S.G. Henderson and B.L. Nelson (Eds.), *Handbook in OR & MS, Vol. 13*

DOI: 10.1016/S0927-0507(06)13007-8

Chapter 7
Implementing Representations of Uncertainty

W. David Kelton

Department of Quantitative Analysis and Operations Management,
University of Cincinnati, USA
E-mail: david.kelton@uc.edu

Abstract

Chapters 3–6 discussed generating the basic random numbers that drive a stochastic simulation, as well as algorithms to convert them into realizations of random input structures like random variables, random vectors, and stochastic processes. Still, there could be different ways to implement such methods and algorithms in a given simulation model, and the choices made can affect the nature and quality of the output. Of particular interest is precisely how the basic random numbers being generated might be used to generate the random input structures for the model. This chapter discusses these and related issues and suggests how implementation might best proceed, not only in particular simulation models but also in simulation software.

1 Introduction

The preceding chapters in Part II of this volume (Chapters 3–6) discussed details of generating the basic random numbers at the root of stochastic simulations, and the ensuing methods for generating the various kinds of random structures (variates, multivariate input processes, arrival processes, random lifetimes, and other objects) that are used directly to drive the simulation.

The purpose of this (very) short chapter is merely to raise some issues, and suggest some potential resolutions to them, for precisely how such generation might best be implemented in the context of a large and complex simulation, especially in terms of the low-level assignment of specific sequences of random numbers to generating the various random structures listed above. You might want to read this if you are a simulation analyst interested in effecting some of these ideas (software permitting), or if you are a simulation-software developer considering implementing some of this in your products, either as defaults or as user-controlled options, to help your customers do a better job with their simulation studies.

The implementation of random-number generation for various random structures matters for reasons of both *validity* of the output (i.e., accuracy), as well as for its *precision*:

- *Validity*. If, as is quite possible (if not almost certain), sequences of random numbers are re-used in an uncontrolled or inappropriate way (most likely unknowingly), the simulation's very validity can be threatened. The reason is that many variate- and process-generation schemes rely on independent draws from the underlying random-number generator (as discussed in Chapters 4–6), and if care is not taken to eliminate the possibility of uncontrolled overlapping or re-using sequences of random numbers, these variate- and process-generation methods can become invalid – and hence the simulation's results could also be invalid.
- *Precision*. You could easily lose an opportunity to improve your simulation's efficiency (either computational or statistical, which can be viewed as the same thing) if you pass up what might be some relatively easy (maybe even automatic, as described in Section 5) steps involving intelligent re-use of random numbers. Now the preceding bullet point railed against re-use of random numbers, claiming that doing so could invalidate your simulation, and now it's being suggested that doing so can improve it (while not harming its validity). The difference is that now what's being considered is controlled and intelligent re-use, as opposed to uncontrolled or even unconscious re-use. A more detailed treatment of the probability and statistics behind this possibility, with an example, is in Section 4.

Section 2 discusses how the underlying random-number generator might be set up in terms of accessing and using it. Section 3 considers how such a random-number-generator structure might be exploited to organize and control how input variates, vectors, processes, and other objects are generated in the interest of improving the simulation's efficiency and thus value. In Section 4 an application of these ideas, to inducing desirable correlation to reduce simulation-output variance, is described along with an example. Finally, Section 5 offers some (personal) opinions about how these ideas might be implemented in high-level simulation-modeling software, which would help everyone simulate better.

Some of this material (and additional related issues, such as robust and automated integration into software of both input modeling and output analysis) appeared in Kelton (2001).

2 Random-number generation

Use of a modern, statistically valid and long-period random-number generator is absolutely critical to any simulation study (see Chapter 3 for a full

treatment of this topic). Somewhat alarmingly, we now have a severe mismatch between computer-hardware capabilities and the characteristics of some older random-number generators still in common use.

First and foremost, random-number generators must be of high statistical quality, and produce a stream of numbers that behave as though they were independent and uniformly distributed on the unit interval, even when scrutinized closely for these properties by powerful structural and statistical tests. Given the speed of computers several decades ago, generators were developed at that time that were adequate to "fool" these tests up to the discriminatory power of the day. One aspect of this is the *cycle length* (or *period*) p of a generator, which is the number of random numbers before the generator repeats itself (as all algorithmic generators eventually will). Starting from some fixed deterministic initialization, the complete sequence of generated random numbers would be $u_1, u_2, \ldots, u_p$ and that is all, since $u_{p+1} = u_1, u_{p+2} = u_2, \ldots$, and in general $u_{kp+i} = u_i$ for $k \in \{1, 2, \ldots\}$. In the 1950s, 1960s, and 1970s, generators with cycle length p around 2^{31} (on the order of 10^9) were developed; this cycle length resulted from the word length of fixed-point integers in computers at that time. These were adequate for many years, and were in wide use. Unfortunately, they are *still* in wide use even though computer speeds have increased to the point that any garden-variety PC can completely exhaust this cycle in just a few minutes! At that point the stream cycles and produces exactly the same "random" numbers in exactly the same order, with obvious potentially deadly implications for the integrity of simulations' results. This dirty little software scandal (including simulation software) is one that few users (or software developers) seem to be aware of, or even seem to care about. But it is easily avoidable. People now have developed and coded algorithms for extremely long-period generators, with cycle lengths p on the order of 10^{57} or more, which display superb statistical behavior; see Chapter 3 for a general discussion, or L'Ecuyer et al. (2002) for a specific example with downloadable code. Their cycle lengths will continue to be "long", even under Moore's "law", for a few centuries to come. And their speed is comparable to the poor little old generators from decades ago. It is hard to think of any excuse at all for not implementing these kinds of generators immediately, especially in commercial simulation software.

Another highly desirable aspect of random-number generators is the ability to specify separate *streams* of the generator, which are really just (long) contiguous subsegments of the entire cycle, and to make these readily available, perhaps via on-the-fly object-oriented instantiation as opposed to storing static seed vectors to initiate each stream. With s streams of length l each ($sl = p$), we could re-index the generated random numbers as u_{ji} being the ith random number in stream j, for $i \in \{1, 2, \ldots, l\}$ and $j \in \{1, 2, \ldots, s\}$. These streams can be further subdivided into *substreams*, and so on, for multidimensional indexing (u_{jki} is the ith random number from substream k within stream j) and assignment of separate and independent chunks of random numbers to separate activities in the simulation; the importance of this is really in vari-

ance reduction via correlation induction, as discussed in Section 4, as well as in Banks et al. (2005, Chapter 12) and Law and Kelton (2000, Chapter 11). The specific generator in L'Ecuyer et al. (2002) provides a very large number of very long streams and substreams (i.e., two-dimensional indexing).

As an example of the utility of such multidimensional stream and substream indexing, you could assign a stream (first index j) to generating, say, service times of a class of parts at a workcenter, and within that stream you would (automatically, one would hope) move to the next substream (second index k) for each new replication of the simulation run. The importance of moving to a new substream with each replication is that, in comparing alternate *scenarios* of the model (e.g., change some input parameters or model logic), different scenarios will in general consume different numbers of random numbers for this purpose in a replication, and moving to the next substream for the next replication will ensure that in replications k subsequent to the first, the same random numbers will still be used for the same purpose; see the discussion of random-number *synchronization* in Section 4, Banks et al. (2005, Chapter 12), or Law and Kelton (2000, Chapter 11). Another way to ensure synchronization throughout multi-replication comparative runs, without substreams, would be to advance all stream numbers at the beginning of each replication after the first by the number of streams being used; with the astronomical number of streams that should be available, there would be no worries about running out of streams. It is easy to imagine the need for stream indexing of dimension higher than two in order to ensure synchronization in more complex projects involving designed experiments or comparing and ranking multiple scenarios (see Chapter 17) – u_{jkdi} is the ith random number for the jth source of randomness during the kth replication of design point d in a designed simulation experiment.

Obviously, being able to do this reliably and generally requires extremely long-period underlying generators, which, as pointed out, now exist. And we also have easy and fast methods to create and reference such multidimensional streams (and the length of even the substreams is astronomically long so there are no worries about exhausting or overlapping them); see, for example L'Ecuyer et al. (2002) for a package based on a generator with cycle length 10^{57}, streams and substreams (i.e., two-dimensional indexing), as well as downloadable code for implementation.

3 Random-structure generation

With multidimensional stream indexing based on a high-quality, long-period underlying random-number generator, it then becomes possible (ideally, automatic) to assign separate streams of random numbers to separate points in the model where uncertainty in the input is required. As mentioned, the reason for doing so is to ensure, so far as possible given the structure of the model(s), that in simulation studies comparing alternate scenarios of a general base model,

the same random numbers are used for the same purposes across the scenarios, to maintain the best synchronization and thus improve the performance of correlation-induced variance-reduction techniques (see Section 4).

Though precisely how this is done will always have some dependence on the model, there are generally two approaches, by *faucets* and by *body art*:

- *Faucets*. Identify the activities in the model where uncertainty is an input. Examples include interarrival times between successive customers, their service times at different stations, their priorities, and points at which probabilistic branching occurs. Assign a separate stream in the random-number generator to each. This is then like having a separate faucet at each such point, pouring out random numbers from a separate reservoir. True, the reservoirs are finite, but with the right underlying random-number generator broken appropriately into streams and substreams, etc., you do not need to worry about draining any reservoirs during your lifetime.
- *Body art*. Identify all the uncertain things that might happen to an entity (like a customer in a queueing model) during its life in the model, and write them on the entity when it is born; these are sometimes called *attributes* of the entity. Examples would include service times, pre-ordained probabilistic branching decisions, and the number of clones made if there is downstream copying of the entity. Use nonindelible ink to allow for possible reassignment of these attributes during the entity's life.

Neither faucets nor body art provides the universal or uniformly best way to go, since, as mentioned, particular model logic might render one or the other difficult or impossible (or meaningless).

It seems that body art would, in general, provide the best and strongest synchronization, since it is more in keeping with the principle that, for all scenarios, we want the "same" customers showing up at the same times (by the "same" customers we mean that they are identical in terms of exogenous traits like service requirements, arrival times, branching decisions, etc.). However, it also makes the entities bloated (thus consuming more memory) in terms of their having to drag around all these attributes when most are not needed until later (maybe some will never be needed).

Faucets are easy to implement (maybe they should be automatic in the software, which would automatically increment the stream/faucet index every time the modeler puts a source of randomness into the model), and in general provide a level of synchronization that is certainly better than nothing. However, given the vagaries of what might happen in a simulation (e.g., branching decisions), they would not appear necessarily to provide the "same-customer" level of synchronization that might be achieved by body art, but this is model-dependent (see the example in Section 4).

Another aspect of how simulations are actually coded (or how the simulation software being used was coded) that can affect the level of synchronization

is the method by which random variates (and vectors and processes) are generated, if there is a choice; this was discussed in Chapters 4–6 and many specific methods and references are in Banks et al. (2005, Chapter 8) and Law and Kelton (2000, Chapter 8). The opinion here is that inversion of the cumulative distribution function (or of a process analogue of it, such as the cumulative rate function for generating realizations of a nonstationary Poisson process, as in Çinlar, 1975) is preferable since:

- the number of random numbers consumed per variate (or vector) is fixed, thus making it easier to ensure synchronization,
- the sign and strength of the correlation between the basic random numbers and the objects realized is protected, and
- it generally uses the fewest number of random numbers (though this is the least important reason since in most complex dynamic systems simulation random-number generation accounts for only a small proportion of the overall execution time).

Now inversion is not universally applicable via simple closed-form formulas, but in some such cases a numerical method to effect the inversion is available and might be considered in order to realize the benefits above. True, this carries a computation-time penalty, but that might be justified by efficiency gains in the statistical sense (and thus ultimately in the computational sense as well if a specific level of precision in the output is sought). This is a consideration that might be brought to bear on that part of simulation-software design concerned with generation of random structures for input to a user's model.

4 Application to variance reduction

As mentioned above, one of the main applications and benefits of careful implementation of random-structure generation is in *variance reduction*. By taking a little care in allocation of random-number streams in the simulation model, it is often possible to achieve better output precision without additional computational effort.

Though not the only variance-reduction technique, one of the most accessible and widely used is *common random numbers* (CRN), also called *correlated sampling* or *matched pairs*. It applies not when simulating a single system configuration, but rather when comparing two (or more) alternative configurations, or *scenarios*, and uses the same random numbers to drive all scenarios, though care must be taken to do so wisely in order to get the most benefit.

One motivation for CRN is common sense – to compare alternative scenarios it seems best to subject them to the same "external" conditions. This way, any observed output differences could be more reliably attributed to differences in the scenarios rather than to the different external conditions that happened to occur in the runs, since such external differences are minimized.

This common sense is backed up by basic probability. If Y_A and Y_B are output responses from scenarios A and B, we might seek to estimate the expected difference $E(Y_A - Y_B)$ between the scenarios from simulation output data by $Y_A - Y_B$. One measure of the quality of this estimator, in terms of precision, is $\text{Var}(Y_A - Y_B) = \text{Var}(Y_A) + \text{Var}(Y_B) - 2\,\text{Cov}(Y_A, Y_B)$. If the covariance is zero, as would happen if the runs were independent, we get a higher variance than if we could make this covariance positive, as we would anticipate if the external conditions were controlled to be mostly the same across the simulations of A and B (barring pathological model behavior, we would expect that, say, heavy customer demands would cause A and B to respond in the same direction). Of course, this thinking goes through with multiple replications on a replication-by-replication basis, and the variance reduction propagates through to the sample means. One way to try to induce such positive covariance in comparative simulations is CRN.

For instance, a facility receives blank parts and processes them. About a fourth of the blank parts need repair before processing. There are currently two separate external sources; from each source, parts arrive one at a time with interarrival times that are exponentially distributed with mean one minute. There is a single repair person, with repair times' being exponential with mean 1.6 minutes. There are two processing operators, both "fed" by a single queue, with processing times' being exponential with mean 0.8 minute. All queues are first-come, first-served, and all probabilistic inputs (interarrival times, service times, and whether a part needs repair) are independent. The simulation starts empty and idle and runs for 24 hours. From 50 replications, a 95% confidence interval for the expected time in system of parts is 3.99 ± 0.19 minutes, and for the expected number of parts present is 8.00 ± 0.40.

Under consideration is taking on a third but smaller source of blank parts, with interarrival times' being exponential with mean five minutes, but otherwise the same in terms of service times and proportion that need repair. Adding this to the model and replicating 50 times, 95% confidence intervals on expected time in system and expected number of parts present are 6.79 ± 0.60 minutes and 15.01 ± 1.39 parts. It is hardly surprising that both went up, but it would have been hard to quantify by how much without a simulation.

How should we make statistically valid statements about the changes in the model responses? For example, are the changes statistically significant? One way is to pair up the results, replication by replication, subtract, and compute a confidence interval on the expected difference (assuming normal-theory statistics are valid, this is the *paired t* method discussed in any basic statistics book). Subtracting in the direction (original model)–(second model), we get 95% confidence intervals of -2.80 ± 0.59 minutes for the difference between expected times in system, and -7.00 ± 1.35 for the difference between expected numbers of parts present; since neither confidence interval contains zero, the differences are statistically significant.

Exactly what random numbers were used (or should have been used) in the comparison? We just ignored the whole issue above and used the default stream for everything in both scenarios, initialized at the same point. Thus, the results from the two scenarios arose from the same sequence of random numbers (the lengths of the sequences were very likely different, but there was substantial overlap) so are not independent, though fortunately the paired t method is still valid in this case. This is probably how the vast majority of simulation comparisons are done, so the same random numbers are in fact used, but not in a controlled way – call this the *default* approach. (We did, however, advance to the next substream for each new replication in both scenarios, as discussed in Section 2, so the random numbers started off the same way in each replication.)

To see if the default approach got us any variance reduction, we reran the comparison, again with 50 replications per scenario, using one stream for everything in A, and a different stream throughout B (and both different from the one used in the default comparison). Again, substreams were advanced for each replication. The 95% confidence intervals on the differences were -2.49 ± 0.47 minutes for time in system and -6.33 ± 1.11 for mean number of parts present, which appear to be of about the same precision as in the default approach. Indeed, a formal F test for equality of variances (on which the confidence-interval half-lengths are based) failed to find a statistically significant difference. So if default CRN achieved any variance reduction at all it was vanishingly small since there was no attempt to synchronize random-number use, even though essentially the same random numbers were used across A and B.

We next synchronized the comparison by faucets, assigning separate streams to each of the random inputs (one for each arrival source, one for the repair decision, one for repair times, and one for processing times), and using the same stream assignments for both A and B. Substreams, as discussed in Section 2, were used to maintain synchronization in replications subsequent to the first. From 50 replications, the confidence intervals on the differences were -2.32 ± 0.29 minutes for time in system and -5.99 ± 0.70 for mean number of parts present, both noticeably tighter than the default and independent approaches, indicating that faucets produced enough commonality in the external conditions to sharpen the comparison. Formally, the F test for equality of variances, in comparison with both the default and independent approaches was highly significant, confirming that synchronizing by faucets did reduce variance. Structurally, though, in this model faucets do not perfectly synchronize the repair decision on a part-by-part basis; this decision is made after parts arrive, and in scenario B additional parts are mixed into the incoming flow yet the sequence of repair decisions is the same. Thus, parts from the original two inputs are not all getting the same repair decisions they got in scenario A.

So finally we synchronized by body art, using separate streams to pre-assign as attributes to each part upon its arrival whether it needs repair, its repair time (which was ignored 3/4 of the time), and its processing time; the input

processes used still separate streams, and the same streams were used for these assignments in A and B (all streams here were different from those used above, so all these experiments are independent). From 50 replications this yielded the tightest confidence intervals of all, -2.33 ± 0.25 minutes for time in system and -5.98 ± 0.59 for mean number of parts present. However, an F test for equality of variances did not signal a statistically significant difference in comparison with faucets synchronization; even though body art makes more intuitive sense for this model than do faucets (due to the repair decision, as described at the end of the preceding paragraph), the impact of body art vs. faucets is evidently minor.

There is no guarantee that CRN will always induce the desired positive covariance and attendant variance reduction, and "backfiring" counterexamples can be constructed (see Wright and Ramsay, 1979) where CRN induces negative covariance and thus increases the variance in the comparison. However, it is generally believed that such examples are rare and basically pathological, so that CRN, especially with care taken to synchronize random-number use as appropriate to the model and scenarios, is usually effective, though just how effective is generally model-dependent.

As mentioned, CRN is but one of several variance-reduction techniques that rely, in one way or another, on using and re-using random numbers in a controlled manner to induce correlation. Other methods such as *antithetic variates* and *control variates* are discussed in, for example, Bratley et al. (1987, Chapter 2) or Law and Kelton (2000, Chapter 11).

5 Conclusions and suggestions

This is an area where it is relatively easy to do a good job, so it is hard to think of justifications for not doing this well. If it were difficult, say, to use a good underlying random-number generator or to ensure reasonably effective synchronization for variance reduction, then it would be understandable (though still incorrect, or at least inefficient) if people avoided doing so.

So simulation analysts should take a little care up front in their projects (and maybe ask the right questions of their software vendors). A little effort here will repay substantial dividends later in terms of accuracy and precision of the output, and thus in terms of the quality of conclusions and decisions as well.

And simulation-software vendors have a responsibility to ensure that they are doing at least the basics right. An opinion about what constitutes "the basics" is (approximately in decreasing order of importance):

1. If you are still using an old random-number generator of period on the order of 2^{31}, replace it immediately with one of the more recent generators with (*much*) longer period and (much) better statistical properties. If you are worried about your users' getting upset about suddenly getting

different results from their old models, then they (and maybe you) are in denial about the fundamental fact of randomness in simulation output.

2. Provide very widely spaced streams in the random-number generator, such that the user can specify the stream to be used. By the way, the old de facto standard of spacing seeds 100,000 random numbers apart is no longer even close to adequate. Use the literature on random-number generators (see Chapter 3) to choose the seeds or seeding method (and the generator itself).
3. Within the streams, provide very widely spaced substreams (that need not be user-addressable) to which each new replication of the simulation is advanced automatically, as discussed in Section 2. The spacing between consecutive substreams (and thus streams) should be (and *can* be) so large that many years of computing would be required to exhaust them. And the number of streams and substreams should be (and *can* be) so large that no simulation project could ever use them up.
4. As a default (that could be overridden by a savvy user with a *very* good reason), move to the next random-number stream every time the user specifies generation of a random object in the model. For instance, every place in the model where generation from a standard univariate probability distribution is called for, there will be a unique stream assigned to it. This would automatically implement synchronization by faucets as described in Section 2. Do not worry about having enough streams, because the underlying random-number generator and the stream/substream structure ensure that no human can build a model that will use them all up.
5. Include support for modeling (estimating) and generating from nonstationary Poisson processes. Almost 25 years of personal experience suggests that these are present in many real systems (even in student projects of limited scope and complexity) and flattening them out, even to the correct aggregate mean rate, does serious damage to such models' validity – imagine an urban-freeway simulation with twice-daily rush hours flattened to a mean that includes 3AM. Recent work on this includes Kuhl et al. (2006) and Leemis (2004).

While the wish list above contains only items that can be implemented right now, there is at least one other modeling capability that would seem to be worth considering, though maybe having it fully automated is not, at present, realistic. Current practice for specifying input probability distributions (or, more generally, input random objects) goes something like this: collect data from the field, fit a distribution to those data using software that comes with the modeling package or separate third-party software, translate the results into the correct syntax for your modeling package, and then type or paste it into your model wherever it belongs. While the syntax-translation part of this sequence is commonly automated, the rest is not – but maybe it could be. The result would be not only streamlining of this process, but also automatic updating as things change and the field data begin to look different. The modeling

software could establish dynamic links between the places in the model where these input-process distributions are needed and the fitting software, and indeed back to the files containing the raw field data themselves, thus making the whole fitting process transparent to the user (and rendering the fitting software invisible). The user would tell the simulation model where the data set is on some input feature, and it would automatically link to the fitting software, analyze the data, and give the results back to the simulation model. Clearly, there would have to be some kind of escape from this automation should problems arise during the fitting phase (e.g., none of the "standard" probability distributions provides an adequate fit to the data), though use of empirical distributions, as suggested by Bratley et al. (1987) could provide something of an automated escape in the case of simple univariate input distributions. This kind of dynamic integration is now common in office-suite software (e.g., updated spreadsheet data automatically update a chart in a word-processing document or presentation or web page), and it could be common in simulation software as well, but would require better integration of activities within simulation software than we now have.

In summary, there are a few nitty-gritty, low-level items almost down at the programming level that should be attended to in order to promote simulation-model building that is more robustly accurate, as well as possibly considerably more efficient. At present, the burden for most of this falls on the user (at least to check that the modeling software is doing things right), but much of this burden could be shifted to software designers to enhance their products' value, even if somewhat unseen to most users.

References

Banks, J., Carson II, J.S., Nelson, B.L., Nicol, D.M. (2005). *Discrete-Event System Simulation*, 4th edition. Prentice-Hall, Upper Saddle River, NJ.

Bratley, P., Fox, B.L., Schrage, L.E. (1987). *A Guide to Simulation*, 2nd edition. Springer-Verlag, New York.

Çinlar, E. (1975). *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, NJ.

Kelton, W.D. (2001). Some modest proposals for simulation software: Design and analysis of experiments. In: *Proceedings of the 34th Annual Simulation Symposium*. Advanced Simulation Technologies Conference Seattle, WA. IEEE Computer Society, New York, pp. 237–242, keynote address.

Kuhl, M., Sumant, S., Wilson, J. (2006). An automated multiresolution procedure for modeling complex arrival processes. *INFORMS Journal on Computing* 18, 3–18.

Law, A.M., Kelton, W.D. (2000). *Simulation Modeling and Analysis*, 3rd edition. McGraw-Hill, New York.

L'Ecuyer, P., Simard, R., Chen, E., Kelton, W. (2002). An object-oriented random-number package with many long streams and substreams. *Operations Research* 50, 1073–1075.

Leemis, L. (2004). Nonparametric estimation and variate generation from event count data. *IIE Transactions* 36, 1155–1160.

Wright, R., Ramsay, T. (1979). On the effectiveness of common random numbers. *Management Science* 25, 649–656.