MATH 477: PROJECTS

1. Pricing an Asian option by Monte Carlo methods

Consider an Asian option that pays $\left(\frac{1}{D}(S_d(1)+\ldots+S_d(D))-K\right)^+$, where $S_d(i)$ is the share price at the end of the ith day.

Write code to implement a function that does a single run. The arguments to the function should be: s, the initial share price; D the time (measured in days); K, the strike price; r, the interested rate (annual rate, continuously compounded); and σ, the volatility of the stock. Or, even better, write code that does n runs simultaneously without looping, where n is a parameter. You would need to find a method to take partial sums across a matrix without loops to do this.

Each time the function is called, it should return the value from a single realization (or n realizations) of $S(t)$ (as a GBM with the risk-neutral distribution).

Write another function to collect a large number of these values, and to find the sample mean and variance.

Price an Asian option on a share with initial price \$61, strike price \$63, volatility 0.3, and interest rate 8% over 6 months. How accurate is your estimate?

How many times would you need to loop to obtain accuracy to within 1 cent? 0.1 cents?

Note: you should avoid storage of large amounts of data. Only store what you need for the computation - you probably don't need any arrays with more than D elements, or $n \times D$ elements; and not even that if you use looping. But: looping is slow, so it is probably a good idea to use one array of size $n \times D$ in the single run function.

2. The rate of convergence of binomial tree prices for European calls

This project is about comparing the cost of a European call obtained from a binomial tree with that obtained from the Black–Scholes formula. Write $C_{BS}(s, t, K, \sigma, r)$ for the Black–Scholes formula and $C_{BT}(s, t, K, \sigma, r, n)$ for the price from the binomial tree with n steps.

Let $\Delta(n) = C_{BT}(s, t, K, \sigma, r, n) - C_{BS}(s, t, K, \sigma, r)$. I want you to understand how big $\Delta(n)$ is. Compute values of $\Delta(n)$ for a wide range of values of n (maybe $n = 50, 100, 200, 400, 800, 1600, \ldots$) with $s = 48$, $t = 0.25$, $K = 50$, $\sigma = 0.32$ and $r = 0.1$)

You should expect to see something like $\Delta(n) \approx Cn^{-\alpha}$. The question is: what is that value of α?

Look up log–log plots or logarithmic regression and use these to estimate α.

Also try plotting $\Delta(n)$ against n for n in the range 1 to 1000.

Note: You can minimize the amount of storage you need. If you have n steps in your tree, then a 2-dimensional array $a_{k,i}$ has to store n^2 numbers. This is a major limitation on how big the values of n are that your program can compute.

However... Once you have computed the values in the binomial tree at the kth level, you don't need the values in the $(k+1)$st levels any more. What this means is that you can just use 1-dimensional arrays. Just keep track of the previous level and the current level. No other information is needed. Now you only have to store $2n$ numbers. This allows you to do the computation for much *bigger binomial trees.*

3. American put exercise boundary

Use the method from section 8.3 of the book to price an American put option on a large binomial tree (as large as possible). At each step, there is a level where it becomes better to hold the option than to exercise it. For each step, keep track of the share price at this level.

Use $s = 51$, $t = \frac{1}{2}$, $K = 53$, $\sigma = 0.32$ and $r = 0.05$.

Now plot a graph with time on the horizontal axis; and the share price on the vertical axis, showing the boundary between where you should exercise the option and where you should hold it.

Do this for several different values of n for the same option to verify your results.

Note: You can minimize the amount of storage you need. If you have n steps in your tree, then a 2-dimensional array $a_{k,i}$ has to store n^2 numbers. This is a major limitation on how big the values of n are that your program can compute.

However... Once you have computed the values in the binomial tree at the kth level, you don't need the values in the $(k+1)$st levels any more. What this means is that you can just use 1-dimensional arrays. Just keep track of the previous level and the current level. No other information is needed. Now you only have to store $2n$ numbers. This allows you to do the computation for much *bigger binomial trees.*

4. Pricing a barrier option by binomial trees

For this project you'll price a call option with an *up-and-out* barrier using binomial trees. Here the values at the nodes are a bit different.

The value at the (k, j) node, $a_{k,j}$ is the current value of the call **provided the share price has never been above the barrier**. (Of course the value is 0 if the share price has already been above the barrier).

Find the recurrence relation for the $a_{k,j}$'s. Working right to left in the binomial tree, obtain a price for a call option with an up-and-out barrier where $s = 51$, $t = \frac{1}{2}$, $K = 53$, $\sigma = 0.32$, $r = 0.05$ and the barrier is at 56.

Note: You can minimize the amount of storage you need. If you have n steps in your tree, then a 2-dimensional array $a_{k,i}$ has to store n^2 numbers. This is a major limitation on how big the values of n are that your program can compute.

*However... Once you have computed the values in the binomial tree at the kth level, you don't need the values in the $(k + 1)$st levels any more. What this means is that you can just use 1-dimensional arrays. Just keep track of the previous level and the current level. No other information is needed. Now you only have to store $2n$ numbers. This allows you to do the computation for **much** bigger binomial trees.*

5. Approximation error for American put options

This project is about the approximation error in the cost of a American put if a binomial tree with n steps is used.

For fixed values of $S = 48$, $t = 0.25$, $K = 50$, $\sigma = 0.32$ and $r = 0.1$, Let $P(n)$ be the price obtained from the binomial tree with n steps. The true option price is $P^* = \lim_{n\to\infty} P(n)$. I want you to understand how big $\Delta(n) = |P(n) - P^*|$ is.

Write a function to compute $P(n)$. You should do a reality check to make sure that it seems to be converging to a positive number.

The value of P^* is, of course, unknown. Obtain an estimate by computing $P(n)$ for a number of consecutive large values of n: N, $N+1, \ldots N+99$ say, and taking the average.

Now use this value to compute $\Delta(n)$ for a sequence of values of n (these values of n should be smaller than N and well spaced apart; powers of 2 up to $N/2$ for example).

You should expect to see something like $\Delta(n) \approx Cn^{-\alpha}$. The question is: what is that value of α?

Look up log–log plots or logarithmic regression and use these to estimate α.

Note: You can minimize the amount of storage you need. If you have n steps in your tree, then a 2-dimensional array $a_{k,i}$ has to store n^2 numbers. This is a major limitation on how big the values of n are that your program can compute.

However... Once you have computed the values in the binomial tree at the kth level, you don't need the values in the $(k+1)$st levels any more. What this means is that you can just use 1-dimensional arrays. Just keep track of the previous level and the current level. No other information is needed. Now you only have to store $2n$ numbers. This allows you to do the computation for much *bigger binomial trees.*

6. Pricing an increasing strike American call option

We know that it never pays to exercise a standard American call option early. In this project, you will price a call option where you can exercise at any time that you want, but the strike price increases over time: at time t, the call can be exercised, but the strike price is Ke^{bt}.

Give a method based on binomial trees for computing the cost of such an option (it should resemble what we did for American call options); and implement it in R (be careful not to store too much data!)

Compute an approximation to the cost of an increasing strike American call option if $s = 64$, $K = 64$, $r = 0.02$, $\sigma = 0.15$, $t = 0.5$ and $b = 0.05$.

Note: You can minimize the amount of storage you need. If you have n steps in your tree, then a 2-dimensional array $a_{k,i}$ has to store n^2 numbers. This is a major limitation on how big the values of n are that your program can compute.

However... Once you have computed the values in the binomial tree at the kth level, you don't need the values in the $(k+1)$st levels any more. What this means is that you can just use 1-dimensional arrays. Just keep track of the previous level and the current level. No other information is needed. Now you only have to store $2n$ numbers. This allows you to do the computation for much *bigger binomial trees.*