9

Input Modeling

Input models provide the driving force for a simulation model. In the simulation of a queueing system, typical input models are the distributions of time between arrivals and of service times. For an inventory-system simulation, input models include the distributions of demand and of lead time. For the simulation of a reliability system, the distribution of time to failure of a component is an example of an input model.

In the examples and exercises in Chapters 2 and 3, the appropriate distributions were specified for you. In real-world simulation applications, however, coming up with appropriate distributions for input data is a major task from the standpoint of time and resource requirements. Regardless of the sophistication of the analyst, faulty models of the inputs will lead to outputs whose interpretation could give rise to misleading recommendations.

There are four steps in the development of a useful model of input data:

1. Collect data from the real system of interest. This often requires a substantial time and resource commitment. Unfortunately, in some situations it is not possible to collect data (for example, when time is extremely limited, when the input process does not yet exist, or when laws or rules prohibit the collection of data). When data are not available, expert opinion and knowledge of the process must be used to make educated guesses.
2. Identify a probability distribution to represent the input process. When data are available, this step typically begins with the development of a frequency distribution, or histogram, of the data. Given the frequency distribution and a structural knowledge of the process, a family of distributions is chosen. Fortunately, as was described in Chapter 5, several well-known distributions often provide good approximations in practice.
3. Choose parameters that determine a specific instance of the distribution family. When data are available, these parameters may be estimated from the data.

4. Evaluate the chosen distribution and the associated parameters for goodness of fit. Goodness of fit may be evaluated informally, via graphical methods, or formally, via statistical tests. The chi-square and the Kolmogorov–Smirnov tests are standard goodness-of-fit tests. If not satisfied that the chosen distribution is a good approximation of the data, then the analyst returns to the second step, chooses a different family of distributions, and repeats the procedure. If several iterations of this procedure fail to yield a fit between an assumed distributional form and the collected data, the empirical form of the distribution may be used, as was described in Section 8.1.5.

Each of these steps is discussed in this chapter. Although software is now widely available to accomplish Steps 2, 3, and 4—including such stand-alone programs as ExpertFit® and Stat::Fit® and such integrated programs as Arena's Input Processor and @Risk's BestFit®—it is still important to understand what the software does, so that it can be used appropriately. Unfortunately, software is not as readily available for input modeling when there is a relationship between two or more variables of interest or when no data are available. These two topics are discussed toward the end of the chapter.

9.1 DATA COLLECTION

Problems are found at the end of each chapter, as exercises for the reader, in textbooks about mathematics, physics, chemistry, and other technical subjects. Years and years of working these problems could give the reader the impression that data are readily available. Nothing could be further from the truth. Data collection is one of the biggest tasks in solving a real problem. It is one of the most important and difficult problems in simulation. And, even when data are available, they have rarely been recorded in a form that is directly useful for simulation input modeling.

"GIGO," or "garbage-in–garbage-out," is a basic concept in computer science, and it applies equally in the area of discrete-system simulation. Even when the model structure is valid, if the input data are inaccurately collected, inappropriately analyzed, or not representative of the environment, the simulation output data will be misleading and possibly damaging or costly when used for policy or decision making.

Example 9.1: The Laundromat

As budding simulation students, the first two authors had assignments to simulate the operation of an ongoing system. One of these systems, which seemed to be a rather simple operation, was a self-service laundromat with 10 washing machines and six dryers.

However, the data-collection aspect of the problem rapidly became rather enormous. The interarrival-time distribution was not homogeneous; it changed by time of day and by day of week. The laundromat was open 7 days a week for 16 hours per day, or 112 hours per week. It would have been impossible to cover the operation of the lanndromat with the limited resources available (two students who were also taking four other courses) and with a tight time constraint (the simulation was to be completed in a 4-week period). Additionally, the distribution of time between arrivals during one week might not have been followed during the next week. As a compromise, a sample of times was selected, and the interarrival-time distributions were classified according to arrival rate (perhaps inappropriately) as "high," "medium," and "low."

Service-time distributions also presented a difficult problem from many perspectives. The proportion of customers demanding the various service combinations had to be observed and recorded. The simplest case was the customer desiring one washer followed by one dryer. However, a customer might choose two washing machines followed by one dryer, one dryer only, and so on. The customers used numbered machines, and it was possible to follow the customers via that reference, rather than remembering them by personal characteristics. Because of the dependence between washer demand and dryer demand for an individual customer, it would have been inappropriate to treat the service times for washers and dryers separately as independent variables.

Some customers waited patiently for their clothes to complete the washing or drying cycle, and then they removed their clothes promptly. Others left the premises and returned after their clothes had finished their cycle on the machine being used. In a very busy period, the manager would remove a customer's clothes after the cycle and set them aside in a basket. It was decided that service termination would be measured as that point in time at which the machine was emptied of its contents.

Also, machines would break down from time to time. The length of the breakdown varied from a few moments, when the manager repaired the machine, to several days (a breakdown on Friday night, requiring a part not in the laundromat storeroom, would not be fixed until the following Monday). The short-term repair times were recorded by the student team. The long-term repair completion times were estimated by the manager. Breakdowns then became part of the simulation.

Many lessons can be learned from an actual experience at data collection. The first five exercises at the end of this chapter suggest some situations in which the student can gain such experience.

The following suggestions might enhance and facilitate data collection, although they are not all inclusive.

1. A useful expenditure of time is in planning. This could begin by a practice or preobserving session. Try to collect data while preobserving. Devise forms for this purpose. It is very likely that these forms will have to be modified several times before the actual data collection begins. Watch for unusnal circumstances, and consider how they will be handled. When possible, videotape the system and extract the data later by viewing the tape. Planning is important, even if data will be collected automatically (e.g., via computer data collection), to ensure that the appropriate data are available. When data have already been collected by someone else, be sure to allow plenty of time for converting the data into a usable format.
2. Try to analyze the data as they are being collected. Figure out whether the data being collected are adequate to provide the distributions needed as input to the simulation. Find out whether any data being collected are useless to the simulation. There is no need to collect superfluous data.
3. Try to combine homogeneous data sets. Check data for homogeneity in successive time periods and during the same time period on successive days. For example, check for homogeneity of data from 2:00 P.M. to 3:00 P.M. and 3:00 P.M. to 4:00 P.M., and check to see whether the data are homogeneous for 2:00 P.M. to 3:00 P.M. on Thursday and Friday. When checking for homogeneity, an initial test is to see whether the means of the distributions (the average interarrival times, for example) are the same. The two-sample t test can be used for this purpose. A more thorough analysis would require a test of the equivalence of the distributions, perhaps via a quantile-quantile plot (described later).
4. Be aware of the possibility of data censoring, in which a quantity of interest is not observed in its entirety. This problem most often occurs when the analyst is interested in the time required to complete some process (for example, produce a part, treat a patient, or have a component fail), but the process begins prior to, or finishes after the completion of, the observation period. Censoring can result in especially long process times being left out of the data sample.
5. To discover whether there is a relationship between two variables, build a scatter diagram. Sometimes an eyeball scan of the scatter diagram will indicate whether there is a relationship between two variables of interest. Section 9.7 describes models for statistically dependent input data.
6. Consider the possibility that a sequence of observations that appear to be independent actually has autocorrelation. Autocorrelation can exist in successive time periods or for successive customers.

For example, the service time for the ith customer could be related to the service time for the $(i+n)$th customer. A brief introduction to autocorrelation was provided in Section 7.4.2, and some input models that account for autocorrelation are presented in Section 9.7.

7. Keep in mind the difference between input data and output or performance data, and be sure to collect input data. Input data typically represent the uncertain quantities that are largely beyond the control of the system and will not be altered by changes made to improve the system. Output data, on the other hand, represent the performance of the system when subjected to the inputs, performance that we might be trying to improve. In a queueing simulation, the customer arrival times are usually inputs, whereas the customer delay is an output. Performance data are useful for model validation, however—see Chapter 10.

Again, these are just a few suggestions. As a rule, data collection and analysis must be approached with great care.

9.2 IDENTIFYING THE DISTRIBUTION WITH DATA

In this section, we discuss methods for selecting families of input distributions when data are available. The specific distribution within a family is specified by estimating its parameters, as described in Section 9.3. Section 9.6 takes up the case in which data are unavailable.

9.2.1 Histograms

A frequency distribution or histogram is useful in identifying the shape of a distribution. A histogram is constructed as follows:

1. Divide the range of the data into intervals. (Intervals are usually of equal width; however, unequal widths may be used if the heights of the frequencies are adjusted.)
2. Label the horizontal axis to conform to the intervals selected.
3. Find the frequency of occurrences within each interval.
4. Label the vertical axis so that the total occurrences can be plotted for each interval.
5. Plot the frequencies on the vertical axis.

The number of class intervals depends on the number of observations and on the amount of scatter or dispersion in the data. Hines, Montgomery, Goldsman, and Borrow [2002] state that choosing the number of class intervals approximately equal to the square root of the sample size often works well in practice. If the intervals are too wide, the histogram will be coarse, or blocky, and its shape and other details will not show well. If the intervals are too narrow, the histogram will be ragged and will not smooth the data. Examples of ragged, coarse, and appropriate histograms of the same data are shown in Figure 9.1. Modern data-analysis software often allows the interval sizes to be changed easily and interactively until a good choice is found.

The histogram for continuous data corresponds to the probability density function of a theoretical distribution. If continuous, a line drawn through the center point of each class interval frequency should result in a shape like that of a pdf.

Histograms for discrete data, where there are a large number of data points, should have a cell for each value in the range of the data. However, if there are few data points, it could be necessary to combine adjacent

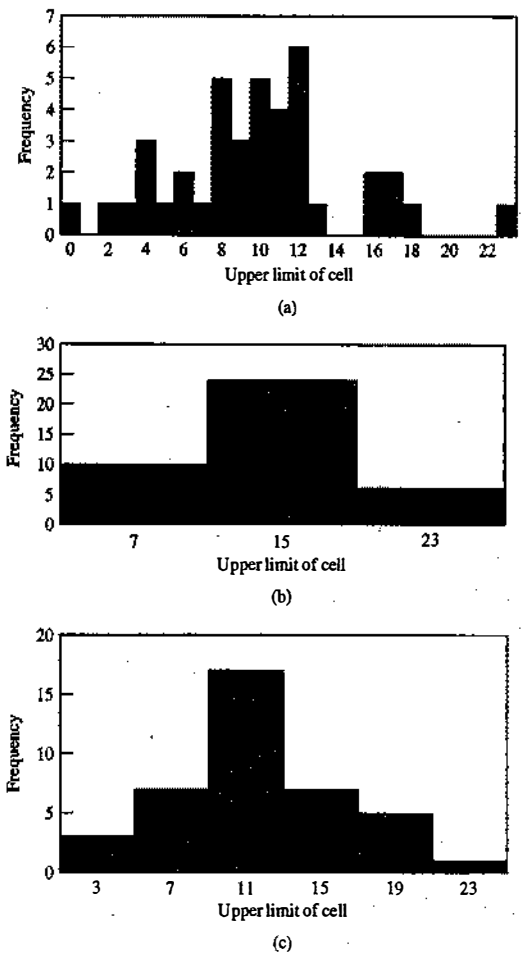


Figure 9.1 Ragged, coarse, and appropriate histograms: (a) original data—too ragged; (b) combining adjacent cells—too coarse; (c) combining adjacent cells—appropriate.

cells to eliminate the ragged appearance of the histogram. If the histogram is associated with discrete data, it should look like a probability mass function.

Example 9.2: Discrete Data

The number of vehicles arriving at the northwest corner of an intersection in a 5-minute period between 7:00 A.M. and 7:05 A.M. was monitored for five workdays over a 20-week period. Table 9.1 shows the resulting data. The first entry in the table indicates that there were 12 5-minute periods during which zero vehicles arrived, 10 periods during which one vehicle arrived, and so on.

The number of automobiles is a discrete variable, and there are ample data, so the histogram may have a cell for each possible value in the range of the data. The resulting histogram is shown in Figure 9.2.

Table 9.1 Number of Arrivals in a 5-Minute Period

Arrivals per Period	*Frequency*	*Arrivals per Period*	*Frequency*
0	12	6	7
1	10	7	5
2	19	8	5
3	17	9	3
4	10	10	3
5	8	11	1

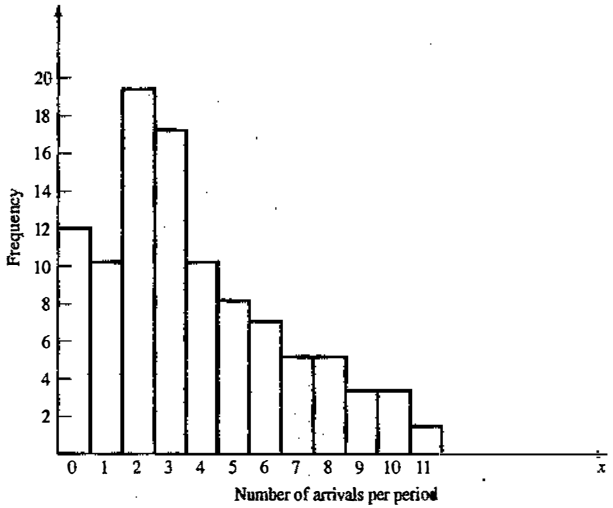


Figure 9.2 Histogram of number of arrivals per period.

Example 9.3: Continuous Data

Life tests were performed on a random sample of electronic components at 1.5 times the nominal voltage, and their lifetime (or time to failure), in days, was recorded:

79.919	3.081	0.062	1.961	5.845
3.027	6.505	0.021	0.013	0.123
6.769	59.899	1.192	34.760	5.009
18.387	0.141	43.565	24.420	0.433
144.695	2.663	17.967	0.091	9.003
0.941	0.878	3.371	2.157	7.579
0.624	5.380	3.148	7.078	23.960
0.590	1.928	0.300	0.002	0.543
7.004	31.764	1.005	1.147	0.219
3.217	14.382	1.008	2.336	4.562

Table 9.2 Electronic Component Data

Component Life (Days)	*Frequency*
$0 \le x_j < 3$	23
$3 \le x_j < 6$	10
$6 \le x_j < 9$	5
$9 \le x_j < 12$	1
$12 \le x_j < 15$	1
$15 \le x_j < 18$	2
$18 \le x_j < 21$	0
$21 \le x_j < 24$	1
$24 \le x_j < 27$	1
$27 \le x_j < 30$	0
$30 \le x_j < 33$	1
$33 \le x_j < 36$	1
.	.
.	.
.	.
$42 \le x_j < 45$	1
.	.
.	.
.	.
$57 \le x_j < 60$	1
.	.
.	.
.	.
$78 \le x_j < 81$	1
.	.
.	.
.	.
$144 \le x_j < 147$	1

Lifetime, usually considered a continuous variable, is recorded here to three-decimal-place accuracy. The histogram is prepared by placing the data in class intervals. The range of the data is rather large, from 0.002 day to 144.695 days. However, most of the values (30 of 50) are in the zero-to-5-day range. Using intervals of width three results in Table 9.2. The data of Table 9.2 are then used to prepare the histogram shown in Figure 9.3.

9.2.2 Selecting the Family of Distributions

In Chapter 5, some distributions that arise often in simulation were described. Additionally, the shapes of these distributions were displayed. The purpose of preparing a histogram is to infer a known pdf or pmf. A family of distributions is selected on the basis of what might arise in the context being investigated along with the shape of the histogram. Thus, if interarrival-time data have been collected, and the histogram has a shape similar to the pdf in Figure 5.9, the assumption of an exponential distribution would be warranted. Similarly, if measurements of the weights of pallets of freight are being made, and the histogram appears

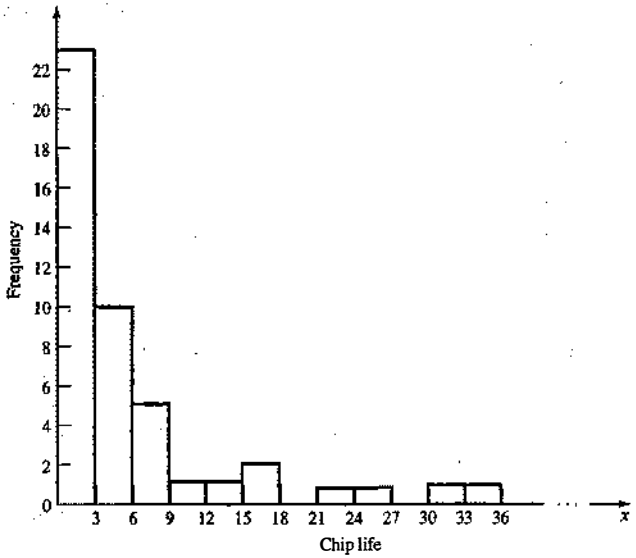


Figure 9.3 Histogram of component life.

symmetric about the mean with a shape like that shown in Figure 5.11, the assumption of a normal distribution would be warranted.

The exponential, normal, and Poisson distributions are frequently encountered and are not difficult to analyze from a computational standpoint. Although more difficult to analyze, the beta, gamma, and Weibull distributions provide a wide array of shapes and should not be overlooked during modeling of an underlying probabilistic process. Perhaps an exponential distribution was assumed, but it was found not to fit the data. The next step would be to examine where the lack of fit occurred. If the lack of fit was in one of the tails of the distribution, perhaps a gamma or Weibull distribution would fit the data more adequately.

There are literally hundreds of probability distributions that have been created; many were created with some specific physical process in mind. One aid to selecting distributions is to use the physical basis of the distributions as a guide. Here are some examples:

Binomial: Models the number of successes in n trials, when the trials are independent with common success probability, p; for example, the number of defective computer chips found in a lot of n chips.

Negative Binomial (includes the geometric distribution): Models the number of trials required to achieve k successes; for example, the number of computer chips that we must inspect to find 4 defective chips.

Poisson: Models the number of independent events that occur in a fixed amount of time or space; for example, the number of customers that arrive to a store during 1 hour, or the number of defects found in 30 square meters of sheet metal.

Normal: Models the distribution of a process that can be thought of as the sum of a number of component processes; for example, a time to assemble a product that is the sum of the times required for each assembly operation. Notice that the normal distribution admits negative values, which could be impossible for process times.

Lognormal: Models the distribution of a process that can be thought of as the product of (meaning to multiply together) a number of component processes—for example, the rate on an investment, when interest is compounded, is the product of the returns for a number of periods.

Exponential: Models the time between independent events, or a process time that is memoryless (knowing how much time has passed gives no information about how much additional time will pass before the process is complete)—for example, the times between the arrivals from a large population of potential customers who act independently of each other. The exponential is a highly variable distribution; it is sometimes overused, because it often leads to mathematically tractable models. Recall that, if the time between events is exponentially distributed, then the number of events in a fixed period of time is Poisson.

Gamma: An extremely flexible distribution used to model nonnegative random variables. The gamma can be shifted away from 0 by adding a constant.

Beta: An extremely flexible distribution used to model bounded (fixed upper and lower limits) random variables. The beta can be shifted away from 0 by adding a constant and can be given a range larger than [0, 1] by multiplying by a constant.

Erlang: Models processes that can be viewed as the sum of several exponentially distributed processes—for example, a computer network fails when a computer and two backup computers fail, and each has a time to failure that is exponentially distributed. The Erlang is a special case of the gamma.

Weibull: Models the time to failure for components—for example, the time to failure for a disk drive. The exponential is a special case of the Weibull.

Discrete or Continuous Uniform: Models complete uncertainty: All outcomes are equally likely. This distribution often is used inappropriately, when there are no data.

Triangular: Models a process for which only the minimum, most likely, and maximum values of the distribution are known; for example, the minimum, most likely, and maximum time required to test a product. This model is often a marked improvement over a uniform distribution.

Empirical: Resamples from the actual data collected; often used when no theoretical distribution seems appropriate.

Do not ignore physical characteristics of the process when selecting distributions. Is the process naturally discrete or continuous valued? Is it bounded, or is there no natural bound? This knowledge, which does not depend on data, can help narrow the family of distributions from which to choose. And keep in mind that there is no "true" distribution for any stochastic input process. An input model is an approximation of reality, so the goal is to obtain an approximation that yields useful results from the simulation experiment.

The reader is encouraged to complete Exercises 6 through 11 to learn more about the shapes of the distributions mentioned in this section. Examining the variations in shape as the parameters change is very instructive.

9.2.3 Quantile–Quantile Plots

The construction of histograms, as discussed in Section 9.2.1, and the recognition of a distributional shape, as discussed in Section 9.2.2, are necessary ingredients for selecting a family of distributions to represent a sample of data. However, a histogram is not as useful for evaluating the *fit* of the chosen distribution. When there is a small number of data points, say 30 or fewer, a histogram can be rather ragged. Further, our perception of the fit depends on the widths of the histogram intervals. But, even if the intervals are chosen well, grouping data into cells makes it difficult to compare a histogram to a continuous probability density function. A quantile–quantile ($q - q$) plot is a useful tool for evaluating distribution fit, one that does not suffer from these problems.

If X is a random variable with cdf F, then the q-quantile of X is that value γ such that $F(\gamma) = P(X \le \gamma) = q$, for $0 < q < 1$. When F has an inverse, we write $\gamma = F^{-1}(q)$.

Now let $\{x_i, i = 1, 2, \ldots, n\}$ be a sample of data from X. Order the observations from the smallest to the largest, and denote these as $\{y_j, j = 1, 2, \ldots, n\}$, where $y_1 \le y_2 \le \cdots \le y_n$. Let j denote the ranking or order number. Therefore, $j = 1$ for the smallest and $j = n$ for the largest. The $q - q$ plot is based on the fact that y_j is an estimate of the $(j - 1/2)/n$ quantile of X. In other words,

$$y_j \text{ is approximately } F^{-1}\left(\frac{j-\frac{1}{2}}{n}\right)$$

Now suppose that we have chosen a distribution with cdf F as a possible representation of the distribution of X. If F is a member of an appropriate family of distributions, then a plot of y_j versus $F^{-1}((j - 1/2)/n)$ will be *approximately a straight line.* If F is from an appropriate family of distributions and also has appropriate parameter values, then the line will have slope 1. On the other hand, if the assumed distribution is inappropriate, the points will deviate from a straight line, usually in a systematic manner. The decision about whether to reject some hypothesized model is subjective.

Example 9.4: Normal $Q - Q$ Plot

A robot is used to install the doors on automobiles along an assembly line. It was thought that the installation times followed a normal distribution. The robot is capable of measuring installation times accurately. A sample of 20 installation times was automatically taken by the robot, with the following results, where the values are in seconds:

99.79	99.56	100.17	100.33
100.26	100.41	99.98	99.83
100.23	100.27	100.02	100.47
99.55	99.62	99.65	99.82
99.96	99.90	100.06	99.85

The sample mean is 99.99 seconds, and the sample variance is $(0.2832)^2$ seconds^2. These values can serve as the parameter estimates for the mean and variance of the normal distribution. The observations are now ordered from smallest to largest as follows:

j	*Value*	j	*Value*	j	*Value*	j	Value
1	99.55	6	99.82	11	99.98	16	100.26
2	99.56	7	99.83	12	100.02	17	100.27
3	99.62	8	99.85	13	100.06	18	100.33
4	99.65	9	99.90	14	100.17	19	100.41
5	99.79	10	99.96	15	100.23	20	100.47

The ordered observations are then plotted versus $F^{-1}((j - 1/2)/20)$, for $j = 1, 2, \ldots, 20$, where F is the cdf of the normal distribution with mean 99.99 and variance $(0.2832)^2$, to obtain a $q - q$ plot. The plotted values are shown in Figure 9.4, along with a histogram of the data that has the density function of the normal distribution superimposed. Notice that it is difficult to tell whether the data are well represented by a normal

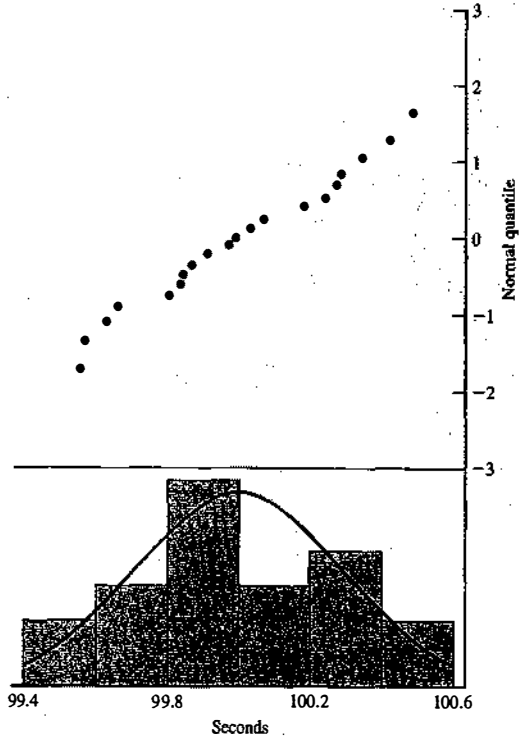


Figure 9.4 Histogram and $q - q$ plot of the installation times.

distribution from looking at the histogram, but the general perception of a straight line is quite clear in the $q - q$ plot and supports the hypothesis of a normal distribution.

In the evaluation of the linearity of a $q - q$ plot, the following should be considered:

1. The observed values will never fall exactly on a straight line.
2. The ordered values are not independent; they have been ranked. Hence, if one point is above a straight line, it is likely that the next point will also lie above the line. And it is unlikely that the points will be scattered about the line.
3. The variances of the extremes (largest and smallest values) are much higher than the variances in the middle of the plot. Greater discrepancies can be accepted at the extremes. The linearity of the points in the middle of the plot is more important than the linearity at the extremes.

Modern data-analysis software often includes tools for generating $q - q$ plots, especially for the normal distribution. The $q - q$ plot can also be used to compare two samples of data to see whether they can be represented by the same distribution (that is, that they are homogeneous). If $x_1, x_2, \ldots, x_n$ are a sample of the random variable X, and $z_1, z_2, \ldots, z_n$ are a sample of the random variable Z, then plotting the ordered values of X versus the ordered values of Z will reveal approximately a straight line if both samples are well represented by the same distribution (Chambers, Cleveland, and Tukey [1983]).

9.3 PARAMETER ESTIMATION

After a family of distributions has been selected, the next step is to estimate the parameters of the distribution. Estimators for many useful distributions are described in this section. In addition, many software packages—some of them integrated into simulation languages—are now available to compute these estimates.

9.3.1 Preliminary Statistics: Sample Mean and Sample Variance

In a number of instances, the sample mean, or the sample mean and sample variance, are used to estimate the parameters of a hypothesized distribution; see Example 9.4. In the following paragraphs, three sets of equations are given for computing the sample mean and sample variance. Equations (9.1) and (9.2) can be used when discrete or continuous raw data are available. Equations (9.3) and (9.4) are used when the data are discrete and have been grouped in a frequency distribution. Equations (9.5) and (9.6) are used when the data are discrete or continuous and have been placed in class intervals. Equations (9.5) and (9.6) are approximations and should be used only when the raw data are unavailable.

If the observations in a sample of size n are $X_1, X_2, \ldots, X_n$, the sample mean ($\bar{X}$) is defined by

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{9.1}$$

and the sample variance, S^2, is defined by

$$S^2 = \frac{\sum_{i=1}^{n} X_i^2 - n\bar{X}^2}{n-1} \tag{9.2}$$

If the data are discrete and have been grouped in a frequency distribution, Equations (9.1) and (9.2) can be modified to provide for much greater computational efficiency. The sample mean can be computed as

$$\bar{X} = \frac{\sum_{j=1}^{k} f_j X_j}{n} \tag{9.3}$$

and the sample variance as

$$S^2 = \frac{\sum_{j=1}^{k} f_j X_j^2 - n\bar{X}^2}{n-1} \tag{9.4}$$

where k is the number of distinct values of X and f_j is the observed frequency of the value X_j of X.

Example 9.5: Grouped Data

The data in Table 9.1 can be analyzed to obtain $n = 100, f_1 = 12, X_1 = 0, f_2 = 10, X_2 = 1, \ldots, \sum_{j=1}^{k} f_j X_j = 364$, and $\sum_{j=1}^{k} = f_j X_j^2 = 2080$. From Equation (9.3),

$$\bar{X} = \frac{364}{100} = 3.64$$

and, from Equation (9.4),

$$S^2 = \frac{2080 - 100(3.64)^2}{99} = 7.63$$

The sample standard deviation, S, is just the square root of the sample variance. In this case, $S = \sqrt{7.63} = 2.76$. Equations (9.1) and (9.2) would have yielded exactly the same results for $\bar{X}$ and S^2.

It is preferable to use the raw data, if possible, when the values are continuous. However, data sometimes are received after having been placed in class intervals. Then it is no longer possible to obtain the exact sample mean and variance. In such cases, the sample mean and sample variance are approximated from the following equations:

$$\bar{X} \doteq \frac{\sum_{j=1}^{c} f_j m_j}{n} \tag{9.5}$$

and

$$S^2 \doteq \frac{\sum_{j=1}^{c} f_j m_j^2 - n\bar{X}^2}{n-1} \tag{9.6}$$

where f_j is the observed frequency in the jth class interval, m_j is the midpoint of the jth interval, and c is the number of class intervals.

Example 9.6: Continuous Data in Class Intervals

Assume that the raw data on component life shown in Example 9.3 either was discarded or was lost. However, the data shown in Table 9.2 are still available. To approximate values for $\bar{X}$ and S^2, Equations (9.5) and (9.6) are used. The following values are created: $f_1 = 23, m_1 = 1.5, f_2 = 10, m_2 = 4.5, \ldots, \sum_{j=1}^{49} f_j m_j = 614$ and $\sum_{j=1}^{49} f_j m_j^2 = 37{,}226.5$. With $n = 50$, $\bar{X}$ is approximated from Equation (9.5) as

$$\bar{X} \doteq \frac{614}{50} = 12.28$$

Then, S^2 is approximated from Equation (9.6) as

$$S^2 \doteq \frac{37{,}226.5 - 50(12.28)^2}{49} = 605.849$$

and

$$S \doteq 24.614$$

Applying Equations (9.1) and (9.2) to the original data in Example 9.3 results in $\bar{X} = 11.894$ and $S = 24.953$. Thus, when the raw data are either discarded or lost, inaccuracies could result.

9.3.2 Suggested Estimators

Numerical estimates of the distribution parameters are needed to reduce the family of distributions to a specific distribution and to test the resulting hypothesis. Table 9.3 contains suggested estimators for distributions often used in simulation, all of which were described in Chapter 5. Except for an adjustment to remove bias in the estimate of σ^2 for the normal distribution, these estimators are the maximum-likelihood estimators based on the raw data. (If the data are in class intervals, these estimators must be modified.) The reader

Table 9.3 Suggested Estimators for Distributions Often Used in Simulation

Distribution	*Parameter(s)*	*Suggested Estimator(s)*
Poisson	α	$\hat{\alpha} = \bar{X}$
Exponential	λ	$\hat{\lambda} = \frac{1}{\bar{X}}$
Gamma	β, θ	$\hat{\beta}$ (see Table A.9) $\hat{\theta} = \frac{1}{\bar{X}}$
Normal	μ, σ^2	$\hat{\mu} = \bar{X}$ $\hat{\sigma}^2 = S^2$ (unbiased)
Lognormal	μ, σ^2	$\hat{\mu} = \bar{X}$ (after taking ln of the data) $\hat{\sigma}^2 = S^2$ (after taking ln of the data)
Weibull with $\nu = 0$	α, β	$\hat{\beta}_0 = \frac{\bar{X}}{S}$ $\hat{\beta}_j = \hat{\beta}_{j-1} - \frac{f(\hat{\beta}_{j-1})}{f'(\hat{\beta}_{j-1})}$ See Equations (9.12) and (9.15) for $f(\hat{\beta})$ and $f'(\hat{\beta})$ Iterate until convergence $\hat{\alpha} = \left(\frac{1}{n}\sum_{i=1}^{n} X_i^{\hat{\beta}}\right)^{1/\hat{\beta}}$
Beta	β_1, β_2	$\Psi(\hat{\beta}_1) + \Psi(\hat{\beta}_1 - \hat{\beta}_2) = \ln(G_1)$ $\Psi(\hat{\beta}_2) + \Psi(\hat{\beta}_1 - \hat{\beta}_2) = \ln(G_2)$ where Ψ is the digamma function, $G_1 = \left(\prod_{i=1}^{n} X_i\right)^{1/n}$ and $G_2 = \left(\prod_{i=1}^{n} (1 - X_i)\right)^{1/n}$

is referred to Fishman [1973] and Law and Kelton [2000] for parameter estimates for the uniform, binomial, and negative binomial distributions. The triangular distribution is usually employed when no data are available, with the parameters obtained from educated guesses for the minimum, most likely, and maximum possible values; the uniform distribution may also be used in this way if only minimum and maximum values are available.

Examples of the use of the estimators are given in the following paragraphs. The reader should keep in mind that a parameter is an unknown constant, but the estimator is a statistic (or random variable), because it depends on the sample values. To distinguish the two clearly here, if, say, a parameter is denoted by α, the estimator will be denoted by $\hat{\alpha}$.

Example 9.7: Poisson Distribution

Assume that the arrival data in Table 9.1 require analysis. By comparison with Figure 5.7, an examination of Figure 9.2 suggests a Poisson distributional assumption with unknown parameter α. From Table 9.3, the estimator of α is $\bar{X}$, which was found in Example 9.5. Thus, $\hat{\alpha} = 3.64$. Recall that the true mean and variance are equal for the Poisson distribution. In Example 9.5, the sample variance was estimated as $S^2 = 7.63$. However, it should never be expected that the sample mean and the sample variance will be precisely equal, because each is a random variable.

Example 9.8: Lognormal Distribution

The rates of return on 10 investments in a portfolio are 18.8, 27.9, 21.0, 6.1, 37.4, 5.0, 22.9, 1.0, 3.1 and 8.3 percent. To estimate the parameters of a lognormal model of these data, we first take the natural log of the data and obtain 2.9, 3.3, 3.0, 1.8, 3.6, 1.6, 3.1, 0, 1.1, and 2.1. Then we set $\hat{\mu} = \bar{X} = 2.3$ and $\hat{\sigma}^2 = S^2 = 1.3$.

Example 9.9: Normal Distribution

The parameters of the normal distribution, μ and σ^2, are estimated by $\bar{X}$ and S^2, as shown in Table 9.3. The $q - q$ plot in Example 9.4 leads to a distributional assumption that the installation times are normal. From Equations (9.1) and (9.2), the data in Example 9.4 yield $\hat{\mu} = \bar{X} = 99.9865$ and $\hat{\sigma} = S^2 = (0.2832)^2$ second^2.

Example 9.10: Gamma Distribution

The estimator $\hat{\beta}$ for the gamma distribution is chosen by the use of Table A.9, from Choi and Wette [1969]. Table A.9 requires the computation of the quantity $1/M$, where

$$M = \ln \bar{X} - \frac{1}{n}\sum_{i=1}^{n} \ln X_i \tag{9.7}$$

Also, it can be seen in Table 9.3 that $\hat{\theta}$ is given by

$$\hat{\theta} = \frac{1}{\bar{X}} \tag{9.8}$$

In Chapter 5, it was stated that lead time is often gamma distributed. Suppose that the lead times (in days) associated with 20 orders have been accurately measured as follows:

Order	*Lead Time (Days)*	*Order*	*Lead Time (Days)*
1	70.292	11	30.215
2	10.107	12	17.137
3	48.386	13	44.024
4	20.480	14	10.552
5	13.053	15	37.298
6	25.292	16	16.314
7	14.713	17	28.073
8	39.166	18	39.019
9	17.421	19	32.330
10	13.905	20	36.547

To estimate $\hat{\beta}$ and $\hat{\theta}$, it is first necessary to compute M from Equation (9.7). Here, $\bar{X}$ is found, from Equation (9.1), to be

$$\bar{X} = \frac{564.32}{20} = 28.22$$

Then,

$$\ln \bar{X} = 3.34$$

Next,

$$\sum_{i=1}^{20} \ln X_i = 63.99$$

Then,

$$M = 3.34 - \frac{63.99}{20} = 0.14$$

and

$$1/M = 7.14$$

By interpolation in Table A.9, $\hat{\beta} = 3.728$. Finally, Equation (9.8) results in

$$\hat{\theta} = \frac{1}{28.22} = 0.035$$

Example 9.11: Exponential Distribution

Assuming that the data in Example 9.3 come from an exponential distribution, the parameter estimate, $\hat{\lambda}$, can be determined. In Table 9.3, $\hat{\lambda}$ is obtained from $\bar{X}$ as follows:

$$\hat{\lambda} = \frac{1}{\bar{X}} = \frac{1}{11.894} = 0.084 \text{ per day}$$

Example 9.12: Weibull Distribution

Suppose that a random sample of size n, $X_1, X_2, \ldots, X_n$, has been taken and that the observations are assumed to come from a Weibull distribution. The likelihood function derived by using the pdf given by Equation (5.47) can be shown to be

$$L(\alpha, \beta) = \frac{\beta^n}{\alpha^{\beta n}} \left[\prod_{i=1}^{n} X_i^{(\beta-1)} \right] \exp\left[-\sum_{i=1}^{n} \left(\frac{X_i}{\alpha} \right)^{\beta} \right] \tag{9.9}$$

The maximum-likelihood estimates are those values of $\hat{\alpha}$ and $\hat{\beta}$ that maximize $L(\alpha, \beta)$ or, equivalently, maximize $\ln L(\alpha, \beta)$, denoted by $l(\alpha, \beta)$. The maximum value of $l(\alpha, \beta)$ is obtained by taking the partial derivatives $\partial l(\alpha, \beta)/\partial \alpha$ and $\partial l(\alpha, \beta)/\partial \beta$, setting each to zero, and solving the resulting equations, which after substitution become

$$f(\beta) = 0 \tag{9.10}$$

and

$$\alpha = \left(\frac{1}{n} \sum_{i=1}^{n} X_i^{\beta} \right)^{1/\beta} \tag{9.11}$$

where

$$f(\beta) = \frac{n}{\beta} + \sum_{i=1}^{n} \ln X_i - \frac{n \sum_{i=1}^{n} X_i^{\beta} \ln X_i}{\sum_{i=1}^{n} X_i^{\beta}} \tag{9.12}$$

The maximum-likelihood estimates, $\hat{\alpha}$ and $\hat{\beta}$, are the solutions of Equations (9.10) and (9.11). First, $\hat{\beta}$ is found via the iterative procedure explained shortly. Then $\hat{\alpha}$ is found from Equation (9.11), with $\beta = \hat{\beta}$.

Equation (9.10) is nonlinear, so it is necessary to use a numerical-analysis technique to solve it. In Table 9.3, an iterative method for computing $\hat{\beta}$ is given as

$$\hat{\beta}_j = \hat{\beta}_{j-1} - \frac{f(\hat{\beta}_{j-1})}{f'(\hat{\beta}_{j-1})} \tag{9.13}$$

Equation (9.13) employs Newton's method in reaching $\hat{\beta}$, where $\hat{\beta}_j$ is the jth iteration, beginning with an initial estimate for $\hat{\beta}_0$, given in Table 9.3, as follows:

$$\hat{\beta}_0 = \frac{\bar{X}}{S} \tag{9.14}$$

If the initial estimate, $\hat{\beta}_0$, is sufficiently close to the solution $\hat{\beta}$, then $\hat{\beta}_j$ approaches $\hat{\beta}$ as $j \to \infty$. In Newton's method, $\hat{\beta}$ is approached through increments of size $f(\hat{\beta}_{j-1})/f'(\hat{\beta}_{j-1})$. Equation (9.12) is used to compute $f(\hat{\beta}_{j-1})$ and Equation (9.15) is used to compute, $f'(\hat{\beta}_{j-1})$ as follows:

$$f'(\beta) = -\frac{n}{\beta^2} - \frac{n \sum_{i=1}^{n} X_i^{\beta} (\ln X_i)^2}{\sum_{i=1}^{n} X_i^{\beta}} + \frac{n \left(\sum_{i=1}^{n} X_i^{\beta} \ln X_i \right)^2}{\left(\sum_{i=1}^{n} X_i^{\beta} \right)^2} \tag{9.15}$$

Equation (9.15) can be derived from Equation (9.12) by differentiating $f(\beta)$ with respect to β. The iterative process continues until $f(\hat{\beta}_i) \doteq 0$, for example, until $\left| f(\hat{\beta}_j) \right| \leq 0.001$.

Consider the data given in Example 9.3. These data concern the failure of electronic components and looks to come from an exponential distribution. In Example 9.11, the parameter $\hat{\lambda}$ was estimated on the hypothesis that the data were from an exponential distribution. If the hypothesis that the data came from an exponential distribution is rejected, an alternative hypothesis is that the data come from a Weibull distribution. The Weibull distribution is suspected because the data pertain to electronic component failures, which occur suddenly.

Equation (9.14) is used to compute $\hat{\beta}_0$. For the data in Example 9.3, $n = 50$, $\bar{X} = 11.894$, $\bar{X}^2 = 141.467$, and $\sum_{i=1}^{50} X_i^2 = 37{,}575.850$; so S^2 is found by Equation (9.2) to be

$$S^2 = \frac{37{,}578.850 - 50(141.467)}{49} = 622.650$$

and $S = 24.953$. Thus,

$$\hat{\beta}_0 = \frac{11.894}{24.953} = 0.477$$

To compute $\hat{\beta}_1$ by using Equation (9.13) requires the calculation of $f(\hat{\beta}_0)$ and $f'(\hat{\beta}_0)$ from Equations (9.12) and (9.15). The following additional values are needed: $\sum_{i=1}^{50} X_i^{\hat{\beta}_0} = 115.125$, $\sum_{i=1}^{50} \ln X_i = 38.294$, $\sum_{i=1}^{50} X_i^{\hat{\beta}_0} \ln X_i = 292.629$, and $\sum_{i=1}^{50} X_i^{\hat{\beta}_0} (\ln X_i)^2 = 1057.781$. Thus,

$$f(\hat{\beta}_0) = \frac{50}{0.477} + 38.294 - \frac{50(292.629)}{115.125} = 16.024$$

and

$$f'(\hat{\beta}_0) = \frac{-50}{(0.477)^2} - \frac{50(1057.781)}{115.125} + \frac{50(292.629)^2}{(115.125)^2} = -356.110$$

Then, by Equation (9.13),

$$\hat{\beta}_1 = 0.477 - \frac{16.024}{-356.110} = 0.522$$

After four iterations, $|f(\hat{\beta}_3)| \le 0.001$, at which point $\hat{\beta} \doteq \hat{\beta}_4 = 0.525$ is the approximate solution to Equation (9.10). Table 9.4 contains the values needed to complete each iteration.

Now, $\hat{\alpha}$ can be computed from Equation (9.11) with $\beta = \hat{\beta} = 0.525$, as follows:

$$\hat{\alpha} = \left[\frac{130.608}{50}\right]^{1/0.525} = 6.227$$

If $\hat{\beta}_0$ is sufficiently close to $\hat{\beta}$, the procedure converges quickly, usually in four to five iterations. However, if the procedure appears to be diverging, try other initial guesses for $\hat{\beta}_0$—for example, one-half the initial estimate or twice the initial estimate.

The difficult task of estimating parameters for the Weibull distribution by hand emphasizes the value of having software support for input modeling.

Table 9.4 Iterative Estimation of Parameters of the Weibull Distribution

j	$\hat{\beta}_j$	$\sum_{i=1}^{50} X_i^{\hat{\beta}_j}$	$\sum_{i=1}^{50} X_i^{\hat{\beta}_j} \ln X_i$	$\sum_{i=1}^{50} X_i^{\hat{\beta}_j} (\ln X_i)^2$	$f(\hat{\beta}_j)$	$f'(\hat{\beta}_j)$	$\hat{\beta}_{j+1}$
0	0.477	115.125	292.629	1057.781	16.024	−356.110	0.522
1	0.522	129.489	344.713	1254.111	1.008	−313.540	0.525
2	0.525	130.603	348.769	1269.547	0.004	−310.853	0.525
3	0.525	130.608	348.786	1269.614	0.000	−310.841	0.525


```
betaMLE := proc(X, n)
            local G1, G2, beta1, beta2, eqns, solns;
            G1 := product(X[i], i=1..n)^(1/n);
            G2 := product(1-X[i],i=1..n)^(1/n);
            eqns := {Psi(beta1) - Psi(beta1 + beta2) = ln(G1),
                     Psi(beta2) - Psi(beta1 + beta2) = ln(G2)};
            solns := fsolve(eqns, {beta1=0..infinity, beta2=0..infinity});
            RETURN(solns);
            end;
```

Figure 9.5 Maple procedure to compute the maximum likelihood estimates for the beta distribution parameters.

Example 9.13: Beta Distribution

The percentage of customers each month who bring in store coupons must be between 0 and 100 percent. Observations at a store for eight months gave the values 25%, 74%, 20%, 32%, 81%, 47%, 31%, and 8%. To fit a beta distribution to these data, we first need to rescale it to the interval (0, 1) by dividing all the values by 100, to get 0.25, 0.74, 0.20, 0.32, 0.81, 0.47, 0.31, 0.08.

The maximum-likelihood estimators of the parameters β_1, β_2 solve the system of equations shown in Table 9.3. Such equations can be solved by modern symbolic/numerical calculation programs, such as Maple; a Maple procedure for the beta parameters is shown in Figure 9.5. In this case, the solutions are $\hat{\beta}_1 = 1.47$ and $\hat{\beta}_2 = 2.16$.

9.4 GOODNESS-OF-FIT TESTS

Hypothesis testing was discussed in Section 7.4 with respect to testing random numbers. In Section 7.4.1, the Kolmogorov–Smirnov test and the chi-square test were introduced. These two tests are applied in this section to hypotheses about distributional forms of input data.

Goodness-of-fit tests provide helpful guidance for evaluating the suitability of a potential input model; however, there is no single correct distribution in a real application, so you should not be a slave to the verdict of such a test. It is especially important to understand the effect of sample size. If very little data are available, then a goodness-of-fit test is unlikely to reject *any* candidate distribution; but if a lot of data are available, then a goodness-of-fit test will likely reject *all* candidate distributions. Therefore, failing to reject a candidate distribution should be taken as one piece of evidence in favor of that choice, and rejecting an input model as only one piece of evidence against the choice.

9.4.1 Chi-Square Test

One procedure for testing the hypothesis that a random sample of size n of the random variable X follows a specific distributional form is the chi-square goodness-of-fit test. This test formalizes the intuitive idea of comparing the histogram of the data to the shape of the candidate density or mass function. The test is valid for large sample sizes and for both discrete and continuous distributional assumptions when parameters are estimated by maximum likelihood. The test procedure begins by arranging the n observations into a set of k class intervals or cells. The test statistic is given by

$$\chi_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{9.16}$$

where O_i is the observed frequency in the ith class interval and E_i is the expected frequency in that class interval. The expected frequency for each class interval is computed as $E_i = np_i$, where p_i is the theoretical, hypothesized probability associated with the ith class interval.

It can be shown that χ_0^2 approximately follows the chi-square distribution with $k - s - 1$ degrees of freedom, where s represents the number of parameters of the hypothesized distribution estimated by the sample statistics. The hypotheses are the following:

H_0: The random variable, X, conforms to the distributional assumption with the parameter(s) given by the parameter estimate(s).

H_1: The random variable X does not conform.

The critical value $\chi^2_{\alpha,k-s-1}$ is found in Table A.6. The null hypothesis, H_0, is rejected if $\chi_0^2 > \chi^2_{\alpha,k-s-1}$.

When applying the test, if expected frequencies are too small, χ_0^2 will reflect not only the departure of the observed from the expected frequency, but also the smallness of the expected frequency as well. Although there is no general agreement regarding the minimum size of E_i, values of 3, 4, and 5 have been widely used. In Section 7.4.1, when the chi-square test was discussed, the minimum expected frequency five was suggested. If an E_i value is too small, it can be combined with expected frequencies in adjacent class intervals. The corresponding O_i values should also be combined, and k should be reduced by one for each cell that is combined.

If the distribution being tested is discrete, each value of the random variable should be a class interval, unless it is necessary to combine adjacent class intervals to meet the minimum-expected-cell-frequency requirement. For the discrete case, if combining adjacent cells is not required,

$$p_i = p(x_i) = P(X = x_i)$$

Otherwise, p_i is found by summing the probabilities of appropriate adjacent cells.

If the distribution being tested is continuous, the class intervals are given by $[a_{i-1}, a_i)$, where a_{i-1} and a_i are the endpoints of the ith class interval. For the continuous case with assumed pdf $f(x)$, or assumed cdf $F(x)$, p_i can be computed as

$$p_i = \int_{a_{i-1}}^{a_i} f(x)dx = F(a_i) - F(a_{i-1})$$

For the discrete case, the number of class intervals is determined by the number of cells resulting after combining adjacent cells as necessary. However, for the continuous case, the number of class intervals must be specified. Although there are no general rules to be followed, the recommendations in Table 9.5 are made to aid in determining the number of class intervals for continuous data.

Table 9.5 Recommendations for Number of Class Intervals for Continuous Data

Sample Size, n	*Number of Class Intervals,* k
20	Do not use the chi-square test
50	5 to 10
100	10 to 20
>100	$\sqrt{n}$ to $n/5$

Example 9.14: Chi-Square Test Applied to Poisson Assumption

In Example 9.7, the vehicle-arrival data presented in Example 9.2 were analyzed. The histogram of the data, shown in Figure 9.2, appeared to follow a Poisson distribution; hence the parameter, $\hat{\alpha} = 3.64$, was found. Thus, the following hypotheses are formed:

H_0: the random variable is Poisson distributed.

H_1: the random variable is not Poisson distributed.

The pmf for the Poisson distribution was given in Equation (5.19):

$$p(x) = \begin{cases} \dfrac{e^{-\alpha}\alpha^x}{x!}, & x = 0, 1, 2, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{9.17}$$

For $\alpha = 3.64$, the probabilities associated with various values of x are obtained from Equation (9.17):

p(0) = 0.026	p(6) = 0.085
p(1) = 0.096	p(7) = 0.044
p(2) = 0.174	p(8) = 0.020
p(3) = 0.211	p(9) = 0.008
p(4) = 0.192	p(10) = 0.003
p(5) = 0.140	p(≥11) = 0.001

From this information, Table 9.6 is constructed. The value of E_1 is given by $np_0 = 100(0.026) = 2.6$. In a similar manner, the remaining E_i values are computed. Since $E_1 = 2.6 < 5$, E_1 and E_2 are combined. In that case, O_1 and O_2 are also combined, and k is reduced by one. The last five class intervals are also combined, for the same reason, and k is further reduced by four.

The calculated χ_0^2 is 27.68. The degrees of freedom for the tabulated value of χ^2 is $k - s - 1 = 7 - 1 - 1 = 5$. Here, $s = 1$, since one parameter, $\hat{\alpha}$ was estimated from the data. At the 0.05 level of significance, the critical value $\chi^2_{0.05,5}$ is 11.1. Thus, H_0 would be rejected at level of significance 0.05. The analyst, therefore, might want to search for a better-fitting model or use the empirical distribution of the data.

Table 9.6 Chi-square Goodness-of-Fit Test for Example 9.14

x_i	*Observed Frequency,* O_i		*Expected Frequency,* E_i		$\frac{(O_i - E_i)^2}{E_i}$
0	12	22	2.6	12.2	7.87
1	10		9.6		
2	19		17.4		0.15
3	17		21.1		0.80
4	10		19.2		4.41
5	8		14.0		2.57
6	7		8.5		0.26
7	5	17	4.4	7.6	11.62
8	5		2.0		
9	3		0.8		
10	3		0.3		
≥11	1		0.1		
	100		100.0		27.68

9.4.2 Chi-Square Test with Equal Probabilities

If a continuous distributional assumption is being tested, class intervals that are equal in probability rather than equal in width of interval should be used. This has been recommended by a number of authors [Mann and Wald, 1942; Gumbel, 1943; Law and Kelton, 2000; Stuart, Ord, and Arnold, 1998]. It should be noted that the procedure is not applicable to data collected in class intervals, where the raw data have been discarded or lost.

Unfortunately, there is as yet no method for figuring out the probability associated with each interval that maximizes the power for a test of a given size. (The power of a test is defined as the probability of rejecting a false hypothesis.) However, if using equal probabilities, then $pi = 1/k$. We recommend

$$E_i = np_i \geq 5$$

so substituting for p_i yields

$$\frac{n}{k} \geq 5$$

and solving for k yields

$$k \leq \frac{n}{5} \tag{9.18}$$

Equation (9.18) was used in coming up with the recommendations for maximum number of class intervals in Table 9.5.

If the assumed distribution is normal, exponential, or Weibull, the method described in this section is straightforward. Example 9.15 indicates how the procedure is accomplished for the exponential distribution. If the assumed distribution is gamma (but not Erlang) or certain other distributions, then the computation of endpoints for class intervals is complex and could require numerical integration of the density function. Statistical-analysis software is very helpful in such cases.

Example 9.15: Chi-Square Test for Exponential Distribution

In Example 9.11, the failure data presented in Example 9.3 were analyzed. The histogram of the data, shown in Figure 9.3, appeared to follow an exponential distribution, so the parameter $\hat{\lambda} = 1/\hat{X} = 0.084$ was computed. Thus, the following hypotheses are formed:

H_0: the random variable is exponentially distributed.
H_1: the random variable is not exponentially distributed.

In order to perform the chi-square test with intervals of equal probability, the endpoints of the class intervals must be found. Equation (9.18) indicates that the number of intervals should be less than or equal to $n/5$. Here, $n = 50$, and so $k \leq 10$. In Table 9.5, it is recommended that 7 to 10 class intervals be used. Let $k = 8$; then each interval will have probability $p = 0.125$. The endpoints for each interval are computed from the cdf for the exponential distribution, given in Equation (5.28), as follows:

$$F(a_i) = 1 - e^{-\lambda a_i} \tag{9.19}$$

where a_i represents the endpoint of the ith interval, $i = 1, 2, \ldots, k$. Since $F(a_i)$ is the cumulative area from zero to a_i, $F(a_i) = ip$, so Equation (9.19) can be written as

$$ip = 1 - e^{-\lambda a_i}$$

or

$$e^{-\lambda a_i} = 1 - ip$$

Taking the logarithm of both sides and solving for a_i gives a general result for the endpoints of k equiprobable intervals for the exponential distribution:

$$a_i = -\frac{1}{\lambda}\ln(1 - ip), \quad i = 0, 1, \ldots, k \tag{9.20}$$

Regardless of the value of λ, Equation (9.20) will always result in $a_0 = 0$ and $a_k = \infty$. With $\hat{\lambda} = 0.084$ and $k = 8$, a_1 is computed from Equation (9.20) as

$$a_1 = -\frac{1}{0.084}\ln(1 - 0.125) = 1.590$$

Continued application of Equation (9.20) for $i = 2, 3, \ldots, 7$ results in $a_2, \ldots, a_7$ as 3.425, 5.595, 8.252, 11.677, 16.503, and 24.755. Since $k = 8$, $a_8 = \infty$. The first interval is [0, 1.590), the second interval is [1.590, 3.425), and so on. The expectation is that 0.125 of the observations will fall in each interval. The observations, the expectations, and the contributions to the calculated value of χ_0^2 are shown in Table 9.7. The calculated value of χ_0^2 is 39.6. The degrees of freedom are given by $k - s - 1 = 8 - 1 - 1 = 6$. At $\alpha = 0.05$, the tabulated value of $\chi^2_{0.05,6}$ is 12.6. Since $\chi_0^2 > \chi^2_{0.05,6}$, the null hypothesis is rejected. (The value of $\chi^2_{0.01,6}$ is 16.8, so the null hypothesis would also be rejected at level of significance $\alpha = 0.01$.)

Table 9.7 Chi-Square Goodness-of-Fit Test for Example 9.15

Class Interval	*Observed Frequency,* O_i	*Expected Frequency,* E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 1.590)	19	6.25	26.01
[1.590, 3.425)	10	6.25	2.25
[3.425, 5.595)	3	6.25	0.81
[5.595, 8.252)	6	6.25	0.01
[8.252, 11.677)	1	6.25	4.41
[11.677, 16.503)	1	6.25	4.41
[16.503, 24.755)	4	6.25	0.81
[24.755, ∞)	6	6.25	0.01
	50	50	39.6

9.4.3 Kolmogorov–Smirnov Goodness-of-Fit Test

The chi-square goodness-of-fit test can accommodate the estimation of parameters from the data with a resultant decrease in the degrees of freedom (one for each parameter estimated). The chi-square test requires that the data be placed in class intervals; in the case of a continuous distributional assumption, this grouping is arbitrary. Changing the number of classes and the interval width affects the value of the calculated and tabulated chi-square. A hypothesis could be accepted when the data are grouped one way, but rejected when they are grouped another way. Also, the distribution of the chi-square test statistic is known only approximately, and the power of the test is sometimes rather low. As a result of these considerations, goodness-of-fit tests other than the chi-square, are desired. The Kolmogorov–Smirnov test formalizes the idea behind examining a $q - q$ plot.

The Kolmogorov–Smirnov test was presented in Section 7.4.1 to test for the uniformity of numbers. Both of these uses fall into the category of testing for goodness of fit. Any continuous distributional assumption can be tested for goodness of fit by using the method of Section 7.4.1.

The Kolmogorov–Smirnov test is particularly useful when sample sizes are small and when no parameters have been estimated from the data. When parameter estimates have been made, the critical values in Table A.8 are biased; in particular, they are too conservative. In this context, "conservative" means that the critical values will be too large, resulting in smaller Type I (α) errors than those specified. The exact value of α can be worked out in some instances, as is discussed at the end of this section.

The Kolmogorov–Smirnov test does not take any special tables when an exponential distribution is assumed. The following example indicates how the test is applied in this instance. (Notice that it is not necessary to estimate the parameter of the distribution in this example, so we may use Table A.8.)

Example 9.16: Kolmogorov–Smirnov Test for Exponential Distribution

Suppose that 50 interarrival times (in minutes) are collected over the following 100-minute interval (arranged in order of occurrence):

0.44	0.53	2.04	2.74	2.00	0.30	2.54	0.52	2.02	1.89	1.53	0.21
2.80	0.04	1.35	8.32	2.34	1.95	0.10	1.42	0.46	0.07	1.09	0.76
5.55	3.93	1.07	2.26	2.88	0.67	1.12	0.26	4.57	5.37	0.12	3.19
1.63	1.46	1.08	2.06	0.85	0.83	2.44	1.02	2.24	2.11	3.15	2.90
6.58	0.64										

The null hypothesis and its alternate are formed as follows:

H_0: the interarrival times are exponentially distributed.
H_1: the interarrival times are not exponentially distributed.

The data were collected over the interval from 0 to $T = 100$ minutes. It can be shown that, if the underlying distribution of interarrival times $\{T_1, T_2, \ldots\}$ is exponential, the arrival times are uniformly distributed on the interval $(0, T)$. The arrival times $T_1, T_1 + T_2, T_1 + T_2 + T_3, \ldots, T_1 + \cdots + T_{50}$ are obtained by adding interarrival times. The arrival times are then normalized to a (0, 1) interval so that the Kolmogorov–Smirnov test, as presented in Section 7.4.1, can be applied. On a (0, 1) interval, the points will be $[T_1/T, (T_1 + T_2)/T, \ldots, (T_1 + \cdots + T_{50})/T]$. The resulting 50 data points are as follows:

0.0044	0.0097	0.0301	0.0575	0.0775	0.0805	0.1059	0.1111	0.1313	0.1502
0.1655	0.1676	0.1956	0.1960	0.2095	0.2927	0.3161	0.3356	0.3366	0.3508
0.3553	0.3561	0.3670	0.3746	0.4300	0.4694	0.4796	0.5027	0.5315	0.5382
0.5494	0.5520	0.5977	0.6514	0.6526	0.6845	0.7008	0.7154	0.7262	0.7468
0.7553	0.7636	0.7880	0.7982	0.8206	0.8417	0.8732	0.9022	0.9680	0.9744

Following the procedure in Example 7.6 produces a D^+ of 0.1054 and a D^- of 0.0080. Therefore, the Kolmogorov–Smirnov statistic is $D = \max(0.1054, 0.0080) = 0.1054$. The critical value of D obtained from Table A.8 for a level of significance of $\alpha = 0.05$ and $n = 50$ is $D_{0.05} = 1.36/\sqrt{n} = 0.1923$; but $D = 0.1054$, so the hypothesis that the interarrival times are exponentially distributed cannot be rejected.

The Kolmogorov–Smirnov test has been modified so that it can be used in several situations where the parameters are estimated from the data. The computation of the test statistic is the same, but different tables of critical values are used. Different tables of critical values are required for different distributional assumptions. Lilliefors [1967] developed a test for normality. The null hypothesis states that the population is one of the family of normal distributions, without specifying the parameters of the distribution. The interested reader might wish to study Lilliefors' original work; he describes how simulation was used to develop the critical values.

Lilliefors [1969] also modified the critical values of the Kolmogorov–Smirnov test for the exponential distribution. Lilliefors again used random sampling to obtain approximate critical values, but Durbin [1975] subsequently obtained the exact distribution. Connover [1998] gives examples of Kolmogorov–Smirnov tests for the normal and exponential distributions. He also refers to several other Kolmogorov–Smirnov-type tests that might be of interest to the reader.

A test that is similar in spirit to the Kolmogorov–Smirnov test is the *Anderson–Darling test*. Like the Kolmogorov–Smirnov test, the Anderson–Darling test is based on the difference between the empirical cdf and the fitted cdf; unlike the Kolmogorov–Smirnov test, the Anderson–Darling test is based on a more comprehensive measure of difference (not just the maximum difference) and is more sensitive to discrepancies in the tails of the distributions. The critical values for the Anderson–Darling test also depend on the candidate distribution and on whether parameters have been estimated. Fortunately, this test and the Kolmogorov–Smirnov test have been implemented in a number of software packages that support simulation-input modeling.

9.4.4 p-Values and "Best Fits"

To apply a goodness-of-fit test, a significance level must be chosen. Recall that the significance level is the probability of falsely rejecting H_0: the random variable conforms to the distributional assumption. The traditional significance levels are 0.1, 0.05 and 0.01. Prior to the availability of high-speed computing, having a small set of standard values made it possible to produce tables of useful critical values. Now most statistical software computes critical values as needed, rather than storing them in tables. Thus, the analyst can employ a different level of significance—say, 0.07.

However, rather than require a prespecified significance level, many software packages compute a *p-value* for the test statistic. The *p*-value is the significance level at which one would *just reject* H_0 for the given value of the test statistic. Therefore, a large *p*-value tends to indicate a good fit (we would have to accept a large chance of error in order to reject), while a small *p*-value suggests a poor fit (to accept we would have to insist on almost no risk).

Recall Example 9.14, in which a chi-square test was used to check the Poisson assumption for the vehicle-arrival data. The value of the test statistic was $\chi_0^2 = 27.58$, with 5 degrees of freedom. The *p*-value for this test statistic is 0.00004, meaning that we would reject the hypothesis that the data are Poisson at the 0.00004 significance level. (Recall that we rejected the hypothesis at the 0.05 level; now we know that we would also to reject it at even lower levels.)

The *p*-value can be viewed as a measure of fit, with larger values being better. This suggests that we could fit every distribution at our disposal, compute a test statistic for each fit, and then choose the distribution that yields the largest *p*-value. We know of no input modeling software that implements this specific algorithm, but many such packages do include a "best fit" option, in which the software recommends an input model to the user after evaluating all feasible models. The software might also take into account other factors—such as whether the data are discrete or continuous, bounded or unbounded—but, in the end, some

summary measure of fit, like the p-value, is used to rank the distributions. There is nothing wrong with this, but there are several things to keep in mind:

1. The software might know nothing about the physical basis of the data, whereas that information can suggest distribution families that are appropriate. (See the list in Section 9.2.2.) Remember that the goal of input modeling is often to fill in gaps or smooth the data, rather than find an input model that conforms as closely as possible to the given sample.
2. Recall that both the Erlang and the exponential distributions are special cases of the gamma and that the exponential is also a special case of the more flexible Weibull. Automated best-fit procedures tend to choose the more flexible distributions (gamma and Weibull over Erlang and exponential), because the extra flexibility allows closer conformance to the data and a better summary measure of fit. But again, close conformance to the data does not always lead to the most appropriate input model.
3. A summary statistic, like the p-value, is just that, a summary measure. It says little or nothing about where the lack of fit occurs (in the body of the distribution, in the right tail, or in the left tail). A human, using graphical tools, can see where the lack of fit occurs and decide whether or not it is important for the application at hand.

Our recommendation is that automated distribution selection be used as one of several ways to suggest candidate distributions. Always inspect the automatic selection, using graphical methods, and remember that the final choice is yours.

9.5 FITTING A NONSTATIONARY POISSON PROCESS

Fitting a nonstationary Poisson process (NSPP) to arrival data is a difficult problem, in general, because we seldom have knowledge about the appropriate form of the arrival rate function $\lambda(t)$. (See Chapter 5, Section 5.5 for the definition of a NSPP). One approach is to choose a very flexible model with lots of parameters and fit it with a method such as maximum likelihood; see Johnson, Lee, and Wilson [1994] for an example of this approach. A second method, and the one we consider here, is to approximate the arrival rate as being constant over some basic interval of time, such as an hour, or a day, or a month, but varying from time interval to time interval. The problem then becomes choosing the basic time interval and estimating the arrival rate within each interval.

Suppose we need to model arrivals over a time period, say $[0, T]$. The approach that we describe is most appropriate when it is possible to observe the time period $[0, T]$ repeatedly and count arrivals. For instance, if the problem involves modeling the arrival of e-mail throughout the business day (8 A.M. to 6 P.M.), and we believe that the arrival rate is approximately constant over half-hour intervals, then we need to be able to count arrivals during half-hour intervals for several days. If it is possible to record actual arrival times, rather than counts, then actual arrival times are clearly better since they can later be grouped into any interval lengths we desire. However, we will assume from here on that only counts are available.

Divide the time period $[0, T]$ into k equal intervals of length $\Delta t = T/k$. For instance, if we are considering a 10-hour business day from 8 A.M. to 6 P.M. and if we allow the rate to change every half hour, then $T = 10$, $k = 20$, and $\Delta t = 1/2$. Over n periods of observation (e.g., n days), let C_{ij} be the number of arrivals that occurred during the ith time interval on the jth period of observation. In our example, C_{23} would be the number of arrivals from 8:30 A.M. to 9 A.M. (second half-hour period) on the third day of observation.

The estimated arrival rate during the ith time period, $(i-1)\Delta t < t \le i\,\Delta t$, is then just the average number of arrivals scaled by the length of the time interval:

$$\hat{\lambda}(t) = \frac{1}{n\Delta t}\sum_{j=1}^{n} C_{ij} \qquad (9.21)$$

Table 9.8 Monday E-mail Arrival Data for NSPP Example

	Number of Arrivals			
Time Period	*Day 1*	*Day 2*	*Day 3*	*Estimated Arrival Rate (arrivals/hour)*
8:00–8:30	12	14	10	24
8:30–9:00	23	26	32	54
9:00–9:30	27	19	32	52
9:30–10:00	20	13	12	30

After the arrival rates for each time interval have been estimated, adjacent intervals whose rates appear to be the same can be combined.

For instance, consider the e-mail arrival counts during the first two hours of the business day on three Mondays, shown in Table 9.8. The estimated arrival rate for 8:30–9:00 is

$$\frac{1}{3(1/2)}(23+26+32) = 54 \text{ arrivals/hour}$$

After seeing these results we might consider combining the interval 8:30–9:00 with the interval 9:00–9:30, because the rates are so similar. Note also that the goodness-of-fit tests described in the previous section can be applied to the data from each time interval individually, to check the Poisson approximation.

9.6 SELECTING INPUT MODELS WITHOUT DATA

Unfortunately, it is often necessary in practice to develop a simulation model—perhaps for demonstration purposes or a preliminary study—before any process data are available. In this case, the modeler must be resourceful in choosing input models and must carefully check the sensitivity of results to the chosen models.

There are a number of ways to obtain information about a process even if data are not available:

Engineering data: Often a product or process has performance ratings provided by the manufacturer (for example, the mean time to failure of a disk drive is 10000 hours; a laser printer can produce 8 pages/minute; the cutting speed of a tool is 1 cm/second; etc.). Company rules might specify time or production standards. These values provide a starting point for input modeling by fixing a central value.

Expert option: Talk to people who are experienced with the process or similar processes. Often, they can provide optimistic, pessimistic, and most-likely times. They might also be able to say whether the process is nearly constant or highly variable, and they might be able to define the source of variability.

Physical or conventional limitations: Most real processes have physical limits on performance—for example, computer data entry cannot be faster than a person can type. Because of company policies, there could be upper limits on how long a process may take. Do not ignore obvious limits or bounds that narrow the range of the input process.

The nature of the process: The description of the distributions in Section 9.2.2 can be used to justify a particular choice even when no data are available.

When data are not available, the uniform, triangular, and beta distributions are often used as input models. The uniform can be a poor choice, because the upper and lower bounds are rarely just as likely as the central

values in real processes. If, in addition to upper and lower bounds, a most-likely value can be given, then the triangular distribution can be used. The triangular distribution places much of its probability near the most-likely value, and much less near the extremes. (See Section 5.4.) If a beta distribution is used, then be sure to plot the density function of the selected distribution; the beta can take unusual shapes.

A useful refinement is obtained when a minimum, a maximum, and one or more "breakpoints" can be given. A breakpoint is an intermediate value together with a probability of being less than or equal to that value. The following example illustrates how breakpoints are used.

Example 9.17

For a production-planning simulation, the sales volume of various products is required. The salesperson responsible for product XYZ-123 says that no fewer than 1000 units will be sold (because of existing contracts) and no more than 5000 units will be sold (because that is the entire market for the product). Given her experience, she believes that there is a 90% chance of selling more than 2000 units, a 25% chance of selling more than 3500 units, and only a 1% chance of selling more than 4500 units.

Table 9.9 summarizes this information. Notice that the chances of exceeding certain sales goals have been translated into the cumulative probability of being less than or equal to those goals. With the information in this form, the method of Section 8.1.5 can be employed to generate simulation-input data.

When input models have been selected without data, it is especially important to test the sensitivity of simulation results to the distribution chosen. Check sensitivity not only to the center of the distribution, but also to the variability or limits. Extreme sensitivity of output results to the input model provides a convincing argument against making critical decisions based on the results and in favor of undertaking data collection.

For additional discussion of input modeling in the absence of data, see Pegden, Shannon, and Sadowski [1995].

9.7 MULTIVARIATE AND TIME-SERIES INPUT MODELS

In Sections 9.1–9.4, the random variables presented were considered to be independent of any other variables within the context of the problem. However, variables may be related, and, if the variables appear in a simulation model as inputs, the relationship should be investigated and taken into consideration.

Example 9.18

An inventory simulation includes the lead time and annual demand for industrial robots. An increase in demand results in an increase in lead time: The final assembly of the robots must be made according to the specifications of the purchaser. Therefore, rather than treat lead time and demand as independent random variables, a multivariate input model should be developed.

Table 9.9 Summary of Sales Information

i	*Interval (Sales)*	*Cumulative Frequency,* c_i
1	$1000 \le x \le 2000$	0.10
2	$2000 < x \le 3500$	0.75
3	$3500 < x \le 4500$	0.99
4	$4500 < x \le 5000$	1.00

Example 9.19

A simulation of the web-based trading site of a stock broker includes the time between arrivals of orders to buy and sell. Investors tend to react to what other investors are doing, so these buy and sell orders arrive in bursts. Therefore, rather than treat the time between arrivals as independent random variables, a time-series model should be developed.

We distinguish *multivariate input models* of a fixed, finite number of random variables (such as the two random variables *lead time* and *annual demand* in Example 9.18) from *time-series input models* of a (conceptually infinite) sequence of related random variables (such as the successive times between orders in Example 9.19). We will describe input models appropriate for these examples after reviewing two measures of dependence, the covariance and the correlation.

9.7.1 Covariance and Correlation

Let X_1 and X_2 be two random variables, and let $\mu_i = E(X_i)$ and $\sigma_i^2 = V(X_i)$ be the mean and variance of X_i, respectively. The *covariance* and *correlation* are measures of the linear dependence between X_1 and X_2. In other words, the covariance and correlation indicate how well the relationship between X_1 and X_2 is described by the model

$$(X_1 - \mu_1) = \beta(X_2 - \mu_2) + \epsilon$$

where ϵ is a random variable with mean 0 that is independent of X_2. If, in fact, $(X_1 - \mu_1) = \beta(X_2 - \mu_2)$, then this model is perfect. On the other hand, if X_1 and X_2 are statistically independent, then $\beta = 0$ and the model is of no value. In general, a positive value of β indicates that X_1 and X_2 tend to be above or below their means together; a negative value of β indicates that they tend to be on opposite sides of their means.

The covariance between X_1 and X_2 is defined to be

$$\text{cov}(X_1, X_2) = E[(X_1 - \mu_1)(X_2 - \mu_2)] = E(X_1 X_2) - \mu_1 \mu_2 \tag{9.22}$$

The value $\text{cov}(X_1, X_2) = 0$ implies $\beta = 0$ in our model of dependence, and $\text{cov}(X_1, X_2) < 0$ (>0) implies $\beta < 0$ (>0).

The covariance can take any value between $-\infty$ and ∞. The correlation standardizes the covariance to be between –1 and 1:

$$\rho = \text{corr}(X_1, X_2) = \frac{\text{cov}(X_1, X_2)}{\sigma_1 \sigma_2} \tag{9.23}$$

Again, the value $\text{corr}(X_1, X_2) = 0$ implies $\beta = 0$ in our model, and $\text{corr}(X_1, X_2) < 0$ (>0) implies $\beta < 0$ (>0). The closer ρ is to –1 or 1, the stronger the linear relationship is between X_1 and X_2.

Now suppose that we have a sequence of random variables $X_1, X_2, X_3, \ldots$ that are identically distributed (implying that they all have the same mean and variance), but could be dependent. We refer to such a sequence as a *time series* and to $\text{cov}(X_t, X_{t+h})$ and $\text{corr}(X_t, X_{t+h})$ as the *lag-h autocovariance* and *lag-h autocorrelation*, respectively. If the value of the autocovariance depends only on h and not on t, then we say that the time series is *covariance stationary*; this concept is discussed further in Chapter 11. For a covariance-stationary time series, we use the shorthand notation

$$\rho_h = \text{corr}(X_t, X_{t+h})$$

for the *lag-h* autocorrelation. Notice that autocorrelation measures the dependence between random variables that are separated by $h - 1$ others in the time series.

9.7.2 Multivariate Input Models

If X_1 and X_2 each are normally distributed, then dependence between them can be modeled by the bivariate normal distribution with parameters μ_1, μ_2, σ_1^2, σ_2^2, and $\rho = \text{corr}(X_1, X_2)$. Estimation of μ_1, μ_2, σ_1^2, and σ_2^2 was described in Section 9.3.2. To estimate ρ, suppose that we have n independent and identically distributed pairs (X_{11}, X_{21}), (X_{12}, X_{22}), ..., (X_{1n}, X_{2n}). Then the sample covariance is

$$\widehat{\text{cov}}(X_1, X_2) = \frac{1}{n-1}\sum_{j=1}^{n}\left(X_{1j} - \bar{X}_1\right)\left(X_{2j} - \bar{X}_2\right)$$

$$= \frac{1}{n-1}\left(\sum_{j=1}^{n} X_{1j}X_{2j} - n\bar{X}_1\bar{X}_2\right) \tag{9.24}$$

where $\bar{X}_1$ and $\bar{X}_2$ are the sample means. The correlation is estimated by

$$\hat{\rho} = \frac{\widehat{\text{cov}}(X_1, X_2)}{\hat{\sigma}_1\hat{\sigma}_2} \tag{9.25}$$

where $\hat{\sigma}_1$ and $\hat{\sigma}_2$ are the sample variances.

Example 9.20: Example 9.18 Continued

Let X_1 represent the average lead time to deliver (in months), and X_2 the annual demand, for industrial robots. The following data are available on demand and lead time for the last ten years:

lead time	demand
6.5	103
4.3	83
6.9	116
6.0	97
6.9	112
6.9	104
5.8	106
7.3	109
4.5	92
6.3	96

Standard calculations give $\bar{X}_1 = 6.14$, $\hat{\sigma}_1 = 1.02$, $\bar{X}_2 = 101.80$, and $\hat{\sigma}_2 = 9.93$ as estimates of μ_1, σ_1, μ_2, and σ_2, respectively. To estimate the correlation, we need

$$\sum_{j=1}^{10} X_{1j}X_{2j} = 6328.5$$

Therefore, $\widehat{\text{cov}} = [6328.5 - (10)(6.14)(101.80)]/(10-1) = 8.66$, and

$$\hat{\rho} = \frac{8.66}{(1.02)(9.93)} = 0.86$$

Clearly, lead time and demand are strongly dependent. Before we accept this model, however, lead time and demand should be checked individually to see whether they are represented well by normal distributions.

In particular, demand is a discrete-valued quantity, so the continuous normal distribution is certainly at best an approximation.

The following simple algorithm can be used to generate bivariate normal random variables:

Step 1. Generate Z_1 and Z_2, independent standard normal random variables (see Section 8.3.1).

Step 2. Set $X_1 = \mu_1 + \sigma_1 Z_1$

Step 3. Set $X_2 = \mu_2 + \sigma_2\left(\rho Z_1 + \sqrt{1-\rho^2}\,Z_2\right)$

Obviously, the bivariate normal distribution will not be appropriate for all multivariate-input modeling problems. It can be generalized to the k-variate normal distribution to model the dependence among more than two random variables, but, in many instances, a normal distribution is not appropriate in any form. We provide one method for handling nonnormal distributions in Section 9.7.4. Good references for other models are Johnson [1987] and Nelson and Yamnitsky [1998].

9.7.3 Time-Series Input Models

If $X_1, X_2, X_3, \ldots$ is a sequence of identically distributed, but dependent and covariance-stationary random variables, then there are a number of time series models that can be used to represent the process. We will describe two models that have the characteristic that the autocorrelations take the form

$$\rho_h = \text{corr}(X_t, X_{t+h}) = \rho^h$$

for $h = 1, 2, \ldots$ Notice that the *lag-h* autocorrelation decreases geometrically as the lag increases, so that observations far apart in time are nearly independent. For one model to be shown shortly, each X_t is normally distributed; for the other model, each X_t is exponentially distributed. More general time-series input models are described in Section 9.7.4 and in Nelson and Yamnitsky [1998].

AR(1) MODEL. Consider the time-series model

$$X_t = \mu + \phi(X_{t-1} - \mu) + \varepsilon_t \tag{9.26}$$

for $t = 2, 3, \ldots$, where $\varepsilon_2, \varepsilon_3, \ldots$ are independent and identically (normally) distributed with mean 0 and variance σ_ε^2, and $-1 < \phi < 1$. If the initial value X_1 is chosen appropriately (see shortly), then $X_1, X_2, \ldots$ are all normally distributed with mean μ, variance $\sigma_\varepsilon^2/(1-\phi^2)$, and

$$\rho_h = \phi^h$$

for $h = 1, 2, \ldots$. This time-series model is called the autoregressive order-1 model, or AR(1) for short.

Estimation of the parameter ϕ can be obtained from the fact that

$$\phi = \rho^1 = \text{corr}(X_t, X_{t+1})$$

the lag-1 autocorrelation. Therefore, to estimate ϕ, we first estimate the lag-1 autocovariance by

$$\widehat{\text{cov}}(X_t, X_{t+1}) = \frac{1}{n-1}\sum_{t=1}^{n-1}\left(X_t - \bar{X}\right)\left(X_{t+1} - \bar{X}\right)$$

$$\doteq \frac{1}{n-1}\left(\sum_{t=1}^{n-1} X_t X_{t+1} - (n-1)\bar{X}_2\right) \tag{9.27}$$

and the variance $\sigma^2 = \text{var}(X)$ by the usual estimator $\hat{\sigma}^2$. Then

$$\hat{\phi} = \frac{\widehat{\text{cov}}(X_t, X_{t+1})}{\hat{\sigma}^2}$$

Finally, estimate μ and σ_ε^2 by $\hat{\mu} = \bar{X}$ and

$$\hat{\sigma}_\varepsilon^2 = \hat{\sigma}^2(1 - \hat{\phi}^2)$$

respectively.

The following algorithm generates a stationary AR(1) time series, given values of the parameters ϕ, μ, and σ_ε^2:

Step 1. Generate X_1 from the normal distribution with mean μ and variance $\sigma_\varepsilon^2/(1-\phi^2)$. Set $t = 2$.

Step 2. Generate ε_t from the normal distribution with mean 0 and variance σ_ε^2.

Step 3. Set $X_t = \mu + \phi(X_{t-1} - \mu) + \varepsilon_t$.

Step 4. Set $t = t + 1$ and go to Step 2.

EAR(1) MODEL. Consider the time-series model

$$X_t = \begin{cases} \phi X_{t-1}, & \text{with probability } \phi \\ \phi X_{t-1} + \varepsilon_t, & \text{with probability } 1-\phi \end{cases} \tag{9.28}$$

for $t = 2, 3, \ldots$, where $\varepsilon_2, \varepsilon_3, \ldots$ are independent and identically (exponentially) distributed with mean $1/\lambda$ and $0 \le \phi < 1$. If the initial value X_1 is chosen appropriately (see shortly), then $X_1, X_2, \ldots$ are all exponentially distributed with mean $1/\lambda$ and

$$\rho_h = \phi^h$$

for $h = 1, 2, \ldots$. This time-series model is called the exponential autoregressive order-1 model, or EAR(1) for short. Only autocorrelations greater than 0 can be represented by this model. Estimation of the parameters proceeds as for the AR(1) by setting $\hat{\phi} = \hat{\rho}$, the estimated lag-1 autocorrelation, and setting $\hat{\lambda} = 1/\bar{X}$.

The following algorithm generates a stationary EAR(1) time series, given values of the parameters ϕ and λ:

Step 1. Generate X_1 from the exponential distribution with mean $1/\lambda$. Set $t = 2$.

Step 2. Generate U from the uniform distribution on [0, 1]. If $U \le \phi$, then set

$$X_t = \phi X_{t-1}$$

Otherwise, generate ε_t from the exponential distribution with mean $1/\lambda$ and set

$$X_t = \phi X_{t-1} + \varepsilon_t$$

Step 3. Set $t = t + 1$ and go to Step 2.

Example 9.21: Example 9.19 Continued

The stock broker would typically have a large sample of data, but, for the sake of illustration, suppose that the following twenty time gaps between customer buy and sell orders had been recorded (in seconds): 1.95, 1.75, 1.58, 1.42, 1.28, 1.15, 1.04, 0.93, 0.84, 0.75, 0.68, 0.61, 11.98, 10.79, 9.71, 14.02, 12.62, 11.36, 10.22, 9.20. Standard calculations give $\bar{X} = 5.2$ and $\hat{\sigma}^2 = 26.7$. To estimate the lag-1 autocorrelation, we need

$$\sum_{j=1}^{19} X_j X_{j+1} = 924.1$$

Thus, $\widehat{\text{cov}} = [924.1 - (20-1)(5.2)^2]/(20-1) = 21.6$, and

$$\hat{\rho} = \frac{21.6}{26.7} = 0.8$$

Therefore, we could model the interarrival times as an EAR(1) process with $\hat{\lambda} = 1/5.2 = 0.192$ and $\hat{\phi} = 0.8$, provided that an exponential distribution is a good model for the individual gaps.

9.7.4 The Normal-to-Anything Transformation

The bivariate normal distribution and the AR(1) and EAR(1) time-series models are useful input models that are easy to fit and simulate. However, the marginal distribution is either normal or exponential, which is certainly not the best choice for many applications. Fortunately, we can start with a bivariate normal or AR(1) model and *transform* it to have any marginal distributions we want (including exponential).

Suppose we want to simulate a random variable X with cdf $F(x)$. Let Z be a standard normal random variable (mean 0 and variance 1), and let $\Phi(z)$ be its cdf. Then it can be shown that

$$R = \Phi(Z)$$

is a $U(0, 1)$ random variable. As we learned in Chapter 8, if we have a $U(0, 1)$ random variable, we can get X by using the inverse cdf transformation

$$X = F^{-1}[R] = F^{-1}[\Phi(Z)]$$

We refer this as the *normal to anything transformation*, or NORTA for short.

Of course, if all we want is X, then there is no reason to go to this trouble; we can just generate R directly, using the methods in Chapter 8. But suppose we want a bivariate random vector (X_1, X_2) such that X_1 and X_2 are correlated but their distributions are not normal. Then we can start with a bivariate normal random vector (Z_1, Z_2) and apply the NORTA transformation to obtain

$$X_1 = F_1^{-1}[\Phi(Z_1)] \text{ and } X_2 = F_2^{-1}[\Phi(Z_2)]$$

There is not even a requirement that F_1 and F_2 be from the same distribution family; for instance, F_1 could be an exponential distribution and F_2 a beta distribution.

The same idea applies for time series. If Z_t is generated by an AR(1) with $N(0, 1)$ marginals, then

$$X_t = F^{-1}[\Phi(Z_t)]$$

will be a time-series model with marginal distribution $F(x)$. To insure that Z_i is $N(0, 1)$, we set $\mu = 0$ and $\sigma_\varepsilon^2 = 1 - \phi^2$ in the AR(1) model.

Although the NORTA method is very general, there are two technical issues that must be addressed to implement it:

1. The NORTA approach requires being able to evaluate that standard normal cdf, $\Phi(z)$, and the inverse cdf of the distributions of interest, $F^{-1}(u)$. There is no closed-form expression for $\Phi(z)$ and no closed-form expression for $F^{-1}(u)$ for many distributions. Therefore, numerical approximations are required. Fortunately, these functions are built into many symbolic calculation and spreadsheet programs, and we give one example next. In addition, Bratley, Fox, and Schrage [1987] contains algorithms for many distributions.
2. The correlation between the standard normal random variables (Z_1, Z_2) is distorted when it passes through the NORTA transformation. To be more specific, if (Z_1, Z_2) have correlation ρ, then in general $X_1 = F_1^{-1}[\Phi(Z_1)]$ and $X_2 = F_2^{-1}[\Phi(Z_2)]$ will have a correlation $\rho_X \neq \rho$. The difference is often small, but not always.

```
NORTARho := proc(rhoX, n)
local Z1, Z2, Ztemp, X1, X2, R1, R2, rho, rhoT, lower, upper;
randomize(123456);
Z1 := [random[normald[0,1]](n)]:
ZTemp := [random[normald[0,1]](n)]:
Z2 := [0]:
# set up bisection search
rho := rhoX:
if (rhoX < 0) then
    lower := -1:
    upper := 0:
else
    lower := 0:
    upper := 1:
fi:
Z2 := rho*Z1 + sqrt(1-rho^2)*ZTemp:
R1 := statevalf[cdf,normald[0,1]](Z1):
R2 := statevalf[cdf,normald[0,1]](Z2):
X1 := statevalf[icdf,exponential[1,0]](R1):
X2 := statevalf[icdf,beta[1,2]](R2):
rhoT := describe[linearcorrelation](X1, X2);
# do bisection search until 5% relative error
while abs(rhoT - rhoX)/abs(rhoX) > 0.05 do
    if (rhoT > rhoX) then
        upper := rho:
    else
        lower := rho:
    fi:
    rho := evalf((lower + upper)/2):
    Z2 := rho*Z1 + sqrt(1-rho^2)*ZTemp:
    R1 := statevalf[cdf,normald[0,1]](Z1):
    R2 := statevalf[cdf,normald[0,1]](Z2):
    X1 := statevalf[icdf,exponential[1,0]](R1):
    X2 := statevalf[icdf,beta[1,2]](R2):
    rhoT := describe[linearcorrelation](X1, X2);
end do;
RETURN(rho);
end;
```

Figure 9.6 Maple procedure to estimate the bivariate normal correlation required for the NORTA method.

The second issue is more critical, because in input-modeling problems we want to specify the bivariate or lag-1 correlation. Thus, we need to find the bivariate normal correlation ρ that gives us the input correlation ρ_X that we want (recall that we specify the time series model via the lag-1 correlation, $\rho_X = \text{corr}(X_i, X_{i+1})$). There has been much research on this problem, including Cario and Nelson [1996, 1998] and Biller and Nelson [2003]. Fortunately, it has been shown that ρ_X is a nondecreasing function of ρ, and ρ and ρ_X will always have the same sign. Thus, we can do a relatively simple search based on the following algorithm:

Step 1. Set $\rho = \rho_X$ to start.

Step 2. Generate a large number of bivariate normal pairs (Z_1, Z_2) with correlation ρ, and transform them into (X_1, X_2)'s, nsing the NORTA transformation.

Step 3. Compute the sample correlation between (X_1, X_2), using Equation (9.24), and call it $\hat{\rho}_T$. If $\hat{\rho}_T > \rho_X$, then reduce ρ and go to Step 2; if $\hat{\rho}_T < \rho_X$, then increase ρ and go to Step 2. If $\hat{\rho}_T \approx \rho_X$ then stop.

Example 9.22

Suppose we needed X_1 to have an exponential distribution with mean 1, X_2 to have a beta distribution with $\beta_1 = 1$, $\beta_2 = 1/2$, and the two of them to have correlation $\rho_X = 0.45$. Figure 9.6 shows a procedure in Maple that will estimate the required value of ρ. In the procedure, n is the number of sample pairs used to estimate the correlation. Running this procedure with n set to 1000 gives $\rho = 0.52$.

9.8 SUMMARY

Input-data collection and analysis require major time and resource commitments in a discrete-event simulation project. However, regardless of the validity or sophistication of the simulation model, unreliable inputs can lead to outputs whose subsequent interpretation could result in faulty recommendations.

This chapter discussed four steps in the development of models of input data: collecting the raw data, identifying the underlying statistical distribution, estimating the parameters, and testing for goodness of fit.

Some suggestions were given for facilitating the data-collection step. However, experience, such as that obtained by completing any of Exercises 1 through 5, will increase awareness of the difficulty of problems that can arise in data collection and of the need for planning.

Once the data have been collected, a statistical model should be hypothesized. Constructing a histogram is very useful at this point if sufficient data are available. A distribution based on the underlying process and on the shape of the histogram can usually be selected for further investigation.

The investigation proceeds with the estimation of parameters for the hypothesized distribution. Suggested estimators were given for distributions used often in simulation. In a number of instances, these are functions of the sample mean and sample variance.

The last step in the process is the testing of the distributional hypothesis. The $q - q$ plot is a useful graphical method for assessing fit. The Kolmogorov–Smirnov, chi-square, and Anderson–Darling goodness-of-fit tests can be applied to many distributional assumptions. When a distributional assumption is rejected, another distribution is tried. When all else fails, the empirical distribution could be used in the model.

Unfortunately, in some situations, a simulation study must be undertaken when there is not time or resources to collect data on which to base input models. When this happens, the analyst mnst use any available

information—such as manufacturer specifications and expert opinion—to construct the input models. When input models are derived without the benefit of data, it is particularly important to examine the sensitivity of the results to the models chosen.

Many, but not all, input processes can be represented as sequences of independent and identically distributed random variables. When inputs should exhibit dependence, then multivariate-input models are required. The bivariate normal distribution (and more generally the multivariate normal distribution) is often used to represent a finite number of dependent random variables. Time-series models are useful for representing a (conceptually infinite) sequence of dependent inputs. The NORTA transformation facilitates developing multivariate-input models with marginal distributions that are not normal.

REFERENCES

BILLER, B., AND B. L. NELSON [2003], "Modeling and Generating Multivariate Time Series with Arbitrary Marginals Using an Autoregressive Technique," *ACM Transactions on Modeling and Computer Simulation*, Vol. 13, pp. 211–237.

BRATLEY, P., B. L. FOX, AND L. E. SCHRAGE [1987], *A Guide to Simulation*, 2d ed., Springer-Verlag, New York.

CARIO, M. C., AND B. L. NELSON [1996], "Autoregressive to Anything: Time-Series Input Processes for Simulation," *Operations Research Letters*, Vol. 19, pp. 51–58.

CARIO, M. C., AND B. L. NELSON [1998], "Numerical Methods for Fitting and Simulating Autoregressive-to-Anything Processes," *INFORMS Journal on Computing*, Vol. 10, pp. 72–81.

CHOI, S. C., AND R. WETTE [1969], "Maximum Likelihood Estimation of the Parameters of the Gamma Distribution and Their Bias," *Technometrics*, Vol. 11, No. 4, pp. 683–890.

CHAMBERS, J. M., CLEVELAND, W. S., AND TUKEY, P. A. [1983], *Graphical Methods for Data Analysis*, CRC Press, Boca Raton, FL.

CONNOVER, W. J. [1998], *Practical Nonparametric Statistics*, 3d ed., Wiley, New York.

DURBIN, J. [1975], "Kolmogorov–Smirnov Tests When Parameters Are Estimated with Applications to Tests of Exponentiality and Tests on Spacings," *Biometrika*, Vol. 65, pp. 5–22.

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

GUMBEL, E. J. [1943], "On the Reliability of the Classical Chi-squared Test," *Annals of Mathematical Statistics*, Vol. 14, pp. 253ff.

HINES, W. W., D. C. MONTGOMERY, D. M. GOLDSMAN, AND C. M. BORROR [2002], *Probability and Statistics in Engineering and Management Science*, 4th ed., Wiley, New York.

JOHNSON, M. A., S. LEE, AND J. R. WILSON [1994], "NPPMLE and NPPSIM: Software for Estimating and Simulating Nonhomogeneous Poisson Processes Having Cyclic Behavior," *Operations Research Letters*, Vol. 15, pp. 273–282.

JOHNSON, M. E. [1987], *Multivariate Statistical Simulation*, Wiley, New York.

LAW, A. M., AND W. D. KELTON [2000], *Simulation Modeling & Analysis*, 3d ed., McGraw–Hill, New York.

LILLIEFORS, H. W. [1967], "On the Kolmogorov–Smirnov Test for Normality with Mean and Variance Unknown," *Journal of the American Statistical Association*, Vol. 62, pp. 339–402.

LILLIEFORS, H. W. [1969], "On the Kolmogorov–Smirnov Test for the Exponential Distribution with Mean Unknown," *Journal of the American Statistical Association*, Vol. 64, pp. 387–389.

MANN, H. B., AND A. WALD [1942], "On the Choice of the Number of Intervals in the Application of the Chi-squared Test," *Annals of Mathematical Statistics*, Vol. 18, p. 50ff.

NELSON, B. L., AND M. YAMNITSKY [1998], "Input Modeling Tools for Complex Problems," in *Proceedings of the 1998 Winter Simulation Conference*, eds. D. Medeiros, E. Watson, J. Carson, and M. Manivannan, pp. 105–112, The Institute for Electrical and Electronics Engineers, Piscataway, NJ.

PEGDEN, C. D., R. E. SHANNON, AND R. P. SADOWSKI [1995], *Introduction to Simulation Using SIMAN*, 2d ed. McGraw–Hill, New York.

STUART, A., J. K. ORD, AND E. ARNOLD [1998], *Kendall's Advanced Theory of Statistics*, 6th ed., Vol. 2, Oxford University Press, Oxford, NY.

EXERCISES

1. In a college library, collect the following information at the books return counter:

   arrival of students for returning books
   service time taken by the counter clerk

   Consolidate the data collected and verify whether it follows any standard distribution. (Prior permission from concerned authorities may be required.)

2. Go to a bank having single window operation. Collect information on arrival of customers, service time, etc. The type of transaction may vary from customer to customer. From service times observed, classify according to the type of transaction and fit arrival and service parameters separately for each type of transaction. (Prior permission from concerned authorities may be required.)

3. Go to a major traffic intersection, and record the interarrival-time distributions from each direction. Some arrivals want to go straight, some turn left, some turn right. The interarrival-time distribution varies during the day and by day of the week. Every now and then an accident occurs.

4. Go to a grocery store, and construct the interarrival and service distributions at the checkout counters. These distributions might vary by time of day and by day of week. Record, also, the number of service channels available at all times. (Make sure that the management gives permission to perform this study.)

5. Go to a laundromat, and "relive" the authors' data-collection experience discussed in Example 9.1. (Make sure that the management gives permission to perform this study.)

6. Draw the pdf of normal distribution with $\mu = 6$, $\sigma = 3$.

7. On one figure, draw the pdfs of the Erlang distribution where $\theta = 1/2$ and $k = 1, 2, 4$, and 8

8. On one figure, draw the pdfs of the Erlang distribution where $\theta = 2$ and $k = 1, 2, 4$, and 8.

9. Draw the pdf of Poisson distribution with $\alpha = 3$, 5, and 6.

10. Draw the exponential pdf with $\lambda = 0.5$. In the same sheet, draw the exponential pdf with $\lambda = 1.5$.

11. Draw the exponential pdf with $\lambda = 1$. In the same sheet, draw the exponential pdf with $\lambda = 3$.

12. The following data are generated randomly from a gamma distribution:

1.691	1.437	8.221	5.976
1.116	4.435	2.345	1.782
3.810	4.589	5.313	10.90
2.649	2.432	1.581	2.432
1.843	2.466	2.833	2.361

    Compute the maximum-likelihood estimators $\hat{\beta}$ and $\hat{\theta}$.

13. The following data are generated randomly from a Weibull distribution where $\nu = 0$:

7.936	5.224	3.937	6.513
4.599	7.563	7.172	5.132
5.259	2.759	4.278	2.696
6.212	2.407	1.857	5.002
4.612	2.003	6.908	3.326

Compute the maximum-likelihood estimators $\hat{\alpha}$ and $\hat{\beta}$. (This exercise requires a programmable calculator, a computer, or a lot of patience.)

14. Time between failures (in months) of a particular bearing is assumed to follow normal distribution. The data collected over 50 failures are

11.394	10.728	6.680	8.050	8.382
8.740	8.287	7.979	5.857	13.521
12.000	9.496	9.248	6.529	12.137
11.383	8.135	11.752	10.040	8.615
8.686	6.416	9.987	11.282	4.732
9.344	7.019	6.735	12.176	4.247
10.099	6.254	5.557	9.376	5.780
7.129	7.835	9.648	4.381	5.801
8.334	9.454	8.486	7.256	10.963
10.544	10.433	10.425	10.078	7.709

Using Kolmogorov–Smirnov test, check whether the distribution follows normal.

15. Show that the Kolmogorov–Smirnov test statistic for Example 9.16 is $D = 0.1054$.

16. Records pertaining to the monthly number of job-related injuries at an underground coalmine were being studied by a federal agency. The values for the past 100 months were as follows:

Injuries per Month	*Frequency of Occurrence*
0	35
1	40
2	13
3	6
4	4
5	1
6	1

(a) Apply the chi-square test to these data to test the hypothesis that the underlying distribution is Poisson. Use the level of significance $\alpha = 0.05$.
(b) Apply the chi-square test to these data to test the hypothesis that the distribution is Poisson with mean 1.0. Again let $\alpha = 0.05$.
(c) What are the differences between parts (a) and (b), and when might each case arise?

17. The interarrival time of tools for repair to a service station is assumed to follow exponential with $\lambda = 1$. The data collected from 50 such arrivals are

1.299	0.234	1.182	0.943	0.038
0.010	2.494	1.104	0.330	0.324
0.059	1.375	1.660	1.748	0.706
2.198	0.537	0.904	1.910	0.387
3.508	2.784	0.237	1.137	0.990

1.002	1.594	0.404	1.467	0.905
1.000	0.143	0.697	0.442	0.395
0.861	1.952	0.016	0.167	2.245
0.812	1.035	0.688	0.565	0.155
0.465	0.451	0.507	0.224	1.441

Based on appropriate test, check whether the assumption is valid.

18. The time spent by customers (in minutes) based on a study conducted in the college canteen is

13.125	12.972	18.985	12.041	14.658
14.151	17.541	17.251	13.400	15.559
16.365	18.946	11.154	11.159	14.883
13.650	15.336	16.990	18.265	18.719
13.763	18.518	16.493	15.869	13.291
16.643	16.712	12.759	14.926	14.412
21.285	13.299	16.589	13.887	15.853
12.995	19.540	17.761	16.290	14.624
14.300	8.497	19.149	14.035	17.076
18.778	11.186	16.263	14.438	15.741

Using appropriate methods, determine how the time is distributed.

19. The time required for the transmission of a message (in minutes) is sampled electronically at a communications center. The last 50 values in the sample are as follows:

7.936	4.612	2.407	4.278	5.132
4.599	5.224	2.003	1.857	2.696
5.259	7.563	3.937	6.908	5.002
6.212	2.759	7.172	6.513	3.326
8.761	4.502	6.188	2.566	5.515
3.785	3.742	4.682	4.346	5.359
3.535	5.061	4.629	5.298	6.492
3.502	4.266	3.129	1.298	3.454
5.289	6.805	3.827	3.912	2.969
4.646	5.963	3.829	4.404	4.924

How are the transmission times distributed? Develop and test an appropriate model.

20. The time spent (in minutes) by a customer in a bus stop awaiting to board a bus is

1.07	10.69	11.81	12.81	13.75
7.19	16.25	12.32	6.72	13.92
6.62	6.10	20.21	9.58	14.13
11.27	3.00	12.53	8.01	14.46
7.28	14.12	7.59	9.33	11.16

10.38	11.13	3.56	4.57	17.85
11.97	16.96	5.04	13.77	6.60
14.34	11.70	11.95	9.24	9.65
13.88	8.93	12.72	9.00	0.89
13.39	10.37	20.53	9.92	3.49

Using appropriate methods, determine how the time is distributed.

21. Daily demands for transmission overhaul kits for the D-3 dragline were maintained by Earth Moving Tractor Company, with the following results:

0	2	0	0	0
1	0	1	1	1
0	1	0	0	0
2	0	1	0	1
0	1	0	0	2
1	0	1	0	0
0	0	0	0	0
1	0	1	0	1
0	0	3	0	1
1	0	0	0	0

How are the daily demands distributed? Develop and test an appropriate model.

22. A simulation is to be conducted of a job shop that performs two operations: milling and planing, in that order. It would be possible to collect data about processing times for each operation, then generate random occurrences from each distribution. However, the shop manager says that the times might be related; large milling jobs take lots of planing. Data are collected for the next 25 orders, with the following results in minutes:

Order	*Milling Time (Minutes)*	*Planing Time (Minutes)*	*Order*	*Milling Time (Minutes)*	*Planing Time (Minutes)*
1	12.3	10.6	14	24.6	16.6
2	20.4	13.9	15	28.5	21.2
3	18.9	14.1	16	11.3	9.9
4	16.5	10.1	17	13.3	10.7
5	8.3	8.4	18	21.0	14.0
6	6.5	8.1	19	19.5	13.0
7	25.2	16.9	20	15.0	11.5
8	17.7	13.7	21	12.6	9.9
9	10.6	10.2	22	14.3	13.2
10	13.7	12.1	23	17.0	12.5
11	26.2	16.0	24	21.2	14.2
12	30.4	18.9	25	28.4	19.1
13	9.9	7.7			

(a) Plot milling time on the horizontal axis and planing time on the vertical axis. Do these data seem dependent?

(b) Compute the sample correlation between milling time and planing time.
(c) Fit a bivariate normal distribution to these data.

23. Write a computer program to compute the maximum-likelihood estimators ($\hat{\alpha}$, $\hat{\beta}$) of the Weibull distribution. Inputs to the program should include the sample size, n; the observations, $x_1, x_2, \ldots, x_n$; a stopping criterion, ϵ (stop when $|f(\hat{\beta}_j)| \leq \epsilon$); and a print option, OPT (usually set = 0). Output would be the estimates $\hat{\alpha}$ and $\hat{\beta}$. If OPT = 1, additional output would be printed, as in Table 9.4, showing convergence. Make the program as "user friendly" as possible.

24. Examine a computer-software library or simulation-support environment to which you have access. Obtain documentation on data-analysis software that would be useful in solving exercises 7 through 24. Use the software as an aid in solving selected problems.

25. The duration of calls in minutes over a telephone line is

2.058 6.407 0.565 0.641 5.989 0.435 0.278 3.447 11.461 1.658 2.913 2.689 4.747 2.587

Develop an input model for the call duration data.

26. The following data represent the time to perform transactions in a bank, measured in minutes: 0.740, 1.28, 1.46, 2.36, 0.354, 0.750, 0.912, 4.44, 0.114, 3.08, 3.24, 1.10, 1.59, 1.47, 1.17, 1.27, 9.12, 11.5, 2.42, 1.77. Develop an input model for these data.

27. Two types of jobs (A and B) are released to the input buffer of a job shop as orders arrive, and the arrival of orders is uncertain. The following data are available from the last week of production:

Day	*Number of Jobs*	*Number of A's*
1	83	53
2	93	62
3	112	66
4	65	41
5	78	55

Develop an input model for the number of new arrivals of each type each day.

28. The following data are available on the processing time at a machine (in minutes): 0.64, 0.59, 1.1, 3.3, 0.54, 0.04, 0.45, 0.25, 4.4, 2.7, 2.4, 1.1, 3.6, 0.61, 0.20, 1.0, 0.27, 1.7, 0.04, 0.34. Develop an input model for the processing time.

29. In the process of the development of an inventory simulation model, demand for a component is

1	2	3	4	3	5	4	3
4	4	6	6	5	4	6	4
5	7	5	5	7	1	5	2
3	4	3	4	2	8	7	2
3	8	4	4	5	3	1	6

Using appropriate model, identify how the demand is distributed.

30. Using the web, research some of the input-modeling software packages mentioned in this chapter. What are their features? What distributions do they include?

10

Verification and Validation of Simulation Models

One of the most important and difficult tasks facing a model developer is the verification and validation of the simulation model. The engineers and analysts who use the model outputs to aid in making design recommendations and the managers who make decisions based on these recommendations—justifiably look upon a model with some degree of skepticism about its validity. It is the job of the model developer to work closely with the end users throughout the period of development and validation to reduce this skepticism and to increase the model's credibility.

The goal of the validation process is twofold: (1) to produce a model that represents true system behavior closely enough for the model to be used as a substitute for the actual system for the purpose of experimenting with the system, analyzing system behavior, and predicting system performance; and (2) to increase to an acceptable level the credibility of the model, so that the model will be used by managers and other decision makers.

Validation should not be seen as an isolated set of procedures that follows model development, but rather as an integral part of model development. Conceptually, however, the verification and validation process consists of the following components:

1. Verification is concerned with building the model correctly. It proceeds by the comparison of the conceptual model to the computer representation that implements that conception. It asks the questions: Is the model implemented correctly in the simulation software? Are the input parameters and logical structure of the model represented correctly?
2. Validation is concerned with building the correct model. It attempts to confirm that a model is an accurate representation of the real system. Validation is usually achieved through the calibration of the model, an iterative process of comparing the model to actual system behavior and using the discrepancies between the two, and the insights gained, to improve the model. This process is repeated until model accuracy is judged to be acceptable.

This chapter describes methods that have been recommended and used in the verification and validation process. Most of the methods are informal subjective comparisons; a few are formal statistical procedures. The use of the latter procedures involves issues related to output analysis, the subject of Chapters 11 and 12. Output analysis refers to analysis of the data produced by a simulation and to drawing inferences from these data about the behavior of the real system. To summarize their relationship, validation is the process by which model users gain confidence that output analysis is making valid inferences about the real system under study.

Many articles and chapters in texts have been written on verification and validation. For discussion of the main issues, the reader is referred to Balci [1994, 1998, 2003], Carson [1986, 2002], Gass [1983], Kleijnen [1995], Law and Kelton [2000], Naylor and Finger [1967], Oren [1981], Sargent [2003], Shannon [1975], and van Horn [1969, 1971]. For statistical techniques relevant to various aspects of validation, the reader can obtain the foregoing references plus those by Balci and Sargent [1982a,b; 1984a], Kleijnen [1987], and Schruben [1980]. For case studies in which validation is emphasized, the reader is referred to Carson *et al.* [1981a,b], Gafarian and Walsh [1970], Kleijnen [1993], and Shechter and Lucas [1980]. Bibliographies on validation have been published by Balci and Sargent [1984b] and by Youngblood [1993].

10.1 MODEL BUILDING, VERIFICATION, AND VALIDATION

The first step in model building consists of observing the real system and the interactions among their various components and of collecting data on their behavior. But observation alone seldom yields sufficient understanding of system behavior. Persons familiar with the system, or any subsystem, should be questioned to take advantage of their special knowledge. Operators, technicians, repair and maintenance personnel, engineers, supervisors, and managers understand certain aspects of the system that might be unfamiliar to others. As model development proceeds, new questions may arise, and the model developers will return to this step of learning true system structure and behavior.

The second step in model building is the construction of a conceptual model—a collection of assumptions about the components and the structure of the system, plus hypotheses about the values of model input parameters. As is illustrated by Figure 10.1, conceptual validation is the comparison of the real system to the conceptual model.

The third step is the implementation of an operational model, usually by using simulation software and incorporating the assumptions of the conceptual model into the worldview and concepts of the simulation software. In actuality, model building is not a linear process with three steps. Instead, the model builder will return to each of these steps many times while building, verifying, and validating the model. Figure 10.1 depicts the ongoing model building process, in which the need for verification and validation causes continual comparison of the real system to the conceptual model and to the operational model and induces repeated modification of the model to improve its accuracy.

10.2 VERIFICATION OF SIMULATION MODELS

The purpose of model verification is to assure that the conceptual model is reflected accurately in the operational model. The conceptual model quite often involves some degree of abstraction about system operations or some amount of simplification of actual operations. Verification asks the following question: Is the conceptual model (assumptions about system components and system structure, parameter values, abstractions, and simplifications) accurately represented by the operational model?

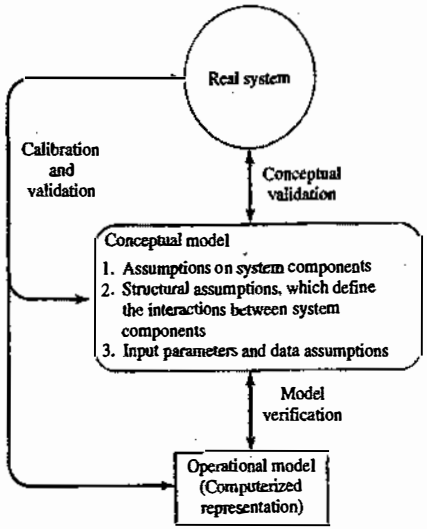


Figure 10.1 Model building, verification, and validation.

Many common-sense suggestions can be given for use in the verification process:

1. Have the operational model checked by someone other than its developer, preferably an expert in the simulation software being used.
2. Make a flow diagram that includes each logically possible action a system can take when an event occurs, and follow the model logic for each action for each event type. (An example of a logic flow diagram is given in Figures 2.2 and 2.3 for the model of a single-server queue.)
3. Closely examine the model output for reasonableness under a variety of settings of the input parameters. Have the implemented model display a wide variety of output statistics, and examine all of them closely.
4. Have the operational model print the input parameters at the end of the simulation, to be sure that these parameter values have not been changed inadvertently.
5. Make the operational model as self-documenting as possible. Give a precise definition of every variable used and a general description of the purpose of each submodel, procedure (or major section of code), component, or other model subdivision.
6. If the operational model is animated, verify that what is seen in the animation imitates the actual system. Examples of errors that can be observed through animation are automated guided vehicles (AGVs) that pass through one another on a unidirectional path or at an intersection and entities that disappear (unintentionally) during a simulation.
7. The Interactive Run Controller (IRC) or debugger is an essential component of successful simulation model building. Even the best of simulation analysts makes mistakes or commits logical errors when building a model. The IRC assists in finding and correcting those errors in the following ways:
   (a) The simulation can be monitored as it progresses. This can be accomplished by advancing the simulation until a desired time has elapsed, then displaying model information at that time. Another possibility is to advance the simulation until a particular condition is in effect, and then display information.

   (b) Attention can be focused on a particular entity line, of code, or procedure. For instance, every time that an entity enters a specified procedure, the simulation will pause so that information can be gathered. As another example, every time that a specified entity becomes active, the simulation will pause.
   (c) Values of selected model components can be observed. When the simulation has paused, the current value or status of variables, attributes, queues, resources, counters, and so on can be observed.
   (d) The simulation can be temporarily suspended, or paused, not only to view information, but also to reassign values or redirect entities.
8. Graphical interfaces are recommended for accomplishing verification and validation [Borts-cheller and Saulnier, 1992]. The graphical representation of the model is essentially a form of self-documentation. It simplifies the task of understanding the model.

These suggestions are basically the same ones any software engineer would follow.

Among these common-sense suggestions, one that is very easily implemented, but quite often overlooked, especially by students who are learning simulation, is a close and thorough examination of model output for reasonableness (suggestion 3). For example, consider a model of a complex network of queues consisting of many service centers in series and parallel configurations. Suppose that the modeler is interested mainly in the response time, defined as the time required for a customer to pass through a designated part of the network. During the verification (and calibration) phase of model development, it is recommended that the program collect and print out many statistics in addition to response times, such as utilizations of servers and time-average number of customers in various subsystems. Examination of the utilization of a server, for example, might reveal that it is unreasonably low (or high), a possible error that could be caused by wrong specification of mean service time, or by a mistake in model logic that sends too few (or too many) customers to this particular server, or by any number of other possible parameter misspecifications or errors in logic.

In a simulation language that automatically collects many standard statistics (average queue lengths, average waiting times, etc.), it takes little or no extra programming effort to display almost all statistics of interest. The effort required can be considerably greater in a general-purpose language such as Java, C, or C++, which do not have statistics-gathering capabilities to aid the programmer.

Two sets of statistics that can give a quick indication of model reasonableness are *current contents* and *total count*. These statistics apply to any system having items of some kind flowing through it, whether these items be called customers, transactions, inventory, or vehicles. "Current contents" refers to the number of items in each component of the system at a given time. "Total count" refers to the total number of items that have entered each component of the system by a given time. In some simulation software, these statistics are kept automatically and can be displayed at any point in simulation time. In other simulation software, simple counters might have to be added to the operational model and displayed at appropriate times. If the current contents in some portion of the system are high, this condition indicates that a large number of entities are delayed. If the output is displayed for successively longer simulation run times and the current contents tend to grow in a more or less linear fashion, it is highly likely that a queue is unstable and that the server(s) will fall further behind as time continues. This indicates possibly that the number of servers is too small or that a service time is misspecified. (Unstable queues were discussed in Chapter 6.) On the other hand, if the total count for some subsystem is zero, this indicates that no items entered that subsystem—again, a highly suspect occurrence. Another possibility is that the current count and total count are equal to one. This could indicate that an entity has captured a resource, but never freed that resource. Careful evaluation of these statistics for various run lengths can aid in the detection of mistakes in model logic and data misspecifications. Checking for output reasonableness will usually fail to detect the more subtle errors, but it is one of the quickest ways to discover gross errors. To aid in error detection, it is best for the model developer to forecast a reasonable range for the value of selected output statistics before making a run of the model. Such a forecast reduces the possibility of rationalizing a discrepancy and failing to investigate the cause of unusual output.

For certain models, it is possible to consider more than whether a particular statistic is reasonable. It is possible to compute certain long-run measures of performance. For example, as seen in Chapter 6, the analyst can compute the long-run server utilization for a large number of queueing systems without any special assumptions regarding interarrival or service-time distributions. Typically, the only information needed is the network configuration, plus arrival and service rates. Any measure of performance that can be computed analytically and then compared to its simulated counterpart provides another valuable tool for verification. Presumably, the objective of the simulation is to estimate some measure of performance, such as mean response time, that cannot be computed analytically; but, as illustrated by the formulas in Chapter 6 for a number of special queues ($M/M/1$, $M/G/1$, etc.), all the measures of performance in a queueing system are interrelated. Thus, if a simulation model is predicting one measure (such as utilization) correctly, then confidence in the model's predictive ability for other related measures (such as response time) is increased (even though the exact relation between the two measures is, of course, unknown in general and varies from model to model). Conversely, if a model incorrectly predicts utilization, its prediction of other quantities, such as mean response time, is highly suspect.

Another important way to aid the verification process is the oft-neglected documentation phase. If a model builder writes brief comments in the operational model, plus definitions of all variables and parameters, plus descriptions of each major section of the operational model, it becomes much simpler for someone else, or the model builder at a later date, to verify the model logic. Documentation is also important as a means of clarifying the logic of a model and verifying its completeness.

A more sophisticated technique is the use of a trace. In general, a trace is a detailed computer printout which gives the value of every variable (in a specified set of variables) in a computer program, every time that one of these variables changes in value. A trace designed specifically for use in a simulation program would give the value of selected variables each time the simulation clock was incremented (i.e., each time an event occurred). Thus, a simulation trace is nothing more than a detailed printout of the state of the simulation model as it changes over time.

Example 10.1

When verifying the operational model (in a general purpose language such as FORTRAN, Pascal, C or C++, or most simulation languages) of the single-server queue model of Example 2.1, an analyst made a run over 16 units of time and observed that the time-average length of the waiting line was $\hat{L}_Q = 0.4375$ customer, which is certainly reasonable for a short run of only 16 time units. Nevertheless, the analyst decided that a more detailed verification would be of value.

The trace in Figure 10.2 gives the hypothetical printout from simulation time CLOCK = 0 to CLOCK = 16 for the simple single-server queue of Example 2.1. This example illustrates how an error can be found with a trace, when no error was apparent from the examination of the summary output statistics (such as $\hat{L}_Q$). Note that, at simulation time CLOCK = 3, the number of customers in the system is NCUST = 1, but the server is idle (STATUS = 0). The source of this error could be incorrect logic, or simply not setting the attribute STATUS to the value 1 (when coding in a general purpose language or most simulation languages).

In any case, the error must be found and corrected. Note that the less sophisticated practice of examining the summary measures, or output, did not detect the error. By using equation (6.1), the reader can verify that $\hat{L}_Q$ was computed correctly from the data ($\hat{L}_Q$ is the time-average value of NCUST minus STATUS):

$$\hat{L}_Q = \frac{(0-0)3+(1-0)2+(0-0)6+(1-0)1+(2-1)4}{3+2+6+1+4}$$

$$= \frac{7}{16} = 0.4375$$

as previously mentioned. Thus, the output measure, $\hat{L}_Q$, had a reasonable value and was computed correctly from the data, but its value was indeed wrong because the attribute STATUS was not assuming correct

Definition of Variables:

CLOCK = Simulation clock
EVTYP = Event type (start, arrival, departure, or stop)
NCUST = Number of customers in system at time 'CLOCK'
STATUS = Status of server (1–busy, 0–idle)

State of System Just After the Named Event Occurs:

CLOCK = 0	EVTYP = 'Start'	NCUST = 0	STATUS = 0
CLOCK = 3	EVTYP = 'Arrival'	NCUST = 1	STATUS = 0
CLOCK = 5	EVTYP = 'Depart'	NCUST = 0	STATUS = 0
CLOCK = 11	EVTYP = 'Arrival'	NCUST = 1	STATUS = 0
CLOCK = 12	EVTYP = 'Arrival'	NCUST = 2	STATUS = 1
CLOCK = 16	EVTYP = 'Depart'	NCUST = 1	STATUS = 1

Figure 10.2 Simulation Trace of Example 2.1.

values. As is seen from Figure 10.2, a trace yields information on the actual history of the model that is more detailed and informative than the summary measures alone.

Most simulation software has a built-in capability to conduct a trace without the programmer having to do any extensive programming. In addition, a 'print' or 'write' statement can be used to implement a tracing capability in a general-purpose language.

As can be easily imagined, a trace over a large span of simulation time can quickly produce an extremely large amount of computer printout, which would be extremely cumbersome to check in detail for correctness. The purpose of the trace is to verify the correctness of the computer program by making detailed paper-and-pencil calculations. To make this practical, a simulation with a trace is usually restricted to a very short period of time. It is desirable, of course, to ensure that each type of event (such as ARRIVAL) occurs at least once, so that its consequences and effect on the model can be checked for accuracy. If an event is especially rare in occurrence, it may be necessary to use artificial data to force it to occur during a simulation of short duration. This is legitimate, as the purpose is to verify that the effect on the system of the rare event is as intended.

Some software allows a selective trace. For example, a trace could be set for specific locations in the model or could be triggered to begin at a specified simulation time. Whenever an entity goes through the designated locations, the simulation software writes a time-stamped message to a trace file. Some simulation software allows tracing a selected entity; any time the designated entity becomes active, the trace is activated and time-stamped messages are written. This trace is very useful in following one entity through the entire model. Another example of a selective trace is to set it for the occurrence of a particular condition. For example, whenever the queue before a certain resource reaches five or more, turn on the trace. This allows running the simulation until something unusual occurs, then examining the behavior from that point forward in time. Different simulation software packages support tracing to various extents. In practice, it is often implemented by the model developer by adding printed messages at appropriate points into a model.

Of the three classes of techniques—the common-sense techniques, thorough documentation, and traces—it is recommended that the first two always be carried out. Close examination of model output for reasonableness is especially valuable and informative. A generalized trace may provide voluminous data, far more than can be used or examined carefully. A selective trace can provide useful information on key model components and keep the amount of data to a manageable level.

10.3 CALIBRATION AND VALIDATION OF MODELS

Verification and validation, although conceptually distinct, usually are conducted simultaneously by the modeler. Validation is the overall process of comparing the model and its behavior to the real system and its behavior. Calibration is the iterative process of comparing the model to the real system, making adjustments (or even major changes) to the model, comparing the revised model to reality, making additional adjustments, comparing again, and so on. Figure 10.3 shows the relationship of model calibration to the overall validation process. The comparison of the model to reality is carried out by a variety of tests—some subjective, others objective. Subjective tests usually involve people, who are knowledgeable about one or more aspects of the system, making judgments about the model and its output. Objective tests always require data on the system's behavior, plus the corresponding data produced by the model. Then one or more statistical tests are performed to compare some aspect of the system data set with the same aspect of the model data set. This iterative process of comparing model with system and then revising both the conceptual and operational models to accommodate any perceived model deficiencies is continued until the model is judged to be sufficiently accurate.

A possible criticism of the calibration phase, were it to stop at this point, is that the model has been validated only for the one data set used—that is, the model has been "fitted" to one data set. One way to alleviate this criticism is to collect a new set of system data (or to reserve a portion of the original system data) to be used at this final stage of validation. That is, after the model has been calibrated by using the original system data set, a "final" validation is conducted, using the second system data set. If unacceptable discrepancies between the model and the real system are discovered in the "final" validation effort, the modeler must return to the calibration phase and modify the model until it becomes acceptable.

Validation is not an either/or proposition—no model is ever totally representative of the system under study. In addition, each revision of the model, as pictured in Figure 10.3, involves some cost, time, and effort. The modeler must weigh the possible, but not guaranteed, increase in model accuracy versus the cost of increased validation effort. Usually, the modeler (and model users) have some maximum discrepancy between model predictions and system behavior that would be acceptable. If this level of accuracy cannot be obtained within the budget constraints, either expectations of model accuracy must be lowered, or the model must be abandoned.

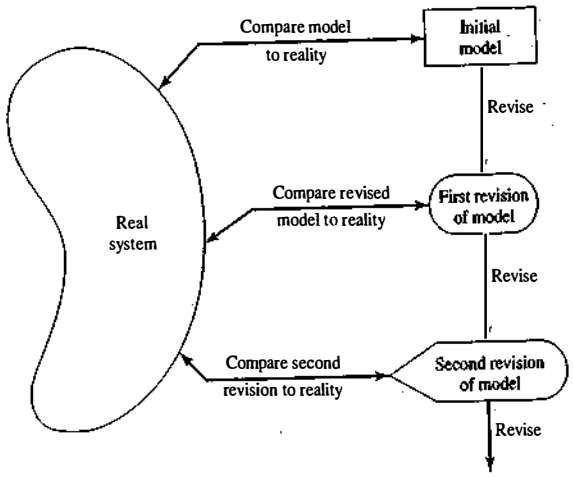


Figure 10.3 Iterative process of calibrating a model.

As an aid in the validation process, Naylor and Finger [1967] formulated a three-step approach that has been widely followed:

1. Build a model that has high face validity.
2. Validate model assumptions.
3. Compare the model input–output transformations to corresponding input–output transformations for the real system.

The next five subsections investigate these three steps in detail.

10.3.1 Face Validity

The first goal of the simulation modeler is to construct a model that appears reasonable on its face to model users and others who are knowledgeable about the real system being simulated. The potential users of a model should be involved in model construction from its conceptualization to its implementation, to ensure that a high degree of realism is built into the model through reasonable assumptions regarding system structure and through reliable data. Potential users and knowledgeable persons can also evaluate model output for reasonableness and can aid in identifying model deficiencies. Thus, the users can be involved in the calibration process as the model is improved iteratively by the insights gained from identification of the initial model deficiencies. Another advantage of user involvement is the increase in the model's perceived validity, or credibility, without which a manager would not be willing to trust simulation results as a basis for decision making.

Sensitivity analysis can also be used to check a model's face validity. The model user is asked whether the model behaves in the expected way when one or more input variables is changed. For example, in most queueing systems, if the arrival rate of customers (or demands for service) were to increase, it would be expected that utilizations of servers, lengths of lines, and delays would tend to increase (although by how much might well be unknown). From experience and from observations on the real system (or similar related systems), the model user and model builder would probably have some notion at least of the direction of change in model output when an input variable is increased or decreased. For most large-scale simulation models, there are many input variables and thus many possible sensitivity tests. The model builder must attempt to choose the most critical input variables for testing if it is too expensive or time consuming to vary all input variables. If real system data are available for at least two settings of the input parameters, objective scientific sensitivity tests can be conducted via appropriate statistical techniques.

10.3.2 Validation of Model Assumptions

Model assumptions fall into two general classes: structural assumptions and data assumptions. Structural assumptions involve questions of how the system operates and usually involve simplifications and abstractions of reality. For example, consider the customer queueing and service facility in a bank. Customers can form one line, or there can be an individual line for each teller. If there are many lines, customers could be served strictly on a first-come–first-served basis, or some customers could change lines if one line is moving faster. The number of tellers could be fixed or variable. These structural assumptions should be verified by actual observation during appropriate time periods and by discussions with managers and tellers regarding bank policies and actual implementation of these policies.

Data assumptions should be based on the collection of reliable data and correct statistical analysis of the data. (Example 9.1 discussed similar issues for a model of a laundromat.) For example, in the bank study previously mentioned, data were collected on

1. interarrival times of customers during several 2-hour periods of peak loading ("rush-hour" traffic);
2. interarrival times during a slack period;

3. service times for commercial accounts;
4. service times for personal accounts.

The reliability of the data was verified by consultation with bank managers, who identified typical rush hours and typical slack times. When combining two or more data sets collected at different times, data reliability can be further enhanced by objective statistical tests for homogeneity of data. (Do two data sets $\{X_i\}$ and $\{Y_i\}$ on service times for personal accounts, collected at two different times, come from the same parent population? If so, the two sets can be combined.) Additional tests might be required, to test for correlation in the data. As soon as the analyst is assured of dealing with a random sample (i.e., correlation is not present), the statistical analysis can begin.

The procedures for analyzing input data from a random sample were discussed in detail in Chapter 9. Whether done manually or by special-purpose software, the analysis consists of three steps:

1. Identify an appropriate probability distribution.
2. Estimate the parameters of the hypothesized distribution.
3. Validate the assumed statistical model by a goodness-of-fit test, such as the chi-square or Kolmogorov–Smirnov test, and by graphical methods.

The use of goodness-of-fit tests is an important part of the validation of data assumptions.

10.3.3 Validating Input–Output Transformations

The ultimate test of a model, and in fact the only objective test of the model as a whole, is the model's ability to predict the future behavior of the real system when the model input data match the real inputs and when a policy implemented in the model is implemented at some point in the system. Furthermore, if the level of some input variables (e.g., the arrival rate of customers to a service facility) were to increase or decrease, the model should accurately predict what would happen in the real system under similar circumstances. In other words, the structure of the model should be accurate enough for the model to make good predictions, not just for one input data set, but for the range of input data sets that are of interest.

In this phase of the validation process, the model is viewed as an input–output transformation—that is, the model accepts values of the input parameters and transforms these inputs into output measures of performance. It is this correspondence that is being validated.

Instead of validating the model input–output transformations by predicting the future, the modeler could use historical data that have been reserved for validation purposes only—that is, if one data set has been used to develop and calibrate the model, it is recommended that a separate data set be used as the final validation test. Thus, accurate "prediction of the past" can replace prediction of the future for the purpose of validating the model.

A model is usually developed with primary interest in a specific set of system responses to be measured under some range of input conditions. For example, in a queueing system, the responses may be server utilization and customer delay, and the range of input conditions (or input variables) may include two or three servers at some station and a choice of scheduling rules. In a production system, the response may be throughput (i.e., production per hour), and the input conditions may be a choice of several machines that run at different speeds, with each machine having its own breakdown and maintenance characteristics.

In any case, the modeler should use the main responses of interest as the primary criteria for validating a model. If the model is used later for a purpose different from its original purpose, the model should be revalidated in terms of the new responses of interest and under the possibly new input conditions.

A necessary condition for the validation of input–output transformations is that some version of the system under study exist, so that system data under at least one set of input conditions can be collected to compare to model predictions. If the system is in the planning stages and no system operating data can be collected, complete input–output validation is not possible. Other types of validation should be conducted, to the extent possible. In some cases, subsystems of the planned system may exist, and a partial input–output validation can be conducted.

Presumably, the model will be used to compare alternative system designs or to investigate system behavior under a range of new input conditions. Assume for now that some version of the system is operating and that the model of the existing system has been validated. What, then, can be said about the validity of the model when different inputs are used?—that is, if model inputs are being changed to represent a new system design, or a new way to operate the system, or even hypothesized future conditions, what can be said about the validity of the model with respect to this new but nonexistent proposed system or to the system under new input conditions?

First, the responses of the two models under similar input conditions will be used as the criteria for comparison of the existing system to the proposed system. Validation increases the modeler's confidence that the model of the existing system is accurate. Second, in many cases, the proposed system is a modification of the existing system, and the modeler hopes that confidence in the model of the existing system can be transferred to the model of the new system. This transfer of confidence usually can be justified if the new model is a relatively minor modification of the old model in terms of changes to the operational model (it may be a major change for the actual system). Changes in the operational model ranging from relatively minor to relatively major include the following:

1. minor changes of single numerical parameters, such as the speed of a machine, the arrival rate of customers (with no change in distributional form of interarrival tiines), the number of servers in a parallel service center, or the mean time to failure or mean time to repair of a machine;
2. minor changes of the form of a statistical distribution, such as the distribution of a service time or a time to failure of a machine;
3. major changes in the logical structure of a subsystem, such as a change in queue discipline for a waiting-line model or a change in the scheduling rule for a job-shop model;
4. major changes involving a different design for the new system, such as a computerized inventory control system replacing an older noncomputerized system, or an automated storage-and-retrieval system replacing a warehouse system in which workers pick items manually using fork trucks.

If the change to the operational model is minor, such as in items 1 or 2, these changes can be carefully verified and output from the new model accepted with considerable confidence. If a sufficiently similar subsystem exists elsewhere, it might be possible to validate the submodel that represents the subsystem and then to integrate this submodel with other validated submodels to build a complete model. In this way, partial validation of the substantial model changes in items 3 and 4 might be possible. Unfortunately, there is no way to validate the input–output transformations of a model of a nonexisting system completely. In any case, within time and budget constraints, the modeler should use as many validation techniques as possible, including input–output validation of subsystem models if operating data can be collected on such subsystems.

Example 10.2 will illustrate some of the techniques that are possible for input–output validation and will discuss the concepts of an input variable, uncontrollable variable, decision variable, output or response variable, and input–output transformation in more detail.

Example 10.2: The Fifth National Bank of Jaspar

The Fifth National Bank of Jaspar, as shown in Figure 10.4, is planning to expand its drive-in service at the corner of Main Street. Currently, there is one drive-in window serviced by one teller. Only one or two transactions are allowed at the drive-in window, so it was assumed that each service time was a random sample from some underlying population. Service times $\{S_i, i = 1, 2,..., 90\}$ and interarrival times $\{A_i, i = 1, 2,..., 90\}$ were collected for the 90 customers who arrived between 11:00 A.M. and 1:00 P.M. on a Friday. This time slot

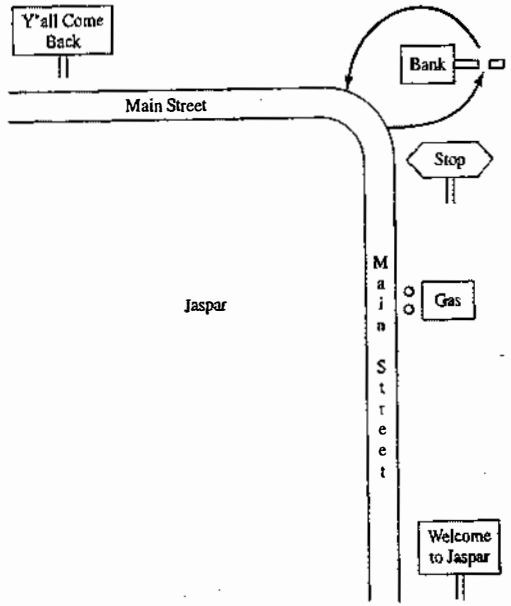


Figure 10.4 Drive-in window at the Fifth National Bank.

was selected for data collection after consultation with management and the teller because it was felt to be representative of a typical rush hour.

Data analysis (as outlined in Chapter 9) led to the conclusion that arrivals could be modeled as a Poisson process at a rate of 45 customers per hour and that service times were approximately normally distributed, with mean 1.1 minutes and standard deviation 0.2 minute. Thus, the model has two input variables:

1. interarrival times, exponentially distributed (i.e., a Poisson arrival process) at rate $\lambda = 45$ per hour;
2. service times, assumed to be $N(1.1, (0.2)^2)$.

Each input variable has a level: the rate ($\lambda = 45$ per hour) for the interarrival times, and the mean 1.1 minutes and standard deviation 0.2 minute for the service times. The interarrival times are examples of uncontrollable variables (i.e., uncontrollable by management in the real system). The service times are also treated as uncontrollable variables, although the level of the service times might be partially controllable. If the mean service time could be decreased to 0.9 minute by installing a computer terminal, the level of the service-time variable becomes a decision variable or controllable parameter. Setting all decision variables at some level constitutes a policy. For example, the current bank policy is one teller ($D_1 = 1$), mean service time $D_2 = 1.1$ minutes, and one line for waiting cars ($D_3 = 1$). (D_1, D_2, ... are used to denote decision variables.) Decision variables are under management's control; the uncontrollable variables, such as arrival rate and actual arrival times, are not under management's control. The arrival rate might change from time to time, but such change is treated as being due to external factors not under management control.

A model of current bank operations was developed and verified in close consultation with bank management and employees. Model assumptions were validated, as discussed in Section 10.3.2. The resulting

model is now viewed as a "black box" that takes all input-variable specifications and transforms them into a set of output or response variables. The output variables consist of all statistics of interest generated by the simulation about the model's behavior. For example, management is interested in the teller's utilization at the drive-in window (percent of time the teller is busy at the window), average delay in minutes of a customer from arrival to beginning of service, and the maximum length of the line during the rush hour. These input and output variables are shown in Figure 10.5 and are listed in Table 10.1, together with some additional output variables. The uncontrollable input variables are denoted by X, the decision variables by D, and the

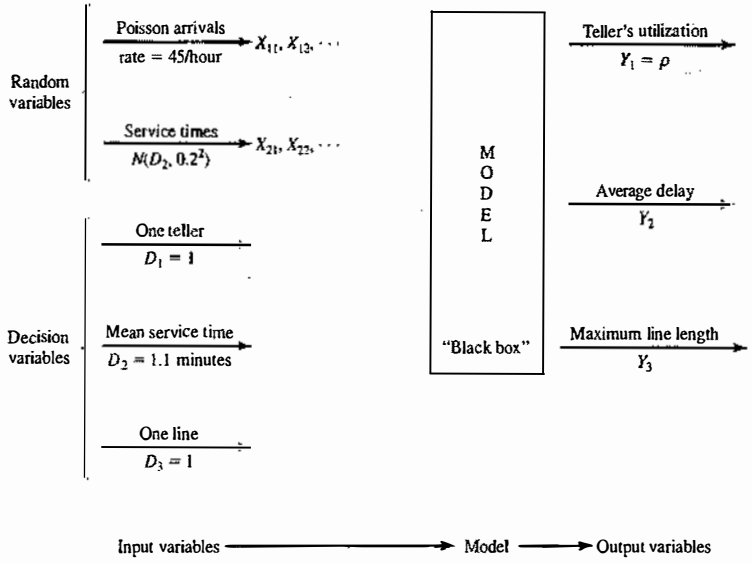


Figure 10.5 Model input–output transformation.

Table 10.1 Input and Output Variables for Model of Current Bank Operations

Input Variables	*Model Output Variables, Y*
D = decision variables	Variables of primary interest to management (Y_1, Y_2, Y_3)
X = other variables	Y_1 = teller's utilization
	Y_2 = average delay
Poisson arrivals at rate = 45/hour	Y_3 = maximum line length
$X_{11}, X_{12}, \ldots$	Other output variables of secondary interest
Service times, $N(D_2, 0.2^2)$	Y_4 = observed arrival rate
$X_{21}, X_{22}, \ldots$	Y_5 = average service time
$D_1 = 1$ (one teller)	Y_6 = sample standard deviation of service times
$D_2 = 1.1$ minutes (mean service time)	Y_7 = average length of waiting line
$D_3 = 1$ (one line)	

output variables by Y. From the "black box" point of view, the model takes the inputs X and D and produces the outputs Y, namely

$$(X, D) \xrightarrow{f} Y$$

or

$$f(X, D) = Y$$

Here f denotes the transformation that is due to the structure of the model. For the Fifth National Bank study, the exponentially distributed interarrival time generated in the model (by the methods of Chapter 8) between customer $n-1$ and customer n is denoted by X_{1n}. (Do not confuse X_{1n} with A_n; the latter was an observation made on the real system.) The normally distributed service time generated in the model for customer n is denoted by X_{2n}. The set of decision variables, or policy, is $D = (D_1, D_2, D_3) = (1, 1.1, 1)$ for current operations. The output, or response, variables are denoted by $Y = (Y_1, Y_2, \ldots, Y_7)$ and are defined in Table 10.1.

For validation of the input-output transformations of the bank model to be possible, real system data must be available, comparable to at least some of the model output Y of Table 10.1. The system responses should have been collected during the same time period (from 11:00 A.M. to 1:00 P.M. on the same Friday) in which the input data $\{A_i, S_i\}$ were collected. This is important because, if system response data were collected on a slower day (say, an arrival rate of 40 per hour), the system responses such as teller utilization (Z_1), average delay (Z_2), and maximum line length (Z_3) would be expected to be lower than the same variables during a time slot when the arrival rate was 45 per hour, as observed. Suppose that the delay of successive customers was measured on the same Friday between 11:00 A.M. and 1:00 P.M. and that the average delay was found to be $Z_2 = 4.3$ minutes. For the purpose of validation, we will consider this to be the true mean value $\mu_0 = 4.3$.

When the model is run with generated random variates X_{1n} and X_{2n}, it is expected that observed values of average delay, Y_2, should be close to $Z_2 = 4.3$ minutes. The generated input values (X_{1n} and X_{2n}) cannot be expected to replicate the actual input values (A_n and S_n) of the real system exactly, but they are expected to replicate the statistical pattern of the actual inputs. Hence, simulation-generated values of Y_2 are expected to be consistent with the observed system variable, $Z_2 = 4.3$ minutes. Now consider how the modeler might test this consistency.

The modeler makes a small number of statistically independent replications of the model. Statistical independence is guaranteed by using nonoverlapping sets of random numbers produced by the random-number generator or by choosing seeds for each replication independently (from a random number table). The results of six independent replications, each of 2 hours duration, are given in Table 10.2.

Table 10.2 Results of Six Replications of the First Bank Model

Replication	Y_4 *(Arrivals/Hour)*	Y_5 *(Minutes)*	Y_2 = *Average Delay (Minutes)*
1	51	1.07	2.79
2	40	1.12	1.12
3	45.5	1.06	2.24
4	50.5	1.10	3.45
5	53	1.09	3.13
6	49	1.07	2.38
Sample mean			2.51
Standard deviation			0.82

Observed arrival rate Y_4 and sample average service time Y_5 for each replication of the model are also noted, to be compared with the specified values of 45/hour and 1.1 minutes, respectively. The validation test consists of comparing the system response, namely average delay $Z_2 = 4.3$ minutes, to the model responses, Y_2. Formally, a statistical test of the null hypothesis

$$H_0 : E(Y_2) = 4.3 \text{ minutes}$$
$$\text{versus} \qquad (10.1)$$
$$H_1 : E(Y_2) \neq 4.3 \text{ minutes}$$

is conducted. If H_0 is not rejected, then, on the basis of this test, there is no reason to consider the model invalid. If H_0 is rejected, the current version of the model is rejected, and the modeler is forced to seek ways to improve the model, as illustrated by Figure 10.3. As formulated here, the appropriate statistical test is the t test, which is conducted in the following manner:

Choose a level of significance, α, and a sample size, n. For the bank model, choose

$$\alpha = 0.05, \quad n = 6$$

Compute the sample mean, $\bar{Y}_2$, and the sample standard deviation, S, over the n replications, by using Equations (9.1) and (9.2):

$$\bar{Y}_2 = \frac{1}{n}\sum_{i=1}^{n} Y_{2i} = 2.51 \text{ minutes}$$

and

$$S = \left[\frac{\sum_{i=1}^{n}(Y_{2i} - \bar{Y}_2)^2}{n-1}\right]^{1/2} = 0.82 \text{ minute}$$

where Y_{2i}, $i = 1, \ldots, 6$, are as shown in Table 10.2.

Get the critical value of t from Table A.5. For a two-sided test, such as that in equation (10.1), use $t_{\alpha/2, n-1}$; for a one-sided test, use $t_{\alpha,n-1}$ or $-t_{\alpha,n-1}$, as appropriate ($n-1$ being the degrees of freedom). From Table A.5, $t_{0.025,5} = 2.571$ for a two-sided test.

Compute the test statistic

$$t_0 = \frac{\bar{Y}_2 - \mu_0}{S/\sqrt{n}} \qquad (10.2)$$

where μ_0 is the specified value in the null hypothesis, H_0. Here $\mu_0 = 4.3$ minutes, so that

$$t_0 = \frac{2.51 - 4.3}{0.82/\sqrt{6}} = -5.34$$

For the two-sided test, if $|t_0| > t_{\alpha/2,n-1}$, reject H_0. Otherwise, do not reject H_0. [For the one-sided test with $H_1 : E(Y_2) > \mu_0$, reject H_0 if $t > t_{\alpha,n-1}$; with $H_1 : E(Y_2) < \mu_0$, reject H_0 if $t < -t_{\alpha,n-1}$.]

Since $|t| = 5.34 > t_{0.025,5} = 2.571$, reject H_0, and conclude that the model is inadequate in its prediction of average customer delay.

Recall that, in the testing of hypotheses, rejection of the null hypothesis H_0 is a strong conclusion, because

$$P(H_0 \text{ rejected} \mid H_0 \text{ is true}) = \alpha \qquad (10.3)$$

and the level of significance α is chosen small, say $\alpha = 0.05$, as was done here. Equation (10.3) says that the probability of making the error of rejecting H_0 when H_0 is in fact true is low ($\alpha = 0.05$)—that is, the probability is small of declaring the model invalid when it is valid (with respect to the variable being tested). The assumptions justifying a t test are that the observations (Y_{2i}) are normally and independently distributed. Are these assumptions met in the present case?

1. The ith observation Y_{2i} is the average delay of all drive-in customers who began service during the ith simulation run of 2 hours; thus, by a Central Limit Theorem effect, it is reasonable to assume that each observation Y_{2i} is approximately normally distributed, provided that the number of customers it is based on is not too small.
2. The observations Y_{2i}, $i = 1, \ldots, 6$, are statistically independent by design—that is, by choice of the random-number seeds independently for each replication or by use of nonoverlapping streams.
3. The t statistic computed by Equation (10.2) is a robust statistic—that is, it is distributed approximately as the t distribution with $n - 1$ degrees of freedom, even when $Y_{21}, Y_{22}, \ldots$ are not exactly normally distributed, and thus the critical values in Table A.5 can reliably be used.

Now that the model of the Fifth National Bank of Jaspar has been found lacking, what should the modeler do? Upon further investigation, the modeler realized that the model contained two unstated assumptions:

1. When a car arrived to find the window immediately available, the teller began service immediately.
2. There is no delay between one service ending and the next beginning, when a car is waiting.

Assumption 2 was found to be approximately correct, because a service time was considered to begin when the teller actually began service but was not considered to have ended until the car had exited the drive-in window and the next car, if any, had begun service, or the teller saw that the line was empty. On the other hand, assumption 1 was found to be incorrect because the teller had other duties—mainly, serving walk-in customers if no cars were present—and tellers always finished with a previous customer before beginning service on a car. It was found that walk-in customers were always present during rush hour; that the transactions were mostly commercial in nature, taking a considerably longer time than the time required to service drive-up customers; and that, when an arriving car found no other cars at the window, it had to wait until the teller finished with the present walk-in customer. To correct this model inadequacy, the structure of the model was changed to include the additional demand on the teller's time, and data were collected on service times of walk-in customers. Analysis of these data found that they were approximately exponentially distributed with a mean of 3 minutes.

The revised model was run, yielding the results in Table 10.3. A test of the null hypothesis $H_0 : E(Y_2) = 4.3$ minutes [as in equation (10.1)] was again conducted, according to the procedure previously outlined.

Choose $\alpha = 0.05$ and $n = 6$ (sample size).

Compute $\bar{Y}_2 = 4.78$ minutes, $S = 1.66$ minutes.

Look up, in Table A.5, the critical value $t_{0.025,5} = 2.571$.

Compute the test statistic $t_0 = (\bar{Y}_2 - \mu_0)/S/\sqrt{n} = 0.710$.

Since $|t_0| < t_{0.025,5} = 2.571$, do not reject H_0, and thus tentatively accept the model as valid.

Failure to reject H_0 must be considered as a weak conclusion unless the power of the test has been estimated and found to be high (close to 1)—that is, it can be concluded only that the data at hand ($Y_{21}, \ldots, Y_{26}$) were not sufficient to reject the hypothesis $H_0 : \mu_0 = 4.3$ minutes. In other words, this test detects no inconsistency between the sample data ($Y_{21}, \ldots, Y_{26}$) and the specified mean μ_0.

The power of a test is the probability of detecting a departure from $H_0 : \mu = \mu_0$ when in fact such a departure exists. In the validation context, the power of the test is the probability of detecting an invalid model.

Table 10.3 Results of Six Replications of the Revised Bank Model

Replication	Y_4 *(Arrivals/Hour)*	Y_5 *(Minutes)*	Y_2 = *Average Delay (Minutes)*
1	51	1.07	5.37
2	40	1.11	1.98
3	45.5	1.06	5.29
4	50.5	1.09	3.82
5	53	1.08	6.74
6	49	1.08	5.49
Sample mean			4.78
Standard deviation			1.66

The power may also be expressed as 1 minus the probability of a Type II, or β, error, where $\beta = P$(Type II error) $= P$(failing to reject $H_0 | H_1$ is true) is the probability of accepting the model as valid when it is not valid.

To consider failure to reject H_0 as a strong conclusion, the modeler would want β to be small. Now, β depends on the sample size n and on the true difference between $E(Y_2)$ and $\mu_0 = 4.3$ minutes—that is, on

$$\delta = \frac{|E(Y_2) - \mu_0|}{\sigma}$$

where σ, the population standard deviation of an individual Y_{2i}, is estimated by S. Tables A.10 and A.11 are typical operating-characteristic (OC) curves, which are graphs of the probability of a Type II error $\beta(\delta)$ versus δ for given sample size n. Table A.10 is for a two-sided t test; Table A.11 is for a one-sided t test. Suppose that the modeler would like to reject H_0 (model validity) with probability at least 0.90 if the true mean delay of the model, $E(Y_2)$, differed from the average delay in the system, $\mu_0 = 4.3$ minutes, by 1 minute. Then δ is estimated by

$$\hat{\delta} = \frac{|E(Y_2) - \mu_0|}{S} = \frac{1}{1.66} = 0.60$$

For the two-sided test with $\alpha = 0.05$, use of Table A.10 results in

$$\beta(\hat{\delta}) = \beta(0.6) = 0.75 \text{ for } n = 6$$

To guarantee that $\beta(\hat{\delta}) \leq 0.10$, as was desired by the modeler, Table A.10 reveals that a sample size of approximately $n = 30$ independent replications would be required—that is, for a sample size $n = 6$ and assuming that the population standard deviation is 1.66, the probability of accepting H_0 (model validity), when in fact the model is invalid ($|E(Y_2) - \mu_0| = 1$ minute), is $\beta = 0.75$, which is quite high. If a 1-minute difference is critical, and if the modeler wants to control the risk of declaring the model valid when model predictions are as much as 1 minute off, a sample size of $n = 30$ replications is required to achieve a power of 0.9. If this sample size is too high, either a higher β risk (lower power) or a larger difference δ must be considered.

In general, it is always best to control the Type II error, or β error, by specifying a critical difference δ and choosing a sample size by making use of an appropriate OC curve. (Computation of power and use of OC curves for a wide range of tests is discussed in Hines, Montgomery, Goldsman, and Borror [2002].) In summary, in the context of model validation, the Type I error is the rejection of a valid model and is easily

Table 10.4 Types of Error in Model Validation

Statistical Terminology	*Modeling Terminology*	*Associated Risk*
Type I: rejecting H_0 when H_0 is true	Rejecting a valid model	α
Type II: failure to reject H_0 when H_1 is true	Failure to reject an invalid model	β

controlled by specifying a small level of significance α (say $\alpha = 0.1$, 0.05, or 0.01). The Type II error is the acceptance of a model as valid when it is invalid. For a fixed sample size n, increasing α will decrease β, the probability of a Type II error. Once α is set, and the critical difference to be detected is selected, the only way to decrease β is to increase the sample size. A Type II error is the more serious of the two types of errors; thus, it is important to design the simulation experiments to control the risk of accepting an invalid model. The two types of error are summarized in Table 10.4, which compares statistical terminology to modeling terminology.

Note that validation is not to be viewed as an either/or proposition, but rather should be viewed in the context of calibrating a model, as conceptually exhibited in Figure 10.3. If the current version of the bank model produces estimates of average delay (Y_2) that are not close enough to real system behavior ($\mu_0 = 4.3$ minutes), the source of the discrepancy is sought, and the model is revised in light of this new knowledge. This iterative scheme is repeated until model accuracy is judged adequate.

Philosophically, the hypothesis-testing approach tries to evaluate whether the simulation and the real system are *the same* with respect to some output performance measure or measures. A different, but closely related, approach is to attempt to evaluate whether the simulation and the real-system performance measures are *close enough* by using confidence intervals.

We continue to assume that there is a known output performance measure for the existing system, denoted by μ_0, and an unknown performance measure of the simulation, μ, that we hope is close. The hypothesis-testing formulation tested whether $\mu = \mu_0$; the confidence-interval formulation tries to bound the difference $|\mu - \mu_0|$ to see whether it is $\leq \varepsilon$, a difference that is small enough to allow valid decisions to be based on the simulation. The value of ε is set by the analyst.

Specifically, if Y is the simulation output, and $\mu = E(Y)$, then we execute the simulation and form a confidence interval for μ, such as $\bar{Y} \pm t_{\alpha/2,n-1} S/\sqrt{n}$. The determination of whether to accept the model as valid or to refine the model depends on the best-case and worst-case error implied by the confidence interval.

1. Suppose the confidence interval does not contain μ_0. (See Figure 10.6(a).)
   - (a) If the best-case error is $> \varepsilon$, then the difference in performance is large enough, even in the best case, to indicate that we need to refine the simulation model.
   - (b) If the worst-case error is $\leq \varepsilon$, then we can accept the simulation model as close enough to be considered valid.
   - (c) If the best-case error is $\leq \varepsilon$, but the worst-case error is $> \varepsilon$, then additional simulation replications are necessary to shrink the confidence interval until a conclusion can be reached.
2. Suppose the confidence interval does contain μ_0. (See Figure 10.6(b).)
   - (a) If either the best-case or worst-case error is $> \varepsilon$, then additional simulation replications are necessary to shrink the confidence interval until a conclusion can be reached.
   - (b) If the worst-case error is $\leq \varepsilon$, then we can accept the simulation model as close enough to be considered valid.

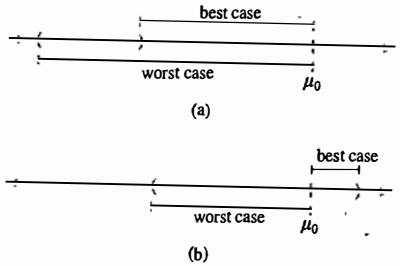


Figure 10.6 Validation of the input–output transformation (a) when the true value falls outside, (b) when the true value falls inside, the confidence interval.

In Example 10.2, $\mu_0 = 4.3$ minutes, and "close enough" was $\varepsilon = 1$ minute of expected customer delay. A 95% confidence interval, based on the 6 replications in Table 10.2, is

$$\bar{Y} \pm t_{0.025,5} S/\sqrt{n}$$

$$2.51 \pm 2.571(0.82/\sqrt{6})$$

yielding the interval [1.65, 3.37]. As in Figure 10.6(a), $\mu_0 = 4.3$ falls outside the confidence interval. Since in the best case $|3.37 - 4.3| = 0.93 < 1$, but in the worst case $|1.65 - 4.3| = 2.65 > 1$, additional replications are needed to reach a decision.

10.3.4 Input–Output Validation: Using Historical Input Data

When using artificially generated data as input data, as was done to test the validity of the bank models in Section 10.3.3, the modeler expects the model to produce event patterns that are compatible with, but not identical to, the event patterns that occurred in the real system during the period of data collection. Thus, in the bank model, artificial input data $\{X_{1n}, X_{2n}, n = 1, 2, \ldots\}$ for interarrival and service times were generated, and replicates of the output data Y_2 were compared to what was observed in the real system by means of the hypothesis test stated in equation (10.1). An alternative to generating input data is to use the actual historical record, $\{A_n, S_n, n = 1, 2, \ldots\}$, to drive the simulation model and then to compare model output with system data.

To implement this technique for the bank model, the data $A_1, A_2, \ldots$ and $S_1, S_2, \ldots$ would have to be entered into the model into arrays, or stored in a file to be read as the need arose. Just after customer n arrived at time $t_n = \sum_{i=1}^{n} A_i$, customer $n + 1$ would be scheduled on the future event list to arrive at future time $t_n + A_n + 1$ (without any random numbers being generated). If customer n were to begin service at time t'_n, a service completion would be scheduled to occur at time $t'_n + S_n$ This event scheduling without random-number generation could be implemented quite easily in a general-purpose programming language or most simulation languages by using arrays to store the data or reading the data from a file.

When using this technique, the modeler hopes that the simulation will duplicate as closely as possible the important events that occurred in the real system. In the model of the Fifth National Bank of Jaspar, the arrival times and service durations will exactly duplicate what happened in the real system on that Friday between 11:00 A.M. and 1:00 P.M. If the model is sufficiently accurate, then the delays of customers, lengths of lines, utilizations of servers, and departure times of customers predicted by the model will be close to what actually happened in the real system. It is, of course, the model-builder's and model-user's judgment that determines the level of accuracy required.

To conduct a validation test using historical input data, it is important that all the input data (A_n, S_n, ...) and all the system response data, such as average delay (Z_2), be collected during the same time period. Otherwise, the comparison of model responses to system responses, such as the comparison of average delay in the model (Y_2) to that in the system (Z_2), could be misleading. The responses (Y_2 and Z_2) depend both on the inputs (A_n and S_n) and on the structure of the system (or model). Implementation of this technique could be difficult for a large system, because of the need for simultaneous data collection of all input variables and those response variables of primary interest. In some systems, electronic counters and devices are used to ease the data-collection task by automatically recording certain types of data. The following example was based on two simulation models reported in Carson *et al.* [1981a, b], in which simultaneous data collection and the subsequent validation were both completed successfully.

Example 10.3: The Candy Factory

The production line at the Sweet Lil' Things Candy Factory in Decatur consists of three machines that make, package, and box their famous candy. One machine (the candy maker) makes and wraps individual pieces of candy and sends them by conveyor to the packer. The second machine (the packer) packs the individual pieces into a box. A third machine (the box maker) forms the boxes and supplies them by conveyor to the packer. The system is illustrated in Figure 10.7.

Each machine is subject to random breakdowns due to jams and other causes. These breakdowns cause the conveyor to begin to empty or fill. The conveyors between the two makers and the packer are used as a temporary storage buffer for in-process inventory. In addition to the randomly occurring breakdowns, if the candy conveyor empties, a packer runtime is interrupted and the packer remains idle until more candy is produced. If the box conveyor empties because of a long random breakdown of the box machine, an operator manually places racks of boxes onto the packing machine. If a conveyor fills, the corresponding maker becomes idle. The purpose of the model is to investigate the frequency of those operator interventions that require manual loading of racks of boxes as a function of various combinations of individual machines and lengths of conveyor. Different machines have different production speeds and breakdown characteristics, and longer conveyors can hold more in-process inventory. The goal is to hold operator interventions to an acceptable level while maximizing production. Machine stoppages (whether due to a full or an empty conveyor) cause damage to the product, so this is also a factor in production.

A simulation model of the Candy Factory was developed, and a validation effort using historical inputs was conducted. Engineers in the Candy Factory set aside a 4-hour time slot from 7:00 A.M. to 11:00 A.M. to

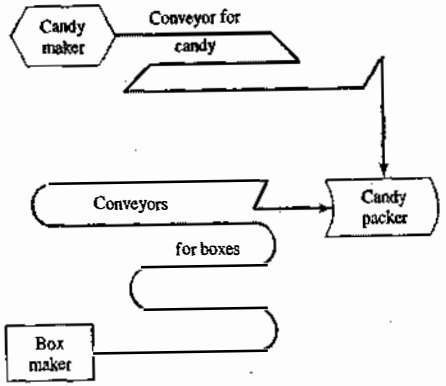


Figure 10.7 Production line at the candy factory.

collect data on an existing production line. For each machine—say, machine i—time to failure and downtime duration

$$T_{i1}, D_{i1}, T_{i2}, D_{i2}, \ldots$$

were collected. For machine $i (i = 1, 2, 3)$, T_{ij} is the jth runtime (or time to failure), and D_{ij} is the successive downtime. A runtime, T_{ij}, can be interrupted by a full or empty conveyor (as appropriate), but resumes when conditions are right. Initial system conditions at 7:00 A.M. were recorded so that they could be duplicated in the model as initial conditions at time 0. Additionally, system responses of primary interest—the production level (Z_1), and the number (Z_2) and time of occurrence (Z_3) of operator interventions—were recorded for comparison with model predictions.

The system input data, T_{ij} and D_{ij}, were fed into the model and used as runtimes and random downtimes. The structure of the model determined the occurrence of shutdowns due to a full or empty conveyor and the occurrence of operator interventions. Model response variables (Y_i, $i = 1, 2, 3$) were collected for comparison to the corresponding system response variables (Z_i, $i = 1, 2, 3$).

The closeness of model predictions to system performance aided the engineering staff considerably in convincing management of the validity of the model. These results are shown in Table 10.5. A simple display such as Table 10.5 can be quite effective in convincing skeptical engineers and managers of a model's validity—perhaps more effectively than the most sophisticated statistical methods!

With only one set of historical input and output data, only one set of simulated output data can be obtained, and thus no simple statistical tests are possible that are based on summary measures; but, if K historical input data sets are collected, and K observations $Z_{i1}, Z_{i2}, \ldots, Z_{iK}$ of some system response variable, Z_i, are collected, such that the output measure Z_{ij} corresponds to the jth input set, an objective statistical test becomes possible. For example, Z_{ij} could be the average delay of all customers who were served during the time the jth input data set was collected. With the K input data sets in hand, the modeler now runs the model K times, once for each input set, and observes the simulated results $W_{i1}, W_{i2}, \ldots, W_{iK}$ corresponding to Z_{ij}, $j = 1, \ldots, K$. Continuing the same example, W_{ij} would be the average delay predicted by the model for the jth input set. The data available for comparison appears as in Table 10.6.

If the K input data sets are fairly homogeneous, it is reasonable to assume that the K observed differences $d_j = Z_{ij} - W_{ij}$, $j = 1, \ldots, K$, are identically distributed. Furthermore, if the collection of the K sets of input data was separated in time—say, on different days—it is reasonable to assume that the K differences $d_1, \ldots, d_K$ are statistically independent and, hence, that the differences $d_1, \ldots, d_K$ constitute a random sample. In many cases, each Z_i and W_i is a sample average over customers, and so (by the Central Limit Theorem) the differences $d_j = Z_{ij} - W_{ij}$ are approximately normally distributed with some mean μ_d and variance σ_d^2. The appropriate statistical test is then a t test of the null hypothesis of no mean difference:

$$H_0 : \mu_d = 0$$

versus the alternative of significant difference:

$$H_1 : \mu_d \neq 0$$

Table 10.5 Validation of the Candy-Factory Model

Response, i	*System, Z_i*	*Model, Y_i*
1. Production level	897,208	883,150
2. Number of operator interventions	3	3
3. Time of occurrence	7:22, 8:41, 10:10	7:24, 8:42, 10:14

Table 10.6 Comparison of System and Model Output Measures for Identical Historical Inputs

Input Data Set	*System Output,* Z_{ij}	*Model Output,* W_{ij}	*Observed Difference,* d_j	*Squared Deviation from Mean,* $(d_j-\bar{d})^2$
1	Z_{i1}	W_{i1}	$d_1 = Z_{i1} - W_{i1}$	$(d_1-\bar{d})^2$
2	Z_{i2}	W_{i2}	$d_2 = Z_{i2} - W_{i2}$	$(d_2-\bar{d})^2$
3	Z_{i3}	W_{i3}	$d_3 = Z_{i3} - W_{i3}$	$(d_3-\bar{d})^2$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
K	Z_{iK}	W_{iK}	$d_K = Z_{iK} - W_{iK}$	$(d_K-\bar{d})^2$
			$\bar{d} = \frac{1}{K}\sum_{j=1}^{K} d_j$	$S_d^2 = \frac{1}{K-1}\sum_{j=1}^{K}(d_j-\bar{d})^2$

The proper test is a paired t test (Z_{i1} is paired with W_{i1}, each having been produced by the first input data set, and so on). First, compute the sample mean difference, $\bar{d}$ and the sample variance, S_d^2, by the formulas given in Table 10.6. Then, compute the t statistic as

$$t_0 = \frac{\bar{d} - \mu_d}{S_d/\sqrt{K}} \tag{10.4}$$

(with $\mu_d = 0$), and get the critical value $t_{\alpha/2,\,K-1}$ from Table A.5, where α is the prespecified significance level and $K-1$ is the number of degrees of freedom. If $|t_0| > t_{\alpha/2,K-1}$, reject the hypothesis H_0 of no mean difference, and conclude that the model is inadequate. If $|t_0| \le t_{\alpha/2,K-1}$, do not reject H_0, and hence conclude that this test provides no evidence of model inadequacy.

Example 10.4: The Candy Factory, Continued

Engineers at the Sweet Lil' Things Candy Factory decided to expand the initial validation effort reported in Example 10.3. Electronic devices were installed that could automatically monitor one of the production lines, and the validation effort of Example 10.3 was repeated with $K = 5$ sets of input data. The system and the model were compared on the basis of production level. The results are shown in Table 10.7.

Table 10.7 Validation of the Candy-Factory Model (Continued)

Input Data Set, j	*System Production,* Z_{1j}	*Model Production,* W_{1j}	*Observed Difference,* d_j	*Squared Deviation from Mean,* $(d_j-\bar{d})^2$
1	897,208	883,150	14,058	7.594×10^7
2	629,126	630,550	−1,424	4.580×10^7
3	735,229	741,420	−6,191	1.330×10^7
4	797,263	788,230	9,033	1.362×10^7
5	825,430	814,190	11,240	3.4772×10^7
			$\bar{d} = 5{,}343.2$	$S_d^2 = 7.580 \times 10^7$

A paired t test was conducted to test $H_0 : \mu_d 0$, or equivalently, $H_0 : E(Z_1) = E(W_1)$, where Z_1 is the system production level and W_1 is the production level predicted by the simulated model. Let the level of significance be $\alpha = 0.05$. Using the results in Table 10.7, the test statistic, as given by equation (10.4), is

$$t_0 = \frac{\bar{d}}{S_d/\sqrt{K}} = \frac{5343.2}{8705.85/\sqrt{5}} = 1.37$$

From Table A.5, the critical value is $t_{\alpha/2,K-1} = t_{0.025,4} = 2.78$. Since $|t_0| = 1.37 < t_{0.025,4} = 2.78$, the null hypothesis cannot be rejected on the basis of this test—that is, no inconsistency is detected between system response and model predictions in terms of mean production level. If H_0 had been rejected, the modeler would have searched for the cause of the discrepancy and revised the model, in the spirit of Figure 10.3.

10.3.5 Input – Output Validation: Using a Turing Test

In addition to statistical tests, or when no statistical test is readily applicable, persons knowledgeable about system behavior can be used to compare model output to system output. For example, suppose that five reports of system performance over five different days are prepared, and simulation output data are used to produce five "fake" reports. The 10 reports should all be in exactly the same format and should contain information of the type that managers and engineers have previously seen on the system. The 10 reports are randomly shuffled and given to the engineer, who is asked to decide which reports are fake and which are real. If the engineer identifies a substantial number of the fake reports, the model builder questions the engineer and uses the information gained to improve the model. If the engineer cannot distinguish between fake and real reports with any consistency, the modeler will conclude that this test provides no evidence of model inadequacy. For further discussion and an application to a real simulation, the reader is referred to Schruben [1980]. This type of validation test is commonly called a Turing test. Its use as model development proceeds can be a valuable tool in detecting model inadequacies and, eventually, in increasing model credibility as the model is improved and refined.

10.4 SUMMARY

Validation of simulation models is of great importance. Decisions are made on the basis of simulation results; thus, the accuracy of these results should be subject to question and investigation.

Quite often, simulations appear realistic on the surface because simulation models, unlike analytic models, can incorporate any level of detail about the real system. To avoid being "fooled" by this apparent realism, it is best to compare system data to model data and to make the comparison by using a wide variety of techniques, including an objective statistical test, if at all possible.

As discussed by Van Horn [1969, 1971], some of the possible validation techniques, in order of increasing cost-to-value ratios, include

1. Develop models with high face validity by consulting persons knowledgeable about system behavior on both model structure, model input, and model output. Use any existing knowledge in the form of previous research and studies, observation, and experience.
2. Conduct simple statistical tests of input data for homogeneity, for randonmess, and for goodness of fit to assumed distributional forms.
3. Conduct a Turing test. Have knowledgeable people (engineers, managers) compare model output to system output and attempt to detect the difference.

4. Compare model output to system output by means of statistical tests.
5. After model development, collect new system data and repeat techniques 2 to 4.
6. Build the new system (or redesign the old one) conforming to the simulation results, collect data on the new system, and use the data to validate the model (not recommended if this is the only technique used).
7. Do little or no validation. Implement simulation results without validating. (Not recommended.)

It is usually too difficult, too expensive, or too time consuming to use all possible validation techniques for every model that is developed. It is an important part of the model-builder's task to choose those validation techniques most appropriate, both to assure model accuracy and to promote model credibility.

REFERENCES

BALCI, O. [1994], "Validation, Verification and Testing Techniques throughout the Life Cycle of a Simulation Study," *Annals of Operations Research*, Vol. 53, pp. 121–174.

BALCI, O. [1998] "Verification, Validation, and Testing," in *Handbook of Simulation*, J. Banks, ed., John Wiley, New York.

BALCI, O. [2003], "Verification, Validation, and Certification of Modeling and Simulation Applications," in *Proceedings of the* 2003 *Winter Simulation Conference*, ed. by S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, pp. 150–158, Association for Computing Machinery, New York.

BALCI, O., AND R. G. SARGENT [1982a], "Some Examples of Simulation Model Validation Using Hypothesis Testing," in *Proceedings of the Winter Simulation Conference*, ed. by H. J. Highland, Y. W. Chao, and O. S. Madrigal, pp. 620–629, Association for Computing Machinery, New York.

BALCI, O., AND R. G. SARGENT [1982b], "Validation of Multivariate Response Models Using Hotelling's Two-Sample T^2 Test," *Simulation*, Vol. 39, No. 6 (Dec), pp. 185–192.

BALCI, O., AND R. G. SARGENT [1984a], "Validation of Simulation Models via Simultaneous Confidence Intervals," *American Journal of Mathematical Management Sciences*, Vol. 4, Nos. 3 & 4, pp. 375–406.

BALCI, O., AND R. G. SARGENT [1984b], "A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models," *Simuletter*, Vol. 15, No. 3, pp. 15–27.

BORTSCHELLER, B. J., AND E. T. SAULNIER [1992], "Model Reusability in a Graphical Simulation Package," in *Proceedings of the Winter Simulation Conference*, ed. by J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, pp. 764–772, Association for Computing Machinery, New York.

CARSON, J. S. [2002], "Model Verification and Validation," in *Proceedings of the Winter Simulation Conference*, ed. by E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, pp. 52–58, Association for Computing Machinery, New York.

CARSON, J. S., N. WILSON, D. CARROLL, AND C. H. WYSOWSKI [1981a], "A Discrete Simulation Model of a Cigarette Fabrication Process," *Proceedings of the Twelfth Modeling and Simulation Conference*, University of Pittsburgh, PA.

CARSON, J. S., N. WILSON, D. CARROLL, AND C. H. WYSOWSKI [1981b], "Simulation of a Filter Rod Manufacturing Process," *Proceedings of the* 1981 *Winter Simulation Conference*, ed. by T. I. Oren, C. M. Delfosse, and C. M. Shub, pp. 535–541, Association for Computing Machinery, New York.

CARSON, J. S., [1986], "Convincing Users of Model's Validity is Challenging Aspect of Modeler's Job," *Industrial Engineering*, June, pp. 76–85.

GAFARIAN, A. V., AND J. E. WALSH [1970], "Methods for Statistical Validation of a Simulation Model for Freeway Traffic near an On-Ramp," *Transportation Research*, Vol. 4, p. 379–384.

GASS, S. I. [1983], "Decision-Aiding Models: Validation, Assessment, and Related Issues for Policy Analysis," *Operations Research*, Vol. 31, No. 4, pp. 601–663.

HINES, W. W., D. C. MONTGOMERY, D. M. GOLDSMAN, AND C. M. BORROR [2002], *Probability and Statistics in Engineering*, 4th ed., Wiley, New York.

KLEIJNEN, J. P. C. [1987], *Statistical Tools for Simulation Practitioners*, Marcel Dekker, New York.

KLEIJNEN, J. P. C. [1993], "Simulation and Optimization in Production Planning: A Case Study," *Decision Support Systems*, Vol. 9, pp. 269–280.

KLEIJNEN, J. P. C. [1995], "Theory and Methodology: Verification and Validation of Simulation Models," *European Journal of Operational Research*, Vol. 82, No. 1, pp. 145–162.

LAW, A. M., AND W. D. KELTON [2000], *Simulation Modeling and Analysis*, 3d ed., McGraw-Hill, New York.

NAYLOR, T. H., AND J. M. FINGER [1967], "Verification of Computer Simulation Models," *Management Science*, Vol. 2, pp. B92–B101.

OREN, T. [1981], "Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference," *Communications of the Association for Computing Machinery*, Vol. 24, No. 4, pp. 180–89.

SARGENT, R. G. [2003], "Verification and Validation of Simulation Models," in *Proceedings of the* 2003 *Winter Simulation Conference*, ed. by S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, pp. 37–48, Association for Computing Machinery, New York.

SCHECTER, M., AND R. C. LUCAS [1980], "Validating a Large Scale Simulation Model of Wilderness Recreation Travel," *Interfaces*, Vol. 10, pp. 11–18.

SCHRUBEN, L. W. [1980], "Establishing the Credibility of Simulations," *Simulation*, Vol. 34, pp. 101–105.

SHANNON, R. E. [1975], *Systems Simulation: The Art and Science*. Prentice-Hall, Englewood Cliffs, N.J.

VAN HORN, R. L. [1969], "Validation," in *The Design of Computer Simulation Experiments*, ed. by T. H. Naylor, Duke University Press, Durham, N.C.

VAN HORN, R. L. [1971], "Validation of Simulation Results," *Management Science*, Vol. 17, pp. 247–258.

YOUNGBLOOD, S. M. [1993] "Literature Review and Commentary on the Verification, Validation and Accreditation of Models," in *Proceedings of the Summer Computer Simulation Conference*, Laurel, MD.

EXERCISES

1. A simulation model of a job shop was developed to investigate different scheduling rules. To validate the model, the scheduling rule currently used was incorporated into the model and the resulting output was compared against observed system behavior. By searching the previous year's database records, it was estimated that the average number of jobs in the shop was 22.5 on a given day. Seven independent replications of the model were run, each of 30 days' duration, with the following results for average number of jobs in the shop:

   18.9 22.0 19.4 22.1 19.8 21.9 20.2

   (a) Develop and conduct a statistical test to evaluate whether model output is consistent with system behavior. Use the level of significance $\alpha = 0.05$.
   (b) What is the power of this test if a difference of two jobs is viewed as critical? What sample size is needed to guarantee a power of 0.8 or higher? (Use $\alpha = 0.05$.)

2. System data for the job shop of Exercise 1 revealed that the average time spent by a job in the shop was approximately 4 working days. The model made the following predictions, on seven independent replications, for average time spent in the shop:

   3.70 4.21 4.35 4.13 3.83 4.32 4.05

   (a) Is model output consistent with system behavior? Conduct a statistical test, using the level of significance $\alpha = 0.01$.
   (b) If it is important to detect a difference of 0.5 day, what sample size is needed to have a power of 0.90? Interpret your results in terms of model validity or invalidity. (Use $\alpha = 0.01$.)

3. For the job shop of Exercise 1, four sets of input data were collected over four different 10-day periods, together with the average number of jobs in the shop (Z_i) for each period. The input data were used to

drive the simulation model for four runs of 10 days each, and model predictions of average number of jobs in the shop (Y_i) were collected, with these results:

i	1	2	3	4
Z_i	21.7	19.2	22.8	19.4
Y_i	24.6	21.1	19.7	24.9

(a) Conduct a statistical test to check the consistency of system output and model output. Use the level of significance $\alpha = 0.05$.
(b) If a difference of two jobs is viewed as important to detect, what sample size is required to guarantee a probability of at least 0.80 of detecting this difference if it indeed exists? (Use $\alpha = 0.05$.)

4. Find several examples of actual simulations reported in the literature in which the authors discuss validation of their model. Is enough detail given to judge the adequacy of the validation effort? If so, compare the reported validation with the criteria set forth in this chapter. Did the authors use any validation technique not discussed in this chapter? [Several potential sources of articles on simulation applications include the journal *Interfaces* and *Simulation*, and the *Winter Simulation Conference Proceedings* at www.informs-cs.org.]

5. (a) Compare validation in simulation to the validation of theories in the physical sciences.
(b) Compare the issues involved and the techniques available for validation of models of physical systems versus models of social systems.
(c) Contrast the difficulties, and compare the techniques, in validating a model of a manually operated warehouse with fork trucks and other manually operated vehicles, versus a model of a facility with automated guided vehicles, conveyors, and an automated storage-and-retrieval system.
(d) Repeat (c) for a model of a production system involving considerable manual labor and human decision making, versus a model of the same production system after it has been automated.

11

Output Analysis for a Single Model

Output analysis is the examination of data generated by a simulation. Its purpose is either to predict the performance of a system or to compare the performance of two or more alternative system designs. This chapter deals with the analysis of a single system; Chapter 12 deals with the comparison of two or more systems. The need for statistical output analysis is based on the observation that the output data from a simulation exhibits random variability when random-number generators are used to produce the values of the input variables—that is, two different streams or sequences of random numbers will produce two sets of outputs, which (probably) will differ. If the performance of the system is measured by a parameter θ, the result of a set of simulation experiments will be an estimator $\hat{\theta}$ of θ. The precision of the estimator $\hat{\theta}$ can be measured by the standard error of $\hat{\theta}$ or by the width of a confidence interval for θ. The purpose of the statistical analysis is either to estimate this standard error or confidence interval or to figure out the number of observations required to achieve a standard error or confidence interval of a given size—or both.

Consider a typical output variable, Y, the total cost per week of an inventory system; Y should be treated as a random variable with an unknown distribution. A simulation run of length 1 week provides a single sample observation from the population of all possible observations on Y. By increasing the run length, the sample size can be increased to n observations, $Y_1, Y_2, \ldots, Y_n$, based on a run length of n weeks. However, these observations do not constitute a random sample, in the classical sense, because they are not statistically independent. In this case, the inventory on hand at the end of one week is the beginning inventory on hand for the next week, and thus the value of Y_i has some influence on the value of Y_{i+1}. Thus, the sequence of random variables $Y_1, Y_2, \ldots, Y_n$, could be autocorrelated (i.e., correlated with itself). This autocorrelation, which is a measure of a lack of statistical independence, means that classical methods of statistics, which assume independence, are not directly applicable to the analysis of these output data. The methods must be properly modified and the simulation experiments properly designed for valid inferences to be made.

In addition to the autocorrelation present in most simulation output data, the specification of the initial conditions of the system at time 0 can pose a problem for the simulation analyst and could influence the output data. For example, the inventory on hand and the number of backorders at time 0 would most likely influence the value of Y_1, the total cost for week 1. Because of the autocorrelation, these initial conditions would also influence the costs $(Y_2, \ldots, Y_n)$ for subsequent weeks. The specified initial conditions, if not chosen well, can have an especially deleterious effect on attempting to estimate the steady-state (long-run) performance of a simulation model. For purposes of statistical analysis, the effect of the initial conditions is that the output observations might not be identically distributed and that the initial observations might not be representative of the steady-state behavior of the system.

Section 11.1 distinguishes between two types of simulation—transient versus steady state—and defines commonly used measures of system performance for each type of simulation. Section 11.2 illustrates by example the inherent variability in a stochastic (i.e., probabilistic) discrete-event simulation and thereby demonstrates the need for a statistical analysis of the output. Section 11.3 covers the statistical estimation of performance measures. Section 11.4 discusses the analysis of transient simulations, Section 11.5 the analysis of steady-state simulations.

11.1 TYPES OF SIMULATIONS WITH RESPECT TO OUTPUT ANALYSIS

In the analyzing of simulation output data, a distinction is made between terminating or transient simulations and steady-state simulations. A *terminating* simulation is one that runs for some duration of time T_E, where E is a specified event (or set of events) that stops the simulation. Such a simulated system "opens" at time 0 under well-specified *initial conditions* and "closes" at the *stopping time* T_E. The next four examples are terminating simulations.

Example 11.1

The Shady Grove Bank opens at 8:30 A.M. (time 0) with no customers present and 8 of the 11 tellers working (initial conditions) and closes at 4:30 P.M. (time $T_E = 480$ minutes). Here, the event E is merely the fact that the bank has been open for 480 minutes. The simulation analyst is interested in modeling the interaction between customers and tellers over the entire day, including the effect of starting up and of closing down at the end of the day.

Example 11.2

Consider the Shady Grove Bank of Example 11.1, but restricted to the period from 11:30 A.M. (time 0) to 1:30 P.M., when it is especially busy. The simulation run length is $T_E = 120$ minutes. The initial conditions at time 0 (11:30 A.M.) could be specified in essentially two ways: (1) the real system could be observed at 11:30 on a number of different days and a distribution of number of customers in system (at 11:30 A.M.) could be estimated, then these data could be used to load the simulation model with customers at time 0; or (2) the model could be simulated from 8:30 A.M. to 11:30 A.M. without collecting output statistics, and the ending conditions at 11:30 A.M. used as initial conditions for the 11:30 A.M. to 1:30 P.M. simulation.

Example 11.3

A communications system consists of several components plus several backup components. It is represented schematically in Figure 11.1. Consider the system over a period of time, T_E, until the system fails. The stopping event E is defined by $E = \{A$ fails, or D fails, or (B and C both fail)$\}$. Initial conditions are that all components are new at time 0.

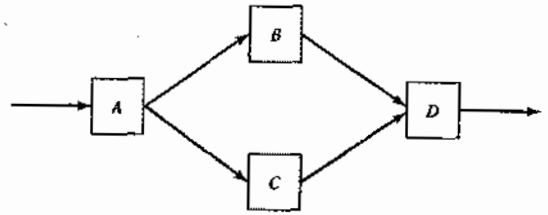


Figure 11.1 Example of a communications system.

Notice that, in the bank model of Example 11.1, the stopping time $T_E = 480$ minutes is known, but in Example 11.3, the stopping time T_E is generally unpredictable in advance; in fact, T_E is probably the output variable of interest, as it represents the total time until the system breaks down. One goal of the simulation might be to estimate $E(T_E)$, the mean time to system failure.

Example 11.4

A widget-manufacturing process runs continuously from Monday mornings until Saturday mornings. The first shift of each workweek is used to load inventory buffers and chemical tanks with the components and catalysts needed to make the final product (28 varieties of widget). These components and catalysts are made continually throughout the week, except for the last shift Friday night, which is used for cleanup and maintenance. Thus, most inventory buffers are near empty at the end of the week. During the first shift on Monday, a buffer stock is built up to cover the eventuality of breakdown in some part of the process. It is desired to simulate this system during the first shift (time 0 to time $T_E = 8$ hours) to study various scheduling policies for loading inventory buffers.

In the simulating of a terminating system, the initial conditions of the system at time 0 must be specified, and the stopping time T_E—or, alternatively, the stopping event E—must be well defined. Although it is certainly true that the Shady Grove Bank in Example 11.1 will open again the next day, the simulation analyst has chosen to consider it a terminating system because the object of interest is one day's operation, including start up and close down. On the other hand, if the simulation analyst were interested in some other aspect of the bank's operations, such as the flow of money or operation of automated teller machines, then the system might be considered as a nonterminating one. Similar comments apply to the communications system of Example 11.3. If the failed component were replaced and the system continued to operate, and, most important, if the simulation analyst were interested in studying its long-run behavior, it might be considered as a nonterminating system. In Example 11.3, however, interest is in its short-run behavior, from time 0 until the first system failure at time T_E. *Therefore, whether a simulation is considered to be terminating depends on both the objectives of the simulation study and the nature of the system.*

Example 11.4 is a terminating system, too. It is also an example of a *transient* (or nonstationary) simulation: the variables of interest are the in-process inventory levels, which are increasing from zero or near zero (at time 0) to full or near full (at time 8 hours).

A *nonterminating system* is a system that runs continuously, or at least over a very long period of time. Examples include assembly lines that shut down infrequently, continuous production systems of many different types, telephone systems and other communications systems such as the Internet, hospital emergency rooms, police dispatching and patrolling operations, fire departments, and continuously operating computer networks.

A simulation of a nonterminating system starts at simulation time 0 under initial conditions defined by the analyst and runs for some analyst-specified period of time T_E. (Significant problems arise concerning the specification of these initial and stopping conditions, problems that we discuss later.) Usually, the analyst wants to study steady-state, or long-run, properties of the system—that is, properties that are not influenced

by the initial conditions of the model at time 0. *A steady-state simulation is a simulation whose objective is to study long-run, or steady-state, behavior of a nonterminating system.* The next two examples are steady-state simulations.

Example 11.5

Consider the widget-manufacturing process of Example 11.4, beginning with the second shift when the complete production process is under way. It is desired to estimate long-run production levels and production efficiencies. For the relatively long period of 13 shifts, this may be considered as a steady-state simulation. To obtain sufficiently precise estimates of production efficiency and other response variables, the analyst could decide to simulate for any length of time, T_E (even longer than 13 shifts)—that is, T_E is not determined by the nature of the problem (as it was in terminating simulations); rather, it is set by the analyst as one parameter in the design of the simulation experiment.

Example 11.6

HAL Inc., a large computer-service bureau, has many customers worldwide. Thus, its large computer system with many servers, workstations, and peripherals runs continuously, 24 hours per day. To handle an increased work load, HAL is considering additional CPUs, memory, and storage devices in various configurations. Although the load on HAL's computers varies throughout the day, management wants the system to be able to accommodate sustained periods of peak load. Furthermore, the time frame in which HAL's business will change in any substantial way is unknown, so there is no fixed planning horizon. Thus, a steady-state simulation at peak-load conditions is appropriate. HAL systems staff develops a simulation model of the existing system with the current peak work load and then explores several possibilities for expanding capacity. HAL is interested in long-run average throughput and utilization of each computer. The stopping time, T_E, is determined not by the nature of the problem, but rather by the simulation analyst, either arbitrarily or with a certain statistical precision in mind.

11.2 STOCHASTIC NATURE OF OUTPUT DATA

Consider one run of a simulation model over a period of time $[0, T_E]$. Since the model is an input-output transformation, as illustrated by Figure 10.5, and since some of the model input variables are random variables, it follows that the model output variables are random variables. Three examples are now given to illustrate the nature of the output data from stochastic simulations and to give a preliminary discussion of several important properties of these data. Do not be concerned if some of these properties and the associated terminology are not entirely clear on a first reading. They will be explained carefully later in the chapter.

Example 11.7: Able and Baker, Revisited

Consider the Able–Baker technical-support call center problem (Example 2.2) which involved customers arriving according to the distribution of Table 2.11 and being served either by Able, whose service-time distribution is given in Table 2.12, or by Baker, whose service-time distribution is given in Table 2.13. The purpose of the simulation is to estimate Able's utilization, ρ, and the mean time spent in the system per customer, w, over the first 2 hours of the workday. Therefore, each run of the model is for a 2-hour period, with the system being empty and idle at time 0. Four statistically independent runs were made by using four distinct streams of random numbers to generate the interarrival and service times. Table 11.1 presents the results. The estimated utilization for run r is given by $\hat{\rho}_r$ and the estimated average system time by $\hat{w}_r$ (i.e., $\hat{w}_r$ is the sample average time in system for all customers served during run r). Notice that, in this sample, the observed utilization ranges from 0.708 to 0.875 and the observed average system time ranges from 3.74 minutes to 4.53 minutes. The stochastic nature of the output data $\{\hat{\rho}_1, \hat{\rho}_2, \hat{\rho}_3, \hat{\rho}_4\}$ and $\{\hat{w}_1, \hat{w}_2, \hat{w}_3, \hat{w}_4\}$ is demonstrated by the results in Table 11.1.

Table 11.1 Results of Four Independent Runs of 2-Hour Duration of the Able–Baker Queueing Problem

Run, r	Able's Utilization $\hat{\rho}_r$	Average System Time, $\hat{w}_r$ (Minutes)
1	0.808	3.74
2	0.875	4.53
3	0.708	3.84
4	0.842	3.98

There are two general questions that we will address by a statistical analysis—say, of the observed utilizations $\hat{\rho}_r$, $r = 1,..., 4$:

1. estimation of the true utilization $\rho = E(\hat{\rho}_r)$ by a single number, called a point estimate;
2. estimation of the error in our point estimate, in the form either of a standard error or of a confidence interval.

These questions are addressed in Section 11.4 for terminating simulations, such as Example 11.7. Classical methods of statistics may be used because $\hat{\rho}_1$, $\hat{\rho}_2$, $\hat{\rho}_3$, and $\hat{\rho}_4$ constitute a random sample—that is, they are independent and identically distributed. In addition, $\rho = E(\hat{\rho}_r)$ is the parameter being estimated, so each $\hat{\rho}_r$ is an unbiased estimate of the true mean utilization ρ. The analysis of Example 11.7 is considered in Example 11.10 of Section 11.4. A survey of statistical methods applicable to terminating simulations is given by Law [1980]. Additional guidance may be found in Alexopoulos and Seila [1998], Kleijnen [1987], Law and Kelton [2000], and Nelson [2001].

The next example illustrates the effects of correlation and initial conditions on the estimation of long-run mean measures of performance of a system.

Example 11.8

Consider a single-server queue with Poisson arrivals at an average rate of one every 10 minutes ($\lambda = 0.1$ per minute) and service times that are normally distributed, with mean 9.5 minutes and standard deviation 1.75 minutes.[1] This is an *M/G/1* queue, which was described and analyzed in Section 6.4.1. By Equation (6.11), the true long-run server utilization is $\rho = \lambda E(S) = (0.1)(9.5) = 0.95$. We typically would not need to simulate such a system, because we can analyze it mathematically; but we simulate it here to illustrate difficulties that occur in trying to estimate the long-run mean queue length, L_Q, defined by Equation (6.4).

Suppose we run a single simulation for a total of 5000 minutes and observe the output process $L_Q\{(t), 0 \le t \le 5000\}$, where $L_Q(t)$ is the number of customers in the waiting line at time t minutes. To make this continuous-time process a little easier to analyze, we divide the time interval [0, 5000) into five equal subintervals of 1000 minutes and compute the average number of customers in queue for each interval individually. Specifically, the average number of customers in the queue from time $(j-1)1000$ to $j(1000)$ is

$$Y_j = \frac{1}{1000}\int_{(j-1)1000}^{j(1000)} L_Q(t)\,dt, \quad j = 1,..., 5 \tag{11.1}$$

[1] The range of a service time is restricted to ±5 standard deviations, to exclude the possibility of a negative service time; that range covers well over 99.999% of the normal distribution.

Thus, $Y_1 = \int_0^{1000} L_Q(t)dt/1000$ is the time-weighted average number of customers in the queue from time 0 to time 1000, $Y_2 = \int_{1000}^{2000} L_Q(t)dt/1000$ is the same average over [1000, 2000), and so on. Equation (11.1) is a special case of Equation (6.4). The observations $\{Y_1, Y_2, Y_3, Y_4, Y_5\}$ provide an example of "batching" of raw simulation data—in this case, $L_Q\{(t), 0 \le t \le 5000\}$—and the Y_j are called *batch means*. The use of batch means in analyzing output data is discussed in Section 11.5.5. For now, simply notice that batching transforms the continuous-time queue-length process, $\{L_Q(t), 0 \le t \le 5000\}$, into a discrete-time batch-means process $\{Y_i, i = 1, 2, 3, 4, 5\}$ where each Y_i is an estimator of L_Q.

The simulation results of three statistically independent replications are shown in Table 11.2. Each replication, or run, uses a distinct stream of random numbers. For replication 1, Y_{1j} is the batch mean for batch j (the jth interval), as defined by Equation (11.1); similarly, Y_{2j} and Y_{3j} are defined for batch j for replications 2 and 3, respectively. Table 11.2 also gives the sample mean over each replication, $\bar{Y}_{r\cdot}$, for replications $r = 1,2,3$.[2] That is,

$$\bar{Y}_{r\cdot} = \frac{1}{5}\sum_{j=1}^{5} Y_{rj}, \qquad r = 1, 2, 3 \tag{11.2}$$

It probably will not surprise you that, if we take batch averages first, then average the batch means, or just average everything together, we get the same thing. In other words, each $\bar{Y}_{r\cdot}$ is equivalent to the time average over the entire interval [0, 5000) for replication r, as given by Equation (6.4).

Table 11.2 illustrates the inherent variability in stochastic simulations both *within* a single replication and *across* different replications. Consider the variability within replication 3, in which the average queue length over the batching intervals varies from a low of $Y_{31} = 7.67$ customers during the first 1000 minutes to a high of $Y_{33} = 20.36$ customers during the third subinterval of 1000 minutes. Table 11.2 also shows the variability across replications. Compare Y_{15} to Y_{25} to Y_{35}, the average queue lengths over the intervals 4000 to 5000 minutes across all three replications.

Suppose, for the moment, that a simulation analyst makes only one replication of this model and gets the result $\bar{Y}_{1\cdot} = 3.75$ customers as an estimate of mean queue length, L_Q. How precise is the estimate? This question is usually answered by attempting to estimate the standard error of $\bar{Y}_{1\cdot}$, or by forming a confidence interval. The simulation analyst might think that the five batch means $Y_{11}, Y_{12}, \ldots, Y_{15}$ could be regarded as a random sample; however, the terms in the sequence are not independent, and in fact they are autocorrelated,

Table 11.2 Batched Average Queue Length for Three Independent Replications

Batching Interval (Minutes)	*Batch, j*	*Replication* 1, Y_{1j}	2, Y_{2j}	3, Y_{3j}
[0, 1000)	1	3.61	2.91	7.67
[1000, 2000)	2	3.21	9.00	19.53
[2000, 3000)	3	2.18	16.15	20.36
[3000, 4000)	4	6.92	24.53	8.11
[4000, 5000)	5	2.82	25.19	12.62
[0, 5000)		$\bar{Y}_{1\cdot} = 3.75$	$\bar{Y}_{2\cdot} = 15.56$	$\bar{Y}_{3\cdot} = 13.66$

[2]The dot, as in the subscript $r\cdot$, indicates summation over the second subscript; the bar, as in $\bar{Y}_{r\cdot}$, indicates an average.

because all of the data are obtained from within one replication. If $Y_{11}, \ldots, Y_{15}$ were mistakenly assumed to be independent observations, and their autocorrelation were ignored, the usual classical methods of statistics might severely underestimate the standard error of $\bar{Y}_{1\cdot}$, possibly resulting in the simulation analyst's thinking that a high degree of precision had been achieved. On the other hand, the averages across the three replications, $\bar{Y}_{1\cdot}$, $\bar{Y}_{2\cdot}$, and $\bar{Y}_{3\cdot}$, can be regarded as independent observations, because they are derived from three different replications.

Intuitively, Y_{11} and Y_{12} are correlated because in replication 1 the queue length at the end of the time interval [0, 1000) is the queue length at the beginning of the interval [1000, 2000)—similarly for any two adjacent batches within a given replication. If the system is congested at the end of one interval, it will be congested for a while at the beginning of the next time interval. Similarly, periods of low congestion tend to follow each other. Within a replication, say for $Y_{r1}, Y_{r2}, \ldots, Y_{r5}$, high values of a batch mean tend to be followed by high values and low values by low. This tendency of adjacent observations to have like values is known as positive autocorrelation. The effect of ignoring autocorrelation when it is present is discussed in more detail in Section 11.5.2.

Now suppose that the purpose of the *M/G/*1 queueing simulation of Example 11.8 is to estimate "steady-state" mean queue length, that is, mean queue length under "typical operating conditions over the long run." However, each of the three replications was begun in the empty and idle state (no customers in the queue and the server available). The empty and idle initial state means that, within a given replication, there will be a higher-than- "typical" probability that the system will be uncongested for times close to 0. The practical effect is that an estimator of L_Q—say, $\bar{Y}_{r\cdot}$ for replication r—will be biased low [i.e., $E(\bar{Y}_{r\cdot}) < L_Q$]. The extent of the bias decreases as the run length increases, but, for short-run-length simulations with atypical initial conditions, this initialization bias can produce misleading results. The problem of initialization bias is discussed further in Section 11.5.1.

11.3 MEASURES OF PERFORMANCE AND THEIR ESTIMATION

Consider the estimation of a performance parameter, θ (or ϕ), of a simulated system. It is desired to have a point estimate and an interval estimate of θ (or ϕ). The length of the interval estimate is a measure of the error in the point estimate. The simulation output data are of the form $\{Y_1, Y_2, \ldots, Y_n\}$ for estimating θ; we refer to such output data as *discrete-time data*, because the index n is discrete valued. The simulation output data are of the form $\{Y(t), 0 \le t \le T_E\}$ for estimating ϕ; we refer to such output data as *continuous-time data*, because the index t is continuous valued. For example, Y_i might be the delay of customer i, or the total cost in week i; $Y(t)$ might be the queue length at time t, or the number of backlogged orders at time t. The parameter θ is an ordinary mean; ϕ will be referred to as a time-weighted mean. Whether we call the performance parameter θ or ϕ does not really matter; we use two different symbols here simply to provide a distinction between ordinary means and time-weighted means.

11.3.1 Point Estimation

The point estimator of θ based on the data $\{Y_1, \ldots, Y_n\}$ is defined by

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} Y_i \tag{11.3}$$

where $\hat{\theta}$ is a sample mean based on a sample of size n. Computer simulation languages may refer to this as a "discrete-time," "collect," "tally" or "observational" statistic.

The point estimator $\hat{\theta}$ is said to be unbiased for θ if its expected value is θ—that is, if

$$E(\hat{\theta}) = \theta \tag{11.4}$$

In general, however,

$$E(\hat{\theta}) \neq \theta \tag{11.5}$$

and $E(\hat{\theta}) - \theta$ is called the *bias* in the point estimator θ. It is desirable to have estimators that are unbiased, or, if this is not possible, have a small bias relative to the magnitude of θ. Examples of estimators of the form of Equation (11.3) include $\hat{w}$ and $\hat{w}_Q$ of Equations (6.5) and (6.7), in which case Y_i is the time spent in the (sub)system by customer i.

The point estimator of ϕ based on the data $\{Y(t), 0 \le t \le T_E\}$, where T_E is the simulation run length, is defined by

$$\hat{\phi} = \frac{1}{T_E}\int_0^{T_E} Y(t)dt \tag{11.6}$$

and is called a time average of $Y(t)$ over $[0, T_E]$. Simulation languages may refer to this as a "continuous-time," "discrete-change" or "time-persistent" statistic. In general,

$$E(\hat{\phi}) \neq \phi \tag{11.7}$$

and $\hat{\phi}$ is said to be biased for ϕ. Again, we would like to obtain unbiased or low-bias estimators. Examples of time averages include L and $\hat{L}_Q$ of Equations (6.3) and (6.4) and Y_j of Equation (11.1).

Generally, θ and ϕ are regarded as mean measures of performance of the system being simulated. Other measures usually can be put into this common framework. For example, consider estimation of the proportion of days on which sales are lost through an out-of-stock situation. In the simulation, let

$$Y_i = \begin{cases} 1, & \text{if out of stock on day } i \\ 0, & \text{otherwise} \end{cases}$$

With n equal to the total number of days, $\hat{\theta}$ defined by Equation (11.3) is a point estimator of θ, the proportion of out-of-stock days. For a second example, consider estimation of the proportion of time queue length is greater than k_0 customers (for example, $k_0 = 10$). If $L_Q(t)$ represents simulated queue length at time t, then (in the simulation) define

$$Y(t) = \begin{cases} 1, & \text{if } L_Q(t) > k_0 \\ 0, & \text{otherwise} \end{cases}$$

Then $\hat{\phi}$, as defined by Equation (11.6), is a point estimator of ϕ, the proportion of time that the queue length is greater than k_0 customers. Thus, estimation of proportions or probabilities is a special case of the estimation of means.

A performance measure that does not fit this common framework is a quantile or percentile. Quantiles describe the level of performance that can be delivered with a given probability, p. For instance, suppose that Y represents the delay in queue that a customer experiences in a service system, measured in minutes. Then the 0.85 quantile of Y is the value θ such that

$$\Pr\{Y \le \theta\} = p \tag{11.8}$$

where $p = 0.85$ in this case. As a percentage, θ is the $100p$th or 85th percentile of customer delay. Therefore, 85% of all customers will experience a delay of θ minutes or less. Stated differently, a customer has only a 0.15 probability of experiencing a delay of longer than θ minutes. A widely used performance measure is the median, which is the 0.5 quantile or 50th percentile.

The problem of estimating a quantile is the inverse of the problem of estimating a proportion or probability. Consider Equation (11.8). In estimating a proportion, θ is given and p is to be estimated; but, in estimating a quantile, p is given and θ is to be estimated.

The most intuitive method for estimating a quantile is to form a histogram of the observed values of Y, then find a value $\hat{\theta}$ such that $100p\%$ of the histogram is to the left of (smaller than) $\hat{\theta}$. For instance, if we observe $n = 250$ customer delays $\{Y_1, \ldots, Y_{250}\}$, then an estimate of the 85th percentile of delay is a value $\hat{\theta}$ such that $(0.85)(250) = 212.5 \approx 213$ of the observed values are less than or equal to θ. An obvious estimate is, therefore, to set $\hat{\theta}$ equal to the 213th smallest value in the sample (this requires sorting the data). When the output is a continuous-time process, such as the queue-length process $\{L_Q t), 0 \le t \le T_E\}$, then a histogram gives the *fraction of time* that the process spent at each possible level (queue length in this example). However, the method for quantile estimation remains the same: Find a value $\hat{\theta}$ such that $100p\%$ of the histogram is to the left of $\hat{\theta}$.

11.3.2 Confidence-Interval Estimation

To understand confidence intervals fully, it is important to understand the difference between a *measure of error* and a *measure of risk*. One way to make the difference clear is to contrast a confidence interval with a *prediction interval* (which is another useful output-analysis tool).

Both confidence and prediction intervals are based on the premise that the data being produced by the simulation is represented well by a probability model. Suppose that model is the normal distribution with mean θ and variance σ^2, both unknown. To make the example concrete, let $\bar{Y}_i$. be the average cycle time for parts produced on the ith replication (representing a day of production) of the simulation. Therefore, θ is the mathematical expectation of $\bar{Y}_i$., and σ is represents the day-to-day variation of the average cycle time.

Suppose our goal is to estimate θ. If we are planning to be in business for a long time, producing parts day after day, then θ is a relevant parameter, because it is the long-run mean daily cycle time. Our average cycle time will vary from day to day, but over the long run the *average of the averages* will be close to θ.

The natural estimator for θ is the overall sample mean of R independent replications, $\bar{Y}.. = \sum_{i=1}^{R} \bar{Y}_i./R$. But $\bar{Y}..$ is not θ, it is an estimate, based on a sample, and it has error. A confidence interval is a measure of that error. Let

$$S^2 = \frac{1}{R-1}\sum_{i=1}^{R}(Y_i. - \bar{Y}..)^2$$

be the sample variance across the R replications. The usual confidence interval, which assumes the Y_i. are normally distributed, is

$$\bar{Y}.. \pm t_{\alpha/2,R-1}\frac{S}{\sqrt{R}}$$

where $t_{\alpha/2,R-1}$ is the quantile of the t distribution with $R - 1$ degrees of freedom that cuts off $\alpha/2$ of the area of each tail. (See Table A.5.) We cannot know for certain exactly how far $\bar{Y}..$ is from θ, but the confidence interval attempts to bound that error. Unfortunately, the confidence interval itself may be wrong. A confidence level, such as 95%, tells us how much we can trust the interval to actually bound the error between $\bar{Y}..$ and θ. The more replications we make, the less error there is in $\bar{Y}..$, and our confidence interval reflects that because $t_{\alpha/2,R-1}S/\sqrt{R}$ will tend to get smaller as R increases, converging to 0 as R goes to infinity.

Now suppose we need to make a promise about what the average cycle time will be on a particular day. A good guess is our estimator $\bar{Y}_{..}$, but it is unlikely to be exactly right. Even θ itself, which is the center of the distribution, is not likely to be the *actual average cycle time* on any particular day, because the daily average cycle time varies. A prediction interval, on the other hand, is designed to be wide enough to contain the actual average cycle time on any particular day with high probability. A prediction interval is a measure of risk; a confidence interval is a measure of error.

The normal-theory prediction interval is

$$\bar{Y}_{..} \pm t_{\alpha/2,R-1} S \sqrt{1+\frac{1}{R}}$$

The length of this interval will not go to 0 as R increases. In fact, in the limit it becomes

$$\theta \pm z_{\alpha/2}\sigma$$

to reflect the fact that, no matter how much we simulate, our daily average cycle time still varies.

In summary, a prediction interval is a measure of risk, and a confidence interval is a measure of error. We can simulate away error by making more and more replications, but we can never simulate away risk, which is an inherent part of the system. We can, however, do a better job of evaluating risk by making more replications.

Example 11.9

Suppose that the overall average of the average cycle time on 120 replications of a manufacturing simulation is 5.80 hours, with a sample standard deviation of 1.60 hours. Since $t_{0.025,119} = 1.98$, a 95% confidence interval for the long-run expected daily average cycle time is $5.80 \pm 1.98(1.60/\sqrt{120})$ or 5.80 ± 0.29 hours. Thus, our best guess of the long-run average of the daily average cycle times is 5.80 hours, but there could be as much as ±0.29 hours error in this estimate.

On any particular day, we are 95% confident that the average cycle time for all parts produced on that day will be

$$5.80 \pm 1.98(1.60)\sqrt{1+\frac{1}{120}}$$

or 5.80 ± 3.18 hours. The ±3.18 hours reflects the inherent variability in the daily average cycle times and the fact that we want to be 95% confident of covering the actual average cycle time on a particular day (rather than simply covering the long-run average).

11.4 OUTPUT ANALYSIS FOR TERMINATING SIMULATIONS

Consider a terminating simulation that runs over a simulated time interval $[0, T_E]$ and results in observations $Y_1, \ldots, Y_n$. The sample size, n, may be a fixed number, or it may be a random variable (say, the number of observations that occur during time T_E). A common goal in simulation is to estimate

$$\theta = E\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right)$$

When the output data are of the form $\{Y(t), 0 \le t \le T_E\}$, the goal is to estimate

$$\phi = E\left(\frac{1}{T_E}\int_0^{T_E} Y(t)\,dt\right)$$

The method used in each case is the method of *independent replications*. The simulation is repeated a total of R times, each run using a different random number stream and independently chosen initial conditions (which includes the case that all runs have identical initial conditions). We now address this problem.

11.4.1 Statistical Background

Perhaps the most confusing aspect of simulation output analysis is distinguishing *within-replication* data from *across-replication* data, and understanding the properties and uses of each. The issue can be further confused by the fact that simulation languages often provide only summary measures, like sample means, sample variances, and confidence intervals, rather than all of the raw data. Sometimes these summary measures are *all* the simulation language provides without a lot of extra work.

To illustrate the key ideas, think in terms of the simulation of a manufacturing system and two performance measures of that system, the cycle time for parts (time from release into the factory until completion) and the work in process (WIP, the total number of parts in the factory at any time). In computer applications, these two measures could correspond to the response time and the length of the task queue at the CPU; in a service application, they could be the time to fulfill a customer's request and the number of requests on the "to do" list; in a supply-chain application, they could be the order fill time and the inventory level. Similar measures appear in many systems.

Here is the usual set up for something like cycle time: Let Y_{ij} be the cycle time for the jth part produced in the ith replication. If each replication represents two shifts of production, then the number of parts produced in each replication might differ. Table 11.3 shows, symbolically, the results of R replications.

The across-replication data are formed by summarizing within-replication data: $\bar{Y}_{i\cdot}$ is the sample mean of the n_i cycle times from the ith replication, S_i^2 is the sample variance of the same data, and

$$H_i = t_{\alpha/2,n_i-1}\frac{S_i}{\sqrt{n_i}} \tag{11.9}$$

is a confidence-interval half-width based on this dataset.

From the across-replication data, we compute overall statistics, the average of the daily cycle time averages

$$\bar{Y}_{..} = \frac{1}{R}\sum_{i=1}^{R} \bar{Y}_{i\cdot} \tag{11.10}$$

Table 11.3 Within- and Across-Replication Cycle-Time Data

Within-Rep Data				*Across-Rep Data*
Y_{11}	Y_{12}	$\cdots$	Y_{1n_1}	$\bar{Y}_{1\cdot}, S_1^2, H_1$
Y_{21}	Y_{22}	$\cdots$	Y_{2n_2}	$\bar{Y}_{2\cdot}, S_2^2, H_2$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
Y_{R1}	Y_{R2}	$\cdots$	Y_{Rn_R}	$\bar{Y}_{R\cdot}, S_R^2, H_R$
				$\bar{Y}_{..}, S^2, H$

the sample variance of the daily cycle time averages

$$S^2 = \frac{1}{R-1}\sum_{i=1}^{R}(\bar{Y}_{i\cdot} - \bar{Y}_{\cdot\cdot})^2 \tag{11.11}$$

and finally, the confidence-interval half-width

$$H = t_{\alpha/2,R-1}\frac{S}{\sqrt{R}} \tag{11.12}$$

The quantity $S/\sqrt{R}$ is the standard error, which is sometimes interpreted as the average error in $\bar{Y}_{\cdot\cdot}$ as an estimator of θ. Notice that S^2 is *not* the average of the within-replication sample variances, S_i^2; rather, it is the sample variance of the within-replication averages $\bar{Y}_{1\cdot}, \bar{Y}_{2\cdot}, \ldots, \bar{Y}_{R\cdot}$.

Within a replication, work in process (WIP) is a continuous-time output, denoted $Y_i(t)$. The stopping time for the ith replication, T_{E_i}, could be a random variable, in general; in this example, it is the end of the second shift. Table 11.4 is an abstract representation of the data produced.

The within-replication sample mean and variance are defined appropriately for continuous-time data:

$$\bar{Y}_{i\cdot} = \frac{1}{T_{E_i}}\int_0^{T_{E_i}} Y_i(t)\,dt \tag{11.13}$$

and

$$S_i^2 = \frac{1}{T_{E_i}}\int_0^{T_{E_i}} (Y_i(t) - \bar{Y}_{i\cdot})^2\,dt \tag{11.14}$$

A definition for H_i is more problematic, but, to be concrete, take it to be

$$H_i = z_{\alpha/2}\frac{S_i}{\sqrt{T_{E_i}}}. \tag{11.15}$$

Frankly, it is difficult to conceive of a situation in which H_i is relevant, a topic we discuss later. Although the definitions of the within-replication data change for continuous-time data, the across-replication statistics are unchanged, and this is a critical observation.

Table 11.4 Within- and Across-Replication WIP Data

Within-Rep Data	*Across-Rep Data*
$Y_1(t), 0 \le t \le T_{E_1}$	$\bar{Y}_{1\cdot}, S_1^2, H_1$
$Y_2(t), 0 \le t \le T_{E_2}$	$\bar{Y}_{2\cdot}, S_2^2, H_2$
$\vdots$	$\vdots$
$Y_R(t), 0 \le t \le T_{E_R}$	$\bar{Y}_{R\cdot}, S_R^2, H_R$
	$\bar{Y}_{\cdot\cdot}, S^2, E$

Here are the key points that must be understood:

- The overall sample average, $\bar{Y}_{\cdot\cdot}$, and the individual replication sample averages, $\bar{Y}_{i\cdot}$, are always unbiased estimators of the expected daily average cycle time or daily average WIP.
- Across-replication data are independent (since they are based on different random numbers), are identically distributed (since we are running the same model on each replication), and tend to be normally distributed if they are averages of within-replication data, as they are here. This implies that the confidence interval $\bar{Y}_{\cdot\cdot} \pm H$ is often pretty good.
- Within-replication data, on the other hand, might have none of these properties. The individual cycle times may not be identically distributed (if the first few parts of the day find the system empty); they are almost certainly not independent (because one part follows another); and whether they are normally distributed is difficult to know in advance. For this reason, S_i^2 and H_i, which are computed under the assumption of independent and identically distributed (i.i.d.) data, tend not to be useful (although there are exceptions).
- There are situations in which $\bar{Y}_{\cdot\cdot}$ and $\bar{Y}_{i\cdot}$ are valid estimators of the expected cycle time for an *individual part* or the *expected WIP at any point in time*, rather than the daily average. (See Section 11.5 on steady-state simulations.) Even when this is the case, the confidence interval $\bar{Y}_{\cdot\cdot} \pm H$ is valid, and $\bar{Y}_{i\cdot} \pm H_i$ is not. The difficulty occurs because S_i^2 is a reasonable estimator of the variance of the cycle time, but S_i^2/n_i and S_i^2/T_{E_i} are not good estimators of the $\text{Var}[\bar{Y}_{i\cdot}]$—more on this in Section 11.5.2.

Example 11.10: The Able–Baker Problem, Continued

Consider Example 11.7, the Able–Baker technical-support call center problem, with the data for $R = 4$ replications given in Table 11.1. The four utilization estimates, $\hat{\rho}_r$, are time averages of the form of Equation (11.13). The simulation produces output data of the form

$$Y_r(t) = \begin{cases} 1, & \text{if Able is busy at time } t \\ 0, & \text{otherwise} \end{cases}$$

and $\hat{\rho}_r = \bar{Y}_{r\cdot}$ as computed by Equation (11.13) with $T_E = 2$ hours. Similarly, the four average system times, $\hat{w}_1, \ldots, \hat{w}_4$, are analogous to $\bar{Y}_{r\cdot}$ of Table 11.3, where Y_{ri} is the actual time spent in system by customer i on replication r.

First, suppose that the analyst desires a 95% confidence interval for Able's true utilization, ρ. Using Equation (11.10) compute an overall point estimator

$$\bar{Y}_{\cdot\cdot} = \hat{\rho} = \frac{0.808 + 0.875 + 0.708 + 0.842}{4} = 0.808$$

Using Equation (11.11), compute its estimated variance:

$$S^2 = \frac{(0.808 - 0.808)^2 + \cdots + (0.842 - 0.808)^2}{4-1} = (0.072)^2$$

Thus, the standard error of $\hat{\rho} = 0.808$ is estimated by s.e. $(\hat{\rho}) = S/\sqrt{4} = 0.036$. Obtain $t_{0.025,3} = 3.18$ from Table A.5, and compute the 95% confidence interval half-width by (11.12) as

$$H = t_{0.025,3}\frac{S}{\sqrt{4}} = (3.18)(0.036) = 0.114$$

giving 0.808 ± 0.114 or, with 95% confidence,

$$0.694 \le \rho \le 0.922$$

In a similar fashion, compute a 95% confidence interval for mean time in system w:

$$\hat{w} = \frac{3.74+4.53+3.84+3.98}{4} = 4.02 \text{ minutes}$$

$$S^2 = \frac{(3.74-4.02)^2 + \ldots + (3.98-4.02)^2}{3-1} = (0.352)^2$$

so that

$$H = t_{0.025,3}\frac{S}{\sqrt{4}} = (3.18)(0.176) = 0.560$$

or

$$4.02 - 0.56 \le w \le 4.02 + 0.56$$

Thus, the 95% confidence interval for w is $3.46 \le w \le 4.58$.

11.4.2 Confidence Intervals with Specified Precision

By Expression (11.12), the half-length H of a $100(1-\alpha)\%$ confidence interval for a mean θ, based on the t distribution, is given by

$$H = t_{\alpha/2,R-1}\frac{S}{\sqrt{R}}$$

where S^2 is the sample variance and R is the number of replications. Suppose that an error criterion ϵ is specified; in other words, it is desired to estimate θ by $\bar{Y}_{..}$ to within $\pm\epsilon$ with high probability—say, at least $1-\alpha$. Thus, it is desired that a sufficiently large sample size, R, be taken to satisfy

$$P(|\bar{Y}_{..} - \theta| < \epsilon) \ge 1-\alpha$$

When the sample size, R, is fixed, no guarantee can be given for the resulting error. But if the sample size can be increased, an error criterion can be specified.

Assume that an initial sample of size R_0 replications has been observed—that is, the simulation analyst initially makes R_0 independent replications. We must have $R_0 \ge 2$, with 10 or more being desirable. The R_0 replications will be used to obtain an initial estimate S_0^2 of the population variance σ^2. To meet the half-length criterion, a sample size R must be chosen such that $R \ge R_0$ and

$$H = t_{\alpha/2,R-1}\frac{S_0}{\sqrt{R}} \le \epsilon \tag{11.16}$$

Solving for R in Inequality (11.23) shows that R is the smallest integer satisfying $R \ge R_0$ and

$$R \ge \left(\frac{t_{\alpha/2,R-1}S_0}{\epsilon}\right)^2 \tag{11.17}$$

Since $t_{\alpha/2,R-1} \ge z_{\alpha/2}$, an initial estimate for R is given by

$$R \ge \left(\frac{z_{\alpha/2}S_0}{\epsilon}\right)^2 \tag{11.18}$$

where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentage point of the standard normal distribution from Table A.3. And since $t_{\alpha/2,R-1} \approx z_{\alpha/2}$ for large R (say, $R \ge 50$), the second inequality for R is adequate when R is large. After determining the final sample size, R, collect $R - R_0$ additional observations (i.e., make $R - R_0$ additional replications, or start over and make R total replications) and form the $100(1-\alpha)\%$ confidence interval for θ by

$$\bar{Y}_{..} - t_{\alpha/2,R-1}\frac{S}{\sqrt{R}} \le \theta \le \bar{Y}_{..} + t_{\alpha/2,R-1}\frac{S}{\sqrt{R}} \tag{11.19}$$

where $\bar{Y}_{..}$ and S^2 are computed on the basis of all R replications, $\bar{Y}_{..}$ by Equation (11.10), and S^2 by Equation (11.11). The half-length of the confidence interval given by Inequality (11.19) should be approximately, ϵ or smaller; however, with the additional $R - R_0$ observations, the variance estimator S^2 could differ somewhat from the initial estimate S_0^2, possibly causing the half-length to be greater than desired. If the confidence interval (11.19) is too large, the procedure may be repeated, using Inequality (11.17), to determine an even larger sample size.

Example 11.11

Suppose that it is desired to estimate Able's utilization in Example 11.7 to within ± 0.04 with probability 0.95. An initial sample of size $R_0 = 4$ is taken, with the results given in Table 11.1. An initial estimate of the population variance is $S_0^2 = (0.072)^2 = 0.00518$. (See Example 11.10 for the relevant data.) The error criterion is $\epsilon = 0.04$, and the confidence coefficient is $1 - \alpha = 0.95$. From Inequality (11.18), the final sample size must be at least as large as

$$\frac{z_{0.025}^2 S_0^2}{\epsilon^2} = \frac{(1.96)^2(0.00518)}{(0.04)^2} = 12.44$$

Next, Inequality (11.17) can be used to test possible candidates ($R = 13, 14, \ldots$) for final sample size, as follows:

R	13	14	15
$t_{0.025,R-1}$	2.18	2.16	2.14
$\frac{t_{0.025,R-1}^2 S_0^2}{\epsilon^2}$	15.39	15.10	14.83

Thus, $R = 15$ is the smallest integer satisfying Inequality (11.17), so $R - R_0 = 15 - 4 = 11$ additional replications are needed. After obtaining the additional outputs, we would again need to compute the half-width H to ensure that it is as small as is desired.

11.4.3 Quantiles

To present the interval estimator for quantiles, it is helpful to review the interval estimator for a mean in the special case when the mean represents a proportion or probability, p. In this book, we have chosen to treat a proportion or probability as just a special case of a mean. However, in many statistics texts, probabilities are treated separately.

When the number of independent replications $Y_1, \ldots, Y_R$ is large enough that $t_{\alpha/2,n-1} \doteq z_{\alpha/2}$, the confidence interval for a probability p is often written as

$$\hat{p} \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{R-1}}$$

where $\hat{p}$ is the sample proportion (tedious algebra shows that this formula for the half-width is precisely equivalent to Equation (11.12) when used in estimating a proportion).

As mentioned in Section 11.3, the quantile-estimation problem is the inverse of the probability-estimation problem: Find θ such that $\Pr\{Y \le \theta\} = p$. Thus, to estimate the p quantile, we find that value $\hat{\theta}$ such that $100p\%$ of the data in a histogram of Y is to the left of $\hat{\theta}$ (or stated differently, the npth smallest value of $Y_1, \ldots, Y_R$).

Extending this idea, an approximate $(1-\alpha)100\%$ confidence interval for θ can be obtained by finding two values: θ_ℓ that cuts off $100p_\ell\%$ of the histogram and θ_u that cuts off $100p_u\%$ of the histogram, where

$$p_\ell = p - z_{\alpha/2}\sqrt{\frac{p(1-p)}{R-1}}$$
$$p_u = p + z_{\alpha/2}\sqrt{\frac{p(1-p)}{R-1}} \qquad (11.20)$$

(Recall that we know p.) In terms of sorted values, $\hat{\theta}_\ell$ is the Rp_ℓ smallest value (rounded down) and $\hat{\theta}_u$ is the Rp_u smallest value (rounded up), of $Y_1, \ldots, Y_R$.

Example 11.12

Suppose that we want to estimate the 0.8 quantile of the time to failure (in hours) for the communications system in Example 11.3 and form a 95% confidence interval for it. A histogram of $R = 500$ independent replications is shown in Figure 11.2.

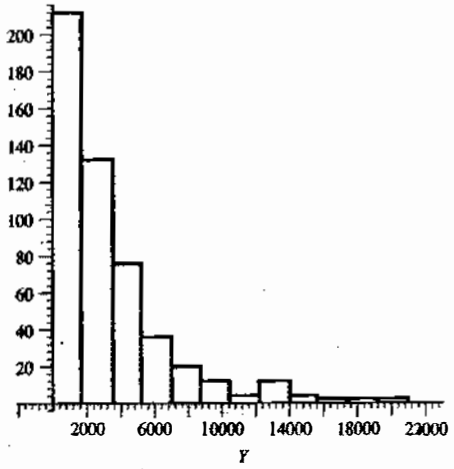


Figure 11.2 Failure data in hours for 500 replications of the communications system.

The point estimator is $\hat{\theta} = 4644$ hours, because 80% of the data in the histogram is to the left of 4644. Equivalently, it is the $500 \times 0.8 = 400$th smallest value of the sorted data.

To obtain the confidence interval we first compute

$$p_\ell = p - z_{\alpha/2}\sqrt{\frac{p(1-p)}{R-1}} = 0.8 - 1.96\sqrt{\frac{0.8(0.2)}{499}} = 0.765$$
$$p_u = p + z_{\alpha/2}\sqrt{\frac{p(1-p)}{R-1}} = 0.8 + 1.96\sqrt{\frac{0.8(0.2)}{499}} = 0.835$$

The lower bound of the confidence interval is $\hat{\theta}_\ell = 4173$ (the $500 \times p_\ell = 382$nd smallest value, rounding down); the upper bound of the confidence interval is $\hat{\theta}_u = 5119$ hours (the $500 \times p_u = 418$th smallest value, rounding up).

11.4.4 Estimating Probabilities and Quantiles from Summary Data

Knowing the equation for the confidence interval half-width is important if all the simulation software provides is $\bar{Y}_{\cdot\cdot}$ and H and you need to work out the number of replications required to get a prespecified precision, or if you need to estimate a probability or quantile. You know the number of replications, so the sample standard deviation can be extracted from H by using the formula

$$S = \frac{H\sqrt{R}}{t_{\alpha/2,R-1}}$$

With this information, the method in Section 11.4.2 can be employed.

The more difficult problem is estimating a probability or quantile from summary data. When all we have available is the sample mean and confidence-interval halfwidth (which gives us the sample standard deviation), then one approach is to use a normal-theory approximation for the probabilities or quantiles we desire, specifically

$$\Pr\{\bar{Y}_{\cdot\cdot} \le c\} \approx \Pr\left\{Z \le \frac{c - \bar{Y}_{\cdot\cdot}}{S}\right\}$$

and

$$\hat{\theta} \approx \bar{Y}_{\cdot\cdot} + z_p S$$

The following example illustrates how this is done.

Example 11.13

From 25 replications of the manufacturing simulation, a 90% confidence interval for the daily average WIP is 218 ± 32. What is the probability that the daily average WIP is less than 350? What is the 85th percentile of daily average WIP?

First, we extract the standard deviation:

$$S = \frac{H\sqrt{R}}{t_{0.05,24}} = \frac{32\sqrt{25}}{1.71} = 93$$

Then, we use the normal approximations and Table A.3 to get

$$\Pr\{\bar{Y}_i \le 350\} \approx \Pr\left\{Z \le \frac{350-218}{93}\right\} = \Pr\{Z \le 1.42\} = 0.92$$

and

$$\hat{\theta} \approx \bar{Y}_i + z_{0.85}S = 218 + 1.04(93) = 315 \text{ parts}$$

There are shortcomings to obtaining our probabilities and quantiles this way. The approximation depends heavily on whether the output variable of interest is normally distributed. If the output variable itself is not an average, then this approximation is suspect. Therefore, we expect the approximation to work well for statements about the average daily cycle time, for instance, but very poorly for the cycle time of an individual part.

11.5 OUTPUT ANALYSIS FOR STEADY-STATE SIMULATIONS

Consider a single run of a simulation model whose purpose is to estimate a *steady-state*, or *long-run*, characteristic of the system. Suppose that the single run produces observations $Y_1, Y_2, \ldots$, which, generally, are samples of an autocorrelated time series. The steady-state (or long-run) measure of performance, θ, is defined by

$$\theta = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i \tag{11.21}$$

with probability 1, where the value of θ is independent of the initial conditions. (The phrase "with probability 1" means that essentially all simulations of the model, using different random numbers, will produce series Y_i, $i = 1, 2, \ldots$ whose sample average converges to θ.) For example, if Y_i was the time customer i spent talking to an operator, then θ would be the long-run average time a customer spends talking to an operator; and, because θ is defined as a limit, it is independent of the call center's conditions at time 0. Similarly, the steady-state performance for a continuous-time output measure $\{Y(t), t \ge 0\}$, such as the number of customers in the call center's hold queue, is defined as

$$\phi = \lim_{T_E\to\infty} \frac{1}{T_E}\int_0^{T_E} Y(t)\,dt$$

with probability 1.

Of course, the simulation analyst could decide to stop the simulation after some number of observations—say, n—have been collected; or the simulation analyst could decide to simulate for some length of time T_E that determines n (although n may vary from run to run). The sample size n (or T_E) is a *design* choice; it is not inherently determined by the nature of the problem. The simulation analyst will choose simulation run length (n or T_E) with several considerations in mind:

1. Any bias in the point estimator that is due to artificial or arbitrary initial conditions. (The bias can be severe if run length is too short, but generally it decreases as run length increases.)
2. The desired precision of the point estimator, as measured by the standard error or confidence interval half-width.
3. Budget constraints on computer resources.

The next subsection discusses initialization bias and the following subsections outline two methods of estimating point-estimator variability. For clarity of presentation, we discuss only estimation of θ from a discrete-time output process. Thus, when discussing one replication (or run), the notation

$$Y_1, Y_2, Y_3, \ldots$$

will be used; if several replications have been made, the output data for replication r will be denoted by

$$Y_{r1}, Y_{r2}, Y_{r3}, \ldots \tag{11.22}$$

11.5.1 Initialization Bias in Steady-State Simulations

There are several methods of reducing the point-estimator bias caused by using artificial and unrealistic initial conditions in a steady-state simulation. The first method is to initialize the simulation in a state that is more representative of long-run conditions. This method is sometimes called intelligent initialization. Examples include

1. setting the inventory levels, number of backorders, and number of items on order and their arrival dates in an inventory simulation;
2. placing customers in queue and in service in a queueing simulation;
3. having some components failed or degraded in a reliability simulation.

There are at least two ways to specify the initial conditions intelligently. If the system exists, collect data on it and use these data to specify more nearly typical initial conditions. This method sometimes requires a large data-collection effort. In addition, if the system being modeled does not exist—for example, if it is a variant of an existing system—this method is impossible to implement. Nevertheless, it is recommended that simulation analysts use any available data on existing systems to help initialize the simulation, as this will usually be better than assuming the system to be "completely stocked," "empty and idle," or "brand new" at time 0.

A related idea is to obtain initial conditions from a second model of the system that has been simplified enough to make it mathematically solvable. The queueing models in Chapter 6 are very useful for this purpose. The simplified model can be solved to find long-run expected or most likely conditions—such as the expected number of customers in the queue—and these conditions can be used to initialize the simulation.

A second method to reduce the impact of initial conditions, possibly used in conjunction with the first, is to divide each simulation run into two phases: first, an initialization phase, from time 0 to time T_0, followed by a data-collection phase from time T_0 to the stopping time $T_0 + T_E$—that is, the simulation begins at time 0 under specified initial conditions I_0 and runs for a specified period of time T_0. Data collection on the response variables of interest does not begin until time T_0 and continues until time $T_0 + T_E$. The choice of T_0 is quite important, because the system state at time T_0, denoted by I, should be more nearly representative of steady-state behavior than are the original initial conditions at time 0, I_0. In addition, the length T_E of the data-collection phase should be long enough to guarantee sufficiently precise estimates of steady-state behavior. Notice that the system state, I, at time T_0 is a random variable and to say that the system has reached an approximate steady state is to say that the probability distribution of the system state at time T_0 is sufficiently close to the steady-state probability distribution as to make the bias in point estimates of response variables negligible. Figure 11.3 illustrates the two phases of a steady-state simulation. The effect of starting a simulation run of a queueing model in the empty and idle state, as well as several useful plots to aid the simulation analyst in choosing an appropriate value of T_0, are given in the following example.

Example 11.14

Consider the *M/G/1* queue discussed in Example 11.8. Suppose that a total of 10 independent replications were made ($R = 10$), each replication beginning in the empty and idle state. The total simulation run length on each replication was $T_0 + T_E = 15{,}000$ minutes. The response variable was queue length, $L_Q(t, r)$, at time t,

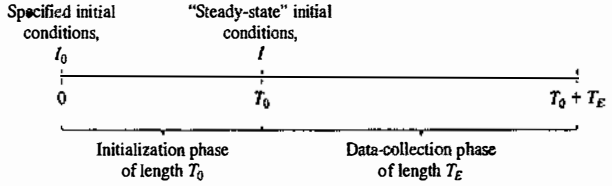


Figure 11.3 Initialization and data collection phases of a steady-state simulation run.

where the second argument, r, denotes the replication ($r = 1,\ldots, 10$). The raw output data were batched, as in Example 11.8, Equation (11.1), in batching intervals of 1000 minutes, to produce the following batch means:

$$Y_{rj} = \frac{1}{1000}\int_{(j-1)1000}^{j(1000)} L_Q(t, r)dt \tag{11.23}$$

for replication $r = 1,\ldots, 10$ and for batch $j = 1, 2,\ldots, 15$. The estimator in Equation (11.23) is simply the time-weighted-average queue length over the time interval $[(j-1)1000, j(1000))$, similar to that in Equation (6.4). The 15 batch means for the 10 replications are given in Table 11.5.

Normally we average all the batch means *within* each replication to obtain a replication average. However, our goal at this stage is to identify the trend in the data due to initialization bias and find out when it dissipates. To do this, we will average corresponding batch means *across* replications and plot them (this idea is usually attributed to Welch [1983]). Such averages are known as *ensemble averages*. Specifically, for each batch j, define the ensemble average across all R replications to be

$$\bar{Y}_{\cdot j} = \frac{1}{R}\sum_{r=1}^{R} Y_{rj} \tag{11.24}$$

($R = 10$ here). The ensemble averages $\bar{Y}_{\cdot j}, j = 1, \ldots, 15$ are displayed in the third column of Table 11.6. Notice that $\bar{Y}_{\cdot 1} = 4.03$ and $\bar{Y}_{\cdot 2} = 5.45$ are estimates of mean queue length over the time periods $[0, 1000)$ and $[1000, 2000)$, respectively, and they are less than all other ensemble averages $\bar{Y}_{\cdot j} (j = 3,\ldots, 15)$. The simulation analyst may suspect that this is due to the downward bias in these estimators, which in turn is due to the queue being empty and idle at time 0. This downward bias is further illustrated in the plots that follow.

Figure 11.4 is a plot of the ensemble averages, $\bar{Y}_{\cdot j}$, versus $1000j$, for $j = 1, 2,\ldots, 15$. The actual values, $\bar{Y}_{\cdot j}$, are the discrete set of points in circles, which have been connected by straight lines as a visual aid. Figure 11.4 illustrates the downward bias of the initial observations. As time becomes larger, the effect of the initial conditions on later observations lessens and the observations appear to vary around a common mean. When the simulation analyst feels that this point has been reached, then the data-collection phase begins.

Table 11.6 also gives the cumulative average sample mean after deleting zero, one, and two batch means from the beginning—that is, using the ensemble average batch means $\bar{Y}_{\cdot j}$, when deleting d observations out of a total of n observations, compute

$$\bar{Y}_{\cdot\cdot}(n, d) = \frac{1}{n-d}\sum_{j=d+1}^{n} \bar{Y}_{\cdot j} \tag{11.25}$$

The results in Table 11.6 for the *M/G/*1 simulation are for $d = 0$, 1, and 2, and $n = d + 1,\ldots, 15$. These cumulative averages with deletion, namely $\bar{Y}_{\cdot\cdot}(n, d)$, are plotted for comparison purposes in Figure 11.5. We do not recommend using cumulative averages to determine the initialization phase, for reasons given next.

Table 11.5 Individual Batch Means (Y_{rj}) for *M/G/*1 Simulation with Empty and Idle Initial State

Replication	Batch 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3.61	3.21	2.18	6.92	2.82	1.59	3.55	5.60	3.04	2.57	1.41	3.07	4.03	2.70	2.71
2	2.91	9.00	16.15	24.53	25.19	21.63	24.47	8.45	8.53	14.84	23.65	27.58	24.19	8.58	4.06
3	7.67	19.53	20.36	8.11	12.62	22.15	14.10	9.87	23.96	24.50	14.56	6.08	4.83	16.04	23.41
4	6.62	1.75	12.87	8.77	1.25	1.16	1.92	6.29	4.74	17.43	18.24	18.59	4.62	2.76	1.57
5	2.18	1.32	2.14	2.18	2.59	1.20	4.11	6.21	7.31	1.58	2.16	3.08	2.32	2.21	3.32
6	0.93	3.54	4.80	0.72	2.95	5.56	1.96	2.07	2.74	3.45	14.24	13.39	7.87	0.94	3.19
7	1.12	2.59	5.05	1.16	2.72	5.12	5.03	4.14	4.98	15.81	9.29	2.14	8.72	29.80	28.94
8	1.54	5.94	5.33	2.91	2.69	1.91	3.27	3.61	10.35	9.66	4.13	6.14	7.90	2.61	7.95
9	8.93	4.78	0.74	2.56	9.43	18.63	8.14	1.49	4.51	1.69	12.62	11.28	3.32	3.42	3.35
10	4.78	2.84	10.39	5.87	1.01	2.59	16.77	27.25	26.81	20.96	7.26	2.32	5.04	8.50	9.11

Table 11.6 Summary of Data for M/G/1 Simulation: Ensemble Batch Means and Cumulative Means, Averaged Over 10 Replications

Run Length T	*Batch j*	*Average Batch Mean,* $\bar{Y}_{\cdot j}$	*Cumulative Average (No Deletion),* $\bar{Y}_{\cdot\cdot}(j, 0)$	*Cumulative Average (Delete 1),* $\bar{Y}_{\cdot\cdot}(j, 1)$	*Cumulative Average (Delete 2),* $\bar{Y}_{\cdot\cdot}(j, 2)$
1,000	1	4.03	4.03	—	—
2,000	2	5.45	4.74	5.45	—
3,000	3	8.00	5.83	6.72	8.00
4,000	4	6.37	5.96	6.61	7.18
5,000	5	6.33	6.04	6.54	6.90
6,000	6	8.15	6.39	6.86	7.21
7,000	7	8.33	6.67	7.11	7.44
8,000	8	7.50	6.77	7.16	7.45
9,000	9	9.70	7.10	7.48	7.77
10,000	10	11.25	7.51	7.90	8.20
11,000	11	10.76	7.81	8.18	8.49
12,000	12	9.37	7.94	8.29	8.58
13,000	13	7.28	7.89	8.21	8.46
14,000	14	7.76	7.88	8.17	8.40
15,000	15	8.76	7.94	8.21	8.43

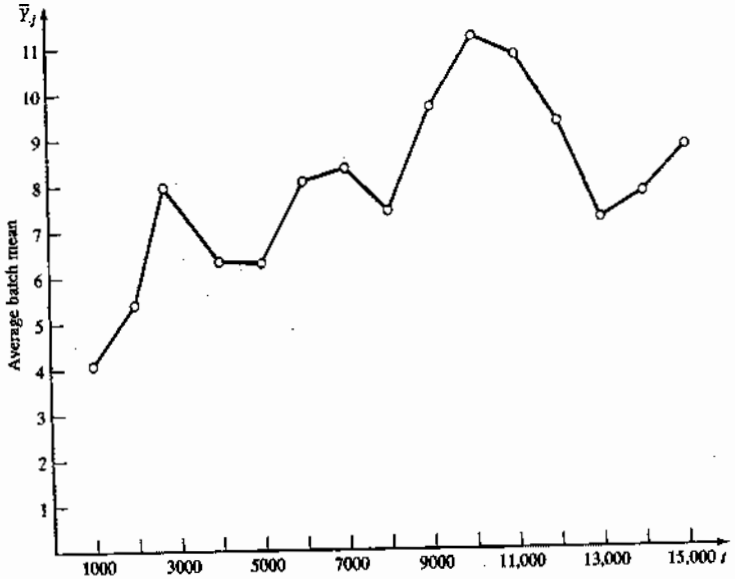


Figure 11.4 Ensemble averages $\bar{Y}_{\cdot j}$ for M/G/1 queue.

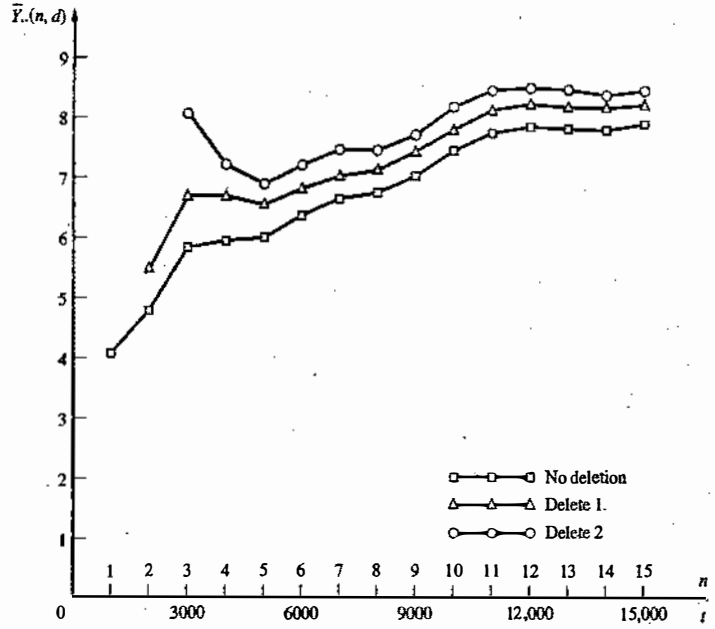


Figure 11.5 Cumulative average queue length $\bar{Y}_{\cdot\cdot}(n, d)$ versus time 1000n.

From Figures 11.4 and 11.5, it is apparent that downward bias is present, and this initialization bias in the point estimator can be reduced by deletion of one or more observations. For the 15 ensemble average batch means, it appears that the first two observations have considerably more bias than any of the remaining ones. The effect of deleting first one and then two batch means is also illustrated in Table 11.6 and Figure 11.5. As expected, the estimators increase in value as more data are deleted; that is, $\bar{Y}_{\cdot\cdot}(15, 2) = 8.43$ and $\bar{Y}_{\cdot\cdot}(15, 1) = 8.21$ are larger than $\bar{Y}_{\cdot\cdot}(15, 0) = 7.94$. It also appears from Figure 11.5 that $\bar{Y}_{\cdot\cdot}(n, d)$ is increasing for $n = 5, 6, \ldots, 11$ (and all $d = 0, 1, 2$), and thus there may still be some initialization bias. It seems, however, that deletion of the first two batches removes most of the bias.

Unfortunately, there is no widely accepted, objective, and proven technique to guide how much data to delete to reduce initialization bias to a negligible level. Plots can, at times, be misleading, but they are still recommended. Several points should be kept in mind:

1. Ensemble averages, such as Figure 11.4, will reveal a smoother and more precise trend as the number of replications, R, is increased. Since each ensemble average is the sample mean of i.i.d. observations, a confidence interval based on the t distribution can be placed around each point, as shown in Figure 11.6, and these intervals can be used to judge whether or not the plot is precise enough to judge that bias has diminished. *This is the preferred method to determine a deletion point.*
2. Ensemble averages can be smoothed further by plotting a moving average, rather than the original ensemble averages. In a moving average, each plotted point is actually the average of several adjacent ensemble averages. Specifically, the jth plot point would be

$$\tilde{Y}_{\cdot j} = \frac{1}{2m+1} \sum_{i=j-m}^{j+m} \bar{Y}_{\cdot i}$$

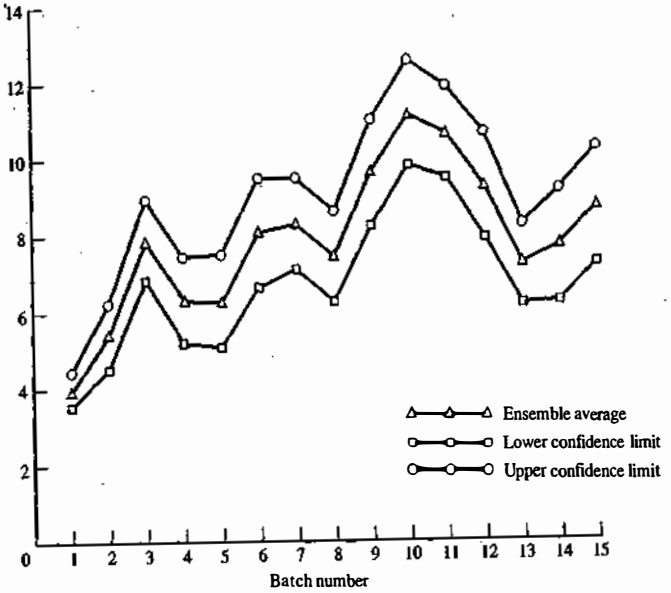


Figure 11.6 Ensemble averages $\bar{Y}_{\cdot j}$ for M/G/1 queue with 95% confidence intervals.

for some $m \geq 1$, rather than the original ensemble average $\bar{Y}_{\cdot j}$. The value of m is typically chosen by trial and error until a smooth plot is obtained. See Law and Kelton [2000] or Welch [1983] for further discussion of smoothing.

3. Cumulative averages, such as in Figure 11.5, become less variable as more data are averaged. Therefore, it is expected that the left side of the curve will always be less smooth than the right side. More importantly, cumulative averages tend to converge more slowly to long-run performance than do ensemble averages, because cumulative averages contain all observations, including the most biased ones from the beginning of the run. *For this reason, cumulative averages should be used only if it is not feasible to compute ensemble averages*, such as when only a single replication is possible.
4. Simulation data, especially from queueing models, usually exhibit positive autocorrelation. The more correlation present, the longer it takes for $\bar{Y}_{\cdot j}$ to approach steady state. The positive correlation between successive observations (i.e., batch means) $\bar{Y}_{\cdot 1}, \bar{Y}_{\cdot 2}, \ldots$ can be seen in Figure 11.4.
5. In most simulation studies, the analyst is interested in several different output performance measures at once, such as the number in queue, customer waiting time, and utilization of the servers. Unfortunately, different performance measures could approach steady state at different rates. Thus, it is important to examine each performance measure individually for initialization bias and use a deletion point that is adequate for all of them.

There has been no shortage of solutions to the initialization-bias problem. Unfortunately, for every "solution" that works well in some situations, there are other situations in which either it is not applicable or it performs poorly. Important ideas include testing for bias (e.g., Kelton and Law [1983], Schruben [1980], Goldsman, Schruben, and Swain [1994]); modeling the bias (e.g., Snell and Schruben [1985]); and randomly sampling the initial conditions on multiple replications (e.g., Kelton [1989]).

11.5.2 Error Estimation for Steady-State Simulation

If $\{Y_1, \ldots, Y_n\}$ are not statistically independent, then S^2/n, given by Equation (11.11), is a biased estimator of the true variance, $V(\hat{\theta})$. This is almost always the case when $\{Y_1, \ldots, Y_n\}$ is a sequence of output observations from within a single replication. In this situation, $Y_1, Y_2, \ldots$ is an autocorrelated sequence, sometimes called a time series. Example 11.8 (the *M/G/*1 queue) provides an illustration of this situation.

Suppose that our point estimator for θ is the sample mean $\bar{Y} = \sum_{i=1}^{n} Y_i/n$. A general result from mathematical statistics is that the variance of $\bar{Y}$ is[3]

$$V(\bar{Y}) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} \text{cov}(Y_i, Y_j) \tag{11.26}$$

where $\text{cov}(Y_i, Y_i) = V(Y_i)$. To construct a confidence interval for θ, an estimate of $V(\bar{Y})$ is required. But obtaining an estimate of (11.26) is pretty much hopeless, because each term $\text{cov}(Y_i, Y_j)$ could be different, in general. Fortunately, systems that have a steady state will, if simulated long enough to pass the transient phase (such as the production-line startup in Example 11.4), produce an output process that is approximately *covariance stationary*. Intuitively, stationarity implies that Y_{i+k} depends on Y_{i+1} in the same manner as Y_k depends on Y_1. In particular, the covariance between two random variables in the time series depends only on the number of observations between them, called the *lag*.

For a covariance-stationary time series, $\{Y_1, Y_2, \ldots\}$, define the lag-k autocovariance by

$$\gamma_k = \text{cov}(Y_1, Y_{1+k}) = \text{cov}(Y_i, Y_{i+k}) \tag{11.27}$$

which, by definition of covariance stationarity, is not a function of i. For $k = 0$, γ_0 becomes the population variance σ^2—that is,

$$\gamma_0 = \text{cov}(Y_i, Y_{i+0}) = V(Y_i) = \sigma^2 \tag{11.28}$$

The lag-k autocorrelation is the correlation between any two observations k apart. It is defined by

$$\rho_k = \frac{\gamma_k}{\sigma^2} \tag{11.29}$$

and has the property that

$$-1 \leq \rho_k \leq 1, \quad k = 1, 2, \ldots$$

If a time series is covariance stationary, then Equation (11.26) can be simplified substantially. Tedious algebra shows that

$$V(\bar{Y}) = \frac{\sigma^2}{n}\left[1 + 2\sum_{k=1}^{n-1}\left(1 - \frac{k}{n}\right)\rho_k\right] \tag{11.30}$$

where ρ_k is the lag-k autocorrelation given by Equation (11.29).

When $\rho_k > 0$ for all k (or most k), the time series is said to be positively autocorrelated. In this case, large observations tend to be followed by large observations, small observations by small ones. Such a series will tend to drift slowly above and then below its mean. Figure 11.7(a) is an example of a stationary time series exhibiting positive autocorrelation. The output data from most queueing simulations are positively autocorrelated.

On the other hand, if some of the $\rho_k < 0$, the series $Y_1, Y_2, \ldots$ will display the characteristics of negative autocorrelation. In this case, large observations tend to be followed by small observations, and vice versa. Figure 11.7(b) is an example of a stationary time series exhibiting negative autocorrelation. The output of certain inventory simulations might be negatively autocorrelated.

[3]This general result can be derived from the fact that, for two random variables Y_1 and Y_2, $V(Y_1 \pm Y_2) = V(Y_1) + V(Y_2) \pm 2\text{cov}(Y_1, Y_2)$.

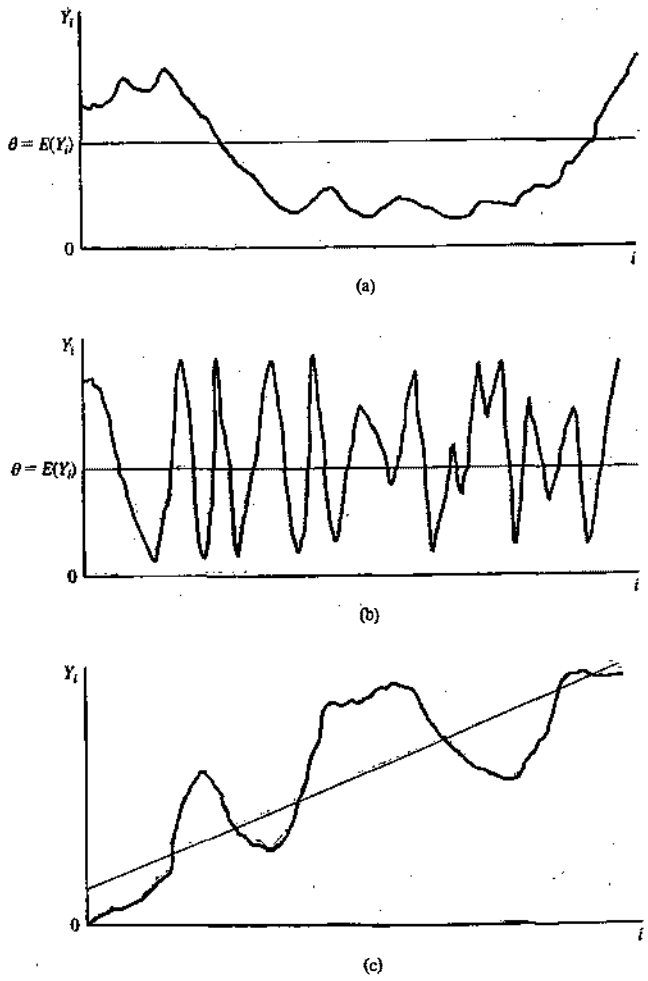


Figure 11.7 (a) Stationary time series Y_i exhibiting positive autocorrelation; (b) stationarytime series Y_i exhibiting negative autocorrelation; (c) nonstationary time series with an upward trend.

Figure 11.7(c) also shows an example of a time series with an upward trend. Such a time series is not stationary; the probability distribution of Y_i is changing with the index i.

Why does autocorrelation make it difficult to estimate $V(\bar{Y})$? Recall that the standard estimator for the variance of a sample mean is S^2/n. By using Equation (11.30), it can be shown [Law, 1977] that the expected value of the variance estimator S^2/n is

$$E\left(\frac{S^2}{n}\right) = BV(\bar{Y}) \tag{11.31}$$

where

$$B = \frac{n/c - 1}{n-1} \tag{11.32}$$

and c is the quantity in brackets in Equation (11.30). The effect of the autocorrelation on the oxestimator S^2/n is derived by an examination of Equations (11.30) and (11.32). There are essentially three possibilities:

Case 1

If the Y_i are independent, then $\rho_k = 0$ for $k = 1, 2, 3, \ldots$. Therefore, $c = 1 + 2\sum_{k=1}^{n-1}(1 - k/n)\rho_k = 1$ and Equation (11.30) reduces to the familiar σ^2/n. Notice also that $B = 1$, so S^2/n is an unbiased estimator of $V(\bar{Y})$. The Y_i will always be independent when they are obtained from different replications; that independence is the primary reason that we prefer experiment designs calling for multiple replications.

Case 2

If the autocorrelations ρ_k are primarily positive, then $c = 1 + 2\sum_{k=1}^{n-1}(1 - k/n)\rho_k > 1$, so that $n/c < n$, and hence $B < 1$. Therefore, S^2/n is biased low as an estimator of $V(\bar{Y})$. If this correlation were ignored, the nominal $100(1 - \alpha)\%$ confidence interval given by Expression (11.12) would be too short, and its true confidence coefficient would be less than $1 - \alpha$. The practical effect would be that the simulation analyst would have unjustified confidence (in the apparent precision of the point estimator) due to the shortness of the confidence interval. If the correlations ρ_k are large, B could be quite small, implying a significant underestimation.

Case 3

If the autocorrelations ρ_k are substantially negative, then $0 \le c < 1$, and it follows that $B > 1$ and S^2/n is biased high for $V(\bar{Y})$. In other words, the true precision of the point estimator $\bar{Y}$ would be greater than what is indicated by its variance estimator S^2/n, because

$$V(\bar{Y}) < E\left(\frac{S^2}{n}\right)$$

As a result, the nominal $100(1 - \alpha)\%$ confidence interval of Expression (11.12) would have true confidence coefficient greater than $1 - \alpha$. This error is less serious than Case 2, because we are unlikely to make incorrect decisions if our estimate is actually more precise than we think it is.

A simple example demonstrates why we are especially concerned about positive correlation: Suppose you want to know how students on a university campus will vote in an upcoming election. To estimate their preferences, you plan to solicit 100 responses. The standard experiment is to randomly select 100 students to poll; call this experiment A. An alternative is to randomly select 20 students and ask each of them to state their preference 5 times in the same day; call this experiment B. Both experiments obtain 100 responses, but clearly an estimate based on experiment B will be less precise (will have larger variance) than an estimate based on experiment A. Experiment A obtains 100 independent responses, whereas experiment B obtains only 20 independent responses and 80 dependent ones. The five opinions from any one student are perfectly positively correlated (assuming a student names the same candidate all five times). Although this is an extreme example, it illustrates that estimates based on positively correlated data are more variable than estimates based on independent data. Therefore, a confidence interval or other measure of error should account correctly for dependent data, but S^2/n does not.

Two methods for eliminating or reducing the deleterious effects of autocorrelation upon estimation of a mean are given in the following sections. Unfortunately, some simulation languages either use or facilitate the use of S^2/n as an estimator of $V(\bar{Y})$, the variance of the sample mean, in all situations. If used uncritically in a simulation with positively autocorrelated output data, the downward bias in S^2/n and the resulting

shortness of a confidence interval for θ will convey the impression of much greater precision than actually exists. When such positive autocorrelation is present in the output data, the true variance of the point estimator, $\bar{Y}$, can be many times greater than is indicated by S^2/n.

11.5.3 Replication Method for Steady-State Simulations

If initialization bias in the point estimator has been reduced to a negligible level (through some combination of intelligent initialization and deletion), then the method of independent replications can be used to estimate point-estimator variability and to construct a confidence interval. The basic idea is simple: Make R replications, initializing and deleting from each one the same way.

If, however, significant bias remains in the point estimator and a large number of replications are used to reduce point-estimator variability, the resulting confidence interval can be misleading. This happens because *bias is not affected by the number of replications* (R); it is affected only by deleting more data (i.e., increasing T_0) or extending the length of each run (i.e., increasing T_E). Thus, increasing the number of replications (R) could produce shorter confidence intervals around the "wrong point." Therefore, it is important to do a thorough job of investigating the initial-condition bias.

If the simulation analyst decides to delete d observations of the total of n observations in a replication, then the point estimator of θ is $\bar{Y}_{\cdot\cdot}(n, d)$, defined by Equation (11.25)—that is, the point estimator is the average of the remaining data. The basic raw output data, $\{Y_{rj}, r = 1,\ldots, R; j = 1,\ldots, n\}$, are exhibited in Table 11.7. Each Y_{rj} is derived in one of the following ways:

Case 1

Y_{rj} is an individual observation from within replication r; for example, Y_{rj} could be the delay of customer j in a queue, or the response time to job j in a job shop.

Case 2

Y_{rj} is a batch mean from within replication r of some number of discrete-time observations. (Batch means are discussed further in Section 11.5.5.)

Case 3

Y_{rj} is a batch mean of a continuous-time process over time interval j; for instance, as in Example 11.14, Equation (11.23) defines Y_{rj} as the time-average (batch mean) number in queue over the interval $[1000(j-1), 1000j)$.

In Case 1, the number d of deleted observations and the total number of observations n might vary from one replication to the next, in which case replace d by d_r and n by n_r. For simplicity, assume that d and n are constant over replications. In Cases 2 and 3, d and n will be constant.

Table 11.7 Raw Output Data from a Steady-State Simulation

	Observations						*Replication*
Replication	*1*	...	*d*	*d + 1*	...	*n*	*Averages*
1	$Y_{1,1}$	...	$Y_{1,d}$	$Y_{1,d+1}$	...	$Y_{1,n}$	$\bar{Y}_{1\cdot}(n, d)$
2	$Y_{2,1}$	...	$Y_{2,d}$	$Y_{2,d+1}$	...	$Y_{2,n}$	$\bar{Y}_{2\cdot}(n, d)$
.	.		.	.		.	.
.	.		.	.		.	.
.	.		.	.		.	.
R	$Y_{R,1}$	...	$Y_{R,d}$	$Y_{R,d+1}$	...	$Y_{R,n}$	$\bar{Y}_{R\cdot}(n, d)$
	$Y_{\cdot 1}$	...	$\bar{Y}_{\cdot d}$	$\bar{Y}_{\cdot,d+1}$	...	$\bar{Y}_{\cdot n}$	$\bar{Y}_{\cdot\cdot}(n, d)$

When using the replication method, each replication is regarded as a single sample for the purpose of estimating θ. For replication r, define

$$\bar{Y}_{r\cdot}(n, d) = \frac{1}{n-d}\sum_{j=d+1}^{n} Y_{rj} \tag{11.33}$$

as the sample mean of all (nondeleted) observations in replication r. Because all replications use different random-number streams and all are initialized at time 0 by the same set of initial conditions (I_0), the replication averages

$$\bar{Y}_{1\cdot}(n, d), \ldots, \bar{Y}_{R\cdot}(n, d)$$

are independent and identically distributed random variables—that is, they constitute a random sample from some underlying population having unknown mean

$$\theta_{n,d} = \mathrm{E}[\bar{Y}_{r\cdot}(n, d)] \tag{11.34}$$

The overall point estimator, given in Equation (11.25), is also given by

$$\bar{Y}_{\cdot\cdot}(n, d) = \frac{1}{R}\sum_{r=1}^{R} \bar{Y}_{r\cdot}(n, d) \tag{11.35}$$

as can be seen from Table 11.7 or from using Equation (11.24). Thus, it follows that

$$E[\bar{Y}_{\cdot\cdot}(n, d)] = \theta_{n,d}$$

also. If d and n are chosen sufficiently large, then $\theta_{n,d} \approx \theta$, and $\bar{Y}_{\cdot\cdot}(n, d)$ is an approximately unbiased estimator of θ. The bias in $\bar{Y}_{\cdot\cdot}(n, d)$ is $\theta_{n,d} - \theta$.

For convenience, when the value of n and d are understood, abbreviate $\bar{Y}_{r\cdot}(n,d)$ (the mean of the undeleted observations from the rth replication) and $\bar{Y}_{\cdot\cdot}(n, d)$ (the mean of $\bar{Y}_{1\cdot}(n, d), \ldots, \bar{Y}_{R\cdot}(n, d)$ by $\bar{Y}_{r\cdot}$ and $\bar{Y}_{\cdot\cdot}$, respectively. To estimate the standard error of $\bar{Y}_{\cdot\cdot}$, first compute the sample variance,

$$S^2 = \frac{1}{R-1}\sum_{r=1}^{R}(\bar{Y}_{r\cdot} - \bar{Y}_{\cdot\cdot})^2 = \frac{1}{R-1}\left(\sum_{r=1}^{R}\bar{Y}_{r\cdot}^2 - R\bar{Y}_{\cdot\cdot}^2\right) \tag{11.36}$$

The standard error of $\bar{Y}_{\cdot\cdot}$ is given by

$$\text{s.e.}(\bar{Y}_{\cdot\cdot}) = \frac{S}{\sqrt{R}} \tag{11.37}$$

A $100(1-\alpha)\%$ confidence interval for θ, based on the t distribution, is given by

$$\bar{Y}_{\cdot\cdot} - t_{\alpha/2,R-1}\frac{S}{\sqrt{R}} \le \theta \le \bar{Y}_{\cdot\cdot} + t_{\alpha/2,R-1}\frac{S}{\sqrt{R}} \tag{11.38}$$

where $t_{\alpha/2,R-1}$ is the $100(1-\alpha/2)$ percentage point of a t distribution with $R-1$ degrees of freedom. This confidence interval is valid only if the bias of $\bar{Y}_{..}$ is approximately zero.

As a rough rule, the length of each replication, beyond the deletion point, should be at least ten times the amount of data deleted. In other words, $(n-d)$ should at least $10d$ (or more generally, T_E should be at least $10T_0$). Given this run length, the number of replications should be as many as time permits, up to about 25 replications. Kelton [1986] established that there is little value in dividing the available time into more than 25 replications, so, if time permits making more than 25 replications of length $T_0 + 10T_0$, then make 25 replications of longer than $T_0 + 10T_0$, instead. Again, these are rough rules that need not be followed slavishly.

Example 11.15

Consider again the *M/G/1* queueing simulation of Examples 11.8 and 11.14. Suppose that the simulation analyst decides to make $R = 10$ replications, each of length $T_E = 15{,}000$ minutes, each starting at time 0 in the empty and idle state, and each initialized for $T_0 = 2000$ minutes before data collection begins. The raw output data consist of the batch means defined by Equation (11.23); recall that each batch mean is simply the average number of customers in queue for a 1000-minute interval. The first two batch means are deleted ($d = 2$). The purpose of the simulation is to estimate, by a 95% confidence interval, the long-run time-average queue length, denoted by L_Q.

The replication averages $\bar{Y}_{r\cdot}(15,2), r = 1,2,\ldots,10$, are shown in Table 11.8 in the rightmost column. The point estimator is computed by Equation (11.35) as

$$\bar{Y}_{\cdot\cdot}(15,2) = 8.43$$

Its standard error is given by Equation (11.37) as

$$\text{s.e.}(\bar{Y}_{\cdot\cdot}(15,2)) = 1.59$$

Table 11.8 Data Summary for M/G/1 Simulation by Replication

Replication, r	*Sample Mean for Replication r*		
	(*No Deletion*) $\bar{Y}_{r\cdot}(15,0)$	(*Delete 1*) $\bar{Y}_{r\cdot}(15,1)$	(*Delete 2*) $\bar{Y}_{r\cdot}(15,2)$
1	3.27	3.24	3.25
2	16.25	17.20	17.83
3	15.19	15.72	15.43
4	7.24	7.28	7.71
5	2.93	2.98	3.11
6	4.56	4.82	4.91
7	8.44	8.96	9.45
8	5.06	5.32	5.27
9	6.33	6.14	6.24
10	10.10	10.48	11.07
$Y_{\cdot\cdot}=(15,d)$	7.94	8.21	8.43
$\sum_{r=1}^{R} \bar{Y}_{r\cdot}^2$	826.20	894.68	938.34
S^2	21.75	24.52	25.30
S	4.66	4.95	5.03
$S/\sqrt{10} = \text{s.e.}(\bar{Y}_{\cdot\cdot})$	1.47	1.57	1.59

and using $\alpha = 0.05$ and $t_{0.025,9} = 2.26$, the 95% confidence interval for long-run mean queue length is given by Inequality (11.38) as

$$8.43 - 2.26(1.59) \le L_Q \le 8.43 + 2.26(1.59)$$

or

$$4.84 \le L_Q \le 12.02$$

The simulation analyst may conclude with a high degree of confidence that the long-run mean queue length is between 4.84 and 12.02 customers. The confidence interval computed here as given by Inequality (11.38) should be used with caution, because a key assumption behind its validity is that enough data have been deleted to remove any significant bias due to initial conditions—that is, that d and n are sufficiently large that the bias $\theta_{n,d} - \theta$ is negligible.

Example 11.16

Suppose that, in Example 11.15, the simulation analyst had decided to delete one batch ($d = 1$) or no batches ($d = 0$). The quantities needed to compute 95% confidence intervals are shown in Table 11.8. The resulting 95% confidence intervals are computed by Inequality (11.38) as follows:

$$(d=1) \quad 4.66 = 8.21 - 2.26(1.57) \le L_Q \le 8.21 + 2.26(1.57) = 11.76$$
$$(d=0) \quad 4.62 = 7.94 - 2.26(1.47) \le L_Q \le 7.94 + 2.26(1.47) = 11.26$$

Notice that, for a fixed total sample size, n, two things happen as fewer data are deleted:

1. The confidence interval shifts downward, reflecting the greater downward bias in $\bar{Y}_{\cdot\cdot}(n, d)$ as d decreases.
2. The standard error of $\bar{Y}_{\cdot\cdot}(n, d)$, namely $S/\sqrt{R}$, decreases as d decreases.

In this example, $\bar{Y}_{\cdot\cdot}(n, d)$ is based on a run length of $T_E = 1000(n-d) = 15{,}000 - 1000d$ minutes. Thus, as d decreases, T_E increases, and, in effect, the sample mean $\bar{Y}_{\cdot\cdot}$ is based on a larger "sample size" (i.e., longer run length). In general, the larger the sample size, the smaller the standard error of the point estimator. This larger sample size can be due to a longer run length (T_E) per replication, or to more replications (R).

Therefore, there is a trade-off between reducing bias and increasing the variance of a point estimator, when the total sample size (R and $T_0 + T_E$) is fixed. The more deletion (i.e., the larger T_0 is and the smaller T_E is, keeping $T_0 + T_E$ fixed), the less bias but greater variance there is in the point estimator.

Recall that each batch in Examples 11.15 and 11.16 consists of 1000 minutes of simulated time. Therefore, discarding $d = 2$ batches really means discarding 2000 minutes of data, a substantial amount. It is not uncommon for very large deletions to be necessary to overcome the initial conditions.

11.5.4 Sample Size in Steady-State Simulations

Suppose it is desired to estimate a long-run performance measure, θ, within $\pm\epsilon$, with confidence $100(1-\alpha)\%$. In a steady-state simulation, a specified precision may be achieved either by increasing the number of replications (R) or by increasing the run length (T_E). The first solution, controlling R, is carried out as given in Section 11.4.2 for terminating simulations.

Example 11.17

Consider the data in Table 11.8 for the *M/G/1* queueing simulation as an initial sample of size $R_0 = 10$. Assuming that $d = 2$ observations were deleted, the initial estimate of variance is $S_0^2 = 25.30$. Suppose that it is desired to estimate long-run mean queue length, L_Q, within $\epsilon = 2$ customers with 90% confidence. The final sample size needed must satisfy Inequality (11.17). Using $\alpha = 0.10$ in Inequality (11.18) yields an initial estimate:

$$R \geq \left(\frac{z_{0.05} S_0}{\epsilon}\right)^2 = \frac{1.645^2(25.30)}{2^2} = 17.1$$

Thus, at least 18 replications will be needed. Proceeding as in Example 11.11, next try $R = 18$, $R = 19$, ... as follows:

R	18	19
$t_{0.05,R-1}$	1.74	1.73
$\left(\frac{t_{0.05,R-1} S_0}{\epsilon}\right)^2$	19.15	18.93

$R = 19 \geq (t_{0.05,18} S_0/\epsilon)^2 = 18.93$ is the smallest integer R satisfying Inequality (11.17), so a total sample size of $R = 19$ replications is needed to estimate L_Q to within ±2 customers. Therefore, $R - R_0 = 19 - 10 = 9$ additional replications are needed to achieve the specified error.

An alternative to increasing R is to increase total run length $T_0 + T_E$ within each replication. If the calculations in Section 11.4.2, as illustrated in Example 11.17, indicate that $R - R_0$ additional replications are needed beyond the initial number, R_0, then an alternative is to increase run length $(T_0 + T_E)$ in the same proportion (R/R_0) to a new run length $(R/R_0)(T_0 + T_E)$. Thus, additional data will be deleted, from time 0 to time $(R/R_0)T_0$, and more data will be used to compute the point estimates, as illustrated by Figure 11.8. However, the total amount of simulation effort is the same as if we had simply increased the number of replications but maintained the same run length. The advantage of increasing total run length per replication and deleting a fixed proportion $[T_0/(T_0 + T_E)]$ of the total run length is that any residual bias in the point estimator should be further reduced by the additional deletion of data at the beginning of the run. A possible

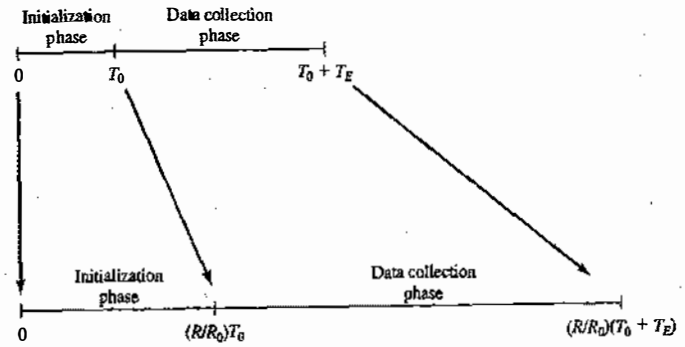


Figure 11.8 Increasing runlength to achieve specified accuracy.

disadvantage of the method is that, in order to continue the simulation of all R replications [from time $T_0 + T_E$ to time $(R/R_0)(T_0 + T_E)$], it is necessary to have saved the state of the model at time $T_0 + T_E$ and to be able to restart the model and run it for the additional required time. Otherwise, the simulations would have to be remn from time 0, which could be time consuming for a complex model. Some simulation languages have the capability to save enough information that a replication can be continued from time T_E onward, rather than having to start over from time 0.

Example 11.18

In Example 11.17, suppose that run length was to be increased to achieve the desired error, ±2 customers. Since $R/R_0 = 19/10 = 1.9$, the run length should be almost doubled to $(R/R_0)(T_0 + T_E) = 1.9(15{,}000) = 28{,}500$ minutes. The data collected from time 0 to time $(R/R_0)T_0 = 1.9(2000) = 3800$ minutes would be deleted, and the data from time 3800 to time 28,500 used to compute new point estimates and confidence intervals.

11.5.5 Batch Means for Interval Estimation in Steady-State Simulations

One disadvantage of the replication method is that data must be deleted on each replication and, in one sense, deleted data are wasted data, or at least lost information. This suggests that there might be merit in using an experiment design that is based on a single, long replication. The disadvantage of a single-replication design arises when we try to compute the standard error of the sample mean. Since we only have data from within one replication, the data are dependent, and the usual estimator is biased.

The method of *batch means* attempts to solve this problem by dividing the output data from one replication (after appropriate deletion) into a few large batches and then treating the means of these batches as if they were independent. When the raw output data after deletion form a continuous-time process, $\{Y(t), T_0 \leq t \leq T_0 + T_E\}$, such as the length of a queue or the level of inventory, then we form k batches of size $m = T_E/k$ and compute the batch means as

$$\bar{Y}_j = \frac{1}{m} \int_{(j-1)m}^{jm} Y(t + T_0)\, dt$$

for $j = 1, 2, \ldots, k$. In other words, the jth batch mean is just the time-weighted average of the process over the time interval $[T_0 + (j-1)m, T_0 + jm)$, exactly as in Example 11.8.

When the raw output data after deletion form a discrete-time process, $\{Y_i, i = d+1, d+2, \ldots, n\}$, such as the customer delays in a queue or the cost per period of an inventory system, then we form k batches of size $m = (n-d)/k$ and compute the batch means as

$$\bar{Y}_j = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} Y_{i+d}$$

for $j = 1, 2, \ldots, k$ (assuming k divides $n - d$ evenly, otherwise round down to the nearest integer). That is, the batch means are formed as shown here:

$$\underbrace{Y_1, \ldots, Y_d}_{\text{deleted}}, \underbrace{Y_{d+1}, \ldots, Y_{d+m}}_{\bar{Y}_1}, \underbrace{Y_{d+m+1}, \ldots, Y_{d+2m}}_{\bar{Y}_2}, \ldots, \underbrace{Y_{d+(k-1)m+1}, \ldots, Y_{d+km}}_{\bar{Y}_k}$$

Starting with either continuous-time or discrete-time data, the variance of the sample mean is estimated by

$$\frac{S^2}{k} = \frac{1}{k} \sum_{j=1}^{k} \frac{(\bar{Y}_j - \bar{\bar{Y}})^2}{k-1} = \frac{\sum_{j=1}^{k} \bar{Y}_j^2 - k\bar{\bar{Y}}^2}{k(k-1)} \qquad (11.39)$$

where $\bar{Y}$ is the overall sample mean of the data after deletion. As was discussed in Section 11.2, the batch means $\bar{Y}_1, \bar{Y}_2, ..., \bar{Y}_k$ are not independent; however, if the batch size is sufficiently large, successive batch means will be approximately independent, and the variance estimator will be approximately unbiased.

Unfortunately, there is no widely accepted and relatively simple method for choosing an acceptable batch size m (or equivalently choosing a number of batches k). But there are some general guidelines that can be culled from the research literature:

- Schmeiser [1982] found that for a *fixed total sample size* there is little benefit from dividing it into more than $k = 30$ batches, even if we could do so and still retain independence between the batch means. Therefore, there is no reason to consider numbers of batches much greater than 30, no matter how much raw data are available. He also found that the performance of the confidence interval, in terms of its width and the variability of its width, is poor for fewer than 10 batches. Therefore, a number of batches between 10 and 30 should be used in most applications.
- Although there is typically autocorrelation between batch means at all lags, the lag-1 autocorrelation $\rho_1 = \text{corr}(\bar{Y}_j, \bar{Y}_{j+1})$ is usually studied to assess the dependence between batch means. When the lag-1 autocorrelation is nearly 0, then the batch means are treated as independent. This approach is based on the observation that the autocorrelation in many stochastic processes decreases as the lag increases. Therefore, all lag autocorrelations should be smaller (in absolute value) than the lag-1 autocorrelation.
- The lag-1 autocorrelation between batch means can be estimated as described shortly. However, the autocorrelation should not be estimated from a small number of batch means (such as the $10 \le k \le 30$ recommended above); there is bias in the autocorrelation estimator. Law and Carson [1979] suggest estimating the lag-1 autocorrelation from a large number of batch means based on a smaller batch size (perhaps $100 \le k \le 400$). When the autocorrelation between these batch means is approximately 0, then the autocorrelation will be even smaller if we rebatch the data to between 10 and 30 batch means based on a larger batch size. Hypothesis tests for 0 autocorrelation are available, as described next.
- If the *total sample size is to be chosen sequentially*, say to attain a specified precision, then it is helpful to allow the batch size and number of batches to grow as the run length increases. It can be shown that a good strategy is to allow the number of batches to increase as the square root of the sample size after first finding a batch size at which the lag-1 autocorrelation is approximately 0. Although we will not discuss this point further, an algorithm based on it can be found in Fishman and Yarberry [1997]; see also Steiger and Wilson [2002].

Given these insights, we recommend the following general strategy:

1. Obtain output data from a single replication and delete as appropriate. Recall our guideline: collecting at least 10 times as much data as is deleted.
2. Form up to $k = 400$ batches (but at least 100 batches) with the retained data, and compute the batch means. Estimate the sample lag-1 autocorrelation of the batch means as

$$\hat{\rho}_1 = \frac{\sum_{j=1}^{k-1}(\bar{Y}_j - \bar{Y})(\bar{Y}_{j+1} - \bar{Y})}{\sum_{j=1}^{k}(\bar{Y}_j - \bar{Y})^2}$$

3. Check the correlation to see whether it is sufficiently small.
   (a) If $\hat{\rho}_1 \le 0.2a$, then rebatch the data into $30 \le k \le 40$ batches, and form a confidence interval using $k-1$ degrees of freedom for the t distribution and Equation (11.39) to estimate the variance of $\bar{Y}$.
   (b) If $\hat{\rho}_1 > 0.2$, then extend the replication by 50% to 100% and go to Step 2. If it is not possible to extend the replication, then rebatch the data into approximately $k = 10$ batches, and form the confidence interval, using $k-1$ degrees of freedom for the t distribution and Equation (11.39) to estimate the variance of $\bar{Y}$.

4. As an additional check on the confidence interval, examine the batch means (at the larger or smaller batch size) for independence, using the following test. (See, for instance, Alexopoulos and Seila [1998].) Compute the test statistic

$$C = \sqrt{\frac{k^2 - 1}{k - 2}}\left(\hat{\rho}_1 + \frac{(\bar{Y}_1 - \bar{Y})^2 + (\bar{Y}_k - \bar{Y})^2}{2\sum_{j=1}^{k}(\bar{Y}_j - \bar{Y})^2}\right)$$

   If $C < z_\beta$ then accept the independence of the batch means, where β is the Type I error level of the test (such as 0.1, 0.05, 0.01). Otherwise, extend the replication by 50% to 100% and go to Step 2. If it is not possible to extend the replication, then rebatch the data into approximately $k = 10$ batches, and form the confidence interval, using $k-1$ degrees of freedom for the t distribution and Equation (11.39) to estimate the variance of $\bar{Y}$.

This procedure, including the final check, is conservative in several respects. First, if the lag-1 autocorrelation is substantially negative then we proceed to form the confidence interval anyway. A dominant negative correlation tends to make the confidence interval wider than necessary, which is an error, but not one that will cause us to make incorrect decisions. The requirement that $\hat{\rho}_1 < 0.2$ at $100 \le k \le 400$ batches is pretty stringent and will tend to force us to get more data (and therefore create larger batches) if there is any hint of positive dependence. And finally, the hypothesis test at the end has a probability of β of forcing us to get more data when none are really needed. But this conservatism is by design; the cost of an incorrect decision is typically much greater than the cost of some additional computer run time.

The batch-means approach to confidence-interval estimation is illustrated in the next example.

Example 11.19

Reconsider the *M/G/*1 simulation of Example 11.8, except that the mean service time is changed from 9.5 minutes to 7 minutes (implying a long-run server utilization of 0.7). Suppose that we want to estimate the steady-state expected delay in queue, w_Q, by a 95% confidence interval. To illustrate the method of batch means, assume that one run of the model has been made, simulating 3000 customers after the deletion point. We then form batch means from $k = 100$ batches of size $m = 30$ and estimate the lag-1 autocorrelation to be $\hat{\rho}_1 = 0.346 > 0.2$. Thus, we decide to extend the simulation to 6000 customers after the deletion point, and again we estimate the lag-1 autocorrelation. This estimate, based on $k = 100$ batches of size $m = 60$, is $\hat{\rho}_1 = 0.004 < 0.2$.

Having passed the correlation check, we rebatch the data into $k = 30$ batches of size $m = 200$. The point estimate is the overall mean

$$\bar{Y} = \frac{1}{6000}\sum_{j=1}^{6000}\bar{Y}_j = 9.04$$

minutes. The variance of $\bar{Y}$, computed from the 30 batch means, is

$$\frac{S^2}{k} = \frac{\sum_{j=1}^{30}\bar{Y}_j^2 - 30\bar{Y}^2}{30(29)} = 0.604$$

Thus, a 95% confidence interval is given by

$$\bar{Y} - t_{0.025,29}\sqrt{0.604} \le w_Q \le \bar{Y} + t_{0.025,29}\sqrt{0.604}$$

or

$$7.45 = 9.04 - 2.04(0.777) \le w_Q \le 9.04 + 2.04(0.777) = 10.63$$

Thus, we assert with 95% confidence that true mean delay in queue, w_Q, is between 7.45 and 10.63 minutes. If these results are not sufficiently precise for practical use, the run length should be increased to achieve greater precision.

As a further check on the validity of the confidence interval, we can apply the correlation hypothesis test. To do so, we compute the test statistic from the $k = 30$ batches of size $m = 200$ used to form the confidence interval. This gives

$$C = -0.31 < 1.96 = z_{0.05}$$

confirming the lack of correlation at the 0.05 significance level. Notice that, at this small number of batches, the estimated lag-1 autocorrelation appears to be slightly negative, illustrating our point about the difficulty of estimating correlation with small numbers of observations.

11.5.6 Quantiles

Constructing confidence intervals for quantile estimates in a steady-state simulation can be tricky, especially if the output process of interest is a continuous-time process, such as $L_Q(t)$, the number of customers in queue at time t. In this section, we outline the main issues.

Taking the easier case first, suppose that the output process from a single replication (after appropriate deletion of initial data) is $Y_{d+1}, \ldots, Y_n$. To be concrete, Y_i might be the delay in queue of the ith customer. Then the point estimate of the pth quantile can be obtained as before, either from the histogram of the data or from the sorted values. Of course, only the data after the deletion point are used. Suppose we make R replications and let $\hat{\theta}_r$ be the quantile estimate from the rth. Then the R quantile estimates, $\hat{\theta}_1, \ldots, \hat{\theta}_R$, are independent and identically distributed. Their average is

$$\hat{\theta}_\cdot = \frac{1}{R}\sum_{i=1}^{R}\hat{\theta}_i$$

It can be used as the point estimator of θ; and an approximate confidence interval is

$$\hat{\theta}_\cdot \pm t_{\alpha/2,R-1}\frac{S}{\sqrt{R}}$$

where S^2 is the usual sample variance of $\hat{\theta}_1, \ldots, \hat{\theta}_R$.

What if only a single replication is obtained? Then the same reasoning applies if we let $\hat{\theta}_i$ be the quantile estimate from *within* the ith batch of data. This requires sorting the data, or forming a histogram, within each batch. If the batches are large enough, then these within-batch quantile estimates will also be approximately i.i.d.

When we have a continuous-time output process, then, in principle, the same methods apply. However, we must be careful not to transform the data in a way that changes the problem. In particular, we cannot first form batch means—as we have done throughout this chapter—and then estimate the quantile from these batch means. The p quantile of the *batch means* of $L_Q(t)$ is *not* the same as the p quantile of $L_Q(t)$ itself. Thus, the quantile point estimate must be formed from the histogram of the raw data—either from each run, if we make replications, or within each batch, if we make a single replication.

11.6 SUMMARY

This chapter emphasized the idea that a stochastic discrete-event simulation is a statistical experiment. Therefore, before sound conclusions can be drawn on the basis of the simulation-generated output data, a proper statistical analysis is required. The purpose of the simulation experiment is to obtain estimates of the performance measures of the system under study. The purpose of the statistical analysis is to acquire some assurance that these estimates are sufficiently precise for the proposed use of the model.

A distinction was made between terminating simulations and steady-state simulations. Steady-state simulation output data are more difficult to analyze, because the simulation analyst must address the problem of initial conditions and the choice of run length. Some suggestions were given regarding these problems, but unfortunately no simple, complete, and satisfactory solution exists. Nevertheless, simulation analysts should be aware of the potential problems, and of the possible solutions—namely, deletion of data and increasing of the run length. More advanced statistical techniques (not discussed in this text) are given in Alexopoulos and Seila [1998], Bratley, Fox, and Schrage [1996], and Law and Kelton [2000].

The statistical precision of point estimators can be measured by a standard-error estimate or by a confidence interval. The method of independent replications was emphasized. With this method, the simulation analyst generates statistically independent observations, and thus standard statistical methods can be employed. For steady-state simulations, the method of batch means was also discussed.

The main point is that simulation output data contain some amount of random variability; without some assessment of its size, the point estimates cannot be used with any degree of reliability.

REFERENCES

ALEXOPOULOS, C., AND A. F. SEILA [1998], "Output Data Analysis," Chapter 7 in *Handbook of Simulation*, J. Banks, ed., Wiley, New York.

BRATLEY, P., B. L. FOX, AND L. E. SCHRAGE [1996], *A Guide to Simulation*, 2d ed., Springer-Verlag, New York.

FISHMAN, G. S., AND L. S. YARBERRY [1997], "An Implementation of the Batch Means Method," *INFORMS Journal on Computing*, Vol. 9, pp. 296–310.

GOLDSMAN, D., L. SCHRUBEN, AND J. J. SWAIN [1994], "Tests for Transient Means in Simulated Time Series," *Naval Research Logistics*, Vol. 41, pp. 171–187.

KELTON, W. D. [1986], "Replication Splitting and Variance for Simulating Discrete-Parameter Stochastic Processes," *Operations Research Letters*, Vol. 4, pp. 275–279.

KELTON, W. D. [1989], "Random Initialization Methods in Simulation," *IIE Transactions*, Vol. 21, pp. 355–367.

KELTON, W. D., AND A. M. LAW [1983], "A New Approach for Dealing with the Startup Problem in Discrete Event Simulation," *Naval Research Logistics Quarterly*, Vol. 30, pp. 641–658.

KLEIJNEN, J. P. C. [1987], *Statistical Tools for Simulation Practitioners*, Dekker, New York.

LAW, A. M. [1977], "Confidence Intervals in Discrete Event Simulation: A Comparison of Replication and Batch Means," *Naval Research Logistics Quarterly*, Vol. 24, pp. 667–78.

LAW, A. M. [1980], "Statistical Analysis of the Output Data from Terminating Simulations," *Naval Research Logistics Quarterly*, Vol. 27, pp. 131–43.

LAW, A. M., AND J. S. CARSON [1979], "A Sequential Procedure for Determining the Length of a Steady-State Simulation," *Operations Research*, Vol. 27, pp. 1011–1025.

LAW, A. M., AND W. D. KELTON [2000], *Simulation Modeling and Analysis*, 3d ed., McGraw-Hill, New York.

NELSON, B. L. [2001], "Statistical Analysis of Simulation Results," Chapter 94 in *Handbook of Industrial Engineering*, 3d ed., G. Salvendy, ed., Wiley, New York.

SCHMEISER, B. [1982], "Batch Size Effects in the Analysis of Simulation Output," *Operations Research*, Vol. 30, pp. 556–568.

SCHRUBEN, L. [1980], "Detecting Initialization Bias in Simulation Output," *Operations Research*, Vol. 30, pp. 569–590.

SNELL, M., AND L. SCHRUBEN [1985], "Weighting Simulation Data to Reduce Initialization Effects," *IIE Transactions*, Vol. 17, pp. 354–363.

STEIGER, N. M., AND J. R. Wilson [2002], "An Improved Batch Means Procedure for Simulation Output Analysis," *Management Science*, Vol. 48, pp. 1569–1586.

WELCH, P. D. [1983], "The Statistical Analysis of Simulation Results," in *The Computer Performance Modeling Handbook*, S. Lavenberg, ed., Academic Press, New York, pp. 268–328.

EXERCISES

1. Suppose that, in Example 11.14, the simulation analyst decided to investigate the bias by using batch means over a batching interval of 2000 minutes. By definition, a batch mean for the interval $[(j-1)2000, j(2000))$ is defined by

$$Y_j = \frac{1}{2000}\int_{(j-1)2000}^{j(2000)} L_Q(t)\,dt$$

   (a) Show algebraically that such a batch mean can be obtained from two adjacent batch means over the two halves of the interval.
   (b) Compute the seven averaged batch means for the intervals [0, 2000), [2000, 4000), ... for the *M/G/*1 simulation. Use the data $(\bar{Y}_{\cdot j})$ in Table 11.6 (ignoring $\bar{Y}_{\cdot 15} = 8.76$).
   (c) Draw plots of the type used in Figures 11.4 and 11.5. Does it still appear that deletion of the data over [0, 2000) (the first "new" batch mean) is sufficient to remove most of the point-estimator bias?

2. Suppose, in Example 11.14, that the simulation analyst could only afford to run 5 independent replications (instead of 10). Use the batch means in Table 11.5 for replications 1 to 5 to compute a 95% confidence interval for mean queue length L_Q. Investigate deletion of initial data. Compare the results from using 5 replications with those from using 10 replications.

3. In Example 11.7, suppose that management desired 95% confidence in the estimate of mean system time w and that the error allowed was $\varepsilon = 0.4$ minute. Using the same initial sample of size $R_0 = 4$ (given in Table 11.1), figure out the required total sample size.

4. Simulate the dump-truck problem in Example 3.4. At first, make the run length $T_E = 40$ hours. Make four independent replications. Compute a 90% confidence interval for mean cycle time, where a cycle time for a given truck is the time between its successive arrivals to the loader. Investigate the effect of different initial conditions (all trucks initially at the loader queue, versus all at the scale, versus all traveling, versus the trucks distributed throughout the system in some manner).

5. Consider an (M, L) inventory system, in which the procurement quantity, Q, is defined by

$$Q = \begin{cases} M - I & \text{if } I < L \\ 0 & \text{if } I \geq L \end{cases}$$

   where I is the level of inventory on hand plus on order at the end of a month, M is the maximum inventory level, and L is the reorder point. M and L are under management control, so the pair (M, L) is called the inventory policy. Under certain conditions, the analytical solution of such a model is possible, but the computational effort can be prohibitive. Use simulation to investigate an (M, L) inventory system with the following properties: The inventory status is checked at the end of each month. Backordering is allowed at a cost of \$4 per item short per month. When an order arrives, it will first be used to relieve the backorder. The lead time is given by a uniform distribution on the interval (0.25, 1.25) months. Let the beginning inventory level stand at 50 units, with no orders outstanding. Let the holding cost be \$1 per unit in inventory per month. Assume that the inventory position is reviewed each month. If an order is placed, its cost is \$60 + \$5Q, where \$60 is the ordering cost and \$5 is the cost of each item. The time between demands is exponentially distributed with a mean of 1/15 month. The sizes of the demands follow this distribution:

Demand	*Probability*
1	1/2
2	1/4
3	1/8
4	1/8

   (a) Make four independent replications, each of run length 100 months preceded by a 12-month initialization period, for the $(M, L) = (50, 30)$ policy. Estimate long-run mean monthly cost with a 90% confidence interval.
   (b) Using the results of part (a), estimate the total number of replications needed to estimate mean monthly cost within \$5.

6. Reconsider Exercise 6, except that, if the inventory level at a monthly review is zero or negative, a rush order for Q units is placed. The cost for a rush order is \$120+\$12Q, where \$120 is the ordering cost and \$12 is the cost of each item. The lead time for a rush order is given by a uniform distribution on the interval (0.10, 0.25) months.

   (a) Make four independent replications for the (M, L) policy, and estimate long-run mean monthly cost with a 90% confidence interval.
   (b) Using the results of part (a), estimate the total number of replications needed to estimate mean monthly cost within \$5.

7. Suppose that the items in Exercise 6 are perishable, with a selling price given by the following data:

On the Shelf (Months)	*Selling Price*
0-1	\$10
1-2	5
>2	0

   Thus, any item that has been on the shelf greater than 2 months cannot be sold. The age is measured at the time the demand occurs. If an item is outdated, it is discarded, and the next item is brought forward. Simulate the system for 100 months.

   (a) Make four independent replications for the $(M, L) = (50, 30)$ policy, and estimate long-run mean monthly cost with a 90% confidence interval.
   (b) Using the results of part (a), estimate the total number of replications needed to estimate mean monthly cost within \$5.

   At first, assume that all the items in the beginning inventory are fresh. Is this a good assumption? What effect does this "all-fresh" assumption have on the estimates of long- run mean monthly cost? What can be done to improve these estimates? Carry out a complete analysis.

8. Consider the following inventory system:

   (a) Whenever the inventory level falls to or below 10 units, an order is placed. Only one order can be outstanding at a time.
   (b) The size of each order is Q. Maintaining an inventory costs \$0.50 per day per item in inventory. Placing an order incurs a fixed cost, \$10.00.
   (c) Lead time is distributed in accordance with a discrete uniform distribution between zero and 5 days.
   (d) If a demand occurs during a period when the inventory level is zero, the sale is lost at a cost of \$2.00 per unit.

(e) The number of customers each day is given by the following distribution:

Number of Customers per Day	*Probability*
1	0.23
2	0.41
3	0.22
4	0.14

(f) The demand on the part of each customer is Poisson distributed with a mean of 3 units.

(g) For simplicity, assume that all demands occur at noon and that all orders are placed immediately thereafter.

Assume further that orders are received at 5:00 P.M., or after the demand that occurred on that day. Consider the policy having $Q = 20$. Make five independent replications, each of length 100 days, and compute a 90% confidence interval for long-run mean daily cost. Investigate the effect of initial inventory level and existence of an outstanding order on the estimate of mean daily cost. Begin with an initial inventory of $Q + 10$ and no outstanding orders.

9. A store selling Mother's Day cards must decide 6 months in advance on the number of cards to stock. Reordering is not allowed. Cards cost \$0.45 and sell for \$1.25. Any cards not sold by Mother's Day go on sale for \$0.50 for 2 weeks. However, sales of the remaining cards is probabilistic in nature according to the following distribution:

32% of the time, all cards remaining get sold.

40% of the time, 80% of all cards remaining are sold.

28% of the time, 60% of all cards remaining are sold.

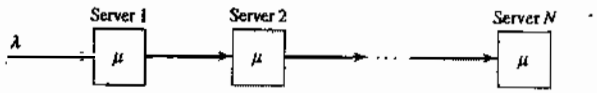
Any cards left after 2 weeks are sold for \$0.25. The card-shop owner is not sure how many cards can be sold, but thinks it is somewhere (i.e., uniformly distributed) between 200 and 400. Suppose that the card-shop owner decides to order 300 cards. Estimate the expected total profit with an error of at most \$5.00. (*Hint*: Make three or four initial replications. Use these data to estimate the total sample size needed. Each replication consists of one Mother's Day.)

10. A very large mining operation has decided to control the inventory of high-pressure piping by a "periodic review, order up to M" policy, where M is a target level. The annual demand for this piping is normally distributed, with mean 600 and variance 800. This demand occurs fairly uniformly over the year. The lead time for resupply is Erlang distributed of order $k = 2$ with its mean at 2 months. The cost of each unit is \$400. The inventory carrying charge, as a proportion of item cost on an annual basis, is expected to fluctuate normally about the mean 0.25 (simple interest), with a standard deviation of 0.01. The cost of making a review and placing an order is \$200, and the cost of a backorder is estimated to be \$100 per unit backordered. Suppose that the inventory level is reviewed every 2 months, and let $M = 337$.

(a) Make five independent replications, each of run length 100 months, to estimate long-run mean monthly cost by means of a 90% confidence interval.

(b) Investigate the effects of initial conditions. Calculate an appropriate number of monthly observations to delete to reduce initialization bias to a negligible level.

11. Consider some number, say N, of $M/M/1$ queues in series. The $M/M/1$ queue, described in Section 6.4, has Poisson arrivals at some rate λ customers per hour, exponentially distributed service times with mean $1/\mu$, and a single server. (Recall that "Poisson arrivals" means that interarrival times are exponentially distributed.) By $M/M/1$ queues in series, it is meant that, upon completion of service at a given server, a customer joins a waiting line for the next server. The system can be shown as follows:



All service times are exponentially distributed with mean $1/\mu$, and the capacity of each waiting line is assumed to be unlimited. Assume that $\lambda = 8$ customers per hour and $1/\mu = 0.1$ hour. The measure of performance is response time, which is defined to be the total time a customer is in the system.

(a) By making appropriate simulation runs, compare the initialization bias for $N = 1$ (i.e., one $M/M/1$ queue) to $N = 2$ (i.e., two $M/M/1$ queues in series). Start each system with all servers idle and no customers present. The purpose of the simulation is to estimate mean response time.

(b) Investigate the initialization bias as a function of N, for $N = 1, 2, 3, 4$, and 5.

(c) Draw some general conclusions concerning initialization bias for "large" queueing systems when at time 0 the system is assumed to be empty and idle.

12. Jobs enter a job shop in random fashion according to a Poisson process at a stationary overall rate, two every 8-hour day. The jobs are of four types. They flow from work station to work station in a fixed order, depending on type, as shown next. The proportions of each type are also shown.

Type	*Flow through Stations*	*Proportion*
1	1, 2, 3, 4	0.4
2	1, 3, 4	0.3
3	2, 4, 3	0.2
4	1, 4	0.1

Processing times per job at each station depend on type, but all times are (approximately) normally distributed with mean and s.d. (in hours) as follows:

	Station			
Type	*1*	*2*	*3*	*4*
1	(20, 3)	(30, 5)	(75, 4)	(20, 3)
2	(18, 2)		(60, 5)	(10, 1)
3		(20, 2)	(50, 8)	(10, 1)
4	(30, 5)			(15, 2)

Station i will have c_i workers ($i = 1, 2, 3, 4$). Each job occupies one worker at a station for the duration of a processing time. All jobs are processed on a first-in–first-out basis, and all queues for waiting jobs are assumed to have unlimited capacity. Simulate the system for 800 hours, preceded by a 200-hour initialization period. Assume that $c_1 = 8$, $c_2 = 8$, $c_3 = 20$, $c_4 = 7$. Based on $R = 5$ replications, compute a 97.5% confidence interval for average worker utilization at each of the four stations. Also, compute a

95% confidence interval for mean total response time for each job type, where a total response time is the total time that a job spends in the shop.

13. Change Exercise 12 to give priority at each station to the jobs by type. Type 1 jobs have priority over type 2, type 2 over type 3, and type 3 over type 4. Use 800 hours as run length, 200 hours as initialization period, and $R = 5$ replications. Compute four 97.5% confidence intervals for mean total response time by type. Also, run the model without priorities and compute the same confidence intervals. Discuss the trade-offs when using *first in, first out* versus a priority system.

14. Consider a single-server queue with Poisson arrivals at rate $\lambda = 10.82$ per minute and normally distributed service times with mean 5.1 seconds and variance 0.98 seconds^2. It is desired to estimate the mean time in the system for a customer who, upon arrival, finds i other customers in the system—that is, to estimate

$$w_i = E(W \mid N = i) \qquad \text{for } i = 0, 1, 2, \ldots$$

where W is a typical system time and N is the number of customers found by an arrival. For example, w_0 is the mean system time for those customers who find the system empty, w_1 is the mean system time for those customers who find one other customer present upon arrival, and so on. The estimate $\hat{w}_i$ of w_i will be a sample mean of system times taken over all arrivals who find i in the system. Plot $\hat{w}_i$ vs i. Hypothesize and attempt to verify a relation between w_i and i.

(a) Simulate for a 10-hour period with empty and idle initial conditions.
(b) Simulate for a 10-hour period after an initialization of one hour. Are there observable differences in the results of (a) and (b)?
(c) Repeat parts (a) and (b) with service times exponentially distributed with mean 5.1 seconds.
(d) Repeat parts (a) and (b) with deterministic service times equal to 5.1 seconds.
(e) Find the number of replications needed to estimate $w_0, w_1, \ldots, w_6$ with a standard error for each of at most 3 seconds. Repeat parts (a)–(d), but using this number of replications.

15. At Smalltown U., there is one specialized graphics workstation for student use located across campus from the computer center. At 2:00 A.M. one day, six students arrive at the workstation to complete an assignment. A student uses the workstation for 10 ± 8 minutes, then leaves to go to the computer center to pick up graphics output. There is a 25% chance that the run will be OK and the student will go to sleep. If it is not OK, the student returns to the workstation and waits until it becomes free. The roundtrip from workstation to computer center and back takes 30 ± 5 minutes. The computer becomes inaccessible at 5:00 A.M. Estimate the probability, p, that at least five of the six students will finish their assignment in the 3-hour period. First, make $R = 10$ replications, and compute a 95% confidence interval for p. Next, work out the number of replications needed to estimate p within $\pm.02$, and make this number of replications. Recompute the 95% confidence interval for p.

16. Four workers are spaced evenly along a conveyor belt. Items needing processing arrive according to a Poisson process at the rate 2 per minute. Processing time is exponentially distributed, with mean 1.6 minutes. If a worker becomes idle, then he or she takes the first item to come by on the conveyor. If a worker is busy when an item comes by, that item moves down the conveyor to the next worker, taking 20 seconds between two successive workers. When a worker finishes processing an item, the item leaves the system. If an item passes by the last worker, it is recirculated on a loop conveyor and will return to the first worker after 5 minutes.

Management is interested in having a balanced workload—that is, management would like worker utilizations to be equal. Let ρ_i be the long-run utilization of worker i, and let ρ be the average utilization of all workers. Thus, $\rho = (\rho_1 + \rho_2 + \rho_3 + \rho_4)/4$. According to queueing theory, ρ can be estimated by $\rho = \lambda/c\mu$, where $\lambda = 2$ arrivals per minute, $c = 4$ servers, and $1/\mu = 1.6$ minutes is the mean service time. Thus, $\rho = \lambda/c\mu = (2/4)1.6 = 0.8$; so, on the average, a worker will be busy 80% of the time.

(a) Make 5 independent replications, each of run length 40 hours preceded by a one hour initialization period. Compute 95% confidence intervals for ρ_1 and ρ_4. Draw conclusions concerning workload balance.
(b) Based on the same 5 replications, test the hypothesis $H_0: \rho_1 = 0.8$ at a level of significance $\alpha = 0.05$. If a difference of $\pm.05$ is important to detect, determine the probability that such a deviation is detected. In addition, if it is desired to detect such a deviation with probability at least 0.9, figure out the sample size needed to do so. (See any basic statistics textbook for guidance on hypothesis testing.)
(c) Repeat (b) for $H_0: \rho_4 = 0.8$.
(d) From the results of (a)–(c), draw conclusions for management about the balancing of workloads.

17. At a small rock quarry, a single power shovel dumps a scoop full of rocks at the loading area approximately every 10 minutes, with the actual time between scoops modeled well as being exponentially distributed, with mean 10 minutes. Three scoops of rocks make a pile; whenever one pile of rocks is completed, the shovel starts a new pile.

The quarry has a single truck that can carry one pile (3 scoops) at a time. It takes approximately 27 minutes for a pile of rocks to be loaded into the truck and for the truck to be driven to the processing plant, unloaded, and return to the loading area. The actual time to do these things (altogether) is modeled well as being normally distributed, with mean 27 minutes and standard deviation 12 minutes.

When the truck returns to the loading area, it will load and transport another pile if one is waiting to be loaded; otherwise, it stays idle until another pile is ready. For safety reasons, no loading of the truck occurs until a complete pile (all three scoops) is waiting.

The quarry operates in this manner for an 8-hour day. We are interested in estimating the utilization of the trucks and the expected number of piles waiting to be transported if an additional truck is purchased.

18. Big Bruin, Inc. plans to open a small grocery store in Juneberry, NC. They expect to have two checkout lanes, with one lane being reserved for customers paying with cash. The question they want to answer is: how many grocery carts do they need?

During business hours (6 A.M.–8 P.M.), cash-paying customers are expected to arrive at 8 per hour. All other customers are expected to arrive at 9 per hour. The time between arrivals of each type can be modeled as exponentially distributed random variables.

The time spent shopping is modeled as normally distributed, with mean 40 minutes and standard deviation 10 minutes. The time required to check out after shopping can be modeled as lognormally distributed, with (a) mean 4 minutes and standard deviation 1 minute for cash-paying customers; (b) mean 6 minutes and standard deviation 1 minute for all other customers.

We will assume that every customer uses a shopping cart and that a customer who finishes shopping leaves the cart in the store so that it is available immediately for another customer. We will also assume that any customer who cannot obtain a cart immediately leaves the store, disgusted.

The primary performance measures of interest to Big Bruin are the expected number of shopping carts in use and the expected number of customers lost per day. Recommend a number of carts for the store, remembering that carts are expensive, but so are lost customers.

19. Develop a simulation model of the total time in the system for an $M/M/1$ queue with service rate $\mu = 1$; therefore, the traffic intensity is $\rho = \lambda/\mu = \lambda$, the arrival rate. Use the simulation, in conjunction with

the technique of plotting ensemble averages, to study the effect of traffic intensity on initialization bias when the queue starts empty. Specifically, see how the initialization phase T_0 changes for $\rho = 0.5, 0.7, 0.8, 0.9, 0.95$.

20. The average waiting data from 10 replication of a queuing system are

Replication	*Average Waiting Time*
1	1.77
2	2.50
3	1.87
4	3.22
5	3.00
6	2.11
7	3.12
8	3.49
9	2.39
10	3.49

Determine 90% confidence interval for the average waiting time.

21. Consider Example 6. If it is required to estimate the average waiting time with an absolute error of 0.25 and confidence level of 90%, determine the nnmber of replications required.

22. In a queuing simulation with 20 replications, 90% confidence interval for average queue length is found to be in the range 1.72–2.41. Determine the probability that the average queue length is less than 2.75.

23. Collect papers dealing with simulation output analysis and study the tools used.

12

Comparison and Evaluation of Alternative System Designs

Chapter 11 dealt with the precise estimation of a measure of performance for one system. This chapter discusses a few of the many statistical methods that can be used to compare two or more system designs on the basis of some performance measure. One of the most important uses of simulation is the comparison of alternative system designs. Because the observations of the response variables contain random variation, statistical analysis is needed to discover whether any observed differences are due to differences in design or merely to the random fluctuation inherent in the models.

The comparison of two system designs is computationally easier than the simultaneous comparison of multiple (more than two) system designs. Section 12.1 discusses the case of two system designs, using two possible statistical techniques: *independent sampling* and *correlated sampling*. Correlated sampling is also known as the *common random numbers* (CRN) technique; simply put, the same random numbers are used to simulate both alternative system designs. If implemented correctly, CRN usually reduces the variance of the estimated difference of the performance measures and thus can provide, for a given sample size, more precise estimates of the mean difference than can independent sampling. Section 12.2 extends the statistical techniques of Section 12.1 to the comparison of multiple (more than two) system designs, using the Bonferroni approach to confidence-interval estimation, screening, and selecting the best. The Bonferroni approach is limited to twenty or fewer system designs, but Section 12.3 describes how a large number of complex system designs can sometimes be represented by a simpler metamodel. Finally, for comparison and evaluation of a very large number of system designs that are related in a less structured way, Section 12.4 presents optimization via simulation.

12.1 COMPARISON OF TWO SYSTEM DESIGNS

Suppose that a simulation analyst desires to compare two possible configurations of a system. In a queueing system, perhaps two possible queue disciplines, or two possible sets of servers, are to be compared. In a supply-chain inventory system, perhaps two possible ordering policies will be compared. A job shop could have two possible scheduling rules; a production system could have in-process inventory buffers of various capacities. Many other examples of alternative system designs can be provided.

The method of replications will be used to analyze the output data. The mean performance measure for system i will be denoted by $\theta_i (i = 1,2)$. If it is a steady-state simulation, it will be assumed that deletion of data, or other appropriate techniques, have been used to ensure that the point estimators are approximately unbiased estimators of the mean performance measures, θ_i. The goal of the simulation experiments is to obtain point and interval estimates of the difference in mean performance, namely $\theta_1 - \theta_2$. Three methods of computing a confidence interval for $\theta_1 - \theta_2$ will be discussed, but first an example and a general framework will be given.

Example 12.1

A vehicle-safety inspection station performs three jobs: (1) brake check, (2) headlight check, and (3) steering check. The present system has three stalls in parallel; that is, a vehicle enters a stall, where one attendant makes all three inspections. The current system is illustrated in Figure 12.1(a). Using data from the existing system, it has been assumed that arrivals occur completely at random (i.e., according to a Poisson process) at an average rate of 9.5 per hour and that the times for a brake check, a headlight check, and a steering check are normally distributed with means of 6.5, 6, and 5.5 minutes, respectively, all having standard deviations of approximately 0.5 minute. There is no limit on the queue of waiting vehicles.

An alternative system design is shown in Figure 12.1(b). Each attendant will specialize in a single task, and each vehicle will pass through three work stations in series. No space is allowed for vehicles between the brake and headlight check, or between the headlight and steering check. Therefore, a vehicle in the brake or headlight check must move to the next attendant, and a vehicle in the steering check must exit before the next vehicle can move ahead. The increased specialization of the inspectors suggests that mean inspection times for each type of check will decrease by 10%: to 5.85, 5.4, and 4.95 minutes, respectively, for the brake, headlight, and steering inspections. The Safety Inspection Council has decided to compare the two systems on the basis of mean response time per vehicle, where a response time is defined as the total time from a vehicle arrival until its departure from the system.

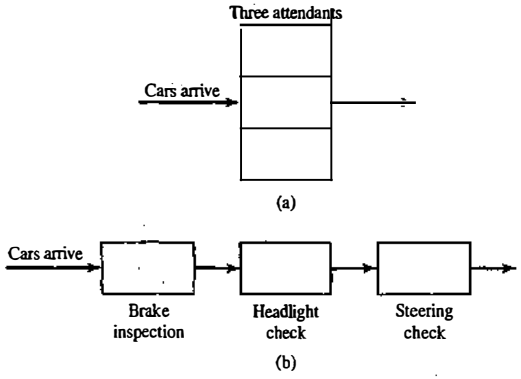


Figure 12.1 Vehicle safety inspection station and a possible alternative design.

When comparing two systems, such as those in Example 12.1, the simulation analyst must decide on a run length $T_E^{(i)}$ for each model ($i = 1, 2$), and a number of replications R_i to be made of each model. From replication r of system i, the simulation analyst obtains an estimate Y_{ri} of the mean performance measure, θ_i. In Example 12.1, Y_{ri} would be the average response time observed during replication r for system i ($r = 1, \ldots, R_i$; $i = 1, 2$). The data, together with the two summary measures, the sample means $\bar{Y}_{\cdot i}$, and the sample variances S_i^2, are exhibited in Table 12.1. Assuming that the estimators Y_{ri} are (at least approximately) unbiased, it follows that

$$\theta_1 = E(Y_{r1}),\ r = 1, \ldots, R_1;\ \theta_2 = E(Y_{r2}),\ r = 1, \ldots, R_2$$

In Example 12.1, the Safety Inspection Council is interested in a comparison of the two system designs, so the simulation analyst decides to compute a confidence interval for $\theta_1 - \theta_2$, the difference between the two mean performance measures. The confidence interval is used to answer two questions: (1) How large is the mean difference, and how precise is the estimator of mean difference? (2) Is there a significant difference between the two systems? This second question will lead to one of three possible conclusions:

1. If the confidence interval (c.i.) for $\theta_1 - \theta_2$ is totally to the left of zero, as shown in Figure 12.2(a), then there is strong evidence for the hypothesis that $\theta_1 - \theta_2 < 0$, or equivalently $\theta_1 < \theta_2$.

Table 12.1 Simulation Output Data and Summary Measures for Comparing Two Systems

System	Replication 1	Replication 2	...	Replication R_i	Sample Mean	Sample Variance
1	Y_{11}	Y_{21}	...	$Y_{R1}1$	$\bar{Y}_{\cdot 1}$	S_1^2
2	Y_{12}	Y_{22}	...	$Y_{R2}2$	$\bar{Y}_{\cdot 2}$	S_2^2

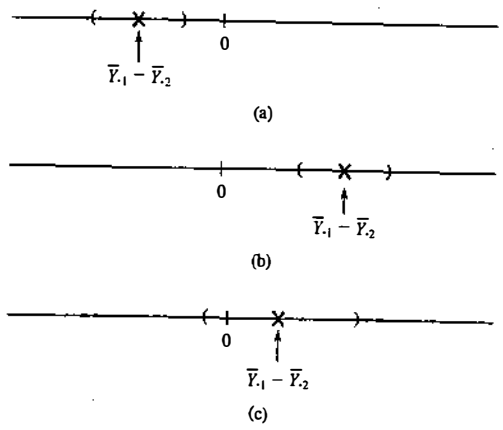


Figure 12.2 Three confidence intervals that can occur in the comparing of two systems.

In Example 12.1, $\theta_1 < \theta_2$ implies that the mean response time for system 1 (the original system) is smaller than for system 2 (the alternative system).

2. If the c.i. for $\theta_1 - \theta_2$ is totally to the right of zero, as shown in Figure 12.2(b), then there is strong evidence that $\theta_1 - \theta_2 > 0$, or equivalently, $\theta_1 > \theta_2$.

In Example 12.1, $\theta_1 > \theta_2$ can be interpreted as system 2 being better than system 1, in the sense that system 2 has smaller mean response time.

3. If the c.i. for $\theta_1 - \theta_2$ contains zero, then, in the data at hand, there is no strong statistical evidence that one system design is better than the other.

Some statistics textbooks say that the weak conclusion $\theta_1 = \theta_2$ can be drawn, but such statements can be misleading. A "weak" conclusion is often no conclusion at all. Most likely, if enough additional data were collected (i.e., R_i increased), the c.i. would shift, and definitely shrink in length, until conclusion 1 or 2 would be drawn. In addition to one of these three conclusions, the confidence interval provides a measure of the precision of the estimator of $\theta_1 - \theta_2$.

In this chapter, a two-sided $100(1-\alpha)\%$ c.i. for $\theta_1 - \theta_2$ will always be of the form

$$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2} \pm t_{\alpha/2,\nu}\,\text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) \tag{12.1}$$

where $\bar{Y}_{\cdot i}$ is the sample mean performance measure for system i over all replications

$$\bar{Y}_{\cdot i} = \frac{1}{R_i}\sum_{r=1}^{R_i} Y_{ri} \tag{12.2}$$

and ν is the degrees of freedom associated with the variance estimator, $t_{\alpha/2,\nu}$ is the $100(1-\alpha/2)$ percentage point of a t distribution with ν degrees of freedom, and s.e.$(\cdot)$ represents the standard error of the specified point estimator. To obtain the standard error and the degrees of freedom, the analyst uses one of three statistical techniques. All three techniques assume that the basic data, Y_{ri} of Table 12.1, are approximately normally distributed. This assumption is reasonable provided that each Y_{ri} is itself a sample mean of observations from replication r (which is indeed the situation in Example 12.1).

By design of the simulation experiment, $Y_{r1}(r = 1, \ldots, R_1)$ are independently and identically distributed (i.i.d.) with mean θ_1 and variance σ_1^2 (say). Similarly, $Y_{r2}(r = 1, \ldots, R_2)$ are i.i.d. with mean θ_2 and variance σ_2^2 (say). The three techniques for computing the confidence interval in (12.1), which are based on three different sets of assumptions, are discussed in the following subsections.

There is an important distinction between *statistically significant* differences and *practically significant* differences in systems performance. Statistical significance answers the following question: Is the observed difference $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$ larger than the variability in $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$? This question can be restated as: Have we collected enough data to be confident that the difference we observed is real, or just chance? Conclusions 1 and 2 imply a statistically significant difference, while Conclusion 3 implies that the observed difference is not statistically significant (even though the systems may indeed be different). Statistical significance is a function of the simulation experiment and the output data.

Practical significance answers the following question: Is the true difference $\theta_1 - \theta_2$ large enough to matter for the decision we need to make? In Example 12.1, we may reach the conclusion that $\theta_1 > \theta_2$ and decide that system 2 is better (smaller expected response time). However, if the actual difference $\theta_1 - \theta_2$ is very small—say, small enough that a customer would not notice the improvement— then it might not be worth the cost to replace system 1 with system 2. Practical significance is a function of the actual difference between the systems and is independent of the simulation experiment.

Confidence intervals do not answer the question of practical significance directly. Instead, they bound (with probability $1 - \alpha$) the true difference $\theta_1 - \theta_2$ within the range

$$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2} - t_{\alpha/2,\nu}\,\text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) \le \theta_1 - \theta_2 \le \bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2} + t_{\alpha/2,\nu}\,\text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2})$$

Whether a difference within these bounds is practically significant depends on the particular problem.

12.1.1 Independent Sampling with Equal Variances

Independent sampling means that different and independent random number streams will be used to simulate the two systems. This implies that all the observations of simulated system 1, namely $\{Y_{r1}, r = 1, \ldots, R_1\}$, are statistically independent of all the observations of simulated system 2, namely $\{Y_{r2}, r = 1, \ldots, R_2\}$. By Equation (12.2) and the independence of the replications, the variance of the sample mean, $\bar{Y}_{\cdot i}$, is given by

$$\text{V}(\bar{Y}_{\cdot i}) = \frac{\text{V}(Y_{ri})}{R_i} = \frac{\sigma_i^2}{R_i}, \quad i = 1, 2$$

For independent sampling, $\bar{Y}_{\cdot 1}$ and $\bar{Y}_{\cdot 2}$ are statistically independent; hence,

$$\begin{aligned}\text{V}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) &= \text{V}(\bar{Y}_{\cdot 1}) + \text{V}(\bar{Y}_{\cdot 2}) \\ &= \frac{\sigma_1^2}{R_1} + \frac{\sigma_2^2}{R_2}\end{aligned} \tag{12.3}$$

In some cases, it is reasonable to assume that the two variances are equal (but unknown in value); that is, $\sigma_1^2 = \sigma_2^2$. The data can be used to test the hypothesis of equal variances; if rejected, the method of Section 12.1.2 must be used. In a steady-state simulation, the variance σ_i^2 decreases as the run length $T_E^{(i)}$ increases; therefore, it might be possible to adjust the two run lengths, $T_E^{(1)}$ and $T_E^{(2)}$, to achieve at least approximate equality of σ_1^2 and σ_2^2.

If it is reasonable to assume that $\sigma_1^2 = \sigma_2^2$ (approximately), a two-sample-t confidence-interval approach can be used. The point estimate of the mean performance difference is

$$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2} \tag{12.4}$$

with $\bar{Y}_{\cdot i}$ given by Equation (12.2). Next, compute the sample variance for system i by

$$\begin{aligned}S_i^2 &= \frac{1}{R_i - 1}\sum_{r=1}^{R_i}(Y_{ri} - \bar{Y}_{\cdot i})^2 \\ &= \frac{1}{R_i - 1}\left(\sum_{r=1}^{R_i} Y_{ri}^2 - R_i\bar{Y}_{\cdot i}^2\right)\end{aligned} \tag{12.5}$$

Note that S_i^2 is an unbiased estimator of the variance σ_i^2. By assumption, $\sigma_1^2 = \sigma_2^2 = \sigma^2$ (say), so a pooled estimate of σ^2 is obtained by

$$S_p^2 = \frac{(R_1 - 1)S_1^2 + (R_2 - 1)S_2^2}{R_1 + R_2 - 2}$$

which has $\nu = R_1 + R_2 - 2$ degrees of freedom. The c.i. for $\theta_1 - \theta_2$ is then given by Expression (12.1) with the standard error computed by

$$\text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) = S_p\sqrt{\frac{1}{R_1} + \frac{1}{R_2}} \tag{12.6}$$

This standard error is an estimate of the standard deviation of the point estimate, which, by Equation (12.3), is given by $\sigma\sqrt{1/R_1 + 1/R_2}$.

In some cases, the simulation analyst could have $R_1 = R_2$, in which case it is safe to use the c.i. in Expression (12.1) with the standard error taken from Equation (12.6), even if the variances (σ_1^2 and σ_2^2) are not equal. However, if the variances are unequal and the sample sizes differ, it has been shown that use of the two-sample-t c.i. could yield invalid confidence intervals whose true probability of containing $\theta_1 - \theta_2$ is much less than $1 - \alpha$. Thus, if there is no evidence that $\sigma_1^2 = \sigma_2^2$, and if $R_1 \neq R_2$, the approximate procedure in the next subsection is recommended.

12.1.2 Independent Sampling with Unequal Variances

If the assumption of equal variances cannot safely be made, an approximate $100(1 - \alpha)\%$ c.i. for $\theta_1 - \theta_2$ can be computed as follows. The point estimate and sample variances are computed by Equations (12.4) and (12.5). The standard error of the point estimate is given by

$$\text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) = \sqrt{\frac{S_1^2}{R_1} + \frac{S_2^2}{R_2}} \tag{12.7}$$

with degrees of freedom, ν, approximated by the expression

$$\nu = \frac{(S_1^2/R_1 + S_2^2/R_2)^2}{[(S_1^2/R_1)^2/(R_1 - 1)] + [(S_2^2/R_2)^2/(R_2 - 1)]} \tag{12.8}$$

rounded to an integer. The confidence interval is then given by Expression (12.1), using the standard error of Equation (12.7). A minimum number of replications $R_1 \geq 6$ and $R_2 \geq 6$ is recommended for this procedure.

12.1.3 Common Random Numbers (CRN)

CRN means that, for each replication, the same random numbers are used to simulate both systems. Therefore, R_1 and R_2 must be equal, say $R_1 = R_2 = R$. Thus, for each replication r, the two estimates, Y_{r1} and Y_{r2}, are no longer independent, but rather are correlated. However, independent streams of random numbers are used on different replications, so the pairs (Y_{r1}, Y_{s2}) are mutually independent when $r \neq s$. (For example, in Table 12.1, the observation Y_{11} is correlated with Y_{12}, but Y_{11} is independent of all other observations.) The purpose of using CRN is to induce a positive correlation between Y_{r1} and Y_{r2} (for each r) and thus to achieve a variance reduction in the point estimator of mean difference, $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$. In general, this variance is given by

$$\begin{aligned} V(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) &= V(\bar{Y}_{\cdot 1}) + V(\bar{Y}_{\cdot 2}) - 2\text{cov}(\bar{Y}_{\cdot 1}, \bar{Y}_{\cdot 2}) \\ &= \frac{\sigma_1^2}{R} + \frac{\sigma_2^2}{R} - \frac{2\rho_{12}\sigma_1\sigma_2}{R} \end{aligned} \tag{12.9}$$

where ρ_{12} is the correlation between Y_{r1} and Y_{r2}. [By definition, $\rho_{12} = \text{cov}(Y_{r1}, Y_{r2})/\sigma_1\sigma_2$, which does not depend on r.]

Now compare the variance of $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$ arising from the use of CRN [Equation (12.9), call it V_{CRN} to the variance arising from the use of independent sampling with equal sample sizes [Equation (12.3) with $R_1 = R_2 = R$, call it V_{IND}]. Notice that

$$V_{CRN} = V_{IND} - \frac{2\rho_{12}\sigma_1\sigma_2}{R} \tag{12.10}$$

If CRN works as intended, the correlation ρ_{12} will be positive; hence, the second term on the right side of Equation (12.9) will be positive, and, therefore,

$$V_{CRN} < V_{IND}$$

That is, the variance of the point estimator will be smaller with CRN than with independent sampling. A smaller variance (for the same sample size R) implies that the estimator based on CRN is more precise, leading to a shorter confidence interval on the difference, which implies that smaller differences in performance can be detected.

To compute a $100(1 - \alpha)\%$ c.i. with correlated data, first compute the differences

$$D_r = Y_{r1} - Y_{r2} \tag{12.11}$$

which, by the definition of CRN, are i.i.d.; then compute the sample mean difference as

$$\bar{D} = \frac{1}{R}\sum_{r=1}^{R} D_r \tag{12.12}$$

(Thus, $\bar{D} = \bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$.) The sample variance of the differences $\{D_r\}$ is computed as

$$\begin{aligned} S_D^2 &= \frac{1}{R-1}\sum_{r=1}^{R}(D_r - \bar{D})^2 \\ &= \frac{1}{R-1}\left(\sum_{r=1}^{R} D_r^2 - R\bar{D}^2\right) \end{aligned} \tag{12.13}$$

which has degrees of freedom $\nu = R - 1$. The $100(1 - \alpha)\%$ c.i. for $\theta_1 - \theta_2$ is given by Expression (12.1), with the standard error of $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$ estimated by

$$\text{s.e.}(\bar{D}) = \text{s.e.}(\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}) = \frac{S_D}{\sqrt{R}} \tag{12.14}$$

Because $S_D/\sqrt{R}$ of Equation (12.14) is an estimate of $\sqrt{V_{CRN}}$ and Expression (12.6) or (12.7) is an estimate of $\sqrt{V_{IND}}$, CRN typically will produce a c.i. that is shorter for a given sample size than the c.i. produced by independent sampling if $\rho_{12} > 0$. In fact, the expected length of the c.i. will be shorter with use of CRN if $\rho_{12} > 0.1$, provided $R > 10$. The larger R is, the smaller ρ_{12} can be and still yield a shorter expected length [Nelson 1987].

For any problem, there are many ways of implementing common random numbers. It is never enough to simply use the same seed on the random-number generator(s). Each random number used in one model for some purpose should be used for the same purpose in the second model—that is, the use of the random numbers must be synchronized. For example, if the ith random number is used to generate a service time at

work station 2 for the 5th arrival in model 1, the ith random number should be used for the very same purpose in model 2. For queueing systems or service facilities, synchronization of the common random numbers guarantees that the two systems face identical work loads: both systems face arrivals at the same instants of time, and these arrivals demand equal amounts of service. (The actual service times of a given arrival in the two models may not be equal; they could be proportional if the server in one model were faster than the server in the other model.) For an inventory system, in comparing of different ordering policies, synchronization guarantees that the two systems face identical demand for a given product. For production or reliability systems, synchronization guarantees that downtimes for a given machine will occur at exactly the same times and will have identical durations, in the two models. On the other hand, if some aspect of one of the systems is totally different from the other system, synchronization could be inappropriate—or even impossible to achieve. In summary, those aspects of the two system designs that are sufficiently similar should be simulated with common random numbers in such a way that the two models "behave" similarly; but those aspects that are totally different should be simulated with independent random numbers.

Implementation of common random numbers is model dependent, but certain guidelines can be given that will make CRN more likely to yield a positive correlation. The purpose of the guidelines is to ensure that synchronization occurs:

1. Dedicate a random-number stream to a specific purpose, and use as many different streams as needed. (Different random-number generators, or widely spaced seeds on the same generator, can be used to get two different, nonoverlapping streams.) In addition, assign independently chosen seeds to each stream at the beginning of each replication. It is not sufficient to assign seeds at the beginning of the first replication and then let the random-number generator merely continue for the second and subsequent replications. If simulation is conducted in this manner, the first replication will be synchronized, but subsequent replications might not be.
2. For systems (or subsystems) with external arrivals: As each entity enters the system, the next interarrival time is generated, and then immediately all random variables (such as service times, order sizes, etc.) needed by the arriving entity and identical in both models are generated in a fixed order and stored as attributes of the entity, to be used later as needed. Apply guideline 1: Dedicate one random-number stream to these external arrivals and all their attributes.
3. For systems having an entity performing given activities in a cyclic or repeating fashion, assign a random-number stream to this entity. (Example: a machine that cycles between two states: up–down–up–down–.... Use a dedicated random-number stream to generate the uptimes and downtimes.)
4. If synchronization is not possible, or if it is inappropriate for some part of the two models, use independent streams of random numbers for this subset of random variates.

Unfortunately, there is no guarantee that CRN will always induce a positive correlation between comparable runs of the two models. It is known that if, for each input random variate X, the estimators Y_{r1} and Y_{r2} are increasing functions of the random variate X (or both are decreasing functions of X), then ρ_{12} will be positive. The intuitive idea is that both models (i.e., both Y_{r1} and Y_{r2}) respond in the same direction to each input random variate, and this results in positive correlation. This increasing or decreasing nature of the response variables (called *monotonicity*) with respect to the input random variables is known to hold for certain queueing systems (such as the *GI/G/c* queues), when the response variable is customer delay, so some evidence exists that common random numbers is a worthwhile technique for queueing simulations. (For simple queues, customer delay is an increasing function of service times and a decreasing function of interarrival times.) Wright and Ramsay [1979] reported a negative correlation for certain inventory simulations, however. In summary, the guidelines recently described should be followed, and some reasonable notion that the response variable of interest is a monotonic function of the random input variables should be evident.

Example 12.1: Continued

The two inspection systems shown in Figure 12.1 will be compared by using both independent sampling and CRN, in order to illustrate the greater precision of CRN when it works.

Each vehicle arriving to be inspected has four input random variables associated with it:

A_n = interarrival time between vehicles n and $n+1$

$S_n^{(1)}$ = brake inspection time for vehicle n in model 1

$S_n^{(2)}$ = headlight inspection time for vehicle n in model 1

$S_n^{(3)}$ = steering inspection time for vehicle n in model 1

For model 2 (of the proposed system), mean service times are decreased by 10%. When using independent sampling, different values of service (and interarrival) times would be generated for models 1 and 2 by using different random numbers. But when using CRN, the random number generator should be used in such a way that exactly the same values are generated for $A_1, A_2, A_3, \ldots$ in both models. For service times, however, we do not want the same service times in both models, because the mean service time for model 2 is 10% smaller, but we do want strongly correlated service times. There are at least two ways to do this:

1. Let $S_n^{(i)}(i = 1,2,3; n = 1,2,\ldots)$ be the service times generated for model 1; then use $S_n^{(i)} - 0.1E(S_n^{(i)})$ as the service times in model 2. In words, we take each service time from model 1 and subtract 10% of its true mean.
2. Recall that normal random variates are usually produced by first generating a standard normal variate and then using Equation (8.29) to obtain the correct mean and variance. Therefore, the service times for, say, a brake inspection could be generated by

$$E(S_n^{(1)}) + \sigma Z_n^{(1)} \tag{12.15}$$

where $Z_n^{(1)}$ is a standard normal variate, $\sigma = 0.5$ minute, but $E(S_n^{(1)}) = 6.5$ minutes for model 1 and $E(S_n^{(1)}) = 5.85$ minutes (10% less) for model 2. The other two inspection times would be generated in a similar fashion. To implement (synchronized) common random numbers, the simulation analyst would generate identical $Z_n^{(i)}$ sequences ($i = 1, 2, 3$; $n = 1, 2, \ldots$) in both models and then use the appropriate version of Equation (12.15) to generate the inspection times.

For the synchronized runs, the service times for a vehicle were generated at the instant of arrival (by guideline 2) and stored as an attribute of the vehicle, to be used as needed. Runs were also made with non-synchronized common random numbers, in which case one random number stream was used as needed.

Table 12.2 gives the average response time for each of $R = 10$ replications, each of run length $T_E = 16$ hours. It was assumed that two cars were present at time 0, waiting to be inspected. Column 1 gives the outputs from model 1. Model 2 was run with independent random numbers (column 2I) and with common random numbers without synchronization (column 2C*) and with synchronization (column 2C). The purpose of the simulation is to estimate mean difference in response times for the two systems.

For the two independent runs (1 and 2I), it was assumed that the variances were not necessarily equal, so the method of Section 12.1.2 was applied. Sample variances and the standard error were computed by Equations (12.5) and (12.7), yielding

$$S_1^2 = 118.9, \qquad S_{2I}^2 = 244.3$$

and

$$\text{s.e.}(\bar{Y}_1 - \bar{Y}_{2I}) = \sqrt{\frac{118.9}{10} + \frac{244.3}{10}} = 6.03$$

Table 12.2 Comparison of System Designs for the Vehicle-Safety Inspection System

	Average Response Time for Model				*Observed Differences*	
Replication	*1*	*21*	*2C**	*2C*	$D_{1,2C^*}$	$D_{1,2C}$
1	29.59	51.62	56.47	29.55	−26.88	0.04
2	23.49	51.91	33.34	24.26	−9.85	−0.77
3	25.68	45.27	35.82	26.03	−10.14	−0.35
4	41.09	30.85	34.29	42.64	6.80	−1.55
5	33.84	56.15	39.07	32.45	−5.23	1.39
6	39.57	28.82	32.07	37.91	7.50	1.66
7	37.04	41.30	51.64	36.48	−14.60	0.56
8	40.20	73.06	41.41	41.24	−1.21	−1.04
9	61.82	23.00	48.29	60.59	13.53	1.23
10	44.00	28.44	22.44	41.49	21.56	2.51
Sample mean	37.63	43.04			−1.85	0.37
Sample variance	118.90	244.33			208.94	1.74
Standard error	6.03				4.57	0.42

with degrees of freedom, ν, equal to 17, as given by Equation (12.8). The point estimate is $\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 21} = -5.4$ minutes, and a 95% c.i. [Expression (12.1)] is given by

$$-5.4 \pm 2.11(6.03)$$

or

$$-18.1 \le \theta_1 - \theta_2 \le 7.3 \tag{12.16}$$

The 95% confidence interval in Inequality (12.16) contains zero, which indicates that there is no strong evidence that the observed difference, −5.4 minutes, is due to anything other than random variation in the output data. In other words, it is not statistically significant. Thus, if the simulation analyst had decided to use independent sampling, no strong conclusion would be possible, because the estimate of $\theta_1 - \theta_2$ is quite imprecise.

For the two sets of correlated runs (1 and 2C*, and 1 and 2C), the observations are paired and analyzed as given in Equations (12.11) through (12.14). The point estimate when not synchronizing the random numbers is given by Equation (12.12) as

$$\bar{D} = -1.9 \text{ minutes}$$

the sample variance by S_D^2 (with $\nu = 9$ degrees of freedom), and the standard error by s.e.$(\bar{D}) = 4.6$. Thus, a 95% c.i. for the true mean difference in response times, as given by expression (12.1), is

$$-1.9 \pm 2.26(4.6)$$

or

$$-12.3 < \theta_1 - \theta_2 < 8.5 \tag{12.17}$$

Again, no strong conclusion is possible, because the confidence interval contains zero. Notice, however, that the estimate of $\theta_1 - \theta_2$ is slightly more precise than that in Inequality (12.16), because the length of the interval is smaller.

When complete synchronization of the random numbers was used, in run 2C, the point estimate of the mean difference in response times was

$$\bar{D} = 0.4 \text{ minute}$$

the sample variance was $S_D^2 = 1.7$ (with $\nu = 9$ degrees of freedom), and the standard error was s.e.$(\bar{D}) = 0.4$. A 95% c.i. for the true mean difference is given by

$$-0.50 < \theta_1 - \theta_2 < 1.30 \tag{12.18}$$

The confidence interval in Inequality (12.18) again contains zero, but it is considerably shorter than the previous two intervals. This greater precision in the estimation of $\theta_1 - \theta_2$ is due to the use of synchronized common random numbers. The short length of the interval in Inequality (12.18) suggests that the true difference, $\theta_1 - \theta_2$, is close to zero. In fact, the upper bound, 1.30, indicates that system 2 is at most 1.30 minutes faster, in expectation. If such a small difference is not practically significant, then there is no need to look further into which system is truly better.

As is seen by comparing the confidence intervals in inequalities (12.16), (12.17), and (12.18), the width of the confidence interval is reduced by 18% when using nonsynchronized common random numbers, by 93% when using common random numbers with full synchronization. Comparing the estimated variance of $\bar{D}$ when using synchronized common random numbers with the variance of $\bar{Y}_1 - \bar{Y}_2$ when using independent sampling shows a variance reduction of 99.5%, which means that, to achieve precision comparable to that achieved by CRN, a total of approximately $R = 209$ independent replications would have to be made.

The next few examples show how common random numbers can be implemented in other contexts.

Example 12.2: The Dump-Truck Problem, Revisited

Consider Example 3.4 (the dump-truck problem), shown in Figure 3.7. Each of the trucks repeatedly goes through three activities: loading, weighing, and traveling. Assume that there are eight trucks and that, at time 0, all eight are at the loaders. Weighing time per truck on the single scale is uniformly distributed between 1 and 9 minutes, and travel time per truck is exponentially distributed, with mean 85 minutes. An unlimited queue is allowed before the loader(s) and before the scale. All trucks can be traveling at the same time. Management desires to compare one fast loader against the two slower loaders currently being used. Each of the slow loaders can fill a truck in from 1 to 27 minutes, uniformly distributed. The new fast loader can fill a truck in from 1 to 19 minutes, uniformly distributed. The basis for comparison is mean system response time, where a response time is defined as the duration of time from a truck arrival at the loader queue to that truck's departure from the scale.

To implement synchronized common random numbers, a separate and distinct random number stream was assigned to each of the eight trucks. At the beginning of each replication (i.e., at time 0), a new and independently chosen set of eight seeds was specified, one seed for each random number stream. Thus, weighing times and travel times for each truck were identical in both models, and the loading time for a given truck's ith visit to the fast loader was proportional to the loading time in the original system (with two slow loaders). Implementation of common random numbers without synchronization (e.g., using one random number stream to generate all loading, weighing, and travel times as needed) would likely lead to a given random number being used to generate a loading time in model 1 but a travel time in model 2, or vice versa, and from that point on the use of a random number would most likely be different in the two models.

Table 12.3 Comparison of System Designs for the Dump Truck Problem

Replication	Average Response Time for Model			Differences, $D_{1,2C}$
	1 (2 Loaders)	*2I* (1 Loader)	*2C* (1 Loader)	
1	21.38	29.01	24.30	−2.92
2	24.06	24.70	27.13	−3.07
3	21.39	26.85	23.04	−1.65
4	21.90	24.49	23.15	−1.25
5	23.55	27.18	26.75	−3.20
6	22.36	26.91	25.62	−3.26
Sample mean	22.44	26.52		−2.56
Sample variance	1.28	2.86		0.767
Sample standard deviation	1.13	1.69		0.876

Six replications of each model were run, each of run length $T_E = 40$ hours. The results are shown in Table 12.3. Both independent sampling and CRN were used, to illustrate the advantage of CRN. The first column (labeled model 1) contains the observed average system response time for the existing system with two loaders. The columns labeled 2I and 2C are for the alternative design having one loader; the independent sampling results are in 2I, and the CRN results are in the column labeled 2C. The rightmost column, labeled $D_{1,2C}$, lists the observed differences between the runs of model 1 and model 2C.

For independent sampling assuming unequal variances, the following summary statistics were computed by using Equations (12.2), (12.5), (12.7), (12.8), and (12.1) and the data (in columns 1 and 2I) in Table 12.3:

Point Estimate: $\bar{Y}_{.1} - \bar{Y}_{.2I} = 22.44 - 26.52 = -4.08$ minutes

Sample variances: $S_1^2 = 1.28$, $S_{2I}^2 = 2.86$

Standard Error: $\text{s.e.}(\bar{Y}_{.1} - \bar{Y}_{.2}) = (S_1^2 / R_1 + S_{2I}^2 / R_2)^{1/2} = 0.831$

Degrees of freedom: $\nu = 8.73 \approx 9$

95% c.i. for $\theta_1 - \theta_2$: $-4.08 \pm 2.26(0.831)$ or -4.08 ± 1.878

$$-5.96 \le \theta_1 - \theta_2 \le -2.20$$

For CRN, implemented by the use of synchronized common random numbers, the following summary statistics were computed by using Equations (12.12), (12.13), (12.14), and (12.1) plus the data (in columns 1 and 2C) in Table 12.3:

Point Estimate: $\bar{D} = \bar{Y}_{.1} - \bar{Y}_{.2C} = -2.56$ minutes

Sample variance: $S_D^2 = 0.767$

Standard Error: $\text{s.e.}(\bar{D}) = S_D / \sqrt{R} = 0.876 / \sqrt{6} = 0.358$

Degrees of freedom: $\nu = R - 1 = 5$

95% c.i. for $\theta_1 - \theta_2$: $-2.56 \pm 2.57(0.358)$ or -2.56 ± 0.919

$$-3.48 \le \theta_1 - \theta_2 \le -1.641$$

By comparing the c.i. widths, we see that the use of CRN with synchronization reduced c.i. width by 50%. This reduction could be important if a difference of as much as, say, 5.96 is considered practically

significant, but a difference of at most 3.48 is not. Equivalently, if equal precision were desired, independent sampling would require approximately four times as many observations as would CRN: approximately 24 replications of each model instead of six.

To illustrate how CRN can fail when not implemented correctly, consider the dump-truck model again. There were eight trucks, and each was assigned its own random number stream. For each of the six replications, eight seeds were randomly chosen, one seed for each random number stream. Therefore, a total of 48 (6 times 8) seeds were specified for the correct implementation of common random numbers. When the authors first developed and ran this example, eight seeds were specified at the beginning of the first replication only; on the remaining five replications the random numbers were generated by continuing down the eight original streams. Since comparable replications with one and two loaders required different numbers of random variables, only the first replications of the two models were synchronized. The remaining five were not synchronized. The resulting confidence interval for $\theta_1 - \theta_2$ under CRN was approximately the same length as, or only slightly shorter than, the confidence interval under independent sampling. Therefore, CRN is quite likely to fail in reducing the standard error of the estimated difference unless proper care is taken to guarantee synchronization of the random number streams on all replications.

Example 12.3

In Example 2.5, two policies for replacing bearings in a milling machine were compared. The bearing-life distribution, assumed discrete in Example 2.5 (Table 2.22), is now more realistically assumed to be continuous on the range from 950 to 1950 hours, with the first column of Table 2.22 giving the midpoint of 10 intervals of width 100 hours. The repairperson delay-time distribution of Table 2.23 is also assumed continuous, in the range from 2.5 to 17.5 minutes, with interval midpoints as given in the first column. The probabilities of each interval are given in the second columns of Tables 2.22 and 2.23.

The two models were run by using CRN and, for illustrative purposes, by using independent sampling, each for $R = 10$ replications. The purpose was to estimate the difference in mean total costs per 10,000 bearing hours, with the cost data given in Example 2.5. The estimated total cost for the two policies is given in Table 12.4.

Table 12.4 Total Costs for Alternative Designs of Bearing Replacement Problem

Replication r	Total Cost for Policy			Difference in Total Cost
	2	*II*	*IC*	$D_{IC,2}$
1	13,340	17,010	17,556	4,216
2	12,760	17,528	17,160	4,400
3	13,002	17,956	17,808	4,806
4	13,524	17,920	18,012	4,488
5	13,754	18,880	18,200	4,446
6	13,318	17,528	17,936	4,618
7	13,432	17,574	18,350	4,918
8	14,208	17,954	19,398	5,190
9	13,224	18,290	17,612	4,388
10	13,178	17,360	17,956	4,778
Sample mean	13,374	17,800		4,624
Sample variance	160,712	276,188		87,353

Policy 1 was to replace each bearing as it failed. Policy 2 was to replace all three bearings whenever one bearing failed. Policy 2 was run first, and then policy 1 was run, using independent sampling (column 1I), and using CRN (column 1C). The 95% confidence intervals for mean cost difference are as follows:

Independent sampling: \$4426 ± 439
CRN: \$4625 ± 211

(The computation of these confidence intervals is left as an exercise for the reader.)

Notice that the confidence interval for mean cost difference when using CRN is approximately 50% of the length of the confidence interval based on independent sampling. Therefore, for the same computer costs, (i.e., for $R = 10$ replications), CRN produces estimates that are twice as precise in this example. If CRN were used, the simulation analyst could conclude with 95% confidence that the mean cost difference between the two policies is between \$4414 and \$4836.

12.1.4 Confidence Intervals with Specified Precision

Section 11.4.2 described a procedure for obtaining confidence intervals with specified precision. Confidence intervals for the *difference* between two systems' performance can be obtained in an analogous manner.

Suppose that we want the error in our estimate of $\theta_1 - \theta_2$ to be less than $\pm\epsilon$ (the quantity ϵ might be a practically significant difference). Therefore, our goal is to find a number of replications R such that

$$H = t_{\alpha/2,\nu}\text{s.e.}(\bar{Y}_{.1} - \bar{Y}_{.2}) \le \epsilon \tag{12.19}$$

As in Section 11.4.2, we begin by making $R_0 \ge 2$ replications of each system to obtain an initial estimate of s.e.$(\bar{Y}_{.1} - \bar{Y}_{.2})$. We then solve for the total number of replications $R \ge R_0$ needed to achieve the half-length criterion (12.19). Finally, we make an additional $R - R_0$ replications (or a fresh R replications) of each system, compute the confidence interval, and check that the half-length criterion has been attained.

Example 12.1: Continued

Recall that $R_0 = 10$ replications and complete synchronization of the random numbers yielded the 95% confidence interval for the difference in expected response time of the two vehicle-inspection stations in Inequality (12.18); this interval can be rewritten as 0.4 ± 0.90 minutes. Although system 2 appears to have the smaller expected response time, the difference is not statistically significant, since the confidence interval contains 0. Suppose that a difference larger than ±0.5 minute is considered to be practically significant. We therefore want to make enough replications to obtain a $H \le \epsilon = 0.5$.

The confidence interval used in Example 12.1 was $\bar{D} \pm t_{\alpha/2,R_0-1} S_D / \sqrt{R_0}$, with the specific values $\bar{D} = 0.4$, $R_0 = 10$, $t_{0.025,9} = 2.26$ and $S_D^2 = 1.7$. To obtain the desired precision, we need to find R such that

$$\frac{t_{\alpha/2,R-1} S_D}{\sqrt{R}} \le \epsilon$$

Therefore, R is the smallest integer satisfying $R \ge R_0$ and

$$R \ge \left(\frac{t_{\alpha/2,R-1} S_D}{\epsilon} \right)^2$$

Since $t_{\alpha/2,R-1} \le t_{\alpha/2,R_0-1}$, a conservative estimate for R is given by

$$R \ge \left(\frac{t_{\alpha/2,R_0-1} S_D}{\epsilon} \right)^2$$

Substituting $t_{0.025,9} = 2.26$ and $S_D^2 = 1.7$, we obtain

$$R \ge \frac{(2.26)^2(1.7)}{(0.5)^2} = 34.73$$

implying that 35 replications are needed, 25 more than in the initial experiment.

12.2 COMPARISON OF SEVERAL SYSTEM DESIGNS

Suppose that a simulation analyst desires to compare K alternative system designs. The comparison will be made on the basis of some specified performance measure, θ_i, of system i, for $i = 1, 2, \ldots, K$. Many different statistical procedures have been developed that can be used to analyze simulation data and draw statistically sound inferences concerning the parameters θ_i. These procedures can be classified as being either fixed-sample-size procedures or sequential-sampling (or *multistage*) procedures. In the first type, a predetermined sample size (i.e., run length and number of replications) is used to draw inferences via hypothesis tests or confidence intervals. Examples of fixed-sample-size procedures include the interval estimation of a mean performance measure (Section 11.3) and the interval estimation of the difference between mean performance measures of two systems [as by Expression (12.1) in Section 12.1]. Advantages of fixed-sample-size procedures include a known or easily estimated cost in terms of computer time before running the experiments. When computer time is limited, or when a pilot study is being conducted, a fixed-sample-size procedure might be appropriate. In some cases, clearly inferior system designs may be ruled out at this early stage. A major disadvantage is that a strong conclusion could be impossible. For example, the confidence interval could be too wide for practical use, since the width is an indication of the precision of the point estimator. A hypothesis test may lead to a failure to reject the null hypothesis, a weak conclusion in general, meaning that there is no strong evidence one way or the other about the truth or falsity of the null hypothesis.

A sequential sampling scheme is one in which more and more data are collected until an estimator with a prespecified precision is achieved or until one of several alternative hypotheses is selected, with the probability of correct selection being larger than a prespecified value. A two-stage (or multistage) procedure is one in which an initial sample is used to estimate how many additional observations are needed to draw conclusions with a specified precision. An example of a two-stage procedure for estimating the performance measure of a single system was given in Section 11.4.2 and 12.1.4.

The proper procedure to use depends on the goal of the simulation analyst. Some possible goals are the following:

1. estimation of each parameter, θ_i;
2. comparison of each performance measure, θ_i, to a control, θ_1 (where θ_1 could represent the mean performance of an existing system);
3. all pairwise comparisons, $\theta_i - \theta_j$, for $i \ne j$;
4. selection of the best θ_i (largest or smallest).

The first three goals will be achieved by the construction of confidence intervals. The number of such confidence intervals is $C = K$, $C = K - 1$, and $C = K(K-1)/2$, respectively. Hochberg and Tamhane [1987] and Hsu [1996] are comprehensive references for such multiple-comparison procedures. The fourth goal requires the use of a type of statistical procedure known as a multiple ranking and selection procedure. Procedures to achieve these and other goals are discussed by Kleijnen [1975, Chapters II and V], who also discusses their relative merit and disadvantages. Goldsman and Nelson [1998] and Law and Kelton [2000]

discuss those selection procedures most relevant to simulation. A comprehensive reference is Bechhofer, Santner, and Goldsman [1995]. The next subsection presents a fixed-sample-size procedure that can be used to meet goals 1, 2, and 3 and is applicable in a wide range of circumstances. Subsections 12.2.2–12.2.3 present related procedures to achieve goal 4.

12.2.1 Bonferroni Approach to Multiple Comparisons

Suppose that C confidence intervals are computed and that the ith interval has confidence coefficient $1 - \alpha_i$. Let $\mathcal{S}_i$ be the statement that the ith confidence interval contains the parameter (or difference of two parameters) being estimated. This statement might be true or false for a given set of data, but the procedure leading to the interval is designed so that statement $\mathcal{S}_i$ will be true with probability $1 - \alpha_i$. When it is desired to make statements about several parameters simultaneously, as in goals 1, 2 and 3, the analyst would like to have high confidence that *all* statements are true simultaneously. The Bonferroni inequality states that

$$P(\text{all statements } \mathcal{S}_i \text{ are true}, i = 1, \ldots, C) \geq 1 - \sum_{j=1}^{C} \alpha_j = 1 - \alpha_E \qquad (12.20)$$

where $\alpha_E = \sum_{j=1}^{C} \alpha_j$ is called the overall error probability. Expression (12.20) can be restated as

$$P(\text{one or more statements } \mathcal{S}_i \text{ is false}, i = 1, \ldots C) \leq \alpha_E$$

or equivalently,

$$P(\text{one or more of the } C \text{ confidence intervals does not contain the parameter being estimated}) \leq \alpha_E$$

Thus, α_E provides an upper bound on the probability of a false conclusion. To conduct an experiment that involves making C comparisons, first select the overall error probability, say $\alpha_E = 0.05$ or 0.10. The individual α_j may be chosen to be equal ($\alpha_j = \alpha_E/C$), or unequal, as desired. The smaller the value of α_j, the wider the jth confidence interval will be. For example, if two 95% c.i.'s ($\alpha_1 = \alpha_2 = 0.05$) are constructed, the overall confidence level will be 90% or greater ($\alpha_E = \alpha_1 + \alpha_2 = 0.10$). If ten 95% c.i.'s are constructed (α_i 0.05, $i = 1, \ldots, 10$), the resulting overall confidence level could be as low as 50% ($\alpha_E = \sum_{i=1}^{10} \alpha_i = 0.50$), which is far too low for practical use. To guarantee an overall confidence level of 95%, when 10 comparisons are being made, one approach is to construct ten 99.5% confidence intervals for the parameters (or differences) of interest.

The Bonferroni approach to multiple confidence intervals is based on expression (12.20). A major advantage is that it holds whether the models for the alternative designs are run with independent sampling or with common random numbers.

The major disadvantage of the Bonferroni approach in making a large number of comparisons is the increased width of each individual interval. For example, for a given set of data and a large sample size, a 99.5% c.i. will be $z_{0.0025}/z_{0.025} = 2.807/1.96 = 1.43$ times longer than a 95% c.i. For small sample sizes—say, for a sample of size 5—a 99.5% c.i. will be $t_{0.0025,4}/t_{0.025,4} = 5.598/2.776 = 1.99$ times longer than an individual 95% c.i. The width of a c.i. is a measure of the precision of the estimate. For these reasons, it is recommended that the Bonferroni approach be used only when a small number of comparisons are being made. Twenty or so comparisons appears to be the practical upper limit.

Corresponding to goals 1, 2, and 3, there are at least three possible ways of using the Bonferroni Inequality (12.20) when comparing K alternative system designs:

1. (*Individual c.i.'s*): Construct a $100(1 - \alpha_i)\%$ c.i. for parameter θ_i by using Expression (11.12), in which case the number of intervals is $C = K$. If independent sampling were used, the K c.i.'s would be mutually independent, and thus the overall confidence level would be $(1 - \alpha_1) \times (1 - \alpha_2) \times \cdots \times (1 - \alpha_C)$, which is larger (but not much larger) than the right side of Expression (12.20). This type of procedure is most often used to estimate multiple parameters of a single system, rather than to compare systems—and, because multiple parameter estimates from the same system are likely to be dependent, the Bonferroni inequality typically is needed.
2. (*Comparison to an existing system*): Compare all designs to one specific design—usually, to an existing system: that is, construct a $100(1 - \alpha_i)\%$ c.i. for $\theta_i - \theta_1 (i = 2, 3, \ldots, K)$, using Expression (12.1). (System 1 with performance measure θ_1 is assumed to be the existing system). In this case, the number of intervals is $C = K - 1$. This type of procedure is most often used to compare several competitors to the present system in order to learn which are better.
3. (*All pairwise comparisons*): Compare all designs to each other—that is, for any two system designs $i \neq j$, construct a $100(1 - \alpha_{ij})\%$ c.i. for $\theta_i - \theta_j$. With K designs, the number of confidence intervals computed is $C = K(K - 1)/2$. The overall confidence coefficient would be bounded below by $1 - \alpha_E = 1 - \sum\sum_{i \neq j} \alpha_{ij}$ (which follows by Expression (12.20)). It is generally believed that CRN will make the true overall confidence level larger than the right side of Expression (12.20), and usually larger than will independent sampling. The right side of Expression (12.20) can be thought of as giving the worst case (i.e., the lowest possible overall confidence level).

Example 12.4

Reconsider the vehicle-inspection station of Example 12.1. Suppose that the construction of additional space to hold one waiting car is being considered. The alternative system designs are the following:

1. existing system (parallel stations);
2. no space between stations in series;
3. one space between brake and headlight inspection only;
4. one space between headlight and steering inspection only.

Design 2 was compared to the existing setup in Example 12.1. Designs 2, 3, and 4 are series queues, as shown in Figure 12.1(b), the only difference being the number or location of a waiting space between two successive inspections. The arrival process and the inspection times are as given in Example 12.1. The basis for comparison will be mean response time, θ_i, for system i, where a response time is the total time it takes for a car to get through the system. Confidence intervals for $\theta_2 - \theta_1$, $\theta_3 - \theta_1$, and $\theta_4 - \theta_1$ will be constructed, each having an overall confidence level of 95%. The run length T_E has now been set at 40 hours (instead of the 16 hours used in Example 12.1), and the number of replications R of each model is 10. Common random numbers will be used in all models, but this does not affect the overall confidence level, because, as mentioned, the Bonferroni Inequality (12.20) holds regardless of the statistical independence or dependence of the data.

Since the overall error probability is $\alpha_E = 0.05$ and $C = 3$ confidence intervals are to be constructed, let $\alpha_i = 0.05/3 = 0.0167$ for $i = 2,3,4$. Then use Expression (12.1) (with proper modifications) to construct $C = 3$ confidence intervals with $\alpha = \alpha_i = 0.0167$ and degrees of freedom $\nu = 10 - 1 = 9$. The standard error is computed by Equation (12.14), because common random numbers are being used. The output data Y_{ri} are displayed in Table 12.5; Y_{ri} is the sample mean response time for replication r on system i ($r = 1, \ldots, 10$; $i = 1, 2, 3, 4$). The differences $D_{ri} = Y_{r1} - Y_{ri}$ are also shown, together with the sample mean differences, $\bar{D}_i$, averaged over all replications as in Equation (12.12), the sample variances $S_{D_i}^2$, and the standard error. By Expression (12.1), the three confidence intervals, with overall confidence coefficient at least $1 - \alpha_E$, are given by

$$\bar{D}_i - t_{\alpha_i/2, R-1}\, s.e.(\bar{D}_i) \leq \theta_1 - \theta_i \leq \bar{D}_i + t_{\alpha_i/2, R-1}\, s.e.(\bar{D}_i), \quad i = 2, 3, 4$$

Table 12.5 Analysis of Output Data for Vehicle Inspection System When Using CRN

	Average Response Time for System Design				*Observed Difference with System Design 1*		
Replication,	*1,*	*2,*	*3,*	*4,*			
r	Y_{r1}	Y_{r2}	Y_{r3}	Y_{r4}	D_{r2}	D_{r3}	D_{r4}
1	63.72	63.06	57.74	62.63	0.66	5.98	1.09
2	32.24	31.78	29.65	31.56	0.46	2.59	0.68
3	40.28	40.32	36.52	39.87	−0.04	3.76	0.41
4	36.94	37.71	35.71	37.35	−0.77	1.23	−0.41
5	36.29	36.79	33.81	36.65	−0.50	2.48	−0.36
6	56.94	57.93	51.54	57.15	−0.99	5.40	−0.21
7	34.10	33.39	31.39	33.30	0.71	2.71	0.80
8	63.36	62.92	57.24	62.21	0.44	6.12	1.15
9	49.29	47.67	42.63	47.46	1.62	6.66	1.83
10	87.20	80.79	67.27	79.60	6.41	19.93	7.60
Sample mean, $\bar{D}_i$					0.80	5.686	1.258
Sample standard deviation, S_{D_i}					2.12	5.338	2.340
Sample variance, $S^2_{D_i}$					4.498	28.498	5.489
Standard error, $S_{D_i}/\sqrt{R}$					0.671	1.688	0.741

The value of $t_{\alpha_i/2,R-1} = t_{0.0083,9} = 2.97$ is obtained from Table A.5 by interpolation. For these data, with 95% confidence, it is stated that

$$-1.19 \le \theta_1 - \theta_2 \le 2.79$$
$$0.67 \le \theta_1 - \theta_3 \le 10.71$$
$$-0.94 \le \theta_1 - \theta_4 \le 3.46$$

The simulation analyst has high confidence (at least 95%) that all three confidence statements are correct. Notice that the c.i. for $\theta_1 - \theta_2$ again contains zero; thus, there is no statistically significant difference between design 1 and design 2, a conclusion that supports the previous results in Example 12.1. The c.i. for $\theta_1 - \theta_3$ lies completely above zero and so provides strong evidence that $\theta_1 - \theta_3 > 0$—that is, that design 3 is better than design 1 because its mean response time is smaller. The c.i. for $\theta_1 - \theta_4$ contains zero, so there is no statistically significant difference between designs 1 and 4.

If the simulation analyst now decides that it would be desirable to compare designs 3 and 4, more simulation runs would be necessary, because it is not formally correct to decide which confidence intervals to compute after the data have been examined. On the other hand, if the simulation analyst had decided to compute all possible confidence intervals (and had made this decision before collecting the data, Y_{ri}), the number of confidence intervals would have been $C = 6$ and the three c.i.'s would have been $t_{0.0042,9}/t_{0.0083,9} \approx 3.32/2.97 = 1.12$ times (or 12%) longer. There is always a trade-off between the number of intervals (C) and the width of each interval. The simulation analyst should carefully consider the possible conclusions before running the simulation experiments and choose those runs and analyses that will provide the most useful information. In particular, the number of confidence intervals computed should be as small as possible—preferably, 20 or less.

For purposes of illustration, 10 replications of each of the four designs were run, using independent sampling (i.e., different random numbers for all runs). The results are presented in Table 12.6, together with sample means ($\bar{Y}_i$), sample standard deviations (S_i), and sample variances (S_i^2), plus the observed difference of sample means ($\bar{Y}_1 - \bar{Y}_i$) and the standard error (s.e.) of the observed difference. It is observed that all three confidence intervals for $\theta_1 - \theta_i (i = 2, 3, 4)$ contain zero. Therefore, no strong conclusion is possible from these data and this sample size. By contrast, a sample size of ten was sufficient, when using CRN, to provide strong evidence that design 3 is superior to design 1.

Notice the large increase in standard error of the estimated difference with independent sampling versus with common random numbers. These standard errors are compared in Table 12.7. In addition, a careful examination of Tables 12.5 and 12.6 illustrates the superiority of CRN. In Table 12.5, in all 10 replications, system design 3 has a smaller average response time than does system design 1. By comparing replications 1 and 2 in Table 12.5, it can be seen that a random-number stream that leads to high congestion and large response times in system design 1, as in the first replication, produces results of similar magnitude across all four system designs. Similarly, when system design 1 exhibits relatively low congestion and low response times, as in the second replication, all system designs produce relatively low average response times. This similarity of results on each replication is due, of course, to the use of common random numbers across systems. By contrast, for independent sampling, Table 12.6 shows no such similarity across system designs. In only 5 of the 10 replications is the average response time for system design 3 smaller than that for system design 1, although the average difference in response times across all 10 replications is approximately the same magnitude in each case: 5.69 minutes when using CRN, and 5.89 minutes when using independent

Table 12.6 Analysis of Output Data for the Vehicle-Inspection System that Uses Independent Sampling

	Average Response Time for System Design			
Replication, r	*1,* Y_{r1}	*2,* Y_{r2}	*3,* Y_{r3}	*4,* Y_{r4}
1	63.72	59.37	52.00	59.03
2	32.24	50.06	47.04	49.97
3	40.28	60.63	53.21	60.18
4	36.94	46.36	40.88	45.44
5	36.29	68.87	50.85	66.65
6	56.94	66.44	60.42	66.03
7	34.10	27.51	26.70	27.45
8	63.36	47.98	40.12	47.50
9	49.29	29.92	28.59	29.84
10	87.20	47.14	41.62	46.44
Sample mean $\bar{Y}_i$	50.04	50.43	44.14	49.85
S_i	17.70	13.98	10.76	13.64
S_i^2	313.38	195.54	115.74	185.98
$\bar{Y}_1 - \bar{Y}_i$		−0.39	5.89	0.18
s.e.$(\bar{Y}_1 - \bar{Y}_i)$		7.13	6.55	7.07

Table 12.7 Comparison of Standard Errors Arising from CRN with those from Independent Sampling, for the Vehicle-Inspection Problem

Difference in Sample Means	*Standard Error When Using*		*Percentage Increase*
	CRN Sampling	*Independent Sampling*	
$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 2}$	0.67	7.13	1064%
$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 3}$	1.69	6.55	388%
$\bar{Y}_{\cdot 1} - \bar{Y}_{\cdot 4}$	0.74	7.07	955%

sampling. The greater variability of independent sampling is reflected also in the standard errors of the point estimates: ±1.69 minutes for CRN versus ± 6.55 minutes for independent sampling, an increase of 388%, as seen in Table 12.7. This example illustrates again the advantage of CRN.

As stated previously, CRN does not yield a variance reduction in all simulation models. It is recommended that a pilot study be undertaken and variances estimated to confirm (or possibly deny) the assumption that CRN will reduce the variance (or standard error) of an estimated difference. The reader is referred to the discussion in Section 12.1.3.

Some of the exercises at the end of this chapter provide an opportunity to compare CRN and independent sampling and to compute simultaneous confidence intervals under the Bonferroni approach.

12.2.2 Bonferroni Approach to Selecting the Best

Suppose that there are K system designs, and the ith design has expected performance θ_i. At a gross level, we are interested in which system is best, where "best" is defined to be having maximum expected performance.[1] At a more refined level, we could also be interested in how much better the best is relative to each alternative, because secondary criteria that are not reflected in the performance measure θ_i (such as ease of installation, cost to maintain, etc.) could tempt us to choose an inferior system if it is not deficient by much.

If system design i is the best, then $\theta_i - \max_{j \neq i} \theta_j$ is equal to the difference in performance between the best and the second best. If system design i is not the best, then $\theta_i - \max_{j \neq i} \theta_j$ is equal to the difference between system i and the best. The selection procedure we describe in this section focuses on the parameters $\theta_i - \max_{j \neq i} \theta_j$ for $i = 1, 2, \ldots, K$.

Let i^* denote the (unknown) index of the best system. As a general rule, the smaller the true difference $\theta_{i^*} - \max_{j \neq i^*} \theta_j$ is, and the more certain we want to be that we find the best system, the more replications are required to achieve our goal. Therefore, instead of demanding that we find i^*, we can compromise and ask to find i^* with high probability whenever the difference between system i^* and the others is at least some practically significant amount. More precisely, we want the probability that we select the best system to be at least $1 - \alpha$ whenever $\theta_{i^*} - \max_{j \neq i^*} \theta_j \geq \epsilon$. If there are one or more systems that are within ϵ of the best, then we will be satisfied to select either the best or any one of the near best. Both the probability of correct selection, $1 - \alpha$, and the practically significant difference, ϵ, will be under our control.

The following procedure achieves the desired probability of correct selection (Nelson and Matejcik [1995]). And because we are also interested in how much each system differs from the best, it also forms $100(1-\alpha)\%$ confidence intervals for $\theta_{i^*} - \max_{j \neq i^*} \theta_j$ for $i = 1, 2, \ldots, K$. The procedure is valid for normally distributed data when either CRN or independent sampling is being used.

[1]If "best" is defined to be having minimum expected performance, then the procedure in this section is easily modified, as we illustrate in the example.

Two-Stage Bonferroni Procedure

1. Specify the practically significant difference ϵ, the probability of correct selection $1 - \alpha$, and the first-stage sample size $R_0 \geq 10$. Let $t = t_{\alpha/(K-1), R_0 - 1}$.
2. Make R_0 replications of system i to obtain $Y_{1i}, Y_{2i}, \ldots, Y_{R_0 i}$, for systems $i = 1, 2, \ldots, K$.
3. Calculate the first-stage sample means $\bar{Y}_{\cdot i}$, $i = 1, 2, \ldots, K$. For all $i \neq j$, calculate the sample variance of the difference,[2]

$$S_{ij}^2 = \frac{1}{R_0 - 1} \sum_{r=1}^{R_0} \left(Y_{ri} - Y_{rj} - (\bar{Y}_{\cdot i} - \bar{Y}_{\cdot j}) \right)^2$$

   Let $\hat{S}^2 = \max_{i \neq j} S_{ij}^2$, the largest sample variance.
4. Calculate the second-stage sample size,

$$R = \max \left\{ R_0, \left\lceil \frac{t^2 \hat{S}^2}{\epsilon^2} \right\rceil \right\}$$

   where $\lceil \cdot \rceil$ means to round up.
5. Make $R - R_0$ additional replications of system i to obtain the output data $Y_{R_0+1,i}, Y_{R_0+2,i}$, for $i = 1, 2, \ldots, K$.[3]
6. Calculate the overall sample means

$$\bar{\bar{Y}}_i = \frac{1}{R} \sum_{r=1}^{R} Y_{ri}$$

   for $i = 1, 2, \ldots, K$.
7. Select the system with largest $\bar{\bar{Y}}_i$ as the best.

   Also form the confidence intervals

$$\min\{0, \bar{\bar{Y}}_i - \max_{j \neq i} \bar{\bar{Y}}_j - \epsilon\} \leq \theta_i - \max_{j \neq i} \theta_j \leq \max\{0, \bar{\bar{Y}}_i - \max_{j \neq i} \bar{\bar{Y}}_j + \epsilon\}$$

   for $i = 1, 2, \ldots, K$.

The confidence intervals in Step 7 are not like the usual ± intervals presented elsewhere in this chapter. Perhaps the most useful interpretation of them is as follows. Let $\hat{I}$ be the index of the system selected as best. Then, for each of the other systems i, we make one of the declarations:

- If $\bar{\bar{Y}}_i - \bar{\bar{Y}}_{\hat{I}} + \epsilon \leq 0$, then declare system i to be inferior to the best.
- If $\bar{\bar{Y}}_i - \bar{\bar{Y}}_{\hat{I}} + \epsilon > 0$, then declare system i to be statistically indistinguishable from the best (and, therefore, system i might be the best).

Example 12.4: Continued

Recall that, in Example 12.4, we considered $K = 4$ different designs for the vehicle-inspection station. Suppose that we would like 0.95% confidence of selecting the best (smallest expected response time) system design when

[2]Notice that S_{ij}^2 is algebraically equivalent to S_D^2, the sample variance of $D_r = Y_{ri} - Y_{rj}$, for $r = 1, 2, \ldots, R_0$.

[3]If it is more convenient, a total of R replications can be generated from system i by restarting the entire experiment.

the best differs from the second best by at least two minutes. This is a minimization problem, so we focus on the differences $\theta_i - \min_{j \neq i} \theta_j$ for $i = 1, 2, 3, 4$. Then we can apply the Two-Stage Bonferroni Procedure as follows:

1. $\epsilon = 2$ minutes, $1 - \alpha = 0.95$, $R_0 = 10$, and $t = t_{0.0167,9} = 2.508$.
2. The data in Table 12.5, which was obtained by using CRN, is employed.
3. From Table 12.5, we get $S_{12}^2 = S_{D_2}^2 = 4.498$, $S_{13}^2 = S_{D_3}^2 = 28.498$, and $S_{14}^2 = S_{D_4}^2 = 5.489$. By similar calcultions, we obtain $S_{23}^2 = 11.857$, $S_{24}^2 = 0.119$, and $S_{34}^2 = 9.849$.
4. Since $\hat{S}^2 = S_{13}^2 = 28.498$ is the largest sample variance,

$$R = \max\left\{10, \left\lceil \frac{(2.508)^2(28.498)}{2^2} \right\rceil\right\} = \max\{10, \lceil 44.8 \rceil\} = 45$$

5. Make $45 - 10 = 35$ additional replications of each system.
6. Calculate the overall sample means

$$\bar{\bar{Y}}_i = \frac{1}{45}\sum_{r=1}^{45} Y_{ir}$$

for $i = 1, 2, 3, 4$.

7. Select the system with smallest $\bar{\bar{Y}}_i$ as the best.
   Also, form the confidence intervals

$$\min\{0, \bar{\bar{Y}}_i - \min_{j \neq i} \bar{\bar{Y}}_j - 2\} \leq \theta_i - \min_{j \neq i} \theta_j \leq \max\{0, \bar{\bar{Y}}_i - \min_{j \neq i} \bar{\bar{Y}}_j + 2\}$$

for $i = 1, 2, 3, 4$.

12.2.3 Bonferroni Approach to Screening

When a two-stage procedure is not possible, or when there are many systems, it could be useful to divide the set of systems into those that could be the best and those that can be eliminated from further consideration. For this purpose, a screening or *subset selection* procedure is useful. The following procedure, due to Nelson *et al.* [2001], guarantees that the retained subset contains the true best system with probability $\geq 1 - \alpha$ when the data are normally distributed and either independent sampling or CRN is used. The subset may contain all K of the systems, only one system, or some number in between, depending on the number of replications and the sample means and sample variances.

Screening Procedure

1. Specify the probability of correct selection $1 - \alpha$ and common sample size from each system, $R \geq 2$. Let $t = t_{\alpha/(K-1),R-1}$.
2. Make R replications of system i to obtain $Y_{1i}, Y_{2i}, \ldots, Y_{Ri}$ for systems $i = 1, 2, \ldots, K$.
3. Calculate the sample means $\bar{Y}_{\cdot i}$ for $i = 1, 2, \ldots, K$. For all $i \neq j$, calculate the sample variance of the difference,

$$S_{ij}^2 = \frac{1}{R-1}\sum_{r=1}^{R}(Y_{ri} - Y_{rj} - (\bar{Y}_{\cdot i} - \bar{Y}_{\cdot j}))^2$$

4. If bigger is better, then retain system i in the selected subset if

$$\bar{Y}_{\cdot i} \geq \bar{Y}_{\cdot j} - t\frac{S_{ij}}{\sqrt{R}} \text{ for all } j \neq i$$

If smaller is better, then retain system i in the selected subset if

$$\bar{Y}_{\cdot i} \leq \bar{Y}_{\cdot j} + t\frac{S_{ij}}{\sqrt{R}} \text{ for all } j \neq i$$

All system designs that are not retained can be eliminated from further consideration.

Example 12.4: Continued

Suppose we want to see whether any of the designs for the vehicle-inspection station can be eliminated on the basis of only the 10 replications in Table 12.5. Summaries of the sample means and variances of the differences are as follows:

Y_i	1	2	3	4
	50.04	49.24	44.35	48.78

S_{ij}^2	1	2	3	4
1		4.498	28.498	5.489
2			11.857	0.119
3				9.84

The appropriate critical value to obtain 95% confidence that the selected subset contains the true best is $t = t_{0.0167,9} = 2.508$. Recall that smaller response time is better. Applying the Subset Selection Procedure, system designs 1, 2, and 4 can all be eliminated, because

$$\bar{Y}_{\cdot 1} = 50.04 \not\leq \bar{Y}_{\cdot 3} + t\sqrt{\frac{S_{13}^2}{R}} = 44.35 + 2.508\sqrt{\frac{28.498}{10}} = 48.58$$

$$\bar{Y}_{\cdot 2} = 49.24 \not\leq \bar{Y}_{\cdot 3} + t\sqrt{\frac{S_{23}^2}{R}} = 44.35 + 2.508\sqrt{\frac{11.857}{10}} = 47.08$$

$$\bar{Y}_{\cdot 4} = 48.78 \not\leq \bar{Y}_{\cdot 3} + t\sqrt{\frac{S_{43}^2}{R}} = 44.35 + 2.508\sqrt{\frac{9.84}{10}} = 46.84$$

Thus, in this case there was adequate data to select the best, system design 3, with 95% confidence. Had more than one system survived the subset selection, then we could perform additional analysis on that subset, perhaps using the Two-Stage Bonferroni Procedure.

12.3 METAMODELING

Suppose that there is a simulation output response variable, Y, that is related to k independent variables, say $x_1, x_2, \ldots, x_k$. The dependent variable, Y, is a random variable, while the independent variables $x_1, x_2, \ldots, x_k$ are called design variables and are usually subject to control. The true relationship between the variables Y and x is represented by the (often complex) simulation model. Our goal is to approximate this relationship by a simpler mathematical function called a metamodel. In some cases, the analyst will know the exact form of the functional relationship between Y and $x_1, x_2, \ldots, x_k$, say $Y = f(x_1, x_2, \ldots, x_k)$. However, in most cases, the functional relationship is unknown, and the analyst must select an appropriate f containing unknown parameters, and then estimate those parameters from a set of data (Y, x). Regression analysis is one method for estimating the parameters.

Example 12.5

An insurance company promises to process all claims it receives each day by the end of the next day. It has developed a simulation model of its proposed claims-processing system to evaluate how hard it will be to meet this promise. The actual number and types of claims that will need to be processed each day will vary, and the number may grow over time. Therefore, the company would like to have a model that predicts the total processing time as a function of the number of claims received.

The primary value of a metamodel is to make it easy to answer "what if" questions, such as, what the processing time will be if there are x claims. Evaluating a function f, or perhaps its derivatives, at a number of values of x is typically much easier than running a simulation experiment for each value.

12.3.1 Simple Linear Regression

Suppose that it is desired to estimate the relationship between a single independent variable x and a dependent variable Y, and suppose that the true relationship between Y and x is suspected to be linear. Mathematically, the expected value of Y for a given value of x is assumed to be

$$E(Y \mid x) = \beta_0 + \beta_1 x \tag{12.21}$$

where β_0 is the intercept on the Y axis, an unknown constant; and β_1 is the slope, or change in Y for a unit change in x, also an unknown constant. It is further assumed that each observation of Y can be described by the model

$$Y = \beta_0 + \beta_1 x + \epsilon \tag{12.22}$$

where ϵ is a random error with mean zero and constant variance σ^2. The regression model given by Equation (12.22) involves a single variable x and is commonly called a simple linear regression model.

Suppose that there are n pairs of observations $(Y_1, x_1), (Y_2, x_2), \ldots, (Y_n, x_n)$. These observations can be used to estimate β_0 and β_1 in Equation (12.22). The method of least squares is commonly used to form the estimates. In the method of least squares, β_0 and β_1 are estimated in such a way that the sum of the squares of the deviations between the observations and the regression line is minimized. The individual observations in Equation (12.22) can be written as

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \; i = 1, 2, \ldots, n \tag{12.23}$$

where $\epsilon_1, \epsilon_2 \ldots$ are assumed to be uncorrelated random variables.

Each ϵ_i in Equation (12.23) is given by

$$\epsilon_i = Y_i - \beta_0 - \beta_1 x_i \tag{12.24}$$

and represents the difference between the observed response, Y_i, and the expected response, $\beta_0 + \beta_1 x_i$, predicted by the model in Equation (12.21). Figure 12.3 shows how ϵ_i is related to x_i, Y_i, and $E(Y_i|x_i)$.

The sum of squares of the deviations given in Equation (12.24) is given by

$$L = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 x_i)^2 \tag{12.25}$$

and L is called the least-squares function. It is convenient to rewrite Y_i as follows:

$$Y_i = \beta_0' + \beta_1 (x_i - \bar{x}) + \epsilon_i \tag{12.26}$$

where $\beta_0' = \beta_0 + \beta_1 \bar{x}$ and $\bar{x} = \sum_{i=1}^{n} x_i / n$. Equation (12.26) is often called the transformed linear regression model. Using Equation (12.26), Equation (12.25) becomes

$$L = \sum_{i=1}^{n} [Y_i - \beta_0' - \beta_1 (x_i - \bar{x})]^2$$

To minimize L, find $\partial L / \partial \beta_0'$ and $\partial L / \partial \beta_1$, set each to zero, and solve for $\hat{\beta}_0'$ and $\hat{\beta}_1$. Taking the partial derivatives and setting each to zero yields

$$n\hat{\beta}_0' = \sum_{i=1}^{n} Y_i$$

$$\hat{\beta}_1 \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} Y_i (x_i - \bar{x}) \tag{12.27}$$

Equations (12.27) are often called the "normal equations," which have the solutions

$$\hat{\beta}_0' = \bar{Y} = \sum_{i=1}^{n} \frac{Y_i}{n} \tag{12.28}$$

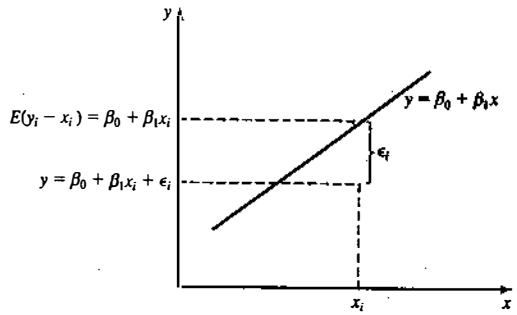


Figure 12.3 Relationship of ϵ_i to x_i, Y_i, and $E(Y_i|x_i)$.

and

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} Y_i(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{12.29}$$

The numerator in Equation (12.29) is rewritten for computational purposes as

$$S_{xy} = \sum_{i=1}^{n} Y_i(x_i - \bar{x}) = \sum_{i=1}^{n} x_i Y_i - \frac{\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} Y_i\right)}{n} \tag{12.30}$$

where S_{xy} denotes the corrected sum of cross products of x and Y. The denominator of Equation (12.29) is rewritten for computational purposes as

$$S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n} \tag{12.31}$$

where S_{xx} denotes the corrected sum of squares of x. The value of $\hat{\beta}_0$ can be retrieved easily as

$$\hat{\beta}_0 = \hat{\beta}_0' - \hat{\beta}_1 \bar{x} \tag{12.32}$$

Example 12.6: Calculating $\hat{\beta}_0$ and $\hat{\beta}_1$

The simulation model of the claims-processing system in Example 12.5 was executed with initial conditions $x = 100$, 150, 200, 250, and 300 claims received the previous day. Three replications were obtained at each setting. The response Y is the number of hours required to process x claims. The results are shown in Table 12.8. The graphical relationship between the number of claims received and total processing time is shown in

Table 12.8 Simulation Results for Processing Time Given x Claims

Number of Claims x	*Hours of Processing Time Y*
100	8.1
100	7.8
100	7.0
150	9.6
150	8.5
150	9.0
200	10.9
200	13.3
200	11.6
250	12.7
250	14.5
250	14.7
300	16.5
300	17.5
300	16.3

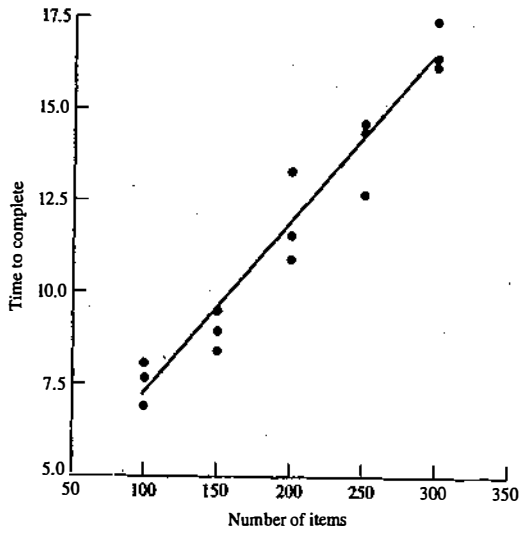


Figure 12.4 Relationship between number of claims and hours of processing time.

Figure 12.4. Such a display is called a scatter diagram. Examination of this scatter diagram indicates that there is a strong relationship between number of claims and processing time. The tentative assumption of the linear model given by Equation (12.22) appears to be reasonable.

With the processing times as the Y_i values (the dependent variables) and the number of claims as the x_i values (the independent variables), $\hat{\beta}_0$ and $\hat{\beta}_1$ can be found by the following computations: $n = 15$, $\sum_{i=1}^{15} x_i = 3000$, $\sum_{i=1}^{15} Y_i = 178$, $\sum_{i=1}^{15} x_i^2 = 675{,}000$, $\sum_{i=1}^{15} x_i Y_i = 39080$, and $\bar{x} = 3000/15 = 200$.

From Equation (12.30) S_{xy} is calculated as

$$S_{xy} = 39{,}080 - \frac{(3000)(178)}{15} = 3480$$

From Equation (12.31), S_{xx} is calculated as

$$S_{xx} = 675{,}000 - \frac{(3000)^2}{15} = 75{,}000$$

Then, $\hat{\beta}_1$ is calculated from Equation (12.29) as

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{3480}{75{,}000} = 0.0464$$

As shown in Equation (12.28), $\hat{\beta}_0'$ is just $\bar{Y}$, or

$$\hat{\beta}_0' = \frac{178}{15} \approx 11.8667$$

To express the model in the original terms, compute $\hat{\beta}_0$ from Equation (12.32) as

$$\hat{\beta}_0 = 11.8667 - 0.0464(200) = 2.5867$$

Then an estimate of the mean of Y given x, $E(Y|x)$, is given by

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x = 2.5867 + 0.0464x \tag{12.33}$$

For a given number of claims, x, this model can be used to predict the number of hours required to process them. The coefficient $\hat{\beta}_1$ has the interpretation that each additional claim received adds an expected 0.0464 hours, or 2.8 minutes, to the expected total processing time.

Regression analysis is widely used and frequently misused. Several of the common abuses are briefly mentioned here. Relationships derived in the manner of Equation (12.33) are valid for values of the independent variable within the range of the original data. The linear relationship that has been tentatively assumed may not be valid outside the original range. In fact, we know from queueing theory that mean processing time may increase rapidly as the number of claims approaches the capacity of the system. Therefore, Equation (12.33) can be considered valid only for $100 \le x \le 300$. Regression models are not advised for extrapolation purposes.

Care should be taken in selecting variables that have a plausible causal relationship with each other. It is quite possible to develop statistical relationships that are unrelated in a practical sense. For example, an attempt might be made to relate monthly output of a steel mill to the weight of reports appearing on a manager's desk during the month. A straight line may appear to provide a good model for the data, but the relationship between the two variables is tenuous. A strong observed relationship does not imply that a causal relationship exists between the variables. Causality can be inferred only when analysis uncovers some plausible reasons for its existence. In Example 12.5 it is reasonable that starting with more claims implies that more time is needed to process them. Therefore, a relationship of the form of Equation (12.33) is at least plausible.

12.3.2 Testing for Significance of Regression

In Section 12.3.1, it was assumed that a linear relationship existed between Y and x. In Example 12.5, a scatter diagram, shown in Figure 12.4, relating number of claims and processing time was prepared to evaluate whether a linear model was a reasonable tentative assumption prior to the calculation of $\hat{\beta}_0$ and $\hat{\beta}_1$. However, the adequacy of the simple linear relationship should be tested prior to using the model for predicting the response, Y_i, given an independent variable, x_i. There are several tests which may be conducted to aid in determining model adequacy. Testing whether the order of the model tentatively assumed is correct, commonly called the "lack-of-fit test," is suggested. The procedure is explained by Box and Draper [1987], Hines, Montgomery, Goldsman, and Borror [2002], and Montgomery [2000].

Testing for the significance of regression provides another means for assessing the adequacy of the model. The hypothesis test described next requires the additional assumption that the error component ϵ_i is normally distributed. Thus, the complete assumptions are that the errors are NID(0, σ^2)—that is, normally and independently distributed with mean zero and constant variance σ^2. The adequacy of the assumptions can and should be checked by residual analysis, discussed by Box and Draper [1987], Hines, Montgomery, Goldsman, and Borror [2002], and Montgomery [2000].

Testing for significance of regression is one of many hypothesis tests that can be developed from the variance properties of $\hat{\beta}_0$ and $\hat{\beta}_1$. The interested reader is referred to the references just cited for extensive discussion of hypothesis testing in regression. Just the highlights of testing for significance of regression are given in this section.

Suppose that the alternative hypotheses are

$H_0 : \beta_1 = 0$
$H_1 : \beta_1 \neq 0$

Failure to reject H_0 indicates that there is no linear relationship between x and Y. This situation is illnstrated in Figure 12.5. Notice that two possibilities exist. In Figure 12.5(a), the implication is that x is of little value in explaining the variability in Y, and that $\hat{y} = \bar{Y}$ is the best estimator. In Figure 12.5(b), the implication is that the true relationship is not linear.

Alternatively, if H_0 is rejected, the implication is that x is of value in explaining the variability in Y. This situation is illustrated in Figure 12.6. Here, also, two possibilities exist. In Figure 12.6(a), the straight-line model is adequate. However, in Figure 12.6(b), even though there is a linear effect of x, a model with higher-order terms (such as x^2, x^3, ...) is necessary. Thus, even though there may be significance of regression, testing of the residuals and testing for lack of fit are needed to confirm the adequacy of the model.

The appropriate test statistic for significance of regression is given by

$$t_0 = \frac{\hat{\beta}_1}{\sqrt{MS_E / S_{xx}}} \tag{12.34}$$

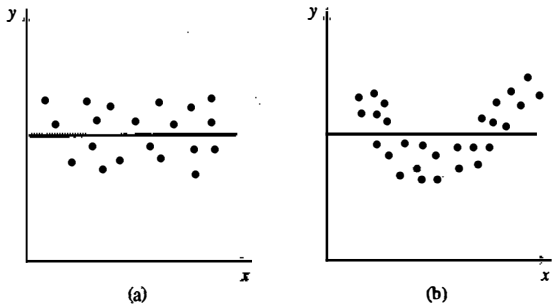


Figure 12.5 Failure to reject $H_0 : \beta_1 = 0$.

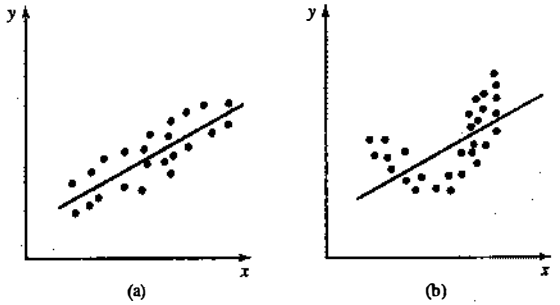


Figure 12.6 $H_0 : \beta_1 = 0$ is rejected.

where MS_E is the mean squared error. The error is the difference between the observed value, Y_i, and the predicted value, $\hat{y}_i$, at x_i, or $e_i = Y_i - \hat{y}_i$. The squared error is given by $\sum_{i=1}^{n} e_i^2$, and the mean squared error, given by

$$MS_E = \sum_{i=1}^{n} \frac{e_i^2}{n-2} \tag{12.35}$$

is an unbiased estimator of $\sigma^2 = V(\epsilon_i)$. The direct method can be used to calculate $\sum_{i=1}^{n} e_i^2$: Calculate each $\hat{y}_i$, compute e_i^2, and sum all the e_i^2 values, $i = 1, 2, \ldots, n$. However, it can be shown that

$$\sum_{i=1}^{n} e_i^2 = S_{yy} - \hat{\beta}_1 S_{xy} \tag{12.36}$$

where S_{yy}, the corrected sum of squares of Y, is given by

$$S_{yy} = \sum_{i=1}^{n} Y_i^2 - \frac{\left(\sum_{i=1}^{n} Y_i\right)^2}{n} \tag{12.37}$$

and S_{xy} is given by Equation (12.30). Equation (12.36) could be easier to use than the direct method.

The statistic defined by Equation (12.34) has the t distribution with $n - 2$ degrees of freedom. The null hypothesis H_0 is rejected if $|t_0| > t_{\alpha/2,n-2}$.

Example 12.7: Testing for Significance of Regression

Given the results in Example 12.6, the test for the significance of regression is conducted. One more computation is needed prior to conducting the test. That is, $\sum_{i=1}^{n} Y_i^2 = 2282.94$. Using Equation (12.37) yields

$$S_{yy} = 2282.94 - \frac{(178)^2}{15} = 170.6734$$

Then $\sum_{i=1}^{15} e_i^2$ is computed according to Equation (12.36) as

$$\sum_{i=1}^{15} e_i^2 = 170.6734 - 0.0464(3480) = 9.2014$$

Now, the value of MS_E is calculated from Equation (12.35):

$$MS_E = \frac{9.2014}{13} = 0.7078$$

The value of t_0 can be calculated by using Equation (12.34) as

$$t_0 = \frac{0.0464}{\sqrt{0.7078/75000}} = 15.13$$

Since $t_{0.025,13} = 2.16$ from Table A.5, we reject the hypothesis that $\beta_1 = 0$. Thus, there is significant evidence that x and Y are related.

12.3.3 Multiple Linear Regression

If the simple linear regression model of Section 12.3.1 is inadequate, several other possibilities exist. There could be several independent variables, so that the relationship is of the form

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m + \epsilon \tag{12.38}$$

Notice that this model is still linear, but has more than one independent variable. Regression models having the form shown in Equation (12.38) are called multiple linear regression models. Another possibility is that the model is of a quadratic form such as

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon \tag{12.39}$$

Equation (12.39) is also a linear model which may be transformed to the form of Equation (12.38) by letting $x_1 = x$ and $x_2 = x^2$.

Yet another possibility is a model of the form such as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon$$

which is also a linear model. The analysis of these three models with the forms just shown, and related models, can be found in Box and Draper [1987], Hines, Montgomery, Goldsman, and Borror [2002], Montgomery [2000], and other applied statistics texts; and also in Kleijnen [1987, 1998], which is concerned primarily with the application of these models in simulation.

12.3.4 Random-Number Assignment for Regression

The assignment of random-number seeds or streams is part of the design of a simulation experiment.[4] Assigning a different seed or stream to different design points (settings for $x_1, x_2, \ldots x_m$ in a multiple linear regression) guarantees that the responses Y from different design points will be statistically independent. Similarly, assigning the same seed or stream to different design points induces dependence among the corresponding responses, by virtue of their all having the same source of randomness.

Many textbook experimental designs assume independent responses across design points. To conform to this assumption, we must assign different seeds or streams to each design point. However, it is often useful to assign the same random number seeds or streams to all of the design points—in other words, to use common random numbers.

The intuition behind common random numbers for metamodels is that a fairer comparison among design points is achieved if the design points are subjected to the same experimental conditions, specifically the same source of randomness. The mathematical justification is as follows: Suppose we fit the simple linear regression $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ and obtain least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$. Then an estimator of the expected difference in performance between design points i and j is

$$\hat{\beta}_0 + \hat{\beta}_1 x_i - (\hat{\beta}_0 + \hat{\beta}_1 x_j) = \hat{\beta}_1 (x_i - x_j)$$

when x_i and x_j are fixed design points, $\hat{\beta}_1$ determines the estimated difference between design points i and j, or for that matter between any other two values of x. Therefore, common random numbers can be expected to reduce the variance of $\hat{\beta}_1$ and, more generally, reduce the variance of all of the slope terms in a multiple linear regression. Common random numbers typically do not reduce the variance of the intercept term, $\hat{\beta}_0$.

[4] This section is based on Nelson [1992].

The least-squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ are appropriate regardless of whether we use common random numbers, but the associated statistical analysis is affected by that choice. For statistical analysis of a metamodel under common random numbers, see Kleijnen [1988] and Nelson [1992].

12.4 OPTIMIZATION VIA SIMULATION

Consider the following examples.[5]

Example 12.8: Materials Handling System (MHS)
Engineers need to design a MHS consisting of a large automated storage and retrieval device, automated guided vehicles (AGVs), AGV stations, lifters, and conveyors. Among the design variables they can control are the number of AGVs, the load per AGV, and the routing algorithm used to dispatch the AGVs. Alternative designs will be evaluated according to AGV utilization, transportation delay for material that needs to be moved, and overall investment and operation costs.

Example 12.9: Liquified Natural Gas (LNG) Transportation
A LNG transportation system will consist of LNG tankers and of loading, unloading, and storage facilities. In order to minimize cost, designers can control tanker size, number of tankers in use, number of jetties at the loading and unloading facilities, and capacity of the storage tanks.

Example 12.10: Automobile Engine Assembly
In an assembly line, a large buffer (queue) between workstations could increase station utilization—because there will tend to be something waiting to be processed—but drive up space requirements and work-in-process inventory. An allocation of buffer capacity that minimizes the sum of these competing costs is desired.

Example 12.11: Traffic Signal Sequencing
Civil engineers want to sequence the traffic signals along a bnsy section of road to reduce driver delay and the congestion occurring along narrow cross streets. For each traffic signal, the length of the red, green, and green-turn-arrow cycles can be set individually.

Example 12.12: On-Line Services
A company offering on-line information services over the Internet is changing its computer architecture from central mainframe computers to distributed workstation computing. The numbers and types of CPUs, the network structure, and the allocation of processing tasks all need to be chosen. Response time to customer queries is the key performance measure.

What do these design problems have in common? Clearly, a simulation model could be useful in each, and all have an implied goal of finding the best design relative to some performance measures (cost, delay, etc.). In each example, there are potentially a very large number of alternative designs, ranging from tens to thousands, and certainly more than the 2 to 10 we considered in Section 12.2.2. Some of the examples contain a diverse collection of decision variables: discrete (number of AGVs, number of CPUs), continuous (tanker size, red-cycle length) and qualitative (routing strategy, algorithm for allocating processing tasks). This makes developing a metamodel, as described in Section 12.3, difficult.

All of these problems fall under the general topic of "optimization via simulation," where the goal is to minimize or maximize some measures of system performance and system performance can be evaluated only by running a computer simulation. Optimization via simulation is a relatively new, but already vast, topic, and commercial software has become widely available. In this section, we describe the key issues that should be considered in undertaking optimization via simulation, provide some pointers to the available literature, and give one example algorithm.

[5]Some of these descriptions are based on Boesel, Nelson, and Ishii [2003].

12.4.1 What Does 'Optimization via Simulation' Mean?

Optimization is a key tool used by operations researchers and management scientists, and there are well-developed algorithms for many classes of problems, the most famous being linear programming. Much of the work on optimization deals with problems in which all aspects of the system are treated as being known with certainty; most critically, the performance of any design (cost, profit, makespan, etc.) can be evaluated exactly.

In stochastic, discrete-event simulation, the result of any simulation run is a random variable. For notation, let $x_1, x_2, \ldots, x_m$ be the m controllable design variables and let $Y(x_1, x_2, \ldots, x_m)$ be the observed simulation output performance on one run. To be concrete, x_1, x_2, x_3 might denote the number of AGVs, the load per AGV, and the routing algorithm used to dispatch the AGVs, respectively, in Example 12.8, while $Y(x_1, x_2, x_3)$ could be total MHS acquisition and operation cost.

What does it mean to "optimize" $Y(x_1, x_2, \ldots, x_m)$ with respect to $x_1, x_2, \ldots, x_m$? Y is a random variable, so we cannot optimize the *actual* value of Y. The most common definition of optimization is

$$\text{maximize or minimize } E(Y(x_1, x_2, \ldots, x_m)) \tag{12.40}$$

In other words, the mathematical expectation, or long-run average, of performance is maximized or minimized. *This is the default definition of optimization used in all commercial packages of which we are aware.* In our example, $E(Y(x_1, x_2, x_3))$ is the expected, or long-run average cost of operating the MHS with x_1 AGVs, x_2 load per AGV, and routing algorithm x_3.

It is important to note that (12.40) is not the only possible definition, however. For instance, we might want to select the MHS design that has the best chance of costing less than $\$D$ to purchase and operate, changing the objective to

$$\text{maximize Pr } (Y(x_1, x_2, x_3) \le D)$$

We can fit this objective into formulation (12.40) by defining a new performance measure

$$Y'(x_1, x_2, x_3) = \begin{cases} 1, & \text{if } Y(x_1, x_2, x_3) \le D \\ 0, & \text{otherwise} \end{cases}$$

and maximizing $E(Y'(x_1, x_2, x_3))$ instead.

A more complex optimization problem occurs when we want to select the system design that is *most likely* to be the best. Such an objective is relevant when one-shot, rather than long-run average, performance matters. Examples include a Space Shuttle launch, or the delivery of a unique, large order of products. Bechhofer, Santner, and Goldsman [1995] address this problem under the topic of "multinomial selection."

We have been assuming that a system design $x_1, x_2, \ldots, x_m$ can be evaluated in terms of a single performance measure, Y, such as cost. Obviously, this may not always be the case. In the MHS example, we might also be interested in some measure of system productivity, such as throughput or cycle time. At present, multiple objective optimization via simulation is not well developed. Therefore, one of three strategies is typically employed:

1. Combine all of the performance measures into a single measure, the most common being cost. For instance, the revenue generated by each completed product in the MHS could represent productivity and be included as a negative cost.
2. Optimize with respect to one key performance measure, but then evaluate the top solutions with respect to secondary performance measures. For instance, the MHS could be optimized with respect to expected cost, and then the cycle time could be compared for the top 5 designs. *This approach requires that information on more than just the best solution be maintained.*

3. Optimize with respect to one key performance measure, but consider only those alternatives that meet certain constraints on the other performance measures. For instance, the MHS could be optimized with respect to expected cost for those alternatives whose expected cycle time is less than a given threshold.

12.4.2 Why is Optimization via Simulation Difficult?

Even when there is no uncertainty, optimization can be very difficult if the number of design variables is large, the problem contains a diverse collection of design variable types, and little is known about the structure of the performance function. Optimization via simulation adds an additional complication: The performance of a particular design cannot be evaluated exactly, but instead must be estimated. Because we have estimates, it is not possible to conclude with assurance that one design is better than another, and this uncertainty frustrates optimization algorithms that try to move in improving directions. In principle, one can eliminate this complication by making so many replications, or such long runs, at each design point that the performance estimate has essentially no variance. In practice, this could mean that very few alternative designs will be explored, because of the time required to simulate each one.

The existence of sampling variability forces optimization via simulation to make compromises. The following are the standard ones:

- **Guarantee a prespecified probability of correct selection.** The Two-Stage Bonferroni Procedure in Section 12.2.2 is an example of this approach, which allows the analyst to specify the desired chance of being right. Such algorithms typically require either that every possible design be simulated or that a strong functional relationship among the designs (such as a metamodel) apply. Other algorithms can be found in Goldsman and Nelson [1998].
- **Guarantee asymptotic convergence.** There are many algorithms that guarantee convergence to the global optimal solution as the simulation effort (number of replications, length of replications) becomes infinite. These guarantees are useful because they indicate that the algorithm tends to get to where the analyst wants it to go. However, convergence can be slow, and there is often no guarantee as to how good the reported solution is when the algorithm is terminated in finite time (as it must be in practice). See Andradóttir [1998] for specific algorithms that apply to discrete- or continuous-variable problems.
- **Optimal for deterministic counterpart.** The idea here is to use an algorithm that would find the optimal solution *if the performance of each design could be evaluated with certainty.* An example might be applying a standard nonlinear programming algorithm to the simulation optimization problem. It is typically up to the analyst to make sure that enough simulation effort is expended (replications or run length) to insure that such an algorithm is not misled by sampling variability. Direct application of an algorithm that assumes deterministic evaluation to a stochastic simulation is not recommended.
- **Robust heuristics.** Many heuristics have been developed for deterministic optimization problems that do not guarantee finding the optimal solution, but nevertheless been shown to be very effective on difficult, practical problems. Some of these heuristics use randonmess as part of their search strategy, so one might argue that they are less sensitive to sampling variability than other types of algorithms. Nevertheless, it is still important to make sure that enough simulation effort is expended (replications or run length) to insure that such an algorithm is not misled by sampling variability.

Robust heuristics are the most common algorithms found in commercial optimization via simulation software. We provide some guidance on their use in the next section. See Fu [2002] for a comprehensive discussion of optimization theory versus practice.

12.4.3 Using Robust Heuristics

By a "robust heuristic" we mean a procedure that does not depend on strong problem structure—such as continuity or convexity of $E(Y(x_1,\ldots,x_m))$—to be effective, can be applied to problems with mixed types of decision variables, and—ideally—is tolerant of some sampling variability. Genetic algorithms (GA) and tabu search (TS) are two prominent examples, but there are many others and many variations of them. Such heuristics form the core of most commercial implementations. To give a sense of these heuristics, we describe GA and TS next. We caution the reader that only a high-level description of the simplest version of each procedure is provided. The commercial implementations are much more sophisticated.

Suppose that there are k possible solutions to the optimization via simulation problem. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ denote the solutions, where the ith solution $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ provides specific settings for the m decision variables. The simulation output at solution $\mathbf{x}_i$ is denoted $Y(\mathbf{x}_i)$; this could be the output of a single replication, or the average of several replications. Our goal is to find the solution $\mathbf{x}^*$ that minimizes $E(Y(\mathbf{x}))$.

On each iteration (known as a "generation"), a GA operates on a "population" of p solutions. Denote the population of solutions on the jth iteration as $\mathbf{P}(j) = \{\mathbf{x}_1(j), \mathbf{x}_2(j), \ldots, \mathbf{x}_p(j)\}$. There may be multiple copies of the same solution in $\mathbf{P}(j)$, and $\mathbf{P}(j)$ may contain solutions that were discovered on previous iterations. From iteration to iteration, this population evolves in such a way that good solutions tend to survive and give birth to new, and hopefully better, solutions, while inferior solutions tend to be removed from the population. The basic GA is given here:

Basic GA

Step 1. Set the iteration counter $j = 0$, and select (perhaps randomly) an initial population of p solutions $\mathbf{P}(0) = \{\mathbf{x}_1(0), \ldots, \mathbf{x}_p(0)\}$.

Step 2. Run simulation experiments to obtain performance estimates $Y(\mathbf{x})$ for all p solutions $\mathbf{x}(j)$ in $\mathbf{P}(j)$.

Step 3. Select a population of p solutions from those in $\mathbf{P}(j)$ in such a way that those with smaller $Y(\mathbf{x})$ values are more likely, but not certain, to be selected. Denote this population of solutions as $\mathbf{P}(j+1)$.

Step 4. Recombine the solutions in $\mathbf{P}(j+1)$ via crossover (which joins parts of two solutions $\mathbf{x}_i(j+1)$ and $\mathbf{x}_\ell(j+1)$ to form a new solution) and mutation (which randomly changes a part of a solution $\mathbf{x}_i(j+1)$.

Step 5. Set $j = j + 1$ and go to Step 2.

The GA can be terminated after a specified number of iterations, when little or no improvement is noted in the population, or when the population contains p copies of the same solution. At termination, the solution $\mathbf{x}^*$ that has the smallest $Y(\mathbf{x})$ value in the last population is chosen as best (or alternatively, the solution with the smallest $Y(\mathbf{x})$ over all iterations could be chosen).

GAs are applicable to almost any optimization problem, because the operations of selection, crossover, and mutation can be defined in a very generic way that does not depend on specifics of the problem. However, when these operations are not tuned to the specific problem, a GA's progress can be very slow. Commercial versions are often self-tuning, meaning that they update selection, crossover, and mutation parameters during the course of the search. There is some evidence that GAs are tolerant of sampling variability in $Y(\mathbf{x})$ because they maintain a population of solutions rather than focusing on improving a current-best solution. In other words, it is not critical that the GA rank the solutions in a population of solutions perfectly, because the next iteration depends on the entire population, not on a single solution.

TS, on the other hand, identifies a current best solution on each iteration and then tries to improve it. Improvements occur by changing the solution via "moves." For example, the solution (x_1, x_2, x_3) could be changed

to the solution $(x_1 + 1, x_2, x_3)$ by the move of adding 1 to the first decision variable (perhaps x_1 represents the number of AGVs in Example 12.8, so the move would add one more AGV). The "neighbors" of solution $\mathbf{x}$ are all of those solutions that can be reached by legal moves. TS finds the best neighbor solution and moves to it. However, to avoid making moves that return the search to a previously visited solution, moves may become "tabu" (not usable) for some number of iterations. Conceptually, think about how you would find your way through a maze: If you took a path that lead to a dead end, then you would avoid taking that path again (it would be tabu).

The basic TS algorithm is given next. The description is based on Glover [1989].

Basic TS

Step 1. Set the iteration counter $j = 0$ and the list of tabu moves to empty. Select an initial solution $\mathbf{x}^*$ in $\mathbf{X}$ (perhaps randomly).

Step 2. Find the solution $\mathbf{x}'$ that minimizes $Y(\mathbf{x})$ over all of the neighbors of $\mathbf{x}^*$ that are not reached by tabu moves, running whatever simulations are needed to do the optimization.

Step 3. If $Y(\mathbf{x}') < Y(\mathbf{x}^*)$, then $x^* = x'$ (move the current best solution to x').

Step 4. Update the list of tabu moves and go to Step 2.

The TS can be terminated when a specified number of iterations have been completed, when some number of iterations has passed without changing $\mathbf{x}^*$, or when there are no more feasible moves. At termination, the solution $\mathbf{x}^*$ is chosen as best.

TS is fundamentally a discrete-decision-variable optimizator, but continuous decision variables can be discretized, as described in Section 12.4.4. TS aggressively pursues improving solutions, and therefore tends to make rapid progress. However, it is more sensitive to random variability in $Y(\mathbf{x})$, because $\mathbf{x}^*$ is taken to be the *true* best solution so far and attempts are made to improve it. There are probabilistic versions of TS that should be less sensitive, however. An important feature of commercial implementations of TS, which is not present in the Basic TS, is a mechanism for overiding the tabu list when doing so is advantageous.

Next, we offer two suggestions for using commercial products that employ a GA, TS, or other robust heuristic controlling sampling variability, and *restarting*.

Control Sampling Variability

In many cases, it will up to the user to determine how much sampling (replications or run length) will be undertaken at each potential solution. This is a difficult problem in general. Ideally, sampling should increase as the heuristic closes in on the better solutions, simply because it is much more difficult to distinguish solutions that are close in expected performance from those that differ widely. Early in the search, it may be easy for the heuristic to identify good solutions and search directions, because clearly inferior solutions are being compared to much better ones, but late in the search this might not be the case.

If the analyst must specify a fixed number of replications per solution that will be used through the search, then a preliminary experiment should be conducted. Simulate several designs, some at the extremes of the solution space and some nearer the center. Compare the apparent best and apparent worst of these designs, using the approaches in Section 12.1. Using the technique described in Section 12.1.4, find the minimum for the number of replications required to declare these designs to be statistically significantly different. This is the minimum number of replications that should be used.

After the optimization run has completed, perform a second set of experiments on the top 5 to 10 designs identified by the heuristic. Use the comparison techniques in Section 12.2–12.2.3 to rigorously evaluate which are the best or near-best of these designs.

Restarting

Because robust heuristics provide no guarantees that they converge to the optimal solution for optimization via simulation, it makes sense to run the optimization two or more times to see which run yields the best solution. Each optimization run should use different random number seeds or streams and, ideally, should start from different initial solutions. Try starting the optimization at solutions on the extremes of the solution space, in the center of the space, and at randomly generated solutions. If people familiar with the system suspect that certain designs will be good, be sure to include them as possible starting solutions for the heuristic.

12.4.4 An Illustration: Random Search

In this section, we present an algorithm for optimization via simulation known as random search. The specific implementation is based on Algorithm 2 in Andradóttir [1998], which provides guaranteed asymptotic convergence under certain conditions. Thus, it will find the true optimal solution if permitted to run long enough. However, in practice, convergence can be slow, and the memory requirements of this particular version of random search can be quite large. Even though random search is not a "robust heuristic," we will also use it to demonstrate some strategies we would employ in conjunction with such heuristics and to demonstrate why optimization via simulation is tricky even with what appears to be an uncomplicated algorithm.

The random-search algorithm that we present requires that there be a finite number of possible system designs (although that number may be quite large). This might seem to rule out problems with continuous decision variables, such as conveyor speed. In practice, however, apparently continuous decision variables can often be discretized in a reasonable way. For instance, if conveyor speed can be anything from 60 to 120 feet per minute, little may be lost by treating the possible conveyor speeds as 60, 61, 62, ..., 120 feet per minute (61 possible values). Note, however, that there are algorithms designed specifically for continuous-variable problems (Andradóttir [1998]).

Again, let the k possible solutions to the optimization via simulation problem be denoted $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, where the ith solution $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ provides specific settings for the m decision variables. The simulation output at solution $\mathbf{x}_i$ is denoted $Y(\mathbf{x}_i)$; this could be the output of a single replication or the average of several replications. Our goal is to find the solution $\mathbf{x}^*$ that minimizes $E(Y(\mathbf{x}))$.

On each iteration of the random-search algorithm, we compare a current good solution to a randomly chosen competitor. If the competitor is better, then it becomes the current good solution. When we terminate the search, the solution we choose is the one that has been visited most often (which means that we expect to revisit solutions many times).

Random-Search Algorithm

Step 1. Initialize counter variables $C(i) = 0$ for $i = 1, 2, \ldots, k$. Select an initial solution i^0, and set $C(i^0) = 1$. ($C(i)$ counts the number of times we visit solution i.)

Step 2. Choose another solution i' from the set of all solutions *except* i^0 in such a way that each solution has an equal chance of being selected.

Step 3. Run simulation experiments at the two solutions i^0 and i' to obtain outputs $Y(i^0)$ and $Y(i')$. If $Y(i') < Y(i^0)$, then set $i^0 = i'$. (See note following Step 4.)

Step 4. Set $C(i^0) = C(i^0) + 1$. If not done, then go to Step 2. If done, then select as the estimated optimal solution xi^* such that $C(i^*)$ is the largest count.

Note that, if the problem is a maximization problem, then replace Step 3 with

Step 3. Run simulation experiments at the two solutions i^0 and i' to obtain outputs $Y(i^0)$ and $Y(i')$. If $Y(i') > Y(i^0)$, then set $i^0 = i'$.

One of the difficult problems with many optimization-via-simulation algorithms is knowing when to stop. (Exceptions include algorithms that guarantee a probability of correct selection.) Typical rules might be to stop after a certain number of iterations, stop when the best solution has not changed much in several iterations, or stop when all time available to solve the problem has been exhausted. Whatever rule is used, we recommend applying a statistical selection procedure, such as the Two-Stage Bonferroni Procedure in Section 12.2.2, to the 5 to 10 apparently best solutions. This is done to evaluate which among them is the true best with guaranteed confidence. If the raw data from the search have been saved, then these data can be used as the first-stage sample for a two-stage selection procedure (Boesel, Nelson, and Ishii [2003]).

Example 12.13: Implementing Random Search

Suppose that a manufacturing system consists of 4 stations in series. The zeroth station always has raw material available. When the zeroth station completes work on a part, it passes the part along to the first station, then the first passes the part to the second, and so on. Buffer space between stations 0 and 1, 1 and 2, and 2 and 3 is limited to 50 parts total. If, say, station 2 finishes a part but there is no buffer space available in front of station 3, then station 2 is blocked, meaning that it cannot do any further work. The question is how to allocate these 50 spaces to minimize the expected cycle time per part over one shift.

Let x_i be the number of buffer spaces in front of station i. Then the decision variables are x_1, x_2, x_3 with the constraint that $x_1 + x_2 + x_3 = 50$ (it makes no sense to allocate fewer buffer spaces than we have available). This implies a total of 1326 possible designs (can you figure out how this number is computed?).

To simplify the presentation of the random-search algorithm, let the counter for solution (x_1, x_2, x_3) be denoted as $C(x_1, x_2, x_3)$.

Random Search Algorithm

Step 1. Initialize 1326 counter variables $C(x_1, x_2, x_3) = 0$, one for each of the possible solutions (x_1, x_2, x_3). Select an initial solution, say $(x_1 = 20, x_2 = 15, x_3 = 15)$ and set $C(20, 15, 15) = 1$.

Step 2. Choose another solution from the set of all solutions *except* (20, 15, 15) in such a way that each solution has an equal chance of being selected. Suppose (11, 35, 4) is chosen.

Step 3. Run simulation experiments at the two solutions to obtain estimates of the expected cycle time $Y(20, 15, 15)$ and $Y(11, 35, 4)$. Suppose that $Y(20, 15, 15) < Y(11, 35, 4)$. Then (20, 15, 15) remains as the current good solution.

Step 4. Set $C(20, 15, 15) = C(20, 15, 15) + 1$.

Step 2. Choose another solution from the set of all solutions *except* (20, 15, 15) in such a way that each solution has an equal chance of being selected. Suppose (28, 12, 10) is chosen.

Step 3. Run simulation experiments at the two solutions to obtain estimates of the expected cycle time $Y(20, 15, 15)$ and $Y(28, 12, 10)$. Suppose that $Y(28, 12, 10) < Y(20, 15, 15)$. Then (28, 12, 10) becomes the current good solution.

Step 4. Set $C(28, 12, 10) = C(28, 12, 10) + 1$.

Step 2. Choose another solution from the set of all solutions *except* (28, 12, 10) in such a way that each solution has an equal chance of being selected. Suppose (0, 14, 36) is chosen.

Step 3. Continue...

When the search is terminated, we select the solution (x_1, x_2, x_3) that gives the largest $C(x_1, x_2, x_3)$ count. As we discussed earlier, the top 5 to 10 solutions should then be subjected to a separate statistical analysis to determine which among them is the true best (with high confidence). In this case, the solutions with the largest counts would receive the second analysis.

Despite the apparent simplicity of the Random-Search Algorithm, we have glossed over a subtle issue that often arises in algorithms with provable performance. In Step 2, the algorithm must randomly choose a solution such that all are equally likely to be selected (except the current one). How can this be accomplished in Example 12.13? The constraint that $x_1 + x_2 + x_3 = 50$ means that x_1, x_2 and x_3 cannot be sampled independently. One might be tempted to sample x_1 as a discrete uniform random variable on 0 to 50, then sample x_2 as a discrete uniform on 0 to $50 - x_1$, and finally set $x_3 = 50 - x_1 - x_2$. But this method does not make all solutions equally likely, as the following illustration shows: Suppose that x_1 is randomly sampled to be 50. Then the trial solution must be (50, 0, 0); there is only one choice. But if $x_1 = 49$, then both (49, 1, 0) and (49, 0, 1) are possible. Thus, $x_1 = 49$ should be more likely than $x_1 = 50$ if all solutions with $x_1 + x_2 + x_3 = 50$ are to be equally likely.

12.5 SUMMARY

This chapter provided a basic introduction to the comparative evaluation of alternative system designs based on data collected from simulation runs. It was assumed that a fixed set of alternative system designs had been selected for consideration. Comparisons based on confidence intervals and the use of common random numbers were emphasized. A brief introduction to metamodels—whose purpose is to describe the relationship between design variables and the output response—and to optimization via simulation—whose purpose is to select the best from among a large and diverse collection of system designs—was also provided. There are many additional topics of potential interest (beyond the scope of this text) in the realm of statistical analysis techniques relevant to simulation. Some of these topics are

1. experimental design models, whose purpose is to discover which factors have a significant impact on the performance of system alternatives;
2. output-analysis methods other than the methods of replication and batch means;
3. variance-reduction techniques, which are methods to improve the statistical efficiency of simulation experiments (common random numbers being an important example).

The reader is referred to Banks [1998] and Law and Kelton [2000] for discussions of these topics and of others relevant to simulation.

The most important idea in Chapters 11 and 12 is that simulation output data require a statistical analysis in order to be interpreted correctly. In particular, a statistical analysis can provide a measure of the precision of the results produced by a simulation and can provide techniques for achieving a specified precision.

REFERENCES

ANDRADÓTTIR, S. [1998], "Simulation Optimization," Chapter 9 in *Handbook of Simulation*, J. Banks, ed., Wiley, New York.

BANKS, J., ed. [1998], *Handbook of Simulation*, Wiley, New York.

BECHHOFER, R. E., T. J. SANTNER, AND D. GOLDSMAN [1995], *Design and Analysis for Statistical Selection, Screening and Multiple Comparisons*, Wiley, New York.

BOESEL, J., B. L. NELSON, AND N. ISHII [2003], "A Framework for Simulation-Optimization Software," *IIE Transactions*, Vol. 35, pp. 221–229.

BOX, G. E. P., AND N. R. DRAPER [1987], *Empirical Model-Building and Response Surfaces*, Wiley, New York.

FU, M. C. [2002], "Optimization for Simulation: Theory vs. Practice," *INFORMS Journal on Computing*, Vol. 14, pp. 192–215.

GLOVER, F. [1989], "Tabu Search—Part I," *ORSA Journal on Computing*, Vol. 1, pp. 190–206.

GOLDSMAN, D., AND B. L. NELSON [1998], "Comparing Systems via Simulation," Chapter 8 in *Handbook of Simulation*, J. Banks, ed., Wiley, New York.

HINES, W. W., D. C. MONTGOMERY, D. M. GOLDSMAN, AND C. M. BORROR [2002], *Probability and Statistics in Engineering*, 4th ed., Wiley, New York.
HOCHBERG Y., AND A. C. TAMHANE [1987], *Multiple Comparison Procedures*, Wiley, New York.
HSU, J. C. [1996], *Multiple Comparisons: Theory and Methods*, Chapman & Hall, New York.
KLEIJNEN, J. P. C. [1975], *Statistical Techniques in Simulation, Parts I and II*, Dekker, New York.
KLEIJNEN, J. P. C. [1987], *Statistical Tools for Simulation Practitioners*, Dekker, New York.
KLEIJNEN, J. P. C. [1988], "Analyzing Simulation Experiments with Common Random Numbers," *Management Science*, Vol. 34, pp. 65–74.
KLEIJNEN, J. P. C. [1998], "Experimental Design for Sensitivity Analysis, Optimization, and Validation of Simulation Models," Chapter 6 in *Handbook of Simulation*, J. Banks, ed., Wiley, New York.
LAW, A. M., AND W. D. KELTON [2000], *Simulation Modeling and Analysis*, 3d ed., McGraw–Hill, New York.
MONTGOMERY, D. C. [2000], *Design and Analysis of Experiments*, 5th ed., Wiley, New York.
NELSON, B. L., AND F. J. MATEJCIK [1995], "Using Common Random Numbers for Indifference-Zone Selection and Multiple Comparisons in Simulation," *Management Science*, Vol. 41, pp. 1935–1945.
NELSON, B. L., J. SWANN, D. GOLDSMAN, AND W.-M. T. SONG [2001], "Simple Procedures for Selecting the Best System when the Number of Alternatives is Large," *Operations Research*, Vol. 49, pp. 950–963.
NELSON, B. L. [1987], "Some Properties of Simulation Interval Estimators Under Dependence Induction," *Operations Research Letters*, Vol. 6, pp. 169–176.
NELSON, B. L. [1992], "Designing Efficient Simulation Experiments," *1992 Winter Simulation Conference Proceedings*, pp. 126–132.
WRIGHT, R. D., AND T. E. RAMSAY, JR. [1979], "On the Effectiveness of Common Random Numbers," *Management Science*, Vol. 25, pp. 649–656.

EXERCISES

1. Reconsider the dump-truck problem of Example 3.5, which was also analyzed in Example 12.2. As business expands, the company buys new trucks, making the total number of trucks now equal to 16. The company desires to have a sufficient number of loaders and scales so that the average number of trucks waiting at the loader queue plus the average number at the weigh queue is no more than three. Investigate the following combinations of number of loaders and number of scales:

Number of Scales	*Number of Loaders* 2	3	4
1	–	–	–
2	–	–	–

The loaders being considered are the "slow" loaders in Example 12.2. Loading time, weighing time, and travel time for each truck are as previously defined in Example 12.2. Use common random numbers to the greatest extent possible when comparing alternative systems designs. The goal is to find the smallest number of loaders and scales to meet the company's objective of an average total queue length of no more than three trucks. In your solution, take into account the initialization conditions, run length, and number of replications needed to achieve a reasonable likelihood of valid conclusions.

2. In Exercise 11.5, consider the following alternative (M, L) policies:

			L *Low* 30	*High* 40
M	Low	50	(50, 30)	(50, 40)
	High	100	(100, 30)	(100, 40)

Investigate the relative costs of these policies, using suitable modifications of the simulation model developed in Exercise 11.5. Compare the four system designs on the basis of long-run mean monthly cost. First make four replications of each (M, L) policy, using common random numbers to the greatest extent possible. Each replication should have a 12-month initialization phase followed by a 100-month data-collection phase. Compute confidence intervals having an overall confidence level of 90% for mean monthly cost for each policy. Then estimate the additional replications needed to achieve confidence intervals that do not overlap. Draw conclusions as to which is the best policy.

3. Reconsider Exercise 11.6. Compare the four inventory policies studied in Exercise 2, taking the cost of rush orders into account when computing monthly cost.

4. In Exercise 11.8, investigate the effect of the order quantity on long-run mean daily cost. Each order arrives on a pallet on a delivery truck, so the permissible order quantities, Q, are multiples of 10 (i.e., Q may equal 10, or 20, or 30, ...). In Exercise 11.8, the policy $Q = 20$ was investigated.
   (a) First, investigate the two policies $Q = 10$ and $Q = 50$. Use the run lengths, and so on, suggested in Exercise 11.8. On the basis of these runs, decide whether the optimal Q, say Q^*, is between 10 and 50 or is greater than 50. (The cost curve as a function of Q should have what kind of shape?)
   (b) Using the results in part (a), suggest two additional values for Q and simulate the two policies. Draw conclusions. Include an analysis of the strength of your conclusions.

5. In Exercise 11.10, find the number of cards Q that the card shop owner should purchase to maximize the profit with an error of approximately \$5.00. Use the following expression to generate Q value

$$Q = 300 \pm 100$$

For each run, generate a uniform random variate to get the Q value and for that Q value compute profit.

6. In Exercise 11.10, investigate the effect of target level M and review period N on mean monthly cost. Consider two target levels, M, determined by ±10 from the target level used in Exercise 11.10, and consider review periods N of 1 month and 3 months. Which (N, M) pair is best, according to these simulations?

7. Reconsider Exercises 11.12 and 11.13, which involved the scheduling rules (or queue disciplines) first-in–first-out (FIFO) and priority-by-type (PR) in a job shop. In addition to these two rules, consider a shortest imminent operation (SIO) scheduling rule. For a given station, all jobs of the type with the smallest mean processing time are given highest priority. For example, when using an SIO rule at station 1, jobs are processed in the following order: type 2 first, then type 1, and type 3 last. Two jobs of the same type are processed on a FIFO basis. Develop a simulation experiment to compare the FIFO, PR, and SIO rules on the basis of mean total response time over all jobs.

8. In Exercise 11.12 (the job shop with FIFO rule), find the minimum number of workers needed at each station to avoid bottlenecks. A bottleneck occurs when average queue lengths at a station increase steadily over time. (Do not confuse increasing average queue length due to an inadequate number of servers with increasing average queue length due to initialization bias. In the former case, average queue length continues to increase indefinitely and server utilization is 1.0. In the latter case, average queue length eventually levels off and server utilization is less than 1.) Report on utilization of workers and total time it takes for a job to get through the job shop, by type and over all types. (*Hint*: If server

utilization at a work station is 1.0, and if average queue length tends to increase linearly as simulation run length increases, it is a good possibility that the work station is unstable and therefore is a bottleneck. In this case, at least one additional worker is needed at the work station. Use queueing theory, namely $\lambda/c_i\mu < 1$, to suggest the minimum number of workers needed at station 1. Recall that λ is the arrival rate, $1/\mu$ is the overall mean service time for one job with one worker, and c_i is the number of workers at station i. Attempt to use the same basic condition, $\lambda/c_i\mu < 1$, to suggest an initial number of servers at station i for $i = 2, 3, 4$.)

9. (a) Repeat Exercise 8 for the PR scheduling rule (see Exercise 11.13).
   (b) Repeat Exercise 8 for the SIO scheduling rule (see Exercise 12.7).
   (c) Compare the minimum required number of workers for each scheduling rule: FIFO, versus PR, versus SIO.

10. With the minimum number of workers found in Exercises 9 and 10 for the job shop of Exercise 11.12, consider adding one worker to the entire shop. This worker can be trained to handle the processing at only one station. At which station should this worker be placed? How does this additional worker affect mean total response time over all jobs? Over type 1 jobs? Investigate the job shop with and without the additional worker for each scheduling rule: FIFO, PR, SIO.

11. In Exercise 11.16, suppose that a buffer of capacity one item is constructed in front of each worker. Design an experiment to investigate whether this change in system design has a significant impact upon individual worker utilizations (ρ_1, ρ_2, ρ_3 and ρ_4). At the very least, compute confidence intervals for $\rho_1^0 - \rho_1^1$ and $\rho_4^0 - \rho_4^1$, where ρ_i^s, is utilization for worker i when the buffer has capacity s.

12. A clerk in the admissions office at Small State University processes requests for admissions materials. The time to process requests depends on the program of interest (e.g., industrial engineering, management science, computer science, etc.) and on the level of the program (Bachelors, Masters, Ph.D.). Suppose that the processing time is modeled well as normally distributed, with mean 7 minutes and standard deviation 2 minutes. At the beginning of the day it takes the clerk some time to get set to begin working on requests; suppose that this time is modeled well as exponentially distributed, with mean 20 minutes. The admissions office typically receives between 40 and 60 requests per day.

    Let x be the number of applications received on a day, and let Y be the time required to process them (including the set-up time). Fit a metamodel for $E(Y|x)$ by making n replications at the design points $x = 40, 50, 60$. Notice that, in this case, we know that the correct model is

$$E(Y|x) = \beta_0 + \beta_1 x = 20 + 7x$$

    (Why?) Begin with $n = 2$ replications at each design point and estimate β_0 and β_1. Gradually increase the number of replications and observe how many are required for the estimates to be close to the true values.

13. Repeat the previous exercise using CRN. How do the results change?

14. The usual statistical analysis used to test for $\beta_1 \neq 0$ does not hold if we use CRN. Where does it break down?

15. Riches and Associates retains its cash reserves primarily in the form of certificates of deposit (CDs), which earn interest at an annual rate of 8%. Periodically, however, withdrawals must be made from these CDs in order to pay suppliers, etc. These cash outflows are made through a checking account that earns no interest. The need for cash cannot be predicted with certainty. Transfers from CDs to checking can be made instantaneously, but there is a "substantial penalty" for early withdrawal from CDs. Therefore, it might make sense for R&A to make use of the overdraft protection on their checking account, which charges interest at a rate of \$0.00033 per dollar per day (i.e., 12% per year) for overdrafts.

    R&A likes simple policies in which it transfers a fixed amount, a fixed number of times, per year. Currently, it makes 6 transfers per year, of \$18,250 each time. Your job is to find a policy that reduces its long-run cost per day.

    Judging from historical patterns, demands for cash arrive a rate of about 1 per day, with the arrivals being modeled well as a Poisson process. The amount of cash needed to satisfy each demand is reasonably represented by a lognormally distributed random variable with mean \$300 and standard deviation \$150.

    The penalty for early withdrawal is different for different CDs. It averages \$150 for each withdrawal (regardless of size), but the actual penalty can be modeled as a uniformly distributed random variable with range \$100 to \$200.

    Use cash level in checking to determine the length of the initialization phase. Make enough replications that your confidence interval for the difference in long-run cost per day does not contain zero. Be sure to use CRN in your experiment design.

16. If you have access to commercial optimization-via-simulation software, test how well it works as the variability of the simulation outputs increases. Use a simple model, such as $Y = x^2 + \varepsilon$, where ε is a random variable with a $N(0, \sigma^2)$ distribution, and for which the optimal solution is known ($x = 0$ for minimization, in this case). See how quickly, or whether, the software can find the true optimal solution as σ^2 increases. Next, try more complex models with more than one design variable.

17. For Example 12.12, show why there are 1326 solutions. Then derive a way to sample x_1, x_2, and x_3 such that $x_1 + x_2 + x_3 = 50$ and all outcomes are equally likely.

18. A critical electronic component with mean time to failure of x years can be purchased for $2x$ thousand dollars (thus, the more reliable the component, the more expensive it is). The value of x is restricted to being between 1 to 10 years, and the actual time to failure is modeled as exponentially distributed. The mission for which the component is to be used lasts one year; if the component fails in less than one year, then there is a cost of \$20,000 for early failure. What value of x should be chosen to minimize the expected total cost (purchase plus early failure)?

    To solve this problem, develop a simulation that generates a total cost for a component with mean time to failure of x years. This requires sampling an exponentially distributed random variable with mean x, and then computing the total cost as $2000x$ plus 20,000 if the failure time is less than 1. Fit a quadratic metamodel in x and use it to find the value of x that minimizes the fitted model. [Hints: Select several values of x between 1 and 10 as design points. At each value of x, let the response variable $Y(x)$ be the average of at least 30 observations of total cost.]

19. The demand for an item follows $N(10, 2)$. It is required to avoid the shortage. Let Q be the order quantity. Assuming Q to be an integer between 10 and 150, determine the optimal value for Q that maximizes the probability, so that the shortage is equal to zero. Use random search algorithm.

20. If you have access, use any optimization via simulation software to solve Exercise 19.

21. Explore the possibility of applying metaheuristics to search for near-optimal solution using simulation models.

Part V

Applications

13

Simulation of Manufacturing and Material-Handling Systems

Manufacturing and material-handling systems provide one of the most important applications of simulation. Simulation has been used successfully as an aid in the design of new production facilities, warehouses, and distribution centers. It has also been used to evaluate suggested improvements to existing systems. Engineers and analysts using simulation have found it valuable for evaluating the impact of capital investments in equipment and physical facility and of proposed changes to material handling and layout. They have also found it useful to evaluate staffing and operating rules and proposed rules and algorithms to be incorporated into production control systems, warehouse-management control software, and material-handling controls. Managers have found simulation useful in providing a "test drive" before making capital investments, without disrupting the existing system with untried changes.

Section 13.1 provides an introduction and discusses some of the features of simulation models of manufacturing and material-handling systems. Section 13.2 discussed the goals of manufacturing simulation and the most common measures of system performance. Section 13.3 discusses a number of the issues common to many manufacturing and material-handling simulations, including the treatment of downtimes and failure, and trace-driven simulations using actual historical data or historical order files. Section 13.4 provides brief abstracts of a number of reported simulation projects, with references for additional reading. Section 13.5 gives an extended example of a simulation of a small production line, emphasizing the experimentation and analysis of system performance to achieve a desired throughput. For an overview of simulation software for manufacturing and material-handling applications, see Section 4.7.

13.1 MANUFACTURING AND MATERIAL-HANDLING SIMULATIONS

As do all modeling projects, manufacturing and material-handling simulation projects need to address the issues of scope and level of detail. Consider scope as analagous to breadth and level of detail as analagous to depth. Scope describes the boundaries of the project: what's in the model, and what's not. For a subsystem, process, machine, or other component, the project scope determines whether the object is in the model. Then, once a component or subsystem is treated as part of a model, often it can be simulated at many different levels of detail.

The proper scope and level of detail should be determined by the objectives of the study and the questions being asked. On the other hand, level of detail could be constrained by the availability of input data and the knowledge of how system components work. For new, nonexistent systems, data availability might be limited, and system knowledge might be based on assumptions.

Some general guidelines can be provided, but the judgment of experienced simulation analysts working with the customer to define, early in the project, the questions the model is being designed to address provides the most effective basis for selecting a proper scope and a proper level of detail.

Should the model simulate each conveyor section or vehicle movement, or can some be replaced by a simple time delay? Should the model simulate auxiliary parts, or the handling of purchased parts, or can the model assume that such parts are always available at the right location when needed for assembly?

At what level of detail does the control system need to be simulated? Many modern manufacturing facilities, distribution centers, baggage-handling systems, and other material-handling systems are computer controlled by a management-control software system. The algorithms built into such control software play a key role in system performance. Simulation is often used to evaluate and compare the effectiveness of competing control schemes and to evaluate suggested improvements. It can be used to debug and fine-tune the logic of a control system before it is installed.

These questions are representative of the issues that need to be addressed in choosing the correct level of model detail and scope of a project. In turn, the scope and level of model detail limit the type of questions that can be addressed by the model. In addition, models can be developed in an iterative fashion, adding detail for peripheral operations at later stages if such operations are later judged to affect the main operation significantly. It is good advice to start as simple as possible and add detail only as needed.

13.1.1 Models of Manufacturing Systems

Models of manufacturing systems might have to take into account a number of characteristics of such systems, some of which are the following:

- Physical layout
- Labor
  - Shift schedules
  - Job duties and certification
- Equipment
  - Rates and capacities
  - Breakdowns
    - Time to failure
    - Time to repair
    - Resources needed for repair
- Maintenance
  - PM schedule
  - Time and resources required
  - Tooling and fixtures

- Workcenters
  - Processing
  - Assembly
  - Disassembly
- Product
  - Product flow, routing, and resources needed
  - Bill of materials
- Production schedules
  - Made-to-stock
  - Made-to-order
    - Customer orders
    - Line items and quantities
- Production control
  - Assignment of jobs to work areas
  - Task selection at workcenters
  - Routing decisions
- Supplies
  - Ordering
  - Receipt and storage
  - Delivery to workcenters
- Storage
  - Supplies
  - Spare parts
  - Work-in-process (WIP)
  - Finished goods
- Packing and shipping
  - Order consolidation
  - Paperwork
  - Loading of trailers

13.1.2 Models of Material Handling Systems

In manufacturing systems, it is not unusual for 80 to 85% of an item's total time in system to be expended on material handling or on waiting for material handling to occur. This work-in-process (WIP) represents a vast investment, and reductions in WIP and associated delays can result in large cost savings. Therefore, for some studies, detailed material-handling simulations are cost effective.

In some production lines, the material-handling system is an essential component. For example, automotive paint shops typically consist of a power-and-free conveyor system that transports automobile bodies or body parts through the paint booths.

In warehouses, distribution centers, and flow-through and cross-docking operations, material handling is clearly a key component of any material-flow model. Manual warehouses typically use manual fork trucks to move pallets from receiving dock to storage and from storage to shipping dock. More automated distribution centers might use extensive conveyor systems to support putaway, order picking, order sortation, and consolidation.

Models of material-handling systems often have to contain some of the following types of subsystems:

- Conveyors
  - Accumulating
  - Nonaccumulating

- Indexing and other special purpose
- Fixed window or random spacing
- Power and free

Transporters
- Unconstrained vehicles (e.g., manually guided fork trucks)
- Guided vehicles (automated or operator controlled, wire guided chemical paths, rail guided)
- Bridge cranes and other overhead lifts

Storage systems
- Pallet storage
- Case storage
- Small-part storage (totes)
- Oversize items
- Rack storage or block stacked
- Automated storage and retrieval systems (AS/RS) with storage–retrieval machines (SRM)

13.1.3 Some Common Material-Handling Equipment

There are numerous types of material-handling devices common to manufacturing, warehousing, and distribution operations. They include unconstrained transporters, such as carts, manually driven fork-lift trucks, and pallet jacks; guided path transporters, such as AGVs (automated guided vehicles); and fixed-path devices, such as various types of conveyor.

The class of unconstrained transporters, sometimes called free-path transporters, includes carts, fork-lift trucks, pallet jacks, and other manually driven vehicles that are free to travel throughout a facility unconstrained by a guide path of any kind. Unconstrained transporters are not constrained to a network of paths and may choose an alternate path or move around an obstruction. In contrast, the guided-path transporters move along a fixed path, such as chemical trails on the floor, wires imbedded in the floor, or infrared lights placed strategically, or by self-guidance, using radio communications, laser guidance and dead reckoning, and rail. Guided-path transporters sometimes contend with each other for space along their paths and usually have limited options upon meeting obstacles and congestion. Examples of guided-path transporters include the automated guided vehicle (AGV); a rail-guided turret truck for storage and retrievals of pallets in rack storage; and a crane in an AS/RS (automated storage and retrieval system).

The conveyor is a fixed-path device for moving entities from point to point, following a fixed path with specific load, stopping or processing points, and unload points. A conveyor system can consist of numerous connected sections with merges and diverts. Each section can be of one of a number of different types. Examples of conveyor types include belt, powered and gravity roller, bucket, chain, tilt tray, and power-and-free, each with its own characteristics that must be modeled accurately.

Most conveyor sections can be classified as either accumulating or nonaccumulating. An accumulating conveyor section runs continuously. If the forward progress of an item is halted while on the accumulating conveyor, slippage occurs, allowing the item to remain stationary and items behind it to continue moving until they reach the stationary item. Some belt and most roller conveyors operate in this manner. Only items that will not be damaged by bumping into each other can be placed on an accumulating conveyor.

In contrast, after an item is on a nonaccumulating conveyor section, its spacing relative to other items does not change. If one item stops moving, the entire section stops moving, and hence all items on the section stop. For example, nonaccumulating conveyor is used for moving televisions not yet in cartons, for they must be kept at a safe distance from each other while moving from one assembly or testing station to the next. Bucket conveyors, tilt-tray conveyors, some belt conveyors, and conveyors designed to carry heavy loads (usually, pallets) are nonaccumulating conveyors.

Conveyors can also be classified as fixed-window or random spacing. In fixed-window spacing, items on the conveyor must always be within zones of equal length, which can be pictured as lines drawn on a belt conveyor or trays pulled by a chain. For example, in a tilt-tray conveyor, continuously moving trays of fixed size are used to move items. The control system is designed to induct items in such a way that each item is in a separate tray; thus it is a nonaccumulating fixed-window conveyor. In contrast, with random spacing, items can be anywhere on the conveyor section relative to other items. To be inducted, they simply require sufficient space.

Besides these basic types, there are innumerable types of specialized conveyors for special purposes. For example, a specialized indexing conveyor may move forward in increments, always maintaining a fixed distance between the trailing edge of the load ahead and the leading edge of the load behind. Its purpose is to form a "slug" of items, equally spaced apart, to be inducted all together onto a transport conveyor. For the local behavior of some systems—that is, the performance at a particular workstation or induction point—a detailed understanding and accurate model of the physical workings and the control logic are essential for accurate results.

13.2 GOALS AND PERFORMANCE MEASURES

The purpose of simulation is insight, not numbers. Those who purchase and use simulation software and services want to gain insight and understanding into how a new or modified system will work. Will it meet throughput expectations? What happens to response time at peak periods? Is the system resilient to short-term surges? What is the recovery time when short-term surges cause congestion and queueing? What are the staffing requirements? What problems occur? If problems occur, what is their cause and how do they arise? What is the system capacity? What conditions and loads cause a system to reach its capacity?

Simulations are expected to provide numeric measures of performance, such as throughput under a given set of conditions, but the major benefit of simulation comes from the insight and understanding gained regarding system operations. Visualization through animation and graphics provides major assistance in the communication of model assumptions, system operations, and model results. Often, visualization is the major contributor to a model's credibility, which in turn leads to acceptance of the model's numeric outputs. Of course, a proper experimental design that includes the right range of experimental conditions plus a rigorous analysis and, for stochastic simulation models, a proper statistical analysis is of utmost importance for the simulation analyst to draw correct conclusions from simulation outputs.

The major goals of manufacturing-simulation models are to identify problem areas and quantify system performance. Common measures of system performance include the following:

- throughput under average and peak loads;
- system cycle time (how long it takes to produce one part);
- utilization of resources, labor, and machines;
- bottlenecks and choke points;
- queueing at work locations;
- queueing and delays caused by material-handling devices and systems;
- WIP storage needs;
- staffing requirements;
- effectiveness of scheduling systems;
- effectiveness of control systems.

Often, material handling is an important part of a manufacturing system and its performance. Non-manufacturing material-handling systems include warehouses, distribution centers, cross-docking

operations, baggage-handling systems at airports and container terminals. The major goals of these nonmanufacturing material-handling systems are similar to those identified for manufacturing systems. Some additional considerations are the following:

- how long it takes to process one day of customer orders;
- effect of changes in order profiles (for distribution centers);
- truck/trailer queueing and delays at receiving and shipping docks;
- effectiveness of material-handling systems at peak loads;
- recovery time from short-term surges (for example, with baggage-handling).

13.3 ISSUES IN MANUFACTURING AND MATERIAL-HANDLING SIMULATIONS

There are a number of modeling issues especially important for the achievement of accurate and valid simulation models of manufacturing and material-handling systems. Two of these issues are the proper modeling of downtimes and whether, for some inputs, to use actual system data or a statistical model of those inputs.

13.3.1 Modeling Downtimes and Failures

Unscheduled random downtimes can have a major effect on the performance of manufacturing systems. Many authors have discussed the proper modeling of downtime data (Williams [1994]; Clark [1994]; Law and Kelton [2000]). This section discusses the problems that can arise when downtime is modeled incorrectly and suggests a number of ways to model machine and system downtimes correctly.

Scheduled downtime, such as for preventive maintenance, or periodic downtime, such as for tool replacement, also can have a major effect on system performance. But these downtimes are usually (or should be) predictable and can be scheduled to minimize disruptions. In addition, engineering efforts or new technology might be able to reduce their duration.

There are a number of alternatives for modeling random unscheduled downtime, some better than others:

1. Ignore it.
2. Do not model it explicitly, but increase processing times in appropriate proportion.
3. Use constant values for time to failure and time to repair.
4. Use statistical distributions for time to failure and time to repair.

Of course, alternative (1) generally is not the suggested approach. This is certainly an irresponsible modeling technique if downtimes have an impact on the results, as they do in almost all situations. One situation in which ignoring downtimes could be appropriate, with the full knowledge of the customer, is to leave out those catastrophic downtimes that occur rarely and leave a production line or plant down for a long period of time. In other words, the model would incorporate normal downtimes but ignore those catastrophic downtimes, such as general power failures, snow storms, cyclones, and hurricanes, that occur rarely but stop all production when they do occur. The documented scope of the project should clearly state the assumed operating conditions and those conditions that are not included in the model. If it is generally known that a plant will be closed for some number of snow days per year, then the simulation need not take these downtimes into account, for the effect of any given number of days can easily be factored into the simulation results when making annual projections.

The second possibility, to factor into the model the effect of downtimes by adjusting processing times applied to each job or part, might be an acceptable approximation under limited circumstances. If each job or part is subject to a large number of small delays associated with downtime of equipment or tools, then the total of such delays may be added to the pure processing time to arrive at an adjusted processing time. If total delay time and pure processing time are random in nature, then an appropriate statistical distribution should be used for the total adjusted processing time. If the pure processing time is constant while the total delay time in one cycle is random and variable, it is almost never accurate to adjust the processing time by a constant factor. For example, if processing time is usually 10 minutes but the equipment is subject to downtimes that cause about a 10% loss in capacity, it is not appropriate to merely change the processing time to a constant 11 minutes. Such a deterministic adjustment might provide reasonably accurate estimates of overall system throughput, but will not provide accurate estimates of such local behavior as queue and buffer space needed at peak times. Queueing and short-term congestion are strongly influenced by randomness and variability.

The third possibility, using constant durations for time to failure and time to repair, might be appropriate when, for example, the downtime is actually due to preventive maintenance that is on a fixed schedule. In almost all other circumstances, the fourth possibility, modeling time to failure and time to repair by appropriate statistical distributions, is the appropriate technique. This requires either actual data for choosing a statistical distribution based on the techniques in Chapter 11, or, when data is lacking, a reasonable assumption based on the physical nature of the causes of downtimes.

The nature of time to failure is also important. Are times to failure completely random in nature, a situation due typically to a large number of possible causes of failure? In this case, exponential distribution might provide a good statistical model. Or are times to failure, rather, more regular—typically, due to some major component—say, a tool—wearing out? In this case, a uniform or (truncated) normal distribution could be more nearly appropriate. In the latter case, the mean of the distribution represents the average time to failure, and the distribution places a plus or minus around the mean.

Time to failure can be measured in a number of different ways:

1. by wall-clock time;
2. by machine or equipment busy time;
3. by number of cycle times;
4. by number of items produced.

Breakdowns or failures can be based on clock time, actual usage, or cycles. Note that the word breakdown or failure is used, even though preventive maintenance could be the reason for a downtime. As mentioned, breakdowns or failures can be probabilistic or deterministic in duration.

Actual usage breakdowns are based on the time during which the resource is used. For example, wear on a machine tool occurs only when the machine is in use. Time to failure is measured against machine-busy time and not against wall-clock time. If the time to failure is 90 hours, then the model keeps track of total busy time since the last downtime ended, and, when 90 hours is reached, processing is interrupted and a downtime occurs.

Clock-time breakdowns might be associated with scheduled maintenance—for example, changes of fluids every three months when a complete lubrication is required. Downtimes based on wall-clock time may also be used for equipment that is always busy or equipment that "runs" even when it is not processing parts.

Cycle breakdowns or failures are based on the number of times the resource is used. For example, after every 50 uses of a tool, it needs to be sharpened. Downtimes based on number of cycle times or number of items produced are implemented by generating the number of times or items and, in the model, simply counting until this number is reached. Typical uses of downtimes based on busy time or cycle times may be for maintenance or tool replacement.

Another issue is what happens to a part at a machine when the breakdown or failure occurs. Possibilities include scrapping the part, rework, or simply continuing processing after repair. In some cases—for example,

when preventive maintenance is due—the part in the machine may complete processing before the repair (or maintenance activity) begins.

Time to repair can also be modeled in two fundamentally different ways:

1. as a pure time delay (no resources required);
2. as a wait time for a resource (e.g., maintenance person) plus a time delay for actual repair.

Of course, there are many variations on these methods in actual modeling situations. When a repair or maintenance person is a limited resource, the second approach will be a more accurate model and provide more information.

The next example illustrates the importance of using the proper approach for modeling downtimes and of the consequences and inaccurate results that sometimes result from inaccurate assumptions.

Example 13.1: Effect of Downtime on Queueing

Consider a single machine that processes a wide variety of parts that arrive in random mixes at random times. Data analysis has shown that an exponentially distributed processing time with a mean of 7.5 minutes provides a fairly accurate representation. Parts arrive at random, time between arrivals being exponentially distributed with mean 10 minutes. The machine fails at random times. Downtime studies have shown that time-to-failure can be reasonably approximated by an exponential distribution with mean time 1000 minutes. The time to repair the resource is also exponentially distributed, with mean time 50 minutes. When a failure occurs, the current part in the machine is removed from the machine; when the repair has been completed, the part resumes its processing.

When a part arrives, it queues and waits its turn at the machine. It is desired to estimate the size of this queue. An experiment was designed to estimate the average number of parts in the queue. To illustrate the effect of an accurate treatment of downtimes, the model was run under a number of different assumptions. For each case and replication, the simulation run length was 100,000 minutes.

Table 13.1 shows the average number of parts in the queue for six different treatments of the time between breakdowns. For each treatment that involves randomness, five replications of those treatments and the average for those five replications are shown.

Case A ignores the breakdowns. The average number in the queue is 2.31 parts. Across the 5 independent replications, the averages range from 2.05 to 2.70 parts. This treatment of breakdowns is not recommended.

Case B increases the average service time from 7.5 minutes to 8.0 minutes in an attempt to approximate the effect of downtimes. On average, each downtime and repair cycle is 1050 minutes, with the machine down for 50 minutes. Thus the machine is down, on the average in the long run, 50/1050 = 4.8% of total time. Thus, some have argued that downtime has approximately the same effect as increasing the processing

Table 13.1 Average Number of Parts in Queue for Machines with Breakdowns

Case	*1st Rep*	*2nd Rep*	*3rd Rep*	*4th Rep*	*5th Rep*	*Avg Rep*
A. Ignore the breakdowns	2.36	2.05	2.38	2.05	2.70	2.31
B. Increase service time to 8.0	3.32	2.82	3.32	2.81	4.03	3.26
C. All random	4.05	3.77	4.36	3.95	4.43	4.11
D. Random processing, deterministic breakdowns	3.24	2.85	3.28	3.05	3.79	3.24
E. All deterministic						0.52
F. Deterministic processing, Random breakdowns	1.06	1.04	1.10	1.32	1.16	1.13

time of each part by 4.8%, which is about 7.86 minutes. Therefore, an assumed constant 8 minutes per part should be (it is argued) a conservative approach. For this treatment of downtimes, the average number of parts in the queue, over the five replications, is about 3.26 parts. Across the 5 replications, the range is from 2.81 to 4.03 parts. (Note that the variability as shown in the range of values is very small compared to the other cases.) The treatment in Case B might be appropriate under some limited circumstances, but, as was discussed in a previous section, it is not appropriate under the assumptions of this example.

The proper treatment, shown as Case C, treats the randomness in processing and breakdowns properly, with the assumed correct exponential distributions. The average value is about 4.11 parts waiting for the machine. Across the 5 replications, the average queue length ranges from 3.77 to 4.43 parts. The average number waiting differs from that of Case B by almost one part.

Case D is a simplification that treats the processing randomly, but treats the breakdowns as deterministic. The results average about 3.24 parts in the queue. The range of averages is from 2.85 to 3.79 parts, quite a reduction in variability from Case C.

Case E treats all of the times as deterministic. Only one replication is needed, because additional replications (using the same seed) will reproduce the result. The average value in the queue is 0.52 parts, well below the value in Case C, or any other case for that matter. The conclusion: Ignoring randomness is dangerous and leads to totally unrealistic results.

Case F treats arrivals and processing as deterministic, but breakdowns are random. The average number of parts in the queue at the machine is about 1.13. The range is from 1.04 to 1.32 parts. For some machines and processing in manufacturing environments, Case F is the realistic situation: Processing times are constant, and arrivals are regulated—that is, are also constant. The reader is left to consider the inaccuracies that would result from making faulty assumptions regarding the nature of time to failure and time to repair.

In conclusion, there can be significant differences between the estimated average numbers in a queue, based on the treatment of randomness. The results using the correct treatment of randomness can be far different from those using alternatives. Often, one is tempted by the unavailability of detailed data and the availability of averages to want to use average time to failure as if it were a constant. Example 13.1 illustrates the dangers of inappropriate assumptions. Both the appropriate technique to use and the appropriate statistical distribution depend on the available data and on the situation at hand.

As discussed by Williams [1994], the accurate treatment of downtimes is essential for achieving valid models of manufacturing systems. Some of the essential ingredients are the following:

- avoidance of oversimplified and inaccurate assumptions;
- careful collection of downtime data;
- accurate representation of time to failure and time to repair by statistical distributions;
- accurate modeling of system logic when a downtime occurs, in terms both of the repair-time logic and of what happens to the part currently processing.

13.3.2 Trace-driven Models

Consider a model of a distribution center that receives customer orders that must be processed and shipped in one day. One modeling question is how to represent the day's set of orders. A typical order will contain one or more line items, and each line item can have a quantity of one or more pieces. For example, when you buy a new stereo, you might purchase an amplifier, a tuner, and a CD player (all separate line items, each having a quantity of one piece), and 4 identical speakers (another line item with a quantity of 4 pieces). The overall order profile can have a major impact on the performance of a particular system design. A system designed to handle large orders going to a small number of customers might not perform well if order profiles shift toward a larger number of customers (or larger number of separate shipments) with one or two items per order.

One approach is to characterize the order profile by using a discrete statistical distribution for each variable in an order:

1. the number of line items
2. for each line item, the number of pieces.

If these two variables are statistically independent, then this approach might provide a valid model of the order profile. For many applications, however, these two variables may be highly correlated in ways that could be difficult to characterize statistically. For example, an apparel and shoe company has six large customers (the large department stores and discount chains), representing 50% of sales volume, which typically order dozens or hundreds of line items and large quantities of many of the items. At the opposite pole, on any given day approximately 50% of the orders are for one or two pairs of shoes (just-in-time with a vengeance!). For this company, the number of line items in an order is highly positively correlated with the quantity ordered; that is, large orders with a large number of line items also usually have large quantities of many of the line items. And small orders with only a few line items typically order small quantities of each item.

What would happen if the two variables, number of line items and quantity per line item, were modeled by independent statistical distributions? When an order began processing, the model would make two random uncorrelated draws, which could result in order profiles quite different from those found in practice. Such an erroneous assumption could result, for example, in far too large a proportion of orders having one or two line items with large unrealistic quantities.

Another common but more serious error is to assume that there is an average order and to simulate only the number of orders in a day with each being the typical order. In the author's experience, analyses of many order profiles has shown (1) that there is no such thing as a typical order and (2) that there is no such thing as a typical order profile.

An alternative approach, and one that has proven successful in many studies, is for the company to provide the actual orders for a sample of days over the previous year. Usually, it is desirable to simulate peak days. A model driven by actual historical data is called a trace-driven model.

A trace-driven model eliminates all possibility of error due to ignoring or misestimating correlations in the data. One apparent limitation could be a customer's desire, at times, to be able to simulate hypothesized changes to the order profile, such as a higher proportion of smaller orders in terms of both line items and quantities. In practice, this limitation can be removed by adding "dials" to the order-profile portion of the model, so that a simulation analyst can "dial up" more or less of certain characteristics, as desired. One approach is to treat the day's orders as a statistical population from which the model draws samples in a random fashion. This approach makes it easy to change overall order volume without modifying the profile. A second related approach would be to subdivide a day's orders into subgroups based on number of line items, quantities or other numeric parameters, and then sample in a specified proportion from each subgroup. By changing the proportion of each subgroup, different order profiles can be "dialed up" and fed into the model. A third approach is to use factors to adjust the number of daily orders, the number of line items, and/or the quantities. In practice one of these approaches might be as accurate as can be expected for hypothesized future order profiles and might provide a cost effective and reasonably accurate model, especially for testing the robustness of a system design for assumed changes in order characteristics.

Other examples of trace-driven models include the following:

- orders to a custom job shop, using actual historical orders;
- product mix and quantities, and production sequencing, for an assembly line making 100 styles and sizes of hot-water heaters;
- time to failure and downtime, using actual maintenance records;
- Truck arrival times to a warehouse, using gate records.

Whether to make an input variable trace-driven or to characterize it as a statistical distribution depends on a number of issues, including the nature of the variable itself, whether it is correlated with or independent of other variables, the availability of accurate data, and the questions being addressed.

13.4 CASE STUDIES OF THE SIMULATION OF MANUFACTURING AND MATERIAL HANDLING SYSTEMS

The *Winter Simulation Conference Proceedings*, IIE Magazine, Modern Material Handling and other periodicals are excellent sources of information for short cases in the simulation of manufacturing and material-handling systems.

An abstract of some of the papers from past *Winter Simulation Conference Proceedings* will provide some insight into the types of problems that can be addressed by simulation. These abstracts have been paraphrased and shortened where appropriate; our goal is to provide an indication of the breadth of real-world applications of simulation.

Session: Semiconductor Wafer Manufacturing
Paper: Modeling and Simulation of Material Handling for Semiconductor Wafer Manufacturing
Authors: Neal G. Pierce and Richard Stafford
Abstract: This paper presents the results of a design study to analyze the interbay material-handling systems for semiconductor wafer manufacturing. The authors developed discrete-event simulation models of the performance of conventional cleanroom material handling including manual and automated systems. The components of a conventional cleanroom material-handling system include an overhead monorail system for interbay (bay-to-bay) transport, work-in-process stockers for lot storage, and manual systems for intrabay movement. The authors constructed models and experiments that assisted with analyzing cleanroom material-handling issues such as designing conventional automated material-handling systems and specifying requirements for transport vehicles.

Session: Simulation in Aerospace Manufacturing
Paper: Modeling Aircraft Assembly Operations
Authors: Harold A. Scott
Abstract: A simulation model is used to aid in the understanding of complex interactions of aircraft assembly operations. Simulation helps to identify the effects of resource constraints on dynamic process capacity and cycle time. To analyze these effects, the model must capture job and crew interactions at the control code level. This paper explores five aspects of developing simulation models to analyze crew operations on aircraft assembly lines:

Representing job precedence relationships
Simulating crew members with different skill and job proficiency levels
Reallocating crew members to assist ongoing jobs
Depicting shifts and overtime
Modeling spatial constraints and crew movements in the production area.

Session: Control of Manufacturing Systems
Paper: Discrete Event Simulation for Shop Floor Control
Authors: J. S. Smith, R. A. Wysk, D. T. Sturrock, S. E. Ramaswamy, G. D. Smith, S. B. Joshi

Abstract: This paper describes an application of simulation to shop floor control of a flexible manufacturing system. The simulation is used not only as an analysis and evaluation tool, but also as a "task generator" for the specification of shop floor control tasks. Using this approach, the effort applied to the development of control logic in the simulation is not duplicated in the development of the control system. Instead the same control logic is used for the control system as was used for the simulation. Additionally, since the simulation implements the control, it provides very high fidelity performance predictions. The paper describes implementation experience in two flexible manufacturing laboratories.

Session: Flexible Manufacturing
Paper: Developing and Analyzing Flexible Cell Systems Using Simulation
Authors: Edward F. Watson and Randall P. Sadowski
Abstract: This paper develops and evaluates flexible cell alternatives to support an agile production environment at a mid-sized manufacturer of industrial equipment. Three work cell alternatives were developed based on traditional flow analysis studies, past experience, and common sense. The simulation model allowed the analyst to evaluate each cell alternative under current conditions as well as anticipated future conditions that included changes to product demand, product mix, and process technology.

Session: Modeling of Production Systems
Paper: Inventory Cost Model for Just-in-Time Production
Authors: Mahesh Mathur
Abstract: This paper presents a simulation model used to compare setup and inventory carrying costs with varying lot sizes. While reduction of lot sizes is a necessary step towards implementation of Just-in-Time (JIT) in a job shop environment, a careful cost study is required to determine the optimum lot size under the present set-up conditions. The simulation model graphically displays the fluctuation of carrying costs and accumulation of set-up costs on a time scale in a dynamic manner. The decision of the optimum lot size can then be based on realistic cost figures.

Session: Analysis of Manufacturing Systems
Paper: Modeling Strain of Manual Work in Manufacturing Systems
Authors: I. Ehrhardt, H. Herper, and H. Gebhardt
Abstract: This paper describes a simulation model that considers manual operations for increasing the effectiveness of planning logistic systems. Even though there is ever increasing automation, there are vital tasks in production and logistics that are still assigned to humans. Present simulation modeling efforts rarely concentrate on the manual activities assigned to humans.

Session: Manufacturing Case Studies
Paper: Simulation Modeling for Quality and Productivity in Steel Cord Manufacturing
Authors: C. H. Turkseven and G. Ertek
Abstract: The paper describes the application of simulation modeling to estimate and improve quality and productivity performance of a steel cord manufacturing system. It focuses on wire fractures, which can be an important source of system disruption.

Session: Manufacturing Analysis and Control
Paper: Shared Resource Capacity Analysis in Biotech Manufacturing

Author: P. V. Saraph
Abstract: This paper discusses an application of simulation in analyzing the capacity needs of a shared resource, the Blast Freezer, at one of the Bayer Corporation's manufacturing facilities. The simulation model was used to analyze the workload patterns, run different workload scenarios, taking into consideration uncertainty and variability, and provide recommendations on a capacity increase plan. This analysis also demonstrated the benefits of certain operational scheduling policies. The analysis outcome was used to determine capital investments for 2002.

Session: Manufacturing Analysis and Control
Paper: Behavior of an Order Release Mechanism in a Make-to-Order Manufacturing System with Selected Order Acceptance
Authors: A. Nandi and P. Rogers
Abstract: The authors used a simulation model to evaluate a controversial policy, namely, holding orders in a pre-shop pool prior to their release to the factory floor. In a make-to-order manufacturing system, if capacity is fixed and exogenous due dates are inflexible, having orders wait in a pre-shop pool may cause the overall due date performance of the system to deteriorate. The model was used to evaluate an alternative approach, the selective rejection of orders for dealing with surges in demand while maintaining acceptable due date performance.

13.5 MANUFACTURING EXAMPLE: AN ASSEMBLY-LINE SIMULATION

This section describes a model of a production line for the final assembly of "gizmos". It then focuses on how simulation can be used to analyze system performance.

13.5.1 System Description and Model Assumptions

At a manufacturing facility, an engineering team has designed a new production line for the final assembly of gizmos. Before making the investment to install the new system, some team members propose using simulation to analyze the system's performance, specifically to predict system throughput (gizmos per 8-hour shift, on the average). In addition, the engineers desire to evaluate potential improvements to the designed system. One such potential improvement is adding buffer space for holding work-in-process (WIP) between adjacent workstations.

The team decides to develop a simulation model and conduct an analysis. The team's primary objective is to predict throughput (completed gizmos per shift on the average) for the given system design and to evaluate whether it meets the desired throughput. In addition, should throughput be less than expected, the team wants to use the model to help in identifying bottlenecks, gaining insight into the system's dynamic behavior and evaluating potential design improvements.

The proposed production line has six workstations and a special rack for WIP storage between adjacent stations. There are four manual stations, each having its own operator, and two automated stations, which share a single operator. The six stations perform production tasks in the following sequence:

Station 1: initial manual station begins final assembly of a new gizmo
Station 2: manual assembly station
Station 3: manual assembly station
Station 4: automatic assembly station
Station 5: automatic testing station
Station 6: manual packing station

At each manual station, an operator loads a gizmo onto a workbench, performs some tasks, and on completion unloads the gizmo and places it into the WIP storage for the next workstation. The operator takes 10 seconds and 5 seconds for the loading and unloading tasks, respectively.

The WIP storage racks between each pair of adjacent stations have limited capacity. If a station completes its tasks on a gizmo but the downstream rack is full, the gizmo must remain in the station, blocking any further work. In the initial design, the WIP storage racks have the capacities shown in Table 13.2. (By assumption, the WIP storage preceding Station 1 is always kept full at 4 units; since it is assumed to always be full, its specific capacity plays no role.) The system design with capacities given in Table 13.2 is called the Baseline configuration.

From time to time, a tool will fail, causing unscheduled downtime or unexpected extra work at a manual or automated station. In addition, all operators are scheduled to take a 30-minute lunch break at the same time. Work is interrupted and resumes where it left off after lunch. This interrupt/resume rule applies to operator tasks including assembly work, parts resupply, and repairs during a downtime.

At the automatic stations, a machine performs an assembly or testing task. The automatic stations might have unscheduled (random) downtimes, but they continue to operate during the operator's lunch break. One operator services both machines to load and unload gizmos (10 seconds and 5 seconds, respectively). After being loaded, a machine processes the gizmo without further operator intervention unless a downtime occurs. At all stations, the operator performs repairs as needed whenever the station experiences a downtime.

Table 13.3 gives the total assembly time and parts resupply times for each station, plus the number of parts in a batch. The assembly time for the manual stations is assumed to vary by plus/minus 2 seconds (uniformly distributed) from the times given in Table 13.3. Parts resupply time does not occur for each gizmo, but rather after a batch of parts has been assembled onto the gizmo. The machines at stations 4 and 5 do not consume parts.

Each station is subject to unscheduled (random) downtime. Manual stations 1–3 have tool failures or other unexpected problems. The automatic stations occasionally jam or have some other problem that requires the assigned operator to fix it. Station 6 (packing) is not subject to these downtimes. Table 13.4

Table 13.2 Capacity of WIP Storage Buffers for Baseline Configuration

Rack Before Station	1	2	3	4	5	6
Buffer Capacity	4	2	2	2	1	2

Table 13.3 Assembly and Parts Resupply Times

Station	*Assembly per Gizmo (Seconds)*	*Part Number*	*Parts Resupply Time (seconds per Batch)*	*No. of Parts per Batch*
1	40	A	10	15
		B	15	10
2	38	C	20	8
		D	15	14
3	38	E	30	25
4	35			
5	35			
6	40	F[a]	30	32

[a]: At station 6, the part number (F) represents the shipping containers.

Table 13.4 Assumptions and Data for Unscheduled Downtimes

Station	*TTF*	*MTTF (Minutes)*	*TTR*	*MTTR (Minutes)*	*+/–*	*Expected Availability*
1	Exponential	36.0	Uniform	4.0	1.0	90%
2	Exponential	4.5	Uniform	0.5	0.1	90%
3	Exponential	27.0	Uniform	3.0	1.0	90%
4	Exponential	9.0	Uniform	1.0	0.5	90%
5	Exponential	18.0	Uniform	2.0	1.0	90%

shows time to failure (TTF) and time to repair (TTR) distributional assumptions and the assumed mean time to failure (MTTF), mean time to repair (MTTR) and spread (+/–) of repair times. For example, at Station 1, repair time is uniformly distributed with mean 4.0 minutes plus or minus 1.0 minutes—that is, uniformly distributed between 3.0 and 5.0 minutes. Failure can only occur when an operator or machine is working; hence, TTF is modeled by measuring only busy or processing time until a failure occurs.

The primary model output or response is average throughput during the assumed 7.5 working hours per 8-hour shift. The model also measures detailed station utilization, including busy or processing time, idle or starved time (no parts ready for processing), blocked time (part cannot leave station, because downstream WIP buffer is full), unscheduled downtime, and time waiting for an operator.

Station starvation occurs when the operator and station are ready to work on the next gizmo, the just-completed gizmo leaves the station, but upstream conditions cause no gizmo to be ready for this production step. In short, the upstream WIP buffer is empty.

Station blockage occurs when a station completes all tasks on a gizmo, but cannot release the part because the downstream WIP buffer is full. For both starvation and blockage, production time is lost at the given station and cannot be made up.

When an operator services more than one station, as does the operator servicing Stations 4 and 5, it is possible for both stations to need the operator at the same time. This could cause additional delay at the station and is measured by a "wait for operator" state. Blockage, starvation, and wait-for-operator at each station will be measured in order to help explain any throughput shortfall, should it occur, and to assist in identifying potential system improvements.

13.5.2 Presimulation Analysis

A presimulation analysis, taking into account the average station cycle time as well as expected station availability (90%), indicates that each station, if unhindered, can achieve the desired throughput. This initial analysis is carried out as described in this section.

From the assumed downtime data, the team was able to estimate expected station availability, under the (ideal) assumption of no interaction between stations. The expected availability shows each station's individual availability during working (nonlunch, nonbreak) hours, assuming that the operator can always place a completed gizmo into the downstream rack storage and the next gizmo is ready to begin work at the station. Expected availability is computed by MTTF/(MTTF + MTTR), or expected busy time during a downtime "cycle" divided by the length of a downtime cycle (a busy cycle plus a repair cycle) and is given in Table 13.4. This calculation ignores certain aspects of the problem, including the parts resupply times and any delay caused by having only one operator to service both Stations 4 and 5.

The design goal for the modeled system is 390 finished gizmos per 8-hour shift. After taking lunch into account, each shift has up to 7.5 hours of available work time. With unscheduled (random) downtime expected to be 10% of available time, this further reduces working time to 0.90×7.5 hours = 6.75 hours. This implies

Table 13.5 Estimated Total Cycle Time at Each Station

Station	*Formula to Estimate Cycle Time (Seconds)*	*Estimate (Seconds)*
1	10 + 40 + 5 + 10/15 + 15/10	57.2
2	10 + 38 +5 + 20/8 + 15/14	56.6
3	10 + 38 + 5 + 30/25	54.2
4	10 + 35 + 5	50.0
5	10 + 35 + 5	50.0
6	10 + 40 + 5 + 30/32	55.9

that the station with the slowest total cycle time must be able to produce 390 gizmos in the available 6.75 hours. Therefore the total cycle time per gizmo at each station must not exceed 6.75 hours/390 = 62.3 seconds.

Now, total cycle time consists of assembly, testing or packing time, and parts resupply time (as given in Table 13.3), plus gizmo loading time of 10 seconds and unload time of 5 seconds. Parts resupply is not taken on every gizmo, but rather after a given number of gizmos corresponding to using all parts in a given batch of parts. For example, using the values in Table 13.3 for Station 1, parts resupply will take 10 seconds every 15 gizmos for Part A, plus 15 seconds every 10 gizmos for Part B, for a total time on the average of 10/15 + 15/10 seconds per gizmo.

Using this information, the (minimum) total cycle time for each station is estimated in Table 13.5. These presimulation estimates indicate, first, that each theoretical cycle time is well below the requirement of 62.3 seconds. Secondly, they indicate that Stations 1 and 2 are potential bottlenecks, if there are any.

As the simulation analysis will later show, Station 1 experiences blockage due to Station 2 downtime, and Station 2 occasionally experiences starvation due to downtime at Station 1 and blockage due to downtime at Station 3. These blockage and starvation conditions reduce the available work time below the calculated 90%; hence, for the Baseline Configuration, they reduce the design throughput well below the desired value, 390 gizmos per shift. In summary, a presimulation analysis, although valuable, at best can provide a rough estimate of system performance. As the simulation will show, ignoring blockage and starvation gives an overly optimistic estimate of system throughput.

13.5.3 Simulation Model and Analysis of the Designed System

Using the simulation model, the first experiment was conducted to estimate system performance of the system as designed. The simulation analyst on the team made 10 replications of the model, each having a 2-hour warm-up or initialization followed by a 5-day simulation (each day being 24 hours). A 95% confidence interval was computed for mean throughput per shift:

95% CI for mean throughput: (364.5, 366.8), *or* 365.7 ± 1.14.

With 95% confidence, the model predicts that mean (or long-run average) throughput will be between 364.5 and 366.8 gizmos per 8-hour shift with the system as designed. This is well below the design throughput, 390 gizmos per shift.

The team decided to conduct further analyses to identify possible bottlenecks and potential areas of improvement.

13.5.4 Analysis of Station Utilization

At this point, the team desired to have some explanation of the shortfall in throughput. They suspected that perhaps it had to do with the small WIP buffer capacity and the resulting blockage and starvation. The same

Table 13.6 Detailed Station Utilization for Baseline Configuration

Station	*% Down*	*% Blocked*	*% Starved*	*% Wait for Operator*
1	(8.8,9.6)	(11.4,12.5)	(0.0,0.0)	(0.0,0.0)
2	(8.2,8.4)	(8.0,8.8)	(4.9,5.6)	(0.0,0.0)
3	(7.9,8.6)	(9.9,10.4)	(6.1,6.9)	(0.0,0.0)
4	(8.9,9.6)	(2.0,2.8)	(7.5,8.2)	(13.1,14.4)
5	(8.3,9.0)	(0.0,0.2)	(19.4,20.4)	(3.9,4.7)

model was used to estimate detailed workstation utilization in hopes that it would provide an explanation of throughput shortfall. Table 13.6 contains 95% confidence-interval estimates for the first five workstations for percent of time down, blocked, starved, and waiting for an operator. (Waiting for operator affects only stations 4 and 5, as these two stations share one operator. The other stations have a dedicated operator. In addition to the utilization statistics in Table 13.6, the operators have a 30-minute lunch per 8-hour shift, representing 6.25% of available time.)

From the results in Table 13.6, it appears that blockage and starvation explain some portion of the shortfall in throughput. In addition, another possible explanation surfaces: Station 4 experiences a significant time waiting for the single operator that services stations 4 and 5. This delay at Station 4 could result in a full WIP buffer, which in turn would help explain the blockage at Station 3 preceding it. Percent of time blocked is higher than percent starved for Stations 1 to 3, so it appears that downstream delays could be a significant bottleneck.

The team proposed some possible system improvements:

1. having two operators to service Stations 4 and 5 (instead of the currently proposed one operator);
2. increasing the capacity of some of the WIP buffers;
3. a combination of both.

The expense of additional WIP storage space induced the team to desire to keep total buffer space as small as possible, and to require an additional operator only if absolutely necessary, while achieving the design goal of 390 gizmos per shift.

13.5.5 Analysis of Potential System Improvements

To evaluate the addition of an operator and larger WIP buffers, the model was revised appropriately to allow these changes, and a new analysis was conducted. In this analysis, the capacity of each WIP buffer for Stations 2 – 6 was allowed to increase by one unit above the Baseline value given in Table 13.2. In addition, the effect of a second operator at Stations 4 and 5 is considered. These possibilities result in a total of 64 scenarios or model configurations. (Why?) Making 10 replications per scenario results in a total of 640 simulation runs.

To facilitate the analysis, the team decided to use the Common Random Number technique discussed in Section 12.1.3. To implement it with proper synchronization, each source of random variability was identified and assigned a dedicated random-number stream. In this model, processing time, TTF, and TTR are modeled by statistical distributions at each of the six workstations. Therefore, a total of 18 random-number streams were defined, with 3 used at each workstation. In this way, in each set of runs, each workstation experienced the same workload and random downtimes no matter which configuration was being simulated. For a given number of replications the CRN technique, also known as correlated sampling, is expected to give shorter confidence intervals for differences in system performance.

The model configurations with the most improvement in system throughput, compared with the Baseline configuration, are shown in Table 13.7. These configurations were chosen for further evaluation because

Table 13.7 Improvement in System Throughput for Alternative Configurations

Number of Operators Stations 4 & 5	*Buffer Capacities*						*Increase in Mean Throughput per Shift (Compared to Baseline)*		
	Buffer 2	*Buffer 3*	*Buffer 4*	*Buffer 5*	*Buffer 6*	*Total*	*Ave. Diff.*	*CI Low*	*CI High*
2	3	3	3	2	2	13	31.7	30.3	33.1
2	3	3	3	2	3	14	31.7	30.4	33.0
2	3	3	2	2	3	13	30.0	28.6	31.3
2	3	3	3	1	3	13	29.8	28.6	31.0
2	3	3	2	2	2	12	29.7	28.1	31.3
2	3	3	3	1	2	12	29.5	28.1	31.0
2	3	3	2	1	3	12	26.6	25.4	27.9
2	2	3	3	2	2	12	26.6	25.1	28.1
2	2	3	3	2	3	13	26.6	25.0	28.1
2	3	2	3	2	3	13	26.5	25.0	28.0
2	3	2	3	2	2	12	26.4	25.3	27.5
2	3	3	2	1	2	11	26.3	25.1	27.5

each shows a potential improvement in throughput of approximately 25 units or more—that is, the lower end of the 95% confidence interval is 25 or higher. The values shown for "Ave Diff" represent the increase in throughput compared to the Baseline configuration. Recall that the Baseline throughput was previously estimated, with 95% confidence, to be in the interval (364.5, 366.8). Being conservative, the engineering team would like to see an improvement of 390 − 364.5 = 25.5 gizmos per shift. The top six configurations in Table 13.7 have a lower confidence interval larger than 25.5 and hence are likely candidates for achieving the desired throughput. Interpreted statistically: The lower end of the confidence interval is larger than 25.5, so the results yield a 95% confidence that mean throughput will increase by 25.5 or more in the top six configurations listed in Table 13.7.

Note that all the most improved configurations include two operators at Stations 4 and 5. The simulation results for configurations with one operator (not shown here) indicate that a 390 throughput cannot be achieved with one operator, at least not with the buffer sizes considered.

Some configurations can be ruled out because a less expensive option achieves a similar throughput. Consider, for example, the first two configurations in Table 13.7. They are identical except for Buffer 6 capacity. Since WIP buffer capacity is expensive, the smaller total buffer capacity will be the less expensive option. Clearly, there is no need to expand from 2 to 3 units at Buffer 6. The "Total" column can assist in quickly ruling out configurations that do no better than a similar one with smaller total buffer capacity.

The model configuration that increases throughput by 25.5 or better and has the smallest total buffer capacity is the fifth one in Table 13.7, with capacities of (3,3,2,2,2) for Buffers 2 to 6, respectively. On these considerations, this system design becomes the team's top candidate for further evaluation. The next step (not included here) would be to conduct a financial analysis of each alternative configuration.

13.5.6 Concluding Words: The Gizmo Assembly-Line Simulation

Real-life examples similar to this example model include assembly lines for automotive parts and automobile bodies, automotive pollution-control assemblies, consumer items such as washing machines, ranges, and

dishwashers, and any number of other assembly operations with a straight flow and limited buffer space between workstations. Similar models and analyses may also apply to a job shop with multiple products, variable routing, and limited work-in-process storage.

13.6 SUMMARY

This chapter introduced some of the ideas and concepts most relevant to manufacturing and material handling simulation. Some of the key points are the importance of modeling downtimes accurately, the advantages of trace-driven simulations with respect to some of the inputs, and the need in some models for accurate modeling of material-handling equipment and the control software.

REFERENCES

BANKS, J. [1994], "Software for Simulation," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 26–33.

CLARK, G. M. [1994], "Introduction to Manufacturing Applications," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 15–21.

EHRHARDT, I., H. HERPER, AND H. GEBHARDT [1994], "Modelling Strain of Manual Work in Manufacturing Systems," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 1044–1049.

LAW, A. M. AND W. D. KELTON [2000], *Simulation Modeling and Analysis*, 3d ed., McGraw–Hill, New York.

NANDI, A., AND P. ROGERS [2003], "Behavior of an Order Release Mechanism in a Make-to-Order Manufacturing System with Selected Order Acceptance," in *2003 Winter Simulation Conference Proceedings*, eds. S. E. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, Association for Computing Machinery, New York, pp. 1251–1259.

PIERCE, N. G., AND R. STAFFORD [1994], "Modeling and Simulation of Material Handling for Semiconductor Wafer Fabrication," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 900–906.

SARAPH, P. V. [2003], "Shared Resource Capacity Analysis in Biotech Manufacturing," in *2003 Winter Simulation Conference Proceedings*, eds. S. E. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, Association for Computing Machinery, New York, pp. 1247–1250.

SCOTT, H. A. [1994], "Modeling Aircraft Assembly Operations," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 920–927.

SMITH, J. S., *et al.* [1994], "Discrete Event Simulation for Shop Floor Control," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 962–969.

TURKSEVEN, C. H., AND G. ERTEK [2003], "Simulation Modeling for Quality and Productivity in Steel Cord Manufacturing," in *2003 Winter Simulation Conference Proceedings*, eds. S. E. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, Association for Computing Machinery, New York, pp. 1225–1229.

WATSON, E. F., AND R. P. SADOWSKI [1994], "Developing and Analyzing Flexible Cell Systems Using Simulation," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 978–985.

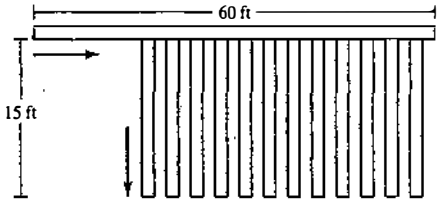
WILLIAMS, E. J. [1994], "Downtime Data—Its Collection, Analysis, and Importance," in *1994 Winter Simulation Conference Proceedings*, eds. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Association for Computing Machinery, New York, pp. 1040–1043.

EXERCISES

Instructions to the student: Many of the following exercises contain material-handling equipment such as conveyors and vehicles. The student is expected to use any simulation language or simulator that supports modeling conveyors and vehicles at a high level.

Some of the following exercises use the uniform, exponential, normal, or triangular distributions. Virtually all simulation languages and simulators support these, plus other distributions. The use of the first three distributions was explained in the note to the exercises in Chapter 4; the use of the triangular is explained in the exercise that requires it. For reference, the properties of these distributions, plus others used in simulation, are given in Chapter 5, and random-variate generation is covered in Chapter 8.

1. A case sortation system consists of one infeed conveyor and 12 sortation lanes, as shown in the following schematic (not to scale):



Cases enter the system from the left at a rate of 50 per minute at random times. All cases are 18 by 12 inches and travel along the 18 inch dimension. The incoming mainline conveyor is 20 inches wide and 60 feet in length (as shown). The sortation lanes are numbered 1 to 12 from left to right, and are 18 inches wide and 15 feet in length, with 2 feet of spacing between adjacent lanes. (Estimate any other dimensions that are needed.) The infeed conveyor runs at 180 feet/minute, the sortation lanes at 90 feet/minute. All conveyor sections are accumulating, but, upon entrance at the left, incoming cases are at least 2 feet apart from leading edge to leading edge. On the sortation lanes, the cases accumulate with no gap between them.

Incoming cases are distributed to the 12 lanes in the following proportions:

1	6%	7	11%
2	6%	8	6%
3	5%	9	5%
4	24%	10	5%
5	15%	11	3%
6	14%	12	0%

The 12th lane is an overflow lane; it is used only if one of the other lanes fill and a divert is not possible.

At the end of the sortation lanes, there is a group of operators who scan each case with a bar-code scanner, apply a label and then place it on a pallet. Operators move from lane to lane as necessary to avoid allowing a lane to fill. There is one pallet per lane, each holding 40 cases. When a pallet is full, assume a new empty one is immediately available. If a lane fills to 10 cases and another case arrives at the divert point, this last case continues to move down the 60-foot mainline conveyor and is diverted into lane 12, the overflow lane.

Assume that one operator can handle 8.5 cases per minute, on the average. Ignore walking time and assignment of an operator to a particular lane; in other words, assume the operators work as a group uniformly spread over all 12 lanes.

(a) Set up an experiment that varies the number of operators and addresses the question: How many operators are needed? The objective is to have the minimum number of operators but also to avoid overflow.

(b) For each experiment in part (a), report the following output statistics:

Operator utilization
Total number of cases palletized
Number of cases palletized by lane
Number of cases to the overflow lane

(c) For each experiment in part a, verify that all cases are being palletized. In other words, verify that the system can handle 50 cases per minute, or explain why it cannot.

2. Redo Exercise 1 to a greater level of detail by modeling operator walking time and operator assignment to lanes. Assume that operators walk at 200 feet per minute and that the walking distance from one lane to the next is 5 feet. Handling time per case is now assumed to be 7.5 cases per minute. Devise a set of rules that can be used by operators for lane changing. (For example, change lanes to that lane with the greatest number of cases only when the current lane is empty or the other lane reaches a certain level.) Assume that each operator is assigned to a certain number of adjacent lanes and handles only those lanes. However, if necessary, two operators (but no more) may be assigned to one lane—that is, operator assignments may overlap.

(a) If your lane-changing rule has any numeric parameters, experiment to find the best settings. Under these circumstances, how many operators are needed? What is the average operator utilization?

(b) Does a model that has more detail, as does Exercise 2a when compared to Exercise 1, always have greater accuracy? How about this particular model? Compare the results of Exercise 2a to the results for Exercise 1. Are the same or different conclusions drawn?

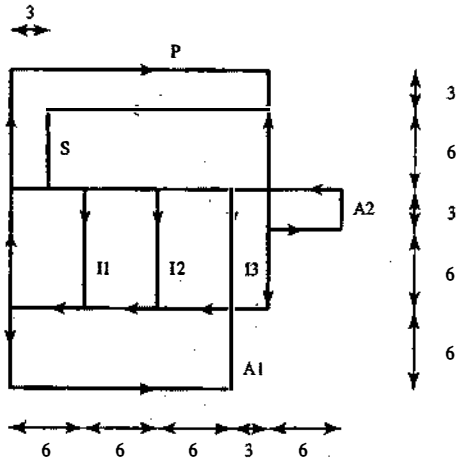
(c) Devise a second lane-changing rule. Compare results between the two rules. Compare total walking time or percent of time spent walking between the two rules.
Suggestion: A lane-changing rule could have one or two "triggers". A one-trigger rule might state that, if a lane reached a certain level, the operator moved to that lane. (Without modification, such a rule could lead to excessive operator movement, if two lanes had about the same number of cases near the trigger level.) A two-trigger rule might state that, if a lane reached a certain level and the operator's current lane became empty, then change to the new lane; but if a lane reaches a specified higher "critical" level, then the operator immediately changes lanes.

(d) Compare your results with those of other students who may have used a different lane-changing rule.

3. Parts carried by the AGV system arrive through three intersections are

Intersection	*Interarrival Time (Minutes)*
I1	10 ± 4
I2	8 ± 2
I3	20 ± 6

The parts are to be assembled in any one of the assembly stations A1 or A2. The assembly time is



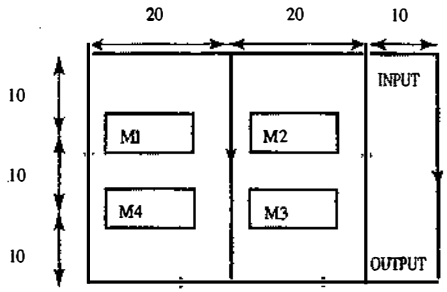
7 ± 2 minutes. After assembly, parts are sent to the output station P. If both A1 and A2 are free, parts have an equal probability of going to either A1 or A2. AGV is required to take the arriving part to assembly station and assembled part to output station. Once AGV becomes free, it responds to any waiting call, otherwise it is sent to staging area S. All links are unidirectional and the distances are shown in meters. The AGV speed is 40 meters per minute. Delay the start of the assembly operations for 30 minutes after parts start arriving to allow a buildup of parts. Simulate the system for 10,000 minutes. Determine the number of AGVs required to ensure that there is always a part available for the assembly operations.

4. Redo the simulation with the assumption that the assembly times are different in A1 and A2 as

Assembly Station	*Assembly Time (Minutes)*
A1	9 ± 2
A2	7 ± 2

Hence if both A1 and A2 are free, the part is taken to the assembly station A2.

5. In a machine shop, there are four machines M1, M2, M3, and M4. They are identical in all respects and served by AGVs. Parts arrive with interarrival time following exponential with a mean of 5 minutes. Machines do not have any buffer space. So an arriving part at the input area must first gain access to a free machine before it can be moved to the machine. When a machine finishes an operation, an AGV is requested and the machine is to be made free only after the part has been picked up by the AGV. Processing time follows normal with a mean of 8 minutes and a standard deviation of 2 minutes. It takes 30 seconds to load and unload the parts. AGV takes the finished parts to the output station and the AGV is free to respond to other requests, or is sent to the input area that serves as a staging area. The AGVs move at a speed of 25 meters per minute. The dimensions shown are in meters, and the intersections are 0 meter in length. Simulate this system for 2,500 minutes. Change the number of AGVs and analyze the impact on parts waiting time.



6. Reconsider Exercise 5. Assume that two types of parts are arriving and the parts are to be processed in more than one machine. Parts arrive with interarrival time following exponential with a mean of 5 minutes. The sequence of operation and the percentage of part types are

Part Type	*Percentage*	*Sequence*
A	60	M1, M2, M4
B	40	M2, M3

Process time at the machines are

Machine	*Process Time*
M1	$N(8,2)$
M2	4 ± 2
M3	$N(8,1)$
M4	9 ± 2

Simulate this system for 2,500 minutes. Change the number of AGVs and analyze the impact on parts waiting time.

7. Develop a model for Example 13.1 and attempt to reproduce qualitatively the results found in the text regarding different assumptions for simulating downtimes. Do not attempt to get exactly the same numerical results, but rather to show the same qualitative results.

(a) Do your models support the conclusions discussed in the text? Provide a discussion and conclusions.

(b) Make a plot of the number of entities in the queue versus time. Can you tell when failures occurred? After a repair, about how long does it take for the queue to get back to "normal"?

8. In Example 13.1, the failures occurred at low frequency compared with the processing time of an entity. Time to failure was 1000 minutes, and interarrival time was 10 minutes, implying that few entities would experience a failure. But, when an entity did experience a failure (of 50 minutes, on average), it was several times larger than the processing time of 7.5 minutes.

Redo the model for Example 13.1, assuming high-frequency failures. Specifically, assume that the time to failure is exponentially distributed, with mean 2 minutes, and the time to repair is exponentially

distributed, with mean 0.1 minute or 6 seconds. As compared with the low-frequency case, entities will tend to experience a number of short downtimes.

For low-frequency versus high-frequency downtimes, compare the average number of downtimes experienced per entity, the average duration of downtime experienced, the average time to complete service (including downtime, if any), and the percent of time down.

Note that the percentage of time the machine is down for repair should be the same in both cases:

$$50/(1000 + 50) = 4.76\%$$
$$6 \text{ sec}/(2\text{min}+6 \text{ sec}) = 4.76\%$$

Verify percentage downtime from the simulation results. Are the results identical? ... close? Should they be identical, or just close? As the simulation run-length increases, what should happen to percentage of time down?

With high-frequency failures, do you come to the same conclusions as were drawn in the text regarding the different ways to simulate downtimes? Make recommendations regarding how to model low-frequency versus high-frequency failures.

9. Redo Exercise 11 (based on Example 13.1), but with one change: When an entity experiences a downtime, it must be reprocessed from the beginning. If service time is random, take a new draw from the assumed distribution. If service time is constant, it starts over again. How does this assumption affect the results?

10. Redo Exercise 11 (based on Example 13.1), but with one change: When an entity experiences a downtime, it is scrapped. How does scrapping entities on failure affect the results in the low-frequency and in the high-frequency situations? What are your recommendations regarding the handling of low-versus high-frequency downtimes when parts are scrapped?

11. Sheets of metal pass sequentially through 4 presses: shear, punch, form, and bend. Each machine is subject to downtime and die change. The parameters for each machine are as follows:

Press	*Process Rate (per min.)*	*Time to Failure (min.)*	*Time to Repair (min.)*	*No. of Sheets to a Die Change (no. sheets)*	*Time to Change Die (min.)*
Shear	4.5	100	8	500	25
Punch	5.5	90	10	400	25
Form	3.8	180	9	750	25
Bend	3.2	240	20	600	25

Note that processing time is given as a rate—for example, the shear press works at a rate of 4.5 sheets per minute. Assume that processing time is constant. The automated equipment makes the time to change a die fairly constant, so it is assumed to be always 25 minutes. Die changes occur between stamping of two sheets after the number shown in the table have gone through a machine. Time to failure is assumed to be exponentially distributed, with the mean given in the table. Time to repair is assumed to be uniformly distributed, with the mean taken from the table and a half-width of 5 minutes. When a failure occurs, 20% of the sheets are scrapped. The remaining 80% are reprocessed at the failed machine after the repair.

Assume that an unlimited supply of material is available in front of the shear press, which processes one sheet after the next as long as there is space available between itself and the next machine, the punch press.

In general, one machine processes one sheet after another continuously, stopping only for a downtime, for a die change, or because the available buffer space between itself and the next machine becomes full. Assume that sheets are taken away after bending at the bend press. Buffer space is divided into 3 separate areas, one between the shear and the punch presses, the second between the punch and form presses, and the last between form and bend.

(a) Assume that there is an unlimited amount of space between machines. Run the simulation for 480 hours (about 1 month with 24 hour days, 5 days per week). Where do backups occur? If the total buffer space for all three buffers is limited to 15 sheets (not counting before shear or after bend), how would you recommend dividing this space among the three adjacent pairs of machines? Does this simulation provide enough information to make a reasonable decision?

(b) Modify the model so that there is a finite buffer between adjacent machines. When the buffer becomes full and the machine feeding the buffer completes a sheet, the sheet is not able to exit the machine. It remains in the machine blocking additional work. Assume that total buffer space is 15 sheets for the 3 buffers.

Use the recommendation from part (a) as a starting point for each buffer size. Attempt to minimize the number of runs. You are allowed to experiment with a maximum of 3 buffer sizes for each buffer. (How many runs does this make?) Run a set of experiments to determine the allocation of buffer space that maximizes production. Simulate each alternative for at least 1000 hours.

Report total production per hour on the average, press utilization (broken down by percentage of time busy, down, changing dies, and idle), and average number of sheets in each buffer.

14

Simulation of Computer Systems

It is only natural that simulation is used extensively to simulate computer systems, because of their great unportance to the everyday operations of business, industry, government, and universities. In this chapter, we look at the motivations for simulating computer systems, the different types of approaches used, and the interplay between characteristics of the model a:.d implementation strategies. We begin the discussion by looking at general characteristics of computer-system simulations. Next, we lay the groundwork for investigating simulation of computer systems by looking at various types of simulation tools used to perform those simulations. In section 14.3, we describe different ways that input is presented or generated for these simulations. We next work through an example of a high-level computer system one might simulate, paying attention to problems of model construction and output analysis. In section 14.5, we turn to the central processing unit (CPU) and point out what is generally simulated and how. Following this, we consider simulation of memory systems, in section 14.6.

14.1 INTRODUCTION

Computer systems are incredibly complex. A computer system exhibits complicated behavior at time scales from the time to "flip" a transistor's state (on the order of 10^{-11} seconds) to the time it takes a human to interact with it (on the order of seconds or minutes). Computer systems are designed hierarchically, in an effort to manage this complexity. Figure 14.1 illustrates the point. At a high level of abstraction (the system level), one might view computational activity in terms of tasks circulating among servers, queueing for service when a server is busy. A lower level in the hierarchy can view the activity as being among components of a given processor (its registers, its memory hierarchy). At a lower level still, one views the activity of functional units that together make up a central processing unit, and, at an even lower level, one can view the logical circuitry that makes it all happen.

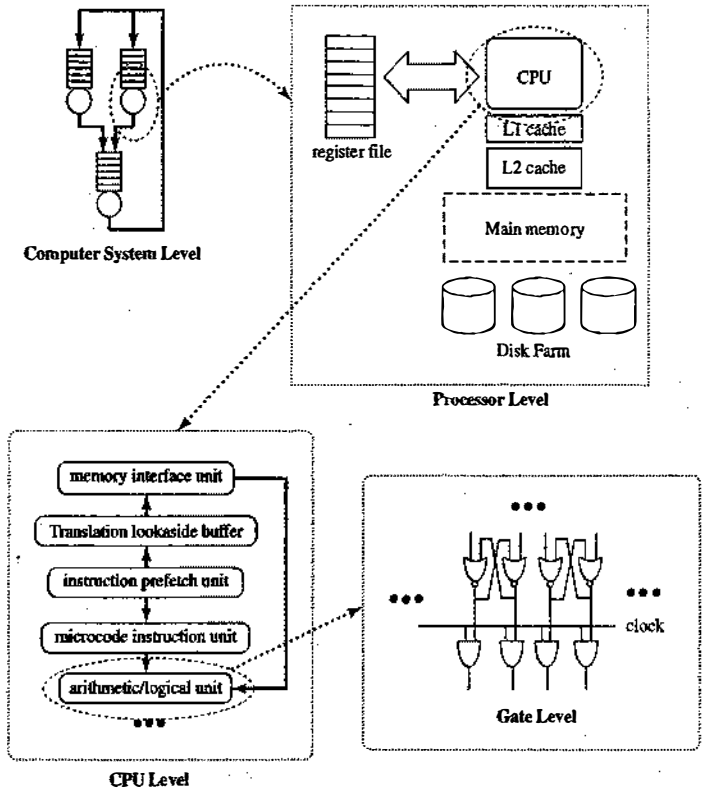


Figure 14.1 Different levels of abstraction in computer systems.

Simulation is used extensively at every level of this hierarchy, with some results from one level being used at another. For instance, engineers working on designing a new chip will begin by partitioning the chip functionally (e.g., the subsystem that does arithmetic, the subsystem that interacts with memory, and so on), establish interfaces between the subsystems, then design and test the subsystems individually. Given a subsystem design, the electrical properties of the circuit are first studied by using a circuit simulator that solves differential equations describing electrical behavior. At this level, engineers work to ensure the correctness of signals' timing throughout the circuit and to ensure that the electrical properties fall within the parameters intended by the design. Once this level of validation has been achieved, the electrical behavior is abstracted into logical behavior (e.g., signals formerly thought of as electrical waveforms are now thought of as logical 1's and 0's). A different type of simulator is next used to test the correctness of the circuit's logical behavior. A common testing technique is to present the design with many different sets of logical inputs ("test vectors") for which the desired logical outputs are known. Discrete-event simulation is used to evaluate the logical response of the circuit to each test vector and is also used to evaluate timing (e.g., the time required to load a register with a datum from the main memory). Once a chip's subsystems are designed and tested, the designs are integrated, and then the whole system is subjected to testing, again by simnlation.

At a higher level, one simulates by using functional abstractions. For instance, a memory chip could be modeled simply as an array of numbers, and a reference to memory as just an indexing operation. A special type of description language exists for this level, called "register-transfer-language" (see, for instance, Mano 1993). This is like a programming language, with reassigned names for registers and other hardware specific entities and with assignment statements used to indicate data transfer between hardware entities. For example, the following sequence loads into register `r3` the data whose memory address is in register `r6`, subtracts one from it, and writes the result into the memory location that is word adjacent (a word in this example is 4 bytes in size) to the location first read:

```
r3 = M[r6];
r3 = r3-1;
r6 = r6+4;
M[r6] = r3;
```

A simulator of such a language might ascribe deterministic time constants to the execution of each of these statements. This is a useful level of abstraction to use when one needs to express sequencing of data transfers at a low level, but not so low as the gates themselves. The abstraction makes sense when one is content to assume that the memory works and that the time to put a datum in or out is a known constant. The "known constant" is a value resulting from analysis at a lower level of abstraction. Functional abstraction is also commonly used to simulate subsystems of a central processing unit (CPU), in the study of how an executing program exercises special architectural features of the CPU.

At a higher level still, one might study how an Input–Output (I/O) system behaves in response to execution of a computer program. The program's behavior may be abstracted to the point of being *modeled*, but with some detailed description of I/O demands (e.g., with a Markov chain that with some specificity describes an I/O operation as the Markov chain transitions). The behavior of the I/O devices may be abstracted to the point that all that is considered is how long it takes to complete a specified I/O operation. Because of these abstractions, one can simulate larger systems, and simulate them more quickly. Continuing in this vein, at a higher level of abstraction still, one dispenses with specificity altogether. The execution of a program is modeled with a randomly sampled CPU service interval; its I/O demand is modeled as a randomly sampled service time on a randomly sampled I/O device.

Different levels of abstraction serve to answer different sorts of questions about a computer system, and different simulation tools exist for each level. Highly abstract models rely on stochastically modeled behavior to estimate high-level system performance, such as throughput (average number of "jobs" processed per unit time) and mean response time (per job). Such models can also incorporate system failure and repair and can estimate metrics such as mean time to failure and availability. Less abstract models are used to evaluate specific systems components. A study of an advanced CPU design might be aimed at estimating the throughput (instructions executed per unit time); a study of a hierarchical memory system might seek to estimate the fraction of time that a sought memory reference was found immediately in the examined memory. As we have already seen, more detailed models are used to evaluate functional correctness of circuit design.

14.2 SIMULATION TOOLS

Hand in hand with different abstraction levels, one finds different tools used to perform and evaluate simulations. We next examine different types of tools and identify important characteristics about their function and their use.

An important characteristic of a tool is how it supports model building. In many tools, one constructs networks of components whose local behavior is already known and already programmed into the tool. This is a powerful paradigm for complex model construction. At the low end of the abstraction hierarchy, electrical

circuit simulators and gate-level simulators are driven by network descriptions. Likewise, at the high end of the abstraction hierarchy, tools that simulate queueing networks and Petri nets are driven by network descriptions, as are sophisticated commercial communication-system simulators that have extensive libraries of preprogrammed protocol behaviors. Some of these tools allow one to incorporate user-programmed behavior, but it appears this is not the norm as a usage pattern.

A very significant player in computer-systems design at lower levels of abstraction is the VHDL language (e.g., see Ashenden [2001]). VHDL is the result of a U.S. effort in the 1980's to standardize the languages used to build electronic systems for the government. It has since undergone the IEEE standardization process and is widely used throughout the industry. As a language for describing digital electronic systems, VHDL serves both as a design specification and as a simulation specification. VHDL is a rich language, full both of constructs specific to digital systems and the constructs one expects to find in a procedural programming language. It achieves its dual role by imposing a clear separation between system topology and system behavior. Design specification is a matter of topology; simulation specification is a matter of behavior. Libraries of predefined subsystems and behaviors are widely available, but the language itself very much promotes user-defined programmed behavior. VHDL is also innovative in its use of abstract interfaces (e.g., to a functional unit) to which different "architectures" at different levels of abstraction may be attached. For instance, the interface to the Arithmetic Logical Unit (ALU) would be VHDL "signals" that identify the input operands, the operation to be applied to them, and the output. One could attach to this interface an architecture that in a few lines of code just performs the operation—if an addition is specified, just one VHDL statement assigns the output signal to be the sum (using the VHDL addition operator) of the two input signals. An alternative architecture could completely specify the gate-level logical design of the ALU. Models that interact with the ALU interface cannot tell how the semantics of the interface are implemented. This separation of interface from architecture supports modular construction of models and allows one to validate a new submodel architecture by comparing the results it returns to the interface with those returned by a different architecture given the same inputs. A substantive treatment of VHDL is well beyond the scope of this book. VHDL is widely used in the electrical and computer engineering community, but is hardly used outside of it.

One drawback to VHDL is that it is a big language, requires a substantial VHDL compiler, and vendors typically target the commercial market at prices that exclude academic research. Of course, other simulation languages exist, and this text describes several in Chapter 4. Such languages are good for modeling certain types of computer systems at a high level, but are not designed or suited for expression of computer-systems modeling at lower levels of the abstraction hierarchy. As a result, when computer scientists need to simulate specialized model behavior, they will often write a simulation (or a simulator) from scratch. For example, if a new policy for moving data between memories in a hierarchy is to be considered, an existing language will not have that policy preprogrammed; when a new architectural feature in a CPU is designed, the modeler will have to describe that feature and its interaction with the rest of the CPU, using a general programming language. A class of tools exists that use a general programming language to express simulation-model behavior, among them SimPack (Fishwick [1992]), C++SIM (Little and McCue [1994]), CSIM (Schwetman [1986]), Awesime (Grunwald [1995]), and SSF (Cowie *et al.* [1999]). This type of tool defines objects and libraries for use with such languages as C, C++, Java. Model behavior is expressed as a computer program that manipulates these predefined objects. The technique is especially powerful when used with object-oriented languages, because the tool can define base-class objects whose behavior is extended by the modeler.

Some commercial simulation languages do support interaction with general programming languages; however, simulation languages are not frequently used in the academic computer-science world. Cost is a partial explanation. Commercial packages are developed with commercial needs and commercial budgets in mind, yet computer scientists can usually develop what they need relatively quickly, themselves. Another explanation is a matter of emphasis: Simulation languages tend to include a rich number of predefined

simulation objects and actions and allow access to a programming language to express object behavior; a simulation model is expressed primarily in the constructs of the simulation language, and the model is evaluated either by compiling the model (using a simulation-language-specific compiler) and running it or by using a simulation-language-specific interpreter.

One of the many advantages to such an approach is that the relative rigidity of the programming model makes possible graphical model building, thereby raising the whole model-building endeavor to a higher level of abstraction. Some tools have so much preprogrammed functionality that it is possible to design and run a model without writing a single line of computer code.

By contrast, programming languages with simulation constructs tend to define a few elemental simulation objects; a simulation model is expressed principally via the notions and control flow of the general programming language, with references to simulation objects interspersed. To evaluate the model, one compiles or interprets the program, using a compiler or interpreter associated with the general programming language, as opposed to one associated with the simulation language. The former approach supports more rapid model development in contexts where the language is tuned to the application; the latter approach supports much greater generality in the sorts of models that can be expressed.

Among tools supporting user-programmed behavior, a fundamental characteristic is the worldview that is supported. In the following two subsections, we look closely at process orientation as it is expressed in SSF, then at an event-oriented approach using a Java base framework.

14.2.1 Process Orientation

A process-oriented view (see Chapter 3) implies that the tool must support separately schedulable threads of control. Threading is a fundamental concept in programming, and a discussion of its capabilities and implementation serves to highlight important issues in simulation modeling. Fundamentally, a "thread" is a separately schedulable unit of execution control, implemented as part of a single executing process (as seen by the operating system; see Nutt [2004]). An operating system has the notion of separate processes (which might interact), which typically have their own separate and independent memory spaces. A group of threads operate in the same process memory space, with each thread having allocated to it a relatively small portion of that space for its own use. That space is used to contain the thread's *state*, which is the full set of all information needed to restart the thread after it is suspended. State would include register values and the thread's runtime stack, which holds variables that are local to the procedures called by the thread. Once a thread is given control, it runs until it yields up control, either via an explicit statement that serves simply to relinquish control or by blocking until signaled by another thread to continue.

These ideas are made more concrete by discussing them in the context of a Java implementation of SSF. Java defines the `Thread` class; a subclass of `Thread` defines the `execute` method, which is defined in the thread body. Threads coordinate with each other through "locks," which provide mutually exclusive access to code segments. Every instance of a Java object has an associated lock (and almost every variable in Java is an object). A thread tries to execute a code fragment protected by the lock for object `obj` via the Java statement

```
synchronized(obj) { /* code fragment */ }
```

A thread must acquire the lock before executing the code fragment, and only one thread has the lock at a time. A thread that executes a `synchronized` statement at an instant at which another thread holds the lock blocks—which could mean suspension, depending on the thread scheduler. Java threads can also coordinate through `wait` and `notify` method calls, also associated with an object's lock. A thread that executes `obj.wait()` suspends. Actually, multiple threads can execute `obj.wait()`, and each will suspend. Eventually some thread executes `obj.notify()`, and the thread scheduler releases one of the suspended threads to continue.

These notions can be used to implement process orientation in a Java simulator. Each simulation process derives from the Java Thread class. One additional thread will maintain an event list; processing for that thread involves removing the least-time event from the event list, reanimating the simulation process thread (or threads) associated with that event, and blocking until those threads have completed. While a process thread is executing, it may cause additional events to be inserted into the scheduler thread's event list. When a process thread completes, it needs to block and to signal the scheduler thread that it is finished. We accomplish all this by using two locks per simulation process. One of these locks is the one Java provides automatically for every object (and a simulation process thread is an object). The other lock is a variable each simulation object defines, which we'll call `lock`. A suspended process thread blocks on a call `lock.wait();` it remains blocked there until the scheduling thread executes `notify()` on that same object variable. After the scheduler does this, it blocks by calling `wait()` on the simulation process object's own built-in lock. So the simulation process thread notifies the scheduler that it is finished by calling `notify()` on its own built-in lock.

SSF code we discussed earlier in Chapter 4 (Figures 4.14 and 4.15) illustrates some of these points. Recall that this code models a single server with exponentially distributed interarrival times and positive normal service times. A cursory glance shows the model to be legitimate Java code that uses SSF base classes.

SSF defines five base classes around which simulation frameworks are built (discussed in Chapter 4). The key one for discussing process orientation is the **process** class; derived classes **Arrivals** in Figure 4.14 and **Server** in Figure 4.15 are examples of it. The base class specifies that method **action** be the thread body; each derived class overrides the base-class definition to specify its own thread's behavior. Every object of a given class derived from **process** defines a separate thread of control, but all execute the same thread code body.

The `waitFor` statement used in `Arrival`'s thread body suspends the thread; its argument specifies how long in simulation time the thread suspends. The Java thread-based scheduling mechanism we described earlier enables implementation of `waitFor` to cause a "wake-up" event to be inserted into the scheduling thread's event list, time stamped with the current time plus the `waitFor` argument. Here variable `time` is the future-event time; method `insertProcess` puts the process into the event queue. A non-Simple process (e.g., one implemented with a Java thread) goes through a sequence of synchronization steps to reach the `notify()` method. (We will say more about Simple processes in 14.2.2.) The scheduler thread has blocked on the process's native lock; this `notify()` releases it. The process then immediately calls `wait()` on its `lock` variable, which suspends the thread until the scheduler executes `notify()` on that same variable. From the point of view of the code fragment executing `waitFor`, the statement following the `waitFor` call executes precisely at the time implied by the `waitFor` argument. The code in Figure 14.2 (taken from an SSF implementation) illustrates this.

```
public void waitFor(long timeinterval){
  time = owner.owner.clock + timeinterval;
  owner.owner.insertProcess(this);
  if (!isSimple()){
    synchronized(lock){
    synchronized(this){
        notify();
      }
      try{lock.wait();}
      catch(InterruptedException e){}
    }
  }
}
```

Figure 14.2 SSF implementation of waitFor statement.

The call to `waitOn` in the **Server**'s `action` has a slightly different implementation. The code implementing `waitOn` first attaches the process to the `inChannel`'s list of processes that are blocked on it, then engages in the same lock synchronization sequence as `waitFor` to block itself and release the scheduler thread. The semantics of releasing a blocked process are defined in terms of SSF `Events`. An `outChannel` object to which an `Event` object is written has almost always been "mapped" to an `inChannel` object. When an `Event` is written to an `outChannel` at time t, the `outChannel`'s `write` method computes the time $t + d$ at which the `Event` is available on the associated `inChannel` (d is a function of delays declared when the `outChannel` is created, the `mapTo` method is called, and the `write` method is called), and an internal event is put on the scheduler's event list, with time stamp $t + d$. The scheduler executes this event (no SSF process does) and releases all processes blocked on the `inChannel` to which the `Event` arrives. Each of these is able to get a copy of the `Event` so delivered, by calling the `inChannel`'s `activeEvents` method.

From these descriptions, we see that, normally, each event has a thread overhead cost: 2 thread reanimations, and 2 thread suspensions. Depending on how thread context switching is implemented, this cost ranges from heavy to very heavy, as compared with a purely event-oriented view. These costs can be avoided in SSF by designing processes to be *simple*, as is described next.

14.2.2 Event Orientation

From a methodological point of view, the process-oriented view is distinguished from the event-oriented view in terms of the focus of the model description. Process orientation allows for a continuous description, with pauses or suspensions. Event orientation does not. From an *implementation* point of view, the key distinguishing feature of process-oriented simulation is the need to support suspension and reanimation, which leads us to threads, as we have seen. In SSF, though, we see that the difference between process and event orientation is not very large: The SSF world encompasses both. The only difference is that, for SSF to be event oriented, its processes need to be *simple*, a technical term for the case when every statement in `action` that might suspend the process would be the last statement executed under normal execution semantics.

The implementation of `waitFor` in Figure 14.2 computes the time when the suspension is lifted and puts a reanimation event in the event list. Synchronization by threads through locks is used only if the process is not *simple*. An implementation of `waitOn` would be entirely similar. If every SSF process in a model is *simple*, there is no true code suspension, and the model is essentially event oriented. The `action` body for a *simple* process is just executed from its normal entry point when the condition that releases that process from "suspension" is satisfied. The only way an **Event** that is written into an `outChannel` is delivered is if the recipient had called `waitOn` for the corresponding `inChannel` at a time prior to that at which the `Event` was written. Thus, we see that some of the "events" implicit in an SSF model with event orientation are kernel events, which decide whether model events ought to be executed as a result. Writing to an `outChannel` schedules a kernel event at the `Event`'s receive time, but the kernel's processing of that event determines whether an `action` body is called. Nevertheless, execution of `action` bodies constitutes the essential "event processing" when SSF is used in a purely event-oriented view. It is interesting that, from a conceptual point of view, there is very little difference between process-oriented and event-oriented SSF.

To conclude this discussion on tools, we remark that *flexibility* is the key requirement in computer-systems simulation. Flexibility in most contexts means the ability to use the full power of a general programming language. This requires a level of programming expertise that is not needed by users of commercial graphically oriented modeling packages. The implementation requirements of an object-oriented event-oriented approach are much less delicate than those of a threaded simulator, and the amount of simulator overhead involved in delivering an event to an object is considerably less than the cost of a context switch in a threaded system. For these reasons, most of the simulators written from scratch take the event-oriented view. However, the underlying simulation framework necessarily provides a lower level of abstraction and so forces a modeler to design and implement more model-management logic. The choice between using a process-oriented or an event-oriented simulator—or writing one's own—is a function of the level of modeling ease, versus execution speed.

To summarize this section, we present a table that lists different levels of abstraction in computer-systems simulation, the sorts of questions whose answers are sought from the models, and the sorts of tools typically used for modeling. The level of abstraction decreases as one descends through Table 14.1.

14.3 MODEL INPUT

Just as there are different levels of abstraction in computer-systems simulation, there are different means of providing input to a model. The model might be driven by stochastically generated input, or it might be given trace input, measured from actual systems. Simulations at the high end of the abstraction hierarchy most typically use stochastic input; simulations at lower levels of abstraction commonly employ trace input. Stochastic input models are particularly useful when one wishes to study system behavior over a range of scenarios; it could be that all that is required is to adjust an input model parameter and rerun the simulation. Of course, using randomly generated input raises the question of how real or representative the input is; that doubt frequently induces systems people to prefer trace data on lower level simulations. Using a trace means one cannot explore different input scenarios, but traces are useful when directly comparing two different implementations of some policy or some mechanism on the same input. The realism of the input gives the simulation added authority.

In all cases, the data used to drive the simulation is intended to exercise whatever facet of the computer system is of interest. High-level systems simulations accept a stream of job descriptions; CPU simulations accept a stream of instruction descriptions; memory simulations accept a stream of memory references; and gate-level simulations accept a stream of logical signals.

Computer systems modeled as queueing networks (recall Chapter 7) typically interpret "customers" as computer programs; servers typically represent services such as attention by the CPU or an Input–Output (I/O) system. Random sampling generates customer interarrival times; it may also be used to govern routing and time in service. However, it is common in computer-systems contexts to have routing and service times be state dependent (e.g., the next server visited is already specified in the customer's description, or could be the attached server with least queue length).

Interarrival processes have historically been modeled as Poisson processes (where times between successive arrivals have an exponential distribution). However, this assumption has fallen from favor as a result of empirical observations that significantly contradict Poisson assumptions in current computer and communication systems. The real value of Poisson assumptions lies in tractability for mathematical analysis, so, as simulationists, we can discard them with little loss.

In the subsections to follow, we look at the mathematical formulation of common input models, stochastic input models for virtual memory, and direct-execution techniques.

Table 14.1 Decreasing Abstraction and Model Results

Typical System	*Model Results*	*Tools*
CPU Network	job throughput, job response time	queueing network, Petri net simulators, scratch
Processor	instruction throughput, time/instruction	VHDL, scratch
Memory System	miss rates, response time	VHDL, scratch
ALU	timing, correctness	VHDL, scratch
Logic Network	timing, correctness	VHDL, scratch

14.3.1 Modulated Poisson Process

Stochastic input models ought to reflect the real-life phenomenon called *burstiness*—that is, brief periods when traffic intensity is much higher than normal. An input model sometimes used to support this, retaining a useful level of mathematical tractability, is a *Modulated Poisson Process*, or MPP. (See Fischer and Meier-Hellstern, [1993].) The underlying framework is a continuous-time Markov chain (CTMC), whose details we sketch so as to employ the concept later. A CTMC is always in some *state*; for descriptive purposes, states are named by the integers: 1, 2, ... The CTMC remains in a state for a random period of time, transitions randomly to another state, stays there for a random period of time, transitions again, and so on. The CTMC behavior is completely described by its *generator matrix*, $\mathcal{Q} = \{q_{i,j}\}$. For states $i \neq j$, entry $q_{i,j}$ describes the rate at which the chain transitions from state i into state j (this is the total transition rate out of state i, times the probability that it transitions then into state j). The rate describes how quickly the transition is made; its units are transitions per unit simulation time. Diagonal element $q_{i,j}$ is the negated sum of all rates out of state i : $q_{i,i} = -\sum_{j \neq i} q_{i,j}$. An operational view of the CTMC is that, upon entering a state i, it remains in that state for an exponentially distributed period of time, the exponential having rate $-q_{i,i}$. When making the transition, it chooses state j with probability $-q_{i,j}/q_{i,i}$. Many CTMCs are *ergodic*, meaning that, if it is left to run forever, every state is visited infinitely often. In an ergodic chain, π_i denotes state i's *stationary probability*, which we can interpret as the long-term average fraction of time the CTMC is in state i. A critical relationship exists between stationary probabilities and transition rates : For every state i,

$$\pi_i \sum_{j \neq i} q_{i,j} = \sum_{j \neq i} \pi_j q_{j,i}$$

If we think of $q_{i,j}$ as describing a probability "flow" that is enabled when the CTMC is in state i, then these equations say that, in the long term, the sum of all flows out of state i is the same as the sum of all flows into the state. We will see in the example that follows that we can use the balance equations to build a stochastic input with desired characteristics. To complete the definition of a MPP, it remains only to associate a customer arrival rate λ_i with state i. When the CTMC is in state i, customers are generated as a Poisson process with rate λ_i.

To illustrate, let us consider an input process that is either OFF, ON, or BURSTY (the output rate is much higher in the BURSTY state than in the ON state). We wish for the process to be OFF half of the time—on average, for 1 second—and, when it is not OFF, we wish for it to be BURSTY for 10% of the time. We will assume that the CTMC transitions into BURSTY only from the ON state and transitions out of BURSTY only into the ON state. We will say that state 0 corresponds to OFF, 1 to ON, and 2 to BURSTY. Our problem statement implies that $\pi_0 = 0.5$, $\pi_1 = 0.45$, and $\pi_2 = 0.05$. The only transition from OFF is to ON, and the mean OFF time is 1, so we infer that $q_{0,1} = 1$. The balance equation for state 0 can be rewritten as

$$0.5 = 0.45 q_{1,0}$$

and hence $q_{1,0} = (0.5/0.45)$. The balance equation for state 1 can be rewritten as

$$0.45((0.5/0.45) + q_{1,2}) = 0.5 + 0.05 q_{2,1}$$

and the balance equation for state 2 is

$$0.05 q_{2,1} = 0.45 q_{1,2}$$

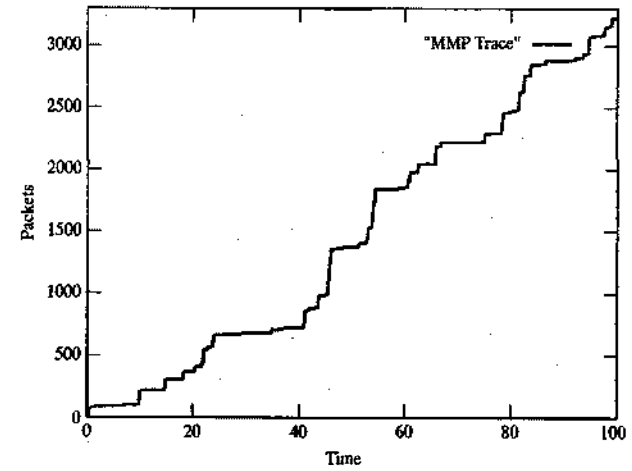
The equations for states 1 and 2 are identical; mathematically, we don't have enough conditions to force a unique solution. If we add the constraint that a BURSTY period lasts, on average, 1/10 of a second, we thereby define that $q_{2,1} = 10$ and, hence, that $q_{1,2} = (0.5/0.45)$. Operationally, the simulation of this CTMC is

straightforward. In state 0, one samples an exponential with mean 1 to determine the state's holding time. Following this period, the CTMC transitions into state 1 and samples a holding time from an exponential with mean 0.45, after which it transitions to OFF or BURSTY with equal probability. In the BURSTY state, it samples an exponential holding time with mean 0.1. Now all that is left is for us to define the state-dependent customer arrival rates. Obviously, $\lambda_0 = 0$; for illustration, we choose $\lambda_1 = 10$ and $\lambda_2 = 500$.

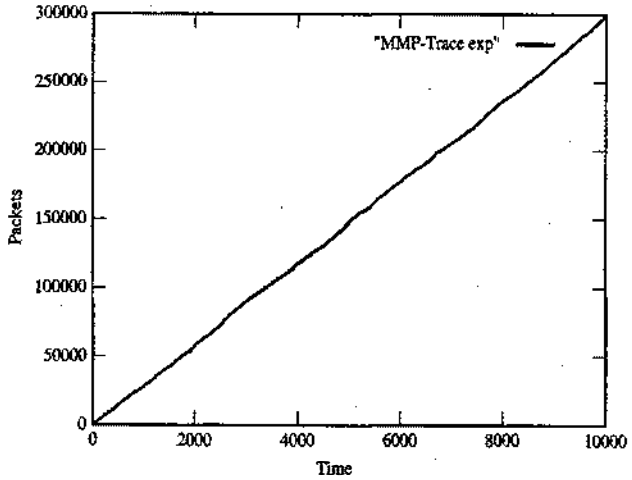
Figure 14.3 presents a snippet of code used to generate times of arrivals in this process. Transitions between states are sampled by using the inverse-transform technique, described in Chapter 9. (The variable `acc` computes the cumulative probability function in the distribution described by the row vector `P[state]`.) Figure 14.4 plots total customers generated as a function of time—for a short period of a sample run, and for a longer period. In the shorter run, we see regions where the graph increases sharply; they correspond to periods in the BURSTY state. While the CTMC is not in this state, a mixture of OFF and ON periods moves the accumulated packet count up at a much more gradual rate. The MPP model can describe burstiness, but the burstiness is limited in time scale. The longer run views the data at a time scale that is two orders of magnitude larger, and we see that the irregularities are largely smoothed.

```
class mpp {

 public static double Finish;          // sim termination
 public static double time = 0.0;      // current clock
 public static double htime, etime;    // transition times
 public static int state = 0;          // current state id
 public static int total = 0;          // total pkts emitted
 public static Random stream;
 ...

public static void main(String argv[]) {
 ...
while( time < Finish ) {

 // generate exponential holding time, state-dependent mean
 htime = time+exponential( stream, hold[state] );

 // emit packets until state transition time. State dependent
 // rate. Note assignment made to etime in while condition test
 while( (etime = time+exponential( stream, 1.0/rate[state]))
                 < min( htime, Finish) ) {
        System.out.println( etime + " " + total);
   total++;
   time = etime; // advance to packet issue time
 }
 time = htime;

 // select next state
 double trans = stream.nextDouble();
 double acc = P[state][0];
 int i = 0;

 while( acc < trans ) acc += P[state][++i];
 state = i;
  }
 }
 ...
}
```

Figure 14.3 Java code generating MPP trace.



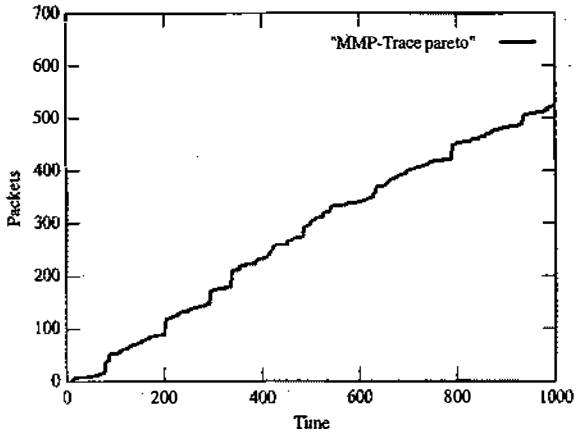
(a) Short run, small time scale



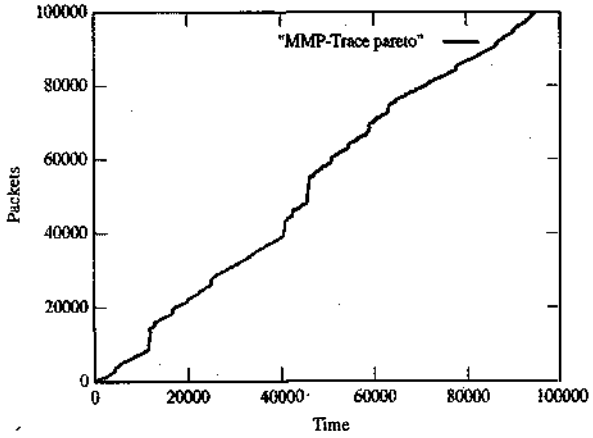
(b) Long run, large time scale

Figure 14.4 Sample runs from MPP model.

In contrast to the Markovian essence of the MPP model, consider a traffic source that remains OFF for an exponentially distributed period of time with mean 1.0, but, when it comes ON, remains on for a period of time sampled from a Pareto distribution. While it is ON, packets arrive as a Poisson process. As we will see in the chapter on simulation of computer networks, the Pareto distribution is of particular interest because it gives rise to "self-similarity," which informally means preservation of irregularities at multiple time scales. Figure 14.5 parallels the MPP data, displaying accumulated packet counts as a function of time; it presents behavior for the first 1000 units of time and for the first 100,000 units of time. Here, despite two orders of



(a) Short run, small time scale



(b) Long run, large time scale

Figure 14.5 Sample runs from self-similar model.

magnitude of difference in run length, the visual impression of behavior is much the same between the two traces. This sort of behavior is frequently seen in computer and communication systems; the long lengths reflect burstiness of packets, file lengths, and demand on a server.

14.3.2 Virtual-Memory Referencing

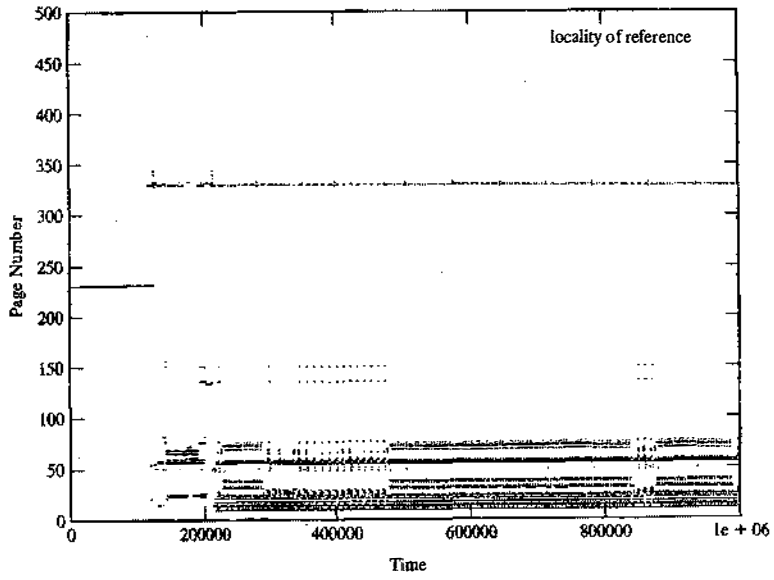
Randomness can also be used to drive models in the middle levels of abstraction. An example is a model of program-execution behavior in a computer with virtual memory. (See Nutt [2004].) In such a system, the data and instructions used by the program are organized in units called *pages*. All pages are the same size, typically 2^{10} to 2^{12} bytes in size. The physical memory of a computer is divided into *page frames*, each capable

of holding exactly one page. The decision of which page to map to which frame is made by the operating system. As the program executes, it makes memory references to the "virtual memory," as if it occupied a very large memory starting at address 0 and were the only occupant of the memory. On every memory reference made by the program, the hardware looks up the identity of the page frame containing the reference and translates the virtual address into a physical address. The hardware might discover that the referenced page is not present in the main memory; this situation is called a *page fault*. When a page fault occurs, the hardware alerts the operating system, which then takes over to bring in the referenced page from a disk and decides which page frame should contain it. The operating system could need to evict a page from a page frame to make room for the new one. The policy the operating system uses to decide which page to evict is called the "replacement policy." The quality of a replacement policy is often measured in terms of the *hit ratio*—the fraction of references made whose page frames are found immediately.

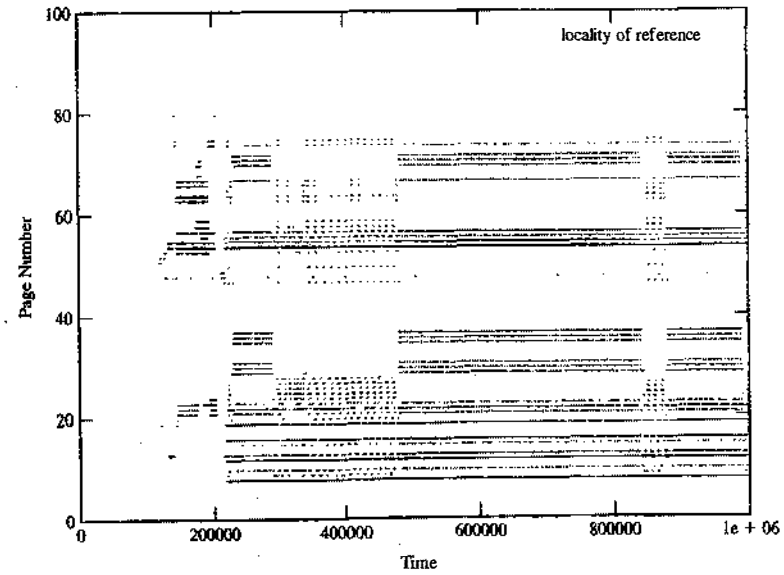
Virtual-memory systems are used in computers that support concurrent execution of multiple programs. In order to study different replacement policies, one could simulate the memory-referencing behavior of several different programs, simulate the replacement policy, and count the number of references that page fault. For this simulation to be meaningful, it is necessary that the stochastically generated references capture essential characteristics of program behavior. Virtual memory works well precisely because programs do tend to exhibit a certain type of behavior; this behavior is called locality of reference. What this means intuitively is that program references tend to cluster in time and space and that, when a reference to a new page is made and the page is brought in from the disk, it is likely that the other data or instructions on the page will also soon be referenced. In this way, the overhead of bringing in the page is amortized over all the references made to that page before it is eventually evicted. A program's referencing behavior can usually be separated into a sequence of "phases"; during each phase, the program makes references to a relatively small collection of pages, called its *working set*. Phase transitions essentially change the program's working set. The challenge for the operating system is to recognize when the pages used by a program are no longer in its working set, for these are the pages it can safely evict to make room for pages that *are* in some program's working set.

Figure 14.6 illustrates a stream of memory references taken from an execution of the commonly used gcc compiler. One graph gives a global picture; the other cuts out references to pages over number 100 and shows more fine detail. Each graph depicts points of the form (i, p_i) where p_i is the page number of the ith reference made by the program (arithmetically shifted so that the smallest page number referenced is 10). The phases are clearly seen; each member of the working set of a phase is seen as lines (which are really just a concatenation of many points). One striking facet of this graph is how certain pages remain in almost all working sets. However, other kinds of programs exhibit other behaviors. A common characteristic of scientific programs is that the execution is dominated by an inner loop that sweeps over arrays of data; the pages containing the instructions are in the working set throughout the loop, but data pages migrate in and out.

Despite various differences, a near-invariant among program executions is the presence of phase-like behavior and of working sets. In the building of a stochastic reference generator, it therefore makes sense to focus modeling effort on phase and working-set definition. As a starting point, we might, with every reference generated, randomly choose (with some small probability) whether to start a new phase (by changing the working set). Given a working set, we would choose to reference some page in the working set with high probability and, if choosing to stay in the set, choose with high probability the same page as the one last referenced in the working set. The inner loop of a program that generates references in this fashion appears in Figure 14.7. Details of working-set definition are hidden inside of routine `new_wrkset` and might vary with the type of program being modeled. For the purposes of illustration here, we wrote a version that defined a working set by randomly choosing a working-set size between 2 and 8 and a maximum page number of 100. A working set of size n is constructed by randomly choosing a "center" page c from among all pages, randomly choosing an integer dispersion factor d from 2 to 6, and then randomly selecting a working set from among all pages within distance $d \times n$ from center page c (with appropriate wraparound of page numbers at the endpoints 0 and 100). In order to model the referencing pattern of a scientific program's



(a) gcc, all references shown



(b) gcc, references < 100 shown

Figure 14.6 Scatter-plotted referencing pattern of gcc compiler. Referenced page number is plotted as a function of reference number ("time"). Horizontal sequences indicate frequent rereferences to the same page number.

```
double ppt = 0.0001;  // Pr{phase transition}
double psw = 0.999;   // Pr{ref in WS}
double psp = 0.9;     // Pr{reference same page}

// method new_wrkset() creates a new working set
// method from_wrkset() samples from the working set
// method not_from_wrkset() samples from outside the working set

int ref;              // last page referenced
int sv_ref;           // save ref
Random stream;        // random number stream

...

for(int i=0; i<length; i++) {
   if( stream.nextDouble() < ppt ) new_wrkset();    // phase transition
   if( stream.nextDouble() < psw ) {                // stay in working set?
    if( psp < stream.nextDouble() )                      // change page, in wrkset
       ref = sv_ref = from_wrkset();
  } else ref = not_from_wrkset();    // step outside of wrkset

System.out.println(i + " " + ref);
ref = sv_ref;
}
```

Figure 14.7 Java pseudocode for generating a reference trace.

instruction stream, we manipulated the logic illustrated above to "lock down" a working set for a long time in the middle of the program execution. Figure 14.8 illustrates the result. As designed, phases and working sets are precisely defined.

The preceding example illustrates how one can in principle generate an execution path stochastically, but simulations at the middle level of abstraction also commonly use traces. Studies of CPU design will use a measured trace of instructions executed by a running program; studies of memory systems will use a measured trace of the addresses referenced by an executing program. Such traces get to be lengthy. A small piece of a typical trace of memory references is shown here:

```
2      430d70
2      430d74
2      415130
0    1000acac
2      414134
1    7fff00ac
2      414138
```

The first number is a code describing the type of access; 2 represents an instruction fetch, 0 a data read, 1 a data write. The second number represents a memory address, in hexadecimal. If the trace were also to describe the instruction stream, a hexadecimal word giving the machine code of the instruction fetched could follow the memory address on every instruction fetch line. Two or three words of memory are needed to represent one reference, even when the information is efficiently packed (not as characters, as shown, which take much more space!). Consider also the amount of computation needed to simulate a CPU or memory for the execution of a significantly long run of a nontrivial program. These observations help us understand the motivation for techniques that compress the address trace and for techniques that allow one to infer information about multiple systems from a single pass through a long trace. We will say more about these techniques later in this chapter.

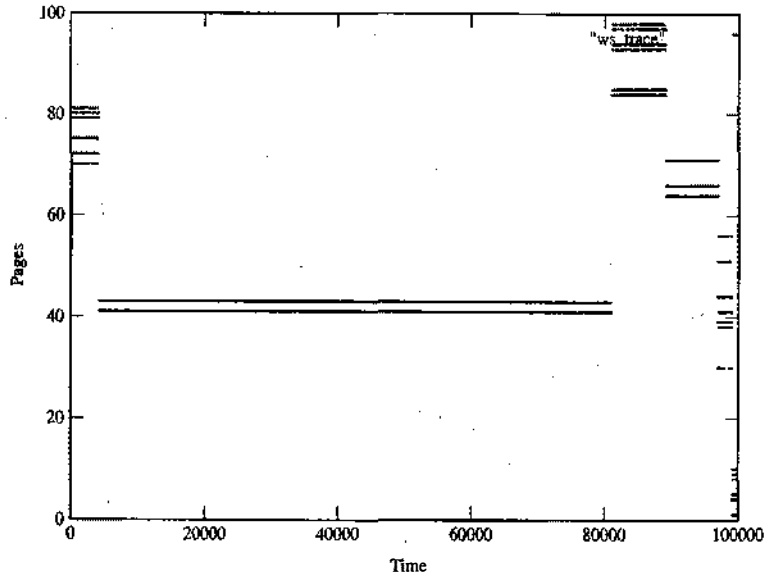


Figure 14.8 A synthetic trace modeling a scientific-program instruction stream.

Another method of generating input is called "direct execution" simulation. (For examples, see Covington *et al.* [1991], Lebeck and Wood [1997], Dickens *et al.* [1996]). One approach to it is illustrated in Figure 14.9. Direct execution is like generating a trace and driving the simulation with that trace, all at once. Computer programs are "instrumented" with additional code that observes the instructions the program executes and the

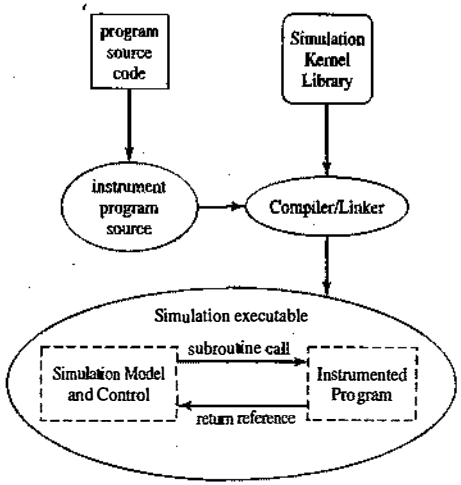


Figure 14.9 Direct-execution simulation.

memory and I/O references the program makes as it executes. The instrumented program is compiled and linked with a simulation kernel library. Execution control rests with the simulation kernel, which calls the instrumented program to provide the next instruction or reference that the program generates. The simulation kernel uses the returned information to drive the model for the next step. The simulation model driven by the program's execution can be of an entirely different CPU design, or a memory system, or even (given multiple instrumented programs) the internals of a communications network. Direct-execution simulation solves the problem of storing very large traces—the trace is consumed as it is being generated. However, it is tricky to modify computer programs to get at the trace information and to coordinate the trace generator with discrete-event simulator. The only practical way an ordinary simulator practitioner can use such methods is when the system has a software tool for making such modifications, but this feature is not common.

14.4 HIGH-LEVEL COMPUTER-SYSTEM SIMULATION

In this section, we illustrate concepts typical of high-level computer simulations by sketching a simulation model of a computer system that services requests from the World Wide Web.

Example 14.1

A company that provides a major website for searching and links to sites for travel, commerce, entertainment, and the like wishes to conduct a capacity-planning study. The overall architecture of its system is shown in Figure 14.10. At the back end, one finds data servers responsible for all aspects of handling specific queries and updating databases. Data servers receive requests for service from application servers—machines dedicated to running specific applications (e.g., a search engine) supported by the site. In front of the applications are Web servers, which manage the interaction of applications with the World Wide Web; the portal to the whole system is a load-balancing router that distributes requests directed to the website among the Web servers.

The goal of the study is to evaluate the site's ability to handle load at peak periods. The desired output is an empirical distribution of the access response time. Thus, the high-level simulation model should focus

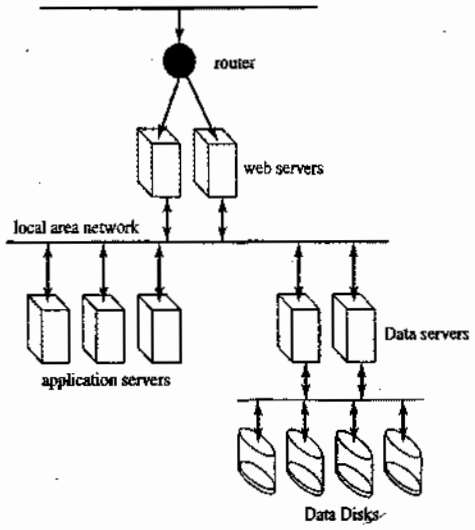


Figure 14.10 Website server system.

on the impact of timing at each level that is used, system factors that affect that timing, and the effects of timing on contention for resources. To understand where those delays occur, let us consider the processing associated with a typical query.

All entries into the system are through a dedicated router, which examines the request and forwards it to some Web server. Time is required to exercise the logic of looking at the request to discern whether it is a new request (requiring load balancing) or part of an ongoing session. It is reasonable to assume one switching time for a preexisting request and a different time for a new request. The result of the first step is selection of a Web server and the enqueueing there of a request for service. A Web server can be thought of as having one queue of threads of new requests, a second queue of threads that are suspended awaiting a response from an application server, and a third queue of threads "ready" to process responses from application servers. An accepted request from the router creates a new request thread. We may assume the Web server has adequate memory to deal with all requests. It has a queueing policy that manages access to the CPU; the distinction between new requests and responses from application servers is maintained for the sake of scheduling and for the sake of assigning service times, the distributions of which depend on the type. The servicing of a new request amounts to identification of an application and the associated application server. A request for service is formatted and forwarded to an application server, and the requesting thread joins the suspended queue. At an application server, requests for service are organized along application types. A new request creates a thread that joins a new-request queue associated with the identified application. An application request is modeled as a sequence of sets of requests from data servers, interspersed with computational bursts—for example,

```
burst 1
request data from D1, D3, and D5
burst 2
request data from D1 and D2
burst 3
```

In this model, we assume that all data requests from a set must be satisfied before the subsequent computational burst can begin. Query search on a database is an example of an application that could generate a long sequence of bursts and data requests, with large numbers of data requests in each set. We need not assume that every execution of an application is identical in terms of data requests or execution bursts; these can be generated stochastically. An application thread's state will include description of its location in its sequence and a list of data requests still outstanding before the thread can execute again. Thus, for each application, we will maintain a list of threads that are ready to execute and a list of threads that are suspended awaiting responses from data servers. An application server will implement a scheduling policy over sets of ready application threads. A data server creates a new thread to respond to a data request and places it in a queue of ready threads. Some data server might implement memory-management policies and could require further coordination with the application server to know when to release used memory. Upon receiving service, the thread requests data from a disk, then suspends until the disk operation completes, at which point the thread is moved from the suspended list to the ready list and, when executed, again reports back to the application server associated with the request. The thread suspended at the application server responds; eventually, the application thread finishes and reports its completion back to the Web-server thread that initiated it, which in turn communicates the results back over the Internet.

Stepping back from the details, we see that a simulation model of this system must specify a number of features, listed in Table 14.2. All of these affecting timing in some way. The query-response-time distribution can be estimated by measuring, for each query, the time between at which a request first hits the router and the time at which the Web-server thread communicates the results. From the set of simulated queries, one can build up a histogram. As should be evident, a response time reflects a great many different factors related to execution bursts, scheduling policies, and disk-access times. Deeper understanding of the system is obtained by measuring behavior at each server of each type. One would look especially for evidence of

Table 14.2 Required Specification for Web System Model

Subsystem	*Specifications*
Router	load-balancing policy, execution times
Web Server	server count, queueing policy, execution times
Application Server	server count, queueing policy, behavior model
Data Server	server count, disk count, queueing policy, memory policy, disk timing

bottlenecks. CPU bottlenecks would be reflected at servers with high CPU utilization; IO bottlenecks at disks with high utilization. To assess system capacity at peak loads, we would simulate to identify bottlenecks, then look to see how to reduce load at bottleneck devices by changes in scheduling policies, by binding of applications to servers, or by increasing the number of CPUs or disks in the system. Normally, one must resimulate a reconfigured system under the same load as before to assess the effects of the changes.

The website model is an excellent candidate for a threaded (process-oriented) approach to modeling. The most natural process-oriented approach is to associate processes with servers. The simulation model is expressed from an abstracted point of view of the servers' operating system. Individual queries become messages that are passed between server processes. In additional to limiting the number of processes, an advantage of this approach is that it explicitly exposes the scheduling of query processing at the user level. The modeler has both the opportunity and the responsibility to provide the logic of scheduling actions that model processing done on behalf of a query. It is a modeling viewpoint that simplifies analysis of server behavior—an overloaded server is easily identified by the (modeler-observable) length of its queue of runnable queries. However, it is a modeling viewpoint that is a bit lower in abstraction than the first one and requires more modeling and coding on the part of the user.

An event-oriented model of this system need not look a great deal different from the second of our process-oriented models. A query passed as a message between servers have an obvious event-oriented expression. A modeler would have to add to the logic, events, and event handlers that describe the way a CPU passes through simulation time. For example, consider a call to *hold(qt)* in a process-oriented model to express that the CPU is allocating qt units of service to a query, during which time it does nothing else. In an event-oriented model, one would need to define events that reflect "starting" and "stopping" the processing of a query, with some scheduling logic interspersed. Additional events and handlers need to be defined for any "signaling" that might be done between servers in a process-oriented model—for example, when a data-server process awaits completion of modeled IO requests sent to its disks. A process-oriented approach, even one focused on servers rather than queries, lifts the level of model expression to a higher level of abstraction and reduces the amount of code that must be written. In a system as complex as the website, one must factor complexity of expression into the overall model-development process.

14.5 CPU SIMULATION

Next, we consider a lower level of abstraction and look at the simulation of a central processing unit. Whereas the high-level simulation of the previous example treated execution time of a program as a constant, at the lower level we do the simulation to discover what the execution time is. The input driving this simulation is a stream of instructions. The simulation works through the mechanics of the CPU's logical design to find out what happens in response to that stream, how long it takes to execute the program, and where bottlenecks exist in the CPU design. Our discussion illustrates some of the functionality of a modern CPU and the model characteristics that such a simulation seeks to discern. Examples of such simulations include

those described in Cmelik and Keppel [1994], Bedicheck [1995], Witchel and Rosenblum [1996], Austin, Larson, and Ernst [2002], Bohrer *et al.* [2004] and Magnusson *et al.* [2002]. The view of the CPU taken in our discussion is similar to that taken by the RSIM system (Hughes *et al.* [2002]).

The main challenge to making effective use of a CPU is to avoid stalling it; stalling happens whenever the CPU commits to executing an instruction whose inputs are not all present. A leading cause of stalls is the latency delay between CPU and main memory, which can be tens of CPU cycles. One instruction might initiate a read—for example,

```
load $2, 4($3)
```

which is an assembly language statement that instructs the CPU to use the data in register 3 (after adding value 4 to it) as a memory address and to put the data found at that address into register 2. If the CPU insisted on waiting for that data to appear in register 2 before further execution, the instruction could stall the CPU for a long time if the referenced address is not found in the cache. High-performance CPUs avoid this by recognizing that additional instructions *can* be executed, up to the point where the CPU attempts to execute an instruction that reads the contents of register 2—for example,

```
add $4,$2,$5
```

This instruction adds the contents of registers 2 and 5, and places the result in register 4. If the data expected in register 2 is not yet present, the CPU will stall. So we see that, to allow the CPU to continue past a memory load, it is necessary to (1) mark the target register as being unready, (2) allow the memory system to load the target register asynchronously while the CPU continues on in the instruction stream, (3) stall the CPU if it attempts to read a register marked as unready, and (4) clear the unready status when the memory operation completes.

The sort of arrangement just described was first used in the earliest supercomputers, designed in the 1960s. Modern microprocessors add some additional capabilities to exploit *instruction level parallelism* (ILP). We outline some of the current architecture ideas in use to illustrate what a simulation model of an ILP CPU involves.

The technique of pipelining has long been recognized as a way of accelerating the execution of computer instructions. (See Patterson and Hennessy [1997].) Pipelining exploits the fact that each instruction goes sequentially through several stages in the course of being processed; separate hardware resources are dedicated to each stage, permitting multiple instructions to be in various stages of processing concurrently. A typical sequence of stages in an ILP CPU is as follows:

1. *Instruction fetch*: The instruction is fetched from the memory.
2. *Instruction decode*: The memory word holding the instruction is interpreted to discover what operation is specified; the registers involved are identified.
3. *Instruction issue*: An instruction is "issued" when there are no constraints holding it back from being executed. Constraints that keep an instruction from being issued include data not yet being ready in an input register and unavailability of a functional unit (e.g., Arithmetic Logical Unit) needed to execute the instruction.
4. *Instruction execute*: The instruction is performed.
5. *Instruction complete*: Results of the instruction are stored in the destination register.
6. *Instruction graduate*: Executed instructions are graduated in the order that they appear in the instruction stream.

Ordinary pipelines permit at most one instruction to be represented in each stage; the degree of parallelism (number of concurrent instructions) is limited to the number of stages. ILP designs allow multiple instructions to be represented in some stages. This necessarily implies the possibility of executing some stages of

successively fetched instructions out of order. For example, it is entirely possible for the n^{th} instruction, I_n, to be constrained from being issued for several clock cycles while the next instruction, I_{n+1}, is not so constrained. An ILP processor will push the evaluation of I_{n+1} along as far as it can without waiting on I_n. However, the *instruction graduate* stage will reimpose order and insist on graduating I_n before I_{n+1}.

ILP CPUs use architectural slight of hand with respect to register useage to accelerate performance. An ILP machine typically has more registers available than appear in the instruction set. Registers named in instructions need not precisely be the registers actually used in the implementation of those instructions. This is acceptable, of course, as long as the effect of the instructions is the same in the end. One factor motivating this design is the possibility of having multiple instructions involving the same logical registers (those named by the instructions themselves) actively being processed concurrently. By providing each instruction with its own "copy" of a register, we eliminate one source of stalls. Another factor involves branches—that is, instructions that interrupt the sequential flow of control. An ILP, encountering a branch instruction, will predict whether the branch is taken or not and possibly alter the instruction stream as a result. Various methods exist to predict branching, but any of them will occasionally predict incorrectly. When an incorrect prediction is made, the register state computed as a result of speculating on branch outcome needs to be discarded and execution resumed at the branch point. Thus, another use of additional registers is to store the "speculative register state." With dedicated hardware resources to track register useage following speculative branch decision, speculative state can be discarded in a single cycle and control resumed at the mispredicted branch point. In all of these cases, the hardware implements techniques for renaming the logical registers that appear in the instructions to physical registers, for maintaining the mapping of logical to physical registers, and for managing physical register useage.

A simulation model of an ILP CPU will model the logic of each stage and coordinate the movement of instructions from stage to stage. We consider each stage in turn.

An instruction-fetch stage could interact with the simulated memory system, if that is present. However, if the CPU simulation is driven by a direct-execution simulation or by a trace file, there is little for a model of this stage to do but get the next instruction in the stream. If a memory system is present, this stage could look into an instruction cache for the next referenced instruction, stalling if a miss is suffered.

Following an instruction fetch, an instruction will be in the CPU's list of active instructions until it exits altogether from the pipeline. The instruction-decode stage places an instruction in this list; a logical register that appears as the target of an operation is assigned a physical register—registers used as operand sources will have been assigned physical registers in instructions that defined their values. (Sequencing issues associated with having multiple representations of the same register are dealt with at a later stage in the pipeline.) Branch instructions are identified in this stage, predictions of branch outcomes are made, and resources for tracking speculative execution are committed here.

Decoded instructions pass into the instruction-issue stage. The logic here is complex and very much timing dependent. An instruction cannot be issued until values in its input registers are available and a functional unit needed to perform the instruction is available. An input value might be not yet in a register, for instance, if that value is loaded from memory by a previous instruction and has not yet appeared. A functional unit could be unavailable because all appropriate ones are busy with multicycle operations initiated by other instructions. Implementation of the issue-stage model (and hardware) depends on marking registers and functional units as busy or pending and on making sure that, when the state of a register or functional unit changes, any instruction that cannot yet issue because of that register or functional unit is reconsidered for issue.

Simulation of the instruction-execute stage is a matter of computing the result specified by the instruction (e.g., an addition). At this point, the action of depositing the result into a register or memory is scheduled for the instruction-complete stage. This latter stage also cleans up the status bits associated with registers and functional units involved in the instruction and resolves the final outcome of a predicted branch. If a branch was mispredicted, the speculatively fetched and processed instructions that follow it are removed

from other pipeline stages, the hardware that tracks speculative instruction is released, and the instruction stream is reset to follow the branch's other decision direction.

Between the instruction-issue and instruction-complete stages, instructions could get processed in an order that does not correspond to the original instruction stream. The last stage, graduation, reorders them. Architecturally, this permits an ILP CPU to associate an exception (e.g., a page fault or a division by zero) with the precise instruction that caused it. Simulation of this stage is a matter of knowing the sequence number of the next instruction to be graduated, then graduating it when it appears.

Example 14.2

An example helps to show what goes on. Consider the following sequence of assembly-language instructions for a hypothetical computer:

```
load $2, 0($6) ; I1-  load $2 from memory
mult $5, 2     ; I2-  multiply $5 with constant 2
add  $4, 12    ; I3-  add constant 12 to $4
add  $5, $2    ; I4-  $5 <- $5 + $2
add  $5, $4    ; I5-  $5 <- $5 + $4
```

Let us suppose that the register load misses the first-level cache but hits in the second-level cache, resulting in a delay of 4 cycles before the register gets the value. Suppose further that separate hardware exists for addition and multiplication, that addition takes one cycle, and that multiplication takes 2 cycles to complete. Time is assumed to advance in units of a single clock tick.

Table 14.3 shows a timeline of when each instruction is in each stage. Cycles in which an instruction cannot proceed through the pipeline are marked as "stall" cycles. Processing is most easily understood by tracing individual instructions through.

I1. After being fetched in cycle 1, the decode of I1 assigns physical register $p1 as the target of the load operation and marks $p1 as unready. No constraints prohibit I1 from being issued in cycle 3 nor executed in cycle 4. Because the memory operation takes 4 cycles to finish, I1 is stalled in cycles 5–8. Cycle 9 commits the data from memory to physical register $p1 and clears its unready flag; the instruction is graduated in cycle 10.

I2. Instruction I2 is fetched in cycle 2 and has physical register $p2 allocated to receive the results of the multiplication in the cycle-3 decode stage; $p2's unready flag is raised. No constraints keep I2 from

Table 14.3 Pipeline Stages, ILP CPU Simulation

Inst./Cycle	1	2	3	4	5	6	7
I1	fetch	decode	issue	execute	stall	stall	stall
I2		fetch	decode	issue	execute	stall	complete
I3			fetch	decode	issue	execute	complete
I4				fetch	decode	stall	stall
I5					fetch	decode	stall
Inst./Cycle	**8**	**9**	**10**	**11**	**12**	**13**	**14**
I1	stall	complete	graduate				
I2	stall	stall	stall	graduate			
I3	stall	stall	stall	stall	graduate		
I4	stall	stall	issue	execute	complete	graduate	
I5	stall	stall	stall	stall	stall	issue	complete

being issued in cycle 4 or executed in cycle 5, but the 2-cycle delay of the multiplier means the result is not committed to register $p2 until cycle 7, at which point the $p2 unready flag is cleared. The instruction remains stalled through cycles 8–10, awaiting the graduation of I1.

I3. Instruction I3 is fetched in cycle 3 and has physical register $p3 allocated to receive the results of its addition in the cycle-4 decode stage. The $p3 unready flag is raised. There are no constraints keeping I3 from being issued in cycle 5 and executed in cycle 6, with results written into $p3 in cycle 7, at which time $p3's unready flag is cleared. I3 must stall, however, during cycles 8–11, awaiting the graduation of I2.

I4. Instruction I4 is fetched in cycle 4 and has physical register $p4 allocated to receive the results of the addition during the cycle-5 decode stage. $p4's unready flag is raised at that point. Physical registers $p1 and $p2 are operands to the addition; I4 stalls in cycles 6–9, waiting for their unready flags to clear. It then passes the remaining stages without further delay, clearing the $p4 unready flag in cycle 12.

I5. Instruction I5 is fetched in cycle 5 and has physical register $p5 allocated to receive the results of its addition in the cycle-6 decode stage, at which point the $p5 unready flag is set. Physical registers $p3 and $p4 contain the addition's operands; I5 stalls through cycles 7–12, waiting for their unready flags both to clear. From that point forward, I5 passes through the remaining stages without further delay.

The performance benefit of pipelining and ILP can be appreciated if we compare the execution time of this sequence on a nonpipelined, non-ILP machine. Assuming that each stage must be performed for each instruction but that one instruction is processed in its entirety before another one begins, 51 cycles are needed to execute I1 through I5. With the advanced architectural features, only 15 cycles are needed. The example illustrates both the parallelism that pipelining exposes and the latency tolerance that the ILP design supports. Even though I1 stalls for four cycles while awaiting a result from memory, the pipeline keeps moving other instructions through to some extent. The bottom line for someone using a model like this is the rate at which instructions are graduated, as this reflects the effectiveness of the CPU design. Secondary statistics would try to pinpoint where in the design stalls occur that might be alleviated (e.g., if many stalls occur because of waiting for the multiplier (no such stalls occur in the example), then one could consider including an additional multiplier in the CPU design).

Our explanation of the model's workings was decidedly process oriented, taking the view of an instruction. However, the computational demands of a model like this are enormous, owing to the very large number of instructions that must be simulated to assess the CPU design on, say, a single program run. The relatively high cost of context switching would deter use of a normal process-oriented language. One could implement what is essentially a process-oriented view by using events—each time an instruction passes through a stage, an event is scheduled to take that instruction through the next stage, accounting for stalls. The amount of simulation work accomplished per event is thus the amount of work done on behalf of one instruction in one stage. An alternative approach is to eschew explicit events altogether and simply use a cycle-by-cycle activity scan. At each cycle, one would examine each active instruction to see whether any activity associated with that instruction can be done. An instruction that was at one stage at cycle j will, at cycle $j + 1$, be examined for constraints that would keep it at cycle j. Finding none, that instruction would be advanced to the next stage. An activity-scanning approach has the attractiveness of eliminating event-list overhead, but the disadvantage of expending computational effect on checking the status of a stalled instruction on every cycle during which it is stalled. Implementation details and model behavior largely determine whether an activity-scanning approach is faster than an event-oriented approach (with the nod going to activity scanning when few instructions stall).

14.6 MEMORY SIMULATION

One of the great challenges of computer architecture is finding ways to deal effectively with the increasing gap in operation speed between CPUs and main memory. A factor of 100 in speed is not far from the mark.

The main technique that has evolved is to build hierarchies of memories. A relatively small memory—the L1 cache—operates at CPU speed. A larger memory—the L2 cache—is larger and operates more slowly. The main memory is larger still and slower still. The smaller memories hold data that was referenced recently and nearby data that one hopes will also be referenced soon. Data moves up the hierarchy on demand and ages out as it becomes disused, to make room for the data in current use. For instance, when the CPU wishes to read memory location 100,000, hardware will look for it in the L1 cache; if it fails to find it there, it will look in the L2 cache. If it is found there, an entire block containing that reference is moved from the L2 cache into the L1 cache. If it is not found in the L2 cache, a (larger) block of data containing location 10,000 is copied from the main memory to the L2 cache, and part of that block (containing location 10,000 of course) is copied into the L1 cache. It could take 50 cycles or more to accomplish this. After this cost has been suffered, the hope and expectation is that the CPU will continue to make references to data in the block brought in, because accesses to L1 data are made at CPU speeds. Fortunately, most programs exhibit locality of reference at this scale (as well as at the paging scale discussed earlier in the chapter), so the strategy works. However, after a block ceases to be referenced for a time, it is ejected from the L1 cache. It could remain in the L2 cache for a while and later be brought back into the L1 cache if any element of the block is referenced again. Eventually a block remains unreferenced long enough so that it is ejected also from the L2 cache.

The astute reader will realize that data that is written into an L1 cache by the CPU creates a consistency problem, in that a memory address then has different values associated with it at different levels of the memory hierarchy. One way of dealing with this is to write through to all cache levels every time there is a write—the new value is asynchronously pushed from L1 through L2 to the main memory. An alternative method copies back a block from one memory level to the lower level, at the point the block is being ejected from the faster level. The write-through strategy avoids writing back blocks when they are ejected, whereas the write-back strategy requires that an entire block be written back when ejected, even if only one word of the block was modified, once. One of the roles simulation plays is to compare performance of these two write-back strategies, taking into consideration all costs and contention for the resources needed to support writing back modifications.

Like paging systems, the principle measure of the quality of a memory hierarchy is its hit ratio at each level. As with CPU models, to evaluate a memory hierarchy design, one must study the design in response to a very long string of memory references. Direct-execution simulation can provide such a reference stream, as can long traces of measured reference traffic. Nearly every caching system is a demand system, which means that a new block is not brought into a cache before a reference is made to a word in that block. Decisions left still to the designer include whether to write-through or write-back modifications, the replacement policy, and the "set associativity."

The concept of set associativity arises in response to the cost of the mechanism used to look for a match. Imagine we have an L2 cache with 2 million memory words (an actual figure from an actual machine). The CPU references location 10000—the main memory has, say, 2^{12} words, so the L2 cache holds but a minute fraction of the memory. How does the hardware find out whether location 10000 is in the L2 cache? It uses what is called an associative memory, one that associates search keys with data. One queries an associative memory by providing some search key. If the key is found in the memory, then the data associated with the key is returned; otherwise, indication of failure is given. In the caching context, the search key is derived from the reference address, and the return data is the data stored at that address. Caches must be very very fast, which means that the search process has to be abbreviated. This is accomplished by dedicating comparison hardware with every location in the associative memory. Presented with a search key, every comparator looks for a match with the key at its location. At most one comparator will see a match and return the data; it is possible that none will. A *fully associative* cache is one where any address can appear anywhere in the cache. This means building the cache to have a unique comparator associated with every address in the cache; doing so is prohibitively expensive. Tricks are played with memory addresses in order to reduce the costs greatly. The idea is to partition the address space into sets. Figure 14.11 illustrates how a 48-bit

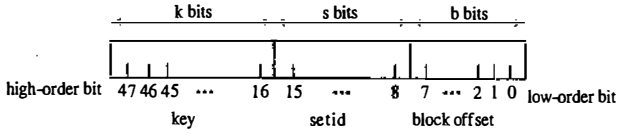


Figure 14.11 48-bit address partitioned for cache.

memory address might be partitioned in key, set id, and block offset. Any given memory address is mapped to the set identified by its set-id address bits. This scheme assigns the first block of 2^b addresses to set 0, the second block of 2^b addresses to set 1, and so on, wrapping around back to set 0 after 2^s blocks have been assigned. Each set is give a small portion of the cache—the set size—typically, 2 or 4 or 8 words. Only those addresses mapped to the same set compete for storage in that space. Only as many comparators are needed as there are words in the set. Given an address, the hardware uses the set-id bits to identify the set number and the key bits to identify the key. The hardware matches the keys of the blocks already in the identified set to comparator inputs and also provides the key of the sought address as input to all the comparators. Comparisons are made in parallel; in the case of a match, the block-offset bits are used to index into the identified block to select the particular address being referenced.

The overall size of this cache is seen to be the total number of sets times the set size. One role of simulation is to work out, for a given cache size, how the space ought to be partitioned into sets. This is largely a cost consideration, for increasing the set size (thereby reducing the number of sets) typically increases the hit ratio. However, if a set size of 4 yields a sufficiently large hit ratio, then there is little point to increasing the set size (and cost).

Least Recently Used (LRU) is the replacement policy most typically used. When a reference is made but is not found in a set, some block in the set is ejected to make room for the one containing the new reference. Under LRU, the block selected for ejection is the one which, among all blocks in the set, was last referenced most distantly in the past.

LRU is one of several replacement policies known as *stack* policies. (See Stone [1990].) These are characterized by the behavior that, for any reference in any reference string, if that reference misses in a cache of size n, then it also misses in every cache of size $m < n$, and that, if it hits in a cache of size m, then it hits in every cache of size $n > m$. Simulations can exploit this fact to compute the miss ratio of many different set sizes, in just one pass of the reference string! Suppose that we do not wish to consider any set size larger than 64. Now we conduct the simulation with set sizes of 64. Every block in the cached set is marked with a priority—namely, the temporal index of the last reference made to it (e.g., the block containing the first reference in the string is marked with 1, the block containing the second reference is marked with a 2 (overwriting the 1, if the same as the previous block), etc.). When a block must be replaced, the one with the smallest index is selected. Imagine that the simulation organizes and maintains the contents of a cached set in LRU order, with the most recently referenced block first in the order. The *stack distance* of a block in this list is its distance from the front; the most recently referenced block has stack distance 1, the block referenced next most recently has stack distance 2, the LRU block has stack distance 64. Presented with a reference, the simulation searches the list of cache blocks for a match. If no match is found, then, by the stack property, no match will be found in any cache of a size smaller than 64, on this reference, for this reference string. If a match is found and the block has stack distance k, then no match will be found in any cache smaller than size k, and a match will always be found in a cache of size larger than k. Rather than record a hit or miss, one increments the k^{th} element of a 64-element array that records the number of matches at each LRU level. To find out how many hits occurred in a cache of size n, one sums up the counts of the first n elements of the array. Thus, with a little arithmetic at the end of the run, one can count (for each set cache) the number of hits for every set of every size between 1 and 64.

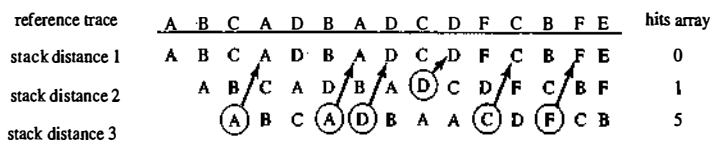


Figure 14.12 LRU stack evolution

Figure 14.12 illustrates the evolution of an LRU list in response to a reference string. Under each reference (given as a alphabetic symbol rather than actual memory address) is the state of the LRU stack *after* the reference is processed. The horizontal direction from left to right symbolizes the trace, read from left to right. A hit is illustrated by a circle, with an arrow showing the migration of the symbol to the top of the heap. The "hits" array counts the number of hits found at each stack distance. Thus we see that a cache of size 1 will have the hit ratio 0/15, a cache of size 1 will have the hit ratio 1/15, and a cache of size 3 will have the hit ratio 6/15.

In the context of a set-associative cache simulation, each set must be managed separately, as shown in the figure. In one pass, one can get hit ratios for varying set sizes, but it is important to note that each change in set size corresponds to a change in the overall size of the entire cache. This technique alone does not let us in one pass discover the hit ratios for all the different ways one might partition a cache of a given capacity (e.g., 256 sets with set size 1 versus 128 sets with set size 2 versus 64 sets with set size 4). It actually is possible to evaluate all these possibilities in one pass, but the technique is beyond the scope of this discussion.

14.7 SUMMARY

This chapter looked at the broad area of simulating computer systems. It emphasized that computer-system simulations are performed at a number of levels of abstraction. Inevitably, it discussed a good deal of computer science along with the simulation aspects, for in computer-systems simulation the two are inseparable.

The chapter outlined fundamental implementation issues behind computer-system simulators—principally, how process orientation is implemented and how object-oriented concepts such as inheritance are fruitfully employed. Next it considered model input, ranging from stochastically generated traffic, to stochastically generated memory-referencing patterns, to measured traces and direct-execution techniques. The chapter was brought to a conclusion by looking at examples of simulation at different levels of abstraction: a WWW-site server system, an instruction-level CPU simulation, and simulation of set-associative memory systems.

The main point is that computer-system simulators are tailored to the tasks at hand. Appropriate levels of abstraction need to be chosen, as must appropriate simulation techniques.

REFERENCES

ASHENDEN, P.J. [2001], *The Designer's Guide to VHDL*, 2d ed., Morgan Kaufmann, San Fransisco.

AUSTIN T., E. LARSON, AND D. ERNST [2002], "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, Vol. 35, No. 2, pp. 59–67.

BEDICHECK, R.C. [1995], "Talisman: Fast and Accurate Multicomputer Simulation," *Proceedings of the 1995 ACM SIGMETRICS Conference*, pp. 14–24, Ottawa, ON, May.

BOHRER P., M. ELNOZAHY, A. GHEITH, G. LEFURGY, T. NAKRA, J. PETERSON, R. RAJAMONY, R. ROCKHOLD, H. SHAFI, R. SIMPSON, E. SPEIGHT, K. SUDEEP, E. VAN HENSGERGEN, AND L. ZHANG [2004], "Mambo—A Full System Simulator for the PowerPC Architecture," *Performance Evaluation Review*, Vol. 31, No. 4, 8–12.

CMELIK, B., AND D. KEPPLE [1994], "Shade: A Fast Instruction-Set Simulator for Execution Profiling," *Proceedings of the 1994 ACM SIGMETRICS Conference*, pp. 128–137, Nashville, TN, May.

COVINGTON, R., S. DWARKADA, J. JUMP, S. MADALA, AND J. SINCLAIR [1991], "Efficient Simulation of Parallel Computer Systems," *International Journal on Computer Simulation*, vol. 1, No. 1, 1991.

COWIE, J., A. OGIELSKI, AND D. NICOL [1999], "Modeling the Global Internet," Vol. 1, No. 1, pp. 42–50.

DICKENS, P., P. HEIDELBERGER, AND D. NICOL [1996], "Parallelized Direct Execution Simulation of Message Passing Programs," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 10, pp. 1090–1105.

FISCHER, W., AND K. MEIER-HELLSTERN [1993], "The Markov-Modulated Poisson (MMPP) Cookbook," *Performance Evaluation*, Vol. 18, No. 2, pp. 149–171.

FISHWICK, P. [1992], "SIMPACK: Getting Started with Simulation Programming in C and C++," *Proceedings of the 1992 Winter Simulation Conference*, pp. 154–162, Washington, DC.

GRUNWALD, D. [1995], *User's Guide to Awesime-II*, Department of Computer Science, Univ. of Colorado, Boulder, CO.

HENNESSY, J.L., AND D.A. PATTERSON [1997], *Computer Organization and Design, The Hardware/Software Interface*, 2d ed., Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

HUGHES, C., V. PAI, P. RANGANATHAN, AND S. ADVE [2002], "RSIM: Simulating Shared Memory Multiprocessors with ILP Processors," *IEEE Computer*, Vol. 35, No. 2, pp. 40–49.

LEBECK, A., AND D. WOOD [1997], "Active Memory: A New Abstraction for Memory System Simulation," *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 1, pp. 42–77.

LITTLE, M.C., AND D.L. McCue [1994], "Construction and Use of A Simulation Package in C++," *C User's Journal*, Vol. 3, No. 12.

MAGNUSSON, P., M. CHRISTENSSON, J. ESKILSON, D. FORSGREN, G. HALLBER, J. HOGBERG, F. LARSSON, A. MOESTEDT, AND B. WERNER [2002], "SIMICS: A Full System Simulation Platform," *IEEE Computer*, Vol. 35, No. 2, pp. 50–58.

MANO, M. [1993], *Computer Systems Architecture*, 3d ed., Prentice Hall, Englewood Cliffs, NJ.

NUTT, G. [2004], *Operating Systems, A Modern Prespective*, 3d ed., Addison-Wesley, Reading MA.

SCHWETMAN, H. [1986], "CSIM: AC-Based, Process-Oriented Simulation Language," *Proceedings of the* 1986 *Winter Simulation Conference*, pp. 387–396.

SCHWETMAN, H.D. [2001], "CSIM 19: A Powerful Tool for Building Systems Models," *Proceedings of the* 2001 *Winter Simulation Conference*, pp. 250–255.

STONE, H. [1990], *High Performance Computer Architecture*, Addison-Wesley, Reading MA.

WITCHEL, E., AND M. ROSENBLUM [1996], "EMBRA: Fast and Flexible Machine Simulation," *Proceedings of the 1996 ACM SIGMETRICS Conference*, pp. 68–79, Philedelphia, PA, May.

EXERCISES

1. Sketch the logic of an event-oriented model of an M/M/1 queue. Estimate the number of events executed when processing the arrival of 5000 jobs. How many context switches on average does a process-oriented implementation of this queue incur if patterned after the SSF implementation of the single-server queue in Chapter 4?

2. For each of the systems listed, sketch the logic of a process-oriented model and of an event-oriented model. For both approaches, develop and simulate the model in any language:

   - a central-server queueing model: when a job leaves the CPU queue, it joins the I/O queue with shortest length.
   - a queueing model of a database system, that implements fork join: a job receives service in two parts. When it first enters the server it spends a small amount of simulation time generating a random number of requests to disks. It then suspends (freeing the server) until such time as *all* the requests it made have finished, and then enqueues for its second phase of service, where it spends a larger amount of simulation time, before finally exiting. Disks may serve requests from various jobs concurrently, but serve them using FCFS ordering. Your model should report on the statistics of a job in service—how long (on average) it waited for phase 1, how long it waits on average for its I/O requests to complete, and how long it waits on average for service after its I/O requests complete.

3. Consider a three-state (OFF, ON, and BURSTY) Markov Modulated Process with the following characteristics
   (a) The MMP is in ON state for 90% of the time on average.
   (b) The MMP is in BURSTY state for 5% of the time on average.
   (c) OFF to ON transitions probability is 0.8 and OFF to BURSTY is 0.05.
   (d) ON to OFF transition probability is 0.9 and ON to BURSTY is 1.
   (e) BURSTY to ON transition probability is 0.5 and BURSTY to OFF is 0.5.

   If the time spent in OFF state is exponential with a mean of 0.3, determine exponential mean values of time spent in ON and BURSTY states by means of simulation.

4. Recall the pseudo-code for generating reference traces (Figure 14.7). Write routines `new_wrkset`, `from_wrkset`, and `not_from_wrkset` to model the following types of programs:
   (a) a scientific program with a large working set during initialization, a small working set for the bulk of the computation, and a different working set to complete the computation. (You will need to modify the control code in the figure slightly to force phase transitions in desired places);
   (b) a program whose working set always contains a core set of pages present in every phase, with the rest of the pages clustered elsewhere in the address space.

5. Consider computer network with three printers (a, b, and c). The type of printer (a or b or c) is selected by the user and some users are high-priority users. Simulate the model using any simulator or language.

6. Using any simulator or language you like, model the router-to-Web-server logic of the system described in section 14.1. Pay special attention to the load-balancing mechanism that the router employs.

7. Using any simulator or language you like, model the interaction between application server and data server described in section 14.4. Pay special attention to the logic of requesting multiple data services and of waiting until all are completed until advancing to the next burst.

8. Consider the following language for describing CPU instructions:

```
op r1 r2
```

   The preceding expressions describe an operation, where

```
op=1  means add, op=2 means subtract. Each require 1 cycle.
op=3  means mult. r1 receives the result r1 op r2. A multiplication requires 2
      cycles.
op=4  means a load from memory, into r1, using the value in r2 as the memory
      address. Every 10th load requires 4 cycles, the remaining loads require 1.
op=5  means a store to memory, storing the data found in r1, using the value in
      r2 as the memory address. Each store requires 1 cycle.
```

   Write a CPU simulation along the lines of that described in 14.5 that accepts a stream of instructions in the format just described. Your simulator should use a logical-to-physical register mapping, use the timing information previously sketched, and use stall instructions as described in the example.

9. Integrate the trace generator created in Problem 4 with the one-pass simulator written in the previous problem, in effect creating a pseudo "direct-execution simulator."

10. Analyze the log of WWW requests to your site's server, produce a stochastic model of the request stream, and simulate it.

15

Simulation of Computer Networks

15.1 INTRODUCTION

Computers and the networks that connect them have become part of modern working life. In this chapter, we illustrate by example some of the ways that discrete-event simulation is used to understand network systems, the software that controls them, and the traffic that they carry.

Like computer systems, network systems exhibit complexity at multiple layers. Networked systems are designed (with varying degrees of fidelity) in accordance with the so-called Open System Interconnection (OSI) Stack Mode (Zimmerman, 1980). The fundamental idea is that each layer provides certain services and guarantees to the layer above it. An application or protocol at a particular layer communicates only with protocols directly above and below it in the stack, implementing communication with a corresponding application or protocol at the same stack layer in a different device. Simulation is used to study behavior at all these layers, although not generally all in the same model. Different layers encapsulate different levels of communication abstraction.

The **Physical Layer** is concerned with the communication of a raw bit-stream, over a physical medium. The specification of a physical layer has to address all the physical aspects of the communication: voltage or radio signal strength, standards for connecting a physical device to the medium, and so on. Models of this layer describe physics.

The **Data Link Layer** implements the communication of so-called *data-frames*, which contain a limited chunk of data and some addressing information. Protocols at the Data Link Layer interact with the physical layer to send and receive frames, but also provide the service of "error-free" communication to the layer above it. Protocols at the Data Link Layer must therefore implement error-detection and retransmission when needed. A critical component of avoiding errors is access control, which ensures that at most one device is transmitting at a time on a shared medium. Techniques for access control have significant impact on how long it takes to deliver data and on the overall capacity of the network to move data. Simulation plays an

important role in understanding tradeoffs between access-control techniques; in this chapter, we will look at some protocols and the characteristics that simulation reveals.

The **Network Layer** is responsible for all aspects of delivering data frames across subnetworks. A given frame may cross multiple physical mediums en-route to its final destination; the Network Layer is responsible for logical addresses across subnets, for routing across subnets, for flow control, and so on. The success of the Internet is due in no small part to widespread adaption of the *Internet Protocol*, more commonly known as IP (Comer, 2000). IP specifies a global addressing scheme that allows communication between devices across the globe. The specification of IP packets includes fields that describe the type of data being carried in the packet, the size of the packet, the protocol suitable for interpreting the packet, source/destination addressing information, and more. The Network Layer provides error-free end-to-end delivery of packets to the layer above it. Simulation is frequently used to study algorithms that manage devices (routers) that implement the Network Layer.

The **Transport Layer** accepts a message from the layer above, segments it into packets that are passed to the Network Layer for transmission, and provides the assurance that received packets are delivered to the layer above in the order in which they appear in the original message, error free, without loss, and without duplication. Thus, the Transport Layer protocol in the sending device coordinates with the Transport Layer protocol in the receiving device in such a way that the receiving device can infer packet-order information. Variants of the Transmission Control Protocol (TCP) are most commonly used at this layer of the stack (Comer, 2000). Dealing with packet loss is the responsibility of the transport layer. Packet loss is distinct from error-free transmission—a packet could be transmitted to a routing device without error, only to find that device does not have the buffering capacity to store it; the packet is received without error, but is deliberately dropped. Transport layer protocols need to detect and react to packet loss, because they're responsible for replacing the packets that are dropped. One of the ways they do this is to apply flow-control algorithms that simultaneously try to utilize the available bandwidth fully, yet avoid the loss of packets. Simulation has historically played a critical role in studying the behavior of different transport protocols, and in this chapter we will examine simulation of TCP.

The first four OSI layers are well defined and separated in actual implementation. The remaining three have not emerged so strongly in practice. Officially the **Session Layer** is responsible for the creation, maintenance, and termination of a "session" abstraction, a session being a prolonged period of interaction between two entities. Above this one finds the **Presentation Layer**, whose specification includes conversion between data formats. An increasingly important conversion function is encryption/decryption. Finally, the **Application Layer** serves as the interface between users and network services. Services typically associated with the Application Layer include email, network management tools, remote printer access, and sharing of other computational resources.

Any simulation of networking must include models of data traffic, and so we begin the discussion there. At the time of this writing, the field of traffic modeling is very active, and we bring to the discussion key elements of an exciting area of current work.

Devices with traffic they wish to transmit must somehow gain access to the networks that carry traffic. Our second area of discussion then considers the problem of how devices coordinate to use the network medium, sometimes called Media Access Control (MAC) protocols. Historically, simulation has played an important role in helping engineers to understand the performance of different MAC protocols.

Finally, we describe the Transport Control Protocol (TCP) and discuss how simulation plays an important role in its study.

15.2 TRAFFIC MODELING

Our discussion of network simulation begins with modeling of the data traffic that the networks carry. We'll consider two levels of detail for this, corresponding to two different levels of abstraction. The first is at the

application level: consideration of how commonly used network applications create demand for a network. Such models are appropriate when one's interest is in the details of a relatively small network and the impact that its native applications have on it. The second level of abstraction is of aggregated application flows. This level is appropriate when one's focus is on the Internet's core infrastructure, where the global impact of global traffic needs to be represented.

One of the easiest models of traffic-load generation is that of moving files across the network. Our interest here is not in the mechanics of the protocols that accomplish the movement so much as it is in the model of the traffic load that is offered to the network. Simulation studies that model file transfer typically are focused on the impact that the traffic has on servers holding the files. A given transfer can be characterized by the size of the file and by the rate at which its bytes are presented to the network. We usually also characterize how often a user initiates an ftp transfer. A simple model of a file-transfer request process is as an on–off source, whose *off* period is randomly distributed (e.g., an exponential think time) and whose *on* period is driven by the arrival of a file. The *on* period lasts as long as needed to push or pull a file of the referenced length. File size is sampled from another probability distribution. Measurements suggest that a heavy-tailed distribution is appropriate. This is especially appropriate given the level of music-sharing activity on the Internet.

Another significant source of application traffic is the World Wide Web. Traffic associated with web pages is more complex than individual file transfers and so bears separate treatment. We describe a model expounded upon in (Barford and Crovella [1998]), called Surge. Here we model the delay between successive sessions with an intersession delay distribution. Within a session, a number of different URLs will be accessed, with another delay time between each such access; this is illustrated in Figure 15.1.

The Surge model incorporates a number of important characteristics of files, most importantly, including

- the distribution of file sizes, among all files on a web server,
- the distribution of the file sizes of those files that are actually *requested*,
- temporal locality of file-referenced file.

The first and third characteristics, coupled with a model of referencing pattern, essentially define the second characteristic. Suppose that we've selected the first k files already—call them $f_1, f_2, \ldots, f_k$—and suppose that this set of references is organized in a Least-Recently-Used stack. We select the $(k+1)^{st}$ file by sampling an integer from a stack-distance distribution. If that sample has value j, the next file selected has position j in the LRU stack (position 1 being the last file referenced). Empirical studies of reference strings of files suggest that a lognormal distribution is appropriate. This distribution places significant weight on small values; hence, it induces temporal locality of reference. When the stack-distance sample is larger than the number of files in the LRU stack, a new file is sampled from the set of files not yet in the reference stream.

This description gives a general, but simplistic idea of the structure of Surge. Its authors pay much attention to issues of identifying distributional parameters that are internally consistent and that produce traffic that can be validated against real traffic. Our goal here is to introduce the fundamental notions behind a model of web traffic.

Models of other interesting and important application types can be found in the literature. We expect that the Internet will increasingly support telephony—"voice over IP (VoIP)" (Black [2001]), and so attendant models should be developed. A sampling of the current literature suggests that a VoIP source be

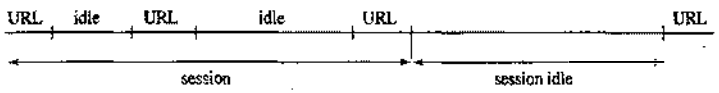


Figure 15.1 Nested on–off periods in Surge WWW traffic generation.

modeled as an on–off process, where both phases have distributions with tails somewhat heavier than exponential (e.g., an appropriate Weibull). Increasingly, the Internet will be used to stream video content. Models for video are more complex, because they must capture a number of facets of video compression, at different time-scales.

All of the application models we've considered describe the traffic workload offered to a network by individual programs. There are contexts in which a modeler needs instead to consider the impact of aggregated application flows on a network device. One *could* create the aggregate stream by piecing together many individual application streams—or one could start with an aggregated model in the first place. We next consider direct models of aggregated offered load.

Classical models of telephone traffic assume that aggregated call arrivals to the telephony network follow a Poisson distribution and that call completions likewise are Poisson. The early days of modeling and engineering *data* networks made the same assumption. However, with time, it became clear that this assumption didn't match reality well. In telephony, the increased use of faxes, and then Internet connections, radically transformed the statistical behavior of traffic. Two things emerged as being particularly different: First, data traffic exhibits a burstiness that flies in the face of the exponential's memoryless property. MMP processes described in Chapter 14 can be used to introduce burstiness explicitly into the arrival pattern of packets to a data network. However, studies indicate that the durations of burstiness aren't Markovian, as in the MMP model. Instead, traffic seems to exhibit long-term temporal dependence—correlations in the number of active sessions that extend past what, statistically, can be expected from MPP models.

Researchers noticed that there is tremendous variance in the size of files transferred within a session. It seemed that a heavy-tailed distribution like the Pareto does a good job of capturing this spread. Heavy-tailed distributions have the characteristic that, infrequently, very very large samples emerge. These large samples are large enough relative to their probability to exert a very significant influence on the moments of the distribution; in some cases, the integral defining variance diverges. It was hypothesized then that long-range dependence in session counts was due to the correlations induced by the concurrency of very long-lived sessions.

A model that appears to capture these explanations is the "Poisson Pareto Burst Process" (Zukerman *et al.* [2003]), in which bursts (e.g., sessions) of traffic arrive as a Poisson process. Each session length has duration sampled from a Pareto distribution. Bursts may be concurrent. More formally, let t_i be the arrival time of the ith burst, equal to $t_{i-1} + e_i$ where e_i is sampled from an exponential, and let b_i be the Pareto-sampled duration of that burst, and let $d_i = t_i + b_i$ be the finishing time of the ith burst. The state at t, $X(t)$, is the number of bursts i, with $t_i \le t \le d_i$.

The Pareto distribution with parameters a and b has the probability distribution function

$$D(x) = 1 - \left(\frac{b}{x}\right)^a$$

for $x \ge b$. The distribution has mean $(ab)/(a-1)$ and variance $ab^2/((a-1)^2(a-2))$. One can sample a Pareto with these parameters, using the inverse transform technique:

$$x = b \times (1.0 - U)^{-1.0/a}$$

In this equation U is a uniformly distributed random variable.

It is instructive to consider how traffic is analyzed for evidence of long-range dependence and whether the style of synthetic traffic generation described here exhibits it. Let $X_1, X_2, \ldots$, be a stationary time series, whose samples have mean μ and variance σ^2. The autocorrelation function $\rho(k)$ describes how well correlated are samples k apart in the time series:

$$\rho(k) = \frac{E[(X_t - \mu)(X_{t+k} - \mu)]}{\sigma^2}$$

The sample autocorrelation function can be constructed from an actual sample by estimating the expectation in the numerator. Long-range dependence is observed when $\rho(k)$ decays slowly as a function of k. Long-range dependence is more formally defined in terms of the autocorrelation function, if there exists a real number $\alpha \in (0, 1)$, and a constant $\beta > 0$ such that

$$\lim_{k \to \infty} \frac{\rho(k)}{\beta k^{-\alpha}} = 1$$

The denominator of this limit describes how slowly $\rho(k)$ needs to go to zero as k increases. The smaller α is, the slower is the degradation. $H = 1 - \alpha/2$ is known as the *Hurst parameter* for the sequence. Values of H with $0.5 < H < 1.0$ define long-range dependence; the larger H is, the more significant is the long-range dependence.

To see evidence that PPBP does yield long-range dependence, we ran an experiment where the mean burst interarrival time was 1 second and the Pareto parameters were $a = 1.1$ and $b = 10$. We computed the sample autocorrelation function, shown in Figure 15.2. Here we see directly that the autocorrelation decays very slowly. We also used the SELFIS tool (Karagiannis *et al.* [2003]) to estimate the Hurst parameter; all of its estimators indicate strong long-range dependence in the sampled series.

Burstiness is not the only consideration in traffic modeling. Traffic intensity exhibits a strong diurnal characteristic—that is, source intensity varies with the source's time of day; furthermore, weekends and holidays behave differently still. To accommodate time-of-day considerations, one can allow the exponential burst interarrival distribution of the PPBP to have a parameter that is dependent on the time of day.

The PPBP describes the number of active sessions $X(t)$ as a function of time. $X(t)$ may be transformed into packet arrival rates, and hence into packets, by including a packet-rate parameter λ. The process $\lambda X(t)$ thus gives an arrival rate of packets from an aggregated set of sources to a network device that handles such.

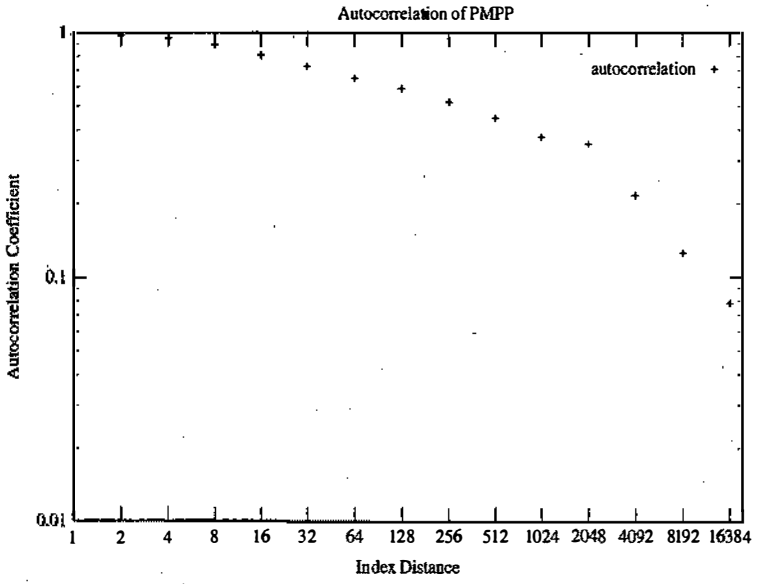


Figure 15.2 Autocorrelation function of aggregated stream of 50 sources.

The main point to be understood about traffic-modeling is that models of aggregated traffic ought to exhibit characteristics of aggregation, whereas application traffic ought to focus on what makes the applications distinct. Next, we look at how traffic acquires a shared medium for carrying traffic.

15.3 MEDIA ACCESS CONTROL

Computers in an office or university environment are usually integrated into a local area network (LAN). Computers access the network through cables (a.k.a. wireline), although an increasing fraction access it through radio (wireless). In either case, when a computer wishes to use the network to transmit some information, it engages in a Media Access Control (MAC) protocol.

Different MAC protocols give traffic different characteristics. Simulation is an extremely important tool for assessing the behavior of a given protocol. A MAC protocol gives traffic specific qualities of latency (average and maximum are usually interesting) and throughput. The behavior of these qualities as a function of "offered load" (traffic intensity) is of critical interest, for some protocols allow throughput to actually *decrease* as the demands on the network go up—a lose–lose situation.

15.3.1 Token-Passing Protocols

One class of MAC protocols is based on the notion of a "token," or permission to transmit. In the "polling protocol" variation, a master controller governs which device on the shared medium may transmit (Kurose and Ross [2002]). The controller selects a network device and sends it the token. If the recipient has "frames" (the basic unit of transmission) buffered up, it sends them, up to a maximum number of frames. The controller listens to the network and detects when the token holder either has selected not to transmit or has finished transmission. The controller then selects another network device and sends it the token. Devices are visited in round-robin fashion.

One drawback of the polling protocol is that the controller is a device with separate functionality from the others. A more homogeneous approach is achieved by using a *token bus* protocol. In this approach a device is programmed to transmit frames (again up to a maximum number) when it receives a token, but is programmed to pass the token directly to a different specified network device after it is finished. There is no controller; the network devices pass the token among themselves, effectively creating a decentralized round-robin polling scheme.

A drawback of both types of token-passing protocol is that a single failure can stop the network in its tracks—in the case of the polling protocol, the network stops if the controller dies; in the case of the token bus, a token passed to a dead device in effect gets lost. In the latter case, one can detect that a device failed to pass the token on and so amend the protocol to deal with like failures.

Token-passing networks are "fair," in the sense that each device is assured its turn within each round. The overhead of access control is the time that the network spends on transmitting the token (rather than data) and the time that the network is idle long enough for a device to ascertain that a transmission has ended or is not going to occur. An important characteristic of token-passing protocols is that the throughput (bits per second of useful traffic) is monotone nondecreasing as a function of the "offered load" (traffic that the network is requested to carry). To illustrate this point, Figure 15.3 plots data from a set of experiments on a modeled 10 Mbits (10 million bits per second) network, with 10 devices, evenly spaced, with a latency delay of 25.6 μsec between the most distant pair. (We use this figure in order to compare this network with one managed by using Ethernet, later.) Five different experiments are displayed on the graph; right now, we are interested only in the one labeled "token bus, Poisson." The experiments assume that the data frame is 1500 bytes long and that the token is 10 bytes long. They assume that, once a device gains the token, it may send at most one frame and then must release the token. This set of data uses a Poisson process to generate frame

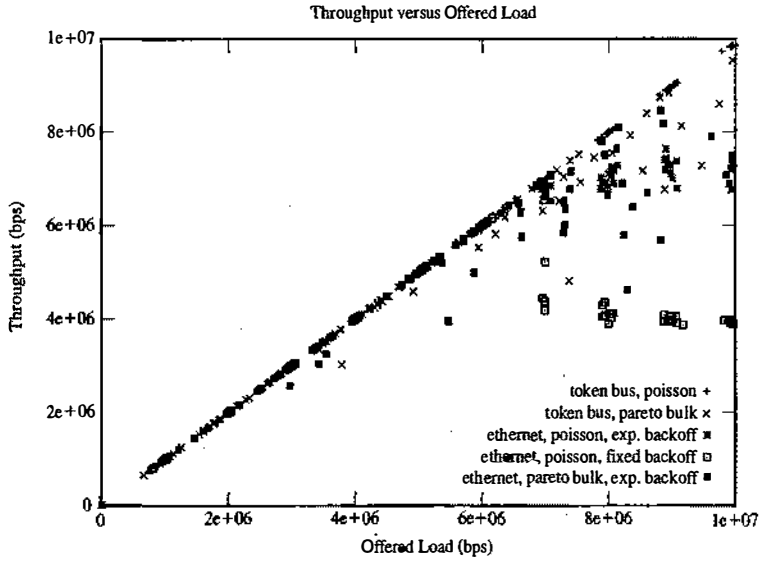


Figure 15.3 Throughput versus Offered Load, for Token Bus and Ethernet MAC protocols, Poisson and Bulk Pareto arrival processes, Exponential and Fixed Backoff (for Ethernet).

arrivals. The x-axis gives the "offered load," measured here as the total sum of bits presented to the network before the simulation end time, divided by the length of the simulation run. The y-axis plots the measured throughput. For each off-time rate, we run 10 independent experiments. For each experiment, we plot the observed pair (offered load, throughput). For the experiment of interest, the throughput increases linearly with the offered load, right up to network saturation. It is interesting to note, though, the impact of a change in the traffic-arrival pattern. We replaced the Poisson arrival process with the arrival process that defines an PPBP, a Poisson bulk arrival process, where the number of frames in each bulk arrival is a truncated Pareto. We use the same Pareto parameters as before ($a = 1.1$ and $b = 10$) and reduced the arrival by a factor of the inverse Pareto mean ($(a-1)/(ab)$) to obtain the same average bit-arrival rate. The set of data points associated with the label "token bus, Pareto bulk" reflect the impact of this change. Throughput grows linearly with offered load until the bus is roughly 60% utilized. For larger loads, we begin to see some deviations from linear. For a point (x, y) off the diagonal, the difference between x and y reflects the volume of unserved frames at the end of the simulation—the frames in queue. This is no surprise; queueing theory tells us that we should expect significant queue lengths when the arrival pattern is highly variant.

Another important aspect is the average time a frame awaits transmission after arrival. Knowledge of queueing theory and the protocol's operation identifies two factors that ought to contribute to growth in the queuing length. One factor is the time required by a token to reach a new frame arrival. As the offered load increases, the amount of work that the token encounters and must serve prior to reaching the new arrival increases linearly. A second factor is from queueing theory; the view from a station is of an M/G/1 queue. In this view, the service time incorporates the time spent waiting for a token to arrive, a mean that increases with the offered load. A job's average time in an M/G/1 queue grows with $1/(1-\rho)$, where $\rho = \lambda/\mu$ is the ratio of arrival rate to service-completion rate. As the offered load grows, ρ increases; this fact explains the second factor of waiting-time growth. As ρ approaches unity, the asymptotic waiting time increases rapidly.

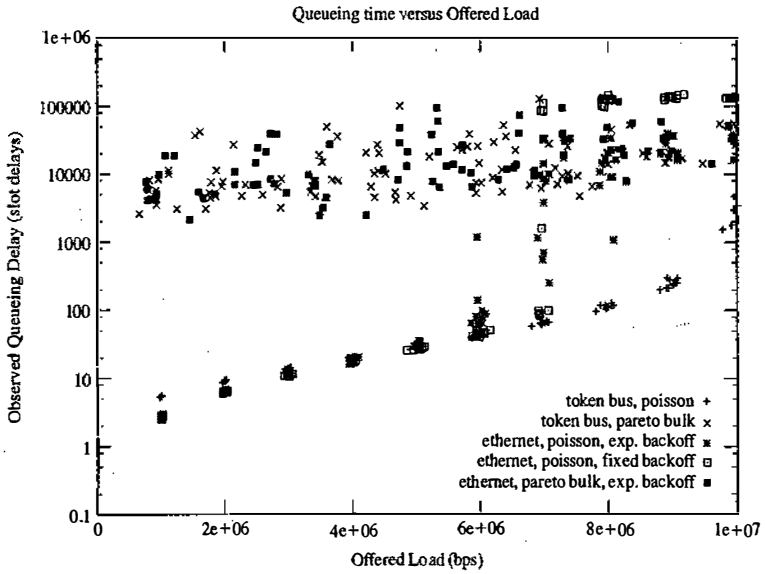


Figure 15.4 Average Queue Delay versus Offered Load: for Token Bus and Ethernet MAC protocols, for Poisson and Bulk Pareto arrival processes, and for Exponential and Fixed Backoff (for Ethernet).

Figure 15.4 confirms this intuition, plotting the average time a frame waits in queue between the time of its arrival and the time at which it begins transmission. Again we execute 10 independent experiments for a given offered load and plot the raw pair (x, q), where x is the average number of bits presented to the network per second in that run and q is the average time a job is enqueued. Units of queueing delay are "slot times", the length of time required for a bit to traverse a cable at the limit of what is permissible for Ethernet (25.6 μsec). The extreme range of queueing delays observed for the five experiment types encourages our use of a log scale on the y-axis. Tracking data from the experiments by using Poisson arrivals, we see stability in the growth pattern, up to the point where the bus is fully saturated. We know to expect extremes there. What is very interesting, though, are the extremely high average queueing delays experienced under the "bulk Pareto" assumptions. If nothing else, these kinds of experiments point out the importance of the traffic model in the analyzing of network behavior.

A straight-forward implementation of a token-bus protocol models devices, the bus, and the explicit and continuous passing of the token among stations. However, this implementation has an undesirable characteristic. Under low traffic load, the model creates a discrete event approximately every 10.84 μsec, the time it takes to transmit a token between adjacent stations. Under low traffic load, the token could completely cycle through the network many times before reaching a point in simulation time when there is a frame available for transmission. Unless the simulation has some particular reason for pushing the token around an otherwise idle network (e.g., if, at each hop, there is a nonzero probability of the token's being lost or corrupted, forcing the protocol to detect and react), there are more efficient ways of executing the simulation, at the cost of incorporating extra logic. We may suppose that each device samples the next future time at which a batch of frames arrives. Before that time, if the device has no frames to transmit, it will make no further demands on the network. When the simulation has reached a time at which no frame is being

transmitted and no device has a frame waiting for transmission, we perform a calculation to advance simulation time past an epoch during which the only activity is token passing. Because the time required to circulate the token around the ring is computable, and the next time at which a frame is available at any station is known, we can advance simulation time to the cycle in which the next frame is transmitted and save ourselves the computational effort of getting to that place by pushing a token around.

15.3.2 Ethernet

Token-based access protocols have been popular, but they have drawbacks when it comes to network management. In particular, every time a device is added to or removed from the network, configuration actions must be taken to ensure that a new device gets the token and that a removed device is never again sent the token. The Ethernet access protocol is a solution to this problem (Spurgeon [2000]). A device attached to an Ethernet cable has no specific idea of other devices on that cable; however, when it wants to use the cable, it must coordinate with such other devices. Consider the problem—a device has a frame to send; when can it send it? Ethernet is a decentralized protocol, meaning that there is no controller granting access. A device can "listen" to the Ethernet cable to see whether it is currently in use. If the cable is already in use, the device holds off until the cable is free. However, two or more devices could independently and more or less simultaneously decide to transmit, shortly after which the transmission on the cable is garbled. Both devices can detect this "collision" (e.g., by comparing what they are transmitting on the cable with what they are receiving from the cable). Collision detection and reaction to it is the one of the key components of the Ethernet protocol; it is a so-called Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol.

The format of an Ethernet frame is illustrated in Figure 15.5. The 8-byte preamble is a special sequence of bits (alternating 1's and 0's, except for the last bit which is also a '1') that listeners on the cable recognize and use to prepare to examine the next frame field, a 6-byte Destination address that may specify one device, a group of devices, or a broadcast to all listening devices. After scanning the full Destination address, a device listening to the cable knows whether it is an intended recipient. The next 6 bytes identify the sending device; then comes a 2-byte field describing the number of data bytes. The data follow, and the frame is terminated with a 4-byte code used for error detection.

When a device decides to transmit, it begins in the knowledge that it is possible for another device to begin also, not yet having heard the new transmission. Ethernet specifications on network design ensure that any transmission will be heard by another device within $\delta = 25.6$ μsec. This is called a slot time. The worst case is that the device begins to transmit at time t, yet before time $t + \delta$, a device at the other end of the cable decides to transmit and does so just before time $t + \delta$, and another δ time is needed by the first device to detect the collision.

The length of the data portion of an ethernet frame is not specified by the protocol. However, there is a lower bound on the allowable length of the data portion. The frame must be large enough so that it takes longer than 2 slot times to transmit it. This bound ensures that, if a collision does occur, the sending device will be transmitting when the effects of the collision reach it, and hence it can detect the collision. This minimum is 46 bytes of data; furthermore, a frame is not permitted to carry more than 1500 bytes of data.

Some of the complexities of Ethernet exist because of physics. An accurate simulation of Ethernet must therefore pay attention to the delicacies of signal latency. The model used to generate Ethernet performance figures specifically accounts for signal latency. It assumes that the devices are evenly spaced along a cable

size (bytes)	8	6	6	2		4
field	Preamble	Destination MAC adrs	Source MAC adrs	Length	Data	Cyclic Redundancy Check

Figure 15.5 Format of Ethernet Frame.

that requires a full slot time (25.6 μsec) for a signal to traverse. When a device listens to the cable to see whether it is free, the model really answers the question of whether the device can, at that instant, hear any transmission that might have already started. This is a matter of measuring the distance between a sending device and a listening device, computing the signal latency time between them, and working out whether the sender started longer ago than that latency. Likewise, when a device has a frame to send and is listening to the cable to find out when it is idle, its view of the cable state is one that accounts for a certain delay between when a transmission ends and when that end is seen by an observer.

A device with a frame to send listens to the cable and, if it hears nothing begins to transmit. If it successfully transmits the frame without collision and has another frame to send, it waits 2 slot times before making the attempt. If a device wanting to send a frame hears that the cable is in use, it simply waits until the cable is quiet and then begins to transmit. The most interesting part of Ethernet is its approach to collisions. If a device transmitting a frame F detects a collision, it continues to transmit—but jumbled—long enough to ensure that it transmits a full minimum frame's worth of bits. This "jamming" ensures that all devices on the cable detect the collision. Next, it backs off and waits a while before trying to send F again.

The backoff period following a collision has been a topic of some study, one in which simulation has played an important role. If the backoff time is short, there is a chance of not overly increasing the delay time of a frame, but there is also a significant chance of incurring another collision. On the other hand, if the backoff time is large, one reduces the risk of a subsequent collision, but ensures that the delay of the frame in the system will be large. Over time, the following strategy, called "exponential backoff", has become the Ethernet standard. Following the mth collision while attempting to transmit frame F, the device randomly samples an integer k from $[0, 2^m-1]$, and waits $2k$ slot times before making another attempt. If 10 attempts are made without success, the frame is simply dropped. The term "exponential backoff" describes the doubling in length of the mean backoff time on each successive collision. Successive collisions are measures, of a sort, of the level of congestion in the network. A device strives to reduce its contribution to the congestion, and so enable other frames to get through and relieve the congestion.

Simulation is a useful tool to investigate both backoff schemes and other variants of Ethernet one might consider. We did experiments (assuming Poisson arrivals) on exponential backoff and on "fixed" backoff—where, after a collision occurs, the sender chooses $k \in [0, 4]$ slot times to wait, uniformly at random. Figure 15.3 illustrates the effects on throughput. Under exponential backoff, throughput increases linearly with offered load until after about 60% utilization. For greater load, throughput hovers in the 70% of bandwidth regime, without significant degradation. The story is quite different under fixed backoff. When offered load is 70% of the network bandwidth, the throughput plummets from 60+% and settles in at around 40% of bandwidth—under higher load, the network delivers poorer service. Queueing delays are affected too, as one would expect. Under high load, the delays under fixed backoff are an order of magnitnde larger than those under exponential backoff.

A final set of experiments used the same Poisson bulk arrival process, with Pareto-based bulk arrivals, assuming exponential backoff. The results are similar to those for the token bus: large and highly variable queueing delays, and some deviation of throughput from linear at high load. This set of experiments suggests that Ethernet may be more sensitive to the Pareto's high variance than is the token-bus protocol.

15.4 DATA LINK LAYER

A network is far more complicated than the single channel seen by a MAC protocol. A frame might be sent and received many times, by many devices, before it reaches its ultimate destination. Consequently, data traveling at the physical layer contains at least two addresses. One address is a hardware address of the intended endpoint of the current hop. This address (like an Ethernet address) is recognizable by a device's network-interface hardware. The second address is the ultimate destination's network address, typically an

IP address. Different types of devices make up the network. A *hub* is a device that simply copies every bit received on one interface to all its other interfaces. Hubs are useful for connecting separated networks, but have the disadvantage that the connection brings those networks into the same Ethernet collision domain.

A *bridge* makes the same sort of connection, but keeps component subnetworks in different collision domains. For every frame heard on one interface, the bridge takes the destination address and looks up in a table the interface through which that destination can be reached. The bridge has nothing to do if one reaches the destination through the same interface as that through which the frame was observed—the destination will recognize the frame for itself. However, if the destination is reached through a different interface, the bridge takes the responsibility of injecting the frame through that interface, moving it closer to its ultimate goal. In injecting the frame, the bridge acts like a source on that subnetwork, engaging in that subnetwork's MAC protocol. The bridge in effect moves a frame from one collision domain and puts it into another. It can also bridge different subdomain technologies (e.g., different types of Ethernet). Contexts where one would consider simulation study of MAC protocols on one subdomain are the sorts of contexts where one would use simulation and involve models of bridges.

A bridge involves only the physical layer and the data link layer. There is a practical limit on devices retaining the physical addresses of other devices, particularly devices that are in different administrative domains. A *router* is a device that can connect more widely dispersed networks, by making its connections at the Network Layer. A frame coming in to a router on one interface is pushed up to the IP layer, where the IP destination address is extracted; the IP address determines which interface should be used to forward the packet. The *forwarding tables* used to direct traffic flow are the result of complex *routing algorithms*, such as OSPF (Moy [1998]) and BGP (van Beijum [2002]). Simulation is frequently used to study variants and optimizations of these protocols.

We will see that network services commonly used provide users with delivery of data error free and in the order it was sent. These attributes are provided in spite of the real possibility that data will be corrupted in transmission or lost in transmission. A router is one place where a frame might be lost, for, if the router experiences a temporary burst of traffic, all to be routed through a particular interface, buffers holding frames waiting to be forwarded could become exhausted. We think of the traffic flowing through a router as being a set of flows, each flow being defined by the source–destination pair involved. When the arrivals become bursty, and the router's buffer becomes saturated, arrivals that cannot be buffered are deliberately dropped. Most flows actively involved in the burst will lose frames. Under TCP, data loss is the signal that congestion exists, and TCP reacts by significantly decreasing the rate at which it injects traffic into the network. But it takes time to detect this loss—a lot more time than it takes to route frames through the router. One idea that has been studied extensively (by using simulation) is *Random Early Detection* (RED) (Floyd and Jacobson [1993]) queue management. The idea behind RED is to have a router continuously monitor the number of frames enqueued for transmission and, when the average length exceeds a threshold, proactively attempt to throttle back arrival rates before the arrivals overwhelm the buffer and cause all of the flows to suffer. RED visits each frame and, with some probability, either preemptively discards it, or marks a "congestion bit" that is available in the TCP header, but is not much used by most TCP implementations. RED chooses a few flows to suffer for the hoped-for sake of the network as a whole. Complexity abounds in finding effective RED parameters (e.g., threshold queue length, probability of dropping a visited frame) and in assessing tradeoffs and impacts that use of RED could have. Simulation, of course, has played and will play a key role in making these assessments.

15.5 TCP

The Transport Control Protocol (TCP) (Comer [2000]) establishes a connection between two devices, both of which view the communication as a stream of bytes. TCP ensures error-free, in-order delivery of that

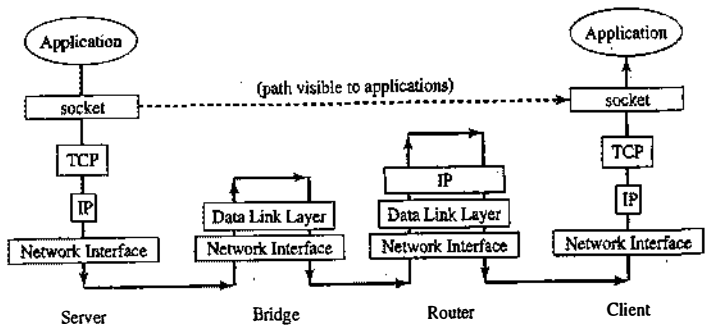


Figure 15.6 Data flow from TCP sender to TCP receiver, passing through network devices.

stream. As we have seen, data frames might be discarded (in response to congestion) somewhere between the sender and receiver; TCP is responsible for recognizing when data loss occurs and for retransmitting data that have gone missing. TCP mechanics are focused on avoiding loss, detecting it, and rapidly responding to it. A number of TCP variants have been proposed and studied; all of these studies use simulation extensively to determine the protocol's behavior under different operating conditions.

Our discussion of TCP serves to illustrate further how different components of networking layers come together. Figure 15.6 illustrates data flow from a server to a client. Two applications intending to communicate establish "sockets" at each side. Sockets are viewed by the applications as buffers into which data could be written and out of which data might be read. Calls to sockets are sometimes blocking calls, in the sense that, if a socket buffer cannot accept more data on a write, or has no data to provide on a read, the calling processes blocks. On the server side, the TCP implementation is responsible for removing data from the socket's buffer and sending it down through the protocol stack to the network. Once on the network, the data pass through different devices. In this figure, we illustrate a bridge (which involves remapping of hardware addresses and does not look at the IP address) and a router (which must decode the IP address to find out the interface through which the data is passed). The client host's IP recognizes that the data ought to go up the stack to TCP, and the client side TCP is responsible for releasing the data to the socket—but only a contiguous stream of data. If the router drops a frame of this flow, the client-side TCP must somehow detect and communicate this absence to the server-side TCP.

TCP segments the data flow into *segments*. Figure 15.7 illustrates the header (in 32-bit words) that is placed around the data. First, note that the only addressing information is "port number" at the source and destination machines—IP is responsible for knowing (and remembering) the identity of the machines involved. From TCP's point of view, there is just a source and a destination. **SeqN** and **AckN** are descriptors of points in the data flow, viewed as a stream of bytes, each numbered. **SeqN** is then the "sequence number" of the first byte in the segment. At the beginning of a connection, a sender and receiver agree upon an initial sequence number (usually random); the **SeqN** value is this initial number plus the byte index within the stream of the first byte carried in the segment. Because the segment size is fixed, the receiver can infer the precise subsequence of the byte stream contained in the segment. The **AckN** field is critical for detecting lost segments. Every time a TCP receiver sends a header about the flow (e.g., in accordance with acknowledgement rules), it puts into the **AckN** field the sequence number of the next byte it needs to receive to maintain a contiguous flow. Since TCP provides a contiguous data stream to the layer above, the value in **AckN** is the initial sequence number plus the index of the next byte it would provide to that layer, if it were available. The linkage of this value with packet loss is subtle. TCP requires a receiver to send an acknowledge for every segment it receives and requires a sender to detect within a certain time limit whether a segment it has sent

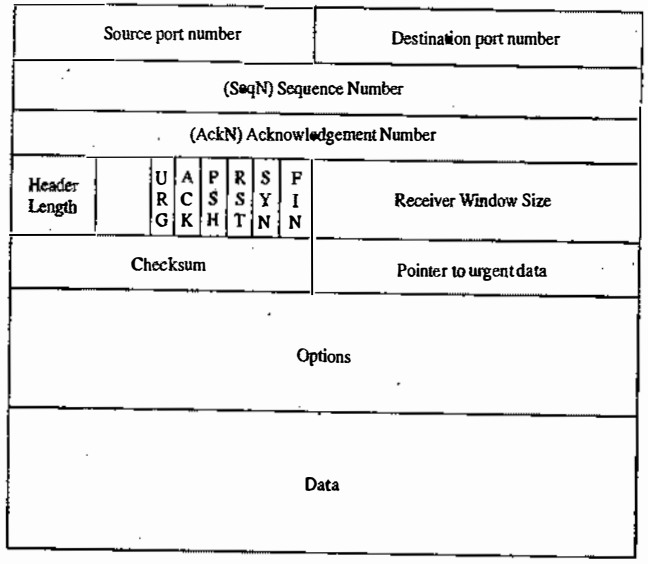


Figure 15.7 TCP header format.

has been acknowledged. Now imagine the effect if 3 segments are sent, and the second one is lost en route. Assume the initial sequence number is 0. The first segment is received, and the receiver sends back an acknowledgement with **AckN** equal to (say) 961 (and the ACK flag set to 1 to indicate that the **AckN** field is valid). The third segment is received, but the receiver notices that the value of **SeqN** is a segment larger than expected—it notices the hole. So it sends back an acknowledgement, but **AckN** in that header is again 961. The second segment sent is not acknowledged, of course, but interestingly, neither is the third. Eventually the TCP sender times out while waiting for these acknowledgements and resends the unacknowledged packets. The only other field in this header, that is critical to our discussion is the Receiver window size, which is included in an Acknowledge to report how many bytes of buffer are currently available to receive data from the sender.

One can visualize TCP as sliding a *send window* over the bytestream. Within the send window are bytes that have been sent, but not yet acknowledged. TCP controls the rate at which it injects segments into the network by maintaining a *congestion window size*, which at any time is the largest the send window is allowed to get. If the send-window size is smaller than the congestion-window size and there are data to send, TCP is free to send it, up until the point where the send window has the same size as the congestion window. When the TCP sender has stopped for this reason, an incoming acknowledgement can reduce the size of the send window (because bytes at the lower end of the window are now acknowledged), and so free more transmission.

TCP tries to find just how much bandwidth it can use for its connection by experimenting with the congestion-window size. When the window is too small, there is bandwidth available but it isn't being used. When the window is too large, the sender contributes to congestion in the network, and the flow could suffer data loss as a result. TCP's philosophy is to grow the congestion window aggressively until there is indication that it has overshot the (unknown) target size, then fall back and advance more slowly. This all is formally described in terms of variables *cwnd* and *ssthresh*. TCP is in *slow start* mode whenever *cwnd* < *ssthresh*, but

in *congestion avoidance* mode whenever *cwnd* > *ssthresh*. Both variables change as TCP executes; *cwnd* grows with acknowledgements a certain way in slow-start, and a different way in congestion-avoidance; *ssthresh* changes when packets are lost. When a TCP connection is first established, *cwnd* is typically set to one segment size and *ssthresh* typically is initialized to a value like 2^{16}. TCP starts in *slow-start* mode, which is distinguished by the characteristic that, for every segment that is acknowledged, *cwnd* grows by a segment's worth of data.

Consider how *cwnd* behaves during slow start by thinking about TCP sending out segments in rounds. In the first round, it sends out one segment, then immediately stalls, because the send-window and congestion-window sizes are equal. When the acknowledgement eventually returns, the sender issues *two* segments as the second round—it replaces the segment that was acknowledged and sends another, because *cwnd* increased by 1. The sender stalls until acknowledgements come in. The *two* acknowledgements for the second round enable the sender to issue *four* segments: half of these due to replacing the ones acknowledged, the other half due to the one-per-acknowledgement increase of *cwnd* rule during slow start. The number of segments issued thus doubles in successive rounds.

Any one of a number of things can halt the doubling of the number of segments sent each round. One is detection of packet loss, the effects of which are to set *ssthresh* to be half the size of the send window, set the send-window size to zero, and set *cwnd* to allow retransmission of one segment (the one in the lost packet). Another way TCP ceases to double the number of segments sent each round is due to the rule that the congestion window may not be increased to exceed certain limits—an internally imposed buffer size at the sender side, or the size of the "receiver window"—the field in ACKs which reports how much space is available for new data. Finally, the doubling effect changes also if *cwnd* grows to exceed *ssthresh*, and so puts TCP into congestion-avoidance mode. Within congestion-avoidance mode, *cwnd* increases, but much more slowly. Intuitively, *cwnd* increases by one segment for every full round that is sent and acknowledged (as opposed to increasing by one segment with every segment that is acknowledged). This is sometimes described as increasing *cwnd* by $1/cwnd$ with every acknowledgement.

Simulation is an excellent tool for understanding how TCP works and many of the subleties of its behavior; we now examine simple examples of that behavior. The first topology is that of a server, a client, and a 800 kbps link between them. The server is to send a 300000 byte file to the client. We attach a monitor that emits a tcpdump formatted trace (see `www.tcpdump.org`) of every TCP packet that passes (in either direction) through the server's network interface. Postprocessing of this trace yields information about how TCP variables of interest behave. In the first situation, we plot the values of **SeqN** in packets sent by the server and the values of **AckN** in packets sent by the receiver in response for the first six rounds, assuming an initial sequence number of 0. This is illustrated in Figure 15.8, where the *Y*-axis is logarithmic in order to illustrate interesting behavior at different scales. The TCP connection is requested by the client at time 192, the first step in TCP's three-way handshake that results in the server sending the first segment at time 192.3 (not actually shown in the graph, to allow higher resolution to later rounds). The **SeqN** in the header of that segment is 1, the index of the first byte in the segment. It takes approximately 100 ms for the segment to reach the client, and another 100 ms for the client's acknowledgment to reach the monitoring point, at time 192.5. (The exact figures are a little different, as they account for the transmission delay caused by the link bandwidth.) The **ACK** bit of that segment is set, and the **AckN** value in the header is 961—the index of the next byte the receiver expects to see. The server's send window now being empty, and *cwnd* having advanced from 1 to 2 by virtue of the received acknowledgement, the server immediately sends *two* segments, one with **SeqN** equal to 961, the next with **SeqN** equal to 961 + 960 = 1921. The graph shows overlapping marks for byte index 961, one from the acknowledgement header, and one from the next segment the server sends. The delay between the server's sending of a segment and the ultimate acknowledgement of that segment is known as the *round-trip time*, or RTT. In this example, the network is as simple as it can be, and the RTT is just the sum of the time to send a segment across the link plus the time to send an acknowledgement back—here, a value very close to 200 ms. At times 192.3 and 192.5, the server stopped sending segments just as soon as

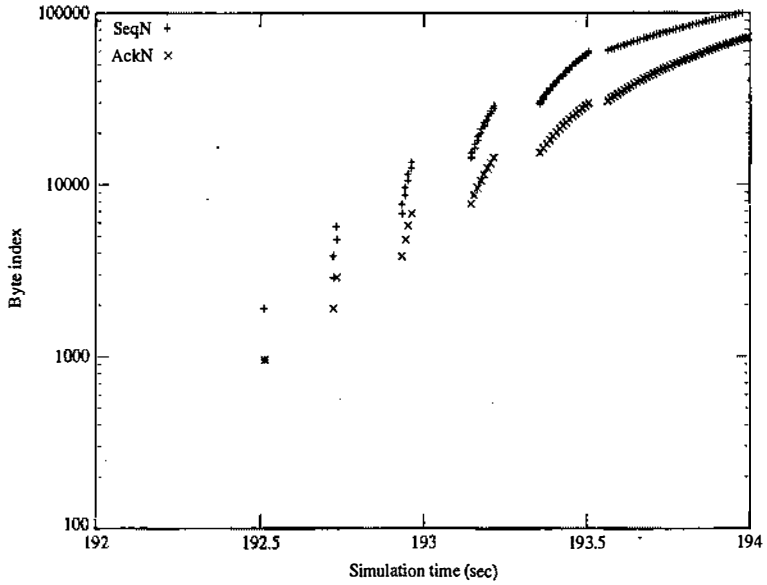


Figure 15.8 Early rounds of TCP connection on 800 kbps/100 ms link, with tcpdump probe at the server's network interface.

its send window and the congestion window were the same size. After one RTT, acknowledgements from the previous round come in; they allow the server to double the number of segments sent from one round to the next. For rounds three and four (at times 192.7 and 192.9, approximately), the graph shows the slight staggering of times associated with acknowledgements coming in and new segments going out.

Figure 15.8 shows how, in slow-start mode, upon receiving a burst of acknowledgements, the server generates a burst of new segments. A moment's reflection shows that, if the acknowledgement for the first segment in that burst is received while the burst is continuing, then the burst will continue ad infinitum. For, at the instant that critical acknowledgement is received, the send window must be smaller than the congestion window, and the send window will not grow after this point, while the congestion window will. We can compute the size of the congestion window at which this phenomenon occurs—it is when the congestion window is large enough that the time needed to transmit that many bytes is precisely the RTT. Back-of-the-envelope calculations indicate that this is 20000 bytes, or just under 21 segments. In these experiments, the receiver window is limited to 32 segments, so this saturation happens before the flow is limited by that buffer. SSFNet initializes *ssthresh* to 65396 bytes, so this saturation point is reached in slow-start, before *cwnd* reaches *ssthresh* and triggers congestion-avoidance mode. Since *cwnd* starts with value 1 and doubles with every round, the server saturates its sends in the middle of the 6th round. This is observed in Figure 15.8, in the round that starts just after time 193.5.

In Figure 15.9, we illustrate this same experiment, along with another that is identical—save that the link latency is 300 ms. A larger epoch of simulation time is illustrated. There is an interesting kink in the **SeqN** data set for the 100 ms network, in the vicinity of **SeqN** = 65*K*. The "slope" of the data set decreases perceptibly. Up to this point, for every acknowledgement received two new segments are transmitted, and they are marked in the tcpdump trace as occurring at the same instant (SSFNet does not ascribe time advance

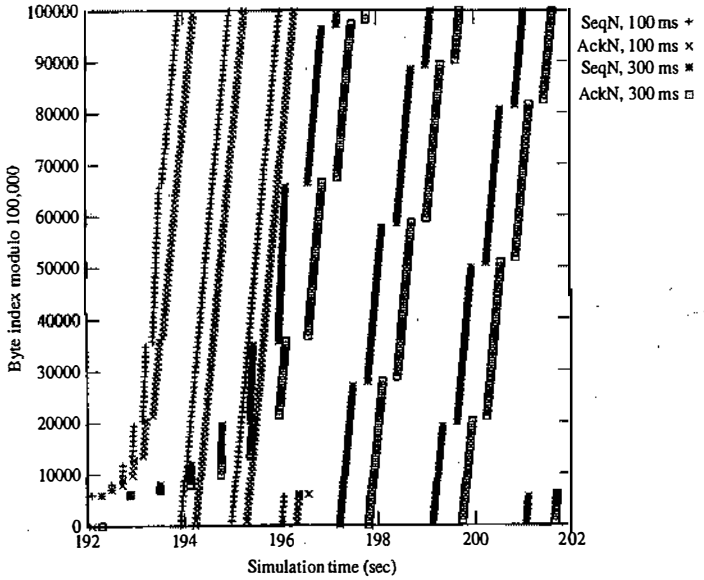


Figure 15.9 TCP connections between server and client: ond 800 kbps/100-ms link, and an 800 kbps/300-ms link.

to protocol actions, only to network transmission). At the point of the kink, the value of *cwnd* becomes equal to the receiver window, 32 segments. The sender window becomes limited by the size of the receiver window, rather than by *cwnd*, so, after the kink, there is a one-to-one correspondence between receipt of an acknowledgement and transmission of another segment. Now consider the experiments using a 300 ms latency. As we'd expect, rounds happen approximately every 600 ms. To saturate the link, the sender window has to become three times as large as in the first experiment—almost 64 segments. However, this will never happen, because the send window will be limited by the receiver window, at 32 segments. Indeed, we see that the change in slope of the **SeqN** trace happens at the same byte index as it did with the first experiment. Likewise, we see visually that there's a gap in transmission time between each successive round.

As a final example of how simulation illustrates the behavior of TCP, we consider an experiment designed to induce packet loss. The topology is that of a server, a router, and a client. Again, the server is to send 300000 bytes to the client. Both server and client connect with the router. The link between server and router has 8 Mbps of bandwidth and 5-ms latency. The link between client and router has 800 kbps of bandwidth and 100-ms latency. The router's interface with the client has a 6000-byte buffer. If a packet arrives to that interface and there is insufficient buffer space available, the packet is dropped. From earlier analysis of TCP, we can foresee, in part, what will happen. In the slow-start phase, the server begins to double the number of segments with each successive round. However, it can push packets towards the router 10 times faster than the router can push packets to the client, so a queue will form at the interface. The buffer holds at most 6 packets, so we expect that, in the round where 8 packets are sent, there will be packet loss. Figure 15.10 illustrates this experiment, adding a trace of *cwnd* behavior to that of **SeqN** and **AckN** (once again measured at the server's network interface). The effects of the packet loss are visually distinctive. Around time 193.5, the server begins to receive a sequence of acknowledgements that all carry the same **AckN** value.

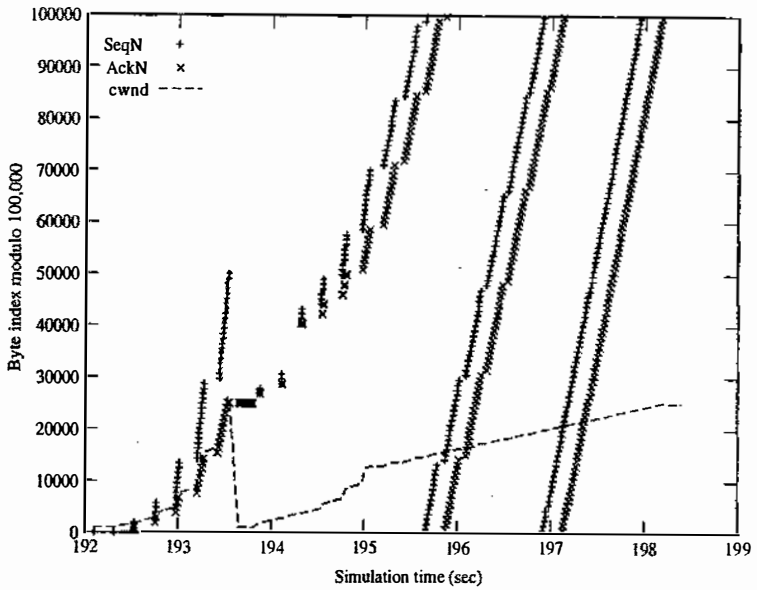


Figure 15.10 TCP connection suffering loss.

These acknowledgements were sent in response to packets that were sent *after* a loss. Recall that TCP rules on **AckN** specify that the receiver identify the sequence number of the next byte it needs to receive to advance the sequence of contiguously received bytes; hence, the repeated **AckN** identifies the beginning of the first lost segment. At the point at which the loss is observed, the send-window size is approximately 25000 bytes; in reaction to the loss, *ssthresh* is set to half this value, *cwnd* is set to 1, and the sender window collapses to size zero in order to cause the retransmission of all segments (from the first lost one forward). In the region between times 193 and 194, we see the impact that loss has on *cwnd* and how the slow-start doubling of *cwnd* with each round begins anew. (Notice the small periods of sharply increased growth at times 194.6, 194.8, and 195.) However, this time, congestion-avoidance mode is entered when *cwnd* reaches *ssthresh*, shortly after time 195; thereafter, it grows more or less linearly with time. This particular transfer ends just before *cwnd* reaches a size that will allow loss once again; had the transfer advanced that far, TCP's treatment of *cwnd* would look very much like the period from 193.8 on.

As these simple examples show, TCP's relatively simple rules create complex behavior. Simulation is an indispensable tool for predicting how TCP will behave in any given context and for understanding that behavior.

15.6 MODEL CONSTRUCTION

SSFNet is a versatile tool for building and analyzing network simulations, used in the previous section to look at how TCP behaves. Suggested homework projects encourage use of SSFNet, and so we describe the general process SSFNet uses in constructing a simulation from an input model. We then illustrate this process, in part, by describing the contents of one input file used in the last subsection. This is not a users' manual for SSFNet; very complete documentation exists at `www.ssfnet.org`. Our aim here to is give a sense of the approach and to encourage readers to investigate further.

15.6.1 Construction

Input to SSFNet is in the form of so-called Domain Modeling Language (DML) files. At the simplest level, a DML file contains just a recursively defined list of attribute–value pairs, where an attribute is a string and a value may be either a string or a list of attribute–value pairs. This structure naturally induces a tree, where interior nodes are attributes (labeled with the attribute string name) and leaves are values of type *string* (rather than of type *list*). To illustrate, consider this DML list:

```
Net [
    frequency 1000000
    host [ id 0
           interface [ id 0 bitrate 800000]
           nhi_route [ dest 1(0) interface 0 ]
    ]
    host [ id 1
           interface [ id 0 bitrate 800000]
           nhi_route [ dest default interface 0 ]
    ]
    link [ attach 0(0) attach 1(0) latency 0.1 ]
]
```

This has some elements of SSFNet DML structure worth noting. Description of a network, elements within the network, and connections between them use a hierarchical naming convention known as the Network-Host-Interface convention, or just NHI. The network is defined in terms of links between interfaces, and each interface has an id number that is unique among all interfaces owned by a common host. That host has an id number that is unique among all hosts in a common net. Each net has an id, unique among all nets contained in the same parent net, and so on. The NHI address 0.1.2(4) refers to an interface named 4, within a host named 2, within a net named 1, within a net named 0. Within a net, a reference such as 2(4) is understood to mean interface 4 associated with the uniquely named host 2 within that understood net. The NHI address of an interface is derived from the nesting described within a DML file. The first interface to appear in the preceding example has NHI address 0(0); the second interface to appear has address 1(0). The `link` attribute in this example specifies two endpoints of the link, in NHI addressing (using the `attach` attribute), and a link latency of 100 ms.

The recursive structure of DML allows it be oxparsed easily and allows one to construct a parse-tree whose interior nodes are attributes and whose leaves are string-valued values. The parse-tree associated with the previous example is illustrated in Figure 15.11. This data structure gives a handy way of methodolically building a model from a DML description. The SSFNet engine recursively traverses the tree and configures core SSFNet objects (such as host). Attributes or values within the tree can be referenced globally by the sequence of attribute labels on nodes from the root to the target. This proves to be useful: one can embed in a DML file a "library" of attribute–value pairs and reference elements of that library.

SSFNet recognizes a variety of attributes, many of which are described in Table 15.1.

15.6.2 Example

Finally, we illustrate some of these ideas by looking at the DML input file for one of our TCP examples. The file is presented in Figure 15.12 (annotated with line numbers for easier reference).

In this particular file, lines 1–8 are comments describing the architecture. Line 10 tells the SSFNet model parser where to find format descriptions of certain constructs; when the parser encounters these constructs in the DML file, it will check against the schema to ensure format correctness. Line 12 starts the

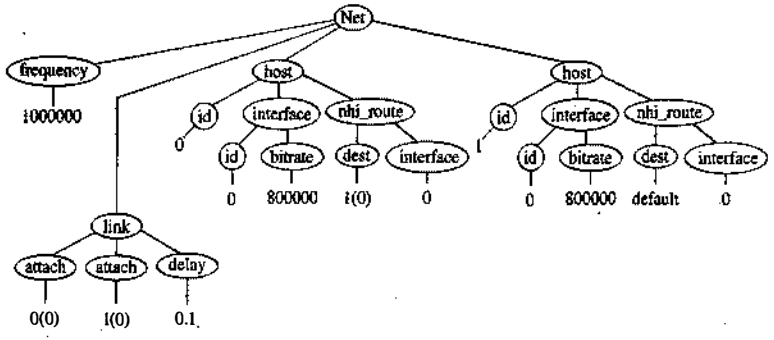


Figure 15.11 Parse tree of simple DML example.

Table 15.1 Common Attributes in SSFNet DML Models

Attribute	*Value*
Net	list describing a network
frequency	number of discrete ticks per simulation second
traffic	list of traffic patterns
pattern	description of traffic pattern, in terms of receiver (client) and server (sender).
servers	list describing a set of servers to which a client might connect—including their NHI identities and port numbers
link	list describing interfaces to be connected, and associated latency
host	list describing a host, and diverse attributes it may have
graph	list of protocols in a host's protocol stack
ProtocolSession	list specifying a protocol
interface	a list describing a connection to the network; attributes include connection bandwidth, and target file for storing monitoring information.
route	description of a forwarding table entry for IP. The *dest* attribute identifies the destination being described; the interface attribute describes which interface packets for that destination should be routed.
dictionary	a list of constants that can be referenced elsewhere within the DML file

overarching list: "Net" followed by a list. Line 14 specifies a clock resolution of 1 microsecond. Lines 15–20 describe the network's traffic, a single pattern that includes host 0 as client. The "servers" attribute gives a list of servers, in this case a single one at NHI address 1(0) (meaning host 1, interface), using port 10.

The "link" attribute at line 24 describes two interfaces to be connected: the one at NHI address 0(0), and the one at NHI address 1(0). The latency across this link is specified to be 0.1 seconds.

A host contains protocols and interfaces to the network. The host beginning at line 28 is given NHI id 1 and contains a "graph" of protocol sessions. Each model of a software component is described as such a session. The order of appearance in the graph is important, descending from higher to lower in the stack. Each protocol session describes its type (e.g., server, client, TCP, IP), and the Java class that describes its behavior. These classes are constructed, by using certain methologies, to be composable; builders of simulation models (in contrast to developers of modeling components these builders use) need not develop new classes,

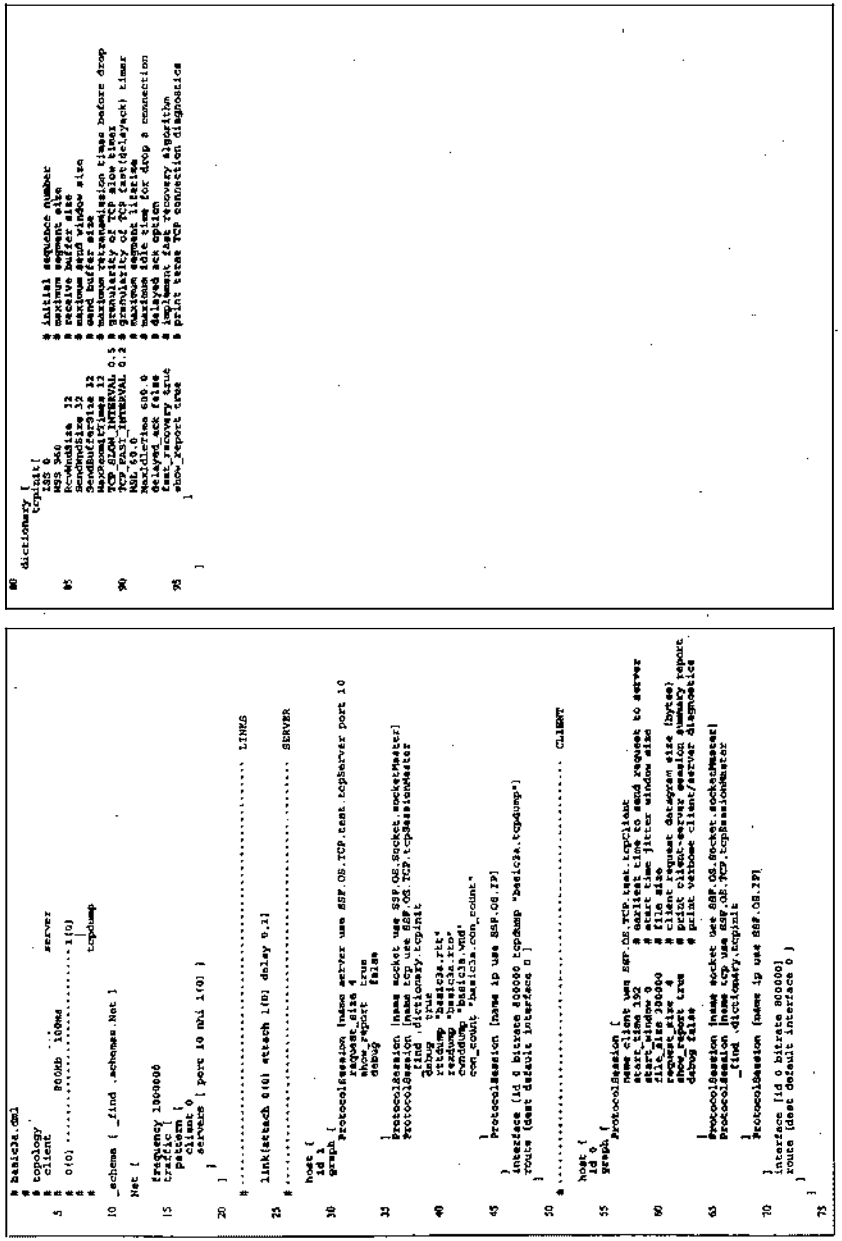


Figure 15.12 Domain Modeling Language specification of a simple network experiment.

but the methodology specifies how one does. A protocol session of a given type may include attributes specific to that type. For example, the tcpServer protocol beginning at line 32 specifies the port through which it is accessible (10). Line 37 begins the declaration of the tcpSessionMaster, a component that manages all TCP sessions. Characteristics of its version of TCP are described by including a list of attributes defined in a list held elsewhere in the DML file. The statement `_find .dictionary.tcpinit` causes the contents of the named list to essentially be inserted at the point of the statement. The string `.dictionary.tcpinit` names the list in terms of how to find it in the file: "." is the highest level list, "`dictionary`" is the name of an attribute in that list, "`tcpinit`" is the attribute associated with the sought list, an attribute of the value-list of `dictionary`. This list starts at line 82.

We quickly describe the meaning of each attribute not obvious from the comments, in order to illustrate the diversity of parameters in SSFNet's implementation of TCP. `RcvWndSize`, `SendWndSize`, and `SendBufferSize` describe units of MSS and limit buffer useage (which affects TCP behavior, as we have already seen). A missing segment will be retransmitted up to `MaxrexmitTimes` times before the TCP session is aborted. `TCP_SLOW_INTERVAL` and `TCP_FAST_INTERVAL` give timer values used to determine when enough time has gone by so that a transmitted segment has not yet been acknowledged. If a TCP session is inactive for `MaxIdleTime` seconds, it is terminated. `delay_ack` and `fast_recovery` are Boolean flags that describe whether to use particular optimizations known for TCP.

Back within the specification of the host (at lines 40–43), we find attributes whose values are files into which the system saves descriptions of how TCP variables behaved during the simulation. Following this (at line 40) is the inclusion of the IP protocol. This, in turn, is followed by declaration of the server's single interface, given id 0 for NHI coordinates and specified to have a bandwidth of 800 Kbits per second. The last attribute for the server is an "`nhi_route`", an element in IP's forwarding table, described in NHI coordinates. The server is not a router and so needs only to direct traffic from IP to one interface. Attribute–value pair `dest default` says to route *everything* through the interface to follow, 0.

Specification of the second host is similar. In this case, the uppermost ProtocolSession is that of a client that requests data, through a socket. Attributes for the client include the simulation time at which it initiates the request. (It actually specifies a window of simulation time in which this occurs, to provide some jitter when multiple clients are to start more or less simultaneously). The length of the transfer being requested is an attribute (line 60). The rest of this host's ProtocolSessions are similar to the server's, although we don't save so much information about TCP's behavior at this host.

15.7 SUMMARY

In this chapter, we touched on some important topics related to simulation of computer networks. Traffic modeling—at different levels of abstraction—is a crucial element of simulating and modeling networks. We emphasized the importance of non-Poisson arrivals models, in some cases to better match characteristics of specific applications, in others to be sure to explain and capture long-range dependence.

Next, we focused on the Data Link layer and on the Media Access Control algorithms. We examined the token-bus and ethernet protocols, discussed subleties of their simulation, and showed by example how significant an impact traffic-model assumptions can have on network performance. Following this, we mentioned issues at the Data Link layer for which simulation has been a critical tool for investigation.

Much of the traffic on the Internet is carried by using TCP. We described TCP's basic rules and used simulation to illustrate some of the consequences of these rules. Finally, we sketched how one builds network models in the SSFNet simulator.

This chapter has barely scratched the surface of how networking uses simulation. Our hope is that what we discuss leads a student to explore more deeply any one of a number of fascinating areas of networking that can be explored only with simulation. The exercises are designed to do this and to teach the student some skill in using SSF and SSFNet.

REFERENCES

BARFORD, P., AND M. CROVELLA [1998], "An Architecture for a WWW Workload Generator," *Proceedings of the 1998 SIGMETRICS Conference*, Madison, WI, pp. 151–160.

BLACK, U. [2001], "Voice Over IP," Prentice Hall, Upper Saddle River, NJ.

COMER, D. [2000], "Networking with TCP/IP Volume I: Principles, Protocols, and Architecture, 4th ed.," Prentice Hall, Upper Saddle River, NJ.

FLOYD, S., AND V. JACOBSON [1993], "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397–413.

KARAGIANNIS T., M. FALOUTSOS, AND M. MOLLE [2003], "A User-Friendly Self-Similarity Analysis Tool," *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 3, pp. 81–93.

KUROSE, J., AND K. ROSS [2002], *Computer Networking: A Top-Down Approach Featuring the Internet*, 2d ed., Addison–Wesley, Reading, MA.

MOY, J. [1998], *OSPF: Anatomy of an Internet Routing Protocol*, Addison–Wesley, Reading, MA.

SPURGEON, C. [2000], *Ethernet: The Definitive Guide*, O'Reilly, Cambridge MA.

VAN BEIJNUM, I. [2002], *BGP: Building Reliable Networks with the Border Gateway Protocol*, O'Reilly, Cambridge, MA.

ZIMMERMAN, H. [1980], "OSI reference model—the ISO model of architecture for open system interconnection" *IEEE Transactions on Communications*, Vol. COM-28, No. 4, pp. 425–432.

ZUKEMAN, M., D. NEAME, AND R. ADDIE [2003], "Internet Traffic Modeling and Future Technology Implications," *Proceedings of the 2003 InfoCom Conference*, San Franciso, CA.

EXERCISES

1. Survey literature in models of Voice-over-IP traffic, and build a simulator that creates traffic load corresponding to one model of particular interest.

2. Create a Markov-Modulated Poisson process (see chapter 14) and a Poisson–Pareto Burst Process that yield the same average bit-rate traffic demand. Acquire the SELFIS tool for analyzing long-range dependence (it's free), and compare traces from the MMP and PPBP models.

3. Get from `www.bcnn.net` the SSF models for the Ethernet protocol experiments reported in this chapter. Design and perform a sensitivity analysis of throughput as a function of the physical distance between ethernet ports. Likewise, design and perform a sensitivity analysis of throughput as a function of maximum frame size.

4. Acquire the SSFNet simulator from `www.ssfnet.org` (free for academic use) and the TCP models described in this chapter from `www.bcnn.net`.

   - Look into how TCP behavior changes in each case by increasing the bandwidth by a factor of 10.
   - Investigate how TCP behavior changes in each case by reducing the link latency by a factor of 10.
   - Work out how TCP behavior changes in each case by increasing the buffer limits expressed in the DML file by a factor of 10.

Appendix

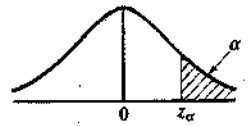
Table A.1 Random Digits

94737	08225	35614	24826	88319	05595	58701	57365	74759
87259	85982	13296	89326	74863	99986	68558	06391	50248
63856	14016	18527	11634	96908	52146	53496	51730	03500
66612	54714	46783	61934	30258	61674	07471	67566	31635
30712	58582	05704	23172	86689	94834	99057	55832	21012
69607	24145	43886	86477	05317	30445	33456	34029	09603
37792	27282	94107	41967	21425	04743	42822	28111	09757
01488	56680	73847	64930	11108	44834	45390	86043	23973
66248	97697	38244	50918	55441	51217	54786	04940	50807
51453	03462	61157	65366	61130	26204	15016	85665	97714
92168	82530	19271	86999	96499	12765	20926	25282	39119
36463	07331	54590	00546	03337	41583	46439	40173	46455
47097	78780	04210	87084	44484	75377	57753	41415	09890
80400	45972	44111	99708	45935	03694	81421	60170	58457
94554	13863	88239	91624	00022	40471	78462	96265	55360
31567	53597	08490	73544	72573	30961	12282	97033	13676
07821	24759	47266	21747	72496	77755	50391	59554	31177
09056	10709	69314	11449	40531	02917	95878	74587	60906
19922	37025	80731	26179	16039	01518	82697	73227	13160
29923	02570	80164	36108	73689	26342	35712	49137	13482
29602	29464	99219	20308	82109	03898	82072	85199	13103
94135	94661	87724	88187	62191	70607	63099	40494	49069
87926	34092	34334	55064	43152	01610	03126	47312	59578
85039	19212	59160	83537	54414	19856	90527	21756	64783
66070	38480	74636	45095	86576	79337	39578	40851	53503
78166	82521	79261	12570	10930	47564	77869	16480	43972
94672	07912	26153	10531	12715	63142	88937	94466	31388
56406	70023	27734	22254	27685	67518	63966	33203	70803
67726	57805	94264	77009	08682	18784	47554	59869	66320
07516	45979	76735	46509	17696	67177	92600	55572	17245
43070	22671	00152	81326	89428	16368	57659	79424	57604
36917	60370	80812	87225	02850	47118	23790	55043	75117
03919	82922	02312	31106	44335	05573	17470	25900	91080
46724	22558	64303	78804	05762	70650	56117	06707	90035
16108	61281	86823	20286	14025	24909	38391	12183	89393
74541	75808	89669	87680	72758	60851	55292	95663	88326
82919	31285	01850	72550	42986	57518	01159	01786	98145
31388	26809	77258	99360	92362	21979	41319	75739	98082
17190	75522	15687	07161	99745	48767	03121	20046	28013
00466	88068	68631	98745	97810	35886	14497	90230	69264

Table A.2 Random Normal Numbers

0.23	−0.17	0.43	2.18	2.13	0.49	2.72	−0.18	0.42
0.24	−1.17	0.02	0.67	−0.59	−0.13	−0.15	−0.46	1.64
−1.16	−0.17	0.36	−1.26	0.91	0.71	−1.00	−1.09	−0.02
−0.02	−0.19	−0.04	1.92	0.71	−0.90	−0.21	−1.40	−0.38
0.39	0.55	0.13	2.55	−0.33	−0.05	−0.34	−1.95	−0.44
0.64	−0.36	0.98	−0.21	−0.52	−0.02	−0.15	−0.43	0.62
−1.90	0.48	−0.54	0.60	−0.35	−1.29	−0.57	0.23	1.41
−1.04	−0.70	−1.69	1.76	0.47	−0.52	−0.73	0.94	−1.63
−.78	0.11	−0.91	−1.13	0.07	0.45	−0.94	1.42	0.75
0.68	1.77	−0.82	−1.68	−2.60	1.59	−0.72	−0.80	0.61
−0.02	0.92	1.76	−0.66	0.18	−1.32	1.26	0.61	0.83
−0.47	1.04	0.83	−2.05	1.00	−0.70	1.12	0.82	0.08
−0.40	1.40	1.20	0.00	0.21	−2.13	−0.22	1.79	0.87
−0.75	0.09	−1.50	0.14	−2.99	−0.41	−0.99	−0.70	0.51
−0.66	−1.97	0.15	−1.16	−0.60	0.50	1.36	1.94	0.11
−0.44	−0.09	−0.59	1.37	0.18	1.44	−0.80	2.11	−1.37
1.41	−2.71	−0.67	1.83	0.97	0.06	−0.28	0.04	−0.21
1.21	−0.52	−0.20	−0.88	−0.78	0.84	−1.08	−0.25	0.17
0.07	0.66	−0.51	−0.04	−0.84	0.04	1.60	−0.92	1.14
−0.08	0.79	−0.09	−1.12	−1.13	0.77	0.40	0.69	−0.12
0.53	−0.36	−2.64	0.22	−0.78	1.92	−0.26	1.04	−1.61
−1.56	1.82	−1.03	1.14	−0.12	−0.78	−0.12	1.42	−0.52
0.03	−1.29	−0.33	2.60	−0.64	1.19	−0.13	0.91	0.78
1.49	1.55	−0.79	1.37	0.97	0.17	0.58	1.43	−1.29
−1.19	1.35	0.16	1.06	−0.17	0.32	−0.28	0.68	0.54
−1.19	−1.03	−0.12	1.07	0.87	−1.40	−0.24	−0.81	0.31
0.11	−1.95	−0.44	−0.39	−0.15	−1.20	−1.98	0.32	2.91
−1.86	0.06	0.19	−1.29	0.33	1.51	−0.36	−0.80	−0.99
0.16	0.28	0.60	−0.78	0.67	0.13	−0.47	−0.18	−0.89
1.21	−1.19	−0.60	−1.22	0.07	−1.13	1.45	0.94	0.54
−0.82	0.54	−0.98	−0.13	1.52	0.77	0.95	−0.84	2.40
0.75	−0.80	−0.28	1.77	−0.16	−0.33	2.43	−1.11	1.63
0.42	0.31	1.56	0.56	0.64	−0.78	0.04	1.34	−0.01
−1.50	−1.78	−0.59	0.16	0.36	1.89	−1.19	0.53	−0.97
−0.89	0.08	0.95	−0.73	1.25	−1.04	−0.47	−0.68	−0.87
0.19	0.85	1.68	−0.57	0.37	−0.48	−0.17	2.36	−0.53
0.49	0.32	−2.08	−1.02	2.59	−0.53	0.15	0.11	0.05
−1.44	0.07	−0.22	−0.93	−1.40	0.54	−1.28	−0.15	0.67
−0.21	−0.48	1.21	0.67	−1.10	−0.75	−0.37	0.68	−0.02
−0.65	−0.12	0.94	−0.44	−1.21	−0.06	−1.28	−1.51	1.39
0.24	−0.83	1.55	0.33	−0.59	−1.24	0.70	0.01	0.15
−0.73	1.24	0.40	−0.61	0.68	0.69	0.07	−0.23	−0.66
−1.93	0.75	−0.32	0.95	1.35	1.51	−0.88	0.10	−1.19
0.08	0.16	0.38	−0.96	1.99	−0.20	0.98	0.16	0.26
−0.47	−1.25	0.32	0.51	−1.04	0.97	2.60	−0.08	1.19

Table A.3 Cumulative Normal Distribution



$$\phi(z_\alpha) = \int_{-\infty}^{z_\alpha} \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du = 1 - \alpha$$

z_α	0.00	0.01	0.02	0.03	0.04	z_α
0.0	0.500 00	0.503 99	0.507 98	0.511 97	0.515 95	**0.0**
0.1	0.539 83	0.543 79	0.547 76	0.551 72	0.555 67	**0.1**
0.2	0.579 26	0.583 17	0.587 06	0.590 95	0.594 83	**0.2**
0.3	0.617 91	0.621 72	0.625 51	0.629 30	0.633 07	**0.3**
0.4	0.655 42	0.659 10	0.662 76	0.666 40	0.67003	**0.4**
0.5	0.691 46	0.694 97	0.698 47	0.701 94	0.705 40	**0.5**
0.6	0.725 75	0.729 07	0.732 37	0.735 65	0.738 91	**0.6**
0.7	0.758 03	0.761 15	0.764 24	0.767 30	0.770 35	**0.7**
0.8	0.788 14	0.791 03	0.793 89	0.796 73	0.799 54	**0.8**
0.9	0.815 94	0.818 59	0.821 21	0.823 81	0.826 39	**0.9**
1.0	0.841 34	0.843 75	0.846 13	0.848 49	0.850 83	**1.0**
1.1	0.864 33	0.866 50	0.868 64	0.870 76	0.872 85	**1.1**
1.2	0.884 93	0.886 86	0.888 77	0.890 65	0.892 51	**1.2**
1.3	0.903 20	0.904 90	0.906 58	0.908 24	0.909 88	**1.3**
1.4	0.919 24	0.920 73	0.922 19	0.923 64	0.925 06	**1.4**
1.5	0.933 19	0.934 48	0.935 74	0.936 99	0.938 22	**1.5**
1.6	0.945 20	0.946 30	0.947 38	0.948 45	0.949 50	**1.6**
1.7	0.955 43	0.956 37	0.957 28	0.958 18	0.959 07	**1.7**
1.8	0.964 07	0.964 85	0.965 62	0.966 37	0.967 11	**1.8**
1.9	0.971 28	0.971 93	0.972 57	0.973 20	0.973 81	**1.9**
2.0	0.977 25	0.977 78	0.978 31	0.978 82	0.979 32	**2.0**
2.1	0.982 14	0.982 57	0.983 00	0.983 41	0.983 82	**2.1**
2.2	0.986 10	0.986 45	0.986 79	0.987 13	0.987 45	**2.2**
2.3	0.989 28	0.989 56	0.989 83	0.990 10	0.990 36	**2.3**
2.4	0.991 80	0.992 02	0.992 24	0.992 45	0.992 66	**2.4**
2.5	0.993 79	0.993 96	0.994 13	0.994 30	0.994 46	**2.5**
2.6	0.995 34	0.995 47	0.995 60	0.995 73	0.995 85	**2.6**
2.7	0.996 53	0.996 64	0.996 74	0.996 83	0.996 93	**2.7**
2.8	0.997 44	0.997 52	0.997 60	0.997 67	0.997 74	**2.8**
2.9	0.998 13	0.998 19	0.998 25	0.998 31	0.998 36	**2.9**
3.0	0.998 65	0.998 69	0.998 74	0.998 78	0.998 82	**3.0**
3.1	0.999 03	0.999 06	0.999 10	0.999 13	0.999 16	**3.1**
3.2	0.999 31	0.999 34	0.999 36	0.999 38	0.999 40	**3.2**
3.3	0.999 52	0.999 53	0.999 55	0.999 57	0.999 58	**3.3**
3.4	0.999 66	0.999 68	0.999 69	0.999 70	0.999 71	**3.4**
3.5	0.999 77	0.999 78	0.999 78	0.999 79	0.999 80	**3.5**
3.6	0.999 84	0.999 85	0.999 85	0.999 86	0.999 86	**3.6**
3.7	0.999 89	0.999 90	0.999 90	0.999 90	0.999 91	**3.7**
3.8	0.999 93	0.999 93	0.999 93	0.999 94	0.999 94	**3.8**
3.9	0.999 95	0.999 95	0.999 96	0.999 96	0.999 96	**3.9**

(continued overleaf)

Table A.3 *(continued)*

z_α	0.05	0.06	0.07	0.08	0.09	z_α
0.0	0.519 94	0.523 92	0.527 90	0.531 88	0.535 86	**0.0**
0.1	0.559 62	0.563 56	0.567 49	0.571 42	0.575 34	**0.1**
0.2	0.598 71	0.602 57	0.606 42	0.610 26	0.614 09	**0.2**
0.3	0.636 83	0.640 58	0.644 31	0.648 03	0.651 73	**0.3**
0.4	0.673 64	0.677 24	0.680 82	0.684 38	0.687 93	**0.4**
0.5	0.708 84	0.712 26	0.715 66	0.719 04	0.722 40	**0.5**
0.6	0.742 15	0.745 37	0.748 57	0.751 75	0.754 90	**0.6**
0.7	0.773 37	0.776 37	0.779 35	0.782 30	0.785 23	**0.7**
0.8	0.802 34	0.805 10	0.807 85	0.810 57	0.813 27	**0.8**
0.9	0.824 94	0.831 47	0.833 97	0.83646	0.838 91	**0.9**
1.0	0.853 14	0.855 43	0.857 69	0.859 93	0.862 14	**1.0**
1.1	0.874 93	0.876 97	0.879 00	0.881 00	0.882 97	**1.1**
1.2	0.894 35	0.896 16	0.897 96	0.899 73	0.901 47	**1.2**
1.3	0.911 49	0.913 08	0.914 65	0.916 21	0.917 73	**1.3**
1.4	0.926 47	0.927 85	0.929 22	0.930 56	0.931 89	**1.4**
1.5	0.939 43	0.94062	0.941 79	0.942 95	0.944 08	**1.5**
1.6	0.950 53	0.951 54	0.952 54	0.953 52	0.954 48	**1.6**
1.7	0.959 94	0.960 80	0.961 64	0.962 46	0.963 27	**1.7**
1.8	0.967 84	0.968 56	0.969 26	0.969 95	0.970 62	**1.8**
1.9	0.974 41	0.975 00	0.975 58	0.976 15	0.976 70	**1.9**
2.0	0.979 82	0.980 30	0.98077	0.981 24	0.981 69	**2.0**
2.1	0.984 22	0.984 61	0.985 00	0.985 37	0.985 74	**2.1**
2.2	0.987 78	0.988 09	0.988 40	0.988 70	0.988 99	**2.2**
2.3	0.990 61	0.990 86	0.991 11	0.991 34	0.991 58	**2.3**
2.4	0.992 86	0.993 05	0.993 24	0.993 43	0.993 61	**2.4**
2.5	0.994 61	0.994 77	0.994 92	0.995 06	0.995 20	**2.5**
2.6	0.995 98	0.996 09	0.996 21	0.996 32	0.996 43	**2.6**
2.7	0.997 02	0.997 11	0.997 20	0.997 28	0.997 36	**2.7**
2.8	0.997 81	0.997 88	0.997 95	0.998 01	0.998 07	**2.8**
2.9	0.998 41	0.998 46	0.998 51	0.998 56	0.998 61	**2.9**
3.0	0.998 86	0.998 89	0.998 93	0.998 97	0.999 00	**3.0**
3.1	0.999 18	0.999 21	0.999 24	0.999 26	0.999 29	**3.1**
3.2	0.999 42	0.999 44	0.999 46	0.999 48	0.999 50	**3.2**
3.3	0.999 60	0.999 61	0.999 62	0.999 64	0.999 65	**3.3**
3.4	0.999 72	0.999 73	0.999 74	0.999 75	0.999 76	**3.4**
3.5	0.999 81	0.999 81	0.999 82	0.999 83	0.999 83	**3.5**
3.6	0.999 87	0.999 87	0.999 88	0.999 88	0.999 89	**3.6**
3.7	0.999 91	0.999 92	0.999 92	0.999 92	0.999 92	**3.7**
3.8	0.999 94	0.999 94	0.999 95	0.999 95	0.999 95	**3.8**
3.9	0.999 96	0.999 96	0.999 96	0.999 97	0.999 97	**3.9**

Source: W. W. Hines and D. C. Montgomery, *Probability and Statistics in Engineering and Management Science*, 2d ed., © 1980, pp. 592–3. Reprinted by permission of John Wiley & Sons, Inc., New York.

Table A.4 Cumulative Poisson Distribution

	α = Mean											
x	.01	.05	.1	.2	.3	.4	.5	.6	.7	.8	.9	x
0	.990	.951	.905	.819	.741	.670	.607	.549	.497	.449	.407	0
1	1.000	.999	.995	.982	.963	.938	.910	.878	.844	.809	.772	1
2		1.000	1.000	.999	.996	.992	.986	.977	.966	.953	.937	2
3				1.000	1.000	.999	.998	.997	.994	.991	.987	3
4						1.000	1.000	1.000	.999	.999	.998	4
5									1.000	1.000	1.000	5

	α = Mean											
x	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	x
0	.368	.333	.301	.273	.247	.223	.202	.183	.165	.150	.135	0
1	.736	.699	.663	.627	.592	.558	.525	.493	.463	.434	.406	1
2	.920	.900	.879	.857	.833	.809	.783	.757	.731	.704	.677	2
3	.981	.974	.966	.957	.946	.934	.921	.907	.891	.875	.857	3
4	.996	.995	.992	.989	.986	.981	.976	.970	.964	.956	.947	4
5	.999	.999	.998	.998	.997	.996	.994	.992	.990	.987	.983	5
6	1.000	1.000	1.000	1.000	.999	.999	.999	.998	.997	.997	.995	6
7					1.000	1.000	1.000	1.000	.999	.999	.999	7
8									1.000	1.000	1.000	8

	α = Mean											
x	2.2	2.4	2.6	2.8	3.0	3.5	4.0	4.5	5.0	5.5	6.0	x
0	.111	.091	.074	.061	.050	.030	.018	.011	.007	.004	.002	0
1	.355	.308	.267	.231	.199	.136	.092	.061	.040	.027	.017	1
2	.623	.570	.518	.469	.423	.321	.238	.174	.125	.088	.062	2
3	.819	.779	.736	.692	.647	.537	.433	.342	.265	.202	.151	3
4	.928	.904	.877	.848	.815	.725	.629	.532	.440	.358	.285	4
5	.975	.964	.951	.935	.916	.858	.785	.703	.616	.529	.446	5
6	.993	.988	.983	.976	.966	.935	.889	.831	.762	.686	.606	6
7	.998	.997	.995	.992	.988	.973	.949	.913	.867	.809	.744	7
8	1.000	.999	.999	.998	.996	.990	.979	.960	.932	.894	.847	8
9		1.000	1.000	.999	.999	.997	.992	.983	.968	.946	.916	9
10				1.000	1.000	.999	.997	.993	.986	.975	.957	10
11						1.000	.999	.998	.995	.989	.980	11
12							1.000	.999	.998	.996	.991	12
13								1.000	.999	.998	.996	13
14									1.000	.999	.999	14
15										1.000	.999	15
16											1.000	16

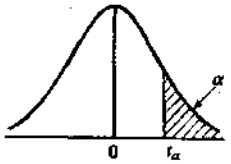
(continued overleaf)

Table A.4 *(continued)*

	α = Mean											
x	6.5	7.0	7.5	8.0	9.0	10.0	12.0	14.0	16.0	18.0	20.0	x
0	.002	.001	.001									0
1	.011	.007	.005	.003	.001							1
2	.043	.030	.020	.014	.006	.003	.001					2
3	.112	.082	.059	.042	.021	.010	.002					3
4	.224	.173	.132	.100	.055	.029	.008	.002				4
5	.369	.301	.241	.191	.116	.067	.020	.006	.001			5
6	.527	.450	.378	.313	.207	.130	.046	.014	.004	.001		6
7	.673	.599	.525	.453	.324	.220	.090	.032	.010	.003	.001	7
8	.792	.729	.662	.593	.456	.333	.155	.062	.022	.007	.002	8
9	.877	.830	.776	.717	.587	.458	.242	.109	.043	.015	.005	9
10	.933	.901	.862	.816	.706	.583	.347	.176	.077	.030	.011	10
11	.966	.947	.921	.888	.803	.697	.462	.260	.127	.055	.021	11
12	.984	.973	.957	.936	.876	.792	.576	.358	.193	.092	.039	12
13	.993	.987	.978	.966	.926	.864	.682	.464	.275	.143	.066	13
14	.997	.994	.990	.983	.959	.917	.772	.570	.368	.208	.105	14
15	.999	.998	.995	.992	.978	.951	.844	.669	.467	.287	.157	15
16	1.000	.999	.998	.996	.989	.973	.899	.756	.566	.375	.221	16
17		1.000	.999	.998	.995	.986	.937	.827	.659	.469	.297	17
18			1.000	.999	.998	.993	.963	.883	.742	.562	.381	18
19				1.000	.999	.997	.979	.923	.812	.651	.470	19
20					1.000	.998	.988	.952	.868	.731	.559	20
21						.999	.994	.971	.911	.799	.644	21
22						1.000	.997	.983	.942	.855	.721	22
23							.999	.991	.963	.899	.787	23
24							.999	.995	.978	.932	.843	24
25							1.000	.997	.987	.955	.888	25
26								.999	.993	.972	.922	26
27								.999	.996	.983	.948	27
28								1.000	.998	.990	.966	28
29									.999	.994	.978	29
30									.999	.997	.987	30
31									1.000	.998	.992	31
32										.999	.995	32
33										1.000	.997	33
34											.999	34
35											.999	35
36											1.000	36

Source: J. Banks and R. G. Heikes, *Handbook of Tables and Graphs for the Industrial Engineer and Manager*, © 1984, pp. 34–35. Reprinted by permission of John Wiley and Sons, Inc., New York.

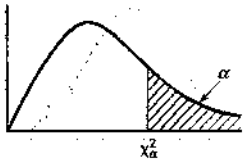
Table A.5 Percentage Points of The Student's t Distribution with v Degrees of Freedom



v	$t_{0.005}$	$t_{0.01}$	$t_{0.025}$	$t_{0.05}$	$t_{0.10}$
1	63.66	31.82	12.71	6.31	3.08
2	9.92	6.92	4.30	2.92	1.89
3	5.84	4.54	3.18	2.35	1.64
4	4.60	3.75	2.78	2.13	1.53
5	4.03	3.36	2.57	2.02	1.48
6	3.71	3.14	2.45	1.94	1.44
7	3.50	3.00	2.36	1.90	1.42
8	3.36	2.90	2.31	1.86	1.40
9	3.25	2.82	2.26	1.83	1.38
10	3.17	2.76	2.23	1.81	1.37
11	3.11	2.72	2.20	1.80	1.36
12	3.06	2.68	2.18	1.78	1.36
13	3.01	2.65	2.16	1.77	1.35
14	2.98	2.62	2.14	1.76	1.34
15	2.95	2.60	2.13	1.75	1.34
16	2.92	2.58	2.12	1.75	1.34
17	2.90	2.57	2.11	1.74	1.33
18	2.88	2.55	2.10	1.73	1.33
19	2.86	2.54	2.09	1.73	1.33
20	2.84	2.53	2.09	1.72	1.32
21	2.83	2.52	2.08	1.72	1.32
22	2.82	2.51	2.07	1.72	1.32
23	2.81	2.50	2.07	1.71	1.32
24	2.80	2.49	2.06	1.71	1.32
25	2.79	2.48	2.06	1.71	1.32
26	2.78	2.48	2.06	1.71	1.32
27	2.77	2.47	2.05	1.70	1.31
28	2.76	2.47	2.05	1.70	1.31
29	2.76	2.46	2.04	1.70	1.31
30	2.75	2.46	2.04	1.70	1.31
40	2.70	2.42	2.02	1.68	1.30
60	2.66	2.39	2.00	1.67	1.30
120	2.62	2.36	1.98	1.66	1.29
∞	2.58	2.33	1.96	1.645	1.28

Source: Robert E. Shannon, *Systems Simulation: The Art and Science*, © 1975, p. 375. Reprinted by permission of Prentice Hall, Upper Saddle River, NJ.

Table A.6 Percentage Points of The Chi-Square Distribution with ν Degrees of Freedom



ν	$\chi^2_{0.005}$	$\chi^2_{0.01}$	$\chi^2_{0.025}$	$\chi^2_{0.05}$	$\chi^2_{0.10}$
1	7.88	6.63	5.02	3.84	2.71
2	10.60	9.21	7.38	5.99	4.61
3	12.84	11.34	9.35	7.81	6.25
4	14.96	13.28	11.14	9.49	7.78
5	16.7	15.1	12.8	11.1	9.2
6	18.5	16.8	14.4	12.6	10.6
7	20.3	18.5	16.0	14.1	12.0
8	22.0	20.1	17.5	15.5	13.4
9	23.6	21.7	19.0	16.9	14.7
10	25.2	23.2	20.5	18.3	16.0
11	26.8	24.7	21.9	19.7	17.3
12	28.3	26.2	23.3	21.0	18.5
13	29.8	27.7	24.7	22.4	19.8
14	31.3	29.1	26.1	23.7	21.1
15	32.8	30.6	27.5	25.0	22.3
16	34.3	32.0	28.8	26.3	23.5
17	35.7	33.4	30.2	27.6	24.8
18	37.2	34.8	31.5	28.9	26.0
19	38.6	36.2	32.9	30.1	27.2
20	40.0	37.6	34.2	31.4	28.4
21	41.4	38.9	35.5	32.7	29.6
22	42.8	40.3	36.8	33.9	30.8
23	44.2	41.6	38.1	35.2	32.0
24	45.6	43.0	39.4	36.4	33.2
25	49.6	44.3	40.6	37.7	34.4
26	48.3	45.6	41.9	38.9	35.6
27	49.6	47.0	43.2	40.1	36.7
28	51.0	48.3	44.5	41.3	37.9
29	52.3	49.6	45.7	42.6	39.1
30	53.7	50.9	47.0	43.8	40.3
40	66.8	63.7	59.3	55.8	51.8
50	79.5	76.2	71.4	67.5	63.2
60	92.0	88.4	83.3	79.1	74.4
70	104.2	100.4	95.0	90.5	85.5
80	116.3	112.3	106.6	101.9	96.6
90	128.3	124.1	118.1	113.1	107.6
100	140.2	135.8	129.6	124.3	118.5

Source: Robert E. Shannon, *Systems Simulation: The Art and Science*, © 1975, p. 372. Reprinted by permission of Prentice Hall, Upper Saddle River, NJ.

Table A.7 Percentage Points of The F Distribution with $\alpha = 0.05$

ν_2 \ ν_1	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	161.4	199.5	215.7	224.6	230.2	234.0	236.8	238.9	240.5	241.9	243.9	245.9	248.0	249.1	250.1	251.1	252.2	253.3	254.3
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38	19.40	19.41	19.43	19.45	19.45	19.46	19.47	19.48	19.49	19.50
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.64	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.53	4.50	4.46	4.43	4.40	4.36
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.84	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.41	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.12	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.90	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.74	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.61	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.51	2.47	2.43	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.42	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.35	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.29	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.24	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.19	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.15	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.08	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.03	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.01	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.98	1.94	1.89	1.84	1.79	1.73
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
26	4.23	3.37	2.98	2.74	2.59	2.47	2.39	2.32	2.27	2.22	2.15	2.07	1.99	1.95	1.90	1.85	1.80	1.75	1.69
27	4.21	3.35	2.96	2.73	2.57	2.46	2.37	2.31	2.25	2.20	2.13	2.06	1.97	1.93	1.88	1.84	1.79	1.73	1.67
28	4.20	3.34	2.95	2.71	2.56	2.45	2.36	2.29	2.24	2.19	2.12	2.04	1.96	1.91	1.87	1.82	1.77	1.71	1.65
29	4.18	3.33	2.93	2.70	2.55	2.43	2.35	2.28	2.22	2.18	2.10	2.03	1.94	1.90	1.85	1.81	1.75	1.70	1.64
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21	2.16	2.09	2.01	1.93	1.89	1.84	1.79	1.74	1.68	1.62
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12	2.08	2.00	1.92	1.84	1.79	1.74	1.69	1.64	1.58	1.51
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04	1.99	1.92	1.84	1.75	1.70	1.65	1.59	1.53	1.47	1.39
120	3.92	3.07	2.68	2.45	2.29	2.17	2.09	2.02	1.96	1.91	1.83	1.75	1.66	1.61	1.55	1.55	1.43	1.35	1.25
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88	1.83	1.75	1.67	1.57	1.52	1.46	1.39	1.32	1.22	1.00

Degrees of Freedom for the Numerator (ν_1)

Degrees of Freedom for Denominator (ν_2)

Source: W. W. Hines and D. C. Montgomery, *Probability and Statistics in Engineering and Management Science*, 2d ed., © 1980, p. 599. Reprinted by permission of John Wiley & Sons, Inc., New York.

Table A.8 Kolmogorov–Smirnov Critical Values

Degrees of Freedom (N)	$D_{0.10}$	$D_{0.05}$	$D_{0.01}$
1	0.950	0.975	0.995
2	0.776	0.842	0.929
3	0.642	0.708	0.828
4	0.564	0.624	0.733
5	0.510	0.565	0.669
6	0.470	0.521	0.618
7	0.438	0.486	0.577
8	0.411	0.457	0.543
9	0.388	0.432	0.514
10	0.368	0.410	0.490
11	0.352	0.391	0.468
12	0.338	0.375	0.450
13	0.325	0.361	0.433
14	0.314	0.349	0.418
15	0.304	0.338	0.404
16	0.295	0.328	0.392
17	0.286	0.318	0.381
18	0.278	0.309	0.371
19	0.272	0.301	0.363
20	0.264	0.294	0.356
25	0.24	0.27	0.32
30	0.22	0.24	0.29
35	0.21	0.23	0.27
Over 35	$\frac{1.22}{\sqrt{N}}$	$\frac{1.36}{\sqrt{N}}$	$\frac{1.63}{\sqrt{N}}$

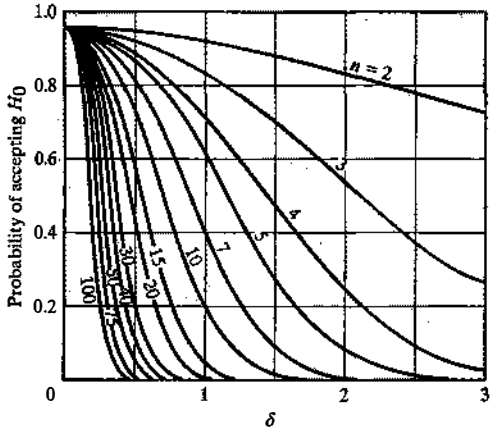
Source: F. J. Massey, "The Kolmogorov–Smirnov Test for Goodness of Fit," *The Journal of the American Statistical Association*, Vol. 46. © 1951, p. 70. Adapted with permission of the American Statistical Association.

Table A.9 Maximum Likelihood Estimates of the Gamma Distribution

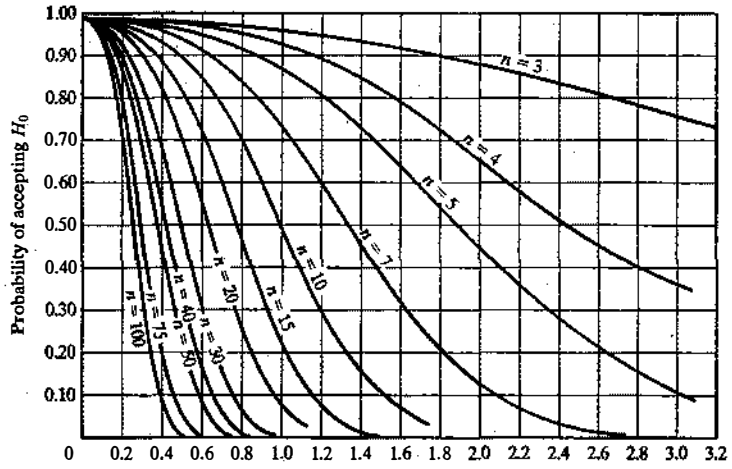
1/M	β	*1/M*	β	*1/M*	β
0.020	0.0187	2.700	1.494	10.300	5.311
0.030	0.0275	2.800	1.545	10.600	5.461
0.040	0.0360	2.900	1.596	10.900	5.611
0.050	0.0442	3.000	1.646	11.200	5.761
0.060	0.0523	3.200	1.748	11.500	5.911
0.070	0.0602	3.400	1.849	11.800	6.061
0.080	0.0679	3.600	1.950	12.100	6.211
0.090	0.0756	3.800	2.051	12.400	6.362
0.100	0.0831	4.000	2.151	12.700	6.512
0.200	0.1532	4.200	2.252	13.000	6.662
0.300	0.2178	4.400	2.353	13.300	6.812
0.400	0.2790	4.600	2.453	13.600	6.962
0.500	0.3381	4.800	2.554	13.900	7.112
0.600	0.3955	5.000	2.654	14.200	7.262
0.700	0.4517	5.200	2.755	14.500	7.412
0.800	0.5070	5.400	2.855	14.800	7.562
0.900	0.5615	5.600	2.956	15.100	7.712
1.000	0.6155	5.800	3.056	15.400	7.862
1.100	0.6690	6.000	3.156	15.700	8.013
1.200	0.7220	6.200	3.257	16.000	8.163
1.300	0.7748	6.400	3.357	16.300	8.313
1.400	0.8272	6.600	3.457	16.600	8.463
1.500	0.8794	6.800	3.558	16.900	8.613
1.600	0.9314	7.000	3.658	17.200	8.763
1.700	0.9832	7.300	3.808	17.500	8.913
1.800	1.034	7.600	3.958	17.800	9.063
1.900	1.086	7.900	4.109	18.100	9.213
2.000	1.137	8.200	4.259	18.400	9.363
2.100	1.188	8.500	4.409	18.700	9.513
2.200	1.240	8.800	4.560	19.000	9.663
2.300	1.291	9.100	4.710	19.300	9.813
2.400	1.342	9.400	4.860	19.600	9.963
2.500	1.393	9.700	5.010	20.000	10.16
2.600	1.444	10.000	5.160		

Source: S. C. Choi and R. Wette, "Maximum Likelihood Estimates of the Gamma Distribution and Their Bias," *Technometrics*, Vol. 11, No. 4, Nov. © 1969, pp. 688–9. Adapted with permission of the American Statistical Association.

Table A.10 Operating Characteristic Curves for The Two-Sided t Test for Different Values of Sample Size n



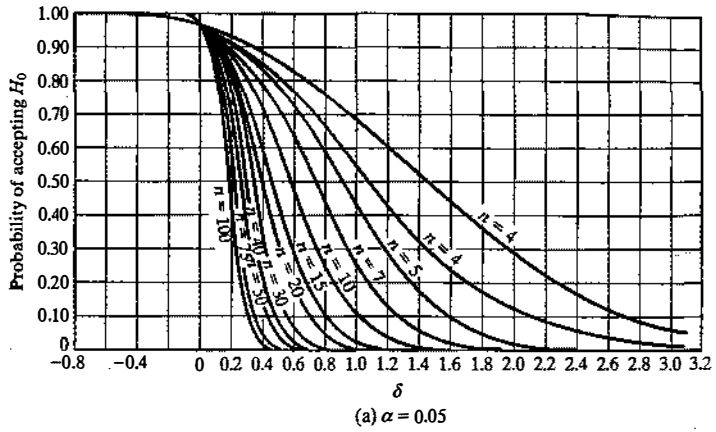
(a) $\alpha = 0.05$



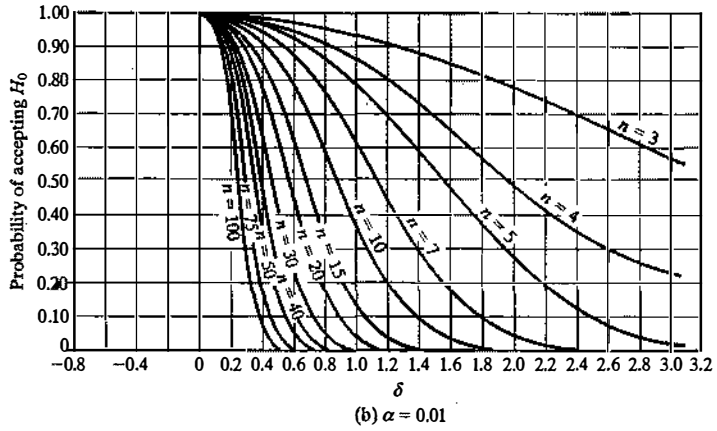
(b) $\alpha = 0.01$

Source: C. L. Ferris, F. E. Grubbs, and C. L. Weaver, "Operating Characteristics for the Common Statistical Tests of Significance," *Annals of Mathematical Statistics*, June 1946. Reproduced with permission of The Institute of Mathematical Statistics.

Table A.11 Operating Characteristic Curves for the One-Sided t Test for Different Values of Sample Size n



(a) $\alpha = 0.05$



(b) $\alpha = 0.01$

Source: A. H. Bowker and G. J. Lieberman, *Engineering Statistics*, 2d ed., © 1972, p. 203. Reprinted by permission of Prentice Hall, Upper Saddle River, NJ.

Index

B

C

D

J

K

L

M

N

O

P

Q

R

S

T