Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

SIMULATION 101 SOFTWARE: WORKSHOP AND BEYOND

Barry Lawson

Department of Mathematics and Computer Science
University of Richmond
Richmond, VA 23173-0001, U.S.A.

Lawrence Leemis

Department of Mathematics
College of William & Mary
Williamsburg, VA 23187-8795, U.S.A.

ABSTRACT

The C source code associated with the Simulation 101 pre-conference workshop (offered at the 2006 and 2007 Winter Simulation Conferences) is presented here. This paper begins with general instructions for downloading, compiling, and executing the software. This is followed by sections on four groups of the software, categorized by functionality: libraries, Monte Carlo simulations, discrete-event simulations, and utilities. The libraries contain code to generate random numbers, code to generate random variates, and code to evaluate probability density functions, cumulative distribution functions, and inverse distribution functions. The Monte Carlo simulations consist of six programs that estimate various probabilities associated with simple probability problems, some with known analytic solutions and others without analytic solutions. The discrete-event simulations consist of various applications from queueing and inventory systems. Finally, the utilities are used to calculate various point and interval estimators from data sets.

1 INTRODUCTION

This paper discusses the use of the simulation software provided with the Simulation 101 workshop and associated with the introductory simulation textbook by Leemis and Park (2006).

2 SOFTWARE INSTALLATION AND EXECUTION

This section describes where to obtain and how to compile and execute the simulation software. Note that more specifics about execution of individual programs will be discussed on a program-by-program basis in Section 3.

2.1 Obtaining the Software

The software is freely available via ⟨http://math.wm.edu/~leemis/Sim101_SourceCode.zip⟩ as a Windows-friendly zip file or via ⟨http://math.wm.edu/~leemis/Sim101_SourceCode.tgz⟩ as a GNU-zip tarfile. Download the file corresponding to the archive type of your choice.

To extract the contents of the zip version, use any standard zip utility (e.g., WinZip for Windows, Stuffit Expander for Mac OS X, `zip` for Unix/Linux). To extract the contents of the GNU-zip tar file, execute the following command:

```
tar -xzvf Sim101_SourceCode.tgz
```

The source code will be placed into a subdirectory named `Sim101_SourceCode/` within the directory from which you initiate the extraction process.

2.2 Compiling and Executing

The software is ANSI C compliant, and so may be compiled by any compiler or development environment that supports ANSI C compilation.

We recommend using the GNU `gcc` compiler, which in the `Makefile` provided with the software is assumed to be the default compiler. If you use `gcc` and the associated `make` utility, following are commands of interest.

- `make`: Compiles and links all software, creating any of the executables that do not exist and executables for any of the source files that were modified since the last compile. The make utility creates executable files having the same names as the associated source files (e.g., the source file `galileo.c` results in the executable file `galileo`).
- `make clean`: Removes all executable and object files.
- `make` *programName*: Compiles and links only those files necessary for *programName*.

If you are not using `make`, then you will need to configure your compiler to appropriately link and compile the necessary software. When compiling any of the provided

source files that include local versions of header (`.h`) file(s), you will need to include the corresponding `.c` file(s) in the compilation and linking process. For example, `ssq4.c` includes both `rngs.h` and `rvgs.h` and so the compilation and linking process must include `rngs.c` and `rvgs.c`:

```
gcc -o ssq4 ssq4.c rngs.c rvgs.c -lm
```

When compiling any source file (e.g., `ssq4.c`) that includes the math library header file `math.h`, you may need to explicitly signify to the compiler to include the math library in the linking process, typically with a `-lm` flag (see the example above).

To execute any of the programs, simply provide the name of the executable as a command, e.g., `./ssq4`. (Note that certain of the source files are to be used as libraries and not to directly create an executable—see Section 3.1.)

3 SOFTWARE OVERVIEW

The Simulation 101 software is easily categorized into four basic groups: libraries, Monte Carlo simulations, discrete-event simulations, and utilities. We also provide an overview of the capabilities and uses of each of the programs. The programs are listed in alphabetical order within each of the groups.

3.1 Libraries

The programs in this group are to be used as libraries for other simulation programs (e.g., the Monte Carlo or discrete-event simulation programs in the next two subsections, or any simulation program you may create from scratch). The libraries provide a multiple-stream Lehmer random number generator (`rngs`), a collection of functions to generate random variates from various distributions (`rvgs`), and utility functions associated with each of the distributions (`rvms`). These are described in more detail below.

rngs: This library implements the functions for a good Lehmer random number generator with capability for multiple (255 by default) streams of random numbers, one per stochastic component of a simulation program. The primary functions of interest provided by this library are as follows:

- `double Random()`: returns a pseudo-random number uniformly distributed between 0.0 and 1.0. The period of the generator is $(m-1)$ distinct random numbers, where $m = 2^{31}-1$. The smallest and largest possible values are $1/m$ and and $(m-1)/m$ respectively. More details are provided in Park and Miller (1988).
- `void PlantSeeds(long x)`: initializes the state of each of the streams. Typically, this function is called once, at the beginning of the simulation program prior to any calls to `Random()`. The value `x` is used to initialize the state of the default stream, and then all remaining streams are initialized appropriately. If `x` is positive, `x` is the state; if `x` is negative, the state is obtained using the system clock; if `x` is 0, the state is to be supplied interactively.
- `void SelectStream(int index)`: sets the current random number generator stream, i.e., the stream from which the next random number will come. A unique stream (i.e., value for `index`) should be used for each distinct stochastic component of a simulation program.

Note that certain applications (though none of the ones presented in the workshop) may have more stringent requirements or need a larger period than the given generator can provide. In those cases, you should consider a different random number generator—see, e.g., L'Ecuyer (1999).

rvgs: This library provides functions for generating random variates from six discrete distributions: *Bernoulli*(p), *Binomial*(n,p), *Equilikely*(a,b) [also known as *Discrete Uniform*], *Geometric*(p), *Pascal*(n,p) [also known as *Negative Binomial*], and *Poisson*(μ). The library also provides functions for generating random variates from seven continuous distributions: *Uniform*(a,b), *Exponential*(μ), *Erlang*(n,b), *Normal*(μ,σ), *Lognormal*(a,b), *Chisquare*(n), and *Student*(n). Refer to the comments in `rvgs.c` for a characterization of the function parameters.

rvms: This library provides functions to evaluate the probability density functions, cumulative distribution functions, and inverse distribution functions (used in the commonly-known probability integral transformation) for the six discrete and seven continuous random variable models provided in `rvgs`. Refer to the comments in `rvms.c` for a characterization of the function parameters.

3.2 Monte Carlo Simulations

This group of programs provides various examples of the power of Monte Carlo simulation, estimating one or more probabilities using the functions provided in the random number generator libraries discussed above. Note that, except for `san`, the corresponding Monte Carlo simulation programs provided with the textbook by (Leemis and Park 2006) use a simpler, single-stream version of `rngs`. For simplicity, all programs provided for the workshop rely only `rngs`.

After compiling (see Section 2), any of these Monte Carlo simulations may be executed by using the program name, e.g., `./buffon`. The number of replications used, initial seed input, etc. may be changed by modifying the

corresponding source code and then recompiling. Refer to Leemis and Park (2006) for more details and, for those of the problems analytically tractable, corresponding analytical solutions.

buffon: produces an estimate of the probability that a needle of length $r > 0$ will cross at least one line when dropped at random onto a plane of infinitely many vertical lines, each infinitely long and spaced one unit apart.

craps: produces an estimate of the probability of winning the simple dice game Craps played with two fair dice.

det: produces an estimate of the probability that the determinant of a 3×3 matrix of random numbers having a particular sign pattern is positive. The number of replications is uncharacteristically large for this simulation, so the execution time will be significantly longer than for the other simulations in the software suite.

galileo: produces an estimate of the probability of each sum $3, 4, \ldots, 18$ obtained when rolling three fair dice.

hat: produces an estimate of the probability that a hat check girl will return all n hats to the wrong owners when she returns n hats at random.

san: produces an estimate of the mean time to complete a stochastic activity network, a classic problem that arises in project management.

3.3 Discrete-Event Simulations

This group of programs provides examples of discrete-event simulation models, specifically focusing on queuing (`ssq` and `msq`), inventory systems (`sis`), and machine shop (`ssms`) models. For pedagogical reasons, the inventory systems and queueing model programs are numbered according to increasing complexity. These programs also provide excellent examples of the use of the `rngs` and `rvgs` library capabilities. Discrete-event simulation differs fundamentally from Monte Carlo simulation in that the passage of time plays a significant role.

After compiling (see Section 2), any of these simulation programs may be executed by using the program name, e.g., `./msq`. The number of replications used, initial seed input, etc. may be changed by modifying the corresponding source code and then recompiling.

msq: implements a next-event simulation of a multiple-server, single-queue queuing model. This program relies on the `rngs` library.

sis{1,2,3,4}: implement progressively more complex versions of a simple inventory system model. `sis1` uses a process-interaction world view implementation and is trace-driven (no use of random numbers), requiring the trace file `sis1.dat`. `sis2` is a modification of `sis1` to use the `rngs` library. `sis3` uses a next-event implementation and adds backlogging and delivery lag to the inventory system model. `sis4` is a modification of `sis3` to use a more realistic Exponential/Geometric demand model.

ssms: an extension of `ssq2` (see below), this program implements a single-server machine shop model using a process-interaction world view, with Exponentially distributed failure times, Uniformly distributed service times, and a FIFO service queue. This program relies on the `rngs` library.

ssq{1,2,3,4}: implement progressively more complex versions of a single-server FIFO service node. `ssq1` uses a process-interaction world view implementation and the arrival and service processes are trace-driven, requiring the trace file `ssq1.dat`. `ssq2` is a modification of `ssq1` to use the `rngs` library, implementing Exponentially distributed interarrival times and Uniformly distributed service times (i.e., an $M/U/1$ queue). `ssq3` is a next-event implementation of the same $M/U/1$ queue from `ssq2`. `ssq4` is a modification of `ssq3` to use a more realistic Erlang service-time model.

ttr: implements a next-event simulation of a think-type-receive timesharing system. The main purpose of this program is to facilitate evaluating the effectiveness of different event-list implementations. This program relies on the `rngs` library.

3.4 Utilities

The following collection of utility programs are provided mainly to aid in analyzing the output of a simulation program. The utilities, however, can be used in a more general context—e.g., `uvs` may be used to compute the mean, standard deviation, minimum, and maximum for any general set of data. After compiling (see Section 2), any of these utilities may be executed by using the program name, e.g., `./uvs`.

acs: computes autocorrelation statistics for a given set of data using a one-pass algorithm. More specifically, for a given sample this utility computes the sample autocorrelation for the first k autocorrelation lags (k should be much smaller than the sample size). This program is designed to be used with redirection, whether piping the standard output of a simulation program directly to `acs`, e.g.,

```
./generateOutput | ./acs
```

or by redirecting the contents of an existing output file, e.g.,

```
./acs < outputFile
```

where the program output or the output file contains one data point per line. A sample data file `acs.dat` is provided.

bvs: computes bivariate statistics for a given set of data. More specifically, this utility reads bivariate data (assumed to be two values per line, with the data values on each line separated by one or more spaces) and computes the

mean, standard deviation, minimum, maximum, correlation coefficient, and regression line angle. Like `acs`, `bvs` is designed to be used with redirection. A sample data file `bvs.dat` is provided.

cdh: computes a continuous-data histogram in tabular (not graphical) format for a set of continuous data. The table displays the midpoint, data-point count, proportion, and density for each bin, as well as the overall sample size, mean, and standard deviation. The minimum and maximum data values and the number of bins are hard-coded in `cdh.c` and as given are tuned for the data set provided in the file `uvs.dat`. To use `cdh` on a different data set, you should modify the minimum, maximum, and number of bins appropriately (perhaps using `uvs` to gain insight) and then recompile. Like the other provided utilities, `cdh` is designed to be used with redirection. The provided data file `uvs.dat` can be used as an example.

ddh: computes a discrete-data histogram in tabular (not graphical) format for a set of discrete data. The table displays each discrete-data value and corresponding count and proportion, as well as the overall sample size, mean, and standard deviation. Unlike `cdh`, `ddh` requires no parameter modification. If the data is not discrete, `cdh` should be used instead. Like the other provided utilities, `ddh` is designed to be used with redirection. A sample data file `ddh.dat` is provided.

estimate: computes a confidence interval estimate for a given set of data. More specifically, this utility reads sample data, either continuous or discrete with one data point per line, and computes an interval estimate for that (unknown) larger set of data from which the sample was drawn. The utility uses Welford's one-pass algorithm (Welford 1962) to compute the sample mean and standard deviation. The level of confidence (95% by default) is hard-coded in `estimate.c`—modify as appropriate and recompile. Like the other provided utilities, `estimate` is designed to be used with redirection.

uvs: computes univariate statistics for a given set of data. More specifically, this utility reads sample data, either continuous or discrete with one data point per line, and computes the mean, standard deviation, minimum, and maximum. The utility uses Welford's one-pass algorithm (Welford 1962). Like the other provided utilities, `uvs` is designed to be used with redirection. A sample data file `uvs.dat` is provided. In addition, `uvs` (as well as the other utilities, though less conveniently) can be used with keyboard input: execute the utility (e.g., `./uvs`), enter the data one value per line, and signify end-of-file at the end of the input (Control-d in Unix/Linux).

4 SUMMARY

For the software provided with workshop, those programs in the Monte Carlo and discrete-event simulation groups are not general-purpose, but are intended as instructional tools for learning the concepts of simulation and as references in the use of the provided libraries and utilities. The library programs and the utility programs are general-purpose and may be used with any simulation programs you may develop. For further information, please contact either of the authors of this paper.

REFERENCES

L'Ecuyer, P. 1999, January. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research* 47 (1): 159–164.

Leemis, L. M., and S. K. Park. 2006. *Discrete-event simulation: A first course*. Upper Saddle River, NJ: Pearson Prentice Hall.

Park, S. K., and K. W. Miller. 1988, October. Random number generators: Good ones are hard to find. *Communications of the ACM* 31 (10): 1192–1201.

Welford, B. P. 1962, August. Note on a method for calculting corrected sums of squares and products. *Technometrics* 4 (3): 419–420.

AUTHOR BIOGRAPHIES

BARRY LAWSON is Assistant Professor of Computer Science in the Department of Mathematics and Computer Science at University of Richmond. He received Ph.D. and M.S. degrees in Computer Science from the College of William & Mary, and a B.S. degree in Mathematics and Computer Information Systems from University of Virginia's College at Wise. His current research interests include computer security, parallel and distributed computing, scheduling, performance analysis, and discrete-event simulation. He previously worked in the Simulation Systems Branch laboratory at NASA Langley in Hampton, VA. His email address is ⟨blawson@richmond.edu⟩.

LAWRENCE LEEMIS is a professor in the Mathematics Department at the College of William & Mary. He received his B.S. and M.S. degrees in Mathematics and his Ph.D. in Industrial Engineering from Purdue University. He has also taught at Baylor University, The University of Oklahoma, and Purdue University. His consulting, short course, and research contract work includes contracts with AT&T, NASA/Langley Research Center, Delco Electronics, Department of Defense (Army, Navy), Air Logistic Command, ICASE, Komag, Federal Aviation Administration, Tinker Air Force Base, Woodmizer, Magnetic Peripherals, and Argonne National Laboratory. His research and teaching interests are in reliability and simulation. He is a member of ASA, IIE, and INFORMS. His email address is ⟨leemis@math.wm.edu⟩.