Pricing Options by the Black-Merton-Scholes Model

In order to create the Option Pricing code, I split the logic into several classes for neat code, reproducibility, debugging, and scalability. First, I created an Option class that initializes option parameters if none are passed or initializes the parameters if they are passed. Parameters include Strike price, Underlying asset price, risk-free-rate, time-to-maturity, and volatility. All parameter values are assumed to be annualized when appropriate. Then I created an abstract class called Pricing_Method. The Pricing_Method declares two pure virtual functions, BSM_Pricer – the Black-Scholes-Merton option pricing method – and Binomial_Pricer – the Binomial option pricing method. Because there exists at least one pure virtual function declared in the Pricing_Method class, Pricing_Method class is an abstract class. Thus, we have to override all pure virtual functions in the derived class. So, I created another class called Option_Price, which derives from both the Option class and the Pricing_Method class. In the derived Option_Price class, the two pricer functions are overridden. It also has public member variables flag and n_intervals, which denote the option type (call or put) and number of time intervals for the multi-period Binomial pricer respectively.

For the BSM pricer, I created a StdNormalCDF class that calculates the CDF using a standard normal distribution. I ensured that if inputs are larger than 6 or smaller than -6, the output would just be 1 or 0 respectively to prevent overflow. Then, I ensured that no input parameter was negative, else the output was -1 for the Option Value and -2 for Delta (since Delta is bounded by -1 and 1 inclusively). Then, I implemented the BSM formulas and returned the output. Another edge case that was taken care of was invalid inputs for Option Types. The formulas used followed the lab as well as the Wikipedia link attached in the PDF.

Similarly, for the binomial pricing method, I followed the pseudocode used in the Wikipedia link attached in the PDF, just translated it to C++. I defined the needed variables for the binomial pricer (up, down, and the respective risk-neutral probabilities) and implemented the formula on a vector the reiteratively updates, having the option value at the first node (or the first index in the vector). The same edge cases and invalid inputs were handled as that of the BSM pricer. I noticed that pricer is less accurate than the way we implemented binomial pricing in our notes in class. So, I created a 2D array, where i+1, j+1 denotes an upward move. Thus, I first calculated price at maturity then discounted back using risk-free-rate with respect to the risk-neutral probabilities.

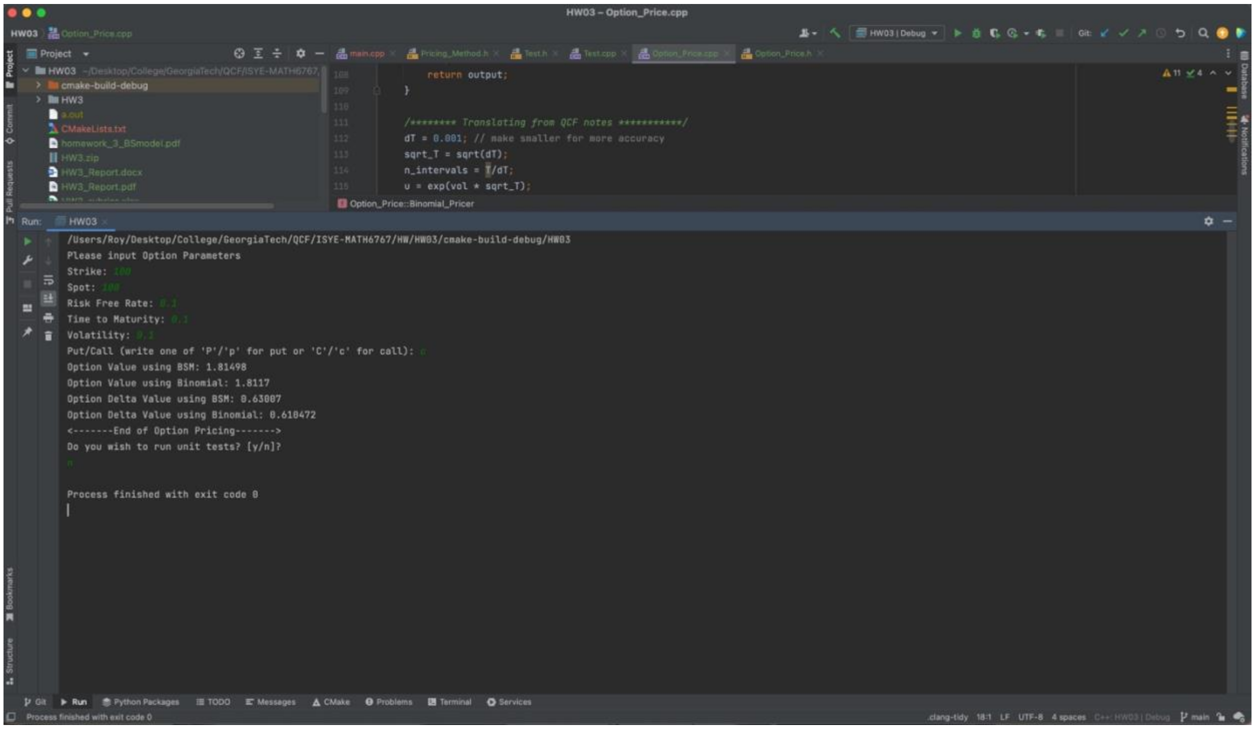
For both pricers, if time-to-maturity was zero, then the option value returned is max(S-K, 0) for call options and max(K-S, 0) for put options. This assumes the user would exercise the option if the option was ITM (otherwise the output would be a trivial solution with 0 always being the option value as the option expires worthless).

Lastly, unit test functions were implemented to test invalid inputs and correctly created constructors. When running the file, the user will be asked to enter y or n if they wish to run unit tests. Please see attached screenshots for details.

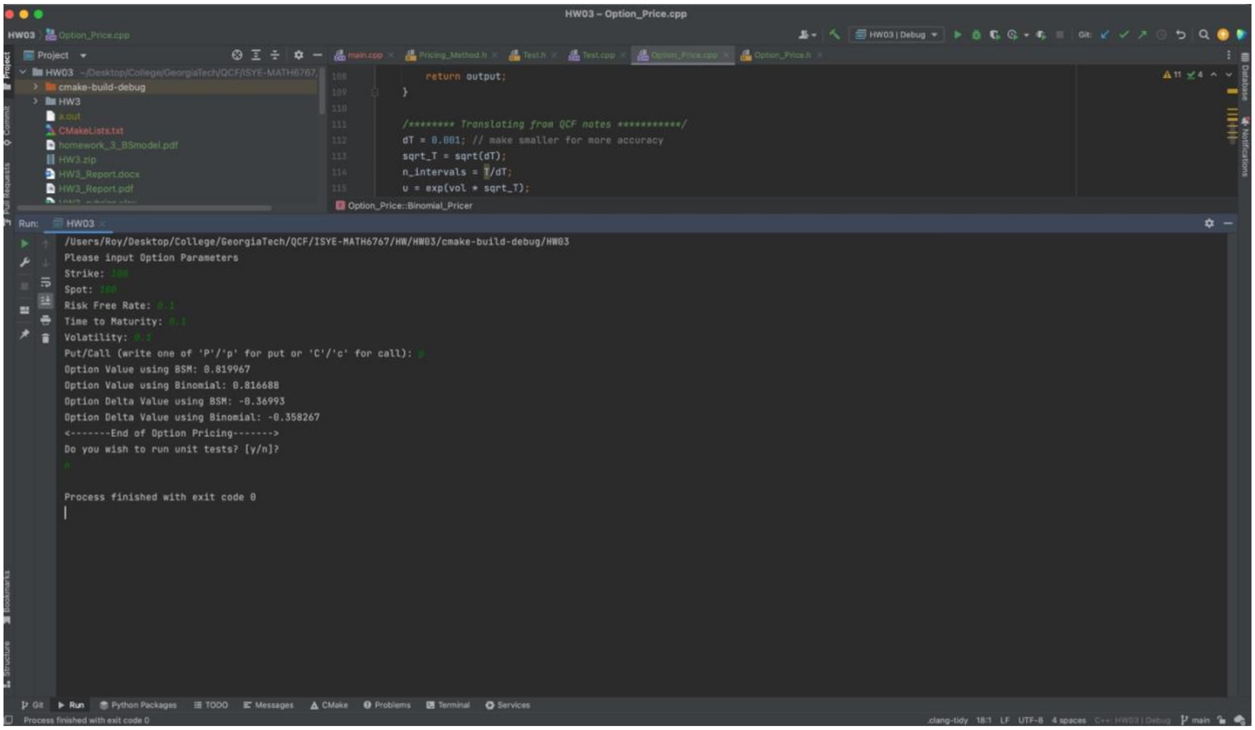
Result Analysis

For ATM options, the binomial pricer and the BSM pricer are very similar in both option value and delta. However, as we go further out of the money or with larger volatility, time-to-expiration, and/or risk-free-rate, the binomial pricing method tends to deviate slightly from the BSM pricer. This is expected since, to an extent, the Binomial pricer is discrete and not as continuous as the BSM pricer. This is due to the delta T value. The smaller delta T is, the more accurate the Binomial pricer. However, there is a limit on how accurate the pricer can get.

Output Screenshots:
The output for S = 100, K = 100, r = 10%, T = 0.1, vol = 10% (delta T value of 0.001)
Call:



Put:



Example using unit tests and not ATM option: