left[\mathbf{I} + \mathbf{F}T\left(\mathbf{I} + \frac{\mathbf{F}T}{2}\left\{ \mathbf{I} + \frac{\mathbf{F}T}{3}\left[\mathbf{I} + \cdots + \frac{\mathbf{F}T}{L-1}\left(\mathbf{I} + \frac{\mathbf{F}T}{L} \right) \right] \cdots \right\} \right) \right] \tag{9.2-40}$$

Starting with the innermost factor, this nested product expansion lends itself very well to digital programming. Since the evaluation starts with the last term first the value of L, the number of terms of the series approximation, must be determined beforehand. The number of terms to be included is

*S. G. Hoppe et al., "A Feasibility Study of Self-learning Adaptive Flight Control for High Performance Aircraft," Report AFFDL-TR-67-18, Cornell Aeronautical Laboratory, February, 1967.

related empirically to the norm of the matrix $\mathbf{FT}$; that is,

$$L = \min\{3\,\|\mathbf{FT}\| + 6,\ 100\} \tag{9.2-41}$$

This relation assures that no more than a 100 terms are included. By experimental verification it can be demonstrated that the series $e^{\mathbf{FT}}$ is accurate to at least six significant figures.

The matrix $\mathbf{B}$ may also be computed by a similar expression by combining (9.2-40) with (9.2-35). This yields

$$\mathbf{B} = T\left(\mathbf{I} + \frac{\mathbf{FT}}{3}\left\{\mathbf{I} + \frac{\mathbf{FT}}{3}\left[\mathbf{I} + \ldots + \frac{\mathbf{FT}}{L-1}\left(\mathbf{I} + \frac{\mathbf{FT}}{L}\right)\ldots\right]\right\}\right)G \tag{9.2-42}$$

Because of the great similarity between (9.2-40) and (9.2-42) the evaluation of the two series may be easily combined into a single computer routine.

EXAMPLE 9.2-7

Determine the state transition matrix $\mathbf{A}$ for the simple harmonic oscillator. The state equations are given by Example 9.2-5.

$$\dot{x}_1 = -\omega^2 x_2$$
$$\dot{x}_2 = x_1$$

By analytical techniques we determine that

$$e^{\mathbf{F}t} = \begin{bmatrix} \cos\omega t & -\omega\sin\omega t \\ \frac{1}{\omega}\sin\omega t & \cos\omega t \end{bmatrix}$$

so that the difference equations are

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \cos\omega T & -\omega\sin\omega T \\ \frac{1}{\omega}\sin\omega T & \cos\omega T \end{bmatrix}\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

The results of this example are discussed next.

9.2-6 Comparison of Techniques

The Euler and Runge-Kutta methods are applicable to nonlinear systems, while the Tustin and state transition methods are restricted to linear systems. The Euler and Runge-Kutta methods require the preparation of an identical subprogram for the evaluation of derivatives; this subprogram is used concurrently with the execution of the integration programs. The Tustin and state transition methods require intermediate programs for the preparation

of the difference equations: the Tustin method a program to implement the Tustin substitution, and the state transition method one to determine the exponential matrices $e^{\mathbf{FT}}$ and $\int e^{\mathbf{FT}}$. These programs are run prior to the integration program. To use the Tustin method the description of the dynamic system must be available in transfer function form, whereas the state transition method requires a state model.

It is probably fair to state that with respect to convenience of application no method is particularly disadvantageous, as long as the digital computer is effectively utilized to carry out the computations.

Speed of Computation

The relative speed of computation of the four methods depends entirely upon the total number of instructions that require execution during the course of the program. Table 9.2-1 shows a listing of computer time elapsed during

Table 9.2-1

	Euler	Runge-Kutta	Tustin	State Transition
Compile time	0:57	1:40	0:59	1:46
Execution time	2:55	12:53	4:26	5:34
Load time	0:19	0:21	0:19	0:22
Total time	4:16	15:18	5:49	7:42

compile, load, and execution stages for identical problems* for which the four methods were used. Time is given in minutes and seconds. It is easily seen that the Euler method is the fastest and the Runge-Kutta method is the slowest, while the Tustin and state transition methods rank second and third. Since the Runge-Kutta method requires four times as many calculations (it is a four-step method) as the Euler method, it is easily explained that roughly four times as much time is required. The total time for the Tustin method is approximately 25 percent longer than the Euler, and the state transition method requires about 75 percent more time than the Euler method.

To draw any meaningful conclusion from this time summary one must also take into consideration the accuracy factor, for a method may require little time for implementation but may be marked by poor accuracy. This we shall examine now.

Accuracy

It goes without saying that accuracy is the most important factor in considering the selection of a numerical method for system simulation. In

*The problem referred to here is the linear oscillator and the computer used is an IBM 7044.

Table 9.2-2 Numerical Results of Four Methods

Time	Runge-Kutta	Euler	Tustin	State Transition
0	5.417	10.00	6.0	5.403
2	−4.010	0.	−2.8	−4.161
3	−9.695	−20.	−9.36	−9.900
4	−6.542	−40.	−8.432	−6.536
5	2.491	−40.	−7.584	2.837
6	9.161	0.	7.522	9.602
7	7.463	80.	9.785	7.540
8	−9.638	160.	4.220	−1.455
9	−8.417	160.	−4.721	−9.111
10	−8.166	0.	−9.885	−8.40
		(a) $T = 1$ second		
1	5.403	5.708	5.410	5.403
2	−4.161	−4.530	−4.146	−4.161
3	−9.900	−1.148	−9.896	−9.900
4	−6.536	−8.097	−6.562	−6.536
5	2.837	3.434	2.797	2.837
6	9.602	1.286	9.588	9.602
7	7.539	1.089	7.577	7.540
8	−1.455	−1.775	−1.389	−1.455
9	−9.111	−1.406	−9.080	−9.111
10	−8.391	−1.409	−8.436	−8.391
		(b) $T = .1$ second		
1	5.403	5.43	5.403	5.403
2	−4.161	−4.203	−4.161	−4.161
3	−9.900	−10.05	−9.900	−9.900
3.14	−10.000	−10.16	−10.00	−10.00
4	−6.536	−6.669	−6.536	−6.536
5	2.837	2.907	2.836	2.836
6	9.602	9.893	9.602	9.602
6.28	10.000	10.32	10.00	10.00
7	7.539	7.809	7.539	7.539
8	−1.455	−1.512	−1.454	−1.454
9	−9.111	−9.529	−9.111	−9.111
10	−8.391	—	—	—
		(c) $T = .01$ second		

the earlier paragraphs of this section we considered the solution of the differential equation

$$\ddot{x} + \omega^2 x = 0 \tag{9.2-10}$$

via the four methods under consideration. It is now our intention to compare the results.

Time	Runge-Kutta	Euler	Tustin	State Transition
1	5.403	5.405	5.403	5.403
2	−4.161	−4.165	−4.161	−4.161
3	−9.900	−9.915	−9.900	−9.900
3.14	−10.00	−10.02	−10.00	−10.00
4	−6.536	−6.549	−6.536	−6.536
5	2.837	2.844	2.837	2.837
6	9.601	9.630	9.601	9.601
6.28	10.00	10.03	10.00	10.00
7	7.539	7.565	7.539	7.539
8	−1.455	−1.461	−1.455	−1.455
9	−9.111	−9.152	−9.111	−9.111
10	−8.390	—	−8.390	−8.390
		(d) $T = .001$ second		
1	5.403	5.403	5.403	5.403
2	−4.161	−4.161	−4.160	−4.160
3	−9.898	−9.900	−9.897	−9.896
3.14	−9.998	−10.00	−9.997	−9.996
4	−6.535	−6.536	−6.534	−6.533
5	2.836	2.836	2.835	2.835
6	9.598	9.601	9.596	9.594
6.28	9.996	9.999	9.994	9.991
7	7.536	7.539	7.534	7.532
8	−1.454	−1.455	−1.454	−1.454
9	−9.106	−9.111	−9.104	−9.100
10	—	—	—	—
		(e) $T = .0001$ second		
1	5.400	5.400	5.401	5.393
2	−4.157	−4.157	−4.147	−4.147
3	−9.882	−9.882	−9.864	−9.848
3.14	−9.982	−9.982	−9.963	−9.945
4	−6.521	−6.521	−6.510	−6.490
5	2.828	2.828	2.815	2.812
6	9.568	9.568	9.533	9.501
6.28	9.963	9.963	9.926	9.889
7.00	7.508	7.508	7.481	7.446
		(f) $T = .00001$ second		

The solution of (9.2-10) represents a circle (for $\omega = 1$) when $\dot{x}$ is plotted versus x. The period of this circle is 2π seconds. Plotting the result therefore can offer a quick visual check on the quality of the solution.

Of interest here are the effects of truncation error and round-off on the accuracy of the solution. For this purpose we show the solution $x(t)$ for six choices of T in Table 9.2-2. The initial conditions are chosen as $x(0) = 10.0$

and $\dot{x}(0) = 0.0$ so that a circle of radius 10.0 results. The solution is run for roughly 10 seconds of time, sufficient to cover one full period. Solution values are shown at full second intervals and for $T \leq .01$, also for $t = 3.14$ and $t = 6.28$, which correspond to half and full periods of the circle.

We recall that error due to truncation of series approximation is present in the Euler, Runge-Kutta, and Tustin methods, but not in the state transition method. Therefore, for the largest value of the increment of integration ($T = 1.0$) we can expect that truncation will be a prominent factor in the Euler, Runge-Kutta, and Tustin methods, but not in the state transition method. Reviewing Table 9.2-2(a), we see that the Euler method produces an unstable solution; the Runge-Kutta and Tustin methods produce stable solutions but with considerable truncation error, while the solution under the state transition method is accurate to within the decimal places shown. We recall that the accuracy of the last method depends in this case only on the accuracy to which the series expansion of $e^{\mathbf{F}T}$ is computed; each entry in the matrix $e^{\mathbf{F}T}$ is accurate to within 10^{-6}. Thus, the solution generated by the state transition method may serve as a reference.

When the increment of integration is reduced to $T = .1$, the Runge-Kutta solution becomes exact to the places shown and the Tustin method is improved considerably. The solution generated by the Euler method appears stable but still suffers considerable truncation error.* These results are shown in Table 9.2-2(b).

When T is further reduced to $T = .01$ Runge-Kutta and Tustin methods are identical to the state transition method. The Euler method, however, still shows truncation error effects. See Table 9.2-2(c). Also shown are values of the solution at $t = 3.14$ and $t = 6.28$ for which the exact solutions are -10.00 and $+10.00$, respectively.

Table 9.2-2(d) shows the solutions for $T = .001$. The results indicate no error for the Runge-Kutta, Tustin, and state transition methods. The Euler method is now accurate to within two places.

A further reduction in the time increment to $T = .0001$ permits the generation of an exact solution (four significant figures) by the Euler method. On the other hand, the other three methods are beginning to show the effects of round-off. See Table 9.2-2(e).

When the increment is selected as small as $T = .00001$, round-off error becomes a significant influence in determining the quality of the solution. Table 9.2-2(f) demonstrates this. The results further indicate that the Runge-Kutta and Euler methods generate identical solutions. This supports the fact that the truncation error for $T = .00001$ is completely negligible in both these methods, and they provide equally good Taylor series approximations.

*Actually, the Euler method does not yield a stable difference equation for this case.

9.3 Use of the State Transition Method in Simulation Studies

The discrete transition method may be employed to handle a variety of problems. Its use is most suitable in the simulation of discrete-time systems. We refer here to two special cases in connection with simulation studies.

9.3-1 Conversion of Nonhomogeneous to Homogeneous State Transition Equations

Normally, the discrete state equivalent of a linear continuous plant with piecewise constant inputs is of the form

$$\mathbf{x}(k+1) = \mathbf{A}(T)\mathbf{x}(k) + \mathbf{B}(T)r(k) \tag{9.3-1}$$

An alternate approach to the solution of (9.3-1) may be followed when this equation represents the discrete-time equation of a transfer function, and the input to the transfer function is derived from a hold element, as shown in Figure 9.3-1. An example is now used to illustrate the procedure to be followed.

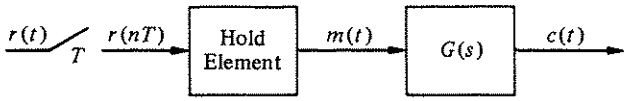


Figure 9.3-1. Linear system driven by hold element.

EXAMPLE 9.3-1

Consider the case when

$$G(s) = \frac{s+1}{(s+2)(s+10)} = \frac{s+1}{s^2+12s+20} \tag{9.3-2}$$

and a zero-order hold circuit is employed.

To derive $\mathbf{A}$ we represent this system using the nested programming method. The state equations are

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -12 & 1 \\ -20 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} m(t) \tag{9.3-3}$$

$$c(t) = x_1$$

Now $m(t)$ is given by

$$m(nT + \tau) = r(nT) \quad \text{for } 0 \leq \tau < T \tag{9.3-4}$$

i.e., $m(t)$ is a piecewise constant input. During any sampling period it is therefore possible to view $m(t)$ as the output of an integrator whose input is zero and whose initial condition is set to $r(nT)$ at the beginning of the nth sampling period. This is shown by the state variable diagram in Figure 9.3-2.

Consider now the combination of the state variable diagram for (9.3-3) and Figure 9.3-2, as shown in Figure 9.3-3. The state equations corresponding to this figure are

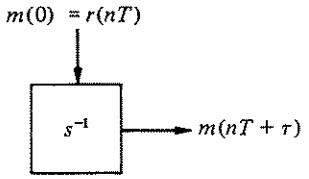


Figure 9.3-2. State variable diagram of zero-order hold.

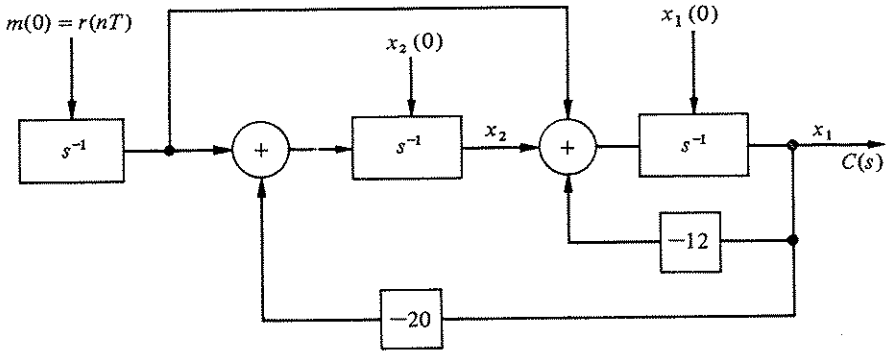


Figure 9.3-3. State variable diagram of $G(s)$ with zero-order hold.

$$\begin{bmatrix} \dot{m} \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -12 & 1 \\ 1 & -20 & 0 \end{bmatrix} \begin{bmatrix} m \\ x_1 \\ x_2 \end{bmatrix} \qquad 0 \leq t < T \tag{9.3-5}$$

$$c = x_1$$

The solution at the end of the first sampling period is

$$\begin{bmatrix} m(T) \\ x_1(T) \\ x_2(T) \end{bmatrix} = e^{\mathbf{F}_a T} \begin{bmatrix} m(0) \\ x_1(0) \\ x_2(0) \end{bmatrix} = e^{\mathbf{F}_a T} \begin{bmatrix} r(0) \\ x_1(0) \\ x_2(0) \end{bmatrix} \tag{9.3-6}$$

where $\mathbf{F}_a$ is the augmented system matrix [e.g. (9.3-5)].

Repeated application of (9.3-6) leads to the recursion relation

$$\begin{bmatrix} m(k+1) \\ x_1(k+1) \\ x_2(k+1) \end{bmatrix} = e^{\mathbf{F}_a T} \begin{bmatrix} r(k) \\ x_1(k) \\ x_2(k) \end{bmatrix} \tag{9.3-7}$$

$$c(k) = x_1(k)$$

In general, by augmenting the matrix $\mathbf{F}$ such that

$$\mathbf{F}_a = \begin{bmatrix} 0 & 0 \\ \mathbf{G} & \mathbf{F} \end{bmatrix}$$

the nonhomogeneous recursion equation (9.3-1) is changed into a homogeneous equation. The obvious advantage is that in the digital computer evaluation of (9.3-7) only one exponential series needs evaluation. It is pointed out, however, that (9.3-6) is completely equivalent to using the nonhomogeneous equation

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = e^{\mathbf{F}T} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \left\{ \int_0^T e^{\mathbf{F}\tau} \begin{bmatrix} 1 \\ 1 \end{bmatrix} d\tau \right\} r(k)$$

or

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \mathbf{A}(T) \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \mathbf{B}(T) r(k) \tag{9.3-8}$$

9.3-2 Evaluation of System Response Between Sampling Periods

Suppose that it is necessary to determine the output of a system that receives piecewise constant inputs at intervals of T seconds at times corresponding to subintervals of T_s seconds such that

$$T = \alpha T_s, \quad \alpha \text{ an integer}$$

It is possible to use either (9.3-1) or (9.3-7) to accomplish this. For instance, let $\alpha = 5$; then $T_s = .2T$.

Approach A

Evaluate $\mathbf{A}(T_s) = e^{\mathbf{F}T_s}$ and $\mathbf{B}(T_s)$. Then at time $t = nT$ use the recursion equation (9.3-1) five times; e.g.,

$$\begin{aligned}
\mathbf{x}(nT + T_s) &= \mathbf{A}(T_s)\mathbf{x}(nT) + \mathbf{B}(T_s)r(nT) \\
\mathbf{x}(nT + 2T_s) &= \mathbf{A}(T_s)\mathbf{x}(nT + T_s) + \mathbf{B}(T_s)r(nT) \\
\mathbf{x}(nT + 3T_s) &= \mathbf{A}(T_s)\mathbf{x}(nT + 2T_s) + \mathbf{B}(T_s)r(nT) \\
\mathbf{x}(nT + 4T_s) &= \mathbf{A}(T_s)\mathbf{x}(nT + 3T_s) + \mathbf{B}(T_s)r(nT) \\
\mathbf{x}[(n+1)T] &= \mathbf{x}(nT + 5T_s) = \mathbf{A}(T_s)\mathbf{x}(nT + 4T_s) + \mathbf{B}(T_s)r(nT)
\end{aligned} \tag{9.3-9}$$

These five equations are repeated every main sampling interval with a new input supplied at times $t = nT$, $n = 1, 2, \ldots$.

Approach B

Evaluate $\mathbf{A}_a(T_s) = e^{\mathbf{F}_a T_s}$. Then use the recursion (9.3-7) five times; e.g.,

$$\begin{aligned}
\begin{bmatrix} m(nT + T_s) \\ \mathbf{x}(nT + T_s) \end{bmatrix} &= e^{\mathbf{F}_a T_s} \begin{bmatrix} r(nT) \\ \mathbf{x}(nT) \end{bmatrix} \\
\begin{bmatrix} m(nT + 2T_s) \\ \mathbf{x}(nT + 2T_s) \end{bmatrix} &= e^{\mathbf{F}_a T_s} \begin{bmatrix} r(nT) \\ \mathbf{x}(nT + T_s) \end{bmatrix} \\
&\vdots \\
\begin{bmatrix} m[(n+1)T] \\ \mathbf{x}[(n+1)T] \end{bmatrix} = \begin{bmatrix} m(nT + 5T_s) \\ \mathbf{x}(nT + 5T_s) \end{bmatrix} &= e^{\mathbf{F}_a T_s} \begin{bmatrix} r(nT) \\ \mathbf{x}(nT + 4T_s) \end{bmatrix}
\end{aligned} \tag{9.3-10}$$

These equations are repeated every main sampling interval with a new input applied at times given by $t = nT$, $n = 1, 2, \ldots$. Of course, the variable $m(nT + T_s)$, $m(nT + 2T_s)$, etc. is not used.

9.4 Digital Computer Simulation of a Digital Control System

As an illustration of the use of numerical integration techniques, we consider the digital simulation of a computer control system. Consider the system shown in Figure 9.4-1. The transfer function of the continuous system is given as

$$G(s) = \frac{k_0(s+1)}{s(s^2 + 4s + 10)} \tag{9.4-1}$$

A zero-order hold element is used. The digital recursion equation is given by

$$\begin{aligned}
e_2(kT) = e_1(kT) &+ .2e_1[(k-1)T] - .2e_1[(k-2)T] \\
&- .6e_2[(k-1)T] + .15e_2[(k-2)T]
\end{aligned} \tag{9.4-2}$$

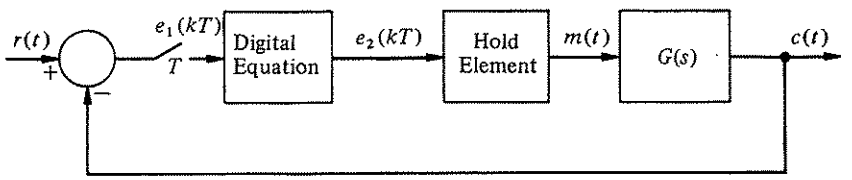


Figure 9.4-1. Computer control system.

It is desired to set up a digital computer simulation to analyze the response of the system for a variety of sampling periods, system gains, and step inputs. It is, therefore, necessary to provide built-in flexibility in the program to accommodate an analysis with respect to these three factors.

Using direct programming, we form the state equations of (9.4-1), which are

$$\begin{aligned}
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} &= \begin{bmatrix} -4 & -10 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} m(t) \\
c(t) &= k_0(x_2 + x_3)
\end{aligned} \tag{9.4-3}$$

Thus

$$\mathbf{F} = \begin{bmatrix} -4 & -10 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{G} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

To compute the transition matrices $\mathbf{A}(T)$ and $\mathbf{B}(T)$ we may use a computer routine such as

```
SUBROUTINE  MATEXP  (F,G,A,B,N,M,T)
```

described in Appendix 2A.

The linear recursion equation of the digital computer can be simulated in the very form it is given.

A general flow chart for the simulation is shown in Figure 9.4-2.

After the flow chart, the program of the simulation is presented.

Definition of symbols:

T = sampling period
GAIN = gain k_0 of system
NRUN = number of runs made
TIME = time of response
XIN = magnitude of step input
NDATA = total number of runs to be made
TIMEFN = total length of each run

```
      DIMENSION F(3,3),G(3), A(3,3), B(3),E1(3),E2(3),X(3)
      READ (5,1) F,G,NDATA,TIMEFN
C     PARAMETERS FOR THIS RUN
   21 READ (5,2) T,GAIN,XIN
C     COMPUTE A(T) AND B(T)
      DO 10 I = 1,3
      B(I) = G(I)
      DO 10 J = 1,3
   10 A (I,J) = F(I,J)
      CALL TRANS (A , B ,3,T,15)
C     INITIALIZE NEW RUN
      NRUN = NRUN + 1
      C = 0.0
      TIME = 0.0
      XM = 0.0
      DO 11 I = 1,3
      E1(I) = 0.0
      E2(I) = 0.0
   11 X(I) = 0.0
C     COMPUTE COMPUTER INPUT
   20 E1(3) = E1(2)
      E1(2) = E1(1)
      E1(1) = XIN-C
C     COMPUTE COMPUTER OUTPUT
      E2(3) = E2(2)
      E2(2) = E2(1)
      E2(1) = E1(1)+.2*E1(2)-.2*E1(3)-.6*E2(2)+.15*E2(3)
C     COMPUTE NEW PLANT INPUT
      XM = E2(1)
C     COMPUTE NEW PLANT OUTPUT
      DO 12 I = 1,3
      X(I) = X(I)+B(I)*XM
      DO 12 J = 1,3
   12 X(I) = X(I)+A(I,J)*X(J)
      C = GAIN*(X(2)+X(3))
C     PRINT OUTPUT
      TIME = TIME+T
      WRITE(6,3) TIME,E1(1),XM,C
C     TEST IF THIS RUN IS COMPLETED
      IF(TIME.LT.TIMEFN) GO TO 20
C     TEST IF A NEW RUN IS TO BE INITIATED
      IF(NRUN. LT. NDATA) GO TO 21
      STOP
```

The program is written in FORTRAN. Not shown are FORMAT statements.

The digital simulation illustrated above can also be carried out by using

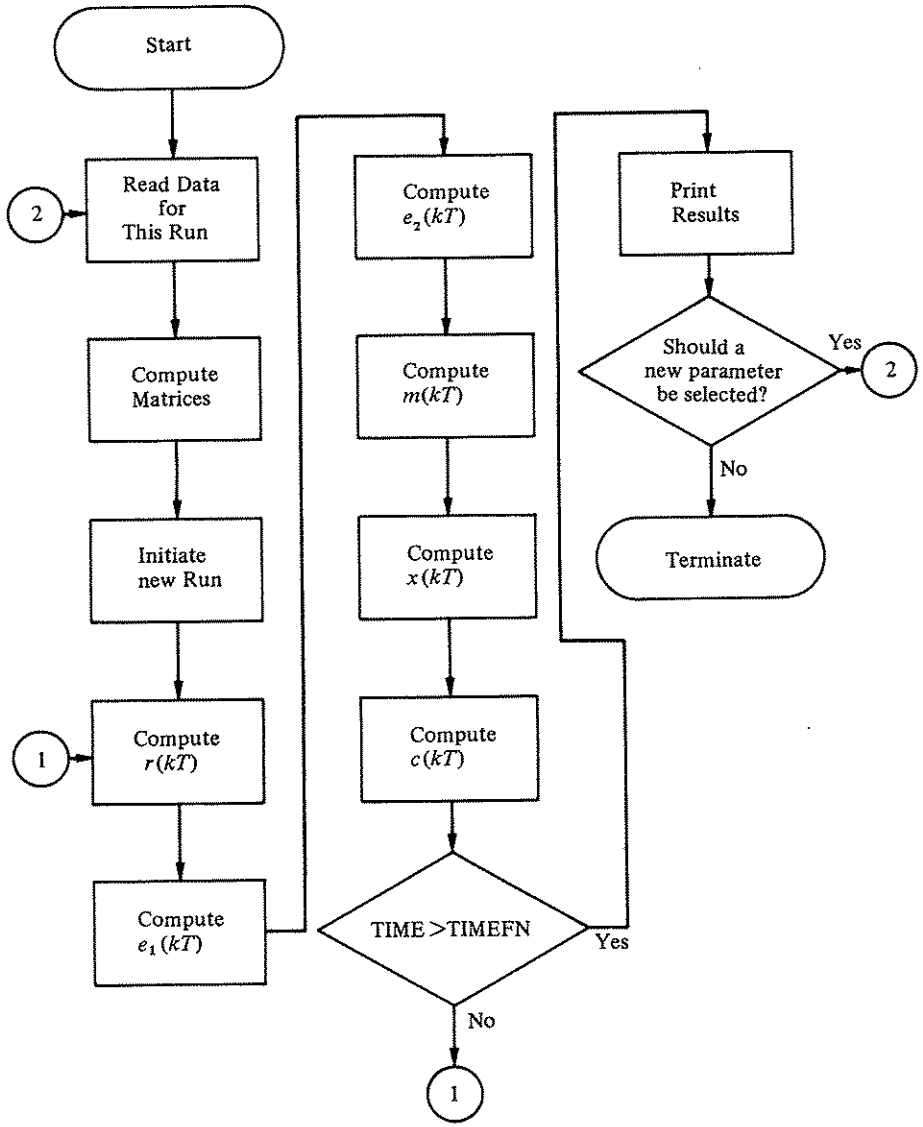


Figure 9.4-2. Flow chart for digital simulation.

the augmented system matrix $\mathbf{F}_a$, since the input to the plant $G(s)$ is a piecewise constant input. The augmented matrix is

$$\mathbf{F}_a = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -4 & -10 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

9.4-1 Other Ways of Implementing a Digital Computer Simulation of a Discrete-time System

The last section demonstrated a simulation procedure of computer control systems by representing the continuous-time parts of the system by discrete-time state transition matrices for the purpose of calculating their response at discrete-time intervals. An alternate approach to the computation of the response of the continuous-time part of a mixed system is the utilization of numerical integration techniques such as the Euler or Runge-Kutta procedures. To follow this approach it is convenient to construct a subroutine that will generate a solution of the differential equations representing the plant over a single sampling interval or some subinterval. SUBROUTINE INTFUN is such a program. (See Appendix 9A.)

The use of the subroutine is illustrated by repeating the simulation of the problem of the previous section. The flow chart is given as shown in Figure 9.4-3. The program follows on page 403.

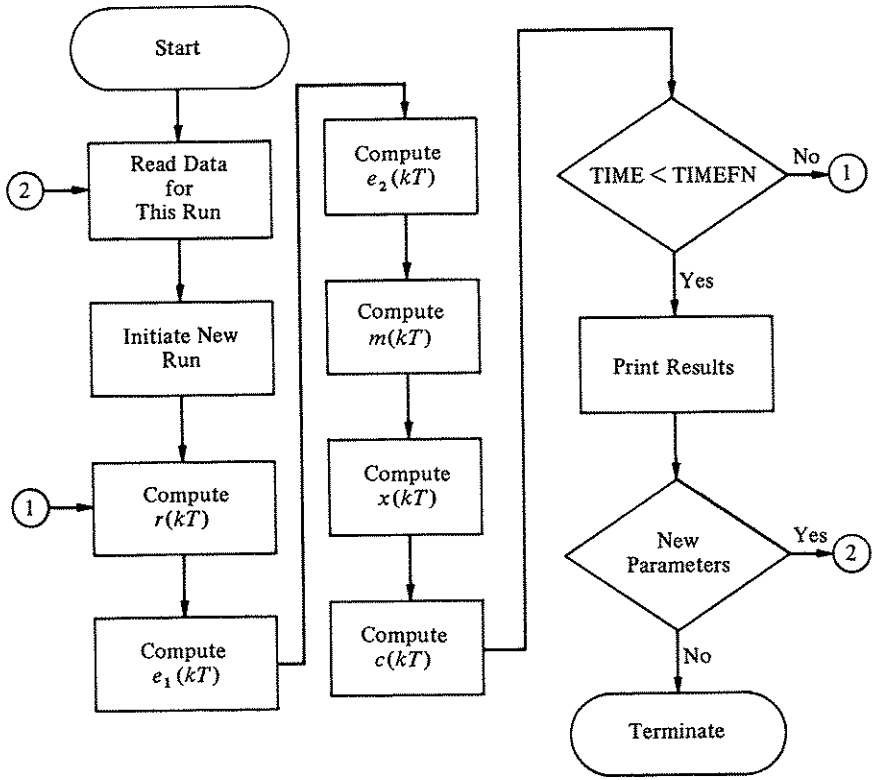


Figure 9.4-3. Flow chart for digital simulation.


```
      DIMENSION E1(3),E2(3), X(4)
      READ (5,1) NDATA,TIMEFN
C     PARAMETERS FOR THIS RUN
21    READ (5,2) T,GAIN,XIN
C     INITIALIZE NEW RUN
      NRUN = NRUN+1
      C = 0.0
      FIVET = 5.*T
      TIME = 0.0
      DO 11 I = 1,3
      E1(I) = 0.0
      E2(I) = 0.0
11    X(I) = 0.0
C     NEW COMPUTER INPUT
20    E1(3) = E1(2)
      E1(2) = E1(1)
      E1(1) = XIN-C
C     NEW COMPUTER OUTPUT
      E2(3) = E2(2)
      E2(2) = E2(1)
      E2(1) = E1(1)+.2*E1(2)-.2*E1(3)-.6*E2(2)+.15*E2(3)
C     NEW PLANT INPUT
      X(4) = E2(1)
C     INTEGRATE PLANT FOR NEXT INTERVAL WITH 5 POINTS
      CALL INTFUN(X,TIME,FIVET,N)
C     NEW PLANT OUTPUT
      C = GAIN*(X(2)+X(3))
C     PRINT OUTPUT
      TTOTAL = TTOTAL+T
      WRITE (6,3) TTOTAL,E1(1),E2(1),C
C     TEST IF THIS RUN IS COMPLETE
      IF (TTOTAL.LT.TIMEFN) GO TO 20
C     TEST IF A NEW RUN IS TO BE INITIATED
      IF (NRUN.IT.NDATA) GO TO 21
      END
      STOP
```

In addition to the above program we need the statements for subroutine DERIV; these are

```
      SUBROUTINE DERIV(X,TIME,DX)
      DIMENSION X(4),DX(3)
      DX(1) = -4.*X(1)-10.*X(2)+X(4)
      DX(2) = X(1)
      DX(3) = X(2)
      RETURN
      END
```

The use of the Runge-Kutta method to provide a solution for the state equations of the plant is particularly appropriate when these equations are nonlinear. In that case, a state transition matrix cannot be obtained.

9.5 Analog Computer Simulation of Systems

The analog computer has been widely employed in the simulation of complex dynamic systems. Simulation is closely related to the solution of state equations by analog computer. However, simulation implies the solution of dynamic systems in a broader sense: It involves the solution of the state equations representing appropriately chosen subsystems of the system with interconnections that are in a one-to-one correspondence with the topology of the system. Let us consider the simulation of the feedback control system shown in Figure 9.5-1.

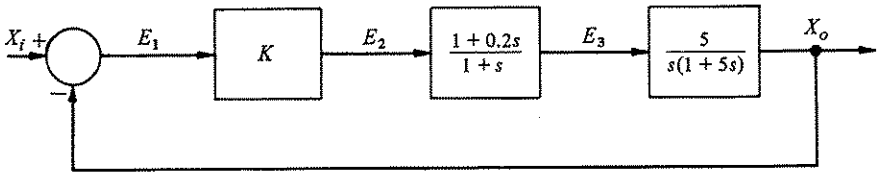
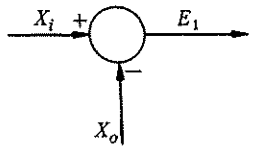


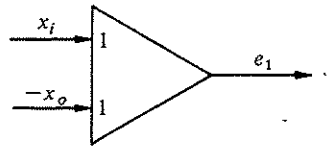
Figure 9.5-1. Block diagram of a control system.

We can dissect the system into four subassemblies.

1. 

$$E_1 = X_i - X_o$$

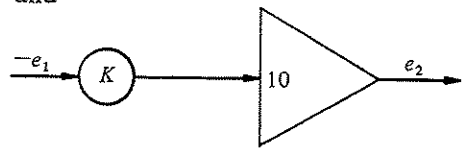
In terms of a computer diagram,



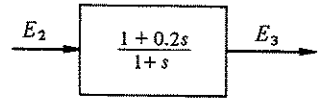
2. 

$$E_2 = KE_1$$

and



$$0 \leqslant K \leqslant 10$$

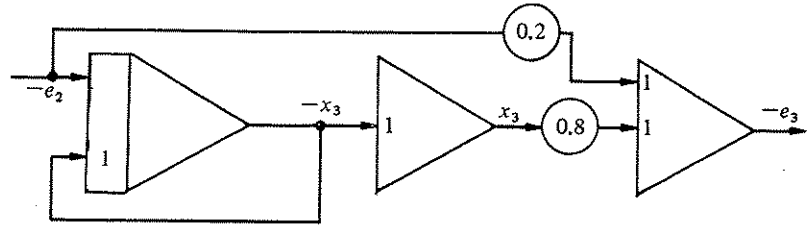
3. 

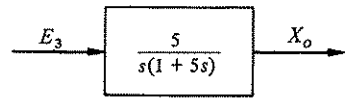
The state equations are

$$\frac{d}{dt}x_3 = -[x_3 - e_2(t)]$$

$$e_3(t) = -[-.8x_3 - .2e_2(t)]$$

Therefore, the corresponding analog diagram is



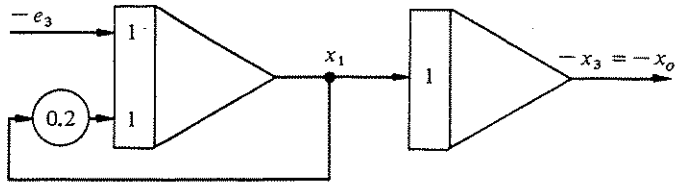
4. 

The state equations are

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -\left\{\begin{bmatrix} +.2 & 0 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} e_3(t)\right\}$$

$$x_o = x_2$$

and the diagram is



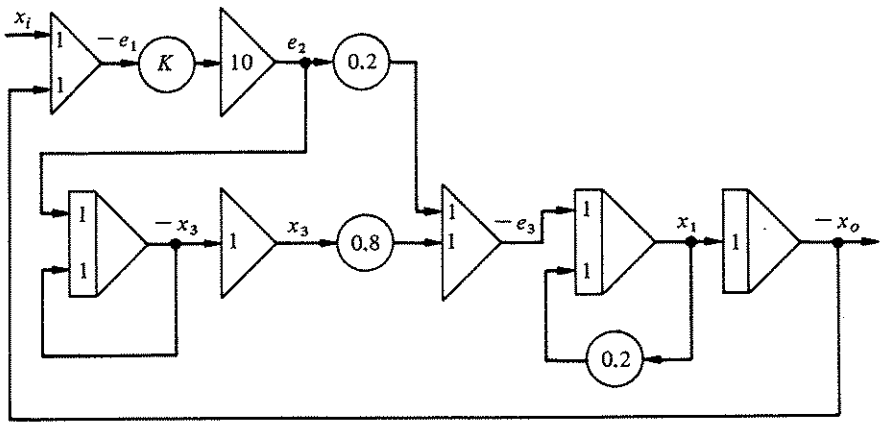


Figure 9.5-2. Analog computer diagram for feedback system.

These four subassemblies may now be integrated into a single diagram representing the entire system. The result is shown in Figure 9.5-2. The analog computer program thus obtained is a simulation of the feedback system. The structural correspondence between the computer diagram and the block diagram of the system is easily recognized. The diagram may be simplified by removing the second and fifth summer-amplifier and combining their functions with those adjacent to them. Figure 9.5-3 shows the simplified

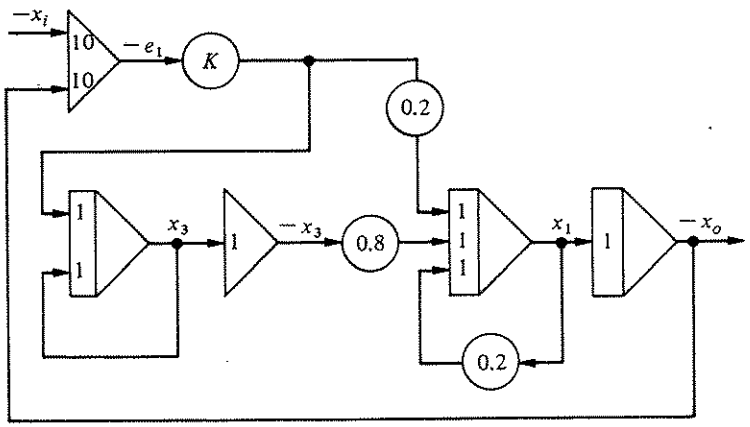


Figure 9.5-3. Simplified computer diagram.

diagram. Although this simplification requires the elimination of the variables e_2 and e_3, it is consistent with good analog computer practice of constructing a diagram of minimum complexity. However, the main structure is still preserved.

In contrast to the simulation approach outlined above, we shall consider

a straight state equation solution of the same problem. The combined "closed-loop" transfer function is given by

$$X_o(s) = \frac{K(.2s + 1)}{s^3 + 1.2s^2 + .2(K+1)s + K} X_i(s)$$

The corresponding state equations by direct programming are

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.2 & -.2(K+1) & -K \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x_{in}(t)$$

$$x_o(t) = K[0 \quad .2 \quad 1]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The analog computer diagram corresponding to these equations is shown in Figure 9.5-4.

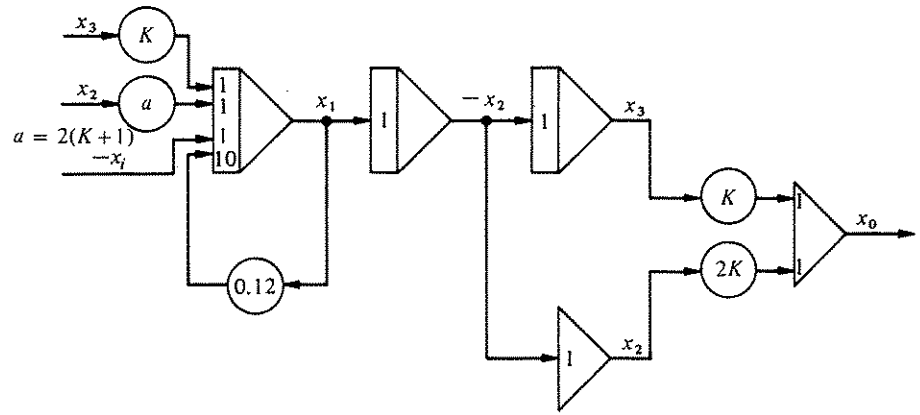


Figure 9.5-4. Direct approach of feedback system solution.

Figures 9.5-3 and 9.5-4 show computer diagrams that represent two approaches to the solution of a dynamic system. Although both will yield identical results in terms of the solution, the simulation approach is usually preferred. Not only is the structure of the system preserved in the computer diagram, but also it is much more readily adaptable for design considerations. For instance, if the parameter K, which represents the gain of the system, is to be selected according to some response specifications, only one potentiometer has to be adjusted on the simulation diagram, against four on the state equation approach.

The example developed above is but a single illustration of analog computer simulation techniques. However, it points out one important characteristic: the breakdown of the overall system into subassemblies, each one of which is modeled individually.

The simulation of systems by analog computer is particularly useful when the system contains isolated nonlinearities. Some subassemblies will be characterized by linear state equations, while others may contain typical nonlinearities that are much more easily dealt with on a subassembly basis than on a complete system approach.

The simulation of discrete systems or sampled-data systems may be most effectively carried out by the use of a hybrid computer if the system operates in continuous time while other parts function in a discrete manner. We shall turn our attention to hybrid computer simulation in the next section.

9.6 Digital Analog-system Simulators

The application of digital computation methods in the solution of technical problems has become widespread. The increased use of digital computation methods is largely due to the availability of larger, faster, and more powerful digital computers and the development of application languages such as FORTRAN and ALGOL. These languages enable the user to express his problem in a computer language closely allied to the language of his own field, thus considerably simplifying the programming process.

During the last ten years an additional class of application languages has been developed which are proving to be very useful. These are the digital analog simulators. Nearly two dozen of these languages have been created, their level of capability roughly corresponding to that of digital computer technology. In an evolutionary manner, two simulation systems that have gained widespread acceptance because of their simplicity and overall effectiveness are MIDAS (Modified Integrator Digital Analog Simulator) and MIMIC. Of these the MIMIC language provides a simple substitute for the hybrid computer, as far as the simulation of discrete and sampled-data systems is concerned. What makes it potentially even more attractive than hybrid computer programming is the ease of programming, the elimination of the need for scaling, and the absence of sign reversals through operational amplifiers.

By education and job experience, the systems designer tends to visualize a system as a complex of subsystems. In this context, a computer control system may well be described by a block diagram consisting of interconnected blocks, each one of which designates a subsystem. These blocks may contain s-transfer functions, z-transfer functions, nonlinear functions, logic functions,

time-varying constants, and other characteristics typically found in a large-scale system.

By its very organization, the hybrid computer provides this building block capability. Digital analog simulation languages provide the means of programming a digital computer like a hybrid or analog computer. The problem is programmed on the digital computer in a manner closely approaching an analog computer solution. The user has at his disposal a set of predefined blocks from which he can assemble the system to be simulated. The blocks perform the same functional operations as standard analog computers, such as integration, multiplication, function generation, switching relays, and the like. In addition, hold circuits and z-transfer functions may be simulated. The assembly of these blocks, corresponding to the patching of the analog computer, is performed by a sequence of connection statements.

Before we proceed with the detailed description of MIMIC, we consider a brief history of the development of digital simulator languages.

9.6-1 Brief History and Recent Developments

The first published account of work on digital analog simulation was presented by R. G. Selfridge.* His work was motivated by the need to simulate larger problems than his analog computer could handle and to achieve better accuracy. His program was developed for one of the early computers without the advantage of automatic compilers such as FORTRAN. Adapting the digital computer to block diagram organization was his major contribution and is the basis for all of the subsequent analog simulator programs. Selfridge's program was an interpretive routine, which means that it accepted and executed certain pseudo-instructions without producing a machine language translation. Also, all computation was done in fixed-point arithmetic, and the problem variables had to be scaled to a definite maximum value.

Digital simulator development has paralleled the general development of digital computers. Programming aids such as floating-point arithmetic and automatic compilation are used to great advantage by the newer simulation programs such as MIMIC, MIDAS,† DAS,‡ PACTOLUS,§ and DSL 90.‖ Although these programs differ in format and language, they all retain

*R. G. Selfridge, "Coding a General-Purpose Digital Computer to Operate as a Differential Analyzer," *Proceedings 1955 Western Joint Computer Conference* (IRE).

†R. T. Harnett et al., "MIDAS . . . An Analog Approach to Digital Computation," *Simulation*, Vol. 3, No. 3, September 1964.

‡R. A. Gaskill et al., "DAS—A Digital Analog Simulator," *AFIPS Conference Proc.*, Vol. 23, p. 83.

§R. D. Brennan and S. Harlan, "PACTOLUS—A Digital Analog Simulator Program for the IBM 1620," *AFIPS Conference Proc.*, Vol. 26, October 1964.

‖W. M. Syn and D. G. Wyman, *DSL-90, User's Guide*, SHARE Library 3358.

the block organization feature. The PACTOLUS program exhibits one of the latest innovations: man-machine interplay. This is a very desirable feature, but it is not practical, since most large computer installations do not allow this procedure.

The DAS (Digital Analog Simulator) is structurally classified as a compiler. In addition to providing a workable and easy-to-use simulation language, it represents the forerunner of the MIDAS program.

9.6-2 MIMIC—A Simulator Language*

The development of simulator languages is an evolutionary process. At the time of this writing it is fair to say that MIMIC is the most versatile and effective simulation language. It represents a direct descendant of MIDAS, with some basic modifications and improvements. In this section a brief description of MIMIC and its use will be given. The description will be only sufficiently complete to permit the reader to obtain a basic appreciation of the operation of the program. For a total understanding only a manual of operation will suffice.

Illustrative Example

The coding of a simulation problem in the MIMIC language is carried out in terms of FORTRAN-like statements. As an introductory example, consider the control system shown in Figure 9.6-1. To code this problem in

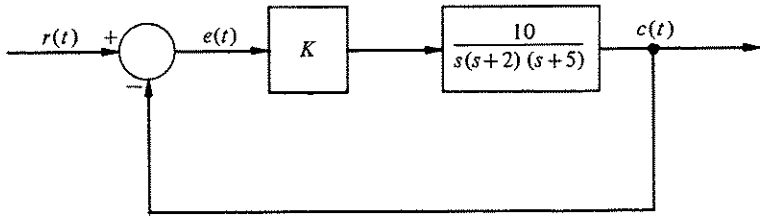


Figure 9.6-1. Simple control system.

MIMIC, we proceed in a manner similar to an analog computer simulation. For each block in the block diagram, a state model is developed. Using direct programming, for the block relating $c(t)$ to $m(t)$, we have

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -7 & -10 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} m(t) \qquad (9.6\text{-}1)$$

$$c(t) = 10x_3$$

*H. G. Peterson and F. J. Sanson, "MIMIC," *A Digital Simulator Program*, SESCA Internal Memo 65-12.

The problem would be programmed by keypunching the following statements

Result	Expression
10	19
E	R−C
XM	AK*E
X1	INT(−7.*X1−10.*X2+1.*XM,0.)
X2	INT(X1,0.)
X3	INT(X2,0.)
C	10.*X3

The relationship between the coding and the equations is so similar to FORTRAN programming that it should not require explanation.

The six equations may be reduced to three, as follows:

Result	Expression
10	19
X1	INT(−7.*X1−10.*X2+AK*(R−10.*X3,0.))
X2	INT(X1,0.)
X3	INT(X2,0.)

The block diagram of Figure 9.6-1 may be restructured to contain only first-order transfer functions, such as those shown in Figure 9.6-2. Then it is possible to utilize a MIMIC function that computes the input/output relations for a first-order transfer function (FTR). The operation of this function is illustrated by still another coding version of the same problem. The FTR solves a first-order differential equation.

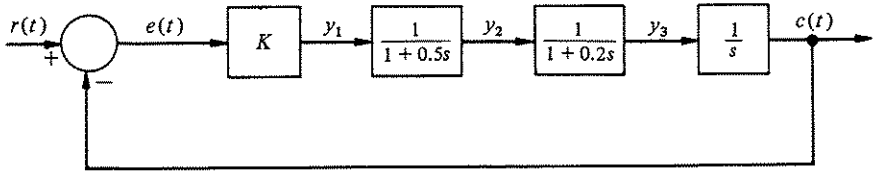


Figure 9.6-2. Alternate block diagram.

Result	Expression
10	19
E	R−C
Y1	K*E
Y2	FTR(Y1,.5)
Y3	FTR(Y2,.2)
C	INT(Y3,0.)

As is illustrated by the above example, it is possible to code a MIMIC simulation program in various ways: directly from a set of differential equations, or directly from a block diagram, or a combination of both. Obviously, a great degree of flexibility is available to the programmer. Whatever approach is used, the computer will automatically sort the instructions into proper order for sequential solution and then proceed to solve the equations.

9.6-3 Selected Features of MIMIC

Variables

As in FORTRAN, a group of from 1 to 6 alphameric characters constitutes the name of a variable in a MIMIC program. There are six reserved names; they are

T	the independent variable
DT	the amount T changes between printouts
DTMAX	the maximum integration step size allowed
DTMIN	the minimum integration step size allowed
TRUE	a logic constant that always has a "true" value
FALSE	a logic constant that always has a "false" value

Integrator with Mode Control

The mode of each integrator in MIMIC can be individually controlled. These are the RESET, OPERATE, and HOLD modes. A completely specified integrator function is given as

Result	Expression
10	19
R	INT(A,B,C,D)

Here the variable A is integrated with initial condition B. The mode is controlled by the variables C and D, according to Table 9.6-1.

Table 9.6-1 Integrator Mode Control

D \ C	TRUE	FALSE
TRUE	OPERATE	HOLD
FALSE	RESET	OPERATE

Logic Control Variables

MIMIC permits the use of logical variables. Logical variables may assume the value TRUE (1) or FALSE (0). They may be generated by use of the function switch FSW or logical switch LSW. One of the uses of the logical variables is to provide a control over the execution of expressions. If a logical variable is entered in the LCV column (column 2-7), the expression on that line will be evaluated when the control variable is TRUE and bypassed when the control variable is FALSE.

Subprograms

Subprograms in the style of a FORTRAN subroutine may be included in a MIMIC program. A subprogram must first be defined by putting the expressions comprising the subprogram between a BSP (begin subprogram) and an ESP (end subprogram). The subprogram name is entered in the result column of the BSP and ESP cards. The inputs to the subprogram are specified as arguments of BSP, while the outputs of the subprogram are obtained as arguments of ESP. To use a subprogram, two other control statements, e.g., CSP and RSP, are used. The name of the subprogram is entered in the result column of the CSP card; the output of the subprogram will be demonstrated in a subsequent example.

Function Switch

The function switch is designed to generate logical variables. It is used as follows:

Result	Expression
10	19
XX	FSW(A,B,C,D)

The result XX is a logical variable that is equal to B, C, or D, depending on whether $A < 0$, $A = 0$, or $A > 0$, respectively.

Many other functions are defined for use by MIMIC, including, for instance, ZOH (zero-order hold), TDL (time delay), TAS (track and store). In addition to the many operational functions, there are input/output functions to generate outputs in printed or plotted form.

EXAMPLE 9.6-1

Develop a MIMIC program to simulate the computer control system shown in Figure 9.6-3.

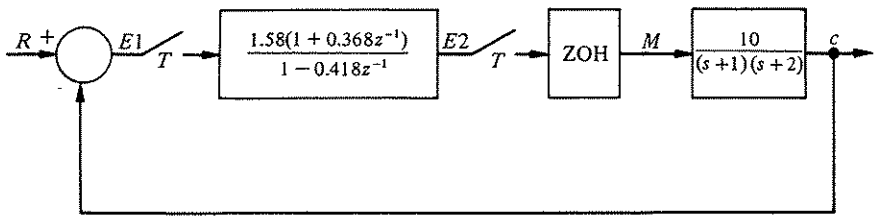


Figure 9.6-3. Computer control system.

It is clear that the operations contained in blocks defining the digital computer program and the zero-order hold are executed only once every sampling period. It would, therefore, be appropriate to introduce a subprogram for these operations and restrict its execution by a logical variable.

A suggested MIMIC program for the entire system which is complete except for initializing and input/output statements is given by

Logical Variable 2	Result 10	Expression 19
	DIG	BSP(R1)
	X	X+TSAM
	C2	C1
	R2	R1
	C1	1.58*R1−.58*R2−.418*C2
	DIG	ESP(C1,X)
	SAMPLE	FSW(T−X,FALSE,TRUE,TRUE)
SAMPLE	DIG	CSP(E1)
		RSP(M1)
	M2	5.*M1
	M3	FTR(M2,1.)
	C	FTR(M3,5.)
	E1	R−C
		FIN(T,5.)
		END

The logical control variable will be TRUE once every TSAM seconds, the length of the sampling period. The program is terminated by the FIN function, when T = 5 seconds.

This introduction to the digital simulation language MIMIC is very brief. It goes without saying that the study of a complete reference manual is required to use MIMIC intelligently. But the brief examples serve to point out the ease with which facility in the use of MIMIC can be reached. MIMIC and other languages like it represent a powerful means of digitally simulating systems.

9.7 Hybrid Computer Techniques and Applications in Simulation

Hybrid computers—computers integrally composed of analog and digital computers—have been developed since the late 1950's. The motivation for this type of computer is founded in the need for high-speed computation and extensive memory and logic capability in large-scale simulations. Utilized to best advantage in this combination are the high-speed simulation capability of the analog computer and the arithmetic, logic capability, and memory of the digital computer.

Cursory references were made to the use of a hybrid computer in previous chapters concerning the analysis of a sampled-data system. This section will give more complete coverage of the characteristics and applications of hybrid computers.

9.7-1 The Organization of a Hybrid Computer

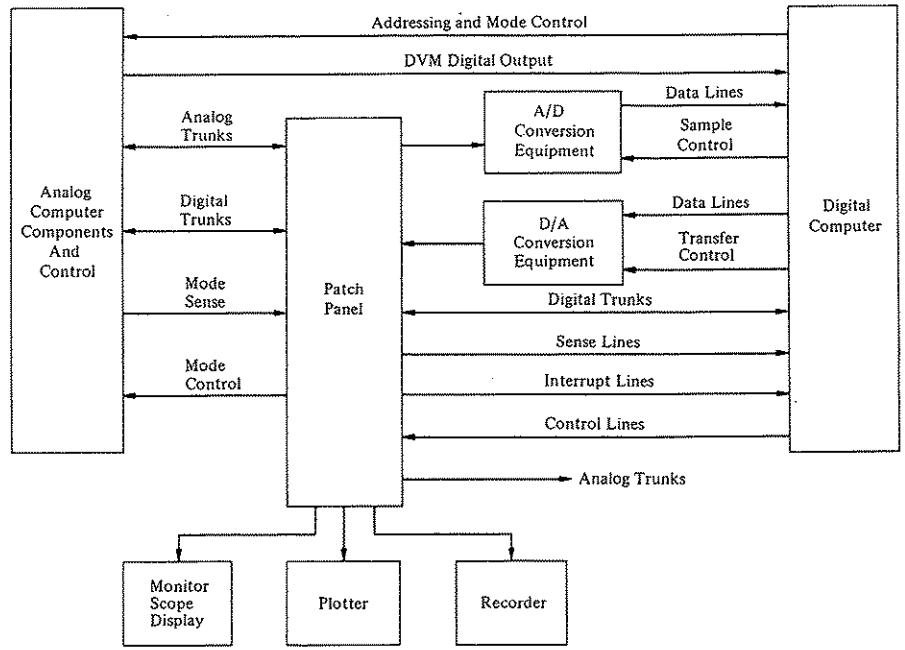


Figure 9.7-1. A typical hybrid computing system.

As is shown in Figure 9.7-1, the hybrid computer is composed of three major parts:

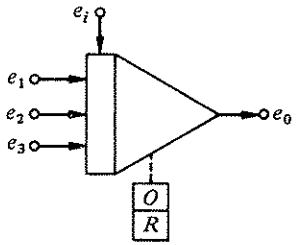
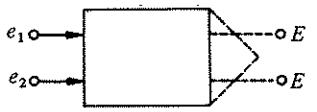
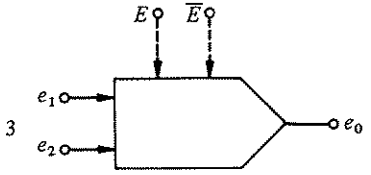
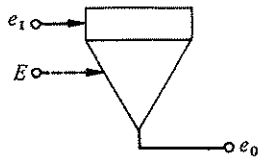
1. A general-purpose digital computer.
2. A general-purpose analog computer.
3. A linkage system to provide for exchange of data and control information between the computers.

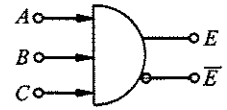
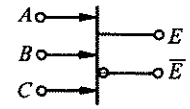
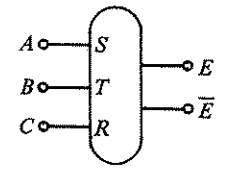
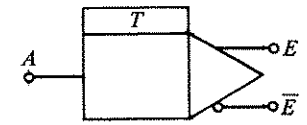
We now describe these parts in more detail.

The Analog Computer

The basic computing elements of the analog computer consist of the components shown in Table 9.7-1. The computer is provided with electronic mode control of all time-dependent components, particularly the integrators. All three modes, RESET, HOLD, and COMPUTE, are controllable either by patchable digital logic timing circuits, by commands from the digital computer, or by manual pushbutton control. By means of special forced-

Table 9.7-1 A Selection of Hybrid Components and Their Symbols

No.	Symbol	Description
1		*Integrator:* Separately controlled through its OPERATE (O) and RESET (R) inputs. $(O, R) = (0, 1)$, RESET $e_0 = -e_i$ $(O, R) = (1, 0)$, OPERATE $e_0 = -\frac{1}{RC}\int \sum e_k dt$ $(O, R) = (0, 0)$, HOLD $e_0 =$ constant
2		*Comparator:* If $e_1 > e_2$, $E = 1$ ($\bar{E} = 0$) If $e_1 < e_2$, $E = 0$ ($\bar{E} = 1$)
3		*Switch:* If $E = 1$ ($\bar{E} = 0$), $e_0 = e_1$ If $E = 0$ ($\bar{E} = 1$), $e_0 = e_2$
4		*Track-store:* If $E = 1$, $e_0 =$ constant (OPERATE) If $E = 0$, $e_0 = e_1$ (RESET)

No.	Symbol	Description
5		*AND Gate:* $E = 1$ if $A = B = C = 1$ $E = 0$ if either A, B, or $C = 0$
6		*OR Gate:* $E = 1$ if either A, B, or $C = 1$ $E = 0$ if $A = B = C = 0$
7		*Flip-flop:* $A = 1$, $C = 0$: $E = 1$—set flip-flop $A = 0$, $C = 1$: $E = 0$—clear flip-flop $A = C = 0$: $E = 1$, 0—store 1 or 0 $A = C = 0$, $B = 0 \longrightarrow 1$: complement E
8		*Monostable:* $A = 0 \longrightarrow 1$: $E = 1$ for T seconds; otherwise $E = 0$

charging circuits the integrators may be operated at high speed, cycling up to 10,000 times a second between the three modes. The mode selection of the integrators is carried out by logic signals when under the control of digital logic.

In addition to the electronic control of integrators, electronic switches are provided. The operation of such a switch is defined by the description of Table 9.7-1.

A special kind of hybrid computer component is the track-store circuit. Since it is foreign to a conventional analog computer, we shall describe its operation here. The track-store circuit is essentially an integrator under electronic mode control with input solely through the initial condition input terminal. When the integrator is in RESET mode it "tracks" the input, whereas in OPERATE it "stores." To illustrate the operation of a track-store unit, consider the waveforms of Figure 9.7-2. It shows a waveform $r(t)$ connected to the input of a track-store unit that is controlled by a timing signal E. The output $c(t)$ is seen in Figure 9.7-2(c).

Thus when $E = 1$ the unit is in STORE mode and when $E = 0$ the unit is in TRACK mode.

The voltage levels of the logic signals are usually such that logic 0 cor-

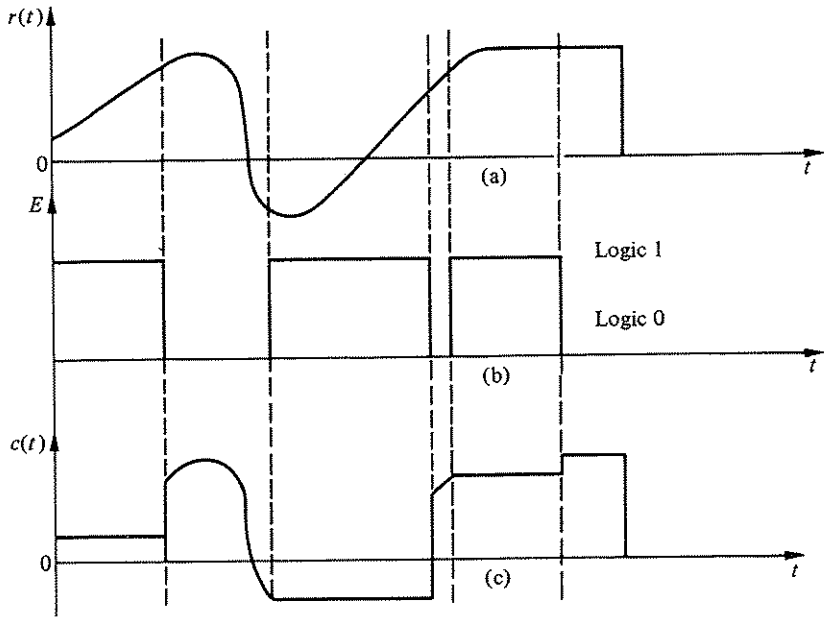


Figure 9.7-2. Track and store operation.

responds to ground level and logic 1 is given by a small positive potential such as 5 volts.

A very useful hybrid computing element is obtained by combining two track-hold units in cascade, as shown in Figure 9.7-3(a). The first unit is driven by logic signal E, while the second is driven by its complement $\bar{E}$. This element functions as an analog memory. Its usefulness can be further enhanced by feeding the output of the second track-hold into the input of the first, as shown by the diagram of Figure 9.7-3(b). In this connection it is an analog accumulator and is usually switched periodically. Thus it functions

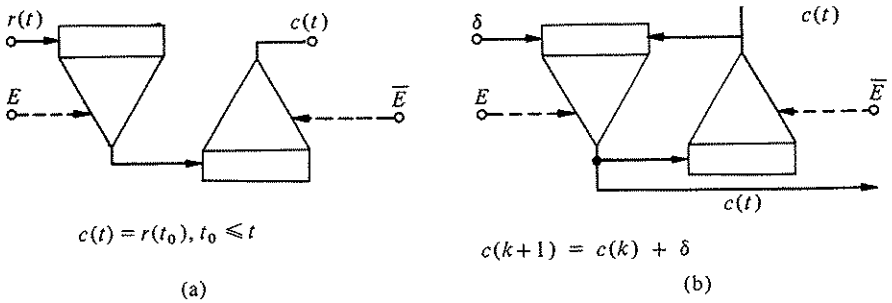


Figure 9.7-3. Cascaded track-store circuits: (a) analog memory; (b) analog accumulator.

according to the difference equation

$$c(k+1) = c(k) + \delta \tag{9.7-1}$$

This is recognized as a discrete integrator.

One other feature worth noting is the use of servo-set potentiometers. All potentiometers are set to four-place accuracy by a remotely controlled servomotor. This arrangement makes it possible to use the digital computer to adjust the potentiometers, greatly enhancing the automatic capabilities of the hybrid computer.

The Digital Computer

The digital computer that is used in a hybrid computer installation may range from a small, 4096-word computer to as large a computer as one may require. Typically, however, the computer used is characterized by the following specifications:

1. Memory size of 8192 with 16- to 18-bit words.
2. Ability to interface with a large number of external devices.
3. Multilevel priority interrupt system, internal and external.
4. Extensive hardware capabilities to permit extended arithmetic operations in floating point.
5. Secondary memory such as discs to permit resident storage of library routines and compilers.
6. Medium volume input/output, such as high-speed paper tape reader and punch.

One of the most important factors determining the usefulness of the digital computer in a hybrid installation is the availability of sophisticated software. Of absolute necessity is a FORTRAN compiler. Next in importance are subroutines that consist of (1) arithmetic and mathematical functions, (2) format routines for numeric conversion, and (3) input/output routines to provide communication and control for all peripherals. There should be software packages specifically created for hybrid operation to permit real-time capability, time and delay statements, and complete control of and communication with the interface between analog and digital computer. Further, there are program setup and checkout routines.

Computers that fall into this category are the IBM 1800, CDC 1700, PDP-9, EAI-640, SDS 920, and many others.

The digital computer in a hybrid installation may be used solely in conjunction with the analog computer. On the other hand, recent developments in digital computer technology permit the time-sharing of the digital computer for a variety of uses. Thus, it may be possible to time-share a larger digital computer with a hybrid installation, provided that hybrid computer uses are granted a "foreground" priority, while all other uses during hybrid com-

putation can be handled on a "background" priority. What this means is that the digital computer makes available all the time needed at the right moment to the hybrid installation; all other users must be satisfied with whatever time is left over from hybrid computation.

The main advantage of a time-shared hybrid computer installation would appear to be the availability of a larger digital computer at less cost. But it is questionable whether the priority requirements can be always satisfied.

A hybrid computer may be controlled by digital instructions or by analog patching. Briefly, these two modes of program execution control are described as follows.

Digital Control

The timing of all analog operations, such as HOLD, OPERATE, and RESET is under digital program control. Data transfer from the digital to the analog computer via digital-to-analog converters and from the analog to the digital computer via analog-to-digital converters is triggered through appropriate commands in the digital computer. The entire simulation is initiated and terminated by digital computer instructions. Under digital control the analog computer is completely slaved to the digital computer.

Analog Control

The timing for RESET, OPERATE, and HOLD modes is controlled through patchable logic circuits such as AND and OR gates, flip-flops, counters, delay elements, etc. Data transfer is triggered by counters or differential switches. The program simulating the discrete-time part of the system is executed periodically upon a receipt of a new data set from the analog-to-digital converter. Under analog control the digital computer is slaved to the analog computer.

Program Instructions for Digital Control

For the sake of general flexibility a hybrid computer is usually under digital control. For this purpose we define, then, the following program instructions. They represent only a partial list of all instructions needed to support sophisticated hybrid computations.

CALL HOLD
: This call puts all integrators into HOLD mode; at the same time, all A/D converter channels sample the output voltage of the amplifiers to which they have been patched.

CALL RESET
: This call puts all integrators into RESET mode.

CALL OPERATE
: This call puts all integrators into OPERATE mode for a preselected interval; at the end of this interval the integrators automatically assume a HOLD mode.

CALL POTSET (KPOT, VALUE)
: This instruction sets the potentiometer identified as KPOT to the value VALUE.

HWRITE(I)X
: This instruction transfers the contents of location X to the output of D/A converter channel I and holds it there.

HREAD(I)X
: This instruction transfers the output of A/D converter channel I into location X.

These instructions are sufficient to develop a simple but complete program. The preparation of this program consists of two parts, the analog computer diagram and the digital computer instructions. An example to be introduced shortly will demonstrate this.

The Interface

The interface provides the link between the two computers. As was mentioned earlier, this linkage must provide for communication and control. As the computers work with physically different signals, digital and analog signals, the primary function of the interface is to provide a number of high-speed analog-to-digital and digital-to-analog channels through which data flow between the computers takes place. The intercomputer information flow is accomplished through A/D converters and D/A converters that operate with an accuracy variable from 6 to 14 bits including the sign bit. Table 9.7-2 shows the relative conversion accuracy that is possible in terms of percentile figures.

Table 9.7-2 Conversion Accuracy

Bits		Powers of 2	Percentage Accuracy
	1	2	50%
	2	4	25
	3	8	12.5
	4	16	6.2
	5	32	3.1
	6	64	1.6
	7	128	.8
Normal range	8	256	.4
	9	512	.2
	10	1024	.1
	11	2048	.05
	12	4096	.02
	13	8192	.01

The analog computer is capable of an accuracy of approximately .01 percent. Thus to carry a conversion to 13 bits achieves maximum reasonable accuracy.

In addition to conversion accuracy, time required to perform the conversion is often of great importance. Typically, an A/D converter can handle 20,000 to 50,000 conversions per second of words ranging in length from 8 to 13 bits. Digital-to-analog converters operate at a speed of a few microseconds. Normally, A/D conversion of several analog signals is carried out by time-sharing a single A/D converter through the use of a multiplexer. This is shown schematically in Figure 9.7-4. The multiplexer selects the channel

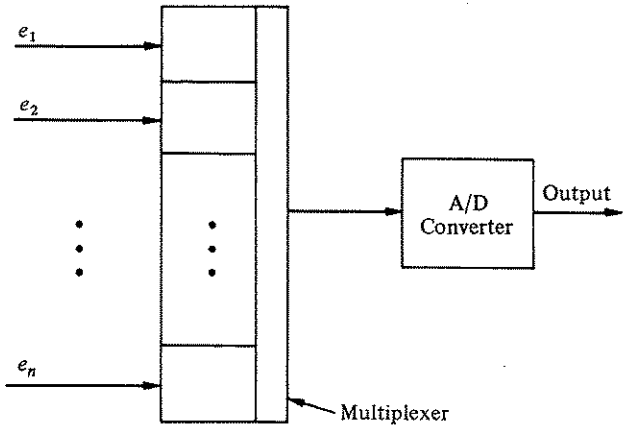


Figure 9.7-4. Time-sharing in A/D conversion.

whose input is to be converted. One problem that is encountered by the use of a multiplexer is that in the case of the conversion of a large number of analog signals, the consecutively generated digital numbers correspond to different moments of time because of the sequential operation. This time difference, called *time-skewing*, could be of significance in a given problem. One effective method of dealing with this problem is to provide sample and hold circuits for a selected number of analog signals and synchronize them. They may be still converted sequentially but may correspond to the same moment in time. The sample and hold amplifier precedes the multiplexer.

The computer linkage provides for intercomputer control for the purposes of timing, interrupt, input and output, and logic decisions during the course of a computer program.

The operation of the linkage can be under control of the analog computer or digital computer. Digital control is usually preferred because of greater flexibility and ease of programming.

Examples of Hybrid Computer Applications

Four examples will be presented here to illustrate the use of a hybrid computer in systems studies.

EXAMPLE 9.7-1

Design for Critical Damping. In the simple control system shown in Figure 9.7-5 the gain is to be automatically adjusted so that the system responds in a critically damped fashion to a step input. To accomplish this,

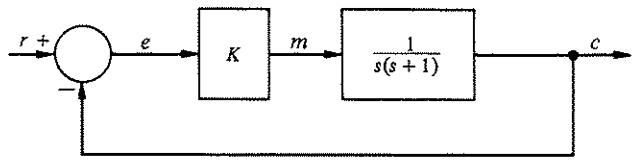


Figure 9.7-5. Gain-adjustable control system.

we shall use a hybrid computer, utilizing only the analog computer as controlled by patchable digital logic. The problem solution is actually fairly simple. The automatic gain adjustment is carried out by use of an electronic multiplexer, whose one input is e and whose other input is K, which is generated from analog accumulator as shown in Figure 9.7-6. It is automatically adjusted according to the iterative equation

$$K_{\text{new}} = K_{\text{old}} + \Delta K \tag{9.7-2}$$

at a frequency synchronized with the OPERATE-RESET cycle of the system simulated.

The parameter a is taken as a variable. A system that adjusts K each time a is changed could operate on the principle that if there is no overshoot during a particular COMPUTE cycle, one should increase the gain, and if there is overshoot, one should decrease the gain. Thus, the gain would tend to oscillate about a critically damped value. When a changes, the system will automatically adjust the gain to reach the new value, giving critical damping. A control circuit operating on this principle is shown in Figure 9.7-7.

At the beginning of each COMPUTE cycle, the storage device is cleared to indicate that no overshoot has been detected in this cycle. If the latching comparator detects overshoot at any time during the COMPUTE cycle, the storage device is set. The condition of the storage device at the end of the compute cycle determines whether the ΔK to be added during the RESET

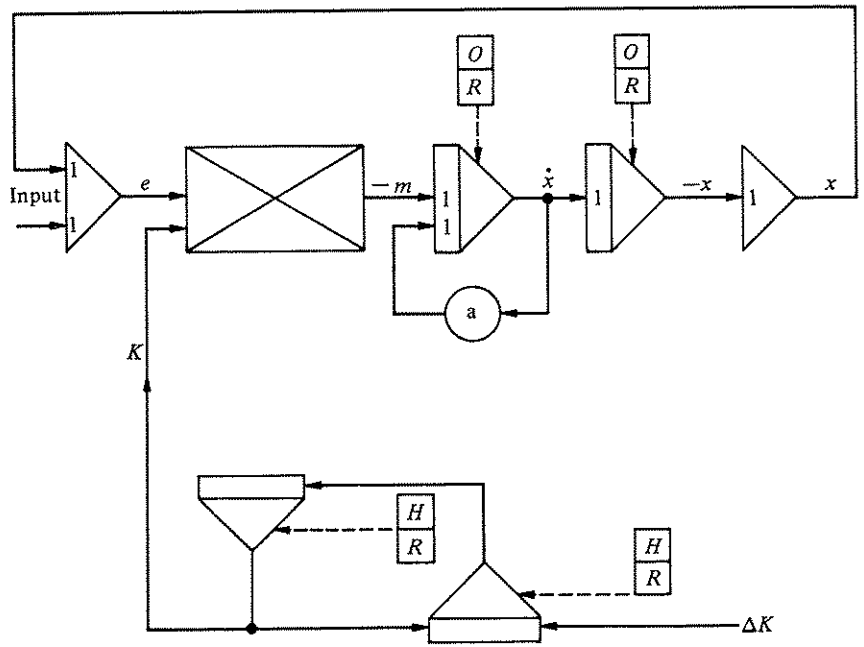


Figure 9.7-6. Circuit diagram.

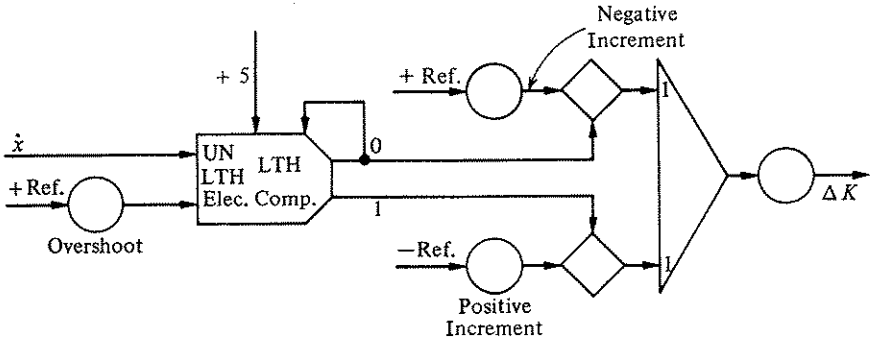


Figure 9.7-7. Control circuit using latching comparator and digital switches.

phase is positive or negative. Thus, the presence or absence of overshoot determines whether K is decreased or increased, respectively.

The OPERATE-RESET signals are timed by decade thumb wheel counters that may be dialed to give the timing for the integrators.

EXAMPLE 9.7-2

Simulation of a Sampled-data Control System. We consider the hybrid simulation of the computer control system shown in Figure 9.7-8. The system

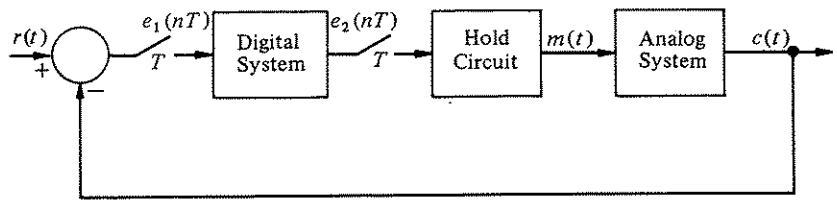


Figure 9.7-8. Digital control system.

is characterized as

Digital system:

$$e_2(nT) = 1.582e_1(nT) - .582e_1[(n-1)T] - .418e_2[(n-1)T] \quad (9.7\text{-}3)$$

Hold circuit:

$$m(nT+\tau) = e_2(nT) \quad \text{for } 0 \leq \tau < T \quad (9.7\text{-}4)$$

Analog system:

$$C(s) = \frac{1}{s(s+1)}M(s) \quad (9.7\text{-}5)$$

or

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} m(t) \quad (9.7\text{-}6)$$

$$c(t) = x_2$$

The analog parts of the system are simulated on the analog computer, requiring two integrators, two potentiometers, one D/A converter, and one A/D converter. Since the D/A converter incorporates a zero-order hold output, no special provision is made for this system element. The diagram for the analog part is shown in Figure 9.7-9.

The digital computer program is given next. The simulation is set up so as to allow digital control.

```
      DIMENSION
      READ (5,1) A,B,NDATA,TIMEFN
C     PARAMETERS FOR THIS RUN
   21 READ (5,2) GAIN,XIN
C     SET COEFFICIENTS
      CALL POTSET(1,GAIN)
      CALL POTSET(2,XIN)
C     INITIALIZE NEW RUN
      NRUN = NRUN+1
```

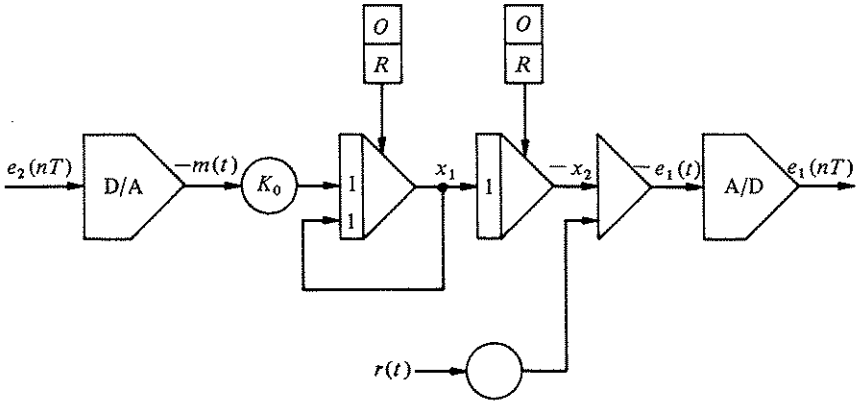


Figure 9.7-9. Analog computer diagram of hybrid simulation.

```
      TIME = 0.0
      DO 11 I = 1,3
      E1(I) = 0.0
      E2(I) = 0.0
   11 X(I) = 0.0
      CALL RESET
C     COMPUTE COMPUTER INPUT
   20 E1(2) = E1(1)
      HREAD(1,E1(1))
C     COMPUTE COMPUTER OUTPUT
      E2(2) = E2(1)
      E2(1) = 1.582*E1(1)-.582*E1(2)-.418E2(2)
      HWRITE(1,E2(1))
C     COMPUTE PLANT OUTPUT FOR NEW INTERVAL
C     NOTE: INTERVAL HAS BEEN SET TO VALUE OF SAMPLING
      PERIOD
      CALL OPERATE
C     PRINT OUTPUT
      TIME = TIME+T
      WRITE(6,3) TIME,E1(1),E2(1)
C     ALL PERTINENT ANALOG VARIABLES ARE RECORDED BY PEN
      RECORDERS
C     TEST IF THIS RUN IS COMPLETED
      IF(TIME.LT.TIMEFN) GO TO 20
C     TEST IF A NEW RUN IS TO BE INITIATED
      IF(NRUN. LT. NDATA) GO TO 21
      STOP
```

EXAMPLE 9.7-3

Design of Digital Compensator. As in our last illustration of the use of hybrid computers in systems studies, we consider the design of a digital

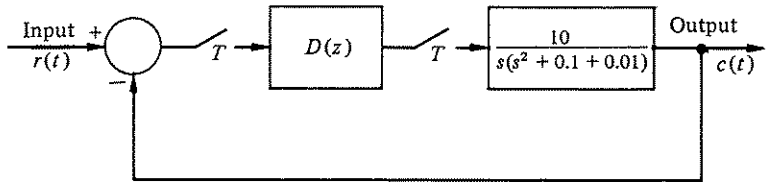


Figure 9.7-10. Digital control system.

compensator. It is required that the digital transfer $D(z)$ shown in Figure 9.7-10 be designed according to a weighted positive combination of the following:

1. Minimum overshoot.
2. Minimum rise time.
3. Zero steady-state error.
4. Minimum settling time.

All conditions are to be satisfied in response to a step input.

There exists no analytical design procedure that may be followed to realize these objectives. Nor is it at all established that the proposed system structure corresponds to the best configuration. Furthermore, the form of $D(z)$ is unknown. A design procedure that performs well under these adverse conditions is based upon a computer experimental approach as outlined below.

The form of $D(z)$ is arbitrarily taken as a second-order digital transfer function

$$D(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{9.7-7}$$

The system is simulated with a set of numerical values assumed for the coefficients of $D(z)$. A performance function is determined which measures the quality of the response relative to the design objectives. A good choice for this performance function might be

$$J = k_1(C_{\max} - R) + k_2(T_0) + k_3(C_\infty - R)^2 + k_4(T_{2\%}) \tag{9.7-8}$$

where $C_{\max}$ = output at time of maximum overshoot
R = magnitude of step input
T_0 = time at which output equals input for the first time
C_∞ = steady state output
$T_{2\%}$ = time at which output has settled to within 2% of input
k_i = weighting constants.

The four successive terms of the performance function (9.7-8) correspond to the four design objectives 1 through 4, respectively.

The general objective of the design procedure consists of adjusting the coefficients of the digital transfer function until the performance function reaches a minimum. The class of techniques that may be used for this purpose is generally referred to as *parametric optimization techniques*. During the past decade considerable research effort has been expended in the development of a great number of optimization techniques.* The use of these techniques has become feasible only recently because of the availability of high-speed digital computers.

The minimization technique used in this example is called *pattern search*.† This method begins by changing the parameters of $D(z)$ one at a time, starting at some arbitrary initial choice. The magnitude of the change is arbitrary, but is usually kept small. Associated with each parameter change is a complete simulation of the system and an evaluation of the performance function. Changing a parameter may result in an increase or decrease of J or, possibly, no change in J may occur. Should an increase be the result, the perturbation is repeated with a parameter change of opposite sign. When all parameters have been perturbed once or twice so that the performance function is reduced in each case, then all parameters are changed at the same time in the manner indicated by the individual changes. The process is then repeated. Thus it can be seen that the pattern search method alternates between perturbing the parameters individually to determine a "direction" or pattern and moving in that direction with all parameters. Depending on how well a new direction compares with a previous one, the successive pattern moves may increase or decrease the magnitude of the parameter adjustments. This procedure, therefore, tends to learn as it goes along and is generally quite effective in adapting to problem peculiarities.

In employing an optimization technique such as the one described above, the overall problem solution is under the control of the optimization program, which, of course, is programmed on the digital computer. The organization of the entire program is shown by the flow chart in Figure 9.7-11. It shows the simulation and the evaluation of the performance function as being separate subroutines of the overall program.

The simulation of the system is carried out by employing the digital computer for the evaluation of the recursion equation and the analog computer for the simulation of $G(s)$. The system simulation as indicated by the flow chart of Figure 9.7-12 requires the preparation of a FORTRAN program for the execution of $D(z)$ and control of simulation. The detailed instructions of the program are very similar to Example 9.7-2 and therefore are not shown here.

*P. E. Fleischer, "Optimization Techniques," pp. 175–216, *System Analysis by Digital Computer*, edited by F. F. Kuo and J. F. Kaiser, Wiley, 1966, New York.

†R. Hooke and T. A. Jeeves, "Direct Search Solutions of Numerical and Statistical Problems," *Journal Assoc. Comp, Mach.*, Vol. 8, April 1962, pp. 212–229.

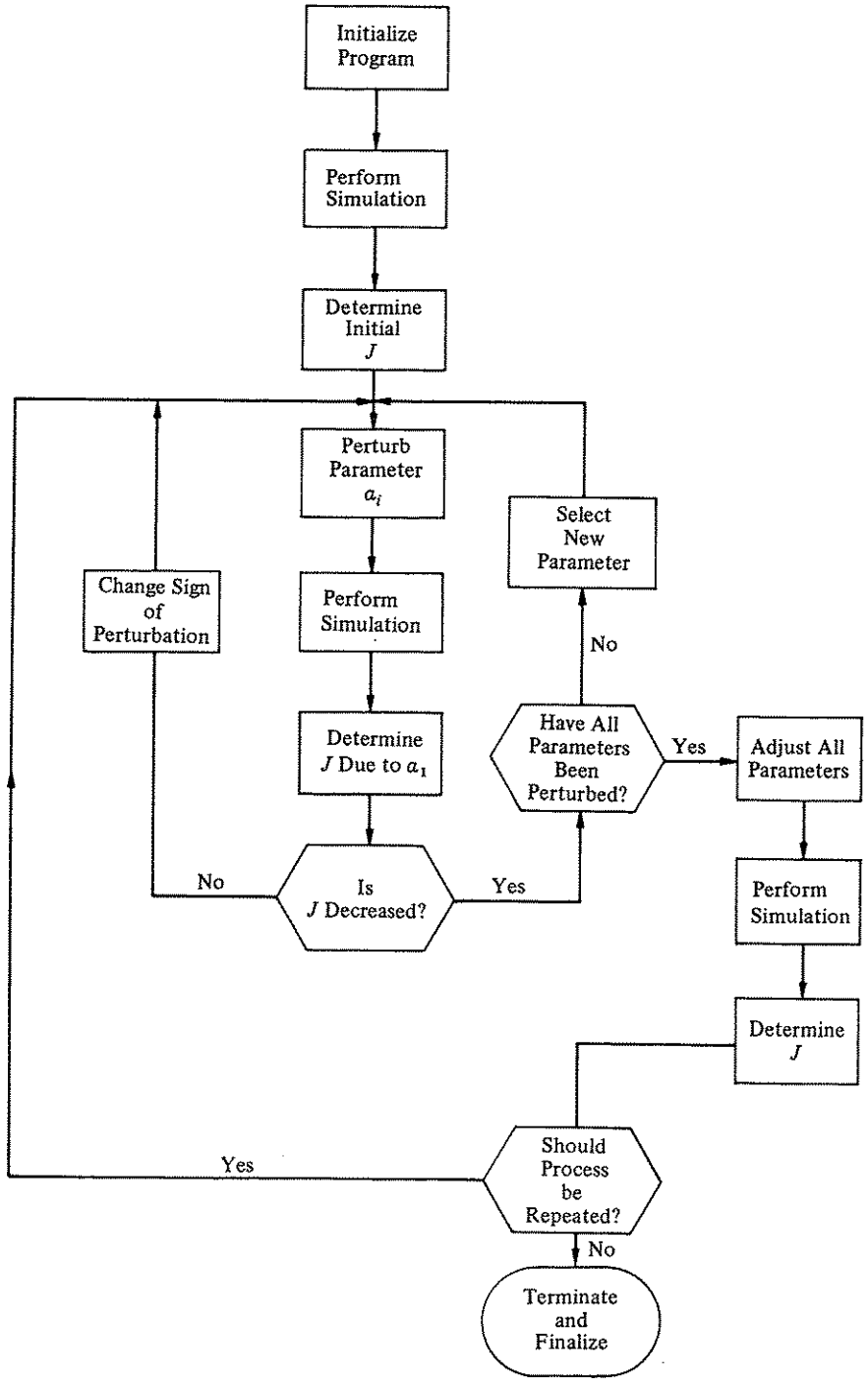


Figure 9.7-11. Flow chart of optimization program.

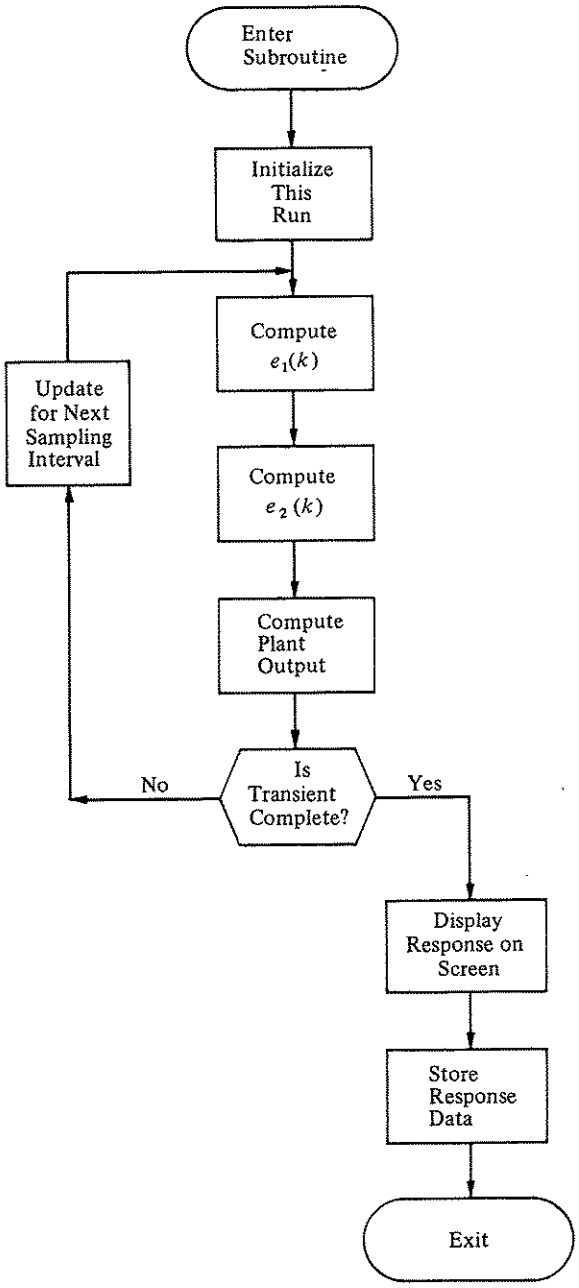


Figure 9.7-12. Flow chart of simulation.

It is reasonable to assume that all digital calculations during the simulation will not require more than 200 microseconds per sampling time. If the sampling time is .1 second and the analog computer operates at a speed of 100 times real time, then it requires 1 millisecond. Thus for each sampling interval 1.2 milliseconds are required. Suppose also that the transient requires 10 seconds or 100 sampling intervals; then the entire simulation requires 120 milliseconds. Thus each parameter perturbation uses up 120 milliseconds. If now eight simulations are required for each major cycle—seven parameter perturbations, one for each parameter plus two extra ones for wrong directions, and one all-parameter adjustment—one full second is used up. During a typical design of the digital compensator possibly from 50 to 100 complete iterations are required. We allow, furthermore, a negligible time of 1 millisecond for other digital computations for the performance functions and general bookkeeping. All factors considered, it is reasonable to expect a complete design every one to two minutes. This amazing speed, of course, is one of the attributes of hybrid computation.

EXAMPLE 9.7-4

A Sampled-data System with Minimum Settling Time. Design and test a digital controller that will permit the system shown in Figure 9.7-13 to respond to step inputs with zero steady-state error in a minimum number of sampling periods. The test will consist of a simulation on a hybrid computing system.

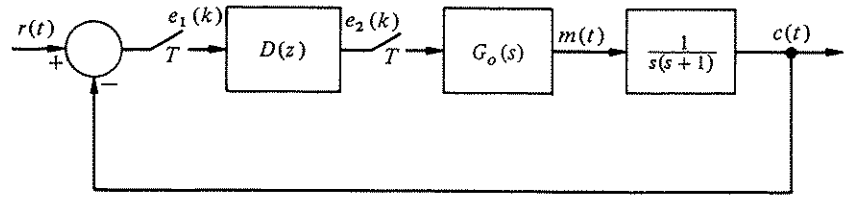


Figure 9.7-13. Computer control system.

The method of design for $D(z)$ is presented in Chapter 7. It is shown that the form of $D(z)$ is a ratio of polynomials of equal degree N. The degree N is equal to the order n of the transfer function of the plant if the plant does not contain a free integrator; $N = n - 1$ if the plant contains a free integrator.

The minimum settling system is simulated on a PDP-8/TR-20 hybrid computing system. The timing for the sampling intervals is under the control of an external timing clock. To carry out the multiplications required in the program in floating point, a number of system subroutines are utilized.

Solution. For the case at hand, the digital transfer function is derived to be

$$D(z) = \frac{F_1 + F_2 z^{-1}}{1 + F_3 z^{-1}} \tag{9.7-9}$$

where

$$F_1 = \frac{1}{T(1 - e^{-T})}, \qquad F_2 = -\frac{e^{-T}}{T(1 - e^{-T})}$$

$$F_3 = \frac{(1 - e^{-T}) - Te^{-T}}{T(1 - e^{-T})} = F_1 + (1 + T)F_2$$

For $T = .1$ sec	For $T = 2$ sec	For $T = 1$ sec
$F_1 = 105.2$	$F_1 = .519$	$F_1 = 1.58$
$F_2 = -95.3$	$F_2 = -.0699$	$F_2 = -.577$
$F_3 = .37$	$F_3 = .3093$	$F_3 = .418$

For $T = 1$, the linear recursion equation to be programmed is

$$e_2(k) = 1.58e_1(k) - .58e_1(k - 1) - .418e_2(k - 1) \tag{9.7-10}$$

In order to program the above recursion equation, it is advisable to scale the variables so that overflows and underflows do not occur.

To scale, first find the smallest power of 2 larger than or equal to the maximum value of the variables.

$$e_2(k) = F_1 e_1(k) + F_2 e_1(k - 1) - F_3 e_2(k - 1)$$

$$e_2'(k)2^{Q_4} = F_1' 2^{Q_1} e_1(k) + F_2' 2^{Q_2} e_1(k - 1) - F_3' 2^{Q_3} e_2(k - 1)2^{Q_4}$$

$$e_2'(k) = [F_1' e'(k)]2^{Q_1 - Q_4} - [F_3' e_2'(k - 1)]2^{Q_3} + [F_2' e_1(k - 1)]2^{Q_2 - Q_4}$$

Each of the terms in square brackets is now scaled. Multiplication by a power of 2 in a binary computer is handled by a shift to the left by as many times as the power (negative power). We select the scaled factors such that the scaled quantities fall within the range (.5, 1.0). Thus

$$|e_1(k)'| \leq 1$$

$$|e_2(k)| \leq 2^{Q_4}$$

$$|F_1| \leq 2^{Q_1}$$

$$|F_2| \leq 2^{Q_2}$$

$$|F_3| \leq 2^{Q_3}$$

so that

$$F_1 = F_1' 2^{Q_1}$$

$$F_2 = F_2' 2^{Q_2}$$

$$F_3 = F_3' 2^{Q_3}$$

$$e_2(k) = e_2'(k)2^{Q_4}$$

Using the above definition, we replace the unscaled variables in the equation.

For $T = 1$,

$$|F_1| = 1.58 \leq 2^1$$

$$|F_2| = |-.577| \leq 2^0 = 1$$

$$|F_3| = |.418| \leq 2^{-1}$$

$$|e_2(k)| \leq 2^1$$

$$Q_1 = 1 \qquad F_1' = .79$$

$$Q_2 = 0 \qquad F_2' = -.577$$

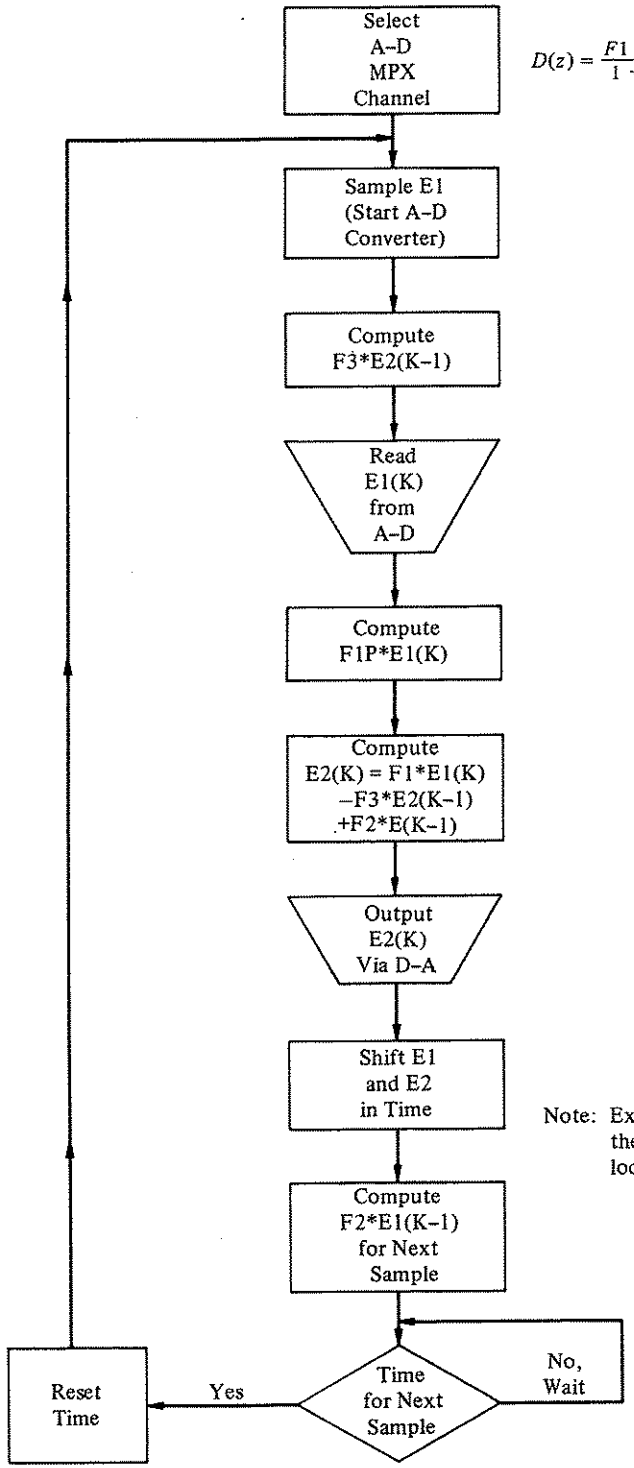
$$Q_3 = -1 \qquad F_3' = .836$$

$$Q_4 = 1 \qquad e_2'(k) = \frac{e_2(k)}{2}$$

The program is now ready to be implemented on a digital computer. A flow chart of the program is shown in Figure 9.7-14. Note that for $k = 1$,

```
F1*E1(K)  =  F3*E2(K-1)
```

Therefore, this subtraction occurs first.



Note: External clock increments the time stored in memory location 4.

Figure 9.7-14. Flow chart of program

The program is coded for execution on the PDP-8.* The language shown is the computer's symbolic assembler. The listing of this program is shown in Table 9.7-3. A block diagram of the complete simulation is shown in Figure 9.7-15, where the plant is simulated by utilizing analog computer elements.

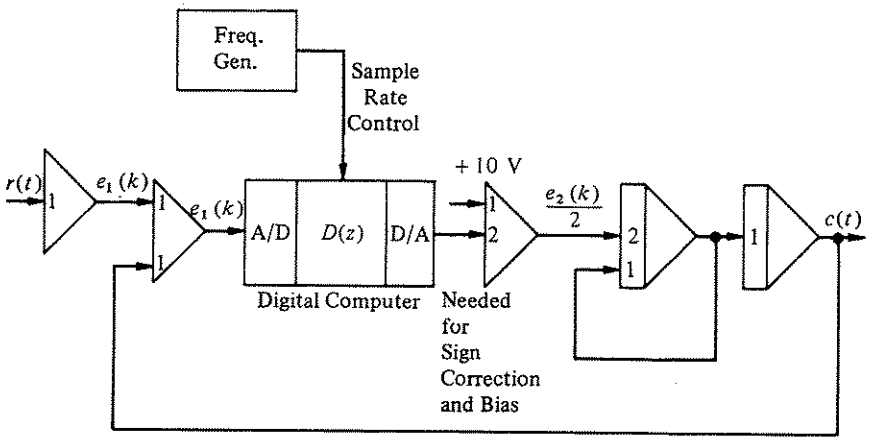


Figure 9.7-15. Block diagram of simulation.

Table 9.7-3 PDP-8 Listing of Program

```
*4
TIME,   -1              /STORE TIME
*1000
STRT,   CLA             /AC←0
        ADCC            /CHANNEL 0
GO,     ADCV            /CONVERT E1(K)
        TAD  E2K1       /E2(K-1)
        JMS  MULT       /F3P*E2(K-1)
F3P,    3260            /.418*2 = F3*2↑(-Q3)
        DCA  SAV3       /MOST SIG. PART
        TAD  MP1
        DCA  SAV3+1     /LEAST SIG. PART
        TAD  M3         /-M3 = NO. OF LEFT SHIFTS
        SMA             /IS IT A LEFT SHIFT?
        JMP  .+4        /NO, M3>OR = 0, SHIFT IN WRONG DIRECTION,
                        SKIP
        JMS  DPSL       /SHIFT PROD LEFT FOR SCALING
        SAV3
        DCA  SAV3
        CLA
        ADRB            /READ E1(K) FROM A-D CONVERTER
        DCA  E1K        /SAVE IT
```

*See Section 9.8.

Table 9.7-3 (cont.)

```
        TAD  E1K
        JMS  MULT       /F1P*E1(K)
F1P,    3121            /1.58*1/2 = F1*2↑(-Q1)
        DCA  SAV1
        TAD  MP1        /GET LEAST SIGNIFICANT PART
        DCA  SAV1+1
        TAD  M1
        SMA             /IS IT A LEFT SHIFT?
        JMP  .+4        /NO, SKIP
        JMS  DPSL       /SHIFT LEFT M1 PLACES
        SAV1
        DCA  SAV1       /SAVE RESULT
        CLA             /CLEAR AC IN CASE ABOVE COMMAND SKIPPED
        TAD  SAV3       /SAV3
        CIA             /-SAV3 = -F3P*E2(K-1)
        TAD  SAV1       /SAV1-SAV3 = F1P*E1(K)-F3P*E2(K-1)
        TAD  SAV2       /SAV1-SAV3+SAV2 = F1P*E1(K)-F3P*E2(K-1)
                         +F2P*E1(K-1)
        6551            /OUTPUT E2P(K) VIA D-A CONVERTER
                        /OUTPUT SHOULD BE MULTIPLIED BY 2↑Q4 TO
                        /CORRECT FOR SCALING IN DIGITAL COMPUTER
        TAD  E2PK       /SHIFT E2
        DCA  E2K1       /IN TIME
        TAD  E1K        /SHIFT E1
        DCA  E1K1       /IN TIME
        TAD  E1K1       /E1(K-1)
        JMS  MULT       /F2P*E1(K-1)
F2P,    -2235           /(-.577)*1 = F2*(2↑-Q2)
        DCA  SAV2       /MOST SIG. PART
        TAD  MP1
        DCA  SAV2+1     /LEAST SIG. PART
        TAD  M2         /-M2 = NUMBER OF RIGHT SHIFTS
        SMA             /IS IT A RIGHT SHIFT?
        JMP  .+4        /NO, SKIP
        JMS  DPSR       /SHIFT PROD. RIGHT FOR SCALING
        SAV2
        DCA  SAV2       /STORE RESULTS
        CLA             /AC←Ø IN CASE ABOVE COMMAND SKIPPED
        TAD  TIME       /AC←Ø+TIME
        SPA             /IS IT TIME FOR NEW SAMPLE?
        JMP  .-3        /NO. WAIT
        CLA  CMA        /YES, AC←-1
        DCA  TIME       /RESET TIME
        JMP  GO
E1K,    Ø               /E1(K)
E1K1,   Ø               /E1(K+1)
E2PK,   Ø               /E2'(K)
E2K1,   Ø               /E2'(K+1)
SAV1,   Ø
        Ø
```


Table 9.7-3 (cont.)

```
SAV2,   Ø
        Ø
SAV3,   Ø
        Ø
M3,     Ø               /-(Q3+1), Q3 IS LEFT SHIFTS
M2,     Ø               /-(Q4-Q2-1), Q2-Q4 IS LEFT SHIFTS
M1,     -1              /-(Q1-Q4+1), Q1-Q4 IS LEFT SHIFTS
                        /CONSTANT 1 ABOVE IS DUE TO MULTIPLY
                         SUBROUTINE
/SUBROUTINES MULT AND DPSL AND DPSR MUST BE ADDED
/
PAUSE
```

Hybrid computers, their development and application are just beginning to assume a significant role in the field of systems analysis and design. Examples 9.7-1 through 9.7-4 cited above show but a limited sample of the potential of hybrid computers.

9.8 Computer Control

Throughout the chapters of this text we have presented techniques for the analysis and design of systems containing a digital computer or digital processor as an integral element. It is reasonable to expect that the reader at this point has a fair appreciation of the underlying principles and engineering considerations of utilizing a digital computer in a system. But if some of this textbook knowledge were to be put to a real test, one which involves hardware and software design, chances are only slight that we would be fully equipped to master such a task. This section, then, is designed to help us improve our understanding of the design of a computer control system. We aim to accomplish this by examining three examples of computer control applications where reference is made to a specific computer and its features. This computer is a DEC PDP-8 computer. A brief description of the specific facility used in the examples is given here to present its salient characteristics.

The computer has 4096 12-bit words. Its memory operates on a cycle time of 1.5 microseconds. It has an external program interrupt and a bussed input-output system. Attached to it are an 11-bit plus sign A/D converter and a multiplexer. The A/D converter is capable of performing a 12-bit conversion in 35 microseconds. The analog voltage must be within the voltage range 0 to −10 volts; it must be scaled and shifted to accept a wider range. Also provided is a D/A converter of comparable accuracy with a conversion speed of 3.75 microseconds. All converters are equipped with

individual 12-bit registers for use as buffers. This enables the A/D converter to sample and hold an analog signal during conversion, while the D/A converter uses its buffers as zero-order-hold outputs.

The computer has been modified with the addition of a special input-output channel whereby, on a programmable basis, a binary word may be transferred between the accumulator and an external binary register in both directions. This I/O channel is useful in exchanging binary information directly with external devices for such purposes as reading a shaft encoder, sending a binary word to a logic circuit, supplying binary coded control instructions, etc.

The PDP-8 is programmable using an assembler language. Examples of programs will be illustrated by detailed flow charts. It can also be programmed in FORTRAN, although this is very limited because of the small memory.

The PDP-8 is a typical representative of a rapidly growing class of small-scale, almost desk-top size, digital computers, which may be used in a variety of on-line systems and data-processing applications.

The examples to be presented involve the PDP-8 as a control computer. The first case treates the computer as a passive discrete systems element which is to evaluate a linear recursion equation periodically. In this case it is used to provide digital compensation through integration. The second problem places a higher demand on the digital computer by using its computational and logic capabilities in the control of an attitude stabilization system. The third example consists of a digital control system using a digital PID (Position-Integral-Derivative) controller. The PID controller is designed by means of a simulation.

EXAMPLE 9.8-1

Digital Compensation through Integration. The system to be considered is shown in Figure 9.8-1. It consists of a single-loop feedback system. The

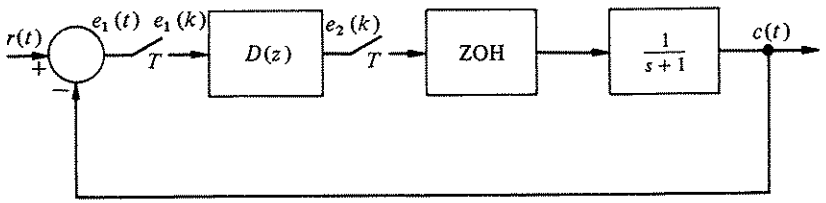


Figure 9.8-1. Digital control system.

digital computer is to control the plant through a zero-order hold such that the error is driven to zero for step input commands. This desired operating characteristic imposes the requirement that the system should be of type 1; i.e., one free integrator should be present in the loop. It is planned to let the

digital computer perform this operation. It could, therefore, be programmed according to the recursive relation

$$e_2(k) = e_2(k-1) + e_1(k) \tag{9.8-1}$$

We next check whether this recursive relationship will drive the steady-state error to zero. To this effect we apply the final value theorem of the z-transform.

The plant and the zero-order hold are characterized by the transfer functions

$$G_0G(s) = \frac{1 - e^{-Ts}}{s}\frac{1}{s+1} \tag{9.8-2}$$

while its z-transform is given by

$$\begin{aligned}\mathscr{Z}\{G_0G(s)\} &= (1-z^{-1})\mathscr{Z}\left\{\frac{1}{s(s+1)}\right\} \\ &= (1-z^{-1})\mathscr{Z}\left\{\frac{1}{s} - \frac{1}{s+1}\right\} \\ &= (1-z^{-1})\left[\frac{1}{1-z^{-1}} - \frac{1}{1-e^{-T}z^{-1}}\right] \\ &= \frac{z^{-1}(1-e^{-T})}{(1-z^{-1})(1-e^{-T}z^{-1})}\end{aligned} \tag{9.8-3}$$

The error is igven by

$$E_1(z) = \frac{R(z)}{1 + D(z)G_0G(z)}$$

Since $D(z)$ is given by

$$D(z) = \frac{1}{1-z^{-1}}$$

$$\begin{aligned}e_1(\infty) &= \lim_{z\to 1}(z-1)E_1(z) \\ &= \lim_{z\to 1}(z-1)\frac{1}{1 + \dfrac{z^{-1}(1-e^{-T})}{(1-z^{-1})(1-z^{-1}e^{-T})}}\frac{1}{1-z^{-1}}\end{aligned}$$

Evaluating the limit yields

$$e_1(\infty) = 0$$

Indeed, we can conclude that the system will behave like a type 1 system with

the digital integration. The characteristic equation is given by

$$1 + D(z)G_0G(z) = 0 \tag{9.8-4}$$

or, upon substitution and simplification,

$$z^2 - 2e^{-T}z + e^{-T} = 0 \tag{9.8-5}$$

It has the characteristic roots

$$z_{1,2} = e^{-T} \pm j\sqrt{e^{-T} - e^{-2T}} \tag{9.8-6}$$

It is interesting to show the derivation of the state transition equation for the closed-loop system for the benefit of comparison.

The transition equation for $G(s)$ is

$$x(k+1) = e^{-T}x(k) + (1 + e^{-T})e_2(k) \tag{9.8-7}$$

The digital computer equation is given by (9.8-1). To develop the transition equations we must convert the digital recursion equation from its present form into a state equation form. By the methods of Section 2.6 we have

$$\begin{aligned} d(k+1) &= d(k) + e_1(k) \\ e_2(k) &= e_1(k) + d(k) \end{aligned} \tag{9.8-8}$$

Combining (9.8-8) and (9.8-7) and using the relation

$$e_1(k) = r(k) - x(k) \tag{9.8-9}$$

we obtain the closed-loop transition equations

$$\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} 2e^{-T} - 1 & 1 - e^{-T} \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} 1 - e^{-T} \\ 1 \end{bmatrix} r(k) \tag{9.8-10}$$

The characteristic roots of these equations are given by

$$\begin{vmatrix} \lambda - 2e^{-T} + 1 & e^{-T} - 1 \\ 1 & \lambda - 1 \end{vmatrix} = 0$$

or

$$\lambda^2 - 2e^{-T}\lambda + e^{-T} = 0$$

which is identical to (9.8-5).

The investigation of the stability of the system as a function of the sampling time T reveals an interesting fact. Figure 9.8-2 shows a root-locus plot

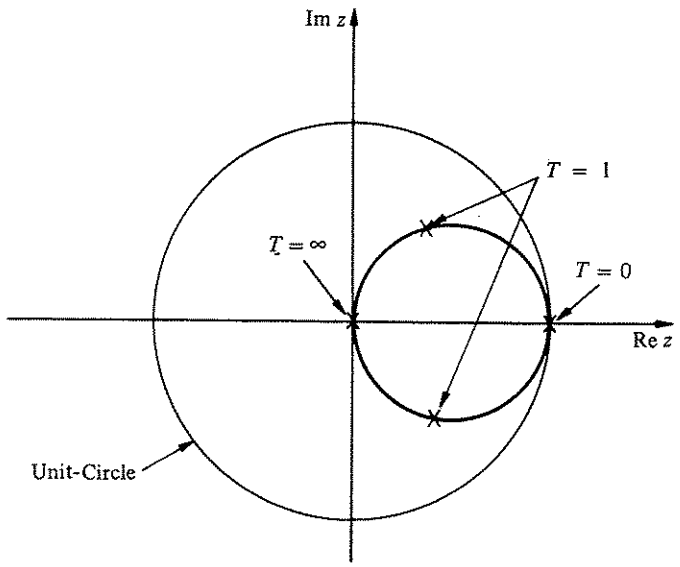


Figure 9.8-2. Root-locus with T as parameter.

of the system with T as a parameter. This is determined by calculating the roots of (9.8-5). This plot shows the location of the two roots to be starting at the origin for $T = \infty$ and migrating to the point (1, 0) for $T = 0$ on two semicircles. This is an interesting result, for it indicates that the response of the system becomes more and more oscillatory as T decreases. This result may be surprising. However, a closer look at the digital integrator quickly reveals that the gain of the system increases in direct proportion to a decrease in the sampling rate. To maintain the same gain, the digital transfer function should be modified to

$$D(z) = \frac{Tz}{z - 1} \tag{9.8-11}$$

The analytical aspects of this problem have now been investigated. We next carry out the design of the digital computer program that will implement the recursion equation (9.8-1). A flow chart for this program is presented in Figure 9.8-3. It is sufficiently detailed to show the necessary machine language instructions.

The program begins with an instruction to clear the accumulator and load the multiplex channel number into the accumulator. This may be any one of channels 1 through 24. The next instruction starts the conversion of the analog signal connected to the selected channel to a digital number. The analog number is sampled and held. The conversion requires 35 microseconds when carried to 12-bit accuracy, during which time the computer may carry out

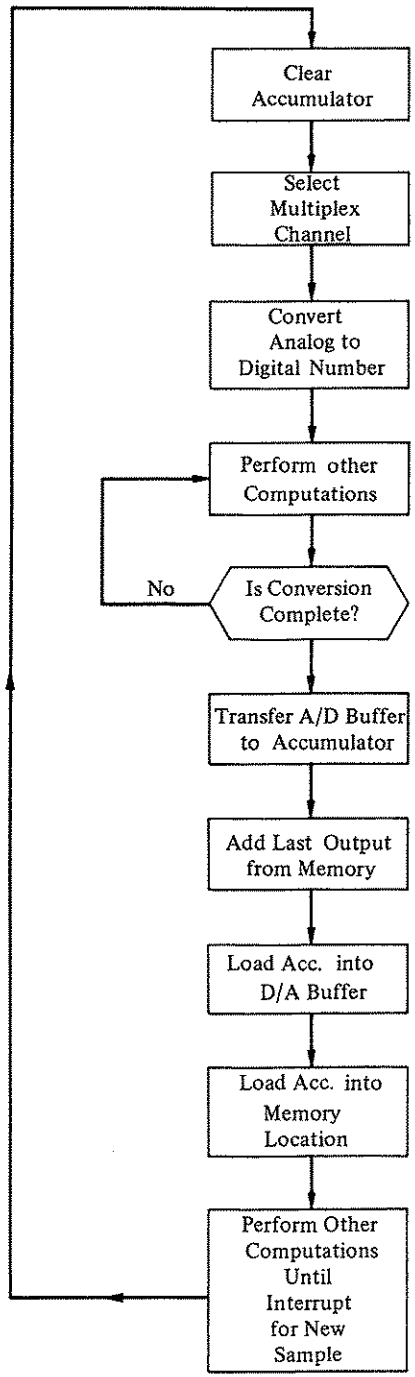


Figure 9.8-3. Flow chart for $D(z)$ program.

other computations independent of the conversion. When the conversion is complete, the contents of the A/D buffer are transferred to the accumulator, representing $e_1(k)$, the new input. To this is added the old output $e_2(k-1)$, resulting in the new output $e_2(k)$, which is stored in memory and also loaded into the D/A converter. Loading the D/A buffer requires 3.75 microseconds, while the D/A output network settles within 3 microseconds. Thus the digital-to-analog conversion is complete as soon as the load instruction is executed. The computer may now perform other functions until an interrupt occurs from an external clock to indicate the need for a new sample and a repetition of the entire cycle. The external clock gives this interrupt command at time intervals that must be larger than the total execution time of the cycle, which is 67 microseconds.

A diagram for the entire system simulation is shown in Figure 9.8-4. An analog computer was used to carry out the simulation of the continuous parts

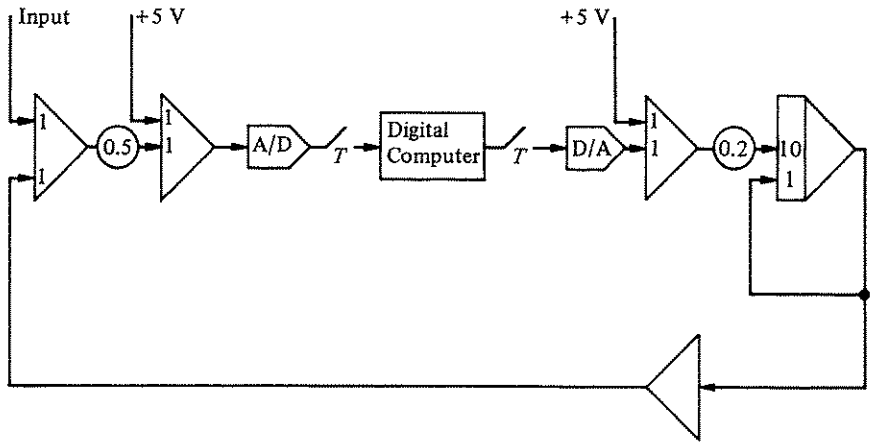


Figure 9.8-4. Complete system simulation.

of the system. Two operational amplifiers are used to perform scaling and shifting of the analog signals before and after the converters. This is needed to change the converter input and output levels (0 to -10 volts) to the analog computer voltage (± 10 volts).

EXAMPLE 9.8-2

Computer Control of Spacecraft Attitude. The second illustration of a computer control system deals with the utilization of the PDP-8 computer (or a computer like it) in the control of the attitude of a spacecraft. For the sake of simplicity we restrict the control requirements to one dimension only; thus the proposed problem can be described schematically as shown in Figure 9.8-5. The principal function of the computer is the determination of the prop-

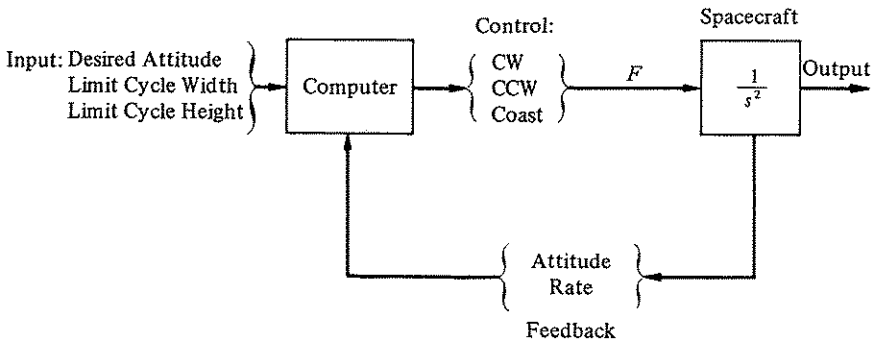


Figure 9.8-5. Attitude control by computer.

er cycling and timing of the control commands, which consist of (1) clockwise ($F = +1$), for which the thrusters are turned on to produce a clockwise moment; (2) counterclockwise ($F = -1$), for which the thrusters are turned on to produce a counterclockwise moment, (3) coast ($F = 0$), for which no thrusters are engaged and the vehicle is rotating at constant angular velocity.

The CW thrusters and the CCW thrusters are controlled by two separate signals which are called FP1 and FM1, respectively, in the computer program. Thus, the control commands are issued by the computer according to the following schedule:

F	FP1	FM1
+1	1	0
−1	0	1
0	0	0

where FP1 and FM1 are names of program variables.

For the determination of its control outputs the computer is provided with input information and feedback information. Since the space vehicle's attitude is controlled by maintaining a limit cycle, the input information consists of the desired attitude angle and rate, the width of the limit cycle, and the height of the limit cycle. These quantities are further defined by the schematic of Figure 9.8-6. A_1 and A_2 define desired attitude angle and attitude rate, respectively. The dimensions of the limit cycle are given by B_1 and B_2. The required feedback information is provided by two sensors measuring angular position and angular rate.

9.8-1 Design of Computer Program

The design of a computer control system requires the development of a computer program capable of satisfactorily managing all demands placed

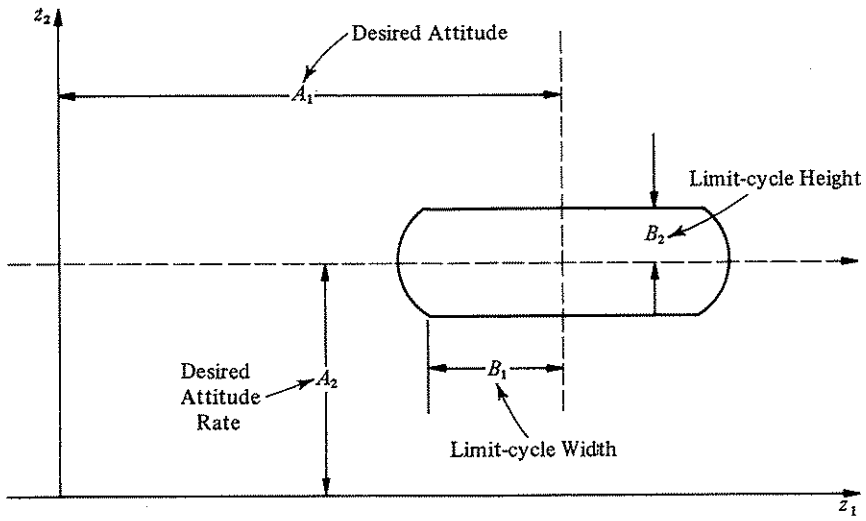


Figure 9.8-6. Limit-cycle operation specifications.

before it. The development of a "bug-free" program is absolutely critical. It is helpful, therefore, if this task can be approached in a methodical manner, although no single programming problem has a unique solution.

In the design of the computer program for the attitude control system two approaches will be demonstrated. The first approach is based upon the technique of flow-charting while the second approach utilizes logic circuit theory. Both programs, of course, yield the same result.

We first consider some relations that are common to both approaches. In considering the requirements of the control problem, it can be established that the choice of the control commands is entirely dependent on: (1) the state of the plant relative to the desired state, and (2) the dimensions of the limit cycle. The two-dimensional state space can be divided into nine regions, as shown in Figure 9.8-7. The dividing lines have the equations

$$x_1 \triangleq z_2 - A_2 - B_2 = 0 \tag{9.8-12}$$

$$x_2 \triangleq z_2 - A_2 + B_2 = 0 \tag{9.8-13}$$

$$x_3 \triangleq z_1 - A_1 + B_1 = 0 \tag{9.8-14}$$

$$x_4 \triangleq z_1 - A_1 - B_1 = 0 \tag{9.8-15}$$

For each of the nine regions a unique choice of control exists, as indicated in Figure 9.8-7. Thus the computer programs must be capable of determining the state of the space vehicle relative to these regions, and the choice of the control command is automatically decided. The structure of the overall program is indicated by the flow chart shown in Figure 9.8-8 It consists of the

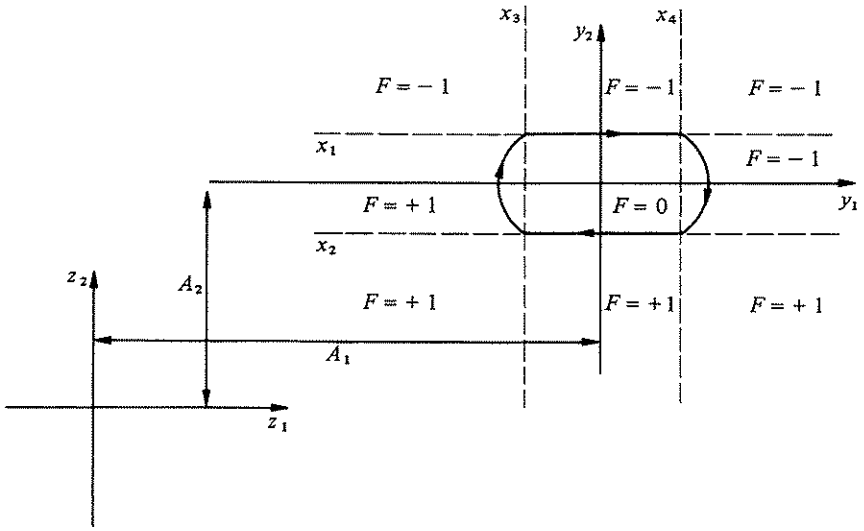


Figure 9.8-7. Regions of control.

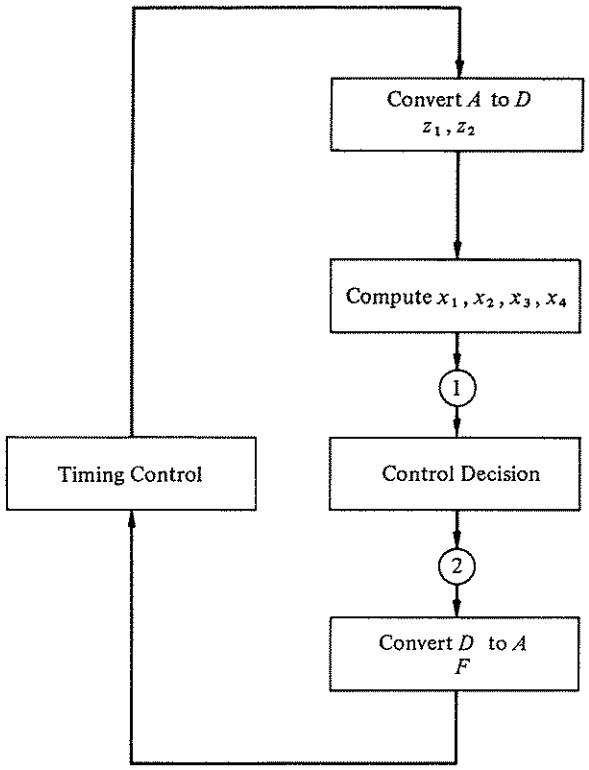


Figure 9.8-8. Gross detail of overall program.

analog-to-digital conversion of angular position and rate, the computation of equations (9.8-12) through (9.8-15), the decision of the proper control commands, and their subsequent digital-to-analog conversion. This sequence of operations is repeated cyclically subject to a timing control. The two versions of determining the control decision will now be presented.

9.8-2 Version I of Control Decision

This version is based upon testing equations (9.8-12) through (9.8-15) as algebraic inequalities. This is carried out by the decision indicated by the flow chart of Figure 9.8-9. A simple program based upon this flow chart is shown toward the end of this example.

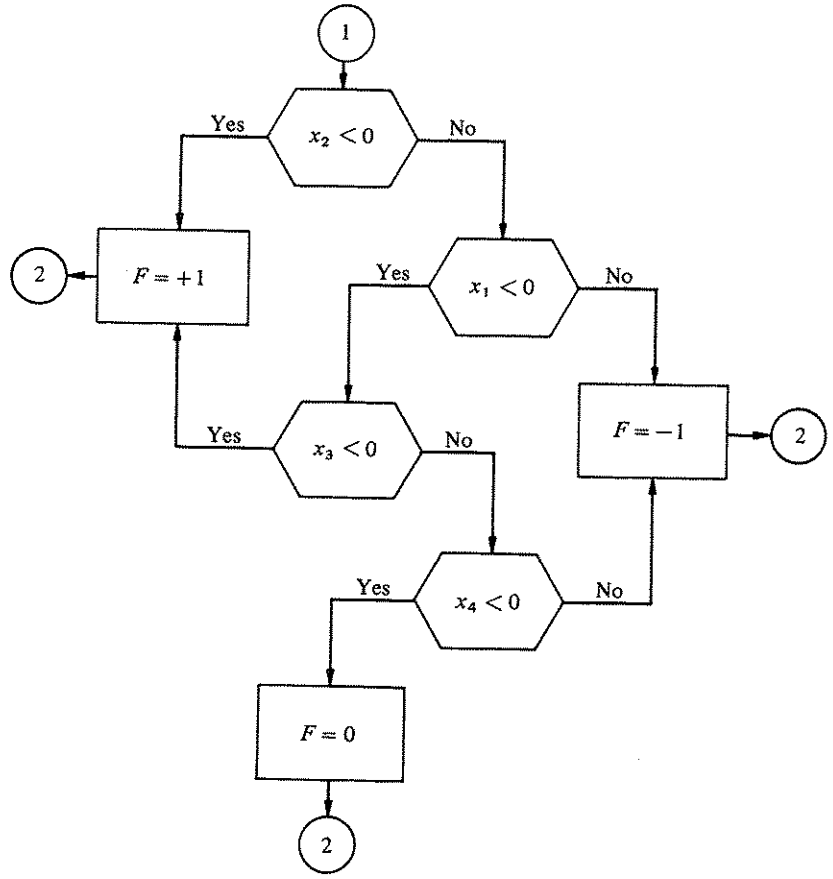


Figure 9.8-9. Flow chart for control command decision.

9.8-3 Version II of Control Decision

An alternate way of implementing the control decision process is by way of combinational logic. If we assign the logic values to the nine regions according to the scheme indicated by Figure 9.8-10, we can set up the table of combinations as given by Table 9.8-1. From this table of combinations,

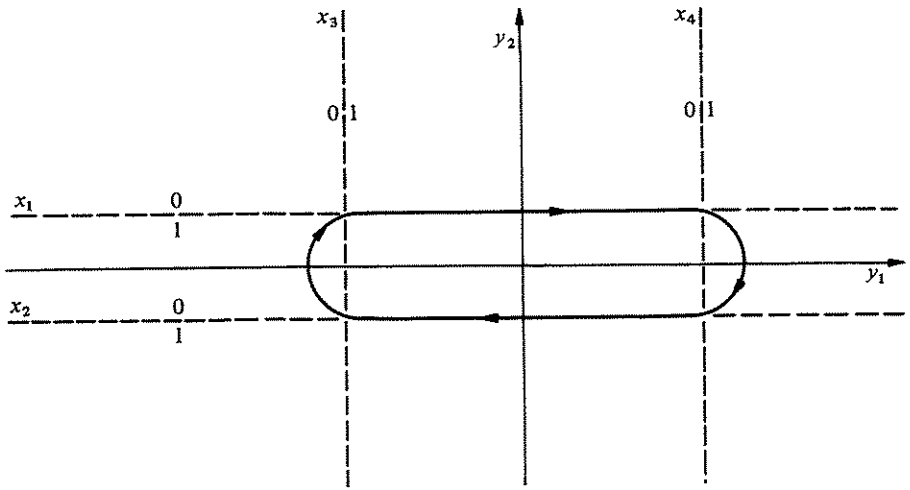


Figure 9.8-10. Logic regions for control decision.

Table 9.8-1 Logic Combinations for Regions of Figure 9.8-10

x_1	x_2	x_3	x_4	$F=+1$	$F=-1$
0*	0	0	0	0	1
0	0	0	1	x†	x
0	0	1	0	0	1
0	0	1	1	0	1
1	0	0	0	1	0
1	0	0	1	x	x
1	0	1	0	0	0
1	0	1	1	0	1
0	1	0	0	x	x
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	x	x
1	1	0	0	1	0
1	1	0	1	x	x
1	1	1	0	1	0
1	1	1	1	1	0

*For assignment of truth values see Figure 9.8-10.
†The x-entry denotes "Don't Cares."

or truth table, it is possible to extract logic equations by the use of a Karnaugh map given by Table 9.8-2. Each entry in this map consists of two parts, corresponding to $F=+1$ and $F=-1$. This table yields the two logic equations

$$\text{FP1} = x_2 + x_1 \cdot x_3' \tag{9.8-16}$$

$$\text{FM1} = x_1' + x_2' \cdot x_4 \tag{9.8-17}$$

Table 9.8-2 Karnaugh Table

x_1x_2 \ x_3x_4	00	01	11	10
00	01	xx	01	01
01	xx	xx	xx	xx
11	10	xx	10	10
10	10	xx	01	00

where the prime denotes complement, the plus sign denotes logic OR, and the dot sign denotes logic AND.

These logic equations may be programmed by utilizing the logic instructions of the computer. A flow chart showing this program is shown in Figure 9.8-11. It shows the same entry points as Version I and may be used interchangeably with Version I.

The entire program may now be assembled in detail. It is shown in terms of symbolic machine language statements in Table 9.8-3.

The digital computer communicates with the outside via data inputs and converters in the following way:

Data Input: A1, A2, B1, B2 may be entered into the computer either through keyboard switches, A/D converters, or direct memory access transfers.

A/D Converter: Z1 and Z2, corresponding to attitude and attitude rate, respectively, are entered through channels 0 and 1.

D/A Converter: FP1 and FM1, corresponding to $F=+1$ and $F=-1$ commands, respectively, are exited through channels 1 and 2. Logic signals FP1 and FM1 activate CW thrusters and CCW thrusters, respectively.

Figure 9.8-12 shows a series of phase-space plots for a variety of conditions. Figure 9.8-12(a) shows the limit cycle as approached from different initial conditions. Part (b) shows a response to a change in attitude reference, from $A_1 = 0$ to $A_1 = 30°$. Part (c) shows various limit cycles with B_2 (coasting velocity) as parameter, while part (d) shows the same with B_1 (dead band) as parameter.

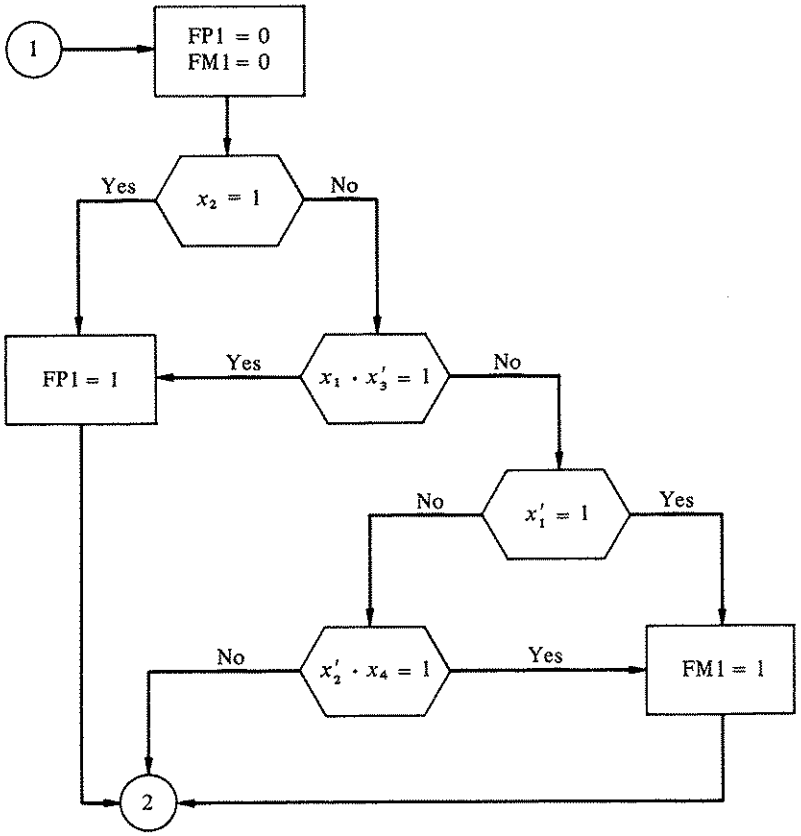


Figure 9.8-11. Flow chart for Version II program.

Table 9.8-3

Symbolic Machine Instruction	Commands
A1, 0 A2, 0 B1, 0 B2, 0 Z1, 0 Z2, 0 X1, 0 X2, 0 X3, 0 X4, 0	Assign storage locations to variable and set initially to zero.
BEGN, 6541	Select Multiplexer Channel 0, assign transfer location BEGN

Table 9.8-3 (cont.)

Symbolic Machine Instruction	Commands
6532	Convert attitude (Z1)
TAD A2	Add A2 to accumulator
TAD B2	Add B2 to acc.
CIA	Change sign of acc.
TAD Z2	Add Z2
DCA X1	Store in X1 and clear acc.
TAD A2	Add A2 to acc.
CIA	Change sign of acc.
TAD B2	Add B2
TAD Z2	Add Z2
DCA X2	Store in X2 and clear acc.
6531	Is conversion done?
JMP .−1	Yes: skip to next instruction No: return to previous instruction
6534	Read A/D buffer into acc. (Z1)
DCA Z1	Store in Z1 and clear accumulator
6544	Select MX Channel 1
6532	Convert attitude rate (Z2)
TAD A1	Add A1 to acc.
CIA	Change sign of acc.
TAD B1	Add B1 to acc.
TAD Z1	Add Z1 to acc.
DCA X3	Store in X3 and clear acc.
TAD B1	Add B1 to acc.
TAD A2	Add A2 to acc.
CIA	Change sign of acc.
TAD Z1	Add Z1 to acc.
DCA X4	Store in X4 and clear acc.
DCA FP1	Set FP1 = 0
DCA FM1	Set FM1 = 0
	VERSION I
TAD X2	Add X2 to acc.
SPA	If acc. is positive, skip next instruction
JMP FP2	If acc. is negative, transfer to FP2
CLA	Clear acc.
TAD X1	Add X1 to acc.
SMA	If acc. is negative, skip next instruction
JMP FM2	If acc. is positive, transfer to FM2
CLA	Clear acc.
TAD X3	Add X3 to acc.
SPA	If acc. is positive, skip next instruction
JMP FP2	If acc. is negative, transfer to FP2
CLA	Clear acc.
TAD X4	Add X4 to acc.
SMA	If acc. is negative, skip next instruction
JMP DTOA	If acc. is positive, transfer to DTOA
JMP FM2	Transfer to FM2

Table 9.8-3 (cont.)

Symbolic Machine Instruction	Commands
	VERSION II
TAD X2	Add X2 to acc.
SPA	Check truth value of X2: X2 = 1 if negative; X2 = 0 if positive; skip if X2 = 0
JMP FP2	If X2 = 1, transfer to FP2
CLA	Clear acc.
TAD X3	Add X3 acc.
CMA	Complement acc. to generate X3′
AND X1	Logic AND with X1 to generate X1X3′
SMA	Check truth value of X1X3′: X1X3′ = 0 if negative; skip if X1X3′ = 0
JMP FP2	if X1X3′ = 1 transfer to FP2
CLA	Clear acc.
TAD X1	Add X1 to acc.
CMA	Complement acc. to generate X1′
SPA	Check truth value of X1′: X1′ = 1 if positive; skip if X1′ = 1
JMP FM2	If X1′ = 0 transfer to FM2
CLA	Clear acc.
TAD X2	Add X2 to acc.
CMA	Complement acc. to generate X2′
AND X4	Logic AND with X4 to generate X2′X4
SPA	Check truth value of X2′X4: X2′X4 = 1 if positive; skip if X2′X4 = 1
JMP DTOA	If X2′X4 = 0 transfer to DTOA
FM2, CLA	This is instruction FM2; clear acc.
TAD REF	Add RFE to acc.
DCA FM1	Store REF in FM1: set $F = -1$
JMP DTOA	Transfer to DTOA
FP2, CLA	This is instruction FP2, clear acc.
TAD REF	Add REF to acc.
DCA FP1	Store REF in FP1: set $F = +1$
DTOA, CLA	This is instruction DTOA; clear acc.
TAD FP1	Load FP1 into acc.
6551	Convert DTOA-channel 1
CLA	Clear acc.
TAD FM1	Load FM1 into acc.
6552	Convert DTOA-channel 2
CLA	Clear acc.
6534	Read A/D buffer into acc. (Z2)
DCA Z2	Store in Z2 and clear acc.
TAD TIME	Load time into acc.
SPA	
JMP BEGN	If TIME < 0 begin new cycle
JMP .−3	If TIME > 0 wait until TIME < 0
FP1, 0	Reserve storage for $F = +1$
FM1, 0	Reserve storage for $F = -1$
REF, 4000	Reference set at decimal +0

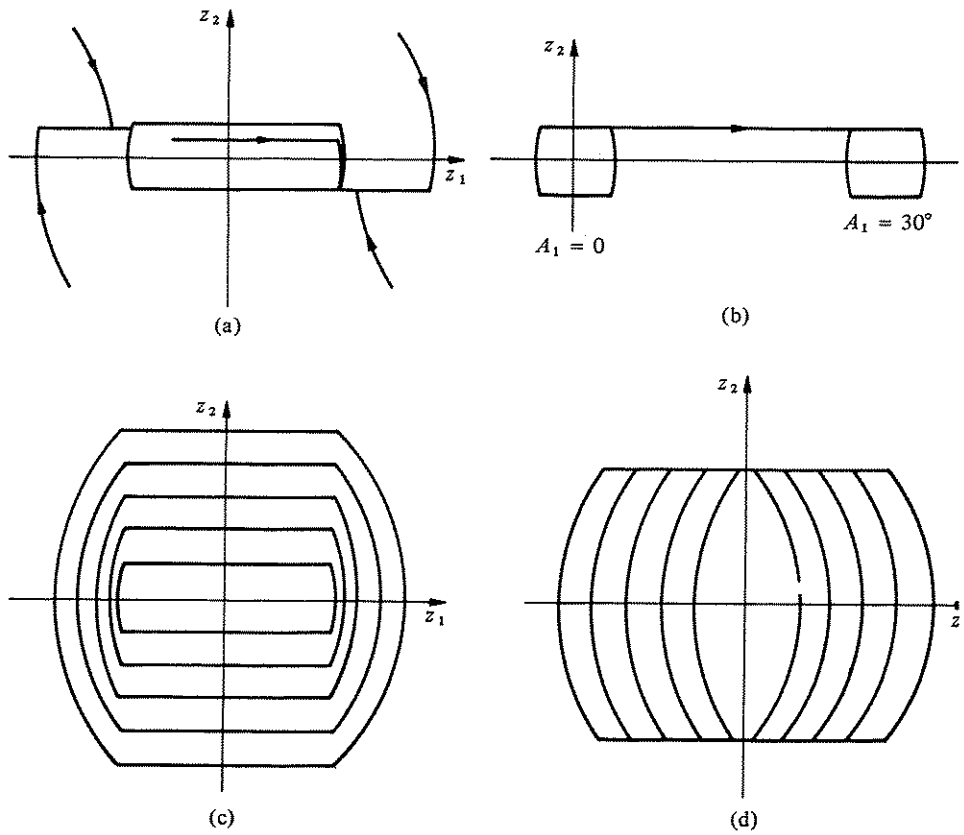


Figure 9.8-12. Phase-space plots of attitude control system.

EXAMPLE 9.8-3

*A Digital PID Controller.** Perform a simulation and determine a suitable design for a PID controller that is being used in the system shown in Figure 9.8-13. The PID controller is to be replaced by a digital version, as shown in Figure 9.8-14. The simulation should be sufficiently flexible to permit the adjustment of the following parameters: T, the sampling interval; K_p, the proportional gain; K_I, the integral gain; and K_d, the derivative gain. The system to be controlled is defined by its transfer function in Figure 9.8-14.

The digital computer is programmed so that the output sequence is given by

$$e_2(k) = K_p e_{21}(k) + K_I e_{22}(k) + K_d e_{23}(k) \tag{9.8-18}$$

*See also Section 3.7.

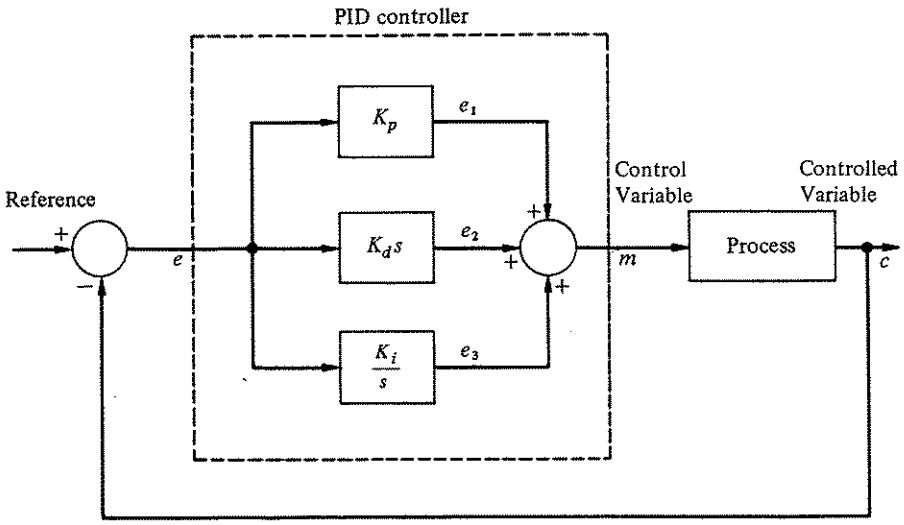


Figure 9.8-13. PID controlled system.

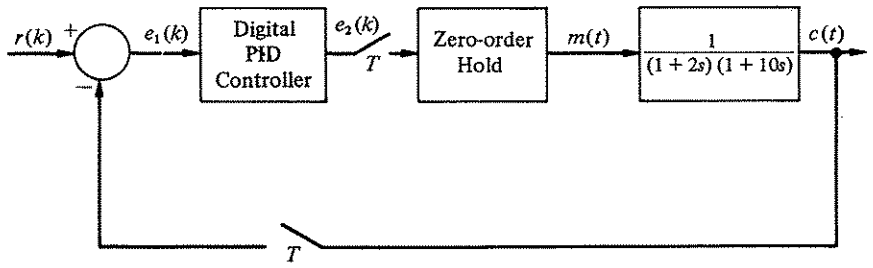


Figure 9.8-14. System with digital PID controller.

The three components of the sum are generated as follows:

Proportional control:

$$e_{21}(k) = e_1(k) \tag{9.8-19}$$

Integral control:

$$e_{22}(k) = e_{22}(k-1) + Te_2(k) \tag{9.8-20}$$

Derivative control:

$$e_{23}(k) = \frac{1}{T}[e_1(k) - e_1(k-1)] \tag{9.8-21}$$

In the application of a digital computer as a PID controller, the quanti-

ties K_p, K_i, K_d, and T must be determined. It has been the practice in the process control industry to adjust the gain constants on the job until a desirable response is obtained. Alternatively, a simulation of the entire system may be performed which permits parameter adjustment. This approach is followed in this study.

A PDP-8/TR-20 simulation is chosen as the method of simulation. The process to be controlled is programmed on an analog computer according to the diagram of Figure 9.8-15. A flow chart of the program is shown in Figure

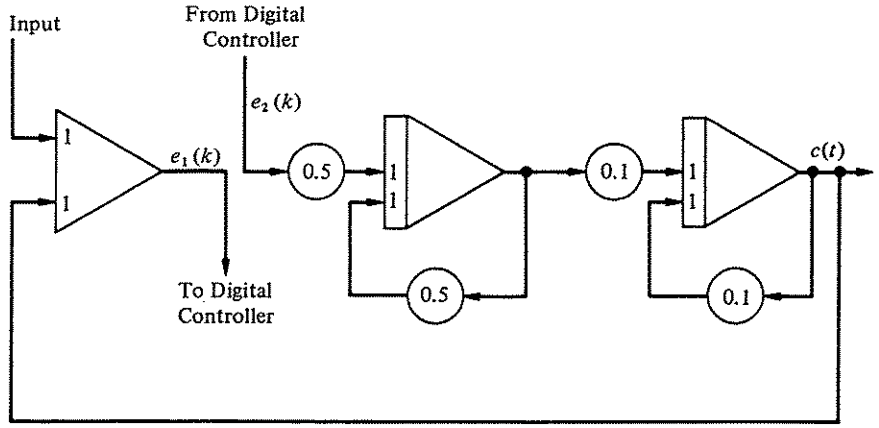


Figure 9.8-15. Simulation of process.

9.8-16. The program is listed in Table 9.8-4. When the program is started, the teletype will ask for the typewriter input of T, KP, KI, and KD in floating point format. In addition to setting these constants, one must set the digital clock to agree with T.

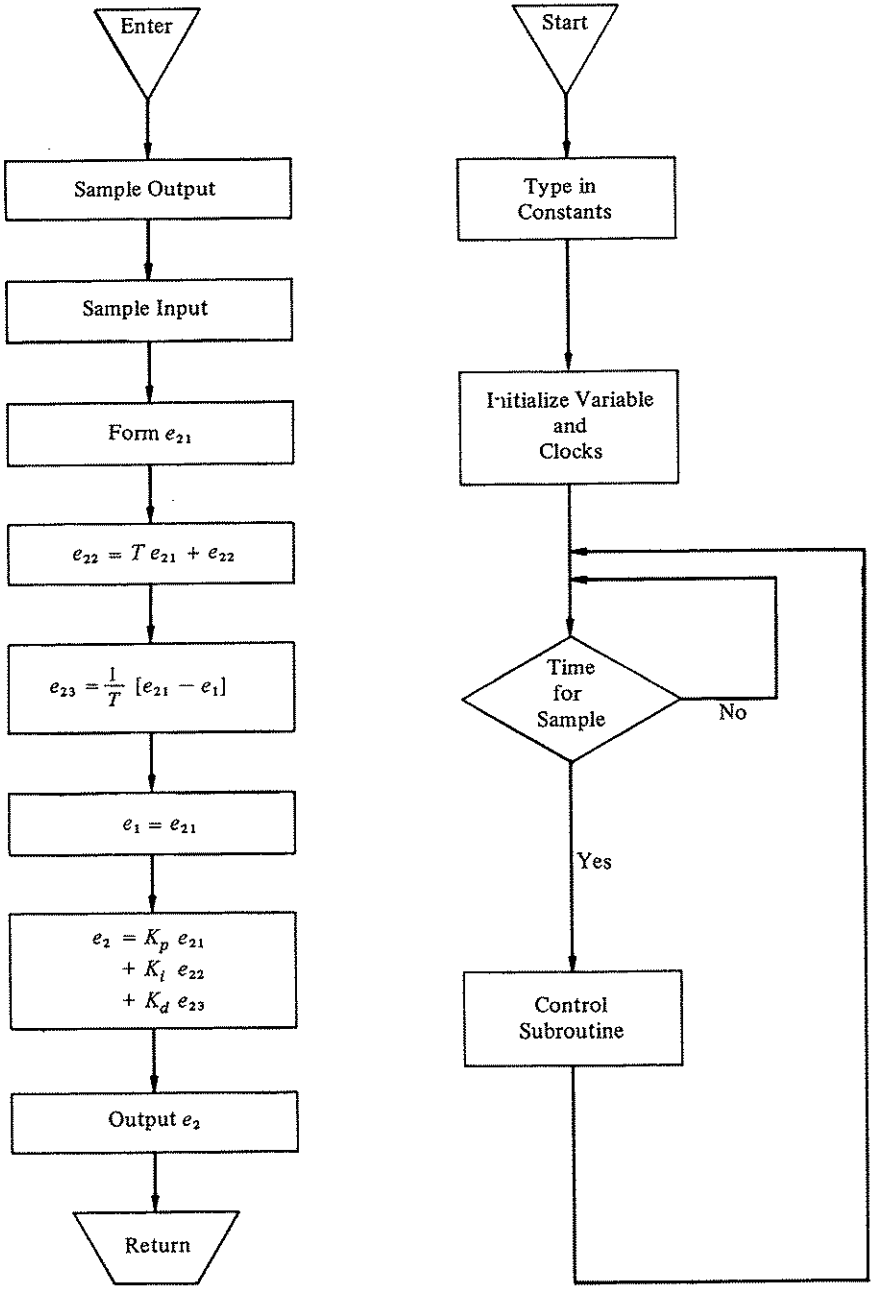


Figure 9.8-16. Flow chart for simulation program: (a) control routine; (b) timing routine.

Table 9.8-4 Listing of Program†

```
I CONTROLLER PROGRAM
/STARTING ADDRESS 200

INPUT = 0003
OUTPUT = 0004
PRNT = 0005

*5
IN,     7400            /ADDRESS OF INPUT ROUTINE
OUT,    7200            /ADDRESS OF OUTPUT ROUTINE
FLOAT,  5600            /ADDRESS OF FLOATING-POINT SYSTEM

*200
BEGIN,  KCC             /CLEAR READER FLAG AND AC
        TLS             /TRANSMIT 0 TO PRINTER
        JMS I FLOAT     /ENTER INTERPRETER
        FGET ZERO       /FAC = 0
        FPUT E21        /INITIALIZE VARIABLES
        FPUT E22
        FPUT E23
        FPUT EK1
        FPUT E2
        FPUT TEMP
        FGET TOUT
        PRNT            /PRINT LF T =
        INPUT           /READ VALUE OF T
        FPUT T          /STORE IT
        FGET KPOUT
        PRNT            /PRINT KP =
        INPUT           /READ VALUE OF KP
        FPUT KP         /STORE IT
        FGET KIOUT      /PRINT KI =
        PRNT
        INPUT           /READ VALUE OF KI
        FPUT KI         /STORE IT
        FGET KDOUT
        PRNT            /PRINT KD =
        INPUT           /READ VALUE OF KD
        FPUT KD         /STORE IT
        FEXT
        CLA CMA         /AC = -1
        DCA TIME        /INITIALIZE CLOCK
WAIT,   TAD TIME        /GET TIME
        SPA CLA         /TIME FOR NEXT SAMPLE?
        JMP WAIT        /NO, GO WAIT
        CLA CMA         /YES, AC = -1
        DCA TIME        /INITIALIZE TIME
        JMS I PID       /GO TO PID CONTROLLER
        CLA
```

†The floating-point package (DIGITAL 8-5A-S) must be loaded before this program is loaded.

Table 9.8-4 (cont.)

```
        6311            /SKIP IF SW 2 OFF
        JMP WAIT
        JMP BEGIN       /SW 2 OFF, GET NEW DATA
PID,    PIDPRO
*400
PIDPRO, 0
        ADCC            /CLEAR MPX TO 0
        ADCV            /START CONVERSION OF C
        CLA
        ADSF            /IS CONVERSION DONE?
        JMP .-1         /NO, WAIT
        ADRB            /YES, READ OUTPUT C
        CIA             /-C
        DCA MC+1        /STORE -C
        CLA IAC         /AC = 1
        ADSC            /GO TO CHANNEL 1
        ADCV            /START CONVERSION OF R
        CLA
        ADSF            /IS CONVERSION OF R DONE?
        JMP .-1         /NO, WAIT
        ADRB            /YES, READ R
        DCA 45          /STORE IN FAC
        DCA 46          /ZERO LOW ORDER PART
        TAD C13         /11 IN DECIMAL
        DCA 44          /STORE AS EXP.
        JMS I FLOAT     /ENTER INTERPRETER
        FNOR            /NORMALIZE
        FPUT TEMP       /SAVE IT
        FGET MC         /GET -C
        FNOR            /NORMALIZE
        FADD TEMP       /R -C
        FPUT E21        /E21 = E1(K)
        FMPY T          /T*E21
        FADD E22        /T*E21+E22
        FPUT E22        /UPDATE E22
        FMPY KI         /KI*E22
        FPUT TEMP       /SAVE IT
        FGET E21        /E21
        FSUB EK1        /E21-EK1
        FDIV T          /<E1(K)-E1(K-1)>/T
        FPUT E23        /UPDATE E23
        FMPY KD         /KD*E23
        FADD TEMP       /KI*E22+KD*E23
        FPUT TEMP       /SAVE IT
        FGET E21        /E1(K)
        FPUT EK1        /E1(K-1) = E1(K)
        FMPY KP         /KP*E21
        FADD TEMP       /KP*E21+KI*E22+KD*E23
        FPUT E2         /E2
```


Table 9.8-4 (cont.)

```
        FEXT            /LEAVE INTERPRETER
        CLA
        TAD 44          /FETCH EXPONENT
        SZA SMA         /IS THE NUMBER <1?
        JMP .+3         /NO
        CLA             /YES, FIX IT TO 0
        JMP DONE+1
        TAD M13         /NO, SET BINARY POINT AT
        SNA             /11 PLACES TO RIGHT OF CURRENT POINT
        JMP DONE        /IT IS ALREADY THERE; ALL DONE
        SMA             /TEST TO SEE IF IT IS TOO LARGE
        JMP ERROR       /YES NUMBER >2**11
        DCA 44          /NO, SET SCALE COUNT
GO,     CLL             /0 TO C(L)
        TAD 45          /FETCH MANTISSA
        SPA             /IS IT <0?
        CML             /YES, PUT A 1 IN LEFT BIT
        RAR             /SCALE RIGHT
        DCA 45          /RESTORE IT
        ISZ 44          /TEST IF SHIFTED ENOUGH
        JMP GO          /NO, CONTINUE
DONE,   TAD 45
        6551            /OUTPUT M
LEAVE,  CLA
        JMP I PIDPRO    /LEAVE
ERROR,  CLA CLL
        TAD 45          /GET VALUE
        SPA CLA         /IS IT POSITIVE?
        CML             /NO, L←1
        TAD MAX         /AC = -2048
        SNL             /WAS ILLEGAL NUMBER NEGATIVE?
        CMA             /NO, CHANGE AC TO 2047
        6551            /OUTPUT MAX VALUE
        CLA
        6301            /SKP PRINTOUT IF SW 1 OFF
        JMS I OUT       /PRINT THE NUMBER IN ERROR
        CLA
        JMP I PIDPRO
MAX,    4000            /-2048
*100
TIME,   -1              /INCREMENTED BY EXTERNAL CLOCK
C13,    13              /DECIMAL 11
M13,    -13
MC,     13              /STORE -C
        0
        0
E21,    0               /PROPORTIONAL VALUE
        0
        0
```

Table 9.8-4 (cont.)

```
T,      0               /SAMPLE TIME
        0
        0
E22,    0               /INTEGRAL OF INPUT
        0
        0
E23,    0               /DERIVATIVE OF INPUT
        0
        0
EK1,    0               /INPUT DELAYED ONE SAMPLE PERIOD
        0
        0
KP,     0               /PROPORTIONAL CONSTANT
        0
        0
KI,     0               /INTEGRAL CONSTANT
        0
        0
KD,     0               /DERIVATIVE CONSTANT
        0
        0
E2,     0               /OUTPUT
        0
        0
TEMP,   0
        0
        0
ZERO,   0
        0
        0
TOUT,   212             /LINE FEED
        324             /T
        275             /=
KPOUT,  313             /K
        320             /P
        275             /=
KIOUT,  313             /K
        311             /I
        275             /=
KDOUT,  313             /K
        304             /D
        275             /=

*6547
        7400
        7200
        3000

*3000
PRNTO,  0               /ALPHANUMERIC PRINTOUT
```


Table 9.8-4 (cont.)

```
        DCA  SAVE       /SAVE ACCUMULATOR
        TAD  44         /GET FIRST CHARACTER FROM FAC
        TSF             /IS PRINTER READY?
        JMP  .-1        /NO, WAIT
        TLS             /YES, PRINT
        CLA
        TAD  45         /GET SECOND CHARACTER
        TSF
        JMP  .-1
        TLS
        CLA
        TAD  46         /GET THIRD CHARACTER
        TSF
        JMP  .-1
        TLS
        CLA
        TAD  SAVE       /RESTORE AC
        JMP  I PRNTO    /LEAVE
SAVE,   0
```

The above examples of computer control systems are perhaps not overly significant from an engineering point of view, but they do serve to illustrate the role that a computer assumes as an active component in a system. In all cases the computer is programmed to periodically execute statements that implement control relationships based upon analytical derivations. The examples are sufficiently simple to permit alternate implementation approaches that do not require a digital computer. However, the use of a computer is completely justified if one realizes that there may be many such control tasks carried by the computer on a time-shared basis. Then there exists almost unlimited flexibility for modification of the computer programs.

9.9 Summary

This chapter discusses a variety of ways in which digital and analog computers, either singly or in combination, may be utilized for work in system analysis and design. Given here are some of the major applications.

Analysis:

1. Computer routines –Digital
2. Computer simulation–Analog, digital, hybrid
3. Simulation languages–Digital

Design:

1. Generation of control laws –Digital

2. Implementation of control laws –Analog, digital
3. Time-shared computer control systems–Digital

In each of these categories the capabilities of a computer are employed in distinct ways.

PROBLEMS

9.1 For the following transfer functions, generate digital simulation models using the Euler method, Tustin method, and State Transition method. Put all results into a linear recursion form, and compare the structure of the model and the coefficients. Investigate the stability and steady-state performance. Finally, perform a digital computation with a step input and an integration interval $T = .1$ second.

(a) $G(s) = \dfrac{1}{s(s+1)}$

(b) $G(s) = \dfrac{s+1}{s(s+4)}$

9.2 Investigate the stability of the discrete model as generated by the Runge-Kutta method for the transfer function of Problem 9.1(a) and the following three choices of integration interval $T = .1$, $T = 1$, and $T = 4$. For this purpose it is necessary to obtain a linear recursion equation.

9.3 Devise a scheme by which the Runge-Kutta-Merson method is used to control the integration interval for a numerical integration. Give consideration to the following: The step size should be decreased when the local truncation error estimate exceeds a preselected maximum level, the step size should be left unchanged when the error falls within a range of acceptable error, and it should be increased when the error is smaller than a preselected minimum level. The range of acceptable error should be in some relation to the step size change, keeping in mind that the local truncation error is proportional to the integration interval raised to the fifth power.

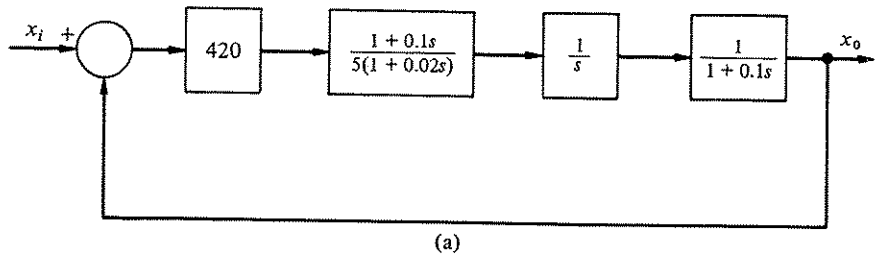
9.4 Perform a digital simulation using the State Transition method of the system defined by Figure 3.5-2. Use the computer program to compute $A(T)$ and $B(T)$.

9.5 Construct an analog computer simulation model for the transfer function

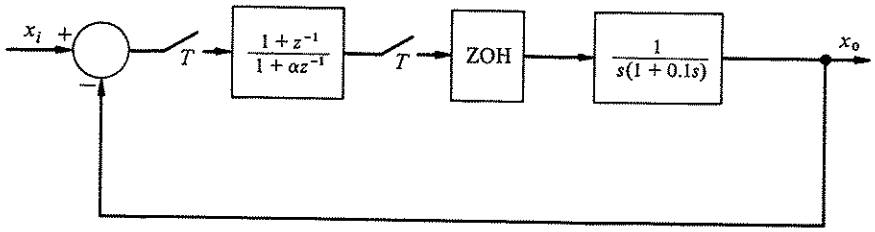
$$G(s) = \frac{s+a}{s+b}$$

by giving consideration to the fact that $a > b$ or $a < b$.

9.6 Prepare a MIMIC program for the digital simulation of the systems shown in Figure P9.6(a) and (b).



(a)



(b)

Figure P9.6

9.7 Perform an analog-hybrid computer simulation of the system shown in Figure P9.6(b). Consider α a parameter to be adjusted.

9.8 Repeat Problem 9.7, but use a hybrid computer for the simulation.