Option Pricing C++

Ryan Sephton

May 7, 2020

Contents

1 Introduction

In each section I will briefly outline the theory behind each pricing method, offer a suitable reference and then talk in some-detail about each method I used to price the particular options. Firstly, I will discuss the structure of the program and certain design features that I opted for.

2 Structure

The structure of my program can be seen in figure 1, I have opted for three main class hierarchies, each of which have multiple layers of inheritance.

2.1 Option Class Family

The option class family is the central point to the project. The option class is an abstract base class with several virtual pricing functions and stores various important properties of the option. The child classes to this class are the vanilla option and asian option classes, these are both in turn abstract classes. The asian option class then has child classes that are the asian call and the asian put classes, in reality the parent class would be an exotic options class, though as we only consider asian options, this step has been omitted. The vanilla option class has two child classes, the american and the european vanilla option classes, again these are abstract. Finally the children of these classes, the respective calls and puts are no longer abstract and implement several pricing schemes that will be discussed later.

This class family has intermittent dependence on the other two families, though straight composition has not been used, as storing the binomial trees or monte carlo pricing objects can have a large overhead in cases, and would mean that each class has very distinct headers that could lead to unnecessary spaghettification. Instead RAII has been followed and only local scope objects have been used, typically in the lowest layers of inheritance.

2.2 Binomial Tree Family

The binomial tree family has CRR Tree as the parent class, which simply includes various parameters of the binomial tree alongside a set of virtual methods. These virtual methods are implemented in the three child classes. The first child class is the early exercise tree, which is called by the american vanilla options. This class is templated to allow full generalisation and minimal code reproduction when it comes to puts and calls. The class stores a pointer to an instance of the option and then infers the payoffs that are needed for the tree. I opted to not implement the payoff function as a functor in my code, though I know this was the status quo in the lecture course. I believe a properly templated class makes more physical sense than options being composed of a payoff class, this is also stylistically nicer as when the whole option instance is passed in, the payoff, strikes and undelying prices can be inferred via getter functions.

The second child class is the maturity only CRR tree which is used by the european options, this class is not templated as the payoffs at maturity are passed in as a parameter, and unlike the american equivalents, these payoffs are not needed again. The last child class is the path dependent tree, which is a templated class with a significant calculation under the bonnet. This class implements the hull-white algorithm for binomially pricing path dependent options, and will be discussed in detail below.

2.3 The Monte Carlo Family

This family implements the monte carlo pricing method for the various options. The base class is the monte carlo pricer, and this is an abstract class which stores several important parameters that are used in the child classes. The first child class is the vanilla monte carlo pricer, which has almost been identically implemented in the lecture course, hence I shan't discuss it here. The second child class is the path dependent monte carlo pricer and this is only a slight modification from the vanilla pricer, albeit for a pre-requisite step in which brownian motion paths are generated for the option, rather than single measure 0 prices.

There is no dependencies between the monte carlo and binomial families.

2.4 The PDE Family?

I decided to not implement the PDE solvers as a separate hierarchy, mainly because there is little overlap between each implementation. As discussed below, I have implemented 2 different implicit algorithms, and one explicit algorithm, hence the class would just be 3 independent classes and would need the heavy overhead of templating again. If I had more time and implemented a PDE solver for the asian options, then I believe a third class family would be sensible, however for now, it would be overkill.

3 European Options

Most of the theory behind the pricing methods for the european options has been discussed in detail during the lecture course, however I will briefly recap some of the theory here.

3.1 Analytic Price

European option prices have exact closed form solutions, which are easily seen via a change of variables. The necessary change of variables can be written as follows.

$$\begin{aligned} S &= Ee^x, & (1)\\ t &= T - \frac{2\tau}{\sigma^2}, & (2)\\ k &= \frac{2\tau}{\sigma^2}, & (3)\\ V &= Eu(x,\tau)\exp\left[-\frac{(k+1)x}{2} - \frac{(k+1)^2\tau}{4}\right], & (4)\end{aligned}$$

where all of the symbols have their usual meanings. Using this transformation of variables, reduces the usual Black-Scholes equation to a diffusion equation of the form,

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}. \tag{5}$$

Solving this equation analytically, produces an answer in terms of a set of variables which are defined as follows,

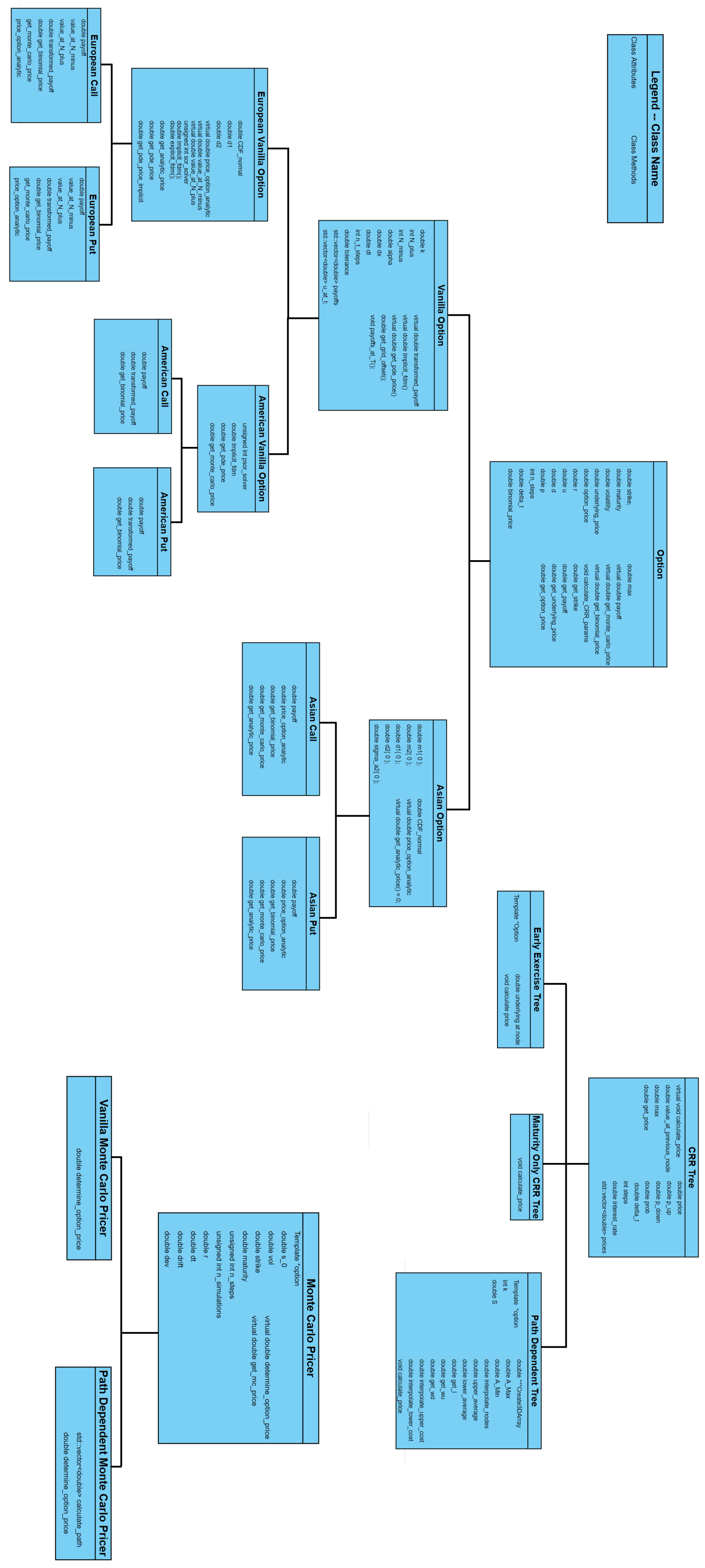
$$\begin{aligned} N(x) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{y^2}{2}}\,dy, & (6)\\ d_1 &= \frac{\log(\frac{S}{E}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}, & (7)\\ d_2 &= \frac{\log(\frac{S}{E}) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}. & (8)\end{aligned}$$

We can then express the fair value of both a call and put european option as follows,

$$\begin{aligned} V_{\text{call}} &= SN(d_1) - Ee^{-r(T-t)}N(d_2), & (9)\\ V_{\text{put}} &= Ee^{-r(T-t)}N(-d_2) - SN(-d_1). & (10)\end{aligned}$$

This has been implemented in my code and is the default pricing method for the european options due to the very fast computation time which is $\mathcal{O}$(constant).

Figure 1: A graphical representation of my program structure. There are 3 main class families. The first being the option family, which encompasses asian, european, and american options. The second family is the monte carlo pricing family, and the last family is the binomial tree family.



3.2 Monte Carlo Price

For european options, the monte carlo pricing method is almost trivial to implement. We start from the efficient market hypothesis and assume that the underlying price s(t + dt) can be expressed in terms of s(t) in the following way,

$$s(t+dt) = s(t)\left(rdt + \sigma dX\right), \tag{11}$$

and now one may invoke ito's lemma to express this more simply as,

$$s(t) = s(0)\exp\left[(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}W\right], \tag{12}$$

$$\text{where} \quad W \approx \mathcal{N}(0,1). \tag{13}$$

This can then be discounted using the appropriate discount factor. This method has been implemented in my code, however the variance that is used in the monte carlo algorithm is actually a biased statistical estimator, which means that the value the algorithm converges to is offset from the true value by a constant. To combat this, I have used antithetic variates to unbias the estimator. This is simple to employ, as I just have a second set of random paths that offset the bias, i.e.

$$\bar{s}(t) = s(0)\exp\left[(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}(-W)\right]. \tag{14}$$

Without antithetic variates, the fair price for the options can then be calculated using the payoff function for the european calls and puts Λ directly i.e.,

$$V = \frac{1}{N_{\text{simulations}}}\sum_{i=0}^{i=N}\Lambda_i(s(T)_i, K). \tag{15}$$

When antithetic variates are used, the option value becomes,

$$V = \frac{1}{N_{\text{simulations}}}\sum_{i=0}^{i=N/2}(\Lambda_i(S(T)_i, K) + \Lambda_i(\bar{s}(T)_i, K), \tag{16}$$

and in general, this produces a much more accurate estimate of the option value. As stated below, point estimates of monte carlo pricing have time complexity $\mathcal{O}(N^3)$ and convergence rate of $\approx 1/\sqrt{N}$.

3.3 PDE Price

I chose to implement two different PDE methods for pricing the european options. Firstly, I implemented the explicit finite difference method that was specified in the lecture notes, however this method is inherently unstable despite it's speed. The second method I implemented was an implicit finite difference method based on the method of successive over-relaxation. I will now give a very brief overview of the ideas.

We start by discretising our parameter space. By (5) this requires a 2-dimensional grid that extends in the x and τ direction. To do this, we use a simple set of heuristics, where we take $dx \approx 0.05$ and introduce the notation $\alpha = d\tau/(dx)^2$. We choose $d\tau$ such that $\alpha <= 0.5$ for reasons which will become clear below. In my program, I have constrained α to be 0.25, but this requires small steps to be taken in the chronological direction, which can make the algorithm sluggish. Larger values of α can be used with the implicit method, however, the computational overhead here offsets any saving in taking larger time steps.

By using the finite difference approximations, we transform (5) into,

$$u_n^{m+1} = \alpha u_{n+1}^m + (1-2\alpha)u_n^m + \alpha u_{n-1}^m, \tag{17}$$

Table 1: Comparison between pricing a european call with parameters: σ=0.4, r = 0.04, K=90, S=100, T=1, N^+=1000, N^-=-1000. Implicit method was the method of successive over relaxation, whilst the explicit method was a standard finite difference method.

α (dt/(dx^2))	Implicit PDE Price	Explicit PDE Price	Implicit Runtime (ms)	Explicit Runtime (ms)
0.1	22.4443	22.4536	22836	1426
0.3	22.3739	22.4019	17896	506
0.5	22.4258	22.4300	12236	278
0.7	22.2633	1.72x10^{10}	8173	198
0.9	22.254	2.20x10^{13}	9307	174

where we have introduced the notation $u_n^m = u(ndx, md\tau)$. To start the explicit finite difference method, we have to specify two boundary conditions for the limits,

$$\lim_{x \to \pm\infty} u(x,\tau) \tag{18}$$

which are easily determined by transforming the usual payoff limits. In reality, the infinite limit is intractable, hence instead a sufficiently large spatial and temporal extent is used s.t. $N^+dx(dt) >>> 0$ and likewise $N^-dx(dt) <<< 0$.

To overcome the instability of the explicit algorithm, the method of successive over relaxation has also been implemented in my code, which can be used with any value of α. The specifics of this method can be found in[1] but briefly, the method of successive over relaxation amounts to making a set of reasonable guesses as to infer the k-th iterate of u i.e. $u_n^{m,k+1}$ by minimising the euclidean distance between the previous iterates value. This can then be back substituted into the matrix equation that specifies (17). A toy set of iterations for varying α is summarised in table 1, outlining the relative merit of each method.

The final point to be made about the PDE methods is that the finite mesh that is created, may not involve the true value of the underlying, lying at a mesh point. To overcome this limitation a simple calculation is used to determine the mesh offset that is needed to shift the underlying value such that it lies on a grid point. This can be done directly, using the diffusion transformation,

$$S = Ee^x, \tag{19}$$

$$\therefore S = Ee^{(n_{\text{nearest}}+\delta)dx}, \tag{20}$$

$$\therefore \delta = \frac{1}{dx}\log\left(\frac{S}{E}\right) - n_{\text{nearest}}. \tag{21}$$

In my code, I have naively taken $N^+ = -N^- = 1000$ and I have linearly spaced the mesh in a regular fashion. A better approach would be to use regular logarithmic steps, or a naive adaptive algorithm that concentrates a greater coarseness to the mesh around the evaluation point corresponding to the underlying price![2]

3.4 Binomial Price

This was implemented in problem sheet 3 project 1, hence I will not dredge up old content here, however I will discuss important variations at length below.

One thing to note is that a full non-recombining binomial tree suffers from combinatorial explosion, as it suffers from time complexity $\mathcal{O}(2^N)$, whereas for european options on a recombining tree, the time complexity

[1]Wilmott, P., Howson, S., Howison, S. and Dewynne, J., 1995. The mathematics of financial derivatives: a student introduction. Cambridge university press.

[2]This is something my research supervisor, Alex Bespalov, is an expert in.

is $\mathcal{O}(N^2)$. Some recent research papers have managed to streamline the induction step into the creation step, which gets the complexity down to $\mathcal{O}(N)$.

4 American Options

American options pose significant new challenges that are not seen with european options, this is because american options offer the holder the right to exercise the option at any time t≤T. This constraint imposes a free boundary, which straight away makes an analytic closed form solution for the option price intractable. Despite this problem, computational approximations are very accurate, and here I have implemented two pricing methods: the binomial pricing method and the pde pricing method. Had I not suffered at the mercy of the hull-white algorithm (see below) I would have implemented a monte carlo method for pricing them, which follows the Longstaff-Schwarz algorithm [3]. The default pricing method implemented here is the binomial tree, which given the time complexity of $\mathcal{O}(2^N)$ where N is the node number, seems bizarre, however, as mentioned above, this tree is also recombining whereas the projected SOR algorithm is claimed to be $\mathcal{O}(N^3)$, hence the PDE method is an order of N slower than the binomial method, which becomes very noticeable at run time.

4.1 Binomial Price

Again, this will not be discussed in too much detail. The specifics are the same as the european binomial trees, except that at each node the payoff function must be evaluated, as it may be in the holder's best interest to exercise the option early.

In terms of implementation, whilst this may seem a simple modification, the class that handles such a method must be far more sophisticated than that of the european binomial trees. Specifically there are two reasonable methods to implementing this class, the first being to use a payoff functor, wherein the american options would be composed of a payoff object internally. As mentioned above, this is what was implemented in the course, but is non-ideal for several reasons. Firstly, it sets a precedent for the code, that almost every option parameter may as well be implemented as a functor and then we would have a class zoo on our hands. Secondly, it is rather lazy to use a functor rather than elegantly making a class templated and type robust. The second method, is the one I chose to implement which was to template the class and allow it to store an instance of the option, from this the payoff and underlying option parameters can be inferred.

The rest of the algorithm is rather trivial and simply involves calculating the payoff of the form

$$(V_m^n)_\pm = \max(S_m^n - E, C_m^n)_\pm. \tag{22}$$

Where C^n denotes the aggregated value of the of the node, defined as $C_m^n = pV_{m+1}^n + (1-p)V_m^n$.

4.2 PDE Price

The PDE method used for pricing the american options is the projected successive over relaxation method, which is an implicit finite difference method that extends the method used by the european vanilla options, discussed above. Whilst the free boundary makes the problem seem intractable for PDE solvers to work, one can express the system of equations and constraints in linear complimentary form, which allows the free boundary to be tracked by another variable and this can then be constrained in a closed sense during the

[3]Bruno, G., 2010. Monte Carlo Simulation for Pricing European and American Basket option. In The R User Conference 2010, National Institute of Standards and Technology.

algorithm. A more thorough discussion can be found in [4] (shock!) but briefly, the problem can be expressed as,

$$\left(\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}\right) \geq 0, \tag{23}$$

$$(u(x,\tau) - g(x,\tau)) \geq 0, \tag{24}$$

$$\left(\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}\right)(u(x,\tau) - g(x,\tau)) = 0. \tag{25}$$

Which now fixes the boundary conditions to be,

$$u(x,0) = g(x,0), \tag{26}$$

$$u(x,\tau) \quad \text{is continuous}, \tag{27}$$

$$\frac{\partial u}{\partial x} \quad \text{is as continuous as} \;\; g(x,\tau), \tag{28}$$

$$\lim_{x\to\pm\infty} u(x,\tau) = \lim_{x\to\pm\infty} g(x,\tau). \tag{29}$$

This can then be used to generalise the successive over relaxation scheme to solve the american option pricing problem.

5 Asian Options

About 30% of my time was spent on these pricing methods as little solid literature exists about implementing the various sub-routines. Asian options are heavily path dependent, consequently any pricing method comes with significant overhead due to the running average having to be tracked. PDE methods do exist, though I did not implement one, mainly because they are extremely slugglish. One can implement an explicit fdm method as with european options, however because we need to track the running average, the grid becomes 3-dimensional, and now the time complexity starts to runaway. If the time complexity doesn't bite, then the instability will, due to the curse of dimensionality pockets of instability in the phase space exist, which make this a messy business altogether. Implicit methods circumvent the instability but carry an even larger overhead in terms of runtime, hence I avoided implementing these methods altogether.

Geometric average rate asian option can actually be solved analytically, unfortunately the arithmetic counterparts cannot. Despite this, a beautiful approximation exists, thanks to Turnbull and Wakeman [5] which circumvents the very problem that makes an exact analytic solution intractable.

The fundamental problem with arithmetic average rate options is that the sum of n lognormally distributed quantities, is itself, in general, not a lognormally distributed quantity, however, I have it on good authority, that one can find an equivalent lognormally distributed set of parameters that sufficiently describe the resultant quantity to within an arbitrary error range. This is done using an edgeworth expansion (details of which I do not understand) but essentially boil down to using a set of successive moments of the underlying distribution to iteratively mimick the lognormal distribution.

I have implemented this approximate analytic solution using the first three moments of the edgeworth expansion and this gives typically, an error of order 10^{-3} with published data. This error certainly holds its own versus the likes of monte carlo methods which scales its error as $1/\sqrt{n}$, implying that 10^6 simulations are

[4]Wilmott, P., Howson, S., Howison, S. and Dewynne, J., 1995. The mathematics of financial derivatives: a student introduction. Cambridge university press.

[5]Turnbull, S.M. and Wakeman, L.M., 1991. A quick algorithm for pricing European average options. Journal of financial and quantitative analysis, 26(3), pp.377-389.

needed to be comparable, which is very sizeable given that the time complexity is significantly larger than $\mathcal{O}(N^3)$ due to the path dependent nature requiring T more iterations per simulation compared to european options. This aside the biasing of the estimator will still incur errors larger than this. The closed form approximation carries constant time complexity, hence in light of all of this, I have used the approximate analytical solution as my default pricing method for asian options.

5.1 Analytic Price

Skipping the reiteration of the above, the approximate solution can be expressed as,

$$m_1 = \frac{e^{rT} - 1}{rT} S, \tag{30}$$

$$m_2 = \frac{2e^{2r\sigma^2 T} S^2}{(r+\sigma^2)(2r) + \sigma^2 T^2} + \frac{2S^2}{rT^2}\left[\frac{1}{(2r\sigma^2 - e^{rT})} - \frac{e^{rT}}{r+\sigma^2}\right], \tag{31}$$

$$\sigma_a^2 = \frac{\log(\frac{m_2}{m_1^2})}{T}, \tag{32}$$

$$d_1 = \frac{\log(\frac{m_1}{K}) + \frac{\sigma_a^2 T}{2}}{\sqrt{\sigma_a^2 T}}, \tag{33}$$

$$d_2 = d_1 - \sqrt{\sigma_a^2 T}. \tag{34}$$

and then d_1 and d_2 can be used with (9) and (10).

5.2 Monte Carlo Price

This has been discussed extensively already, there is nothing different here except that the paths are given by sums of (14) and then averaged. The code also describes this in more detail. Again the estimator is heavily biased hence, antithetic variates are used once more.

5.3 Binomial Price

This was an extensive piece of work that implements the hull white algorithm. As the option is path dependent, the typical strategy for back integrating all of the paths on the tree causes combinatorial explosion due to the paths not recombining (this is because there is $\binom{n}{m}$ paths to any node of tree layer n, at position m. Hull and White came up with a genius work around, which was to give a tree a third index m which tracks the individual running average at the node, and rather than track this explicitly we just find the highest running average and smallest running average and then linearly interpolate between them. One can then determine the running average from the previous node and find a bounding value of m, which we denote as l, which specifies the position on the linearly interpolated line. These interpolations can then be done on the cost as well.

Mathematically, this can be written as follows. We start by denoting the running average as $A(i,j)$, where (i,j) denotes an individual node on the tree. The maximum and minimum running average at a node can be expressed as,

$$A_{\max}(i,j) = S_0(1 + u + u^2 + \cdots + u^{i-j} + u^{i-j}d + \cdots + u^{i-j}d^j)/(i+1), \tag{35}$$

$$\therefore A_{\max}(i,j) = S_0 \frac{1 + u\frac{1-u^{i-j}}{1-u} + u^{i-j}d\frac{1-d^j}{1-d}}{i+1}. \tag{36}$$

Using similar logic, we can express the minimum running average as,

$$A_{\min}(i,j) = S_0 \frac{1 + d\frac{1-d^j}{1-d} + d^j u\frac{1-u^{i-j}}{1-u}}{i+1}. \tag{37}$$

Any intermediate average can now be expressed as a point on the linear line that joins the two averages, i.e.

$$A(i,j,m) = \frac{M-m}{M} A_{\max}(i,j) + \frac{m}{M} A_{\min}(i,j), \tag{38}$$

where several heuristics exist for determining M. One such heuristic, used by Hull and White themselves, was to propose that the value of the underlying at a node, took the form $S(i,j) = S(0,0)e^{mh}$, where h is a grid spacing parameter that is set by the user (typical values ≈ 0.01). As the simplifying assumption that was made in this project is that the arithmetic average is taken once per day, using the hull white approximation can make a very intense memory requirement, hence instead I have opted to constrain $M \approx 300$.

After this, the upper running average can be determined using,

$$A_u = \frac{(i+1)A(i,j,k) + S_0 u^{i+1-j} d^j}{i+2}. \tag{39}$$

We wish to bound the average in the interval [A(i+1,j,l), A(i+1,j,l-1)], and doing so means the tree can now be back integrated as,

$$C_u = w_u C(i+1,j,l) + (1-w_u)C(i+1,j,l-1), \tag{40}$$

$$\text{Where} \quad w_u = \frac{A(i+1,j,l-1) - A_u}{A(i+1,j,l-1) - A(i+1,j,l)}. \tag{41}$$

The same logic can be applied for the downwards variants, from which we can determine C_d.

After this, we can aggregate the cost as usual using,

$$C(i,j,k) = \left[pC_u + (1-p)C_d\right] e^{-rdt}, \tag{42}$$

and back integrate, until we calculate the cost at t=0.

The whole algorithm isn't particularly quick, though faster than the monte carlo algorithm, however it is interesting to note that most of the pseudo code and outlines of the algorithm in the literature are extremely poor and sometimes even wrong. The only helpful pseudo code I could find to assist me was a set textbook from a lecturer in a chinese tapei university, hence I will leave this as a suitable reference to prevent this write up from turning into a diatribe.

The performance of the algorithm, I have been told is $\mathcal{O}(N^3)$ which is significantly better than 2^N. As for the estimates, they are very fair, typically they are accurate to 10^{-2} or more. There are a few refinements that can be made to this algorithm such as using quadratic rather than linear interpolants, and dynamically tuning the value of M depending on the node. The error grows as $1/M$ so these adjustments would create huge improvements, however I spent 5 days implementing the algorithm, so a day longer would have required a huge amount of pain[6].

6 Specific Tests

For the specific tests, all of the data is contained in the accompanying excel file, complete with the 8 required graphs (on sheet 2).

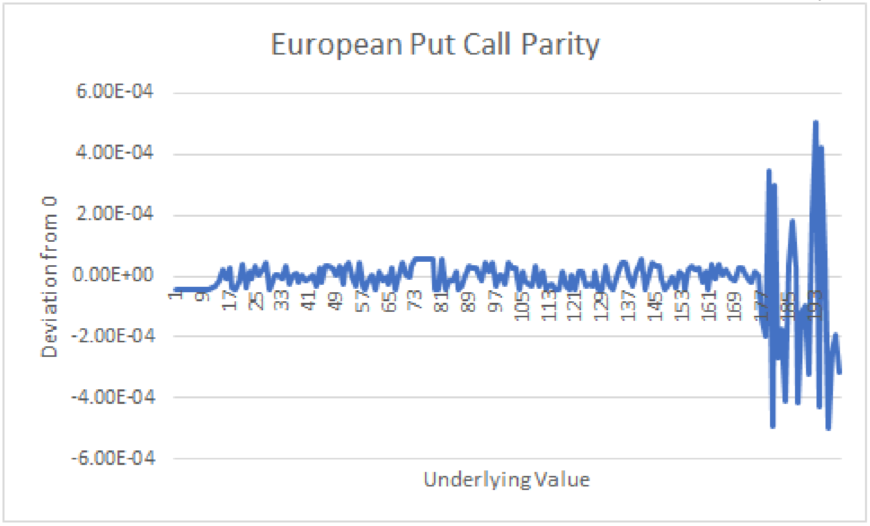
6.1 European Put-Call Parity

Put-call parity is obeyed reasonably well, as expected for the european options, and this was discussed in problem sheet 3, but is given by,

$$C(t) - P(t) = S(t) - Ke^{-r(T-t)}. \tag{43}$$

[6]Lyuu, Y.D., 2001. Financial engineering and computation: principles, mathematics, algorithms. Cambridge University Press.

Figure 2: A graphical representation of the deviation from 0, for the put-call parity of the european options.



Where C and P denote the respective calls and puts. A representation for the agreement can be seen in figure 2, clearly large departures from 0 occur as the underlying price starts to approach infinity.

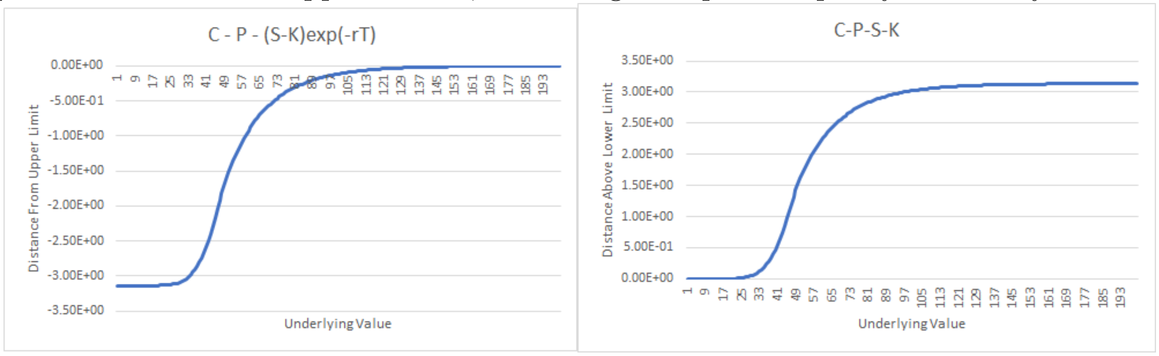
6.2 American Put-Call Parity

American Put-Call Parity can be expressed as,

$$S - K \leq C - P \leq S - Ke^{-r(t-t)}. \tag{44}$$

This was also found to hold true to reasonable accuracy, and can be found on the third sheet of the excel

Figure 3: The relative distance from the upper and lower bounding inequality in (36). The distance asymptotically approaches 0 but never approaches it, indicating that put-call parity holds very well.



document, as the grey line is bounded by the other two. Figure 3 illustrates the agreement, wherein the value of C_P is plotted against both sides of the inequality in (36), and sigmoid patterns can be seen in both.

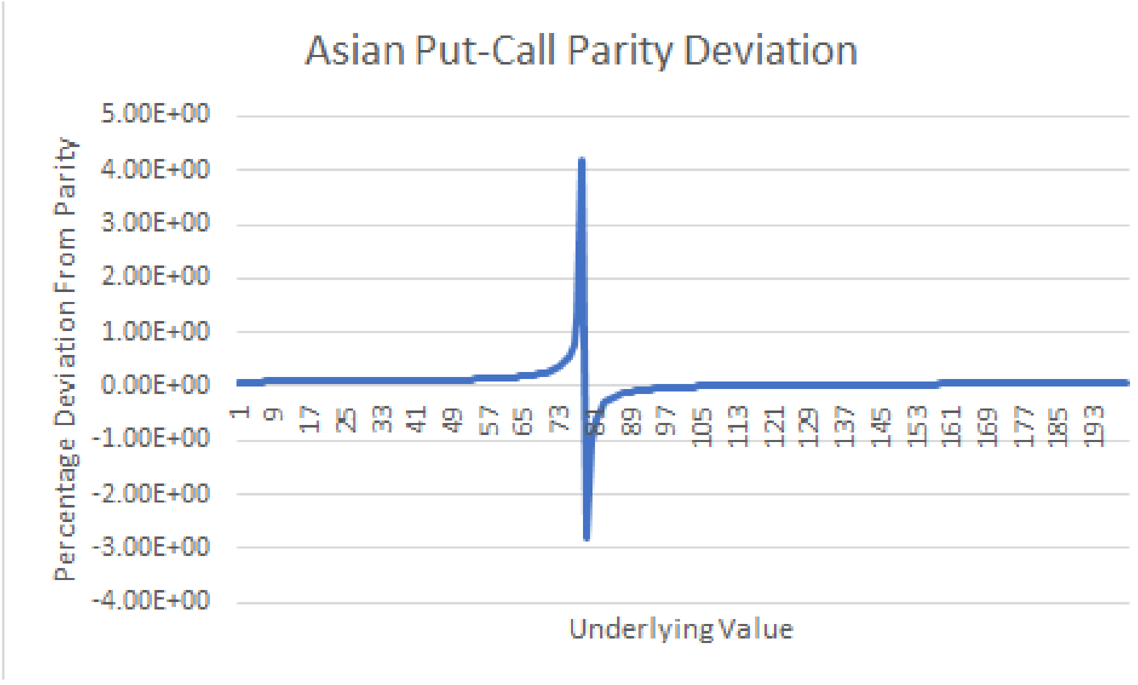
6.3 Asian Put-Call Parity

Asian Put-Call Parity can be expressed as,

$$C - P = e^{-rT}(E[S] - K), \tag{45}$$

where we have introduced the expectation operator $E[\cdot]$. This proved to hold pretty well, though there was a singularity as S approached the strike price (See figure 4 for details). The raw data can be seen on sheet

Figure 4: The fractional deviation from parity, for 1 year expiry asian options. Note, the singularity about the strike price due to the crude approximation used.



four of the excel document.

This deviation about the strike price is most likely due to the naivety of the approximation that I have used, which was to take $E[S] \approxeq S$ which resulted in a singularity on the RHS of (37).

7 Why are things dearer than others?

Throughout the project, we notice that certain options are systematically more or less expensive than others. For non dividend paying stocks, both european and american calls have identical values, due to the early exercise never proving optimal if there is no dividend to be paid out. Asian calls are inherently lower in value than their european and american counterparts as the averaging, also averages out the volatility, meaning the room for potential profit is narrower, hence the option is worth less.

For put options, the american puts are always dearer than their european equivalents, due to the added early exercise flexibility, which can prove optimal for the put options. Mathematically, this has to hold true else there would be an arbitrage opportunity. The american puts converge to the european put values as the underlying approaches infinity, obviously this is due to the early exercise boundary proving to never be optimal. Asian puts are the cheapest still, for the exact same reason as above.

As can be seen from the graphs, all of the options have greater values when the maturity increases. This is as expected, and can easily be seen from (43), (44), and (45), wherein as T increases, the discount factor diminishes the offset of the strike price, causing a larger spread between the put and call prices, causing both to move upwards and further apart with expiry. This effect is also naively attributed to the increasing time value of back dating the cash flows from the options.

All the options have value larger than the discounted payoff, except for the asian call options, which dip below this threshold with increasing S. The reason for this is from the put call parity itself (45) where it can be seen that the price of an asian call option is given by,

$$C = P + e^{-rT}(E[S] - K), \tag{46}$$

whilst the discounted payoff is given by,

$$\Omega = S - Ke^{-rT}. \tag{47}$$

We can now express the difference between the call option price and the discounted payoff as,

$$C - \Omega = P - S + e^{-rT}E[S] \approx P - S(1 - e^{-rT}), \tag{48}$$

where the right hand side now falls below 0 for increasing S.

8 Plots of Prices

Figure 5: Plots of option prices and the discounted payoff for three different put options with four varying values of expiry.

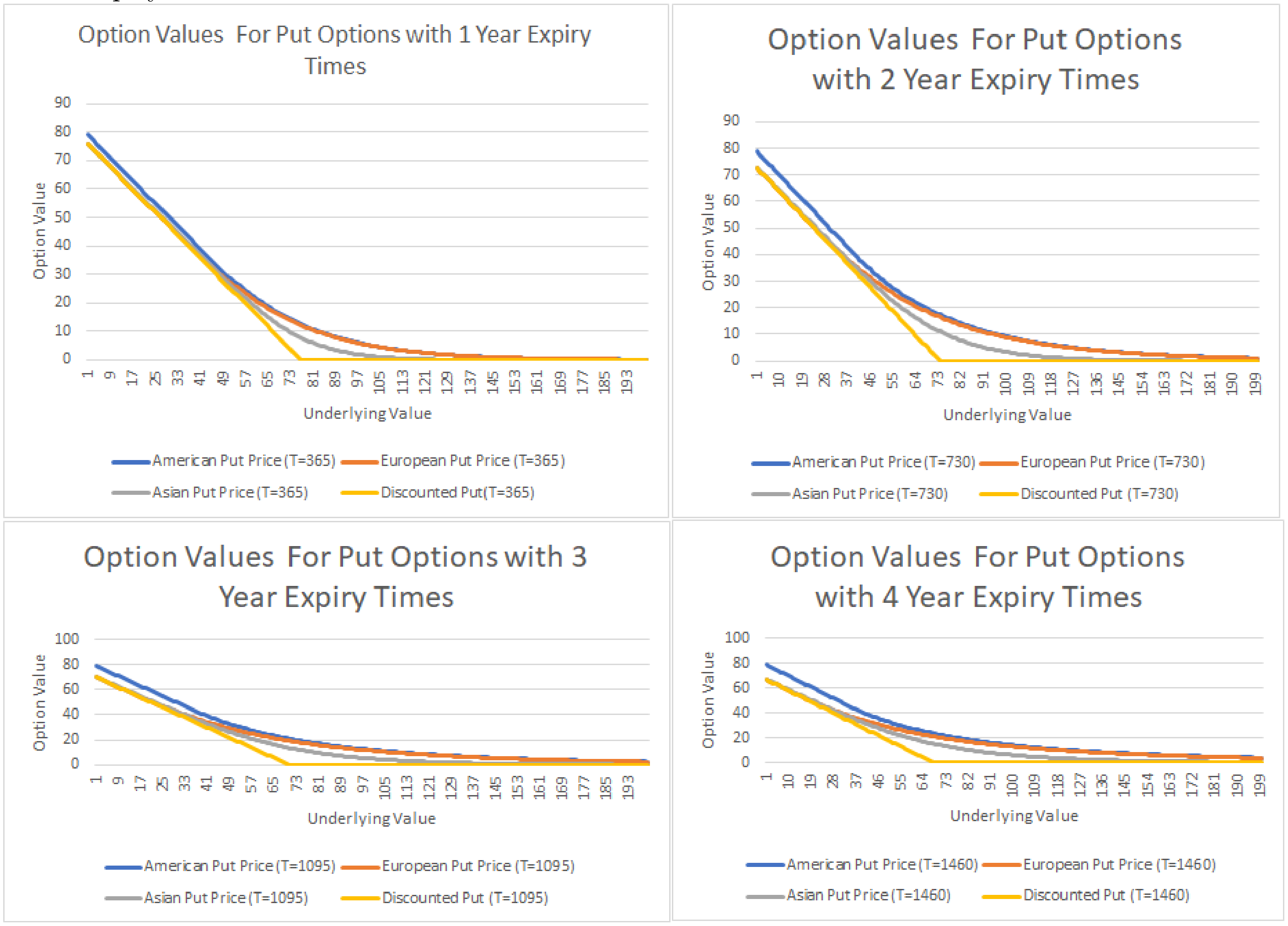


Figure 6: Plots of option prices and the discounted payoff for three different call options with four varying values of expiry.

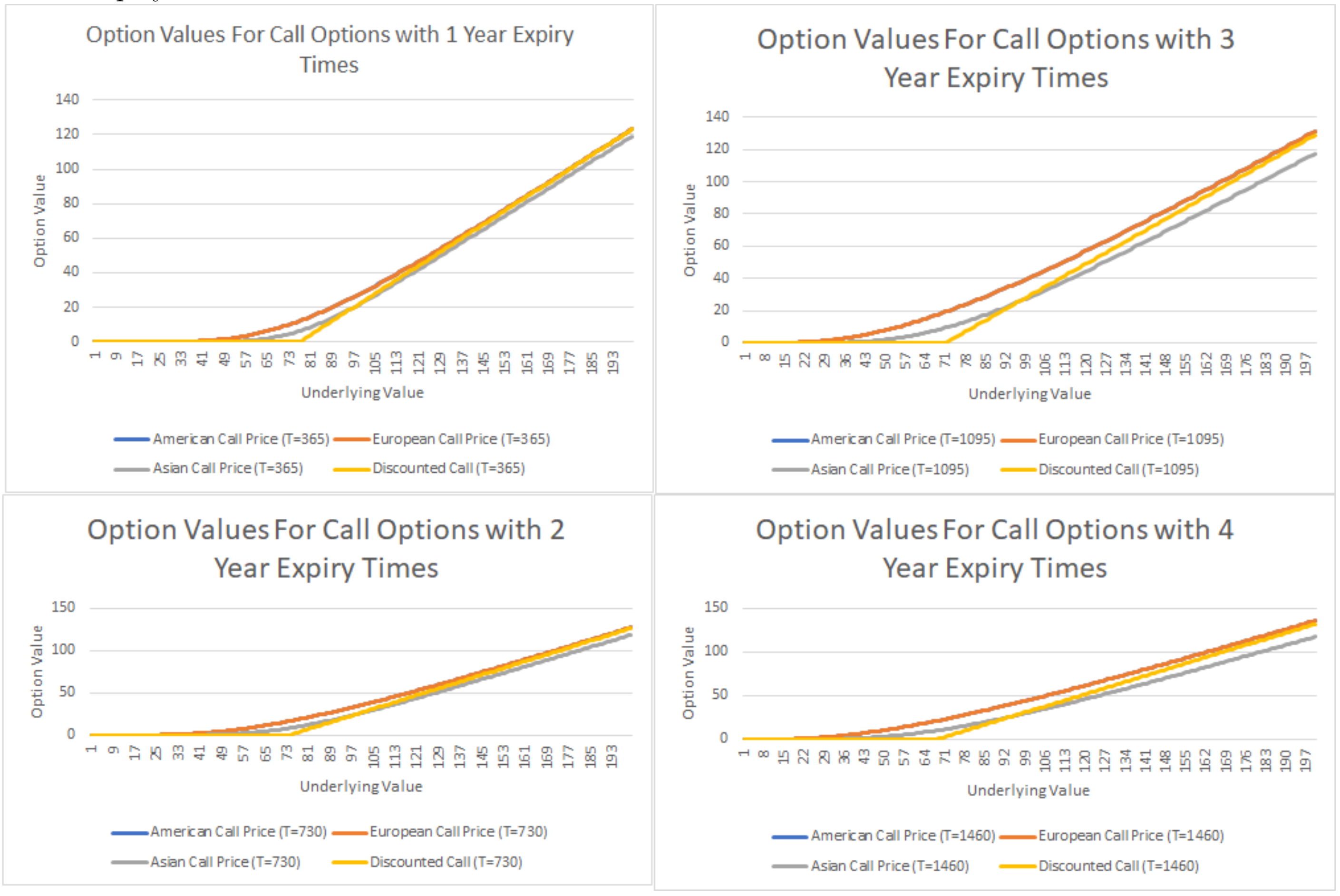
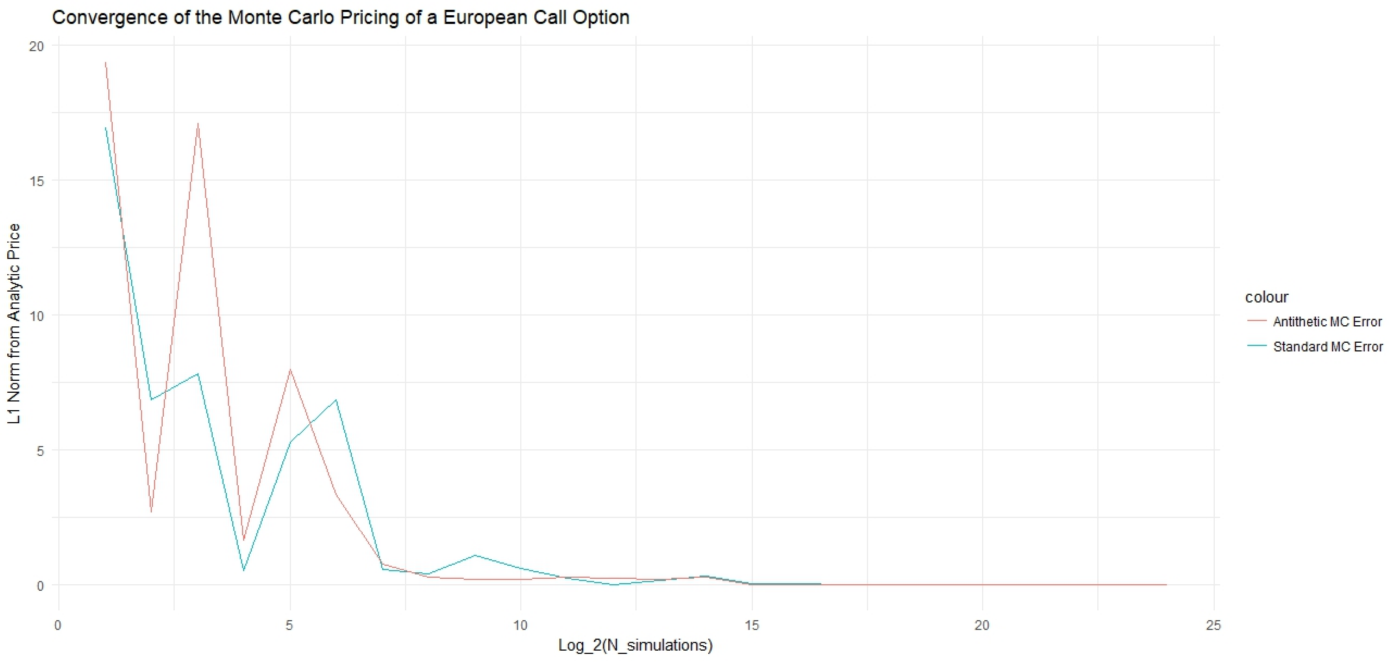


Figure 7: Plots showing the rate of convergence of the monte carlo estimation of a vanilla, one year to maturity, call option. One monte carlo method employed antithetic variates, whilst the other did not.


Convergence of the Monte Carlo Pricing of a European Call Option
L1 Norm from Analytic Price
Log_2(N_simulations)
colour
Antithetic MC Error
Standard MC Error