Mathematics of Finance, Courant Institute, Fall 2015

http://www.math.nyu.edu/faculty/goodman/teaching/mathFin/index.html

Always check the class message board before doing any work on the assignment.

Assignment 8, due *Monday*, November 30

Corrections: (none yet. See message board)

1. (*Please do this by Monday, November 23. Do not hand it in then*) The binomial tree model is based on the random dynamics

$$S \longrightarrow \left\{ \begin{array}{ll} u\,S\ , & u = 1+\sigma\sqrt{\Delta t}\ ,\ \text{ prob } = q_u = \frac{1}{2}\left(1+\frac{r}{\sigma}\sqrt{\Delta t}\right) \\ d\,S\ , & d = 1-\sigma\sqrt{\Delta t}\ ,\ \text{ prob } = q_d = \frac{1}{2}\left(1-\frac{r}{\sigma}\sqrt{\Delta t}\right) \end{array} \right. \tag{1}$$

This is what happens in an interval of time of size Δt. We have not emphasized it, but every step in the tree is independent of every other step. Steps in the future, whether to go up or down, are not influenced by decisions in the past. The parameters are chosen so that

$$\left. \begin{array}{ll} \mathrm{E}_{\mathrm{RN}}[S_{t+\Delta t} - S_t] & = rS_t\Delta t \\ \mathrm{var}_{\mathrm{RN}}(S_{t+\Delta t} - S_t) & = \sigma^2 S_t \Delta t\ . \end{array} \right\} \tag{2}$$

Here, t could be any of the discrete times $t_k = k\Delta t$, and then $t+\Delta t = t_{k+1}$. We sometimes *abuse notation* by writing S_k to mean $S_{t_k} = S_{k\Delta t}$. The binomial tree log process is $X_t = \log(S_t/S_0)$. When we do the binomial tree, we write $X_k = \log(S_k/S_0)$. The log is always the natural log, which is the log base e with $\log(e) = 1$.

   (a) Calculate the expected value and variance in (2) exactly using the probability rules (1). Show that the formulas (2) are not exact, but are correct but up to order Δt as given.

   (b) Show that $X_{k+1} = X_k + Y_k$, where the Y_k is a random variable with two possible values. These values and probabilities are the same for every k. What are those values and the probabilities of getting those values?

   (c) Calculate the Taylor series of the log function up to second order: $\log(1+\varepsilon) = a\varepsilon + b\varepsilon^2 +$ higher order. There is no constant term because $\log(1) = 0$. Find a and b. (We need this because we will take $\varepsilon = \sqrt{\Delta t}$, or something like that, and we need the result correct up to order Δt.)

   (d) Calculate the mean, $\mathrm{E}_{\mathrm{RN}}[Y_k]$. Do not give the exact result, but give it only up to order Δt.

   (e) Show that $\mathrm{var}_{\mathrm{RN}}[Y_k] = \sigma^2\Delta t$, at least up to order Δt.

(f) Find an approximate formula for $m = \mathrm{E}_{\mathrm{RN}}[X_n]$ and for $v = \mathrm{var}_{\mathrm{RN}}[X_n]$. Use the approximations of parts (d) and (e) to do this. Hint: For the variance, use the fact that the Y_k are independent. We write v for the variance, instead of the usual σ^2, because σ is the volatility of S, which is not the same thing as the variance of X_n.

(g) A more accurate formula for the variance has the form

$$\mathrm{var}_{\mathrm{RN}}[Y_k] = \sigma^2 \Delta t + a \Delta t^2 \ .$$

Show that the value of a does not change the result of (f). That is, as $\Delta t \to 0$, there is a limiting value of v which does not depend on a. (This is why we calculate the results of (d) and (e) to include the Δt term but not Δt^2 and higher terms, which do not effect the important result (f).)

(h) The *central limit theorem* states that if random variable X_n is the sum of a large number of independent random variables Y_k that have the same distribution, then X_n is approximately Gaussian. Use the central limit theorem to write an approximate formula for the PDF of X_n for large n. Hint: A Gaussian distribution is determined by the mean, m, and the variance, v.

2. Suppose X is a normal random variable with mean μ and variance σ^2, which we write $X \sim \mathcal{N}(\mu, \sigma^2)$. Suppose $S = S_0 e^X$ Calculate $\mathrm{E}[S]$ and $\mathrm{var}[S]$ in terms of μ and σ. Some hints to make the calculations quicker: X has the same density as $\mu + \sigma Z$, where Z is a *standard normal* with PDF $f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$. You can write the expectation as an integral over z. You can do it by completing the square in the exponent, and then using the fact that

$$\int_{-\infty}^{\infty} e^{-\frac{1}{2}(z-a)^2} \, dz = \sqrt{2\pi} \ ,$$

for any value of a. For the variance, note that $S^2 = S_0^2 e^{2X}$. Therefore $\mathrm{E}\left[S^2\right]$ involves a Gaussian with mean 2μ and variance $4\sigma^2$. (The Black Scholes formula uses these calculations.)

3. (*Extra credit*) A function $f(x)$ is *convex* if the graph "bends upward". That means, the slope increases as you go right, which means that the second derivative is positive so the first derivative is increasing. That's the calculus point of view. The geometric point of view is that the line (more properly, line segment) connecting two points on the graph lies above the graph. Two points on the graph, in (x, y) coordinates, are $(a, f(a))$ and $(b, f(b))$. Suppose t is a number in the interval $[0, 1]$. A point on the line segment has $x-$coordinate $x_t = ta + (1-t)b$, and $y-$coordinate $y_t = tf(a) + (1-t)f(b)$. This point is above the graph if the $y-$*coordinate* on the line is larger than the $y-$coordinate on the graph:

$$y_t = tf(a) + (1-t)f(b) > f(x_t) = f(ta + (1-t)b) \ . \qquad (3)$$

The probability point of view looks at this inequality using probability. The numbers t and $1-t$ are both positive and they add up to 1. Therefore, we may call them probabilities, $p_a = 1$, and $p_b = 1 - t$. We think of p_a as the probability that $x = a$ and p_b as the probability that $x = b$. Then the $y-$coordinate on the line is the expected value of the random variable $f(X)$:

$$\begin{aligned} \mathrm{E}[f(X)] &= f(a)\Pr(X = a) + f(b)\Pr(X = b) \\ &= f(a)p_a + f(b)p_b \\ &= tf(a) + (1-t)f(b) \\ &= y_t \ . \end{aligned}$$

In the same way, the $x-$coordinate is the expected value of X. The geometric inequality (3) is equivalent to the inequality of expected values:

$$\mathrm{E}[f(X)] > f(\,\mathrm{E}[X]) \ . \qquad (4)$$

This is called *Jensen's inequality.* It is true for any random variable X with any PDF. If $f(x)$ is a convex function of x, the inequality (4) applies. We didn't give the proof here, but it isn't hard.

We call the numbers in a binomial tree "convex" if they satisfy the geometric inequality (3) as much as possible. Show that the binomial tree prices for a European call are a convex function of $X = \log(S)$ in this sense. This essentially means that $q_u f_{k,j+1} + q_d f_{k,j-1} > f_{kj}$. Hint: at the expiration time $k = n$ this is true because the payout is related to e^X. It's true at stage k if it's true at stage $k + 1$.

4. Forward pricing works as follows (see Chapter 4). You agree today to do a transaction at time T. No money changes hands today. At time T, party A will produce a risky asset and receive payment F (the forward price, which is the price of the forward contract). Party B will take delivery of the risky asset and pay F. The spot price of the risky asset today is S_0 and is known today. The price at time T is S_T and is not known today. There is a fixed continuous interest rate r. Party A can hedge today by borrowing S_0 and using the money today to buy the risky asset. At time T, party A will deliver the risky asset and receive F. Party A owes S_0e^{rT}. If $F = S_0e^{rT}$, then party A has exactly the right amount of cash to repay the loan. If $F > S_0e^{rT}$, then party A has an arbitrage opportunity, because she will have $F - S_0e^{rT} > 0$ left, and without risk. (This is the *cash and carry* argument for pricing forward contracts. A forward contract is often just called a "forward". The price $F = S_0e^{rT}$ is the *forward price.*)

   (a) Show that if $F < S_0e^{rT}$, then party B has an arbitrage.

   (b) Suppose the risky asset is delivered at time T but the cash is delivered at a different time $t \neq T$. What should F be so that neither party has an arbitrage opportunity?

(c) Suppose it's a *layaway*[1] contract in which party B pays a continuous "coupon" with rate c (that is, there is a payment $c\,dt$ for each time interval dt) up to time T, when party B stops paying and A delivers the risky asset. What should the payment rate be so that neither party has an arbitrage?

5. The program this week illustrates *modularity*, which is a principle of all computer programming, and *scientific visualization*, which is a principle of scientific computing. You can't do any but the most basic programming if your codes aren't modular. You can't know what answer you got if you can't graph it.

   Modularity means identifying sub-tasks and writing separate pieces of code called *functions* (in `R`, *procedures* in C++, *methods* in Java) for them. Typically, the code that defines a function will be in a different file. There is an *interface* that defines how information is transferred into and out of the function. The *calling code* has two ways to transfer information to the function. One way is through function *arguments*. When the calling code *calls* the function, it copies the values of the calling arguments in the calling code to the corresponding variables in the function. (The other way is not mentioned here.) One way for the function to transfer information back to the calling code is by giving a *return* value.

   The code in `CallingCode.R` calls functions defined in the file `CalledCode.R`. The line `c = add( a, b)` *calls* the function `add` with arguments `a` and `b`. The `R` interpreter copies the first argument in the calling code, which is `a` to the first argument in the function definition, which is `x`. It also copies the value of second argument, `b`, to to `y`. It then executes the function definition, which is the code in lines 10 through 14 in `CalledCode.R`. You can see that `a` has the value `5`, which becomes the value of `x` inside the function `add`. The last line of `add` is just `sum`. The `R` interpreter takes this to be the *return value* of the function. It brings that value back as the value of `add` in the calling program. The interpreter than passes *control* back to the calling program and does the next line, which is `cat(" the sum of ", a,...`. The control in the calling program was passed to the function and then is passed back to the calling program when the function reaches its last close curley. The line `d = 7*add( a*b, 1)` shows that a `user defined` function (that's one you give the code for, not a builtin function like `exp`) can be used in the ways a builtin function is used. You can do arithmetic in defining the arguments, and you can do arithmetic on the return value.

   The line `add = function( x, y){` defines `add` as an *object* in `R` whose *value* is the function definition. An object in `R` can be a number, a character string, a list, or even a function. Every object has a name (such as `add` or `x`) and a definition. A definition could be something like: "I'm a floating point number and my value is 2.71828.". Or, it could be: "I'm a

[1]This is a real thing that you can find on the web.

list with 8 elements, which are ... ". Or, it could be: "I'm a function with two arguments and the following code...". The line `sad = add` creates an object whose name is `sad` and whose value is the value of the object `add`. In this case, that makes value of `sad` the function that adds `x` and `y`. The rest of the file defines a function called `BirthdayWish`. You can see that it has comments saying what it does. Every function needs comments like that. It has one argument, which should be a string.

The script `PlotScript.R` is an example of plotting in `R`. It calls a function called `mod_wave` to evaluate $f(x)$. The function depends on a parameter k, which is fixed but has to be communicated to the code that evaluates f. The function given is a *modulated wave* $f(x) = x\sin(kx)$. The *wave number* is k. *Modulated* means that the amplitude, A, of the wave $A\, sin(kx)$ is slowly changing (modulating). The comments in the code say how the plot function works.

Programming assignment: Write an `R` script to plot the option price of a European call as a function of the spot price S_0. Put the *pricer* in a separate file that defines a function `CallPrice(S_0, T, sig, r, K)`. The number of binomial tree steps, n, should be defined in the function. It should be large enough that the price is reasonably accurate (try some values to find a good one). Draw on the same plot the intrinsic value. Choose a range of S_0 on both sides of K so that you can see: (1) the option price is below the intrinsic value for deep in-the-money option, (2) the option price goes to zero for deep out-of-the-money options.

Here are some requirements for the code:

(a) It should be well commented, particularly the part that says what the pricer function does. The comments should say what the arguments to the pricer are. When a new variable is introduced, there is usually a comment saying what it's for.

(b) There should be white space to make the code easy to read.

(c) Variable names should be meaningful, not too long and not too short (unless they are mathematical variables like S, t, k, etc.).

(d) The plot titles should contain information about the option so that you can tell what you're looking at when looking at the plot.

(e) Hand in a couple of plots (at least two but maybe a few more, not too many) showing something interesting about the computation (suggestions: how the result depends on one or more parameters, or on the number of steps in the binomial tree.).