Easy Simulation on Spreadsheets without Add-ins

John T. Simon, College of Commerce and Economics, Sultan Qaboos University, Sultanate of Oman

Simulation is one of the popular tools used in production and operations management. Although large simulation projects need code written in a programming language (often a simulation language such as Simscript or SIMAN), it is feasible now to use spreadsheets for simulation of some of the smaller systems.

Most of the simulation done on spreadsheets make use of add-in software such as @Risk (from Palisade Corporation) or Crystal Ball (from Decioneering, Inc.). Inventory models, waiting lines, financial models with risk, PERT networks, etc., are all problems that can be simulated using spreadsheets with add-ins (see Winston, 1996). This is particularly appealing to business students who are reasonably familiar with spreadsheets.

While teaching operations management, management science, and simulation courses, I introduce my students to simulation using spreadsheets (specifically, using Microsoft Excel version 5.0 for Windows). They seem to like it since it builds on something they already know (instead of having to learn a new language), and spreadsheet modeling allows them to understand how simulation works.

Fortunately, Microsoft Excel version 5 and later versions have a number of functions and capabilities built in so that many simulations can be carried out without any add-ins. This is helpful because almost all students and professionals have access to spreadsheets (such as Microsoft Excel), and one need not incur the extra expense (and the trouble of network installation) of additional software. What follows is a brief description of how I teach simulation on spreadsheets without the use of add-ins. The add-ins do add more functionality and ease of use; so this is not a case against them. Rather, this gives you a choice to do simulations without an add-in, obviously with some loss of functionality and ease of use.

Features Provided by Add-ins

From my perspective, the add-ins primarily allow us to:

1. Change any cell of the spreadsheet into a random variable.
2. Store the values of chosen cells (the "result") obtained from each of the repeated calculations of the spreadsheet.
3. Present the results in a useful (graphical) format.

For example, consider the following simple model:

profit = revenue – cost.

This can be implemented on the spreadsheet by using specific values for revenue and cost, say 100 and 80, respectively, by typing the entries in the corresponding cells as shown in Table 1.

The cell A3 will yield the profit, 20 in this case.

Now let us introduce randomness in the above model, by assuming that the revenue and cost are independent random variables. Suppose that we would like to find the distribution of profit, given that revenue is normally distributed with mean 100, standard deviation 10, and cost is nor-

cell	entry
A1	100
A2	80
A3	= A1 – A2

Table 1

John T. Simon
is an assistant professor in the Department of Production and Operations Management at the Sultan Qaboos University in the Sultanate of Oman. He holds a Ph.D. in industrial engineering and management sciences from Northwestern University, Evanston, Illinois. His research has appeared in International Journal of Production Research, IIE Transactions, *and* Queueing Systems: Theory and Applications. *Email: jsimon@squ.edu.om.*

mally distributed with mean 80, standard deviation 5. Using @Risk, we could (following the three features outlined above):

1a. Make the cell A1 to have a value sampled from a normal population with mean 100 and standard deviation 10.

1b. Make the cell A2 to have a value sampled from a normal population with mean 80 and standard deviation 5, and

1c. (As before) make the cell A3 to have the formula "=A1 – A2".

2. Repeatedly recalculate the spreadsheet, say 100 times, and store the values obtained from the 'result' cell (A3) during each recalculation.

3. Obtain a graphical representation of the distribution of the values stored.

This is the basic procedure for most spreadsheet simulations, although the models would be more sophisticated.

How to Survive without Add-ins

How can we perform the three features mentioned above without the use of an add-in software? With some familiarity of charts or histograms in Microsoft Excel, the feature (3) mentioned above is quite straightforward **if** the results of the repeated simulations are available. Unfortunately there is no mechanism to store the values of a cell during repeated recalculations of the spreadsheet: they are rewritten each time the spreadsheet is recalculated (say by pressing the function key F9 in Excel), and the only result available is the result of the last recalculation. Thus feature (2) is a problem. I will show later how this can be resolved.

Feature (1) is solved in part by Microsoft Excel's uniform random number generator function "RAND( )" and also by the availability of statistical functions in Excel (such as beta, binomial, chi-squared, exponential, F, gamma, lognormal, normal, Poisson, t). The function "RAND( )" returns a uniformly distributed random variable between 0 and 1. By multiplying it by a constant or adding a constant, and using the "integer" function "INT( )", we can obtain discrete and continuous uniform random variables in any range. To get other random variables, we could use the standard 'inverse cumulative distribution function' approach. For example, to get a normal random variable with mean 100 and standard deviation 10, we may use the Microsoft Excel function "NORMINV(RAND( ),100,10)." The procedure is equally straight forward for other continuous random variables mentioned above since their 'inverse cumulative distribution functions' are available in Excel (for exponential random variable, we could use "-LN(RAND( )) / MU" where mu is the rate). For discrete random variables, a 'table look up' is necessary.

To implement the above model, type the entries in the corresponding cells as shown in Table 2.

cell	entry
A1	= NORMINV(RAND( ),100,10)
A2	= NORMINV(RAND( ),80,5)
A3	= A1 – A2

Table 2

Now, each recalculation (by repeatedly pressing the function key F9) of the spreadsheet gives us one simulated value of profit.

Thus the main remaining difficulty is in storing the intermediate values of the "result" (the value in cell A3) during recalculations of the spreadsheet. An easy solution is to do all calculations on one sheet—for example, in the above problem, copy the range A1:A3 to several columns to the right, say B through J, to get ten values of profit. Now the values in the range A3:J3 are the different values of the profit that we are seeking.

Such an approach will not suit a larger model or a large number of replications. Hence, I prepared the following macro named "Recorder" for my students. (To create this macro, use 'Insert | Macro > Module' menu command in Microsoft Excel version 5, or 'Tools | Macro > Visual Basic Editor' and then 'Insert | Module' in version 7, and type the following in the new module. After creating the macro, in version 7, get back to Excel using 'File | Close and Return to Microsoft Excel'.)

```
Sub Recorder( )
    For iteration = 1 To 100
    Calculate
    Selection.Copy
    Sheets("Sheet2").Select
    ActiveCell.Offset(1, 0).
    Range("A1").Select
    Selection.PasteSpecial Paste :=
    xlValues
    Sheets("Sheet1").Select
    ActiveCell.Select
    Application.CutCopyMode = False
    Next iteration
End Sub
```

{Note: In the sixth line of the subprogram, the last word is x'el'Values, not x'one' Values.}

How does it work? I am assuming that the model is in "Sheet1", and "Sheet2" is available for recording the "results." Open Sheet2 and click on any cell below which you want the results to be stored. Then go to Sheet1, and click on the cell that contains the "result" (A3 in our example). Now run the macro (in version 5, open the menu 'Tools | Macro', select and run "Recorder"; in version 7, open the menu 'Tools | Macro > Macros', select and run "Recorder"). The workbook will be recalculated 100 times, and for each recalculation, the "result" in Sheet1 will be recorded in a column in Sheet2. Once the simulation stops, we have all the values stored in Sheet2 to be used in any analysis we like; for example, we could draw its histogram.

By modifying the above macro, we can change the number of iterations to any value; the "result" could be a range instead of a single cell; different sheets (i.e., other than Sheet1 and Sheet2) can be used for model and results, etc. If we don't need the actual values stored but only need the average, the macro can be simplified to keeping track of the sum and count instead of recording all values.

With the above approach I am able to guide my students through a number of simulation models (such as the ones mentioned in Winston, 1996). An added attraction is that there is no 'black box'; everything is quite plain to understand. Furthermore, all the data generated during the simulation are available for any further analysis, which is not usually the case with add-ins. But on slower machines, the execution time may be a factor in favor of add-ins, since the macros are not compiled—while running the macro, one can actually see the computer screen change between the model sheet and the data record sheet for each iteration of the simulation. And finally, there is the distraction that some of the students get interested in writing macros—but then they are on their own! ■

Acknowledgments

I would like to thank Professors Jay Ghosh and Thomas Jefferson for their helpful comments.

Reference

Winston, Wayne L. (1996). *Simulation Modeling Using @Risk*, Duxbury Press, Belmont, California.

Dr. G. Keong Leong
The Ohio State University
College of Business
Department of Management Sciences
1775 College Road
Columbus, OH 43210-1399
email: leong.1@osu.edu.