WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Individual Assignment Cover Sheet

Student Number:	**5581454**
Module Code:	**IB9JH0**
Module Title:	**Programming for Quantitative Finance**
Submission Deadline:	**25 January 2024**
Date Submitted:	**25 January 2024**
Word Count:	**2917**
Number of Pages:	**10**
Question Attempted: *(question number/title, or description of assignment)*	**all**
Have you used Artificial Intelligence (AI) in any part of this assignment?	**No**

IB9JH0 Programming Assignment 1

Chance Keown

January 25, 2024

Abstract

The main purpose of this project was to price vanilla options–including European puts and calls, as well as their American counterparts–using the Cox, Ross, and Rubenstein (CRR) Binomial model. This was implemented in C though two different methods, both for the European and American options. Ultimately, the runtime of each method was analyzed along with the convergence of European calls to that of the Black-Scholes analytic solution.

Unless otherwise noted, the following input parameters will be used:
Spot: 100, Strike: 95, Risk-free rate: .02, Dividend rate: .05, Volatility: .05, Expiration time: 1.

1 Benchmarking

First, we looked at the runtime of each of the binomial pricing implementations: backward induction and recursion. Both methods rely on the recursive definition for the option value along the binomial tree, which is given as follows for height j and node depth i for European options:

$$V_i^j = e^{-\Delta t}\big[p * V_{i+1}^{j+1} + (1-p) * V_{i+1}^{j}\big].$$

The corresponding recursive definition for American options considers the maximum of both the current value of the option and the weighted sum of future values, which are determined from the option's value at expiration. Here we define Δt as the change in time between node depths, while p is the risk-neutral probability of the underlying asset moving to the up state.

Differences arise in how the two methods implement this definition. The recursive function starts at the initial node ($i = 0, j = 0$) of the tree structure, and makes successive calls to itself until it reaches the ends of the tree at depth n, where the option can be valued at expiration and subsequently used to value prior nodes that feed into it. This effectively requires that every possible path in the tree structure be traced out to final nodes. In a binomial-expanding tree, there exist 2^n possible paths to traverse, where n is the depth of the tree (starting at 0 for only one existing node). This implies that the recursive implementation will call itself on the order of 2^n times to value an option with the accuracy that a tree of depth n provides. In fact, it will call itself $\left[\Sigma_{i=1}^{n} 2^i\right] - 1 = 2^{n+1} - 2$ times, given it will make 2 calls for each previous function call at depth i, starting with 2 function calls at the initial node.

The backward induction implementation on the other hand, requires far fewer operations. It starts with the end of the tree, at depth n, whhere it values the option at expiry, and uses the recursive definition to work its way to the first node, which has the current asset price. In each iteration of the double-nested for loop, two values from a higher-depth node are replaced with the corresponding value one branch earlier. At each subsequent depth, one fewer valuation must occur, as there is one fewer height at this lower tree depth. This leads to a total number of valuing operations performed by the algorithm on the order of

$$\sum_{i=0}^{n}(n+1-i) = \sum_{i=1}^{n+1} i = (n+1)(n+2)/2.$$

Rather than being exponential, the backward induction implementation only performs operations on the order of $O(n^2)$. It also avoids the messy number of recursive function calls, as it performs the option valuation in a self-contained manner.

Ultimately, we expect to see runtimes on the order of 2^n for the recursive implementation, while we only expect runtimes on the order of n^2 for the inductive function. By running the benchmarking file, we establish mean runtimes for each algorithm, in the context of European and American calls (as only the payoff function changes, we can assume that the put functions will perform similarly). With the recursive function, we take input depths $5, 10, 15, 18, 20$, and we expand this for the inductive function to $5, 10, 15, 25, 50, 200, 1000$ due to the sheer out-performance of this algorithm compared to the former. No benchmarks are run for earlier depths due to the lack of value the function outputs have at that point, which can be seen by the high error in early-depth trees in below figures.

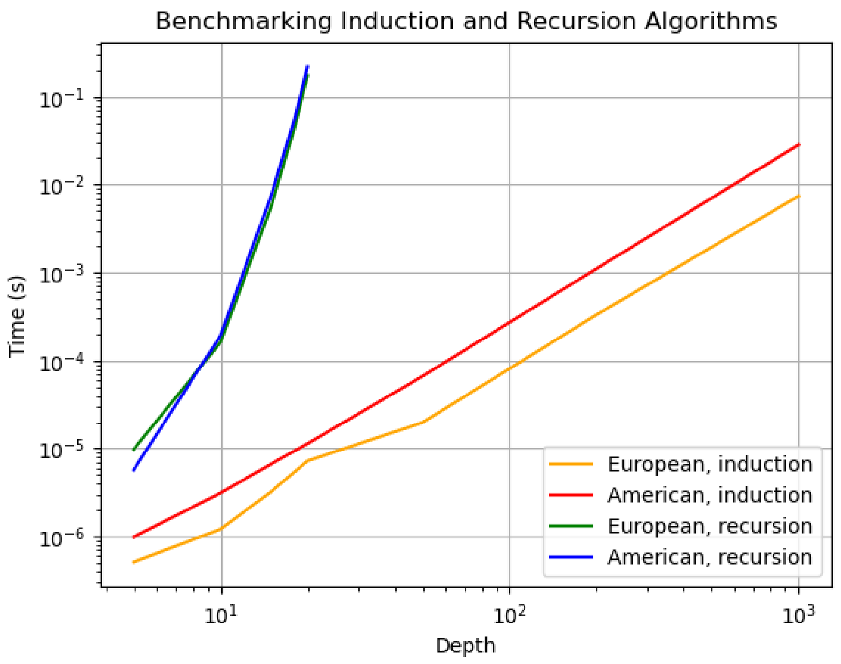

Figure 1: Runtime associated with induction and recursion implementations.

After plotting the runtimes on a log-log plot, we see a convincingly linear relationship for both inductive functions. The American call seems to take longer to execute, but this is justified by the fact that it must compare the recursive option value at expiration to the current value. Additionally, taking the slope of the induction plots shows that a $10-$fold increase in depth leads to about a $100-$fold increase in runtime. This is consistent with the algorithm being $O(n)$. As for the recursive runtimes, they quickly exceed that of the inductive algorithms, even with fewer than 5% of the depth that the inductive algorithm is plotted to. The different depths used allow one to easily see that the slopes of these plots are increasing, signifying an exponential increase in runtime.

2 Convergence

In the next aspect of the project, we look at convergence of the inductive algorithm to the theoretical Black-Scholes solutions. We look at European options, for which there exist an analytic solution, and in particular, we look at the error of the inductive call implementation, as the magnitude of the corresponding put error is the same due to put-call parity.

We make use of the same parameters in analysing convergence that we did with benchmarking. We will see that our option error for these parameters–with a strike price of 95–plotted by the blue line in Figures $2-4$, is quite oscillatory. However, we may wonder if our choice of parameters has anything to do with the oscillatory nature of our resulting error plot. Edward Omberg, in a paper on binomial-pricing convergence, analyses the convergence of this binomial model. He looks for possible parameter sets such that there exists smooth convergence from below when the set of exercise dates is doubled–that is, the tree depth doubles, containing all exercise dates that the previous tree contained[Omberg, 1987]. He notes that lack of a robust testing of parameter sets leaves "the attractive possibility that the binomial parameters can be chosen so as to guarantee uniform convergence", though the paper's final conclusion, by contradiction, is ultimately that no such parameter set exists[Omberg, 1987]. In context, our error oscillations are not a unique product of parameter choice. Therefore, we will only consider this parameter choice, and that with the altered strike price $K = 100$. The alternate strike price of $K = 100$ is used to demonstrate how convergence changes when the strike price falls in the center of the binomial tree.

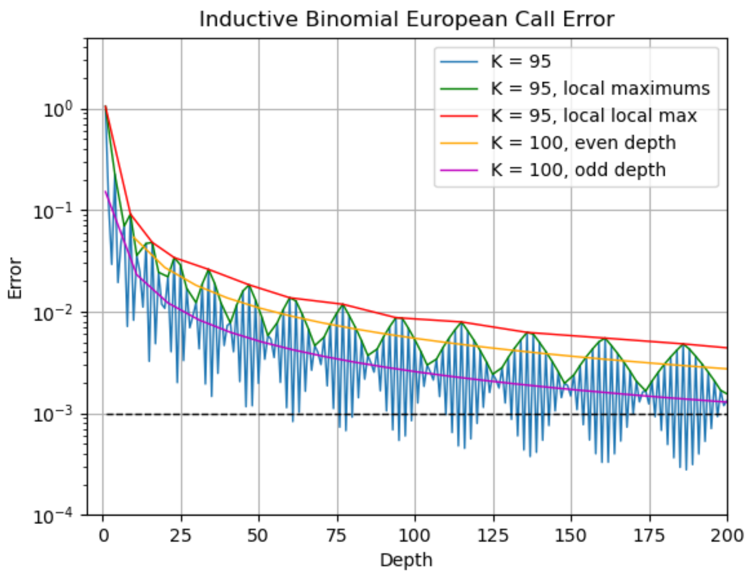

Figure 2: Error as a function of depth for European calls. For strike price $K = 95$, all depths are plotted so the full structure of the errors may be considered. Here, with strike prie $K = 100$, we see a much smoother connvergence.

Regarding strike prices in the center of the binomial tree, we first consider a model by Tian. In his paper on improvements to the Binomial pricing model, Tian (1999) describes how u and d don't have to be defined exactly as

$$u = e^{\sigma\sqrt{\Delta t}}, \qquad d = \frac{1}{u}$$

in order for the binomial method to converge to the Black-Scholes option price [Tian, 1999]. In fact, he notes that we only specifically require that the "discrete distribution of the asset price, represented by the binomial tree, converges to the continuous-time limit of a log-normal distribution"[Tian, 1999]. This occurs so long as u and d are defined as follows:

$$u = e^{\sigma\sqrt{\Delta t}+O(\Delta t)}, \qquad d = \frac{1}{u},$$

allowing a degree of freedom in the binomial model[Tian, 1999]. Tian goes on to suggest a Flexible Binomial model that utilises this in the form of a tilt parameter to center the strike price in the binomial tree [Tian, 1999]. The reason for centering the strike price involves the convergence results that arise in this setting. In particular, we are concerned with smooth convergence, where error decreases monotonically. Chang and Palmer remark on this smooth convergence of the Binomial model in special cases, including "at the money European calls, since for such options K always falls on a terminal stock price if n is even and is positioned at the geometric average of the two middle stock prices if n is odd"[Chang and Palmer, 2007]. Tian's flexible model effectively allows the strike price to be "at-the-money" in a sense, and leads to smooth convergence. In order to demonstrate this effect, we plot both even and odd tree depths (every 10 depths starting at either 1 or 10) for our altered strike price $K = 100$, which is at-the-money. We see that in both cases, our error plot seems to converge much more smoothly, with error being lower for the odd depths.

When looking at theoretical rate of convergence, we find that Ratibenyakool and Neammanee quickly outline the established convergence of the CRR model, starting with Heston and Zhou noting it was at least $1/\sqrt{n}$ [Ratibenyakool and Neammanee, 2020]. The paper then cites multiple authors, including Chang and Palmer, and Diener and Diener, who improved this bound to that of $1/n$ [Ratibenyakool and Neammanee, 2020]. The authors also note that there exist more quickly convergent models for option pricing, with rates on the order of $1/n\sqrt{n}$. In this regard, Chang and Palmer also note that Heath proved that the rate of convergence for the general CRR model cannot be improved beyond $1/n$ [Chang and Palmer, 2007].

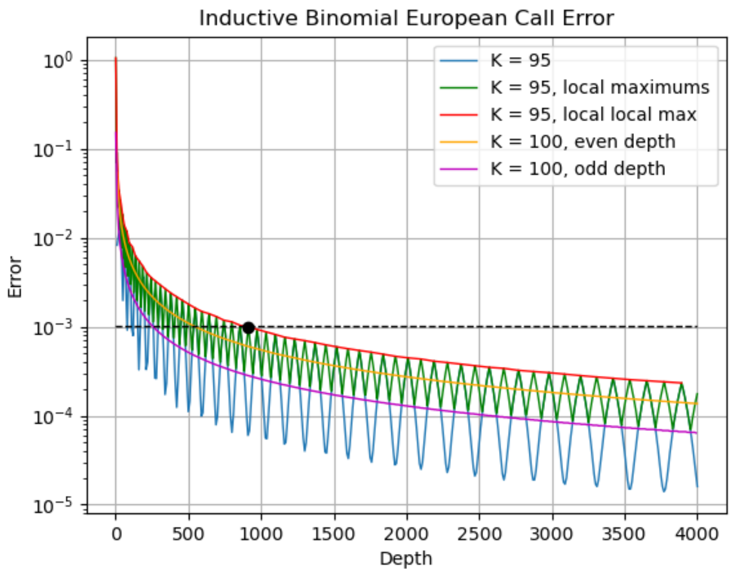

Figure 3: Here we look at a wider range of tree depths than Figure 2, but only plot every 10 depths of $K = 95$ for visibility purposes.

With our data, we look to confirm the above. In particular, we analyse two different sets of parameters, varied over tree depth. The first set of parameters is that outlined in the abstract. The second places the strike price at $K = 100$ such that it falls in the middle of our binomial tree. Each of these two cases has multiple associated graphs. First, we will look at Figure 2. Here, we have plotted in blue the error for our strike $K = 95$, plotting every single depth. We see that the error is incredibly oscillatory, and even looks oscillatory when considering the local maxima of the plot. In order to find more structure, we plot in green all of these local maxima, and then in red plot the local maxima of that set. This gives a much smoother looking line, which will be useful in visualising the $1/N$ convergence of the binomial method when the plot axes are changed. Next, we plot errors for $K = 100$, plotting first even-depth errors (every 10 depths starting at depth 10) in orange, and odd-depth errors (every 10 depths starting at depth 1) in magenta. Visually, we see a much smoother plot of error as a function of depth. This is in line with expected results based on the result mentioned by Tian.

In Figure 3, we replicate this plot with a higher final depth of 4000. Additionally, we only plot every 10 depths of $K = 95$, for a better visualisation. However, this removes some of the structure of the error plot that is clearer when all depths are plotted in Figure 1.

Finally, we look at these error plots on a log-log scale, in order to see if there is any structure. If the convergence is of order $1/N$, we should see a negatively sloping linear-looking plot of our errors for $K = 95$. The error for $K = 100$ should have a similar structure, as should any set of parameters. For the sake of analysis, we will look at the line of second-layer local maximums for $K = 100$, as we would like to look past the effect of the oscillations in the error. The red line that represents this in Figure 4 appears to have the property we are looking for, as do both error lines for $K = 100$.

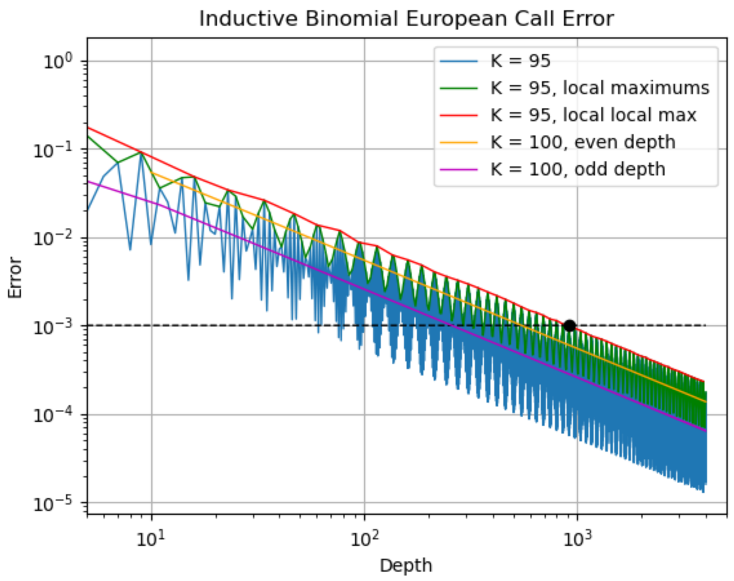

Figure 4: Here we plot both depth and error on a log scale. Both the strike $K = 100$ error lines and the second local maximum line for $K = 95$ demonstrate linear behaviour on this plot, implying that both are likely to converge at a rate of $1/N$.

While looking at the behaviour of the errors, we want to know when we can expect errors to fall below a certain level. In this case, we consider an error such that our option price will be accurate to .001, which is good for at least 4 significant figures of accuracy. We plot a line at this error value. Additionally, for $K = 95$, we include a point at which the the second set of local maxima passes below this value. This seems to be a good method for determining a depth where we can be sure error falls below .001 as it appears to be monotonically decreasing. In figures 3 and 4, we see that this error is certainly passed by the point where the tree depth is set to about 1000. In figure 2 we can see that the first time this error is surpassed is between a tree depth of 50 and 75, although oscillations yield higher errors, without sureness that error does not exceed .001 until we reach about the 1000 depth mark.

References

[Chang and Palmer, 2007] Chang, L.-B. and Palmer, K. (2007). Smooth convergence in the binomial model. *Finance and Stochastics*, 11(1):91–105.

[Omberg, 1987] Omberg, E. (1987). A note on the convergence of binomial-pricing and compound-option models. *The Journal of Finance*, 42(2):463–469.

[Ratibenyakool and Neammanee, 2020] Ratibenyakool, Y. and Neammanee, K. (2020). Rate of convergence of binomial formula for option pricing. *Communications in Statistics - Theory and Methods*, 49(14):3537–3556.

[Tian, 1999] Tian, Y. 1999). A flexible binomial option pricing model. *Journal of Futures Markets*, 19(7):817–843.

Appendix

Benchmarking Plots

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#depths at which induction and recursion runtimes are evaluated
the_depths_in = [5, 10, 15, 20, 50, 200, 1000]
the_depths_re = [5, 10, 15, 18, 20]

#induction runtimes, european and american calls
the_times_eu_in = [513.948 *10**(-9), 1.21112 *10**(-6), 3.25492 *10**(-6), 7.24232 *10**(-6),
                  19.7451 *10**(-6) , 331.786 *10**(-6), 7.40752 *10**(-3)]
the_times_am_in = [988.071 *10**(-9), 3.12426 *10**(-6),   6.70745 *10**(-6), 11.3658 *10**(-6),
                   66.4801 *10**(-6), 1.10932 *10**(-3), 28.3342 *10**(-3)]

#recursion runtimes, european and american calls
the_times_eu_re = [ 9.76787 *10**(-6), 161.994 *10**(-6), 5.80136 *10**(-3),
                   43.5669 *10**(-3), 174.447 *10**(-3) ]
the_times_am_re = [5.71864 *10**(-6),  192.344 *10**(-6),  7.76419 *10**(-3),
                   54.6707 *10**(-3),  219.444 *10**(-3)]


#plot induction runtimes, european and american calls
plt.plot(the_depths_in, the_times_eu_in, color = 'orange', label = "European, induction")
plt.plot(the_depths_in, the_times_am_in, color = 'r', label = "American, induction")

#plot recursion runtimes, european and american calls
plt.plot(the_depths_re, the_times_eu_re, color = 'g', label = "European, recursion")
plt.plot(the_depths_re, the_times_am_re, color = 'b', label = "American, recursion")

#formatting plot
plt.title("Benchmarking Induction and Recursion Algorithms")
plt.xlabel("Depth")
plt.ylabel("Time (s)")
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```

Data organisation

Below is the code used to generate convergence plots for European call errors.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#reads csv of errors for european call, with depths 1 to 4000
#parameters: Volatility = .05, Risk free rate = .02, Dividend Rate = .05
#            Strike Price = 95, Initial Price = 100, Expiration Time = 1
convergence = pd.read_csv("output (7).csv")

#creates a list of errors corresponding to local maximum errors in depth v. error graph
#creates a list of depths at which there exists a local maximum error
list_depths = []
list_errors = []
for index, row in convergence.iterrows():
    if row['Depth'] == 1:
        if convergence['Call error'][index +1] < row['Call error']:
            a = row['Depth']
            b = row['Call error']
            list_depths.append(a)
            list_errors.append(b)
    if row['Depth'] >1:
        if convergence['Call error'][index -1] < row['Call error'] and convergence[
                        'Call error'][index +1] < row['Call error']:
            a = row['Depth']
            b = row['Call error']
            list_depths.append(a)
            list_errors.append(b)

#creates a list of errors corresponding to local maximums of the local maximums list
#creates a list of depths corresponding to local error maximums
bound_depths = [list_depths[0]]
bounds = [list_errors[0]]
for i in range(1, len(list_errors)-1):
    if list_errors[i-1]<list_errors[i] and list_errors[i+1]<list_errors[i]:
        bound_depths.append(list_depths[i])
        bounds.append(list_errors[i])

#finds the first point where local maximums of local maximums falls below a given error
#error considered here is .001
point_x = []
point_y = []
error = .001
for i in range(1, len(bounds)-1):
    if bounds[i] > error and bounds[i+1] < error:
        point_x.append(bound_depths[i+1])
        point_y.append(bounds[i+1])

#creates a data frame of errors for a changed strike price K = 100
#depths 10 to 4000 spaced every 10 depths
df = pd.read_csv('output (10).csv')
#depths 1 to 3991 every 10 depths
df3 = pd.read_csv('output (9).csv')
```

```
#creates data frames of errors for strike price K = 95
# depths 10 to 4000 every 10 depths
df2 = pd.read_csv('output (6).csv')

#creates values to produce a line at a given error level in our chart
#error considered is .001
sigfig = np.linspace(error, error, 4001)
steps = np.linspace(1, 4000, 4001)
```

Error plot, y-values log-scaled

The following creates the plot of errors, with y axis log-scaled. This code is used to create both figures 2 and 3.

```
#y-log plot of the errors from Black_Scholes solutions for two different strike prices
#plotted on depth 1 to 4000

#plots oscillating errors for call with strike K = 100 (every 10 depths)
ax = df2.plot(x = 'Depth', y = 'Call error', linewidth = 1,
              color = '#1f77b4', label = "K = 95")

#plots set of local maximums and set of those local maximums
plt.plot(list_depths, list_errors, 'g', linewidth = 1, label = "K = 95, local maximums")
plt.plot(bound_depths, bounds, linewidth = 1, color = 'r', label = "K = 95, local local max")

#plots even and odd tree depth errors for changed strike price K = 95
ax = df.plot(ax = ax, x = 'Depth', y = 'Call error', linewidth = 1,
             color = 'orange', label = "K = 100, even depth")
df3.plot(ax = ax, x = 'Depth', y = 'Call error', linewidth = 1,
         color = 'm', label = "K = 100, odd depth")

#plots line of given error
# plots first point where second layer of local maximums passes below given error
plt.plot(steps, sigfig, 'k--', linewidth = 1)
plt.plot(point_x, point_y, 'o', color = 'k')

#Adusts plot to y-log axis, adds grid lines, labels, and x-axis lmits
plt.title('Inductive Binomial European Call Error')
plt.yscale('log')
plt.grid()
plt.legend()
plt.ylabel('Error')
plt.show()
```

Log-Log plot of errors

The following code creates the log-log plot of errors.

```
#Log-Log plot of the errors from Black_Scholes solutions for two different strike prices
#plotted on depth 1 to 4000

#Plot with strike price of K = 95, depths 1 to 4000
ax = convergence.plot(x = 'Depth', y = 'Call error', linewidth = 1,
                      color = '#1f77b4', label = "K = 95")

#plots set of local maximums and set of those local maximums
plt.plot(list_depths, list_errors, 'g', linewidth = 1,
         label = "K = 95, local maximums")
```

```
plt.plot(bound_depths, bounds, 'r', linewidth = 1,
         label = "K = 95, local local max")

#plots line of given error
# plots first point where second layer of local maximums passes below given error
plt.plot(steps, sigfig, 'k--', linewidth = 1)
plt.plot(point_x, point_y, 'o', color = 'k')

#plots even and odd tree depth errors for changed strike price K = 95
df.plot(ax = ax, x = 'Depth', y = 'Call error', linewidth = 1,
        color = 'orange', label = "K = 100, even depth")
df3.plot(ax = ax, x = 'Depth', y = 'Call error', linewidth = 1,
         color = 'm', label = "K = 100, odd depth")

#Adusts plot to log-log axes, adds grid lines, labels, and x-axis lmits
plt.title('Inductive Binomial European Call Error')
plt.yscale('log')
plt.xscale('log')
plt.grid()
plt.legend()
plt.ylabel('Error')
plt.xlim(5, 5000)
plt.show()
```