Solution to IMC Take Home

By Anjisht Gosain

# Part A : American Option Pricing

## Method Overview

The project requires the implementation of four different variants of the classical Black Scholes model, which I refer to as:

1. Black Scholes,
2. Black Scholes with Dividend,
3. Black Scholes with Single Normal Jump,
4. Black Scholes with Double Normal Jump.

The last two models can be considered as simplifications of the Merton Jump Diffusion model, as proposed by Merton (1976), and the Kou Jump Diffusion model, as proposed by Kou (2002), respectively. In the former, the Poisson process with Normal jumps has been replaced with a single Normal jump at a deterministic time; and in the latter, the Poisson process with Double Exponential jumps has been replaced with a single deterministic Double Normal jump.

I implemented tree methods to price American options under each of the above four models. For the Black Scholes diffusion part of the models, I implemented 2-state jumps, with probabilities under the Cox, Ross and Rubinstein (1979) and Tian (1993) models. For each Normal jump is used a 5-state moment matching scheme.

As the discrete deterministic dividend have no impact on the diffusion or jump processes, I first constructed the tree of underlying prices under the diffusion and jump parts, and then deducted the value of the divided from each node after the dividend payment date. I also enforced a floor of 0 on the underlying price, and made it an absorbing state.

## Method Justification

I decided to use trees because they can cope with path dependent options, as they run in backward time, and can achieve very fast pricing. Although Least Squares Monte Carlo methods, such as Longstaff Swartz, can also price path dependent options, they have a slower speed of pricing than trees.

An issue with tree methods is that they belong to the explicit family of finite difference schemes, and thus are not unconditionally stable. In other words, it is not guaranteed that the method converges to a price. However, these instances do not occur frequently. Moreover, unconditionally stable schemes such as ADI, or even Crank-Nicholson, are significantly more complex, and thus require greater code development time.

Since the option value is not differentiable at the Strike price, tree methods can at times yield an unstable price. Stability can be increased by ensuring that the Strike price always falls exactly on a node, or in the middle of a node. This was not implemented because doing so is not trivial in the presence of jumps.

## Black Scholes

Joshi (2007) and Chan et al. (2009) conclude that the classical Binomial tree method can be extended to have similar performance as the Trinomial tree method. These extensions form the basis of my implementation for the Black Scholes model, which in turn is used across the other three models as well.

I implemented the following extensions:

* Up and down state probabilities from Cox, Ross and Rubinstein (1979), and Tian (1993).
* Richardson Extrapolation : take the average of the option prices given by and time steps. This averages out the oscillatory behaviour of tree prices.
* Smoothing : at the last discrete time before expiry, an American option is simply a European option. Hence, the American option value at all tree nodes at this time point can be set to the greater of the instrinsic value and the closed form solution for the price of a European option under Black Scholes.

In the BlackScholesTest.h test harness file (in the provided code) I compared the performance of the different extensions by pricing a selection of European options. I found the inclusion of Richardson Extrapolation and Smoothing significantly improved the rate of convergence of the Binomial tree, especially compared to Monte Carlo. I found the use of the state probabilities from Tian (1993) to offer insignificant performance gains over those from Cox, Ross and Rubinstein (1979). Hence, for all my results I simply used the latter.

In particular, I selected the classical up and down jumps of , with the up state probability of

and chose the down state probability such that the two probabilities summed up to 1.

In the above equation is the time step size for the tree, with the other parameters as per usual. I kept the time step size constant, which results in the jump sizes and probabilities being invariant with time. The time step size can be selected at run-time such that the option price converges, and that any discrete events such as dividend payments or price jumps occur in between discretisation times.

After constructing the tree, the value of the American option is determined in backward time. The option value at each node is equal to the greater of the intrinsic value at that node, and the discounted average of the option value from the two forward nodes. This process continues up until the single node at time 0 is hit.

## Black Scholes with Dividend

This model is exactly the same as the previous Black Scholes model, except that at a pre-predetermined time, the underlying price drops by a pre-determined amount. As this drop does not impact the diffusion process, I constructed the tree for this model in two steps. First, I constructed the tree all the way up to option maturity based on the diffusion process. Then I deducted the dividend amount from all nodes after the dividend payment time.

## Black Scholes with Single Normal Jump

My method for modelling the Single Normal jump was inspired by that from Hilliard and Schwartz (2005). I discretised the Normal jump into 5 states via moment matching the first four non-central normal distribution moments and requiring the probabilities to sum to 1.

The 5 jump states I selected were , , , , . As the normal distribution is symmetric, the moment matching procedure yielded probabilities of 1/12, 1/6, 1/2, 1/6 and 1/12 respectively.

I constructed a recombining Binomial tree up until the discrete time immediately before the jump. Then, I applied ten jump diffusion nodes to each node at this time. Each jump diffusion node involves a diffusion up or down state, and one of the 5 jump states. From each of these jump diffusion nodes, I then constructed a new non-recombining Binomial tree up until option maturity.

Although I also implemented a recombining version, I found it to produce unstable results, thus requiring further investigation. The deprecated implementation is nonetheless available in the code.

The dividend was handled in the same way as the previous model.

## Black Scholes with Double Normal Jump

Originally I was requested to implement the following dynamics:

where for , and a Bernoulli variable with probability . That is, the jump is Double Normal. This is the model which I implemented in my code.

Luckily, the model in the revised take-home is a simplification of the above model. We set and The rest is exactly is same.

Now, the Double Normal jump in the fourth model will result in a bimodal jump, depending on the distance between the mean of the two Normals. However, as the two Normal are independent, the 5-state moment matching can be performed independently for each Normal. This yields 10 jump states for the Double Normal. The probability for one of the 10 Double Normal jump states is then the Bernoulli probability of hitting the corresponding Normal, multiplied by the probability of hitting the corresponding state of that Normal. This results in a total of 20 jump diffusion states.

All else was done in the same way as the previous model.

## Validation

To validate my tree construction, I implemented Monte Carlo pricers for each of the four models, as well the analytic European option pricing formula for the Black Scholes model. I constructed a variety of tests, using predominantly European options, to check event timing and the validity of the distributions implied by the trees. The whole suite of tests is included in my C++ implementation.

# Part B : Optimisation

## Method Overview

I implemented a Differential Evolution optimiser to solve for the model parameters. I selected Differential evolution because:

1. it returns the global minimum;
2. it optimises in multidimensional real space, and so it can optimise for the five parameters of interest;
3. and, it does not rely on gradients (i.e. the Greeks in our case).

## Method Justification

I considered Levenberg-Marquardt as an alternative to Differential Evolution, but decided against using that algorithm because it finds local minimum (although with some tweaks its chance of finding the global minimum can be improved) and will be dependent on the Greeks. Given that the implementation of the Greeks in the tree pricers would require further study (particularly for stability on the tree), I felt that Differential Evolution was the more appropriate choice.

## Method Details

The aim of a Differential Evolution algorithm is to find the global minimum of a function which maps a dimensional real space into a dimensional real space. That is, the algorithm finds the values of the parameters which minimise the function.

The main idea is to begin with a random selection within the space, and then iteratively adjust the random selection until the minimum is found. In particular, the algorithm is composed of 5 stages:

1. Initialisation : make a random selection (within upper and lower bounds) of points in the dimentional space. These points are referred to as the *target* *vectors*, and can be denoted as for in *.*
2. Mutation : For each one of the target vectors , randomly select three other target vectors , and for . Compute the *donor vector* , where the *mutation factor* is a constant selected between 0 and 2, inclusive.
3. Recombination : construct the *trial vector* as a random mixture of the target and donor vectors. In particular

where is a standard uniform random variable, is a random integer form ; and is the *selection criterion.*

1. Selection : Value the function at the *trial* and *target* vectors, and select the one which has a lower value. The selected vector is then the target vector for the next iteration of the algorithm.
2. Repeat the above four steps until the global minimum is found within the desired tolerance.

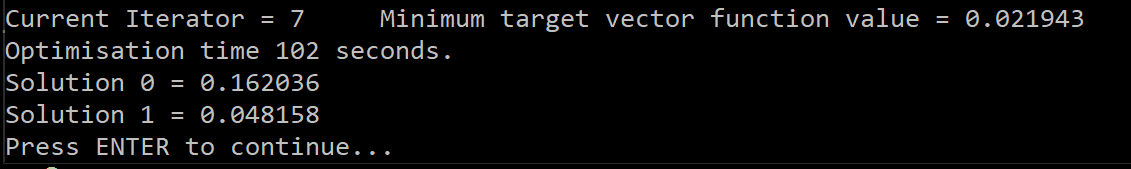
An issue with the Differential Evolution algorithm is premature convergence, whereby the algorithm converges to a local rather than global minimum. This can be overcome by increasing the population size of the target vectors, at the expense of speed, or by making a better selection of the initial target vectors. One way to achieve the latter is by using quasi-random points, from say a Sobol sequence, rather than psuedo random points. For the present implementation, pseudo random points were used.

## Solution

The pricing results for Part B5 can be found in the file B5.json. These results were generated with 7 time steps in the jump diffusion tree (i.e. the price is an average from 14 and 7 time steps), with a total run time of only a few seconds. Prices were observed to converge reasonably well with even 4 time steps, with the whole set of options being priced in less than a second.

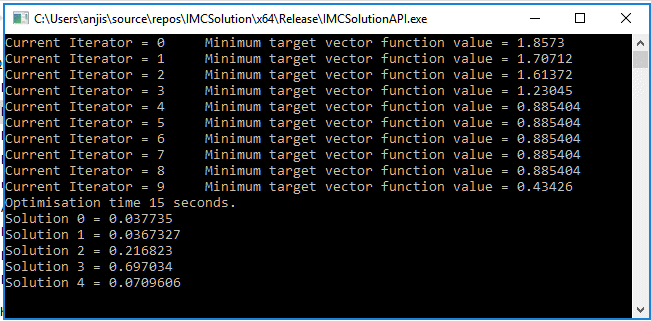
I completed part B6 in two stages. First, I set the number of time steps in the jump diffusion tree to 5 for each of the options, along with , , and a lower bound of 0.01% and an upper bound of for both of the two volatilities. I minimised the mean squared error across the option prices to get values of approximately and .

I then restarted the algorithm with 4 times steps in the jump diffusion tree, , and bound between and 20%, and bound between 1% and 10%. The results after 7 iterations of the algorithm were as below.



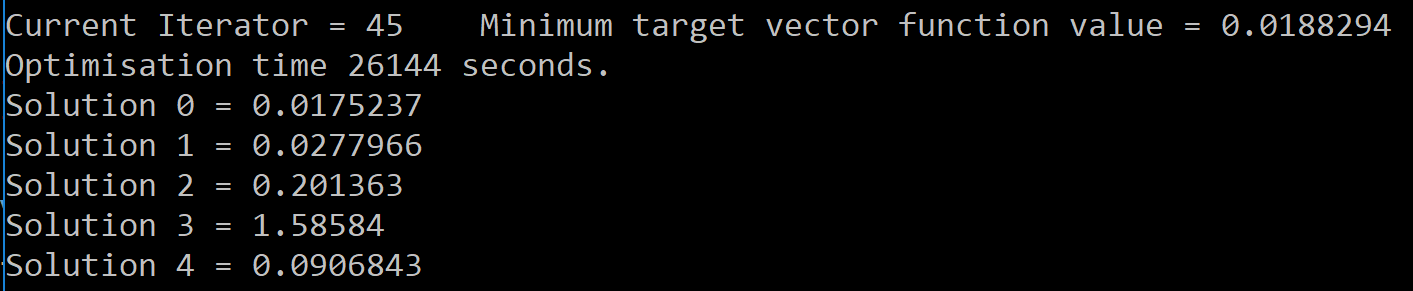
That is, and , with a mean square error of 0.021943 across the prices of the provided options.

For part B7, I first chose 4 times steps in the jump diffusion tree, , and boundaries of The results after 9 iterations of the algorithm were as below.



I then restarted the algorithm with 4 times steps in the jump diffusion tree, , and boundaries of The results after 46 iterations of the algorithm were and , with a mean squared error of 0.033.

Observing that the dividend amount was close to its upper bound of , I restarted the algorithm one final time with the same parameters as above, except with , and . The results after 45 iterations (an overnight run) of the algorithm were and , with a mean squared error of , as per below.



Running the algorithms for a longer period of time would result in further reduction of the mean squared error.

## Validation

I validated my Differential Evolution implementation by minimising the Ackley function, which is a multi-dimensional function that has a large number of local minimums, but only one global minimum.

# Solution C++ Implementation

## Projects/Namespaces

I have implemented my solution in C++11, in the Visual Studio solution IMCSolution.sln. Although the solution is designed to compiled on Windows, the code should be portable to Linux.

Other than the C++11 Standard Template Library, the only other package I have used was Boost.

The solution contains the following Visual Studio Projects/C++ namespaces:

* IMCSolutionAPI : contains the individual function APIs requested in the questions. Each API is a wrapper around classes and functions in the other projects.
* Instruments : contains the derivative payoffs, which in the present case is simply vanilla options.
* Models : the different models required in the solution, such as the Black Scholes, Black Scholes with Normal Jump etc. These classes describe the dynamics of the underlying asset price, and simulate values for use by the Pricers.
* Pricers : these are the model implementations used to price the derivatives. For example, the namespace contains tree, Monte Carlo, and analytic pricing methods for vanilla options. Pricers and Models are linked together via the Delegate design pattern. Each Pricer is set up to work with a unique pure virtual class of model, which is then inherited by the actual models in the Models project.
* Optimisers : contains the implementation of the Differential Evolution optimiser. The optimiser requires input of a function object, parameter constraints, and convergence criterion. It returns the parameters which minimise the function. The optimiser has been designed to minimise any continuous function, not just the ones from the Pricers project.
* TestHarness : this project contains a simple test harness and tests for the validation of the tree Pricer and Differential Evolution optimiser.
* Enumerations : avoids the use of strings for the identification of commonly used option features such as exercise type, option right, underlying asset codes etc.

All random generation was done using the Standard Template Library implementation of the Mersenne Twister pseudo random generation algorithm.

## Compile Instructions

The solution has been set up to compile on Windows. I can generate the .makefiles required for running on Unix.

My results were generated using Visual Studio SDK 10.0.16299.0 and Boost 1.66. Assuming these are already installed, the steps to run the solution are as follows:

1. Open the “IMCSolution” Visual Studio Solution in Visual Studio.
2. From Solution Explorer, right click on “IMCSolutionAPI”, and select “Set as Start Up Project”.
3. Under Build, select Configuration Manager, and set the Active solution configuration to Release, and select x64 as the platform. This compiles the solution with compiler optimisation (equivalent to -O2 command line flag) in a 64-bit environment.
4. Hit F5. This will open a command prompt displaying the outputs.

## Boost Installation

The steps to install Boost are as follows:

1. Download boost\_1\_66\_0.zip from "http://www.boost.org/users/history/version\_1\_66\_0.html
2. Unzip the downloaded .zip file in C:\Program Files\boost\

All of the Visual Studio projects which are dependent on boost already have C:\Program Files\boost\boost\_1\_66\_0\ as an Include Directory.

# References

Chan, J. H., Joshi, M., Tang, R. and Yang, C. (2009), Trinomial or binomial: Accelerating American put option price on trees. J. Fut. Mark., 29: 826–839. doi:10.1002/fut.20389

Cox, J. C.; Ross, S. A.; Rubinstein, M. (1979). "Option pricing: A simplified approach". Journal of Financial Economics. 7 (3): 229. doi:10.1016/0304-405X(79)90015-1.

Hilliard, Jimmy E., and Adam Schwartz. “Pricing European and American Derivatives under a Jump-Diffusion Process: A Bivariate Tree Approach.” The Journal of Financial and Quantitative Analysis, vol. 40, no. 3, 2005, pp. 671–691. JSTOR, JSTOR, www.jstor.org/stable/27647216.

Kou, S. G. (2002). A jump diffusion model for option pricing. Management Science 48, 1086-1101.

Joshi, Mark S., The Convergence of Binomial Trees for Pricing the American Put (November 14, 2007). Available at SSRN: https://ssrn.com/abstract=1030143 or <http://dx.doi.org/10.2139/ssrn.1030143>

Merton, R. C., 1976, “Option Pricing When Underlying Stock Returns Are Continuous,” Journal of Financial Economics, 3, 125-144.

Tian, Y. (1993), A modified lattice approach to option pricing. J. Fut. Mark., 13: 563–577. doi:10.1002/fut.3990130509