Eastern Economy Edition

System Simulation with Digital Computer

Narsingh Deo

Rs. 95.00

SYSTEM SIMULATION WITH DIGITAL COMPUTER
by Narsingh Deo

ISBN-81-203-0028-9

Nineteenth Printing ... ... September, 2006

Published by Asoke K. Ghosh, Prentice-Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001 and Printed by Rajkamal Electric Press, B-35/9, G.T. Kamal Road Industrial Area, Delhi-110033.

Contents

SYSTEM SIMULATION
WITH
DIGITAL COMPUTER

1

Introduction

Simulation is a powerful technique for solving a wide variety of problems. To simulate is to copy the behaviour of a system or phenomenon under study. Strictly speaking, we will be dealing with only *numerical sequential simulation;* numerical because there are other forms of simulation—for example, electrical analogue or physical simulation; and sequential because the calculations proceed in a time sequence. Some of the basic ideas in simulation can be best understood by performing actual simulations. Let us, therefore, consider the following two very simple examples and see how simulation is actually done.

1-1. Simulation of a pure pursuit problem—an example

A fighter aircraft sights an enemy bomber and flies directly toward it, in order to catch up with the bomber and destroy it. The bomber (the target) continues flying (along a specified curve) so the fighter (the pursuer) has to change its direction to keep pointed toward the target. We are interested in determining the attack course of the fighter and in knowing how long it would take for it to catch up with the bomber.

If the target flies along a straight line, the problem can be solved directly with analytic techniques. (The proof of such a closed-form expression which gives the course of the pursuer, when the target flies in a straight line, is left as an exercise for you. Problem 1-2.)

However, if the path of the target is curved, the problem is much more difficult and normally cannot be solved directly. We will use simulation to solve this problem, under the following simplifying conditions:

1. The target and the pursuer are flying in the same horizontal plane when the fighter first sights the bomber, and both stay in that plane. This makes the pursuit model two-dimensional.
2. The fighter's speed VF is constant (20 kms/minute).
3. The target's path (i.e., its position as a function of time) is specified.
4. After a fixed time span Δt (every minute, in this case) the fighter changes its direction in order to point itself toward the bomber.

Let us introduce a rectangular coordinate system coincident with the horizontal plane in which the two aircraft are flying. We choose the point due south of the fighter and due west of the target (at the beginning of the pursuit) as the origin of this coordinate system. Let the distances be given

in kilometers and the time in minutes. We start measuring the time when the fighter first sights the bomber. (See Fig. 1-1.)

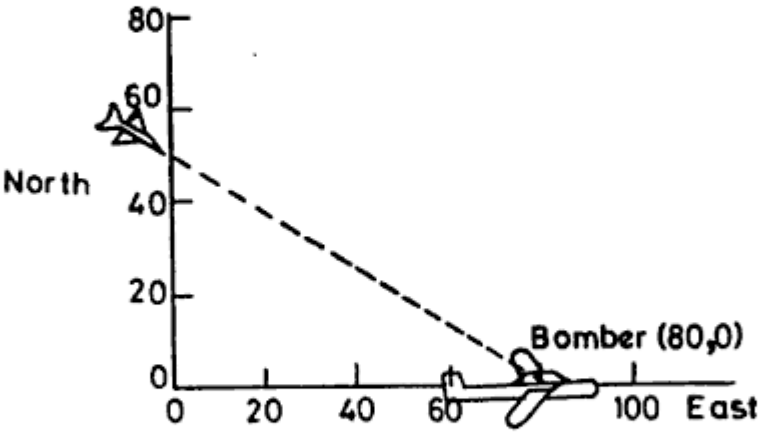


Fig. 1-1: Positions of Pursuer and Target at Time Zero.

We will represent the path of the bomber (which is known to us in advance) by two arrays, the east coordinates and the north coordinates at specified moments (each minute). We call these coordinates $XB(t)$ and $YB(t)$, respectively. They are presented in the form of a table (in kilometers) below.

Time, t	0	1	2	3	4	5	6	7	8	9	10	11	12
$XB(t)$	80	90	99	108	116	125	133	141	151	160	169	179	180
$YB(t)$	0	−2	−5	−9	−15	−18	−23	−29	−28	−25	−21	−20	−17

Table 1-1.

Likewise, we will represent the path of the fighter plane by two arrays $XF(t)$ and $YF(t)$. In this example, initially we are given

$$YF(0) = 50 \text{ kms}, \quad XF(0) = 0 \text{ kms}.$$

Our purpose is to compute the positions of the pursuer, namely, $XF(t)$, $YF(t)$ for $t = 1, 2, \ldots, 12$, or until the fighter catches up with the bomber. We will assume that once the fighter is within 10 kms of the bomber, the fighter shoots down its target by firing a missile, and the pursuit is over. In case the target is not caught up within 12 minutes, the pursuit is abandoned, and the target is considered escaped. From the time $t = 0$ till the target is shot down, the attack course is determined as follows:

The fighter uses the following simple strategy: It looks at the target at instant t, aligns its velocity vector with the line of sight (i.e., points itself toward the target). It continues to fly in that direction for one minute,

till instant $(t+1)$. At time $(t+1)$ it looks at the target again and realigns itself.

The distance DIST (t) at a given time t between the target and the pursuer is given by

$$\text{DIST}(t) = \sqrt{(YB(t) - YF(t))^2 + (XB(t) - XF(t))^2} \qquad \ldots(1\text{-}1)$$

The angle θ of the line from the fighter to the target at a given time t is given by

$$\text{Sin}\,\theta = \frac{YB(t) - YF(t)}{\text{DIST}(t)} \qquad \ldots(1\text{-}2)$$

$$\text{Cos}\,\theta = \frac{XB(t) - XF(t)}{\text{DIST}(t)} \qquad \ldots(1\text{-}3)$$

Using this value of the position of the fighter at time $(t+1)$ is determined by

$$XF(t+1) = XF(t) + VF\,\text{Cos}\,\theta \qquad \ldots(1\text{-}4)$$

$$YF(t+1) = YF(t) + VF\,\text{Sin}\,\theta \qquad \ldots(1\text{-}5)$$

With these new coordinates of the pursuer, its distance from the target is again computed using Eq. (1-1). If this distance is 10 kms. or less the pursuit is over, otherwise θ is recomputed, and the process continues.

A flowchart of the logic of this problem is given in Fig. 1-2 (page 4).

The following FORTRAN program (a format-free version) will implement the flowchart.

```
      DIMENSION XB(25), YB(25), XF(25), YF(25)
      INTEGER T, J
      READ, (XB(T), YB(T), T = 1,13)
      READ, XF(1), YF(1), VF
      T = 1
  100 DIST = SQRT((YB(T) - YF(T))** 2 + (XB(T) - XF(T))** 2)
      IF (DIST. LE. 10.0) GO TO 110
      IF (T.GT.12) GO TO 120
      XF(T + 1) = XF(T) + VF* (XB(T) - XF(T))/DIST
      YF(T + 1) = YF(T) + VF* (YB(T) - YF(T))/DIST
      T = T + 1
      GO TO 100
  110 PRINT 990, T, DIST
  990 FORMAT (10X, 10H CAUGHT AT, 13, 8H MTS AND, F10.3, 4H KMS)
      STOP
  120 PRINT 1000
 1000 FORMAT (10X, 16H TARGET ESCAPED)
      STOP
      END
```

(Note that since Fortran does not permit 0 as an index we had to set $T = 1$ to correspond to $t = 0$ in the flowchart.)

The output of this program for the specified data is as follows:

```
CAUGHT AT 10 MTS AND 2.969 KMS
```

This is an example of simulation. We had to resort to simulation be-

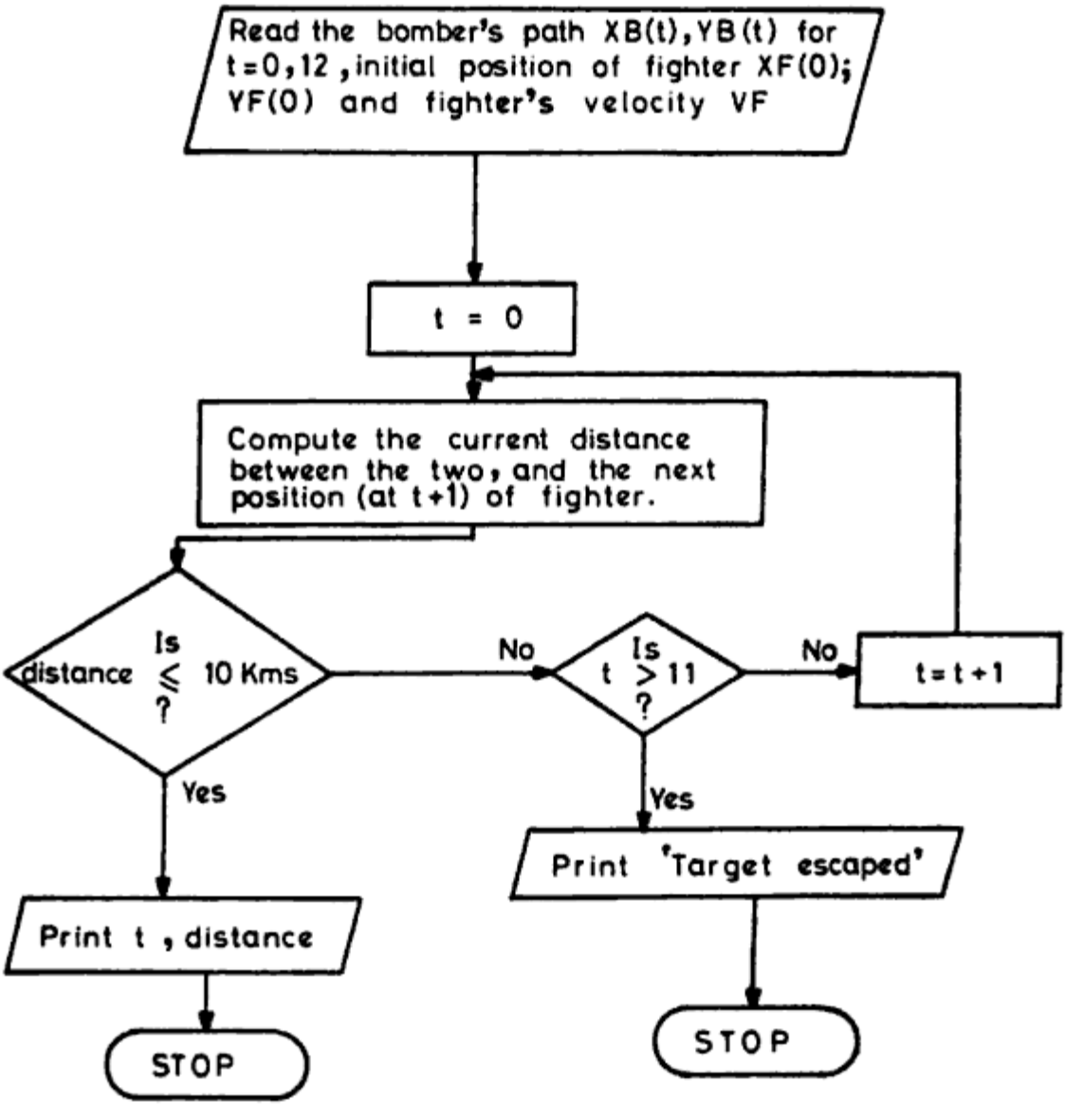


Fig. 1-2: Flowchart of pursuit problem.

cause analytically we could not make a long-term prediction about the path that the fighter plane would take (given the initial position and path of the target). But by simulation we were able to make the computer go through the instant-to-instant predictions for as many instants as we wanted. This was possible only because we knew the basic process involved, namely, how the fighter plane behaves at any particular instant. Such knowledge of the basic process of the system under study is essential for all simulation experiments.

The simple strategy, of pursuer redirecting himself toward the target at fixed intervals of time, while the target goes on its predetermined path without making any effort to evade the pursuer, is called *pure pursuit*. Although in many situations, the strategy used by the pursuer is more sophisticated, this basic approach can be used for any pursuit problem as long as we know

the path the pursued takes and the rule by which the pursuer redirects himself.

1-2. A system and its model

In any simulation experiment we have a system whose behaviour we are studying. Loosely speaking, a *system* is a collection of distinct objects which interact with each other. In the pure-pursuit example, the system consists of the two aircraft. In order to study a system we generally gather some relevant information about it. For most studies it is not necessary to consider all the details of the system. For example, in the example in Sec. 1-1 we did not get the information about the sizes or weights of the aircraft. Such a collection of pertinent information about a system is called a *model* of the system.

The construction of an appropriate model of a system under study is a delicate and an all-important affair in simulation (as it is in analytic study). For example, we have assumed that the target takes no evasive action, nor does the pursuer use any predictive technique. We assumed that the pursuer corrects its course only once a minute. Thus we know the 'law' that governs the system. The same system may be described by different models. Several variations of the pure-pursuit model are considered in problems at the end of this chapter.

Constructing appropriate mathematical models of complex, real-life systems is a vast and intricate subject in itself. It requires a thorough knowledge of the system as well as of modelling techniques. In this book, we will in most cases assume that a model has been arrived at and our concern is how to program the computer so that it behaves like the model as the time progresses.

For a given system there are a number of variables that describe what is known as the *state of the system* at any given time. For example, the state of our system (of pure pursuit) is described by the positions of the two aircraft and their velocities.

For further understanding these and other concepts, let us simulate another very simple system, which is different in a fundamental sense from the pure-pursuit system.

1-3. Simulation of an inventory problem

Suppose you work in a retail store and it is your responsibility to keep replenishing a certain item (say, automobile tyres) in the store by ordering it from the wholesaler. You want to adopt a simple policy for ordering new supplies:

> 'When my stock goes down to P items (called *reorder point*), I will order Q more items (called *reorder quantity*) from the wholesaler.'

If the demand on any day exceeds the amount of inventory on hand, the

excess represents lost sale and loss of goodwill. On the other hand, overstocking implies increased carrying cost (i.e., cost of storage, insurance, interest, deterioration, etc.). Ordering too frequently will result in excessive reorder cost. Assume the following conditions:

1. There is a 3-day lag between the order and arrival. The merchandise is ordered at the end of the day and is received at the beginning of the fourth day. That is, merchandise ordered on the evening of the ith day is received on the morning of the $(i + 3)$rd day.
2. For each unit of inventory the carrying cost for each night is Re. 0.75.
3. Each unit out of stock when ordered results into a loss of goodwill worth Rs. 2.00 per unit plus loss of Rs. 16.00 net income, that would have resulted in its sale, or a total loss of Rs. 18.00 per unit. Lost sales are lost forever; they cannot be backordered.
4. Placement of each order costs Rs. 75.00 regardless of the number of units ordered.
5. The demand in a day can be for any number of units between 0 and 99, each equiprobable.
6. There is never more than one replenishment order outstanding.
7. Initially we have 115 units on hand and no reorder outstanding.

With these conditions in force you have been asked to compare the following five replenishment policies and select the one that has the minimum total cost (i.e., reorder cost + carrying cost + lost sales cost).

	P (reorder point)	Q (reorder quantity)
Policy I	125	150
Policy II	125	250
Policy III	150	250
Policy IV	175	250
Policy V	175	300

(Since we are interested in simulation here and not in inventory theory, we will not investigate the reasonableness of the replenishment policy too critically. Ours is undoubtedly a very simplified model.)

The problem does not easily lend itself to an analytic solution; it is best therefore to solve it by simulation. Let us simulate the running of the store for about six months (180 days) under each of the five policies and then compare their costs.

A simulation model of this inventory system can be easily constructed by stepping time forward in the fixed increment of a day, starting with Day 1, and continuing up to Day 180. On a typical day, Day i, first we check to see if merchandise is due to arrive today. If yes, then the existing stock S is

increased by Q (the quantity that was ordered). If DEM is the demand for today, and $DEM \leqslant S$, our new stock at the end of today will be $(S - DEM)$ units. If $DEM > S$, then our new stock will be zero. In either case, we calculate the total cost resulting from today's transactions, and add it to the total cost C incurred till yesterday. Then we determine if the inventory on hand plus units on order is greater than P, the reorder point. If not, place

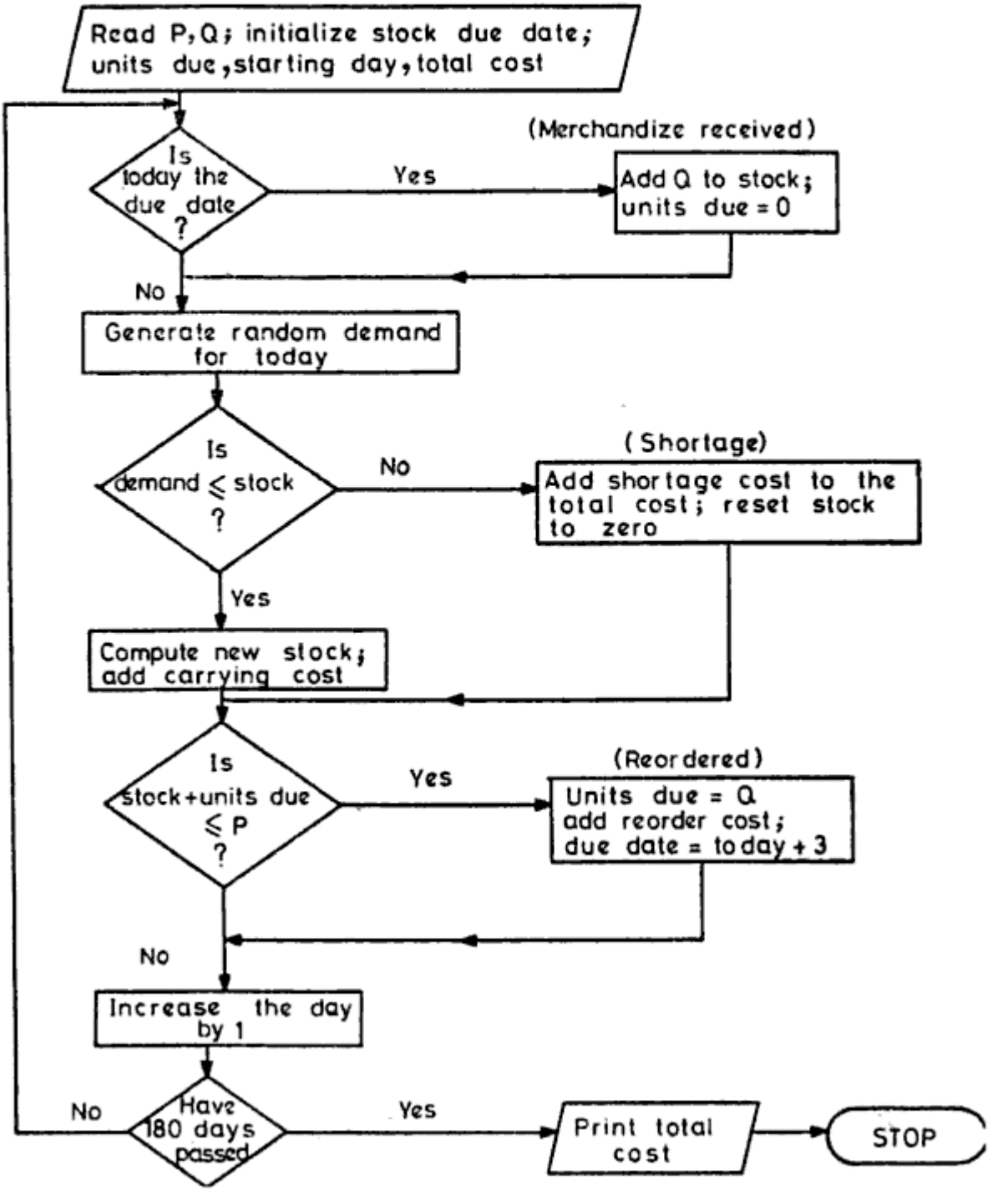


Fig. 1-3: Cost of inventory policy (P, Q).

an order (to be delivered three days hence), by stating the amount ordered and the day it is due to be received. We repeat this procedure for 180 days.

Initially we set day number $i = 1$, stock $S = 115$, number of units due $UD = 0$ (because there is no outstanding order), and the day they are due $DD = 0$.

The demand, *DEM*, for each day is not a fixed quantity but a random variable. It could assume any integral value from 00 to 99 and each with equal probability. We will use a special subroutine, which generates a 2-digit random integer, and will use the numbers thus produced as the daily demands. (Random number generators will be discussed in Chapter 3.) A flowchart for simulating this problem is shown in Fig. 1-3 (page 7).

The following Fortran program (format free) implements the flowchart for evaluating the total cost for a given replenishment policy (P, Q) for 180 days. Statement No. 110 in the program makes use of the subprogram *RNDY1* (*DUM*) which is a subprogram to generate a real pseudorandom number between zero and one. The argument of this function can be any dummy number or variable.

Notice how condition 6, that there is no more than one reorder outstanding, has been taken care of. In Statement 130, we add the number of units due (if any) *UD* to the current stock S to get the equivalent stock, *ES*. It is this number which is compared to P before an order is placed. Since $UD > P$ if we already have a replenishment order outstanding another order will not be placed.

This is an extremely simple model of an inventory-control system. More realistic models will be dealt with in Chapter 6

```
      INTEGER P, Q, S, ES, UD, DD, DEM
      READ, P, Q
      C = 0.0
      S = 115
      I = 1
      UD = 0
      DD = 0
  100 IF (DD .NE. I) GO TO 110
      S = S + Q
      UD = 0
  110 DEM = RNDY1 (DUM)* 100.0
      IF (DEM .LE. S) GO TO 120
      C = C+ (FLOAT (DEM) – FLOAT (S))*18.0
      S = 0
      GO TO 130
  120 S = S – DEM
      C = C + FLOAT (S)*.75
  130 ES = S + UD
      IF (ES .GT. P) GO TO 140
      UD = Q
      DD = I + 3
      C = C + 75.0
```

```
140  I= I + 1
     IF (I .LE. 180) GO TO 100
     PRINT, P, Q, C
     STOP
     END
```

The program yielded the following cost figures for the five inventory policies:

P	Q	Cost in Rs.
125	150	38679.75
125	250	31268.25
150	250	29699.25
175	250	26094.00
175	300	27773.25

Thus, Policy IV ($P = 175$, $Q = 250$) is the best amongst the five considered.

1-4. The basic nature of simulation

There does not seem to be much that is common between the two problems and the methods of solving them. The first problem (the pure pursuit) is basically continuous, in the sense that its state changes continuously with time; whereas the second problem is discrete, because the arrival and sale of merchandise occurs only in discrete steps. The first problem is deterministic, while the second one is stochastic in nature. Yet there are some common features and these are the features essential to simulation.

In both cases we started from a mathematical model of the system under study. Some initial conditions (state at time zero) were assumed in both cases. The change of state occurred in accordance with some equations (rules or laws). Using these rules numerically we continued changing the state of the system as time moved forward. This was done for as long a period as needed. At the end of this period we collected the desired information about the system (which is the solution to the problem). These calculations were programmed for a digital computer. Thus the computer was made to simulate or mimic the real-life system as its variables changed. Through this process we managed to get around the necessity of obtaining an analytic solution and therein lies the great advantage of simulation.

It should be noted that the simulation in both examples (pure pursuit as well as inventory control) could have been performed manually with pencil and paper. In theory any system that can be simulated on a digital computer can also be simulated by hand. In practice, however, simulation as an analytic tool is useful only when done on a computer. This is because the practical problems that require simulation are complex and need a very

large number of simple, repetitive calculations. In fact, simulation is one of the most important uses of the digital computer.

To simulate is to experiment: Simulation is basically an experimental technique. It is a fast and relatively inexpensive method of doing an 'experiment' on the computer. Consider, for example, the inventory-control problem. Instead of trying out in the actual store the five replenishment policies, each for a period of six months, and then selecting the best one, we conducted the experiment on the computer and obtained the results within a few minutes, at a very small cost (provided, of course, our model reflects reality). This is why computer simulation is often referred to as performing a *simulation experiment.*

No unified theory: There is no unifying theory of computer simulation. Learning simulation does not consist of learning a few fundamental theorems and then using them and their various corollaries to solve problems. There are no underlying principles guiding the formulation of simulation models. Each application of simulation is *ad hoc* to a great extent. In this sense simulation is an art, and one often hears the expression 'the art of simulation.'

1-5. When to simulate

All of us in our daily lives encounter problems, which although mathematical in nature, are too complex to lend themselves to exact mathematical analysis. The performance of such a system (say, weather or traffic jam) may be difficult to predict, either because the system itself is complex or the theory is not yet sufficiently developed. The difficulties in handling such problems (by means of classical mathematical tools) may also arise due to the effect of uncertainties, or due to dynamic interactions between decisions and subsequent events, or due to complex interdependencies among variables in the system, or due to some combination of these. Formerly (i.e., before the days of computer simulation), in such situations either an intuitive decision was made, or, if the stakes were too high to rely on intuition, elaborate laboratory experiments had to be conducted, which were usually expensive and time consuming. Simulation provides a third alternative which is cheap and fast, and thus fills the gap between exact analysis and physical intuition. Occasionally, simulation is also used even when an exact analytic solution is possible, but it is too expensive in terms of computation time.

Simulation in science and engineering research: Simulation has changed, in a very fundamental sense, the way in which research is conducted today. Earlier most experiments were carried out physically in the laboratories. Thousands and even millions were spent on physical models (e.g., wind tunnels, river basin models, network analyzers, aircraft flight simulators) and expensive experiments. Today a majority of these experiments are simulated on a computer. 'Computer experiments' besides being much faster, cheaper, and easier, frequently provide better insight into the system than laboratory experiments do. Not all laboratory experiments, of course,

can be replaced with computer simulations. Typically, a few key experiments are performed in the laboratory after, say, 80–90 per cent of the experimenting has been done on the computer.

Simulation in soft sciences: Simulation can be expected to play even a more vital role in biology, sociology, economics, medicine, psychology, etc., where experimenting could be very expensive, dangerous, or even impossible. In these areas, the mathematical theories are even less developed than in physical sciences. Moreover, in fields such as biology and economics the problems are truly large, involving tens of thousands of variables. The complications caused by uncertainty are also greater in these areas than in physical sciences. Thus simulation has become an indispensable tool for a modern researcher in most social, biological and life sciences.

Simulation for business executive: There are many problems faced by management that cannot be solved by standard operations research tools like linear and dynamic programming, inventory and queueing theory. Therefore, a business executive had to make decisions based solely on his intuition and experience. Now he can use computer simulation to make better and more meaningful decisions. Utilizing the power of a digital computer, he can build and study a simulation model (of his system) containing arbitrarily high-order complexities and a huge number of interdependencies, as well as uncertainties. Simulation has been used widely for inventory control, facility planning, production scheduling, and the like.

1-6. Remarks and references

Simulation is a very powerful, problem-solving technique. Its applicability is so general that it would be hard to point out disciplines or systems to which simulation has not been applied. The basic idea behind simulation is simple, namely, model the given system by means of some equations and then determine its time-dependent behaviour. The simplicity of approach when combined with the computational power of the high-speed digital computer makes simulation a powerful tool. Normally, simulation is used when either an exact analytic expression for the behaviour of the system under investigation is not available, or the analytic solution is too time-consuming or expensive.

Since the popularity and usefulness of simulation is dependent on the capabilities and availability of the computer, the subject essentially started in the late 1950's. It has grown—in its power, sophistication, and applicability—very rapidly in the past 15–20 years, and is still expanding at a rapid rate. One of the early textbooks and still a good one is:

TOCHER, K.D., *The Art of Simulation*, Van Nostrand Co., Princeton, N.J., 1963.

An extremely readable and elementary 100-page textbook on essentials of computer simulation is

SMITH, J., *Computer Simulation Models*, Hafner Publishing Company, New York, N.Y., 1968.

Some of the other general textbooks on simulation are:

EMSHOFF, J. R. and R. L. SISSON, *Design and Use of Computer Simulation Models*, Macmillan Co., New York, 1970.

GORDON, G., *System Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1969.

MARTIN, F.F., *Computer Modelling and Simulation*, John Wiley and Sons, New York, 1968.

MIZE, J. H. and J. G. COX, *Essentials of Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1968.

NAYLOR, T. H., J. L. BALINTFY, D.S. BURDICK, and K. CHU, *Computer Simulation Techniques*, John Wiley and Sons, New York, 1966.

SHANNON, R. E., *Systems Simulation: the Art and Science*, Prentice-Hall, Englewood Cliffs, N.Y., 1975.

FISHMAN, G. S., *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley and Sons, New York, 1973.

ZEIGLER, B.P. *Theory of Modelling and Simulation*, Wiley-Interscience, New York, 1976.

MAISEL, H. and GNUGNOLI, *Simulation of Discrete Stochastic Systems*, Science Research Associates, Chicago, 1975.

The literature on simulation is vast. Numerous bibliographies and survey articles on simulation have been published, containing hundreds of references. Many of these references can be found in the textbooks given above.

Simulation requires several skills. You must be familiar with the system being simulated and know enough of system theory and modelling techniques to be able to make a realistic model of the system. The five different models of the pure-pursuit problem in Exercises 1-2 to 1-6 in the next section are included to emphasize the importance of modelling. Secondly, a certain amount of skill in computer programming is also required. Also, some knowledge of statistics is needed for designing the simulation experiment. Different textbooks lay different amounts of emphasis on these three aspects of simulation.

Besides general textbooks on simulation, there are a number of specialized texts devoted to application of simulation to a particular area such as economics, industrial systems, psychology, social sciences, international relations, ecology, and war games.

There have been a number of cases where more than a million dollars have been spent on a single simulation project. Many of the large successful simulation projects are never publicized because of their military application or their proprietary interest of private industry.

1-7. Exercises

1-1. Identify six different problems from your own experience that you think should be solved using digital simulation rather than analytically.

1-2. In the pure-pursuit problem of Sec. 1-1, suppose the initial (x, y) coordinates of the bomber at time 0 are $(0, 0)$ and of the fighter $(0, d)$. The bomber (the target) moves in a straight line along the x axis (east) at a constant velocity VB and the fighter at a velocity VF pointing itself continously

toward the target. Show that the path followed by the fighter can be described by the following equation:

$$x = \tfrac{1}{2}\left[\frac{y^{1+r}}{(1+r)\,d^{r}} - \frac{y^{1-r}}{(1-r)\,d^{-r}}\right] + \frac{r\,.\,d}{1-r^{2}},$$

where $$r = \frac{VB}{VF},$$

and (x, y) are the coordinates of the fighter.

1-3. In the special case when the target moves along the x axis, using the FORTRAN program given in Sec. 1-1, find the point where the pursuer catches up with the target. Compare the simulation result with the one obtained using the analytic expression in Exercise 1-2. Explain the difference between the two results, if any.

1-4. The pure-pursuit situation in Sec. 1-1 can also be simulated in terms of polar coordinates (R, θ) instead of rectangular coordinates (x, y). The relevant equations (2-dimensional problem) are

$$\frac{dR}{dt} = VB \cdot \cos\theta - VF$$

$$R\frac{d\theta}{dt} = -VB \cdot \sin\theta$$

where R is the distance between the bomber and the fighter (denoted by DIST in Sec. 1-1) and θ is the angle of the line from the fighter to the bomber. Write a FORTRAN program using polar coordinates.

1-5. Write a program for simulating a 3-dimensional pure-pursuit problem.

1-6. In the pure-pursuit course the fighter flies directly toward the target. In a *lead-pursuit course* the fighter flies in a direction ahead of the line of sight by a lead angle φ. Write a computer program to simulate a 2-dimensional lead-pursuit course with a specified lead angle, the velocities, and the initial positions of the two aircraft.

1-7. In a field there are four animals—a dog, a mongoose, a snake, and a mouse. Dogs kill mongooses, mongooses kill snakes, and snakes kill mice. The speeds of the mouse, snake, mongoose and dog are, respectively, 8, 12, 18 and 30 km/hour. Simulate the chase with different starting positions to see which animal gets killed first.

1-8. Simulate on a computer the following game between two players (called the game of matching pennies). The two players simultaneously flip a coin each. If both sides are the same (both heads or both tails), player A wins a rupee from B, otherwise B wins a rupee from A. Each player starts with 10 rupees. By conducting the experiment 500 times, find out how long the game will last on the average before one of the players goes broke.

1-9. Suppose you are the manufacturer of a car which is in great demand. At present you make 1,000 cars per month and sell it easily at Rs. 20,000 each (ignore excise duty, sales tax, etc., for the time being). By using overtime, etc., you can increase the production, but you must also increase the price in order to make a profit (10 per cent on sale). The number of cars you can produce and the corresponding price (of all units) is given below:

Cars	Production	1,000	1,200	1,300	1,500	2,000
	Price	20,000	21,000	22,000	25,000	30,000

The demand for your cars is predicted to rise at 5 per cent per year provided the price stays at Rs. 20,000 each. As the price rises a certain number of customers will switch to other cars (or scooters). This switchover percentage is estimated to be as below:

Price	20,000	21,000	22,000	25,000	30,000
Percentage switched	0	5	10	25	50

By investing capital you can increase your production without increasing the cost per unit. But you must pay 8 per cent interest on the additional capital. The investments and the corresponding increases in production are as follows:

Investment in Rs.	1,000,000	2,000,000	5,000,000
Increase in production of cars	300	500	1,000

Simulate the company's situation over the next 20 years. By experimenting with the model find out which investment should be made and when (if any).

2

Simulation of Continuous Systems

From the viewpoint of simulation there are two fundamentally different types of systems: (i) systems in which the state changes smoothly or continuously with time are called *continuous systems*, and (ii) systems in which the state changes abruptly at discrete points in time are called *discrete systems*. The system of pure pursuit in Sec. 1-1 is an example of a continuous system, because the positions of the two aircraft can vary continuously with time. The inventory system in Sec. 1-3, on the other hand, is a discrete system because the addition to and the demand from the stock can occur only in discrete numbers. Hence the stock-level can change only discretely with time. A queueing situation is another example of a discrete system because the customers join or leave the queue only in discrete numbers. Usually, the simulation of most systems in engineering and physical sciences turns out to be continuous, whereas most systems encountered in operations research and management science are discrete. Most studies of communication, transportation, and computer systems also involve discrete system simulation. As we will see in this and subsequent chapters, the methodologies of discrete and continuous simulations are inherently different. In this chapter we will deal with continuous systems only.

Continuous dynamic systems, those systems in which the state or the variables vary continously with time, can generally be described by means of differential equations. If the set of (simultaneous) differential equations describing a system are ordinary, linear, and time-invariant (i.e., have constant coefficients), an analytic solution is usually easy to obtain. In general, differential equations of a more difficult nature can only be solved numerically. Simulating the system often gives added insight into the problem besides giving the required numerical solution. As an elementary example of a continuous dynamic system let us consider the following simple chemical plant.

2-1. A chemical reactor

In a certain chemical reaction when two substances A and B are brought together they produce a third chemical substance C. It is known that 1 gram of A combines with 1 gram of B to produce 2 grams of C. Furthermore, the rate of formation of C is proportional to the product of the amounts of A and B present. In addition to this *forward reaction* there is also a *backward reaction* decomposing C back into A and B. The rate of decomposi-

tion of C is proportional to the amount of C present in the mixer. In other words, at any time t if a, b, and c are the quantities of the chemicals A, B and C present, respectively, then their rates of increases are described by the following three differential equations:

$$\frac{da}{dt} = k_2 c - k_1 ab, \qquad \ldots(2\text{-}1)$$

$$\frac{db}{dt} = k_2 c - k_1 ab, \qquad \ldots(2\text{-}2)$$

$$\frac{dc}{dt} = 2k_1 ab - 2k_2 c, \qquad \ldots(2\text{-}3)$$

where k_1 and k_2 are the *rate constants*. (These constants will vary with temperature and pressure, but we do not allow the temperature or pressure of the reaction to vary.) Given the values of the constants k_1 and k_2 and the initial quantities of the chemicals A and B (and $c = 0$), we wish to determine how much of C has been produced as a function of time. Determination of the rate of such chemical reactions is important in many industrial processes.

A straightforward method of simulating this system is to start at time zero and increment time in small steps of Δt. We assume that the quantities of chemicals remain unaltered during each step and only change 'instantaneously' at the end of the step. Thus the quantity of A (or B or C) at the end of one such step is given in terms of the quantity at the beginning of the step as

$$a(t + \Delta t) = a(t) + \frac{da(t)}{dt} \cdot \Delta t \qquad \ldots(2\text{-}4)$$

If Δt is sufficiently small Eq. (2-4) is a reasonable representation. Identical equations can be written for $b(t + \Delta t)$ and $c(t + \Delta t)$.

Suppose we wish to simulate the system for a period T. We will divide this period T into a large number N of small periods Δt. That is

$$T = N . \Delta t$$

At time zero, we know $a(0)$, $b(0)$, $c(0)$. From these initial values and the values k_1 and k_2 we compute the amounts of chemicals at time Δt as

$$a(\Delta t) = a(0) + [k_2 . c(0) - k_1 . a(0) . b(0)] \Delta t$$
$$b(\Delta t) = b(0) + [k_2 . c(0) - k_1 . a(0) . b(0)] \Delta t$$
$$c(\Delta t) = c(0) + [2k_1 . a(0) . b(0) - 2k_2 . c(0)] \Delta t$$

Using these values we calculate the next state of the system, i.e., at time $2t$ as

$$a(2\Delta t) = a(\Delta t) + [k_2 . c(\Delta t) - k_1 a(\Delta t) . b(\Delta t)] . \Delta t$$
$$b(2\Delta t) = b(\Delta t) + [k_2 . c(\Delta t) - k_1 a(\Delta t) . b(\Delta t)] . \Delta t$$
$$c(2\Delta t) = c(\Delta t) + [2k_1 . a(\Delta t) . b(\Delta t) - 2k_2 . c(\Delta t)] . \Delta t$$

Using the state of the system at 2 Δt, we determine its state at 3 Δt, and so on. We continue in this vein, moving time forward by Δt and determining the state of the system from the previous state, for N steps, at the end of which we have the desired result. This procedure is shown in the form of a flow-chart in Fig. 2-1.

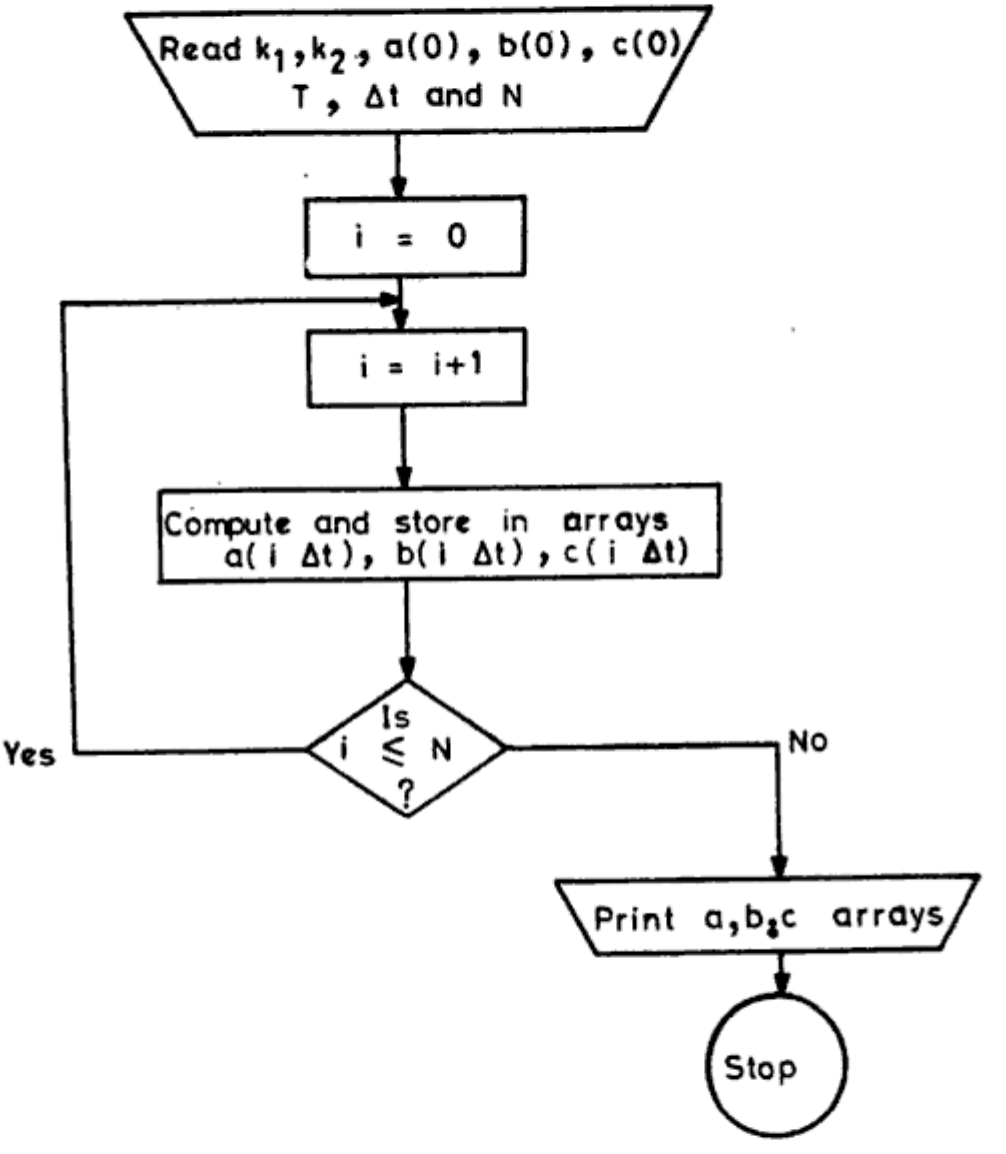


Fig. 2-1: Flowchart of a chemical reaction simulator.

A FORTRAN (format free) program which simulates the system with $k_1 = 0.008$ gram^{-1} min^{-1}, $k_2 = 0.002$ min^{-1}, and $a(0) = 100$ grams, $b(0) = 50$ grams, $c(0) = 0$ grams, for a period of $T = 5$ minutes in steps of $\Delta t = 0.1$ min. is as follows ($N = 50$).

```
      DIMENSION A(52), B(52), C(52),
      REAL K1, K2
      A(1) = 100.0
      B(1) = 50.0
      C(1) = 0.0
```

```
      DELTA = 0.1
      T = 0
      K1 = 0.008
      K2 = 0.002
      DO 3 I = 1,51
      PRINT, T, A(I), B(I), C(I)
      A(I + 1) = A(I) + (K2*C(I) - K1*A(I)*B(I))*DELTA
      B(I + 1) = B(I) + (K2*C(I) - K1*A(I)*B(I))*DELTA
      C(I + 1) = C(I) + 2.*(K1*A(I)*B(I) - K2*C(I))*DELTA
      T = T + DELTA
    3 CONTINUE
      STOP
      END
```

The following is the output of this program, which gives the state of the system (i.e., the values of *a*, *b*, and *c*) for 5 minutes at the intervals of 0.1 minute.

TIME	A(I)	B(I)	C(I)
0.00	100.00	50.00	0.00
0.10	96.00	46.00	8.00
0.20	92.47	42.47	15.06
0.30	89.33	39.33	21.34
0.40	86.52	36.52	26.95
0.50	84.00	34.00	32.00
0.60	81.72	31.72	36.55
0.70	79.66	29.66	40.69
0.80	77.77	27.77	44.45
0.90	76.05	26.05	47.89
1.00	74.48	24.48	51.04
1.10	73.03	23.03	53.94
1.20	71.70	21.70	56.61
1.30	70.46	20.46	59.07
1.40	69.32	19.32	61.36
1.50	68.26	18.26	63.48
1.60	67.28	17.28	65.44
1.70	66.36	16.36	67.28
1.80	65.51	15.51	68.99
1.90	64.71	14.71	70.59
2.00	63.96	13.96	72.08

(Continued...)

(. . . *Continued*)

TIME	A(I)	B(I)	C(I)
2.10	63.26	13.26	73.48
2.20	62.60	12.60	74.79
2.30	61.99	11.99	76.03
2.40	61.41	11.41	77.18
2.50	60.86	10.86	78.27
2.60	60.35	10.35	79.30
2.70	59.87	9.87	80.27
2.80	59.41	9.41	81.18
2.90	58.98	8.98	82.04
3.00	58.57	8.57	82.86
3.10	58.19	8.19	83.63
3.20	57.82	7.82	84.36
3.30	57.48	7.48	85.05
3.40	57.15	7.15	85.70
3.50	56.84	6.84	86.32
3.60	56.55	6.55	86.91
3.70	56.27	6.27	87.46
3.80	56.00	6.00	87.99
3.90	55.75	5.75	88.50
4.00	55.51	5.51	88.97
4.10	55.29	5.29	89.43
4.20	55.07	5.07	89.86
4.30	54.86	4.86	90.27
4.40	54.67	4.67	90.66
4.50	54.48	4.48	91.03
4.60	54.31	4.31	91.39
4.70	54.14	4.14	91.73
4.80	53.98	3.98	92.05
4.90	53.82	3.82	92.35
5.00	53.68	3.68	92.65

2-2. Numerical integration vs. continuous system simulation

Some of you may have recognized that all we have done for studying the dynamics of the reaction rates in Sec. 2-1 is to use Euler formula to perform numerical integration of three simultaneous differential equations, Eqs

(2-1)–(2-3). What then, you might ask, is the difference between an ordinary numerical integration and a continuous system simulation? What is it that makes one study of a dynamic system a computation and another study, a simulation? Although the dividing line between simulation and ordinary computation often becomes quite thin, there are two distinctive features of simulation : (i) In simulation, we always keep track of the state of the system explicitly. The outcome of each step (of numerical calculations) in a simulation experiment can be interpreted directly as the state of the system at some point in time. Simulation, essentially, consists of constructing a state history of the system—a succession of explicit state descriptions at each instant as we move forward in time. Thus there is a one-to-one correspondence between what the computer does and what takes place in the system. In a numerical solution of equations no such correspondence is preserved. Usually in pure computations shortcuts are taken, parameters are lumped and mathematical equations manipulated before the computer program is developed. These destroy the one-to-one correspondence between the computer steps and the original system from which the equations were derived. Consequently the output data have to be interpreted in the light of earlier manipulations before conclusions can be drawn about the system. (ii) Secondly, there is also a difference of attitude. In case of a pure numerical calculation we only see the given set of differential equations as a mathematical object and proceed to integrate them. In simulation we look upon the equations as one of the steps in the entire process. We know the real-life system, and we are aware of the approximations in the model that is being simulated. Finally, by looking at the output data (which directly represent the dynamics of the system) we are also prepared to modify the model, if necessary.

2-3. Selection of an integration formula

As we observed earlier, continuous dynamic systems are generally represented by means of differential equations. Numerical solution of these differential equations involves some integrating procedure. Many different integration formulas are available for this purpose, and the selection of an appropriate formula for integration is the most crucial decision to be made in a continuous system simulation. In Sec. 2-1 we used the simplest possible integration formula, which is known as Euler formula. There are much more efficient ways of performing numerical integration which do not rely simply on the last known values of the variables. Instead, some of them use several previous values to predict the rate at which the variables are changing and also in some formulas the integration step size Δt is adjusted to match the rate at which the variables are changing. As a matter of fact, the simple Euler formula employed in Sec. 2-1 is rarely used in practice because of the rapid accumulation of errors (to be discussed shortly). There are improved versions of the Euler formula available, such as, Euler-

Richardson formula, Euler-Gauss formula. Some of the integration formulas commonly used in simulation are: Simpson's rule ($\frac{1}{3}$ rule, 3/8 rule); trapezoidal rule; the Runge-Kutta family of formulas (second, third or fourth order); predictor-corrector methods; and so on. Many of these are available as standard subroutines. There is no integration method which is the best for simulations. One has to consider several factors when choosing an integration formula. Accuracy and the speed of computation are the most important considerations. Other factors are self-starting ability, solution stability, presence or absence of discontinuity, and ease with which error can be estimated. Any detailed discussion of these is beyond the scope of this book. The following is a very brief excursion.

Errors: There are basically two types of computation errors that occur at each step of integration: (i) *Round off* errors are introduced because of the limited size of the computer word. Every number has to be represented within this size. For example, suppose a number obtained as a product of two 4-digit numbers

$$.2102 \times .8534 = .17938468$$

has to be accommodated within 4-digits. It is therefore rounded off to .1794, introducing thereby an error. Some computer systems (compilers such as ALPS, BASIC and some versions of FORTRAN) have automatic rounding built into them, but many systems do not round off; they simply *chop off*; which is even worse. Thus the product in the foregoing example would end up being 0.1793. (ii) *Truncation errors* are caused when an infinite mathematical series is approximated by a finite number of terms. In continuous system simulation truncation errors arise mainly due to the inadequacy of an integration formula being used. For example, when Euler integration formula is used we are using only the first two terms of the Taylor series (which is infinite).

$$f(t+\Delta t) = f(t) + \frac{df(t)}{dt} \cdot \Delta t + \frac{d^2f(t)}{dt^2} \cdot \frac{(\Delta t)^2}{2!} + \frac{d^3f(t)}{dt^3} \frac{(\Delta t)^3}{3!} + \cdots$$

when Δt is sufficiently small the truncation error in Euler formula can be approximated with

$$\frac{d^2f(t)}{dt^2} \cdot \frac{(\Delta t)^2}{2}$$

The second derivative can be approximated to

$$\frac{d^2f(t)}{dt^2} = \frac{1}{\Delta t}\left[\frac{df(t+\Delta t)}{dt} - \frac{df(t)}{dt}\right]$$

On dividing the error term with value of $f(t)$ to get a relative error and on getting rid of the sign, we get

$$\text{Relative error} = Er = \left[\frac{df(t+\Delta t)}{dt} - \frac{df(t)}{dt}\right]\frac{\Delta t}{2f(t)} \qquad (2\text{-}5)$$

In each individual step *Er* may be small but when accumulated over hundreds of consecutive steps in an integration the truncation error could make the simulation results meaningless.

Integration step: Choice of the step size Δt of integration is another very important decision. The smaller the integration step the smaller is the error. The relative integration error in using Euler formula, for example, is given by Eq. (2-5). Clearly *Er* can be made as small as we please by making Δt sufficiently small. But the number of computation steps and therefore the computation time will increase inversely in proportion to Δt. More steps would also accumulate an increased amount of round off errors. A compromise has to be made between the conflicting requirements of speed and accuracy. For a given accuracy, it is often possible to arrive at an optimal combination of an integration formula and the step size.

Sometimes, when the error varies widely, it is advisable to use a *varying-step integration* formula. That is, the error is evaluated at each integration step and the step size Δt for the next step automatically adjusted accordingly.

It often happens that the time intervals for which we require the output of integration are much larger than the integration step size. In that case the program is so written that values of the variables are saved once, say, every 100 steps of integration. This will be illustrated in Sec. 2-5.

2-4. Runge-Kutta integration formulas

In Euler formula (used in Sec. 2-1) to estimate the value of the variable at time $(t + \Delta t)$ we used its slope at time t,

$$y(t + \Delta t) = y(t) + \frac{dy(t)}{dt} \cdot \Delta t.$$

In other words, the value of y at $(i + 1)$th instant is estimated in terms of its value and slope at the ith instant. That is,

$$y_{i+1} = y_i + \dot{y} \cdot \Delta t$$

where $\dot{y}$ is the usual short-hand notation for the first derivative of y.

As mentioned earlier, Eular formula is the simplest and crudest method of numerical integration. It is not very efficient because it requires a very small step-size Δt for reasonable accuracy.

A much more refined, accurate, and commonly used method is the Runge-Kutta method. It is a method designed to approximate the Taylor series without our having to evaluate the higher order derivatives. The main idea is to compute the first derivate $f(t, y)$ of y at several carefully chosen points in the interval $(t, t + \Delta t)$, and to combine these values in such a way as to get good accuracy in the computed increment $y_{i+1} - y_i$. Based on this strategy, a family of formulas have been derived, known as Runge-Kutta formulas. The best amongst these, and by far the most popularly used

```
      READ, A11, A21, A22, B1, B2, N1, N2, HS, HL
      T = 0.
      V1 = 0.
      V2 = 0.
      H = HS
      N = N1 + N2
      DO 130 I = 1, N
      IF(I.GT.N1) GO TO 120
C     EVALUATE RUNGE-KUTTA TERMS—4 FOR EACH EQUATION
100   U11 = H*((A11* V1) + E1)
      U12 = H*((A21* V1) + (A22*V2) + B2)
      U21 = H*(A11*(V1 + .5*U11) + B1)
      U22 = H*(A21*(V1 + .5*U11) + A22*(V2 + .5*U12) + B2)
      U31 = H*(A11*(V1 + .5*U21) + B1)
      U32 = H*(A21*(V1 + .5*U21) + A22*(V2 + .5*U22) + B2)
      U41 = H*(A11*(V1 + U31) + B1)
      U42 = H*(A21*(V1 + U31) + A22*(V2 + U32) + B2)
C     EVALUATE VOLTAGES
      V1 = V1+(U11+2.*U21+2.*U31+U41)/6.
      V2 = V2 + (U12 + 2.*U22 + 2.*U32 + U42)/6.
      T = T + H
110   PRINT, I, T, V1, V2
      GO TO 130
C     STEP-SIZE INCREASES
120   H = HL
      GO TO 100
130   CONTINUE
      STOP
      END
```

2-5. Simulation of a servo system

A very important application of continuous system simulation is in design and analysis of control systems. Let us study the behaviour of a second-order nonlinear feedback system represented by the following block diagram.

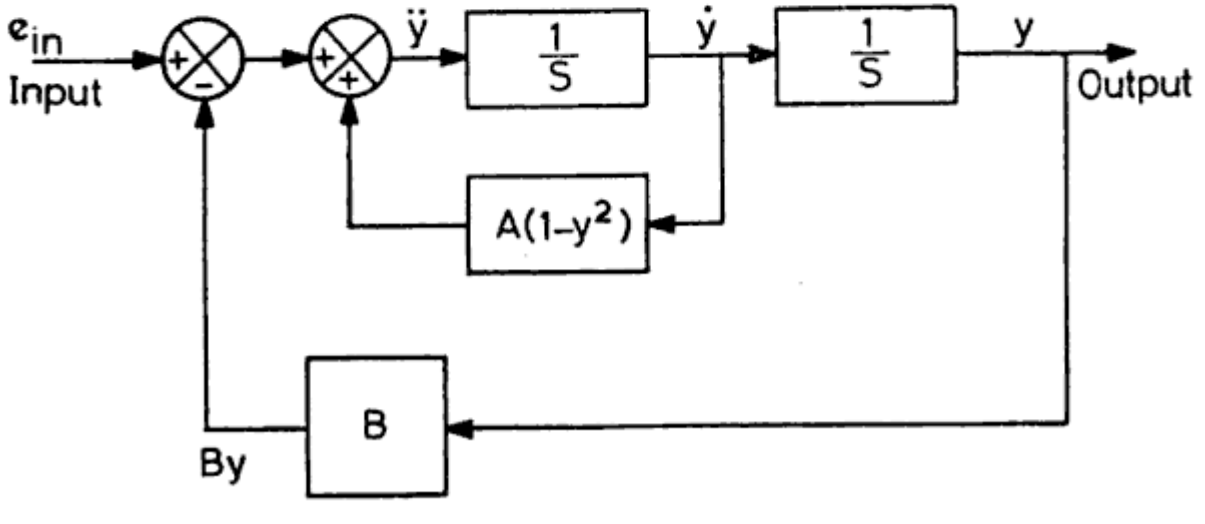


Fig. 2-3: A nonlinear second-order servo system.

(Those of you not familiar with the control theory symbols may ignore the diagram and simply start from the differential equation.)

This block diagram represents many natural as well as man-made servo systems. Some examples are: beating of the heart, periodic opening and closing of flowers in response to the sunlight, rate of variation of prices, squeaking of door with rusty hinges, dripping of a leaky tap, a neon-lamp oscillator, to name a few.

The system of Fig. 2-3 can also be described by the following differential equation

$$\ddot{y} = A(1 - y^2)\dot{y} - By + e_{in}$$

where A and B are positive constants.

In the case of zero input signal, the equation becomes

$$\ddot{y} = A(1 - y^2)\dot{y} - By \qquad \ldots(2\text{-}9)$$

This is the well-known Van der Pol non-linear equation. (It can be seen that when the amplitude y is small, the damping term $A(1 - y^2)$ is negative, but when y becomes large the damping becomes positive. Thus small-amplitude oscillations will tend to build up, while large-amplitude oscillations will be damped out.)

Let us simulate this system. The second-order differential equation can be written as a set of two simultaneous equations of first order. We replace Eq. (2-9) with (using the variable y_1 in place of y)

$$\dot{y}_1 = y_2$$
$$\dot{y}_2 = A(1 - y_1^2)y_2 - By_1$$

To be more specific let constants $A = 0.1$, $B = 1.0$ and let the initial conditions be

$$y_1(0) = 1.0$$

and

$$y_2(0) = 0.$$

Our equations therefore become

$$\dot{y}_1 = y_2$$

and

$$\dot{y}_2 = 0.1(1 - y_1^2)y_2 - y_1 \qquad \ldots(2\text{-}10)$$

An instantaneous description of the state of the system is given by the outputs of the two integrators, i.e., by variables y_1 and y_2. We will use, once again, the fourth-order Runge-Kutta method to obtain the values of y_1 and y_2 as a function of time. We will choose the step-size $\Delta t = H = .001$ second and simulate the system for 5 seconds. Thus the number of steps will be $N = 5{,}000$. This is too large a number of outputs to be plotted or examined. We will, therefore, print out the values of y_1 and y_2 only once every 100 integration steps. This can be implemented by keeping a counter K which is decremented by 1 for each integration step. Every time K equals zero,

jected demand. This system (called a simple run-of-river storage demand system) is represented symbolically in Fig. 2-4.

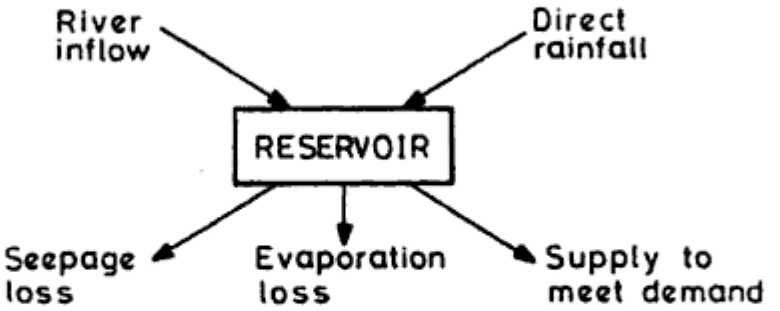


Fig. 2-4: A simple run-of-river storage demand system.

The amount of seepage loss is not a constant but depends on the volume of the water stored. We have been given a curve (converted into a table) showing the seepage loss as a function of volume for the proposed reservoir. Likewise, the evaporation loss depends on the area of the exposed surface and the coefficient of evaporation. We are given another curve showing the surface area as a function of volume as well as the seasonal variation of the coefficient of evaporation. Therefore, for a given volume of water in the reservoir at a particular time of the year we can calculate the two losses.

In reality no reasonable finite-sized reservoir can provide an absolute guarantee of meeting the demand 100% of the time because the river inflow, the rainfall, the losses, the demand are all random variables. To build such a large dam which will never fail (to meet the demand) through its entire life will generally be uneconomical. Therefore, in practice one determines the reservoir size which will meet the demand with a specified risk of failure (of water shortage). For example, a 2% failure means that once in 50 years the reservoir would become empty before meeting the demand for water. The objective of the study is to determine the size of the reservoir with a specified risk of failure.

There is a single state variable in this system, namely, the volume of water in the reservoir. Since the volume varies continuously with time, we are dealing with a continuous system. It is reasonable to take one month as the basic time interval for the simulation study. Thus, for example, if we wish to simulate the system for 100 years, the simulation run length will be 1200. The simulation will be repeated assuming several different capacities of the reservoir. The output will be in series of ranked shortages for each capacity.

The basic procedure, to be repeated for each time step, may be expressed in terms of the following steps:

(1) For the current month *M* of the current year *IY* determine the total amount of river inflow and the total rainfall directly over the reservoir. Let the sum of two inputs be denoted by VIN (= RAIN + RFLOW).

(2) Add the input volume VIN to the volume left over in the reservoir at the end of the last month, VOL $(m-1)$. This gives us the gross volume, GROSSV = VIN + VOL $(m-1)$.

(3) On the basis of the last month's volume VOL $(m-1)$ calculate this month's seepage and evaporation losses and add them as total loss TLOSS = SEEP + EVAP.

(4) From the demand curve (stored as a table in the computer memory) determine the demand of water for the current month DEM.

(5) If the TLOSS $\geqslant$ GROSSV, then the reservoir runs dry without supplying any water and therefore shortage, SHORT = DEM. The volume of water left at the end of the current month VOL $(m) = 0$. Spillage SPILL = 0 and go to Step 8; else if TLOSS $<$ GROSSV then the net water volume available to satisfy the demand is VNET = GROSSV − TLOSS.

(6) If DEM $\geqslant$ VNET the reservoir runs dry and the shortage is given by SHORT = DEM − VNET, and SPILL = 0. Go to Step 8; else if DEM $<$ VNET, then the difference DIFF = VNET − DEM is the water left over.

(7) If this leftover water exceeds the capacity CAP of the reservoir there will be a spill over, i.e., if DIFF $>$ CAP then SPILL = DIFF − CAP and VOL (m) = CAP; else if DIFF $\leqslant$ CAP then SPILL = 0 and VOL (m) = DIFF.

(8) Print out SPILL and SHORT for this month, and move to the next month. If the period exceeds the intended simulation length stop, else go to Step 1.

The following format-free FORTRAN program implements these steps:

```
READ, N, VOL, CAP
IY = 1
M = 1
DO 30 IY = 1, N
DO 30 M = 1, 12
SPILL = 0.
SHORT = 0.
CALL RIVFLO (IY,M,RFLOW)
CALL RAINF (IY,M,RAIN)
VIN = RFLOW + RAIN
GROSSV = VOL + VIN
CALL SEPEJ (VOL,SEEP)
CALL EVPRSN (M,VOL, EVAP)
TLOSS = SEEP + EVAP
CALL DEMAND (IY,M,DEM)
VNET = GROSSV − TLOSS
IF (VNET .LE. O.) VNET = 0.
DIFF = VNET − DEM
IF (DIFF .GE. O.) GO TO 10
SHORT = −DIFF
```

```
      VOL = 0.
      GO TO 20
10    VOL = DIFF
      IF (CAP .GT. DIFF) GO TO 20
      SPILL = DIFF - CAP
      VOL = CAP
20    PRINT, FLOW, RAIN, TLOSS, SHORT, SPILL, VOL
30    CONTINUE
      STOP
      END
```

Subroutines

The foregoing program contains five subroutines requiring data about the riverflow, rainfall and the demand as a function of time; the seepage loss as a function of water stored in the reservoir; and the evaporation loss as a function of the volume (and hence the exposed surface) and the particular month of the year. The long sequence of data for river inflow and the rainfall could either be obtained from historical records or generated using suitable pseudorandom number generators. Similarly we can design the other subroutines, from an intimate knowledge of the system. As an example, we will write down the subroutine EVPRSN, for computing the evaporation loss.

Let the unit of measuring the volume be a million cubic meters. Suppose the highest possible dam at this site will create a reservoir of capacity 1000 units. Also suppose that we have a curve that gives the exposed surface area as a function of volume from 0 to 1000 units. Let the x-axis be divided into 100 equispaced ranges; and the data be stored in the form of a table (SURTBL) with 100 columns giving the surface area at volume 10, 20, ..., 1000 units. The surface area SAREA for any intermediate value of VOL can be computed using an appropriate interpolation formula. Let us assume that a linear interpolation will suffice. We are also given 12 values for the coefficient of evaporation COEF—one for each month of the year. Then the following subroutine will yield the evaporation loss.

```
      SUBROUTINE EVPRSN (M, VOL, EVAP)
      REAL SURTBL (100), COEF (12)
      DATA SURTBL /..., ..., ... /
      DATA COEF /..., ..., ... /
      IVOL = VOL/10.
      RVOL = IVOL
      FRAC = VOL/10. - RVOL
      SAREA = SURTBL (IVOL) + FRAC* (SURTBL(IVOL+1)-SURTBL(IVOL))
      EVAP = SAREA * COEF (M)
      RETURN
      END
```

Note that the third statement requires 100 values and the fourth statement requires 12 values. Other subroutines can be written down similarly.

Output: The output will be a series of monthly shortages and spills.

The shortages could be combined into total annual shortages. These annual shortages can be ranked according to the amount of shortage involved. From this ranked series of shortages for each capacity of the reservoir we would determine the acceptable reservoir size.

In the foregoing model no distinction was made regarding how the shortage is distributed month-wise within a particular year. For example, the total failure to meet any demand for one month may be more serious than a 10 per cent shortage for ten consecutive months. Such refinement can be easily incorporated by a procedure of assigning weights to different types of failures within a particular year.

There is another refinement which should be made in our computation. The losses for the entire current month were computed on the basis of the volume in storage at the end of last month. If a large change in volume takes place between two consecutive time steps, this would lead to errors. This could be corrected by first finding a temporary value of VOL (m) on the basis of given VOL ($m-1$) (as usual) and then recalculating the losses on the basis of the average value

$$\frac{\text{VOL}(m) + \text{VOL}(m-1)}{2}$$

of the two volumes.

2-7. Analog vs. digital simulation

Block-diagram programming system: Let us once again consider the continuous, dynamic nonlinear system described by the van der Pol equation in Sec. 2-5. The simulation program for that system can be expressed by means of the following block diagram.

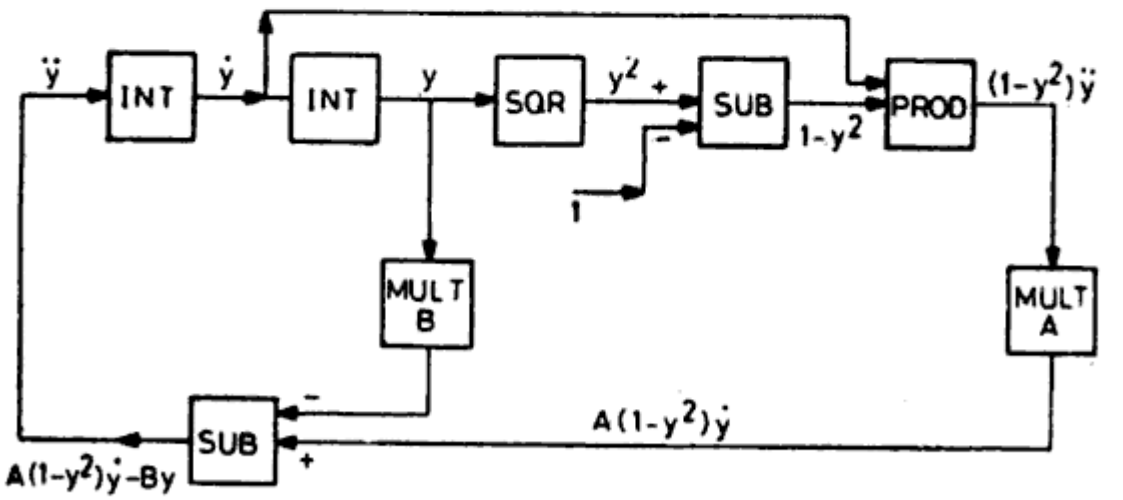


Fig. 2-5: Block diagram for $\ddot{y} = A(1 - y^2)\dot{y} - By$.

Each of the blocks performs a mathematical operation. Block INT performs integration, SQR squares, PROD takes the product, MULT multiplies with a constant, and SUB subtracts. The blocks may be looked upon as

subroutines. The interconnection of the blocks is self-explanatory. Note that the variables $\dot{y}$, $\dot{y}$, y, y^2, etc., vary with time and must be updated every H (i.e., Δt) seconds. The complete sequence of operations (by these blocks) is, therefore, repeated every H seconds if we use Euler integration (or four times every H seconds if we employ the 4th order Runge-Kutta integration).

Block-diagram oriented languages: Any continuous system (requiring numerical solution of differential equations), however complex, can be simulated by suitably 'patching' together a number of blocks such as these. Different systems will vary only in the number of blocks required and in their interconnections. Since the same few types of blocks are used again and again for simulation of all continuous systems, it would be very useful to provide a package of standard subroutines to perform the operations of these blocks. It would be even more convenient to have a special-purpose programming language which allowed direct implementation of block diagrams such as the one in Fig. 2-5. Development in the simulation area has taken place precisely in this direction. A large number of such continuous system simulation languages have been designed and implemented. One or more of these languages are available at almost every computing centre. These languages will be discussed in greater detail in Chapter 9.

Analog implementation: Theoretically speaking, the means by which each of the blocks is actually implemented is immaterial, as long as it does its assigned task so that the entire set-up gives us the state of the system as a function of time. These block functions may all be performed by one digital computer or by, say, different microprocessors—each specially programmed to do a specific job.

A block diagram, such as in Fig. 2-5, can also be simulated on an (electronic) analog computer, which consists of special-purpose elements such as integrators, multipliers, adders, function generation, and nonlinear elements. These elements are interconnected to imitate the system under study. The variables and their derivatives are represented by means of voltages (which vary according to the equations governing the given system). The voltages can be monitored continuously at suitable points in the set-up to get the state of the system as a function of time.

Historically analog computers came into being years before digital computers did, and have played a major role in simulation of continuous dynamic systems. Although analog computers are giving way to digital computers at an increasing pace, they are still in occasional use. The following are some of the disadvantages of analog computers.

1. **Inadequate accuracy:** In general, the result from a digital simulation is more accurate than that from an analog simulation. The accuracy of analog simulation depends on the accuracy of the components being used, which can vary from .01% to 2%. When accuracy required of components is more than 0.1%, the cost of components increases rapidly. An accuracy

of about 1% for a simulation of system with modest complexity is considered good. This limited accuracy cannot meet the need in simulating systems like missiles and space vehicles.

2. **Scaling needed:** The magnitudes of dependant variables are represented in an analog computer by voltages. For a 10-volt analog computer the maximum range of voltages is from -10 to $+10$. All program variables must be scaled carefully so that none exceeds the maximum voltage. If a variable exceeds the maximum voltage, then the corresponding amplifier becomes saturated and results become inaccurate. This magnitude scaling is a tedious task in an analog simulation, because there are usually many variables and their maximum values are not known in advance.

In a digital computer with floating point arithmetic the magnitude scaling problem does not arise, because the quantities that can be represented on a digital computer have a very large range. For the IBM 360, for example, the range is from -10^{75} to $+10^{75}$. It has a precision of 7 decimal digits. Therefore, normally no magnitude scaling is needed.

In addition to magnitude scaling, analog computers also require time scaling. The maximum computing time for an analog computer is limited because of drifts affecting the accuracy of the results. The time scale factor, i.e., the ratio of computing-time to problem-time has to be determined. (If this ratio is 1, simulation is *real time*.) Since there is only one time scale factor, the task of time scaling is much easier. In digital computer simulation time scaling is generally not needed.

3. **Hardware set up necessary:** In analog simulation input constants and initial conditions are incorporated by setting up voltages and potentiometers. Then the various elements (amplifiers, multipliers, voltmeters, etc.) have to be connected on a patch board. Such setting up is not needed in a digital simulation.

A simulation program on a digital computer can be easily stored for reuse. Availability of various mathematical functions on a digital computer is another advantage. Since in a digital simulation no set up is required nor any calibration and accuracy test is needed, rapid switching from one simulation to another can be made. This results in a better machine utilization.

The two main advantages of analog simulation have been higher speed of solution (necessary for certain applications) and direct access to and immediate display of the computed results. Both these advantages are vanishing in the presence of superspeed digital computers available in a multiprogramming environment with on-line CRT displays.

We will not discuss analog simulation any further. The reader may refer to any of several excellent textbooks available on analog computers.

2-8. Remarks and references

Electronic analog computers became available soon after World War II

and they were found to be extremely valuable in analysis and design of numerous engineering systems. Analog computers were used for simulating various continuous dynamic systems which were too difficult to lend themselves to analytic studies. About ten years later, in the mid 1950's, when digital computers became commercially available, their advantages (such as greater accuracy, no need of scaling, greater flexibility, etc., as discussed in Sec. 2-7) over analog computers in simulating complex continuous systems were recognized, and they began to be used for this purpose.

A general purpose digital computer was required to have certain minimum hardware facilities if it was to be used successfully for simulating large continuous systems; such as hardware floating point arithmetic, long word length to reduce round-off errors, and a reasonably large random-access memory for handling a large amount of intermediate data. As the hardware (and software) of the digital computers became increasingly better and cheaper during the last twenty years, the digital computer began to be used more and more for simulating continuous dynamic systems.

The most comprehensive and an early textbook dealing with the simulation of continuous systems on a digital computer is

CHU, Y., *Digital Simulation of Continuous Systems*, McGraw-Hill, New York, 1969.

Some more recent books on this topic are

ORD-SMITH, R. J. and J. STEPHENSON, *Computer Simulation of Continuous Systems*, Cambridge University Press, Cambridge, U.K., 1975.

STEPHENSON, R. E., *Computer Simulation for Engineers*, Harcourt, Brace, Jovanovich, New York, 1971.

SHAH, M. J., *Engineering Simulation Using Small Scientific Computers*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

A journal devoted entirely to computer simulation was started in 1963 and is called SIMULATION. A good collection of articles, dealing exclusively with simulation of continuous systems, from this journal can be found in

McLEOD, J. (ed.)., *SIMULATION : The dynamic Modeling of Ideas and Systems with Computers*, McGraw-Hill, New York, 1968.

As pointed out in the beginning of this chapter, continuous system simulation is encountered mostly in engineering and physical sciences. For example, simulation has played a crucial role in the development of modern aerospace industry and chemical-process industries. A few applications in these areas have already been illustrated, namely, a chemical reactor, an aircraft pursuit system, an electrical network, a feedback control system, and a water reservoir system. Some additional applications are pointed out in the form of exercises. These are trajectory calculations, the multibody problem, vibration and shock absorbers, and econometric models. This list of continuous dynamic systems where simulation has been applied pro-

fitably is indeed very large. The examples we chose in this chapter were simplified to keep the problems manageable. A real problem is more involved computationally (but not conceptually).

The examples of simulation applied to various engineering and other disciplines can best be found in journals, technical reports, and textbooks in respective fields. For instance, the following provides a good reference for simulation of water resource systems (a simple example of which was given in Sec. 2-6).

HUFSCHMIDT, M. and M. B. FIERING, *Simulation Techniques for the Design of Water Resources Systems*, Harvard University Press, Cambridge, Mass., 1966.

Likewise an excellent treatment on digital simulation of chemical engineering plants can be found in Chapter 5 of

LUYBEN, W. L., *Process Modeling, Simulation and Control for Chemical Engineers*, McGraw-Hill, New York, 1973.

An integrator is the heart of an analog simulation. Likewise, the selection of an appropriate integration subroutine (from amongst several available on your computer system or at least in the literature) is of central importance to the program for simulation of a continuous system. There are a number of excellent textbooks on numerical analysis that discuss various integration formulas and their merits from the viewpoint of speed, accuracy, etc. A few of these are:

ACTON, F. S., *Numerical Methods that Work*, Harper and Row, New York, 1970.

CONTE, S. D. and C. De BOOR, *Elementary Numerical Analysis : An Algorithmic Approach*, (2nd Ed.), McGraw-Hill, New York, 1968.

DAHLQUIST G. and A. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

RALSTON, A., *A First Course in Numerical Analysis*, McGraw-Hill, New York, 1965.

RAJARAMAN, V., *Computer Oriented Numerical Methods*. Prentice-Hall of India, New Delhi, 1971.

It is the technology of electronic devices which has tilted the balance overwhelmingly in favour of digital simulation for continuous systems, although analog simulation appears more natural. In view of the advancing technology, particularly of microprocessors, the field is far from having been stabilized. An interesting article on the effect of electronic technology on continuous system simulation is

AUS, H. M. and G. A. KORN, *The Future of Continuous-System Simulation*. Proc. AFIPS/FJCSS, 1969.

2-9. Exercises

I. TRAJECTORY SIMULATION

Simulation has been used in computing trajectories of various types of rockets, bullets, space vehicles, etc. Because of the drag (air-resistance) the

equations of motion get sufficiently complicated for a closed form mathematical solution. In the case of a rocket the mass itself does not remain constant but reduces as the fuel is used up.

2-1. Given a cannon that fires spherical balls of mass m, you are asked to produce a range table, i.e., a table that gives ranges for various values of muzzle velocity and gun elevation (firing angle). You are also asked to investigate the sensitivity of range to small changes in muzzle velocity and gun elevation. For simplicity assume that the drag is proportional to the square of the instantaneous velocity of the cannon ball, and it is purely along the direction of flight. Making use of the following four equations, write a computer program to produce a range table:

$$m\frac{dv}{dt} + mg\sin\theta + cv^2 = 0$$

$$mv\frac{d\theta}{dt} + mg\cos\theta = 0$$

$$\frac{dx}{dt} = v\cos\theta$$

$$\frac{dy}{dt} = v\cos\theta$$

Constant c is the drag coefficient for the cannon ball. Variables v and θ are the instantaneous velocity and angle of elevation of the cannon ball, and x, y are the coordinates of its instantaneous position. For various values of the starting conditions, i,e., muzzle velocity v_0, elevation θ_0, and $x_0 = y_0 = 0$, the program should yield the range.

2-2. A missile is fired vertically up. It has a rocket motor that produces a certain constant upward thrust F for as long as the fuel-burning motor is on. The motor burns the fuel at a constant rate b. Assume that the missile does not go so high as to vary the gravity constant g nor does the drag coefficient vary during the flight. Set up the equation of the vertical motion. Write a program which gives the vertical distance y at any time t.

2-3. Modify the simulation program in Exercise 2.2 in order to make it valid for higher altitudes. That is, take into account the reduction in the air resistance and change in the value of the gravitational constant g as the missile rises higher.

2-4. Suppose a 50-kg. heat-seeking missile is fired at a jet bomber flying at a constant speed of 1,200 Km/hour. The missile has a rocket motor that produces a thrust of 5000 Kg. for a period of 4 minutes. Simulate the missile system to test its effectiveness and to estimate how close the target should be before a missile is fired. Assume the drag coefficient $c = .04$ Kg/Km^3 and the weight of liquid fuel is 20 Kg. (which is burned in 4 minutes at a constant rate).

2-5. Assuming that the drag force is proportional to the square of the

2-12. The gross national product $GNP(t)$ during time t can be expressed as a sum

$$GNP(t) = GS(t) + IN(t) + C(t) \qquad \ldots(2\text{-}12)$$

Where $GS(t)$ is the government spending during time t, $IN(t)$ is the investment during t, and $C(t)$ is the consumption. Two of these three quantities can themselves be expressed as

$$IN(t) = m + c\,[GNP(t-1) - GNP(t-2)] \qquad \ldots(2\text{-}13)$$

$$C(t) = a + b.GNP(t-1) + d.GNP(t-2) \qquad \ldots(2\text{-}14)$$

where a, b, c, m, d are constants which have been determined for this system. At time t the previous year's GNP is known. Simulate the system for predicting the current year's GNP, given that the government spending has been announced.

3

Discrete System Simulation

In the last chapter our objects of simulation were continuous systems—those systems in which the state changed smoothly with time. In this chapter (and in all subsequent chapters) we will deal with discrete systems—systems in which the changes are discontinuous. Each change in the state of the system is called an *event*. For example, arrival or departure of a customer in a queue is an event. Likewise, sale of an item from the stock or arrival of an order to replenish the stock is an event in an inventory system. Arrival of a car at an intersection is an event if we are simulating street traffic. Therefore, the simulation of a discrete system is often referred to as *discrete-event simulation.*

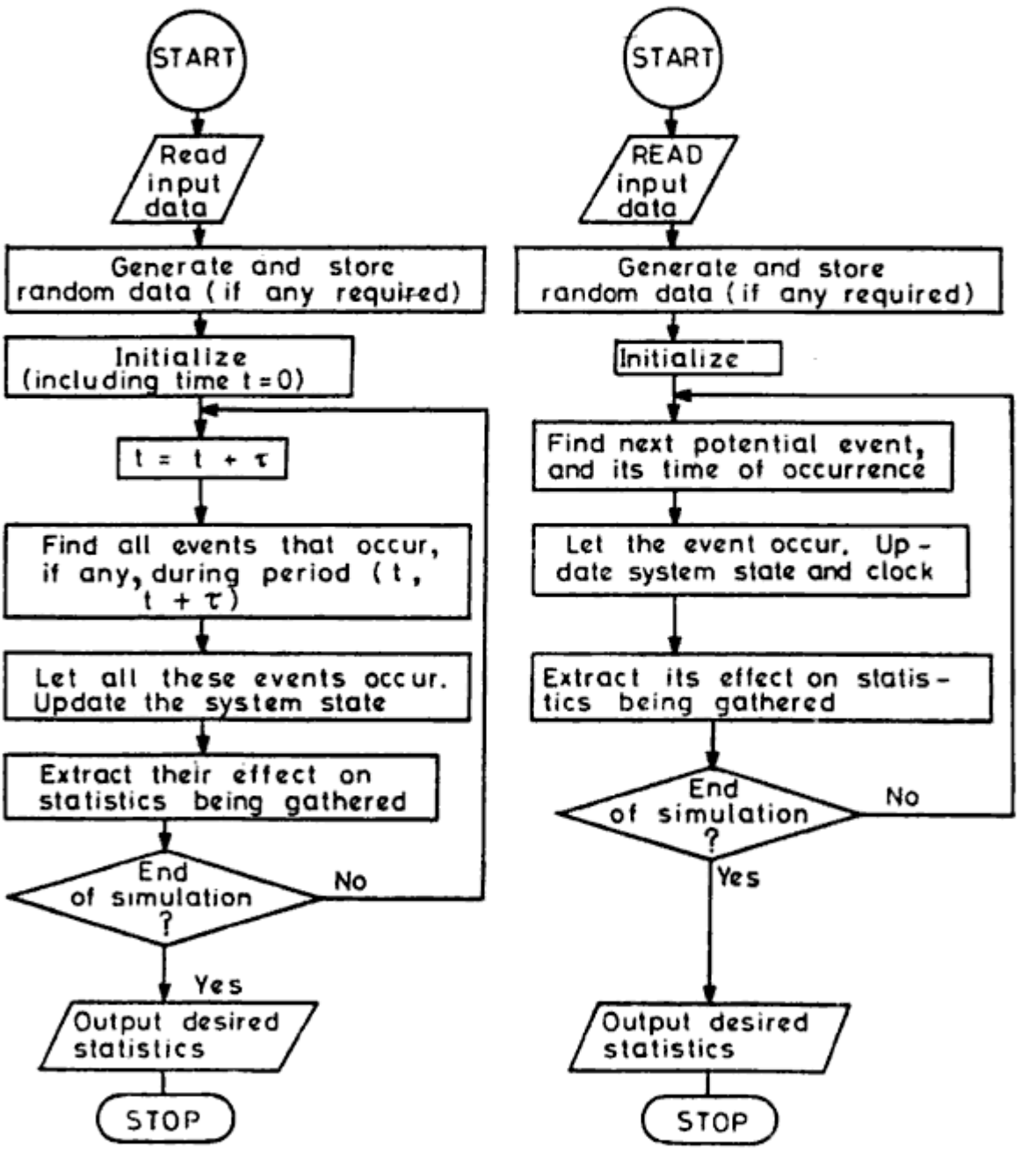
Discrete-event simulation is commonly used by operations research workers to study large, complex systems which do not lend themselves to a conventional analytic approach. The very simple inventory problem simulated in Sec. 1-3 is an example. Some other examples are the study of sea and airports, steel melting shops, telephone exchanges, production line, stock of goods, scheduling of projects, to name a few. Discrete system simulation is more diverse and has less of a theory than continuous system simulation. There are no overall sets of equations to be solved in discrete-event simulation.

3-1. Fixed time-step vs. event-to-event model

In simulating any dynamic system—continuous or discrete—there must be a mechanism for the flow of time. For we must advance time, keep track of the total elapsed time, determine the state of the system at the new point in time, and terminate the simulation when the total elapsed time equals or exceeds the simulation period. For continuous systems, in Chapter 2, we advanced time in small increments of Δt for as long as was needed. In simulation of discrete systems, there are two fundamentally different models for moving a system through time : the *fixed time-step* model and the *event-to-event* (or *next event*) model. In a fixed time-step model a "timer" or "clock" is simulated by the computer. This clock is up-dated by a fixed time interval τ, and the system is examined to see if any event has taken place during this time interval (minutes, hours, days, whatever). All events that take place during this period are treated as if they occurred simultaneously at the tail end of this interval. In a next-event simulation model the

computer advances time to the occurrence of the next event. It shifts from event to event. The system state does not change in between. Only those points in time are kept track of when something of interest happens to the system.

Flowcharts for these two methods of simulating a discrete system in their most general forms are shown in Fig. 3-1.



(a) Fixed time-step simulation (b) Next-event simulation

Fig. 3-1: Flowcharts for discrete system simulation.

output may not be predictable analytically even for very simple inputs.

Validating existing systems: When the simulated system exists in real life, then the most obvious and the best approach is to use the real world inputs to the model and compare its output with that of the real system. This process of validation is straightforward enough in principle but may present some difficulties when carried out. Firstly, it may not always be easy to obtain input and output data from a real-life system without disturbing it. Secondly, even if we could get actual input output of an existing model it would not generally be for very long periods. Since the data are usually probabilistic, we know from earlier sections in the chapter, that for small lengths of simulation runs the variability of the model output would be large. Therefore designing tests that work with small samples is difficult. What usually is done is to simulate the model several times (replicate) with different sequences of random numbers and obtain the range of variation amongst these. Then, if the model is valid, the real output should lie somewhere in the middle of the range of model output. The third problem is to establish that the model output and the real-system outputs are 'practically' from the same population.

If the outputs to be compared are sample means (e.g., average queue length, waiting time, idle time), one could use any of a number of statistical tests (called 'goodness of fit' tests) available to measure the discrepancy between the two outputs (i.e., model output and the real-system output). One such test is the chi-square test, which has already been described in Chapter 3. Others are Kolmogorov-Smirnov test, Cramer-von Mises test, and Moments test. One could also use *hypothesis testing* to determine if there is any significant difference between, say, the averages of the independent set of observations. One illustration of test of means is given in Exercise 7-7.

In most simulation experiments the output that is obtained is a time series (such as the queue lengths at various times, or stock level at different times), which are autocorrelated. How to compare two autocorrelated time series and test them for equivalence? Sepectral analysis is one of the methods that has been suggested in the literature.

Validating first-time model: If a model is intended to describe a proposed or hypothetical system (which does not exist at present or did not exist in the past), then the task of validation is even more difficult. There are no historical data available to compare its performance with. Since hypothetical systems are, by their very nature, based upon assumptions, it is the validity of these assumptions the simulation model is dependent on. A number of guidelines for testing validity of such system have been found useful. These are—

1. **Subsystem validity:** A model itself may not have any existing system to compare it with, but it may consist of known subsystems each of hose validity can be tested separately.

performance that are of interest to us and data collection for determining values of the system parameters.

3. Flowcharting—to show the logical steps to be performed by the computer in simulating the mathematical model obtained in Step 2 above.
4. Choice of a programming language. There are usually two options available at this point. The first option is to use a high-level general-purpose language such as FORTRAN, (or ALGOL, PL/1, BASIC, etc.) as we have done throughout this book. The second option is to use a special-purpose simulation language, such as, GPSS, (or SIMSCRIPT, SIMULA, etc.). Both of these approaches are extensively used in simulation studies. Use of a simulation language can save time and drudgery for the programmer if the system being simulated is very complex. On the negative side, he would have to learn a new language and the program written in a special purpose language would usually eat up more computer time and memory.
5. Program writing. While converting the flowchart into a running program (in the language chosen in Step 4) special attention should be paid to modularization. The simulator should consist of a number of natural and simple modules (subprograms) which can be tested independently and allow changes in the model conveniently. Intermediate print statements should also be included for diagnosis in validating the debugging stages. These statements may be removed before running the validated program. The format of the final output should be designed for easy interpretation.
6. Validation of the model.
7. Design of experiments—including removal of unwanted transients, determination of run lengths, initial conditions, and number of replications.
8. Execution of simulation run, analysis and interpretation of results.

Some of these steps have to be performed iteratively till acceptable results are obtained.

7-6. Remarks and references

The problems associated with the designing of simulation experiments have been classified into two types by Conway in

CONWAY, R. W., "Some Tactical Problems in Digital Simulation," *Management Science*, Vol. 10, No. 1, October 1963, pp. 47–61.

The types are (a) *strategic planning problems*, and (b) *tactical problems*. The strategic planning is concerned with how to lay out a set of simulation experiments so that the desired information is obtained. The tactical planning involves the conducting of each experiment most efficiently. It includes the decisions on run length, number of replications, starting con-

8

Simulation Languages

As we saw in previous chapters, computer simulation is essentially an experimental technique, used for studying a wide variety of systems. The purpose of the experiment is to observe the behaviour of a given system (actually a specific model of the system) within a given environment which would eventually provide a basis for understanding the system and making decisions. The abstract model of the system being simulated takes the form of a computer program, and the system behaviour is given by the output, as the program runs.

Throughout this book we have been using FORTRAN for this purpose, because it is the best known and most readily available computer language. (We could have used ALGOL, PL/I, COBOL, or any of the other general-purpose languages, if we had desired to do so.)

You might have observed that even a moderately complex simulation model (such as a 2-server queue in Sec. 4-3 or an activity network in Sec. 5-5) can become difficult to program, to debug, and to modify. A programmer who has to perform simulation frequently would be better off learning a higher-level special-purpose language, which facilitates simulation programming. The extra effort he invests in learning such a simulation language would be made up by the ease and increased speed with which he can now write a simulation program.

Besides being a programming aid, a simulation language often serves another very useful purpose. It provides a conceptual framework in which the system analyst can work while building a model of the system. The language imposes a certain amount of precise thinking on the system analyst and forces him to seek answers to numerous useful and appropriate questions. A simulation language also provides a terminology for describing a complex dynamic system.

Unfortunately, there is a bewildering variety of simulation languages. Of scores of simulation languages that have been designed, implemented, and used, there is no single language which can be termed as the "best," "most useful," or "universally" available. Most of these languages are suited for a narrow class of applications. If we were simulating a military system we might pick MILITRAN; but for simulating a world dynamics model we would perhaps use DYNAMO; for simulating a sampled-data system we might use BLODIB; but for simulating a job-shop scheduling, GPSS would be preferred.

In addition to the nature of a system being simulated, the availability of the hardware and the software also dictates the choice of a language. If we have only a minicomputer at our disposal, we would not consider implementing a "large" language such as SIMSCRIPT II.5 for our simulation experiments. Availability of a simulation language is an obvious but very important consideration in choosing a language. All languages are not available on all computer systems.

In this chapter we will present an overview of simulation languages without describing any one specific language in all its details. The details can be found in various manuals.

8-1. Continuous and discrete simulation languages

As discussed earlier, simulation is divided into two categories: discrete and continuous. Accordingly, most of the simulation languages also fall into one of the two classes. Continuous simulation languages are designed for simulating continuous models and discrete simulation languages for discrete models—models that change their states abruptly at discrete points in time. More recently, a few languages have been designed which are suitable for both discrete as well as continuous models. Such a language, called a *combined simulation language*, is written particularly for system-models in which some of the variables change continuously and other variables change discretely. These continuous and discrete variables, of course, interact with each other. For example, the number of passengers in a lift change discretely whereas the lift's distance from the ground floor or its speed varies continuously. GASP IV is perhaps the best known example of a combined simulation language. Other examples are GSL-A (*G*eneralized *S*imulation *L*anguage), CLASS (*C*omposite *L*anguage *A*pproach for *S*ystem *S*imulation), and PROSE (*Pro*blem Level Programming System).

8-2. Continuous simulation languages

Before the digital computers came into widespread use, analog computers were being used for simulating continuous dynamic systems. The system being simulated was generally an engineering system, described by differential equations, for which analytic solutions were hard to obtain. The system was represented by means of a block diagram (such as the one shown in Fig. 2-5 of Chapter 2), which was then simulated on the analog computer by suitably patching together corresponding blocks available on the analog computer. As soon as the digital computer arrived, some of its advantages (such as greater accuracy, freedom from scaling) over the analog computer became obvious. Therefore, special program packages (canned programs) were implemented on digital computers to make a digital computer appear like an analog computer.

These so-called *digital analog simulators* (program packages) were meant either to replace an analog computer or check the results of an analog simu-

Index

System Simulation with Digital Computer

by

Narsingh Deo

Designed for students of engineering and business administration, as well as for practicing systems analysts, industrial engineers and operations research specialists, this text gives a rigorous grounding in the use of simulation techniques to solve simple, but mathematically intractable problems.

The book emphasises simulation of discrete, stochastic, and dynamic systems; however, other types of systems are also discussed. It presents a complete overview of computer simulation and its applications with emphasis on digital computer simulation of continuous systems. It also provides indepth discussion on queuing systems, stochastic networks, and inventory systems, and comprehensively covers the very important aspect of stochastic simulation—the design of a simulation experiment, which includes problems of statistical reliability in evaluating simulation experiments.

This text was class-tested and grew out of a simulation courses given to students of electrical engineering and computer science at the Indian Institute of Technology Kanpur.

ISBN 81-203-0028-9

9 788120 300286

Rs. 95.00