SSJ: Stochastic Simulation in Java

Overview

Version: September 12, 2007

SSJ is a Java library for stochastic simulation, developed in the Département d'Informatique et de Recherche Opérationnelle (DIRO), at the Université de Montréal. It provides facilities for generating uniform and nonuniform random variates, computing different measures related to probability distributions, performing goodness-of-fit tests, applying quasi-Monte Carlo methods, collecting statistics (elementary), and programming discrete-event simulations with both events and processes. Additional Java packages are also developed on top of SSJ for simulation applications in finance, call centers management, communication networks, etc.

Introduction and overview

Simulation models can be implemented in many ways [4]. One can use general-purpose programming languages such as FORTRAN, C, C++, Java, or specialized simulation languages such as *GPSS*, *SIMAN*, and *SIMSCRIPT*. The general-purpose languages may be more familiar to the programmer, but usually do not have the necessary built-in tools to perform simulation. Implementing a model can become complex and tedious. Specialized simulation languages must be learned before models can be implemented, and they are not as widely available and supported as the most popular general-purpose languages.

Over the past few decades, commercial simulation tools with point-and-click graphical user interfaces such as *Arena*, *Automod*, *Witness*, and many others, have become by far the most widely used tools to develop simulation models. Among their main advantages, these tools do not require knowledge of a programming language, provide graphical animation, have automatic facilities to collect statistics and perform experiments, and can sometimes perform optimization to a certain extent. On the other hand, these specialized simulation tools, especially the point-and-click tools, are often too restrictive, because they are usually targeted at a limited class of models. With these tools, simulating a system whose logic is complicated or unconventional may become quite difficult. All the graphical and automatic devices also tend to slow down the simulation significantly. Fast execution times are important for example in a context of optimization, where thousands of variants of a base system may have to be simulated, or for on-line applications where a fast response time is required.

SSJ is an organized set of packages whose purpose is to facilitate simulation programming in the Java language. A first description was given in [5]. Some of the tools can also be used for modeling (e.g., selecting and fitting distributions). As these lines are being written, SSJ is still growing. Several new packages, classes, and methods will certainly be added in forthcoming years and others will be refined.

The facilities offered are grouped into different packages, each one having its own user's guide, in the form of a PDF file. There is also a set of commented examples of simulation programs in a separate directory with its own guide. Programs are given for some of the examples used in the books of Law and Kelton [4] and Glasserman [1], for instance. The best way to learn about SSJ, at the beginning, is probably to study these examples and refer to the user guides of the different packages when needed. The PDF files are the official documentation. There is also a simplified on-line documentation in HTML format, produced via `javadoc`.

The packages currently offered are the following:

`util` contains utility classes used in the implementation of SSJ, and which are often useful elsewhere. For example, there are timers (for CPU usage), utilities to read or format numbers and arrays from/to text, operations on binary vectors and matrices, some mathematical functions and constants, root-finding tools, facilities for SQL database interface, and so on.

`probdist` contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, and inverse distribution functions for many discrete and continuous probability distributions, as well as estimating the parameters of these distributions.

`probdistmulti` contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, for some multi-dimensionnal discrete and continuous probability distributions.

`randvar` provides a collection of classes for non-uniform random variate generation, primarily from standard distributions.

`randvarmulti` provides a collection of classes for random number generators for some multi-dimensional distributions.

`rng` provides facilities for generating uniform random numbers over the interval $(0, 1)$, or over a given range of integer values, and other types of simple random objects such as random permutations.

`hups` provides classes implementing highly uniform point sets and sequences (HUPS), also called low-discrepancy sets and sequences, and tools for their randomization.

`gof` contains tools for performing univariate goodness-of-fit (GOF) statistical tests.

`stat` provides elementary tools for collecting statistics and computing confidence intervals.

`stat.list` this subpackage of `stat` provides support to manage lists of statistical collectors.

`simevents` provides and manages the event-driven simulation facilities as well as the simulation clock. Can manage several simulations in parallel, in the same program.

`simevents.eventlist` this subpackage of `simevents` offers several kinds of event list implementations.

`simprocs` provides and manages the process-driven simulation facilities.

`functions` contains classes that allow one to pass an arbitrary function of one variable as argument to a method and to apply elementary mathematical operations on generic functions.

Dependence on other libraries

SSJ uses some classes from the `Colt` library. The `probdist` package also use an external linear algebra library and an optimization package to compute maximum likelihood estimators. Finally, an interface to the UNURAN library is offered. We now describe these external libraries.

The Colt library, developed at the Centre Européen de Recherche Nucléaire (CERN) in Geneva [2], is a large library that provides a wide range of facilities for high performance scientific and technical computing in Java. SSJ uses the class `DoubleArrayList` from Colt in a few of its classes, namely in packages `stat` and `hups`. The reason is that this class provides a very efficient and convenient implementation of an (automatically) extensible array of `double`, together with several methods for computing statistics for the observations stored in the array (see, e.g., `Descriptive`). The Colt library is distributed with the SSJ package. Here is the Colt License Agreement copied from the Colt web site:

Colt License Agreement

SSJ also provides an interface to the UNURAN library for nonuniform random number generation [6], in the `randvar` package. UNURAN does not have to be installed to be used with SSJ, because it is linked statically with the appropriate SSJ native library. However, the UNURAN documentation will be required to take full advantage of the library.

The linear_algebra library is based on public domain LINPACK routines. They were translated from Fortran to Java by Steve Verrill at the USDA Forest Products Laboratory Madison, Wisconsin, USA. This software is also in the public domain and is included in the SSJ distribution as the **Blas.jar** archive, which must be in the CLASSPATH environment variable. It is used only in the `probdist` package to compute maximum likelihood estimators.

The optimization package of Steve Verrill includes Java translations of the MINPACK routines [3] for nonlinear least squares problems as well as UNCMIN routines [7] for unconstrained optimization. They were translated from Fortran to Java by Steve Verrill and are in the public domain. They are included in the SSJ distribution as the **optimization.jar** archive, which must be in the CLASSPATH environment variable. It is used only in the `probdist` package to compute maximum likelihood estimators.

Acknowledgments

SSJ was designed and implemented under the supervision of Pierre L'Ecuyer, with the contribution of the following persons

Mathieu Bague, Sylvain Bonnet, Éric Buist, Yves Edel, Regina H. S. Hong, Alexander Keller, Pierre L'Ecuyer, Étienne Marcotte, Lakhdar Meliani, Abdelazziz Milib, François Panneton, Richard Simard, Pierre-Alexandre Tremblay, Jean Vaucher.

Its development has been supported by NSERC-Canada grant No. ODGP0110050, NATEQ-Québec grant No. 02ER3218, a Killam fellowship, and a Canada Research Chair to the author.

References

[1] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.

[2] Wolfgang Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. CERN, Geneva, 2004. Available at `http://dsd.lbl.gov/~hoschek/colt/`.

[3] J. J. Moré and B. S. Garbow and K. E. Hillstrom. *User Guide for MINPACK-1, Report ANL-80-74*. Argonne, Illinois, USA, 1980. See `http://www-fp.mcs.anl.gov/otc/Guide/softwareGuide/Blurbs/minpack.html`.

[4] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, NY, third edition, 2000.

[5] P. L'Ecuyer, L. Meliani, and J. Vaucher. SSJ: A framework for stochastic simulation in Java. In E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 234–242. IEEE Press, 2002.

[6] J. Leydold and W. Hörmann. *UNURAN—A Library for Universal Non-Uniform Random Number Generators*, 2002. Available at `http://statistik.wu-wien.ac.at/unuran`.

[7] R. B. Schnabel. *UNCMIN—Unconstrained Optimization Package, FORTRAN*. University of Colorado at Boulder. See `http://www.ici.ro/camo/unconstr/uncmin.htm`.