7.9 SIMULATION

GLOSSARY

Discrete-event simulation: a simulation of a time-evolving stochastic system in which changes to the state of the system can only occur at discrete instants.

Monte Carlo simulation: a simulation used to study both deterministic and stochastic phenomenon, in which the passage of time is not material.

Pseudo-random numbers: numbers that are generated in a predictable fashion, but which appear to behave like independent and identically distributed random numbers.

Simulation: a technique for studying numerically the behavior of complex stochastic systems and estimating their performance.

Variance reduction: techniques for obtaining greater precision for a fixed amount of sampling.

INTRODUCTION

Simulation is a technique for numerically estimating the performance of a complex stochastic system when analytic solution is not feasible [LaKe91, Sc90]. This section discusses both *discrete-event* and *Monte Carlo* simulation techniques. In discrete-event simulation models, the passage of time plays a key role, as changes to the state of the system occur only at certain points in simulated time. Queueing and inventory systems can be studied by discrete-event simulation models. Monte Carlo simulation models, on the other hand, do not require the passage of time. Monte Carlo simulation models have been used in estimating eigenvalues, estimating π, and estimating the quantiles of a mathematically intractable test statistic in hypothesis testing.

Simulation has been described [BrEtal87] as "driving a model of a system with suitable inputs and observing the corresponding outputs." Accordingly, the following three subsections discuss input modeling, output analysis, and simulation programming languages.

7.9.1 INPUT MODELING

This section addresses three key issues in constructing a simulation model:

- Determining a source of randomness to drive the probabilistic aspects of the model.
- Input model selection to determine the appropriate probabilistic models to drive the simulation.
- Random variate generation algorithms that transform random numbers to random variates.

Definitions:

Random numbers are real numbers generated uniformly over the interval $(0, 1)$.

A **random number generator** is any mechanism or algorithm for generating random numbers.

Pseudo-random numbers are values that are generated deterministically, but which appear to behave like independent and identically distributed random numbers.

Le m be a large prime integer. A **multiplicative linear congruential generator** produces a stream of pseudo-random numbers x_i/m for $i = 1, 2, \ldots$ based on the following recursive relationship:

$$x_{i+1} := ax_i \bmod m,$$

where a is an integer multiplier between 1 and $m-1$, and x_0 is an integer seed between 1 and $m-1$.

An **input model** characterizes the stochastic elements of a discrete-event simulation model.

A **trace-driven** input model generates a process that is identical to the collected data values without relying on a parametric model.

A **random variate** is a realization of a random variable.

RANDOM NUMBER GENERATION:

Stochastic simulations typically derive their source of randomness from random numbers. Desirable properties for random number generators include: (1) uniformity, (2) independence, (3) speed, (4) minimal memory requirements, (5) ease of implementation, (6) portability across various computer systems, (7) reproducibility, and (8) multiple stream capability.

Facts:

1. Although numerous methods have been proposed for generating random numbers, multiplicative linear congruential generators are typically used to produce a stream of pseudo-random numbers.

2. Due to the prevalence of 32-bit computer architecture, m is often chosen to be $2^{31}-1$, which is prime.

3. A *full period* generator, which cycles through all of the $m-1$ possible x_i values prior to repeating, is obtained by selecting a to be a primitive element modulo m. [See Section 4.5]

4. If the multiplier a is chosen so that it is "modulus-compatible" with m, then potential overflow can be averted for large values of m. Two values of a that are often used with $m = 2^{31} - 1$ are $a = 7^5 = 16807$ and $a = 48271$ [PaMi88].

INPUT SELECTION:

We consider here how to model an element (e.g., arrival process or service time distribution for a queueing system) in a discrete-event simulation, given a data set collected on the element of interest. Successful input modeling requires a close match between the input model and the true underlying probabilistic mechanism associated with the system.

Facts:

5. One of the first steps in determining an appropriate input model for an element of a discrete-event simulation is to assess whether the observations are independent and identically distributed (iid).

6. An input model can be specified using a cumulative distribution function, joint probability density function, hazard function, intensity function, variate-generation algorithm, etc.

7. Many input models rely on parametric probabilistic models such as the binomial, normal, and Weibull distributions. Maximum likelihood is typically used to estimate parameters.

8. Bézier curves [FlWi93] offer a unique combination of the parametric and nonparametric approaches. After an initial distribution is fitted to the data set, the modeler decides whether differences between the empirical and fitted models represent sampling variability (chance variation) or an aspect of the distribution that should be included in the input model.

9. Multivariate distributions (e.g., the multivariate normal distribution with mean μ and variance-covariance matrix Σ) are considered by [Jo87].

Examples:

1. Suppose that a sequence of arrival times (e.g., of customers at a bank) is collected over a 24-hour time period. A trace-driven input model for the arrival process is generated by having arrivals occur at the same times as the observed values.

2. Let $t_1, t_2, \ldots, t_n$ be the arrival times to a queue collected on the time interval $(0, c]$.

(a) If the times between arrivals are iid, a parametric or nonparametric model can be fitted to the data. In the former case, parameters are often estimated by maximizing the likelihood function [LaKe91]

$$L(\theta) = \prod_{i=1}^{n} f(x_i, \theta),$$

where $x_i = t_i - t_{i-1}$ for $i = 1, 2, \ldots, n$, $t_0 \equiv 0$, $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$ is a vector of unknown parameters, and $f(x_i, \theta)$ is the probability density function of the interarrival times.

(b) If the interarrival times are not iid, then a nonstationary point process might be considered, such as a nonhomogeneous Poisson process, where the arrival rate $\lambda(t; \theta)$ varies over time. One parametric model is the *power law* process, with intensity function $\lambda(t; \lambda, \kappa) = \lambda^{\kappa} \kappa t^{\kappa - 1}$ for $t > 0$, where λ and κ are positive parameters. The likelihood function for the single realization on $(0, c]$ is

$$L(\lambda, \kappa) = \left(\prod_{i=1}^{n} \lambda(t_i; \lambda, \kappa) \right) \exp \left(- \int_0^c \lambda(t; \lambda, \kappa) dt \right).$$

Maximum likelihood estimators can be determined by maximizing $L(\lambda, \kappa)$ or its logarithm with respect to λ and κ. Confidence regions for the unknown parameters can be found by using asymptotic properties of the likelihood ratio statistic or the observed information matrix. As with all statistical modeling, goodness-of-fit tests should be performed in order to assess the model adequacy.

RANDOM VARIATE GENERATION:

Once an input model has been chosen, random variate generation algorithms are used to transform random numbers to variates from the input model.

Facts:

10. Devroye [De86] gives algorithms for converting random numbers to random variates associated with input models chosen to drive the simulation.

11. Techniques commonly used for generating random variates from univariate probability distributions are: inverse transformation, composition, acceptance/rejection, and special properties.

12. Algorithm 1, which shows the inversion method, is based on the probability integral transformation. It is assumed that the cumulative distribution function $F(x)$ for the input model of interest has the inverse $F^{-1}(U)$.

13. Other topics in variate generation include table methods, generating from multivariate distributions, random sampling, estimating integrals, and generating processes correlated in time.

ALGORITHM 1: Inverse Transformation

procedure *generate*(F: cumulative distribution)
1. Generate $U \sim \mathrm{U}(0,1)$.
2. $X \leftarrow F^{-1}(U)$.

Example:

3. The Weibull distribution has cumulative distribution function $F(x) = 1 - e^{-(\lambda x)^\kappa}$ for $x > 0$. The inverse cumulative distribution function is $F^{-1}(y) = [-\log(1-y)]^{1/\kappa}/\lambda$ for $0 < y < 1$. Algorithm 1 can be used to generate a Weibull variate according to

$$X := [-\log(1-U)]^{1/\kappa}/\lambda.$$

7.9.2 OUTPUT ANALYSIS

Once a verified and validated simulation model has been developed, a modeler typically wants to estimate measures of performance associated with outputs of the model. Although there are often several performance measures of interest, the focus here is on a single measure of performance θ (e.g., the mean wait time in a queue). This section discusses the following issues in analyzing the output from a simulation model:

- Using point estimation to determine an estimate for θ.
- Determining a confidence interval to assess the accuracy of the point estimate.
- Using variance reduction techniques to obtain more precise point estimates.

Definitions:

Suppose $\{Y_i\}$ is the **output stochastic process**. If the output stochastic process consists of independent observations obtained from a population with cumulative distribution function F_Y, the pth **quantile** of F_Y is the value y_p such that $F_Y(y_p) = p$. Note that the **median** of F_Y corresponds to $p = 0.5$.

The **sample mean** of the observations $Y_1, Y_2, \ldots, Y_n$ is given by $\bar{Y} = (1/n)\sum_{i=1}^n Y_i$.

If the values $Y_1, Y_2, \ldots, Y_n$ are rearranged so that $Y_{(1)} \le Y_{(2)} \le \cdots \le Y_{(n)}$ then $Y_{(i)}$ is the ith **order statistic**.

The **mean** μ_Y of the process is the asymptotic mean of the output process $\{Y_i\}$. The **variance** σ_Y^2 of the process is the asymptotic mean of the output process $\{(Y_i - \bar{Y})^2\}$.

The **probability** $P[A]$ of event A is the asymptotic mean of the output process $\{I(A)\}$, where I is the 0−1 indicator function for event A.

The output process $Y_1, Y_2, \ldots$ is **covariance stationary** if, for finite mean μ and finite variance $\sigma^2 > 0$, $E[Y_i] = \mu$, $i = 1, 2, \ldots$, $\mathrm{Var}[Y_i] = \sigma^2$, $i = 1, 2, \ldots$, and $\mathrm{Cov}[Y_i, Y_{i+j}]$ is independent of i, for $j = 1, 2, \ldots$.

Variance reduction techniques are strategies for obtaining greater precision for a fixed amount of sampling.

Facts:

1. The two most common measures of performance to be estimated are means and quantiles.

2. Point estimates for $\mu_Y, \sigma_Y^2, P[A]$ are typically given by the associated sample means.

3. A simple estimator of $y_p = F_Y^{-1}(p)$ is $Y_{(s)}$, where $s = \lfloor p(n+1) \rfloor$. This estimator can be improved (with respect to bias) by estimating $F_Y^{-1}(p)$ with the linear combination $(1-\alpha)Y_{(s)} + \alpha Y_{(s+1)}$, where $\alpha = p(n+1) - \lfloor p(n+1) \rfloor$.

4. One of the simplest methods of interval estimation is based on *replication*, in which several runs of a simulation model are used. Classical confidence intervals based on the central limit theorem for the measures of interest can then be applied to the output.

5. The presence of autocorrelation among observations (e.g., the wait times of adjacent customers in a queue) significantly complicates the statistical analysis of simulation output from a single run.

6. To analyze a single simulation run with autocorrelation present, techniques have been developed for determining interval estimates whose actual coverage is close to the stated coverage. For many of these techniques, the output is assumed to be covariance stationary. These techniques include batch means, overlapping batch means, standardized time series, regeneration, spectral analysis, and autoregression.

7. Variance reduction can be carried out using *common random numbers*, in which two or more alternative system configurations are analyzed using the same set of random numbers for particular purposes (e.g., generating service times). Using common random numbers insures that the output differences are due to the configurations rather than the sampling variability in the random numbers.

8. A second variance reduction technique, *antithetic variates*, is applicable to the analysis of a single system. If the random numbers $\{U_i\}$ are used for a particular purpose in one simulation run, then using $\{1 - U_i\}$ in a second run typically induces a negative correlation between the outputs of the two runs. Thus the average of the output measures from the two runs will have a reduced variance. Wilson [Wi84] overviews variance reduction techniques.

9. Additional topics in output analysis that have not been discussed here include: initialization bias detection, ranking and selection, comparing alternative system designs, experimental design, and optimization.

Examples:

1. (*Confidence Intervals for Expected Waiting Times*) Let $X_1, X_2, \ldots, X_n$ be the averages of the waiting times of customers in a single-server queue from n independent replications of a discrete-event simulation model. A $100(1-\alpha)\%$ confidence interval for μ, the steady-state mean waiting time in the queue, is

$$\bar{X} - t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} < \mu < \bar{X} + t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} ,$$

where $\bar{X}$ is the sample mean, s is the sample standard deviation, and $t_{\alpha/2,n-1}$ is the $1-\alpha/2$ fractile of the t distribution with $n-1$ degrees of freedom. The replications must be "warmed up" to avoid initialization bias. The asymptotic normality of $X_1, X_2, \ldots, X_n$ is assured by the central limit theorem and independence is based on the use of independent random number streams.

2. (*Common Random Numbers*) Law and Kelton [LaKe91, pp. 620–621] compare the $M/M/1$ and $M/M/2$ queueing models with a utilization of 0.9 using the waiting times in the queue of the first 100 customers. With $n = 100$ independent replications of each system, they compare the two models in four ways:

- independent runs (I)
- arrival streams using common random numbers (A)
- service times using common random numbers (S)
- arrival streams and service times using common random numbers ($A\&S$)

Common random numbers is a variance reduction technique that feeds identical interarrival and/or service times into the two different queueing models to increase the likelihood that observed differences in the waiting times are due to the system configurations ($M/M/1$ versus $M/M/2$) rather than sampling error. The half-widths of the confidence intervals ($\alpha = 0.10$) from Example 1 that they reported are $0.70(I), 0.49(A), 0.49(S)$, and $0.04(A\&S)$.

7.9.3 SIMULATION LANGUAGES

This section considers the history and features of simulation programming languages developed over the years. The use of a general-purpose simulation programming language (SPL) expedites model development, input modeling, output analysis, and animation. In addition, SPLs have accelerated the use of simulation as an analysis tool by bringing down the cost of developing a simulation model. In a history of the development of SPLs from 1955 to 1986, Nance [Na93] defines six requirements that a SPL must meet:

- random number generation
- variate generation
- list processing capabilities so that objects can be created, altered, and deleted
- statistical analysis routines
- summary report generators
- a timing executive or event calendar to model the passage of time

These SPLs may take the form of (1) a set of subprograms in a general purpose language (GPL) such as FORTRAN or C that can be called to meet the six requirements, (2) a preprocessor that converts statements or symbols to lines of code in a GPL, or (3) a conventional programming language.

Table 1 shows a division of the historical period into five distinct periods, including the names of several languages that came into existence in each period.

TABLE 1 Historical development of simulation languages

Period	Characteristics	Languages
1955–1960	period of search	GSP
1961–1965	the advent	CLP, CSL, DYNAMO, GASP, GPSS, MILITRAN, OPS, QUIKSCRIPT, SIMSCRIPT, SIMULA, SOL
1966–1970	formative period	AS, BOSS, Q-GERT, SLANG, SPL
1971–1978	expansion period	DRAFT, HOCUS, PBQ, SIMPL
1979–1986	consolidation and regeneration	INS, SIMAN, SLAM

The General Purpose System Simulator (GPSS) was first developed on various IBM computers in the early 1960's. Its block semantics were ideally suited for queueing simulations. Algol-based SIMULA was also developed in the 1960's and had features that were ahead of its time. These included abstract data types, inheritance, the co-routine concept, and quasi-parallel execution. SIMSCRIPT was developed by the RAND Corporation with the purpose of decreasing model and program development times. SIMSCRIPT models are described in terms of entities, attributes, and sets. The syntax and program organization was influenced by FORTRAN. The Control and Simulation Language (CSL) takes an "activity scanning" approach to language design, where the activity is the basic descriptive unit. The General Activity Simulation Program (GASP), as with several of the other languages, used flow-chart symbols to bridge the gap between personnel unfamiliar with programming and programmers unfamiliar with the application area. Although originally written in Algol, GASP provided FORTRAN subroutines for list-processing capabilities (e.g., queue insertion). GASP was a forerunner to the Simulation Language for Alternative Modeling (SLAM) and SIMulation ANalysis (SIMAN) languages. SLAM was the first language to include three modeling perspectives in one language: network (process orientation), discrete-event, and continuous (state variables). SIMAN was the first major SPL executable on an IBM PC.

REFERENCES

[BrEtal87] P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation*, 2nd ed., Springer–Verlag, New York, 1987.

[De86] L. Devroye, *Non-Uniform Random Variate Generation*, Springer–Verlag, New York, 1986.

[FlWi93] M. Flanigan–Wagner and J. R. Wilson, "Using univariate Bézier distributions to model simulation input processes". In *Proceedings of the 1993 Winter Simulation Conference*, G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, eds., 365–373, Institute of Electrical and Electronics Engineers, San Francisco, 1993.

[Jo87] M. E. Johnson, *Multivariate Statistical Simulation*, Wiley, New York, 1987.

[LaKe91] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd ed., McGraw–Hill, New York, 1991.

[Na93] R. E. Nance, "A history of discrete event simulation programming languages", *ACM SIGPLAN Notices*, 28 (1993), 149–175.

[PaMi88] S. K. Park and K. W. Miller, "Random number generators: good ones are

hard to find", *Communications of the ACM*, 31 (1988), 1192–1201.
[Sc90] B. Schmeiser, "Simulation experiments". In *Handbooks in OR & MS*, D. P. Heyman and M. J. Sobel, eds., 296–330, Elsevier Science Publishers, New York, 1990.
[Wi84] J. R. Wilson, "Variance reduction techniques for digital simulation", *American Journal of Mathematical and Management Sciences*, 4 (1984), 277–312.