Spring 2005

An Excel Application for Valuing European Options with Monte Carlo Analysis

Tom Arnold
University of Richmond, tarnold@richmond.edu

Stephen C. Henry

Recommended Citation

Arnold, Tom, and Stephen C. Henry. "An Excel Application for Valuing European Options with Monte Carlo Analysis." *Journal of Financial Education* 31 (Spring 2005): 86-97.

An Excel Application for Valuing European Options with Monte Carlo Analysis

Tom Arnold and Stephen C. Henry
University of Richmond and Sam Houston State University

By developing the basic intuition of how Monte Carlo simulation works within an Excel spreadsheet framework, this paper allows the undergraduate student to use Monte Carlo simulation techniques to price European style options without additional sophisticated software. Further, the skills and intuition developed provide the basis for much more complex simulation techniques.

INTRODUCTION

Monte Carlo simulation allows an analyst to generate many scenarios for a given security's future price. The frequency with which a particular future price occurs within a set of simulations becomes a measure of the likelihood that that future price will actually occur. Essentially, it is equivalent to rolling a pair of dice many times to determine how frequently certain values occur rather than using a predetermined probability density function to generate the frequencies of these same values.

With regard to security prices, the most common application of Monte Carlo analysis is for the pricing of options. Although closed form solutions such as the Black-Scholes model (1973) exist, these models cannot match the intuition and pedagogic value of Monte Carlo analysis. In fact, when closed form solutions are not available or need to be tested, Monte Carlo analysis often becomes the only pricing mechanism available for the task.

In this paper, we provide the basic structure of Monte Carlo analysis and then apply it to the pricing of European-style options (i.e. the option can only be exercised at maturity). Although sophisticated software packages for performing Monte Carlo simulation exist, Excel and VBA are suitable for the task, and are generally available to students.

In section one, the basic principles of stochastic processes and risk-neutral pricing are introduced without relying on stochastic calculus or continuously adjusted hedges. The second section provides the necessary Excel and VBA commands to produce a Monte Carlo analysis for a European-style option. Section three concludes the paper.

BASIC PRINCIPLES OF STOCHASTIC PROCESSES

Stochastic processes are probability distribution functions that contain a time component. More specifically, the mean and variance of the distribution change depending upon the length of the time-step under consideration. If one makes a prediction of a particular stock price one minute into the future, the risk or uncertainty relative to the accuracy of the prediction is probably fairly low. The predicted stock price is the mean of all of the possible future prices considered and the risk assessment of the accuracy of the mean (or predicted price) is the variance of the predicted price. By changing the duration of the prediction time-step from one minute to one year, the mean and the variance of the predicted future stock price change dramatically. In other words the probability distribution of the predicted future price is time dependent relative to its mean and variance. Consequently, a stochastic process is appropriate for assessing future security price movement.

The most basic stochastic process is a Wiener process. Observations from a Wiener process are normally distributed with a mean of zero and with a variance equal to an infinitesimally small time-step, "*dt*." The notation for a Wiener process is "*dZ*" and the notation for the associated normal probability distribution is "*N(*mean, variance*)*" or in this case, "$N(0, dt)$." The variance of the Wiener process can be scaled by multiplying dZ by a value "σ" making σdZ distributed $N(0, \sigma^2 dt)$. Further, the mean of a Wiener process can be scaled by simply adding a value, "μ," to dZ making $\mu + dZ$ distributed $N(\mu, dt)$. By combining these two properties and adjusting the μ parameter to include a time component (i.e. multiply μ by dt), Arithmetic Brownian Motion (ABM) emerges as: $\mu dt + \sigma dZ$ and is distributed $N(\mu dt, \sigma^2 dt)$.

Although one may be tempted to use ABM as the appropriate process to describe the future movement of a stock price, it is inappropriate despite having a mean and variance that is dependent on time due to the *dt* parameter. To illustrate the point, consider a $0.25 increase in price for a stock that had a price of $1.00 ten seconds in the past and compare it to a $0.25 price increase for a stock that had a price of $80.00 ten seconds in the past. The returns on the two stocks for an equivalent movement in price are very different. Because an ABM process acts independently of where it starts or where it is at any given moment in time, it is very much like the example above in which a $0.25 change in price is equally valid for an $80.00 stock or a $1.00 stock. If one can adjust the ABM process to make the price movement scaled relative to the most recent security price, then a potentially appropriate model for stock price movement emerges.

Let "S_t" be the stock price at a given moment in time and let "dS" symbolize the change in price looking forward into the future one time-step of length dt. By adjusting μdt and σ to include S_t, Geometric Brownian Motion (GBM) is created: $dS = \mu S_t dt + \sigma S_t dZ$ and is distributed $N(\mu S_t dt, [\sigma S_t]^2 dt)$. Notice, by simply multiplying μdt and σ by S_t, the new stochastic process has price movements that are scaled by the current stock price. μ is viewed as the expected return for the stock and σ is viewed as the volatility (or standard deviation) of the expected stock return.

The only statistical principles that have been applied thus far are: 1) a probability distribution with a mean of α and a variance of β^2 multiplied by a constant, "λ," creates a new probability distribution with a mean of $(\lambda\alpha)$ and a variance of $(\lambda\beta)^2$ and 2) when a constant λ is added to a probability distribution with a mean of α and a variance of β^2, it creates a new probability distribution with a mean of $(\lambda+\alpha)$ and a variance of β^2. The first principle was applied to scale the variance of a Wiener process and the second principle was applied to scale the mean of the Wiener process. It is critical to not make the creation of ABM and GBM from a Wiener process any more complicated than the above illustration, because the same application of these two statistical principles allows one to produce a basic Monte Carlo analysis in Excel.

One additional statistical principle is necessary to enable option pricing within a Monte Carlo analysis: the ability to change probability measure within a probability distribution. When a stock is priced, the price is the discounted value of all of the future cash flows associated with the stock. Suppose a given stock has an equal probability of being \$25.00 or \$26.50 one year into the future. Thus the expected future price is \$25.75 (equals 0.50×\$25.00 + 0.50×\$26.50). Given a current price of \$23.41, an expected future annual return of 10% is calculated (equals [\$25.75 - \$23.41] ÷ \$23.41). Suppose, for a reason not yet explained, one wants the future annual return to be 9% instead of 10%, the probabilities of the two future prices can be changed to allow a 9% return. By making the probability of the \$25.00 future price 65.3% instead of 50% and the associated probability for the \$26.50 future price 34.7% instead of 50%, the 9% expected annual return is achieved.

This is what is meant by changing the probability measure. Instead of adjusting the future outcomes of \$25.00 and \$26.50 to accommodate the 9% expected annual return (a standard transformation technique in statistics), the probabilities of the future prices are adjusted instead. GBM and ABM allow for a change in probability measure to be accomplished very easily. For more details, the interested reader is referred to Neftci (2000), Arnold and Henry (2003), Johnstone (2002), or Arnold and Crack (2003).

When pricing options, one knows the current price of the underlying security and can model the future prices (using a GBM process) of that underlying security associated with the expiration of the option. For ease of exposition, assume the option can only be exercised at expiration (i.e. a European-style option). To find the option price, one values the option based on its payoffs relative to the possible underlying security prices at the time of expiration. An expectation of the option payoffs is calculated and discounted to find the option price. Although one may use the underlying security's expected return to determine the possible future prices of the underlying security (and generate associated probabilities for these same future prices), the discount rate for the option is different than the expected return for the underlying security. The option is much riskier and any transformation from the underlying security's rate of return is subject to the accuracy of having selected the correct rate of return for the underlying security. Consequently, the task of determining the option's discount rate is not simple and fortunately not necessary.

By changing the probability measure of the stochastic process (i.e. a GBM process) modeling the underlying security's future price, the discount rate for the underlying security can be made equivalent to the discount rate for the associated option. By adjusting the underlying security return to be the risk-free rate, the new probability distribution of the stochastic process for the underlying security is risk-neutral because the underlying security's return now has a zero risk premium. Given this risk-neutral future price distribution for the underlying security price, the associated probabilities for the option payoffs are also risk-neutral, meaning, the expected future option payoff can be discounted at the risk-free rate as well. Because risk-free rates are easy to obtain, the use of risk-neutral pricing for options is very pragmatic in addition to being a simplifying probability transformation.

Note that the ability to change the probability measure is a property of GBM and not reliant on the ability to create a risk-neutral hedge nor on the assumption that participants are risk-neutral. These latter conditions are sufficient for risk-neutral pricing but not necessary. Although a formal proof of the ability to change probability measure is not given here (it is an application of Girsinov's Theorem), the reader is referred again to Neftci (2000).

MONTE CARLO ANALYSIS IN EXCEL

Monte Carlo analysis uses a stochastic process to model future security prices. By simulating prices using a given stochastic process, a probability distribution of the future security price is generated by determining the frequency of particular future prices over many simulation trials. The frequency with which a certain price appears throughout the simulation trials becomes the probability of that particular price occurring in the future. The probability equals the number of simulation trials in which the price appears divided by the total number of simulation trials.

For example, ten simulations of a future stock price produce: \$15.00, \$16.50, \$17.00, \$14.75, \$15.00, \$15.00, \$14.75, \$16.50. \$15.00, and \$16.50. \$15.00 appears four times out of the ten trials yielding a probability of 40% (i.e. 4 occurrences ÷ 10 trials). Correspondingly, the probabilities of \$14.75, \$16.50, and \$17.00 are 20%, 30%, and 10%. The expected future price of the security is calculated either by taking the average of all of the simulated prices or by taking the mean based on the final prices with associated probabilities. Generally, the former calculation of the expected future price is preferred over the latter calculation.

To apply a stochastic process to a Monte Carlo analysis in Excel, a random number generator from a normal distribution is necessary. The Excel command =RAND( ) produces a uniformly distributed random number between zero and one. The Excel command =NORMINV(*a, b, c*) produces a value from a normal distribution that has the cumulative probability associated with the value a ($0 \leq a \leq 1$), given a normal distribution with mean b and variance c^2. By combining the two commands, =NORMINV(RAND( ), *b, c*), a random number from a normal distribution with mean b and a variance of c^2 is generated.[1]

Similarly, a random draw from a Wiener process can be generated by the formula

=NORMINV(RAND( ), 0, SQRT(*dt*)), where SQRT( ) indicates the square root of the value *dt*. The Wiener process can then be adjusted to produce random draws from ABM and GBM processes by adjusting the mean and variance. Recall that ABM is normally distributed $N(\mu dt, \sigma^2 dt)$ from the specification $\mu dt + \sigma dZ$. The associated random number generator in Excel is =NORMINV(RAND(), μ**dt*, σ*SQRT(*dt*)). GBM is normally distributed $N(\mu S_t dt, [\sigma S_t]^2 dt)$ from the specification $dS = \mu S_t dt + \sigma S_t dZ$, where S_t is the most recent security price. The associated random number generator in Excel is =NORMINV(RAND( ), μ*S_t**dt*, σ* S_t*SQRT(*dt*)).

To generate a simulation, one must decide at what point in the future a security price should be evaluated and how many time intervals there should be prior to reaching that final point of time in the future. In fact, what is being asked is: just how small of a time increment does *dt* need to be. In the equations, *dt* is infinitesimally small, but in reality the length of *dt* must be specified by the modeler. Suppose it is decided that the appropriate point in the future (say, the option's expiration date) is three months from today and that there will be five time intervals within the three month period. Consequently, three months is 0.25 years, which is further divided by five to make each time interval 0.05 years. Thus, *dt* is 0.05 years in length. For each simulation of the three month future price, five consecutive draws from normal distributions with a dt specification of 0.05 years are necessary.

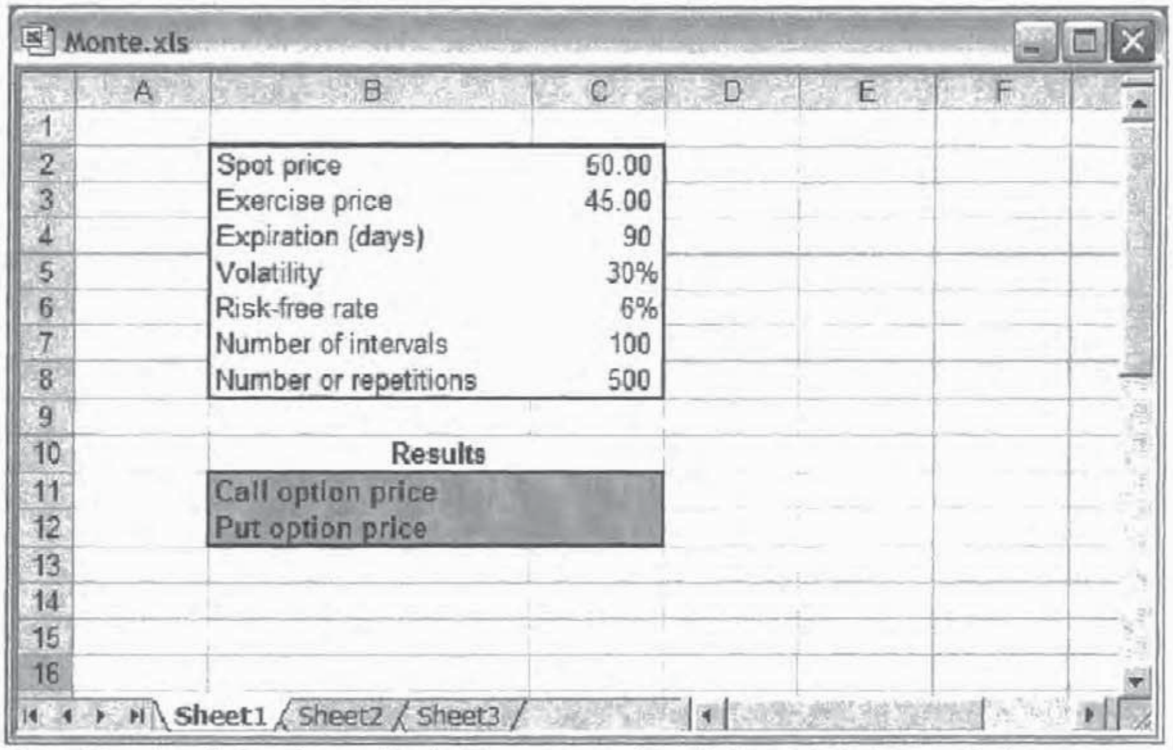
The question often arises: why not just make *dt* 0.25 years and not be concerned about the intervals in between? There are two reasons. First, the simulation increases in accuracy as *dt* becomes smaller, making *dt* resemble more closely the infinitesimally small value that it represents in theory. The second reason applies to GBM processes. Because the most recent security price, S_t, is part of the mean and variance specification, S_t should be updated as often as possible (i.e. at every interval of *dt*).

To price European-style options, the appropriate simulation length is the time until expiration of the option. The modeler determines the number of time intervals within a simulation and the number of repetitions of the simulation. Again, increasing the number of time intervals within a simulation makes *dt* smaller and increases accuracy. Increasing the number repetitions of the simulation also improves the accuracy of the model. The only other parameters to consider are: the current price of the underlying security, the annual risk-free rate (this is used instead of the expected return on the underlying security to generate risk neutral prices), the annual volatility of the underlying security's return (volatility is the standard deviation, i.e. the square root of the variance, and is the traditional input for Monte Carlo simulation), and the option contract specifications. The inputs are displayed in Figure 1.

The call option and put option prices in cells C12 and C13 respectively in Figure 1 still must be calculated using a Monte Carlo simulation. Cells C2 through C8 provide the necessary inputs for the analysis and can be changed to repeat the analysis under different scenarios.

Obtaining a meaningful expected ending price requires a large number of observations;

Figure 1. Spreadsheet Layout for Monte Carlo Analysis



	A	B	C
1			
2		Spot price	50.00
3		Exercise price	45.00
4		Expiration (days)	90
5		Volatility	30%
6		Risk-free rate	6%
7		Number of intervals	100
8		Number or repetitions	500
9			
10		Results	
11		Call option price	
12		Put option price	

Note: For simplicity, the option is European-style and the underlying stock does not pay dividends.

thus, some sort of automation is necessary to conduct useful simulations. Excel provides us with a tool that is ideally suited to the task: the Visual Basic for Applications (VBA) programming language.

Visual Basic for Applications is an adaptation of Microsoft's general-purpose programming language Visual Basic. Originally released in 1993, VBA was chosen by Microsoft to replace the Excel macro facility, and later to provide a consistent interface for programming and automation across the Microsoft Office suite of applications. Like the macro facility it replaced, VBA allows users to record and play back sequences of operations to automate tedious or repetitive tasks. Unlike the macro facility, though, VBA includes many elements of a general-purpose programming language (such as loops, branching, and conditional execution) that enable users to develop sophisticated applications for data retrieval and worksheet manipulation.

The repetitive task of generating Monte Carlo observations certainly requires some sort of automation. The process is conceptually simple: our program needs to (1) generate a new set of random disturbances (price changes) (2) compute the ending price (by adding the random price changes to the starting price), (3) calculate the payoff to the option holder (and store it for further analysis), and (4) repeat the process a large number of times. Although the procedure is straightforward, it requires more programming sophistication than a keystroke-macro facility can provide. VBA is an ideal tool for this exercise.

We begin the process of creating a VBA program by opening the Visual Basic editor, either through the [Tools]->[Macro] menu, or by pressing <Alt>-<F11>. This is the window in which VBA code can be developed and executed. At the left of this window is the Project Explorer, showing the structure of all workbooks and other collections of objects currently open in Excel. If the Project Explorer window is missing, open it using the [View] menu. One should be aware that format and default settings will vary between computers, however, the information contained in the different VBA windows will be equivalent. Consequently, the figures containing screen images in this article may not match the reader's computer screen exactly.

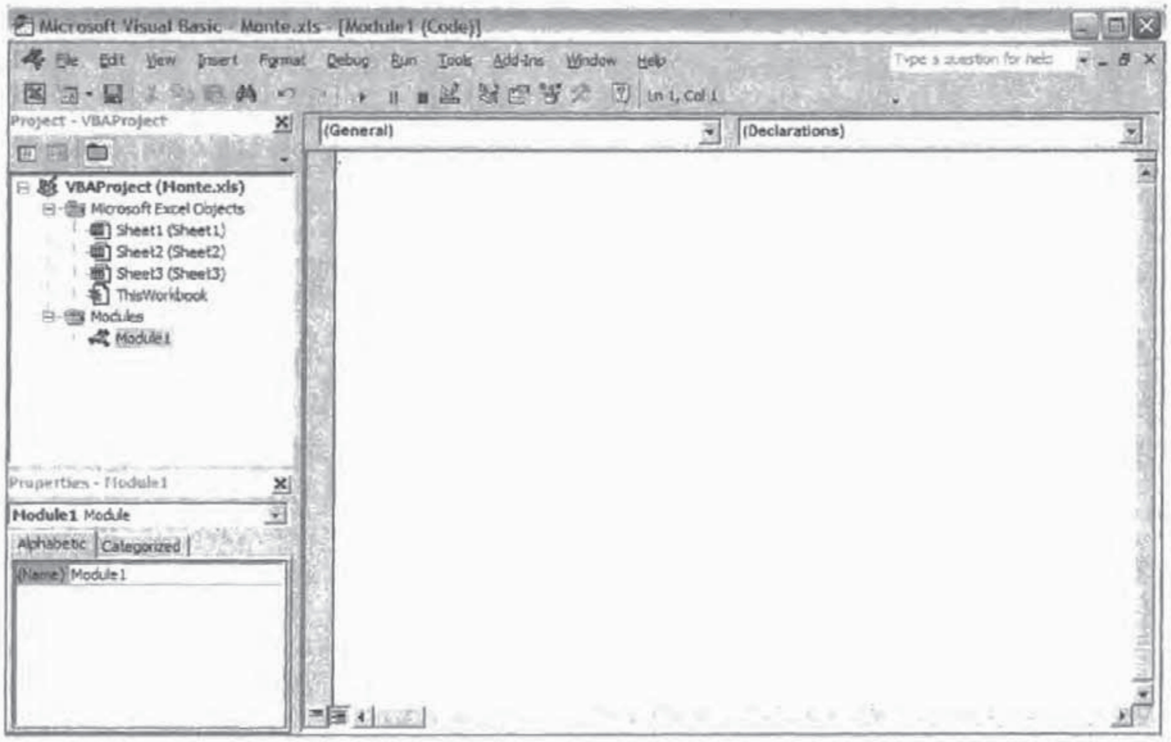
While a detailed discussion of Excel's object hierarchy is beyond the scope of this paper[2], it is important to note that an Excel workbook is essentially a collection of objects of various types: most commonly, worksheets, charts, and code modules. In order to execute properly, VBA programs must be contained in *standard* code modules. In particular, when a user-defined function is invoked from the worksheet, Excel searches for the appropriate code only in standard code modules. Functions stored inside other types of modules will not be found.

By default, a newly created workbook contains *sheet* code modules associated with the three initial worksheets ("Sheet1" – "Sheet3"). In fact, when the Project Explorer window first appears, "Sheet 1" is generally highlighted. While VBA code can be stored here, sheet code modules are not suitable for storing user-defined functions, because Excel will be unable to locate them[3]. The first step, then, is to create a *module* object (a standard code module) to hold our program. This is accomplished by selecting the name of the current workbook in the Project Explorer window ("monte.xls" in the example) and activating the menu sequence: [Insert] -> [Module]. Additional modules can be inserted in the same manner. The result should be, as in Figure 2, a new, blank standard code module "Module1" inside the current workbook.

With a blank module in place, we can begin developing the code to automate our simulation. The next step is to declare the name of our program and indicate what parameters it will receive. The statement "Function SimCall(S, X, T, sigma, rf, intervals, reps)" tells the VBA interpreter that we are creating a user-defined function called "SimCall" that requires seven parameter inputs. When the line is typed into the VBA editor, a corresponding "End Function" entry is generated automatically, defining the beginning and end of our program.

The VBA language allows programs to be declared either as *functions* or as *subroutines*. Setting aside issues of "good programming style" for the moment, the primary difference between the two types of programming is in the manner in which they are invoked. VBA programs written as subroutines must be executed from the [Tools] -> [Macro] menu, or from within the VBA editor itself. Programs that perform a sequence of operations without requiring any input are well suited to be written as subroutines. For example, recorded macros are declared as VBA subroutines.

Figure 2. VBA Editor

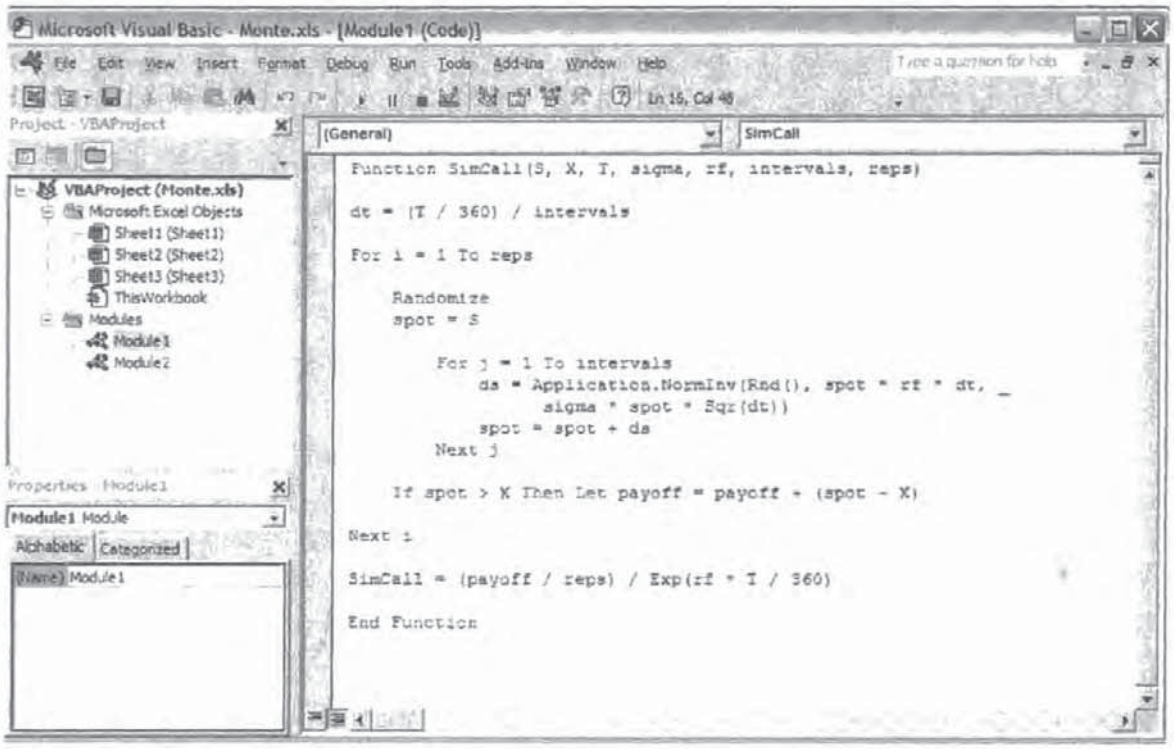


Alternatively, functions are programs that accept input in the form of arguments, perform some computations, and return the results. In Excel, VBA functions can be invoked directly from within the worksheet, just like the built-in functions (e.g. =RAND()). That is, once our program is complete, we will be able to run the simulation simply by typing the formula =SimCall(…) into a cell. Functions are commonly used to extend the capabilities of Excel by performing calculations not originally implemented by its designers.

Example code for conducting a Monte Carlo simulation is shown in Figure 3. The seven arguments to the SimCall( ) function are variable names chosen to represent the usual five parameters defining the option value, as well as the desired number of intervals and iterations for the simulation. By listing these variables as inputs for the function, they can be assumed to exist inside the program, and can be referred to by name when performing calculations. A similar function for a put option will be created in a second module (simply perform the [Insert] -> [Module] sequence a second time).

The program for the SimCall( ) function begins by calculating the value of *dt*, the length of each interval. Then a For/Next loop is established to repeat the simulation the number of times specified by the argument *reps.* The beginning stock price *S* is copied to the temporary variable *spot*, and a second loop simulates the stock price process by

Figure 3. Function Code for Monte Carlo Analysis



```
Function SimCall(S, X, T, sigma, rf, intervals, reps)

dt = (T / 360) / intervals

For i = 1 To reps

    Randomize
    spot = S

        For j = 1 To intervals
            ds = Application.NormInv(Rnd(), spot * rf * dt, _
                  sigma * spot * Sqr(dt))
            spot = spot + ds
        Next j

    If spot > X Then Let payoff = payoff + (spot - X)

Next i

SimCall = (payoff / reps) / Exp(rf * T / 360)

End Function
```

adding random price changes (*ds*) to the starting price. At the end of each simulation, the resulting call option payoff is calculated and averaged. The resulting expected payoff is assigned to a variable having the same name as the function itself, which is passed back to the worksheet cell when the program ends.

With the basic simulation framework in place, pricing different types of options becomes a matter of adjusting the payoff calculation. For example, one can implement a =SimPut( ) function for pricing a European-style put option in the second module by simply making a copy of SimCall( ) function: highlight the program text, copy the text from the first module to the second module, adjust the text relative to the name of the new function, and change the payoff calculation to read:

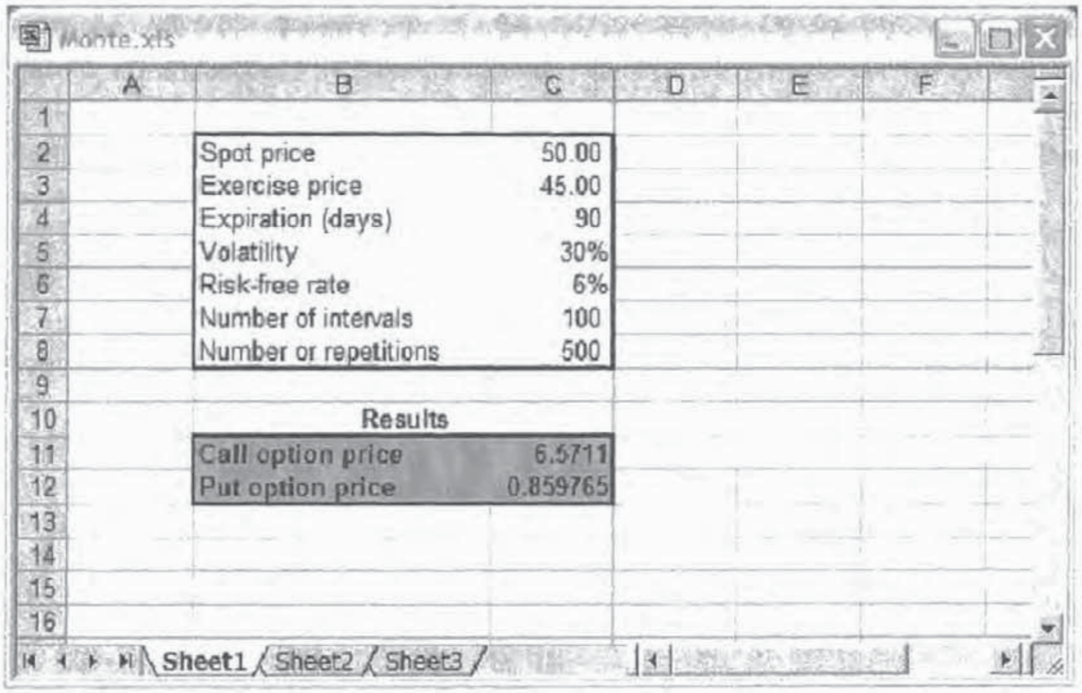
If spot < X Then Let payoff = payoff + (X - spot)

instead of

If spot > X Then Let payoff = payoff + (spot - X) .

Again, VBA functions are invoked either from within a worksheet or by another VBA routine. In this case, we want to estimate the option value based on parameters already entered into worksheet cells. This can be accomplished simply by typing the

Figure 4. Simulation Result



	A	B	C	D	E	F
1						
2		Spot price	50.00			
3		Exercise price	45.00			
4		Expiration (days)	90			
5		Volatility	30%			
6		Risk-free rate	6%			
7		Number of intervals	100			
8		Number or repetitions	500			
9						
10		Results				
11		Call option price	6.5711			
12		Put option price	0.859765			
13						
14						
15						
16						

formula =simcall(C2,C3,C4,C5,C6,C7,C8) into a worksheet cell. Figure 4 illustrates the results of one such simulation.

As noted previously, increasing the number of simulations and time intervals within a simulation increases the accuracy of the Monte Carlo analysis. However, it is not uncommon for the simulated option price to be \$0.20 or more above or below the theoretical Black-Scholes option price unless many simulations and successively smaller time intervals are used. This can lead to greatly increased execution time and can cause Excel to "crash". Further, when the reader performs the Monte Carlo simulation, it is very unlikely that results identical to the result displayed in Figure 4 will emerge (due to the nature of random numbers). In fact, the reader should not be surprised by the varying results that emerge from successive implementations of the Monte Carlo analysis. Consequently, some improvements to the simulation are necessary (see chapter 18.6 of Hull (2003) for a more extensive discussion).

One improvement is to increment the price based on simulating the risk-free return process for the underlying security. The return process is simply the natural logarithm of the price process. The (risk-neutralized) price process is GBM, $N(R_F S_t dt, [\sigma S_t]^2 dt)$, where R_F is the risk-free rate. By applying Ito's lemma, the risk-free return process is ABM, $N([R_F - 0.5\sigma^2]dt, \sigma^2 dt)$ (see Arnold and Henry (2003) for a discussion of Ito's lemma and this particular transformation). To illustrate how the simulation works, suppose the initial risk-free rate is 5% annually with dt equal to 0.25 years and an initial underlying security price of \$50.00. Moving forward one time step of dt, the risk-free rate increases hypothetically by an increment of 0.1% annually (or 0.025% relative to the time step) making the new risk-free rate 5.1% annually. The new underlying security price becomes \$50.00 × *exp*(5.1% × 0.25) = \$50.64, where *exp()* is the exponential function. Although it is the risk-free rate process that is simulated, a corresponding simulation is generated for the underlying security price. The greater precision of using the risk-free return process simulation relative to the previous GBM simulations (as measured against the Black-Scholes option price) is impressive. As to why the risk-free return process simulation works much better, the interested reader is again referred to Hull (2003).

Making the change to the VBA code is straightforward. Rather than simulating changes in the stock price (denoted *ds* in SimCall( )), we want to simulate the rate of return process (to be denoted by *dr*). We simply edit the SimCall( ) function, replacing the lines:

```
  ds = Application.NormInv(Rnd(), spot * rf * dt,  sigma * spot * Sqr(dt))
spot = spot + ds
```

With the following code:

```
  dr = Application.NormInv(Rnd(), (rf - (0.5 * sigma ^ 2)) * dt, sigma * Sqr(dt))
spot = spot * Exp(dr)
```

Where the original program generated simulated observations of changes in the stock price (normally distributed with mean $S_t \times R_f \times dt$ and standard deviation $\sigma \times St \times \sqrt{dt}$), the revised version simulates *dr*, which has a mean of $[R_f - (\sigma^2/2)]dt$, and a standard deviation of $\sqrt{dt}$). The spot price of the stock at any point in time is calculated as the previous period's price multiplied by e^{dr}.

Because there is no optimal rule for setting the number of "intervals" and "repetitions" within a Monte Carlo analysis, experimentation by the modeler is encouraged. In the current case, an analysis with 30 intervals within 5000 repetitions (using the revised function) generally produces accurate option prices when compared to the Black-Scholes value. However, the simulation will not be perfect every time and one must consider the value of computer time when increasing intervals and/or repetitions relative to the value of pricing accuracy. As computer technology improves, this issue becomes less important.

CONCLUSION

Monte Carlo analysis is an intuitive method for estimating the probability of future events. When coupled with risk-neutral pricing, it can make the pricing of options a relatively simple exercise, but at the same time it emphasizes how much of the analysis is under the modeler's control. In this paper, determining the appropriate number of repetitions of a simulation and the number of intervals within a simulation is emphasized. However, making a decision between the use of GBM or ABM process cannot be overlooked, nor can determining the parameters used within the given stochastic process be taken lightly.

Combined with a basic understanding of stochastic processes, the rudimentary programming techniques introduced in the previous section can enable students to conduct much more comprehensive Monte Carlo analysis in Excel. For example, even continuing with the option pricing examples from this paper, one can introduce the payment of dividends or investigate alternative volatility specifications for the underlying stock. The models can be relatively simple like the model of Arnold and Henry (2003) or more complex like the models of Johnson and Shanno (1987) and Das and Sundaram (1999). Ultimately, the goal is to enable the student to understand and become comfortable with Monte Carlo analysis and basic stochastic processes. By using Excel, the student and teacher can accomplish this goal without the need for expensive additional software.

ENDNOTES

[1] Note that literal Excel command strings are printed in courier type, while *italics* denote symbolic values (which will be replaced with cell references in Excel).

[2] Jackson and Staunton (2001) provides an excellent introduction to this material.

[3] *Sheet* and *workbook* code modules are intended to store *event procedures*, that is, code to be executed when particular events (such as mouse clicks and key presses) occur.

REFERENCES

Arnold T. and Crack, T.F. (2003) The Irrelevance of Risk-Adjusted Discount Rates in Option Pricing (Risk Neutral Pricing Without Risk Neutral Hedging). *Working Paper.*

Arnold T. and Henry, S.C. (2003) Visualizing the Stochastic Calculus of Option Pricing with Excel and VBA. *Journal of Applied Finance.* 13(1) 56-65.

Black, F. and Scholes M. (1973) The Pricing of Options and Corporate Liabilities. *Journal of Political Economy.* 81(3) 637-654.

Das, S.R. and R.K. Sundaram, R. K. (1999) Of Smiles and Smirks: A Term Structure Perspective. *Journal of Financial and Quantitative Analysis.* 34(2) 211-239.

Hull, J. (2003) *Options, Futures, and Other Derivatives.* Upper Saddle River, NJ: Prentice Hall.

Jackson, M., and Staunton, M. (2001) *Advanced Modeling in Finance using Excel and VBA.* New York, NY: John Wiley & Sons

Johnson, H. and Shanno D. (1987) Option Pricing when the Variance is Changing. *Journal of Financial and Quantitative Analysis.* 22(2) 143-151.

Johnstone, D. (2002) Risk-Neutral Option Pricing From EPV Without CAPM. *Journal of Financial Education.* 28(Summer) 72-78.

Neftci, S.N. (2000) *An Introduction to the Mathematics of Financial Derivatives.* San Diego,CA: Academic Press.