Using Simpson's Rule to Evaluate the Normal Cumulative Distribution Function in an Algorithmic Option Pricing Problem

Jeff Whitworth

Algorithmic problems are frequently used to help ensure academic honesty during assessments in quantitative courses, as they allow instructors to easily generate multiple versions of a test, quiz, or assignment with questions that are similar but whose input variables and correct answers differ. In general, instructors who create an algorithmic problem must program its solution by entering an algebraic expression using the input variables. While this is usually straightforward enough, it can be problematic when the solution requires a more advanced function that has no algebraic representation. One such challenge occurs with financial option pricing problems based on the Black-Scholes model, which relies on the normal cumulative distribution function (CDF). Although students can easily obtain normal CDF values from a calculator or spreadsheet function, no such function is available within testing software, presenting an obstacle for instructors attempting to author algorithmic problems on option pricing. I demonstrate how to overcome this difficulty by using Simpson's Rule to numerically integrate the underlying normal probability density function, thereby generating a highly accurate approximation of the CDF that can be programmed into the software and used in the Black-Scholes formula.

INTRODUCTION

An *algorithmic problem* is a test question or homework exercise (usually computational in nature) that can be presented to students in different ways, such that each variation of the problem requires the same solution method but has a different correct answer computed from different values of randomly chosen input variables. In recent years, algorithmic problems have become more widely used in quantitatively oriented courses, as textbook publishing companies have included them in automatically graded online homework (e.g. Cengage MindTap, McGraw-Hill Connect, Pearson MyLab) and/or in test generation software provided to instructors (e.g. ExamView, Cognero, TestGen). Instructors can author their own algorithmic problems within many of these publisher-provided resources, and within commonly used course management systems (e.g. Blackboard) as well. Being able to quickly and easily generate multiple versions of the same problem has obvious advantages in assessment and learning. First, instructors can create multiple versions of a test or quiz that are of equal difficulty and assess the same learning objectives but have different answers to each problem to help deter cheating. In addition, it is easy to give students additional practice problems. This may be extremely helpful, for example, when a struggling student who has finally found the solution to a particular homework exercise needs to check his/her skills on a new (but similar) problem with a different answer.

Designing a good algorithmic problem requires a little more work upfront than simply creating a regular "static" problem. At a minimum, the author must specify (1) one or more variables whose values are to be randomly chosen with each instance of the question, (2) allowable ranges for these variables, and (3) a formula indicating how to calculate the correct answer. For multiple-choice assessments, distractors (i.e. incorrect answers) may also be calculated from formulas based on common calculation errors that students might make, or they may simply be randomly generated values that are close enough to the correct answer to be plausible.

For most problems, only the standard mathematical operators (addition, subtraction, multiplication, division, and exponentiation) are needed to write a formula that will compute the correct answer, although test-generating programs generally recognize at least a few additional functions (e.g. logarithms, factorials, trigonometric functions) since they are needed often enough in mathematics courses. However, there are times when the testing software lacks a function needed to solve a problem. For example, the normal cumulative distribution function (CDF) is not included in currently available test generation software or learning management systems, yet it has numerous applications in probability theory, statistical hypothesis testing, and financial option pricing. While students generally can use a table or graphing calculator function to find the appropriate values, instructors attempting to create algorithmic problems face a challenge, since their testing software incorporates no such tables or functions, and the CDF itself has no algebraic representation.

Despite this difficulty, it is possible to create good algorithmic problems that require use of the normal CDF. Although the underlying probability density function cannot be integrated analytically, it can be integrated numerically. Simpson's Rule is a powerful technique capable of estimating definite integrals very closely – much more so than other numerical methods (e.g. Riemann sums, trapezoidal method). For the normal CDF, this method produces approximations that are accurate to at least five decimal places using a relatively small number of values of the underlying density function. This paper shows how to create a user-defined function in ExamView (a test-generation program distributed with thousands of textbooks) that precisely approximates the normal CDF using Simpson's Rule, and how to use this function to program the solution to an algorithmic problem where students must price a financial option using the Black-Scholes formula.

The next section reviews the Black-Scholes option pricing model and the standard normal distribution. I show how to numerically approximate the integral needed to compute CDF values. Next, I demonstrate how to set up an algorithmic option pricing problem in ExamView, define its input variables, and create the aforementioned function that is ultimately used to find the solution to the problem. The final section concludes and discusses other potential applications of this technique.

THEORY

Black and Scholes (1973) showed that in the absence of taxes, transaction costs, and short-selling restrictions, the value of a call option on a stock that does not pay dividends is

$$C = SN(d_1) - Ke^{-rT}N(d_2),$$

where S is the stock's current market price, K is the option's exercise price, r is the continuously compounded risk-free interest rate, T is the number of years until option expiration, and σ is the annualized volatility of the stock's returns (which are assumed to be continuous, normally distributed, and serially independent). The quantity d_1 is defined as

$$\frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}},$$

and $d_2 = d_1 - \sigma\sqrt{T}$. $N(d_1)$ and $N(d_2)$ are the respective values of the standard normal CDF at d_1 and d_2.

The standard normal *probability density function* (pdf) is given by

$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$$

for $-\infty < x < \infty$. The cumulative distribution function $N(z)$, which denotes the probability that a random variable drawn from the standard normal distribution will be less than z, is calculated by integrating the pdf over all values up to z:

$$N(z) = \int_{-\infty}^{z} f(x)dx = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx$$

As is necessary for any well-defined probability distribution, the area under the entire density function is 1, so that

$$N(\infty) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx = 1.$$

Furthermore, since the density function $f(x)$ is symmetric about $x = 0$,

$$N(0) = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx = \frac{N(\infty)}{2} = 0.5,$$

so that for any real value of z,

$$N(z) = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx + \int_{0}^{z} \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx = 0.5 + \frac{1}{\sqrt{2\pi}}\int_{0}^{z} e^{-x^2/2}dx.$$

Unfortunately, this cannot be evaluated directly since $e^{-x^2/2}$ has no closed-form antiderivative. However, numerical techniques can accurately approximate the values of definite integrals for most functions, even those which cannot be integrated analytically. These methods generally involve dividing the interval over which the function is to be integrated into subintervals, using one or more values of the function within each subinterval to estimate the area under that portion of the curve, and then adding up all of the estimated areas.

For example, to estimate $N(1.6)$, which requires evaluation of $\int_0^{1.6} e^{-x^2/2}dx$, one might divide [0,1.6] into four subintervals of [0, 0.4], [0.4, 0.8], [0.8, 1.2], and [1.2, 1.6], and then multiply the function's value at each midpoint by the width of each subinterval. Using this method, the estimated value of the integral would be

$$\int_0^{1.6} e^{-x^2/2}dx \approx 0.4e^{-0.2^2/2} + 0.4e^{-0.6^2/2} + 0.4e^{-1.0^2/2} + 0.4e^{-1.4^2/2} = 1.11892.$$

Our estimate of $N(1.6)$ would then be $0.5 + \frac{1}{\sqrt{2\pi}}(1.11892) = 0.94639$, which overestimates the actual value of 0.94520 by 0.00119.

This method (known as the "Midpoint Rule") is often sufficient, but there are times when an even closer approximation is desirable. Simpson's Rule is a more powerful numerical integration technique that models the interval endpoints as points along a parabola, thereby taking into account the function's curvature. Because of the method used, Simpson's Rule is sometimes referred to as the Parabolic Rule. To use it, we again partition the interval [x_0, x_n] over which the function will be integrated into a series of n subintervals [x_0, x_1], [x_1, x_2], …, [x_{n-1}, x_n] of equal width, and apply the formula

$$\int_{x_0}^{x_n} f(x)dx \approx \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)],$$

where $h = \frac{x_n - x_0}{n}$, the width of each subinterval. Unlike the Midpoint Rule where any number of intervals can be used, Simpson's Rule requires that n be even. Note that the coefficients in the bracketed expression are 1 for the initial and last terms in the series, 2 for the other even-numbered

terms, and 4 for the odd-numbered terms. Using this method to evaluate $\int_0^{1.6} e^{-x^2/2}dx$ with the same four subintervals from the previous example yields

$$\int_0^{1.6} e^{-x^2/2}dx \approx \frac{0.4}{3}[e^{-0.0^2/2} + 4e^{-0.4^2/2} + 2e^{-0.8^2/2} + 4e^{-1.2^2/2} + e^{-1.6^2/2}] = 1.11597.$$

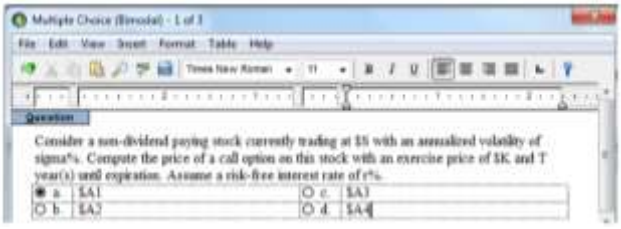
$N(1.6)$ would then be estimated as $0.5 + \frac{1}{\sqrt{2\pi}}(1.11597)$ = 0.94521, which differs from the correct value by only 0.00001. This is an incredibly close approximation. Even greater precision is achievable by using more (and narrower) subintervals. With ten subintervals, the maximum estimation error for any value of z is 3.7×10^{-6} (realized when $z = \pm 2.8$), making the approximation accurate to at least five decimal places in all cases.

APPLICATION

I now show how to set up an algorithmic option pricing problem and use Simpson's Rule to program the solution indicated by the Black-Scholes model. Although this process is demonstrated with ExamView Test Generator, the same or similar methods may be implemented in other testing programs.

Within the question bank editor, we begin by opening a dialog box to create a new question. In ExamView, I generally select the "bimodal" question type, as it allows the instructor to build a multiple-choice question now yet retain the ability to present it to students later in free-response format. We enter the question and the answer choices as follows:

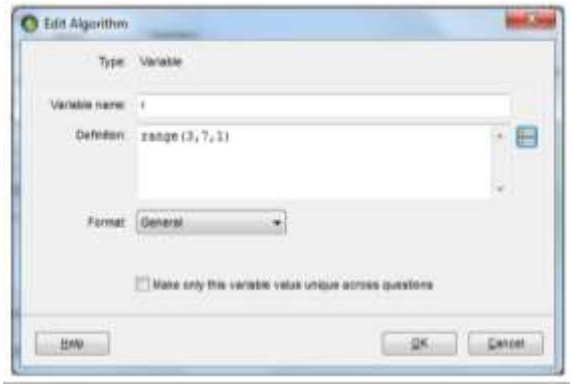
Figure 1: Initial Problem Setup



Note that we have initially typed the input variable names (`S`, `sigma`, `K`, `T`, and `r`) and possible answer choices (`A1` through `A4`) as placeholders, without actually defining what values to display. We must now specify parameters for randomly generating the inputs and how to compute each of the answer choices. To do so, we select "Edit" and then "Algorithm Definitions" from the menu. In the "Edit Algorithm Definitions" window that appears, we then click the "New" button, bringing up the "New Algorithm" dialog box.

We start by defining `r`, the risk-free interest rate, to be a random value between 3 and 7 percent, with increments of 1 percent – i.e. randomly chosen from the set {3, 4, 5, 6, 7}. Note that ExamView does not have an option to define a variable as a decimal number (e.g. 0.05) and then display it to students as a percentage (e.g. 5%), so we define `r` as a whole number and then divide it by 100 as necessary in subsequent computations. Therefore, we enter `range(3,7,1)` for the definition, as shown below.

Figure 2: Defining an Input Variable



These parameters are obviously customizable if the instructor wishes to use a different allowable range of interest rates or finer increments than a full percentage point between possible rates.

After clicking "OK," we then return to the "Edit Algorithm Definitions" window and click "New" again. This time, we define `T`, the number of years to option expiration, as `range(0.25,1,0.25)` – i.e. randomly chosen from 0.25, 0.50, 0.75, or 1.00 year(s). Again, this is customizable based on the instructor's preference.

Next, we define `K`, the option's exercise price in dollars, as `range(30,75,5)` – i.e. a random value from the set {30, 35, 40, … 70, 75}.

To make the problem realistic, and to ensure that the student knows how to use both variables properly in the Black-Scholes formula, the current stock price should be different from but reasonably close to the option's exercise price. We therefore define `S` as `K+range(-8,8,1)` to ensure that `S` will be within $8 of `K`. Additionally, because the `range` function never returns zero when its lower and upper limits have opposite signs, `S` and `K` will have distinct values.

We define `sigma`, the annualized standard deviation (i.e. volatility) of the stock's return, as `range(20,50,1)` – a value between 20 and 50 percent. As with the interest rate, `sigma` is defined as a whole number since ExamView offers no formatting option to display a decimal number to students as a percentage. To facilitate subsequent calculations, we then define `var` as `(sigma/100)^2`, representing the variance of the stock's return in decimal form. While `var` is never actually displayed to students in this question and is not absolutely essential to define here, it is convenient to do so since the variance appears in the option pricing formula.

Next, we define the variable `d1`, corresponding to d_1 in the Black-Scholes model. The appropriate formula is

`(ln(S/K)+(r/100+var/2)*T)/sqrt(var*T)`. The variable `d2` is then analogously defined as `d1-sqrt(var*T)`.

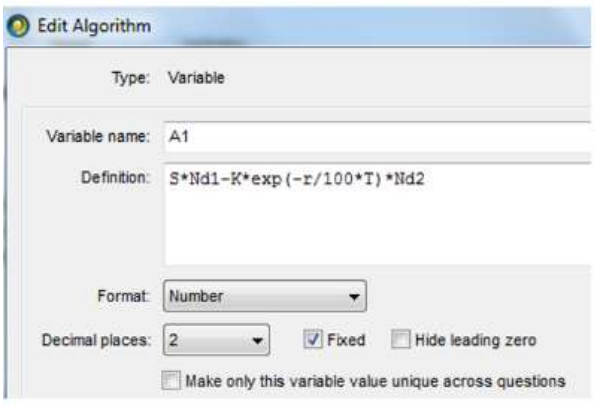
The Black-Scholes formula now requires us to evaluate the standard normal CDF at both `d1` and `d2`. As previously discussed, ExamView does not have a built-in function for computing this directly. However, it does allow the user to create additional functions. To do this, we bring up the "New Algorithm" dialog box again but this time select "Function" from the "Type" drop-down menu. Entering `N(z)` for the function name indicates that whenever `N` is called, the user must specify a single argument `z` from which the function's value will be calculated. The following formula is entered for the function definition:

```
0.5 + z/30 * (1+4*exp(-.5*(.1*z)^2)
+2*exp(-.5*(.2*z)^2)+4*exp(-.5*(.3*z)^2)
+2*exp(-.5*(.4*z)^2)+4*exp(-.5*(.5*z)^2)
+2*exp(-.5*(.6*z)^2)+4*exp(-.5*(.7*z)^2)
+2*exp(-.5*(.8*z)^2)+4*exp(-.5*(.9*z)^2)
+exp(-.5*z^2)) / sqrt(2*pi)
```

This directly applies the method presented in the previous section, using Simpson's Rule with ten subintervals to evaluate $N(z)$. Note that `z/30` in the expression above corresponds to $h/3$ from the previously stated Simpson's Rule formula, since the definite integral of $e^{-x^2/2}$ is being evaluated from 0 to z, so that the width h of one subinterval is $z/10$. Also note that if `z` happens to be negative, the above function definition works just as well since $N(-z) = 1 - N(z)$ due to the symmetry of the normal distribution.

From this point, we proceed to create two more variables `Nd1` and `Nd2`, defining them as `N(d1)` and `N(d2)` using our newly-created function. If our test-generating software did not permit user-defined functions, we could simply write expressions for `Nd1` and `Nd2` identical to that for `N(z)` above, but substituting `d1` or `d2` for `z` as appropriate.

Figure 3: Programming the Correct Answer



We are now ready to program the correct answer to the problem. To do so, we create variable `A1` and define it as `S*Nd1-K*exp(-r/100*T)*Nd2`, in accordance with the Black-Scholes formula. As shown above, we format this variable as a number rounded to 2 decimal places and check the "Fixed" box to ensure that both decimal places will always be displayed, even if the answer ends in one or more zeroes.

For the distractors (i.e. incorrect answer choices), we create three variables `A2`, `A3`, and `A4`, each formatted the same as `A1`, and each defined as `A1*range(0.4,1.6,0.001)`. This causes each incorrect answer choice to have a random value from 40-160% of the correct answer. As always, this range is adjustable, and some instructors may even choose to program some or all of the distractors based on mistakes that students could potentially make. For example, we might define `A2` as `K*exp(-r/100*T)*(1-Nd2)-S*(1-Nd1)`, the value of a put option with the same parameters; and we might let `A3` be `max(0,S-K)`, the call option's exercise value (which is less than its market value).

Regardless of how the distractors are defined, it is a good idea to ensure that all answer choices are sufficiently different from each other to eliminate potential confusion. To include such a condition in ExamView, we bring up the "New Algorithm" dialog box again and change the algorithm type to "Condition." We then enter the following:

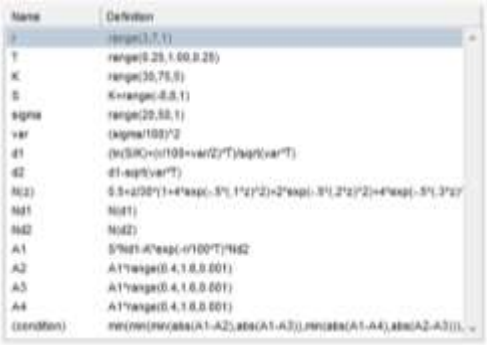
```
min(min(min(abs(A1-A2),abs(A1-
A3)),min(abs(A1-A4),abs(A2-A3))),
  min(abs(A2-A4),abs(A3-A4)))>=0.2
```

This ensures that no two answer choices are closer than $0.20, a difference large enough that no student should miss the question merely due to a rounding error. If the program happens to generate a set of values that would violate this condition, it will go back and generate new values until the condition is satisfied. Unfortunately, the above expression is necessarily somewhat complex because ExamView's `min` function can evaluate only two arguments at a time. If a different software package or an updated version of ExamView allowed for multiple arguments, this condition could be simplified to the following:

```
min(abs(A1-A2),abs(A1-A3),abs(A1-
A4),abs(A2-A3),abs(A2-A4),abs(A3-
A4))>=0.2
```

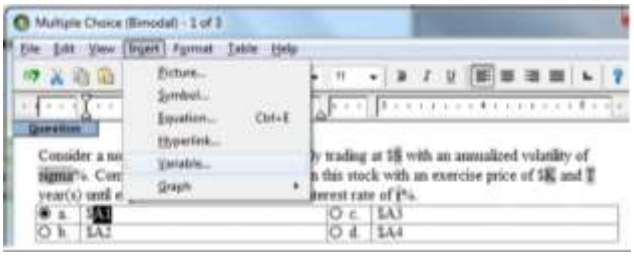
The list of algorithm definitions should now look something like this:

Figure 4: Final List of All Variables, Functions, and Conditions Defined



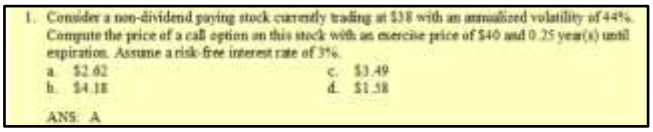
At this point, we click "Done" to return to the main question window. We now highlight each of the variables in turn and select "Insert" and "Variable" from the menu. This instructs the program to actually display the variable's value (rather than the name typed previously) when an iteration of the question is presented to students. In the illustration below, we have done this for variables `S`, `sigma`, `K`, `T`, and `r` in the question stem (all of which are now highlighted in gray) and are about to do so for the first answer choice `A1`.

Figure 5: Inserting the Variables into the Problem



After doing this for all of the variables shown (including answer choices `A1` through `A4`), we click "Record" and return to the main question bank editor window, at which point we should see randomly generated values for all of the input variables, along with the calculated correct answer displayed in choice "a":

Figure 6: Problem Displayed in Question Bank with Random Variables



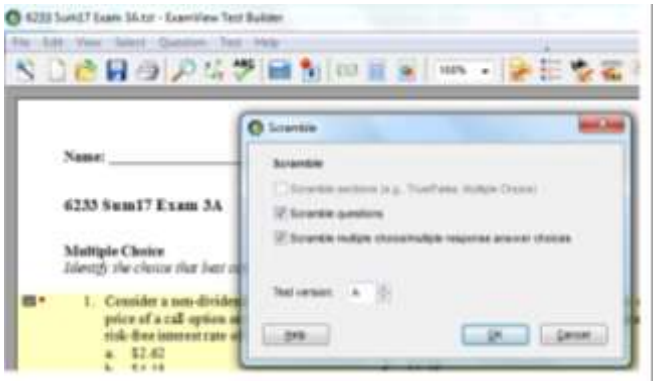
1. Consider a non-dividend paying stock currently trading at $38 with an annualized volatility of 44%. Compute the price of a call option on this stock with an exercise price of $40 and 0.25 year(s) until expiration. Assume a risk-free interest rate of 3%.

a. $2.62 c. $3.49
b. $4.18 d. $1.58

ANS: A

Regardless of which answer the instructor programs to be the correct one, it is generally a good idea to have ExamView scramble the answer choices each time it generates a new exam or quiz version, so that for instance, a student from an earlier section cannot simply tell someone in a later section to "mark 'A' on the Black-Scholes call option problem." Fortunately, this is easy to address by making sure that the default choice "All" is selected from the "Scramble" drop-down menu on each individual question; then before finally exporting the test to a printable document, select "Question" and "Scramble" from the menu before checking the appropriate box(es) as shown below:

Figure 7: Scrambling the Questions and Answer Choices in an Exam



DISCUSSION AND CONCLUSION

Algorithmic problems are very useful time-saving tools that allow instructors in quantitative courses to easily generate multiple versions of an assessment that are equal in difficulty, yet with questions whose details and correct answers differ so that it is hard for students to cheat, either by observing a nearby student's test paper in the same classroom or by receiving information from someone in a different section of the course. While this does not relieve the instructor of the responsibility to monitor students, it does help to encourage academic integrity and frees up some time and energy that might otherwise be spent dealing with honesty violations. In addition to these benefits, algorithmic problems can also be valuable for instructors who wish to provide students with more practice problems on a given topic.

While it is usually fairly straightforward to enter the formula for the correct answer to an algorithmic problem into testing software, occasionally it is more challenging. Financial option pricing problems present a challenge because the normal cumulative distribution function (CDF), which is required to use the Black-Scholes formula, is neither built into test-generating programs nor representable with a closed-form algebraic expression. This paper explains how Simpson's Rule can be used to numerically integrate the standard normal probability density function, and how to implement this method to create a user-defined function within ExamView Test Generator that approximates the normal CDF very precisely. I demonstrate how to set up an algorithmic option pricing problem in ExamView and then use the new function to program the solution indicated by the Black-Scholes option pricing model. In slightly less versatile

software packages that permit users to define variables but not functions, the same method could still be applied to each variable for which the normal CDF needed to be evaluated with minimal additional effort.

Although the numerical approximation method presented here is demonstrated in the context of a financial asset pricing problem, it is also useful in any situation where students must use the normal distribution to compute probabilities or test hypotheses. Of course, many statistical tests are based on different distributions, but Simpson's Rule could certainly be effective in integrating other probability density functions where a closed form CDF does not exist. As exemplified here, writing algorithmic problems on advanced topics does occasionally present challenges, but with creativity, they can usually be overcome.

REFERENCES

Black, Fischer and Myron Scholes, 1973, The Pricing of Options and Corporate Liabilities, *Journal of Political Economy* 81, 637-654.

Jeff Whitworth is an Associate Professor of Finance at the University of Houston-Clear Lake.