

# Some Modest Proposals for Simulation Software: Design and Analysis of Experiments

W. David Kelton

*Department of Quantitative Analysis and Operations Management*

*University of Cincinnati*

*Cincinnati, Ohio 45221-0130, USA*

*<http://www.econqa.cba.uc.edu/~keltond/>*

*[david.kelton@uc.edu](mailto:david.kelton@uc.edu)*

## Abstract

*Simulation software has made great advances in recent years along the dimensions of modeling capabilities, animated graphics, and ease of use. There have also been real improvements in terms of both generality to model a wide variety of systems, and in specialization for quick and accurate modeling in specific application domains. And, of course, the dramatic improvement in computing/cost ratios has rendered truly commonplace what were just a few years ago impossibly time- and memory-consuming simulations. However, the statistical design-and-analysis capabilities of simulation software have not kept pace with the modeling, graphics, and ease-of-use advances; nor have they kept pace with ongoing research on the underlying methods. This paper discusses a wide variety of design-and-analysis capabilities that the author feels should be as available and as easy-to-use as are current modeling and graphics functions. The list includes only methods that are in existence today, so is not just a “wish list” of perhaps-impossible dreams. The paper concludes with speculation on why this imbalance exists, and suggests how it might be effectively addressed.*

## 1. Introduction

In the last decade or so there have been truly great advances in simulation software in terms of modeling capabilities, animated graphics, ease of use, and, to some extent, generality. It is now possible for a relative novice to learn how to build relatively elaborate simulation models in a relatively short time frame. Indeed, this is in part responsible for the great and growing popularity and use of simulation (along with the obvious leaps in computer performance/price ratios).

At the same time, though, implementation of design-and-analysis capabilities has lagged behind. This is not due to paucity of research on such methods, or a large inventory of important and unsolved problems, since basic methodological research in these fields has forged ahead unabated for at least 30 years, with many fundamental and highly applicable advances in evidence.

So now we have a wealth of extremely good (and good-looking) simulation models, built without huge investments in time, people, or software dollars, alongside an impoverishment of design-and-analysis tools and applications. The result is that we have a large inventory of wonderful models that are being tragically underused and underanalyzed, to the detriment of simulation’s potential to help and have impact.

So what to do? Here I’ll make some modest proposals to beef up the design-and-analysis capabilities of simulation software, and will throughout remain well within the realm of what is possible *now*, with our current knowledge of underlying methods, steering clear of pie-in-the-sky wish lists of being able to do things we don’t know how to do.

In Sections 2, 3, and 4 I’ll look at the input side of simulation analysis. In Section 5 is the groundwork for the output side, that of analyzing a single simulated system. Sections 6 through 10 give some possibilities for analyzing and understanding multiple simulated systems and the relationships between them. In Section 11 I’ll conclude with some speculation on concrete steps that can be taken. The references at the end of the paper mention several books and recent survey articles on these topics, which in turn contain many more references into the literature.

Some of what I’d like to see, as described in this paper, is indeed available in some software today. But a lot of it isn’t, and I think it should be and could be.

## 2. Input analysis

In developing a stochastic simulation, one must decide what input distributions and processes to specify to agree as closely as possible with reality. This involves various kinds of statistical analyses of observed real-world data, in order to specify realistic probabilistic distributions from which to sample to drive the simulation.

This is quite different from the kind of statistical analysis one finds in standard statistical packages, many of which are oriented to social-science survey or economic data. And so these packages are seldom adequate to give the simulation modeler what is needed.

We need to be able to “fit” standard distributions to observed data, as well as specify nonparametric empirical distributions when standard distributions fail (or even as a matter of course, whether standard distributions fail or not). Beyond this, we must be able to fit multivariate processes, either by estimating the full-blown joint distribution, or perhaps just estimating the marginal distributions and the correlation matrix; there are well-known examples of simulation models’ going seriously astray if such correlations are ignored. Another common need is to estimate some kind of dynamic process that might exhibit marked nonstationarity, for instance to model an arrival process of cars to a freeway interchange over the course of a day that contains two rush periods separated by long periods of relative inactivity.

Importantly, it should be easy or almost transparent to the user how to link the results of this input analysis to the simulation-modeling software. It is easy to do so in terms of syntax of text entries for fitted input distributions, but it would be better to establish dynamic links between the places in the model where these input-process distributions are needed and the fitting software, or perhaps even back to the data themselves, thus making the whole fitting process transparent to the user. I would tell the simulation model where my data set is on some input feature, and it would automatically link to the fitting software, analyze the data, and give the results to the simulation model. This kind of dynamic integration is now common in office-suite software, and it could be common in simulation software as well, but will require better integration than we now have.

## 3. Random-number generators

I like to describe random-number generators as the “engine room” of stochastic simulation – buried, out-of-sight, humming along silently, yet clearly critical to moving ahead. Somewhat alarmingly, we now have a severe mismatch between computer-hardware capabilities and the

characteristics of random-number generators 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 statistical tests. With the speed of computers several decades ago, generators were developed that were adequate to “fool” these tests up to the discriminatory power of the day.

One aspect of this is the *cycle length* of a generator, which is the number of random numbers before the generator repeats itself (as all algorithmic generators eventually will). In the 1960s and 1970s, generators with cycle length 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 (almost-universal) use today, when computer speeds have increased to the point that my low-end laptop can completely exhaust this cycle in just a few minutes! At that point the stream cycles and I get exactly the same “random” numbers in exactly the same order, with obvious deadly implications for the integrity of my simulations’ results. This is a dirty little scandal in simulation software, and is one that few users (or software developers) are aware of, or seem to care about.

We now have developed and coded algorithms for extremely long-period generators (with cycle lengths on the order of  $10^{57}$  or more), which display superb statistical behavior. They are “long” even under Moore’s law, for centuries to come. And their speed is comparable to the poor little old generators from decades ago. I cannot think of any excuse at all for not implementing these kinds of generators immediately.

Another aspect of random-number generators is the ability to specify separate “streams” of the generator, which are really just (long) subsegments of the entire cycle, and to make these readily available, perhaps via on-the-fly object-oriented instantiation. These streams can be further subdivided into substreams, and so on, for multi-dimensional indexing and assignment of separate and independent chunks of random numbers to separate activities in the simulation; the importance of this is really in variance reduction, discussed below in Section 7. Of course, being able to do this requires an extremely long-period underlying generator, but we now have those. And we also have easy and fast methods to create and reference such a multi-dimensional streams structure.

## 4. Variate and process generation

Putting input analysis and random-number generation together, we arrive at the point of needing to generate realizations of the input process to drive the simulation.

For some years we have had a variety of good methods to generate variates from the standard univariate probability distributions. Some of these, however, were developed at a time when conserving computer time was paramount, which still might be the case in some applications requiring extremely large generated samples. However, in many applications it can become more important to preserve the one-to-one mapping of random numbers onto random variates (to preserve synchronization for variance reduction), which can be done by using the inverse-distribution method of variate generation. This also maximizes the correlation between random numbers and variates, which improves variance reduction. Inverse-distribution methods are thus preferred, even if one has to resort to a numerical root-finding scheme to make them work in the case of some distributions.

Going along with specification of multivariate, correlated, and stochastic-process input-model specification discussed above in Section 2, we need broader coverage of the ability to generate such random structures in simulation, which can be critical to the validity of some simulations. At this point, there is almost no coverage of these abilities, without tedious user-written helper codes.

And, like input-distribution specification, these generation methods should be linkable all the way back to the observed real-world data, without the need for intermediate user intervention to supply data, fit distributions or processes, and then select them in the modeling software. I would like to say to the simulation software: “Here are my data, now you generate a random structure that behaves like these data, and you decide whether things like correlations and nonstationarity are needed to do so in order to represent reality faithfully.”

## 5. Statistical analysis of a single system

Once the model (including the random inputs) are set up, we’ll run the model to get some outputs. Of course, these outputs will be subject to sometimes-substantial statistical variability if the inputs are random. So it’s essential to assess the precision of the simulation’s output, and to take corrective action if the precision is unacceptable.

If there’s just a single system of interest (a rare but possible situation, e.g. in proving in specifications) a common and appropriate approach is to build confidence intervals on key output performance measures.

A central question is whether the time frame of the simulation is short-run (terminating) or long-run (steady-state); the software should demand that the user state which is of interest, as well as what the key output performance measures are, and then the software should take over the decision making about how much simulating is needed.

For terminating systems, the software should perform replications until the confidence intervals meet a user-specified precision requirement, stated in either absolute or relative terms. Such sequential-sampling procedures are by now well-understood and are relatively simple to explain and to code. This would produce confidence intervals that satisfy the user’s precision requirements, perhaps with a gentle warning about the Bonferroni inequality and the curse of multiple comparisons. There might also be a “bail-out” option for the user if the initial precision demands are so tight that the simulation would have to run until, say, the thermal death of the sun (the required simulation effort can be estimated in advance); in this the user could be given the opportunity to reconsider and relax the precision demands.

For steady-state simulations, the software should use the batch-means method in a single long run, perhaps in concert with some initial-data deletion in a warmup phase, and should itself decide on things like appropriate batch size and run length. Again, sequential sampling would be used, but this time with batches and batch sizes instead of replications, to satisfy the user’s precision requirements.

While it might be rare that only a single system is of interest, the methods for statistical analysis in such cases form the basis for the more complex (and realistic) settings discussed in the coming sections.

## 6. Comparison, selection, and ranking of several given systems

In many studies we’re faced with a relatively short list of possible systems and are asked to compare them or rank them or select the best one(s). There should be provision for doing everything discussed in this section in the context of either terminating or steady-state simulations (using, respectively, replications or batches).

If there are only two systems, the software should produce paired- $t$  confidence intervals on the differences between key performance measures in the two systems. The variance-reduction technique of common random numbers (see Section 7) should always be used (as it in fact always is today in simulation software, though unconsciously) but with proper synchronization (as it in fact almost never is today in simulation software without somewhat heroic intervention on the part of the user, which seldom hap-

pens). This synchronization should be automatic, and I discuss this further in Section 7.

If the number of systems is more than two, the software should have easy provision for calling for all comparisons against a standard, as well as all pairwise comparisons, producing just marginal confidence intervals but warning about Bonferroni multiple comparisons and the danger of overstating the overall confidence level.

Multiple ranking-and-selection procedures have been in development for over 50 years, but simulation now provides a near-perfect opportunity for their effective use. There should be easy entry in the simulation software to such methods, probably using indifference-zone formulations for means as well as multinomial probabilities, with user specification of the indifference zone and correct-selection-probability requirements. The output should declare the selected system(s), together with quantification of how much worse the inferior systems are in case the best system proves undesirable for other reasons.

## 7. Variance reduction

Since computer simulations use (repeatable) random-number generators rather than unrepeatable natural randomness, we can “swindle” the output noise by judicious re-use of previously-used random numbers, resulting in better precision for the same effort, or less effort to achieve a given precision.

Probably the best-known and most effective of these methods is common random numbers (CRN), meaning that we use the same random numbers to simulate all the different systems of interest, rather than using independent random numbers to simulate them independently. This is “comparing like with like,” or *blocking* in experimental-design parlance where the blocking factor is the sequence of random numbers being used.

A critical but usually-overlooked aspect of effective use of CRN is that not only are the same random numbers used to simulate across the different systems, but they must be used *for the same purposes* across the different systems. Thus, it is necessary to *synchronize* random-number use across the systems, and this can require care and even cunning in some situations.

Simulation software can go a long way toward making synchronization happen by devoting separate streams of random numbers (see Section 3) to separate sources of randomness in the model. This clearly requires an ample supply of long streams, which is yet another reason for moving to modern long-period stream-delimited generators, as argued in Section 3. It is also important to be able to re-synchronize across the different models at the beginning of new replications or batches, which is a good use for substreams within streams.

Current simulation software makes it difficult for the user *not* to use common random numbers, since the default is to use the same stream with the same seed for all runs, including those made with different parameter settings representing the different systems. However, it is currently quite difficult to synchronize properly. By ticking up a stream counter with each source of randomness put into the model, and ticking up a substream counter with each additional replication or batch, simulation software could promote near-perfect synchronization, with sometimes-dramatic improvements in the precision and sharpness of the comparisons across the different systems.

## 8. Sensitivity and gradient estimation

A natural question to ask of many simulations is “what happens if I change this parameter a little bit?” This is in essence a question of estimating a derivative or a gradient vector if several parameters are moved at once.

I would like to be able to tell the software that these three or four parameters are critical in this region, and have the runs give me back estimates of the partial derivatives with respect to these parameters (and, of course, precision statements, maybe as confidence intervals, on these derivative estimates). A currently-available and reliable method to do this is known as *finite differences*.

## 9. Experimental design

Related to the question of sensitivity estimation is to state a somewhat cruder result in terms of the “effect” of input factors, and whether (and to what extent) these factors might interact with each other. The fundamental principles of experimental design have been in development since the 1920s, and while being born out of physical (often agricultural) experiments, they are eminently applicable to simulation experiments as well.

There should be provision for automatic running of full factorial experiments, given an identification of the factors and a coding chart, producing main effects and interactions across replicated designs for confidence intervals on all effects, and presented in graphical formats. Fractional factorial designs should be just as accessible, asking the user for factors, their coding chart, and the desired precision on the effects estimates, reporting back on fractionating possibilities, an estimate of run time, and asking the user for advice on how to proceed; the results should again be graphically displayed as in full-factorial designs. There should be factor-screening designs available to pare down an initial large number of factors, as well as integration of regression methods for building response-surface

metamodels of the key outputs as functions of the important inputs.

## 10. Optimum seeking

Of current interest (and development) are various “controller” packages to run and rerun simulation models in pursuit of a combination of input factors that optimizes some key output performance measure. The controller package makes its own decisions about how to move which input parameters in search of the optimum, based on a wide variety of methods and ideas borrowed from the nonlinear-programming world. This capability is of obvious value, which accounts for the relatively high level of current interest.

Probably the best hope for widely robust, reliable, and stable methods lies in the realm of heuristic-search methods. These methods avoid simplifying yet highly suspect assumptions (e.g., linearity, convexity or concavity) and are thus widely applicable. However, they cannot *guarantee* an optimum, and it is important to realize this (regardless of what might be claimed) in understanding what we are getting.

## 11. Conclusions

The above represents quite a laundry list of desires, but *it is all doable at this time*. I have not mentioned anything at all for which there are still open research questions or uncertainties or gaps in our methodological knowledge.

So why haven't these things been done already? My own speculation is that due to commercial and competitive pressures, software developers have been forced into short-term reactions to demands from perhaps-short-sighted customers that development effort be devoted instead to more “visible” enhancements like graphics (3-d, shadows, reflections, just like the movies) and extreme ease of use. These customers are themselves probably only reacting to the short-term thinking of their management, and on and on. So I don't “blame” the software developers or their customers for this state of affairs, but only seek to point out that we are in a situation of considerable imbalance between modeling and design/analysis capabilities of simulation software.

I feel it is up to those of us in the research community to address this imbalance, by educating about the real needs for the quite-feasible developments I've outlined here. So maybe this paper can be a start in that direction.

## References

Please note that the *Winter Simulation Conference Proceedings* for 1997–2000 are freely available at <http://www.informs-cs.org/wscpapers.html>.

- [1] C. Alexopoulos and A.F. Seila, “Output Analysis for Simulations”, *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 101–108.
- [2] J. Banks, J.S. Carson, B.L. Nelson, and D.M. Nicol, *Discrete-Event System Simulation*, 3d ed., Prentice-Hall, Upper Saddle River, NJ, 2000.
- [3] P. Bratley, B.L. Fox, and L.E. Schrage, *A Guide to Simulation*, 2d ed., Springer-Verlag, New York, 1987.
- [4] L. Devroye, *Non-Uniform Variate Generation*, Springer-Verlag, New York, 1986.
- [5] G.S. Fishman, *Principles of Discrete Event Simulation*, John Wiley, New York, 1978.
- [6] D. Goldsman and B.L. Nelson, “Statistical Screening, Selection, and Multiple Comparison Procedures in Computer Simulation”, *1998 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Washington, DC, 1998, pp. 159–166.
- [7] D. Goldsman, B.L. Nelson, T. Opicka, and A.A.B. Pritsker, “A Ranking and Selection Project: Experiences from a University-Industry Collaboration”, *1999 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Phoenix, AZ, 1999, pp. 83–92.
- [8] D. Goldsman and G. Tokol, “Output Analysis Procedures for Computer Simulations,” *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 39–45.
- [9] W.D. Kelton, “Perspectives on Simulation Research and Practice,” *ORSA Journal on Computing* 6, 1994, pp. 318–328.
- [10] W.D. Kelton, “Statistical Analysis of Simulation Output”, *1997 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Atlanta, GA, 1997, pp. 23–30.
- [11] W.D. Kelton, “Experimental Design for Simulation”, *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 32–38.
- [12] W.D. Kelton, R.P. Sadowski, and D.A. Sadowski, *Simulation with Arena*, McGraw-Hill, New York, 1998.
- [13] A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis*, 3d ed., McGraw-Hill, New York, 2000.
- [14] A.M. Law and M.G. McComas, “Simulation-Based Optimization”, *2000 Winter Simulation Conference Pro-*

*ceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 46–49.

[15] P. L'Ecuyer, "Uniform Random Number Generators", *1998 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Washington, DC, 1998, pp. 97–104.

[16] L. Leemis, "Input Modeling", *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 17–25.

[17] B.L. Nelson and M. Yamnitsky, "Input Modeling Tools for Complex Problems", *1998 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Washington, DC, 1998, pp. 105–112.

[18] S.M. Sanchez, "ABC's of Output Analysis", *1999 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Phoenix, AZ, 1999, pp. 24–32.

[19] S.M. Sanchez, "Robust Design: Seeking the Best of All Possible Worlds", *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 69–76.

[20] B. Schmeiser, "When Standard Input Models Fail", *1999 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Phoenix, AZ, 1999, pp. 110–115.

[21] J.R. Swisher, P.D. Hyden, S.H. Jacobson, and L.W. Schruben, "A Survey of Simulation Optimization Techniques and Procedures", *2000 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Orlando, FL, 2000, pp. 119–128.

[22] J.R. Wilson, "Modeling Dependencies in Stochastic Simulation Inputs", *1997 Winter Simulation Conference Proceedings*, Winter Simulation Conference Board of Directors, Atlanta, GA, 1997, pp. 47–52.