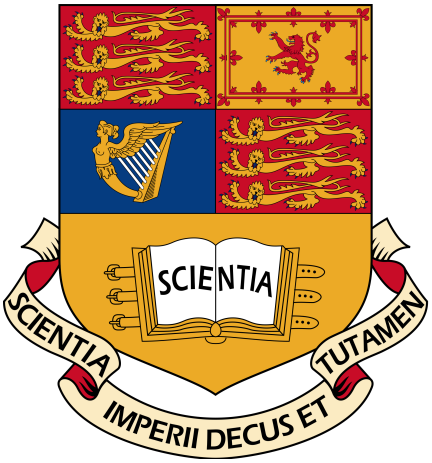
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2017

---




| | |
|---|---|
| Project Title: | **Asian Option Pricing Optimisation Techniques for GPUs** |
| Student: | **Nicholas A. Robertson** |
| CID: | **00842277** |
| Course: | **EIE4** |
| Project Supervisor: | **Dr David B. Thomas** |
| Second Marker: | **Dr John Wickerson** |

**Abstract**

Option pricing is often a computationally intensive process, particularly when the option requires the use of Monte Carlo based pricing methods. Last year, Pieter Fabry developed a new method based on Monte Carlo pricing over Cox-Ross-Rubinstein binomial trees, using random asset paths based on flipping a biased coin, rather than generating a log-normal distributed sample. He discovered that the parallelisation benefits of the discrete model outweighed the expected losses in accuracy hardware for FPGAs.

This project concerns the analysis and implementation of discrete space pricing schemes for GPUs. The project considers whether the parallelisation advantages posited for FPGAs hold for GPUs. The basis of this discrete-space investigation considers Jarrow-Rudd binomial trees, multinomial trees, antithetic sampling, control variates, and multi-level techniques.

It is found that normally sampled multinomial trees can be constructed with the same convergence properties as a continuous Gaussian walk. On a GPU, the continuous model can be replaced with a sampled multinomial tree with 64 discretisations - this resulted in a 3.14× increase in throughput compared to the Gaussian walk. Antithetic sampling improved the throughput increase to 3.65×. The addition of a geometric control variate as well as the antithetic variate resulted in an 18.82× speed up for a 99% confidence interval of size $2 \times 10^{-3}$.

**Acknowledgements**

I would like to thank David Thomas for all the time and advice he has given throughout the course of this project as well as my time at Imperial; I say with complete sincerity that I would not have been able get to this stage of my degree without his guidance. I would also like to thank Pieter Fabry for taking the time to elucidate the finer details of his project.

# Contents

## Formal Definitions

| | |
|---|---|
| **Option (Derivative)**: | a contract between two parties that gives the holder the right to buy or sell a stock by a certain date for a certain price |
| **Stock Price**: | the initial price of a stock |
| **Strike Price**: | the price of the underlying stock at which the option can be exercised |
| **Implied Volatility**: | the expected volatility of the underlying stock |
| **Exercise (Expiration)**: | the date at which the option can be executed |
| **Arbitrage**: | mispricing of assets leading to abnormal profits |
| **Short Position**: | position in which the investor owes the stock and expects the value to decrease |
| **Long Position**: | position in which the investor owns the stock and expects the value to increase |
| **Black-Scholes Equation**: | partial differential equation used for pricing vanilla European put and call options [4] |
| **Kemna-Vorst Equation**: | partial differential equation used for pricing geometric Asian put and call options [24] |

## Notation

| | |
|---|---|
| $S_0$ | Initial stock price |
| $K$ | Strike price |
| $T$ | Time to expiration of option |
| $S_T$ | Stock price on the expiration date |
| $\sigma$ | Volatility of the stock asset |
| $r$ | Risk-free interest rate |
| $\Delta t$ | Size of discrete time step |
| $MSE$ | Mean Square Error |
| $RMSE$ | Root-Mean-Square Error |
| $T_r$ | *True* Value of an Option |
| $c$ | Value of a call option |
| $p$ | Value of a put option |
| $M$ | Steps in a Monte Carlo Simulation |
| $N$ | Iterations in a Monte Carlo Simulation |

# 1 Introduction

Derivatives are one of the most important tools for investors in financial markets. This is in part down to derivatives' intrinsic properties that make them ideal for hedging, speculation, and arbitrage. Consequently, the derivatives market has ballooned, and it is now greater in size than the stock market (in terms of assets) [19]. Options, a class of derivatives, are an agreement between a buyer and a seller that gives the holder the right to buy or sell an underlying financial asset by a certain date for an agreed price. There exist many types of options, each with slightly different properties (e.g. execution conditions). Asian options were devised to reduce the ability of either party to manipulate the underlying asset price at the time of execution by considering the average value of the asset throughout the life of the option. This property results in path dependence for the option price which introduces complexity making it more difficult to quickly and accurately price than its more vanilla counterparts. Investors want to price options rapidly as spotting arbitrage opportunities allows them to profit before the market corrects itself.

This project seeks to expand on the ideas presented in Fabry's paper, *Option-pricing using Monte Carlo over Lattices* [9]. The paper is a thorough investigation of discrete time and space Monte Carlo simulations to price Asian Options on FPGAs. The basis of the discrete space model is considering a set of values a stock can take at a given point in time as opposed to considering a log-normal distributed sample. He discovered that the parallelisation benefits of the discrete model outweighed the expected losses in accuracy for FPGAs. The focus of this paper is an investigation into the plausibility of discrete time and space Monte Carlo simulations to price Asian Options on GPUs, and an exploration into further techniques to improve convergence and execution speed through control variate, antithetic, and multi-level pricing simulations.

This report contains the background research carried, followed by a clarification of the goals of the project based on these findings. It then examines the convergence properties of a large variety of discrete space Monte Carlo simulations, the continuous model, and a series of variance reduction techniques. With this information the report proceeds to seek out the optimal methods for applying each of these algorithms on a GPU.

It is found that normally sampled multinomial trees can be constructed with the same convergence properties as a continuous Gaussian walk. On a GPU, the continuous model can be replaced with a sampled multinomial tree with 64 discretisations - this resulted in a 3.14× increase in throughput on a NVIDIA Tesla K80. Antithetic sampling improved the throughput increase to 3.65×. When compared to Fabry's FPGA implementation it was found that the antithetic sampled multinomial tree uses 24% less energy to calculate an accuracy of $10^{-3}$. The introduction of a geometric control variate resulted in a 18.68× speed up for a 99% confidence interval of size $2 \times 10^{-3}$, and combined with antithetic sampling, a speed up of 18.82×.

# 2 Project Specification

The primary objective of the project is to investigate the properties associated with discrete time and discrete space pricing models for Asian Options on a GPU. The investigation will be oriented about answering these questions:

1. How do the convergence properties of discrete space models differ from the Gaussian continuous space model?

   A basic level of understanding needs to be acquired in the difference and similarities in convergence of the discrete and continuous space methods in order to present an appropriate basis that allows for the comparison of the two.

2. How should the algorithms be parallelised and split across compute units?

   Each pricing method has different performance characteristics which in turn will result in different approaches to splitting the computation of the simulation.

3. What advantages of Fabry's discrete time and discrete space model on an FPGA are directly applicable to GPUs?

   This question provides an important entry point to understand the performance characteristics of GPUs and informs the subsequent sections of the project.

4. What algorithmic changes should be made when targeting a GPU?

   Following on from the previous question, understanding how to alter the models to increase their throughput on GPUs is critical in maximising their performance.

5. What improvements in performance and accuracy can be obtained using a non-recombinant models on a GPU?

   An essential component of Fabry's optimisations for FPGAs was using the LUT to store the discrete space the stock path could take. It is as such of interest to consider whether dynamically generating discrete space models rather than storing it in memory is an appropriate approach to optimise the pricing methods.

6. What improvements in performance and accuracy can be obtained using antithetic sampling, control variates, multi-level, and Quasi-Monte Carlo method on a GPU?

   The continuous model is able to have its performance enhanced through the use of variance reduction techniques and stochastic improvement techniques. These methods should be considered to understand how they improve the performance and accuracy of the simulation as well as considering their applicability to discrete space models on GPUs.

7. Of the methods examined which has the best execution properties for a GPU and how does this compare to Fabry's FPGA approach?

   Understanding how the project compares with the existing methods available determines whether the GPU architecture is appropriate hardware for running Monte Carlo option pricing on.

# 3 Background

## 3.1 Options

A derivative is a contract between two or more parties that's value is based on an underlying variable. Common types of derivatives include futures, options, forwards, swaps, and warrants. Derivatives can be traded either over-the-counter or on an exchange. A derivatives exchange is a marketplace where individuals trade standardised contracts that are defined by the exchange. Originally these markets were an open-outcry system, however with the prevalence of digital technology there has been a massive shift towards electronic trading. The over-the-counter market is a network of dealers in which two parties are able to buy and sell specially designed contracts (non-standardised).

A stock option is a contract between two parties that gives the holder the right but not the obligation to buy or sell a stock by a certain date for a certain price. There are two types of options: a call option gives the holder the right to buy the underlying asset by a certain date for a certain price, a put option gives the holder the right to sell the underlying asset at a certain date for a certain price. The price specified in the contract is referred to as the strike price; the date specified in the contract is the expiration date or maturity. The investor who has purchased the option has taken a long position, and the investor who has written (or sold) the option has take a short position. [19]

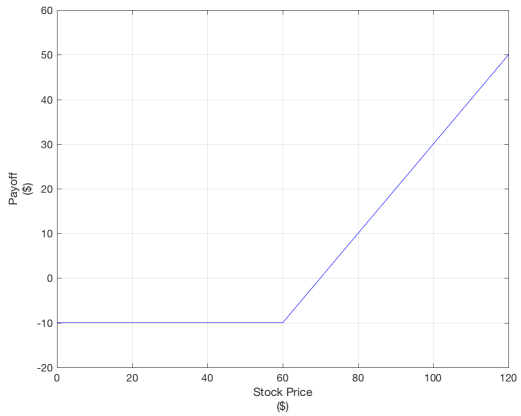
There are many variants of options. The two standard options are European and American, other options are regarded at exotic (e.g. Asian). European options are those than can only be exercised on the expiration date. American options can be exercised at any point up to and including the expiration date. Asian options can only be exercised at expiration date however the value of the underlying stock is determined by considering the average value of the stock over the life of the option.

Asian Options have the property that investors are less able to force arbitrage as any attempts to manipulate the stock price before the expiration date will have negligible effects on the average price of the stock. As the option considers the average value of the underlying stock, an Asian option is less dependent on the volatility of the stock in comparison to its European and American counterparts.

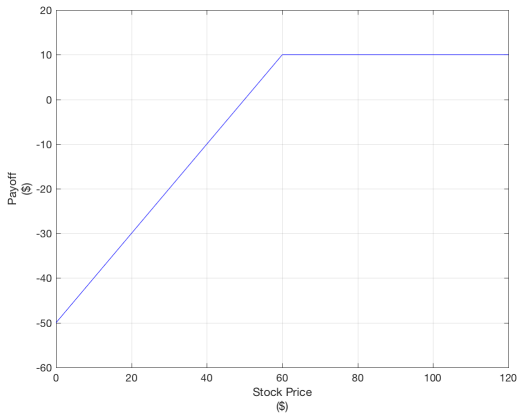
An important underlying relationship between a European put and call option that have the same strike price and time to maturity; this is a Put-Call Parity. The relationship asserts that shorting a call is the same as longing a put and longing a call is the same as shorting a call [41]. The equation below shows the relationship between the call and put price. The arbitrage basis of this argument holds for Asian options.

$$c + Ke^{-rT} = p + S_0 \tag{3.1}$$

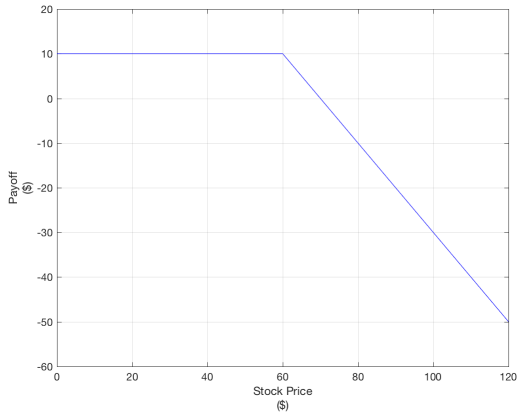
Pricing Options is the calculation carried out in order to determine the amount of money that the option should be worth given the *known* variables about the underlying asset. European options that pay no dividends can be priced precisely using the Black-Scholes model [4]. Asian Options cannot be directly calculated due to their path dependence. There are two families of methods for pricing



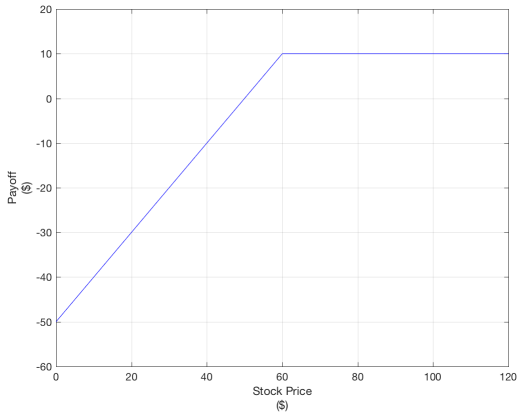

(a) Long Call




(b) Long Put




(c) Short Call




(d) Short Put

Figure 3.1: Payoffs for all available option positions

them: Backward-Tree models and Monte Carlo-forward simulations.

## 3.2 Gaussian Walks

The idea that underpins the Black-Scholes model [4] is that stock prices can be modelled as a Brownian random walk.

A Markov process is a type of stochastic process where only the current value of a variable is relevant for generating future values. Stock prices are modelled as Markov processes as it is assumed that past values do not influence any future values of the stock. As such the probability distribution of the price at any particular time in the future is independent of the previous prices.

A Wiener process is a subset of Markov Processes with additional properties. It is one that has mean change zero and variance one (per year). This particular type of process is also known in the realm of physics as Brownian Motion. This in turn gives the process two properties:

1. The change $\Delta z$ during a small period of time $\Delta t$ is

$$\Delta z = \epsilon \times \sqrt{\Delta t} \tag{3.2}$$

where $\epsilon$ is a standard normal distribution

2. The values of $\Delta z$ for any two different short intervals of time, $\Delta t$, are independent.

It follows that the mean of $\Delta z = 0$, the standard deviation of $\Delta z = \sqrt{\Delta t}$, and the variance of $\Delta z = \Delta t$. [19]

A generalised Wiener process for a variable $x$ can be represented as:

$$dx = adt + bdz \tag{3.3}$$

where $a$ and $b$ are constants. The $adt$ term implies that $x$ has a drift rate per unit of time, and $bdz$ is noise component resulting from a basic Wiener process.

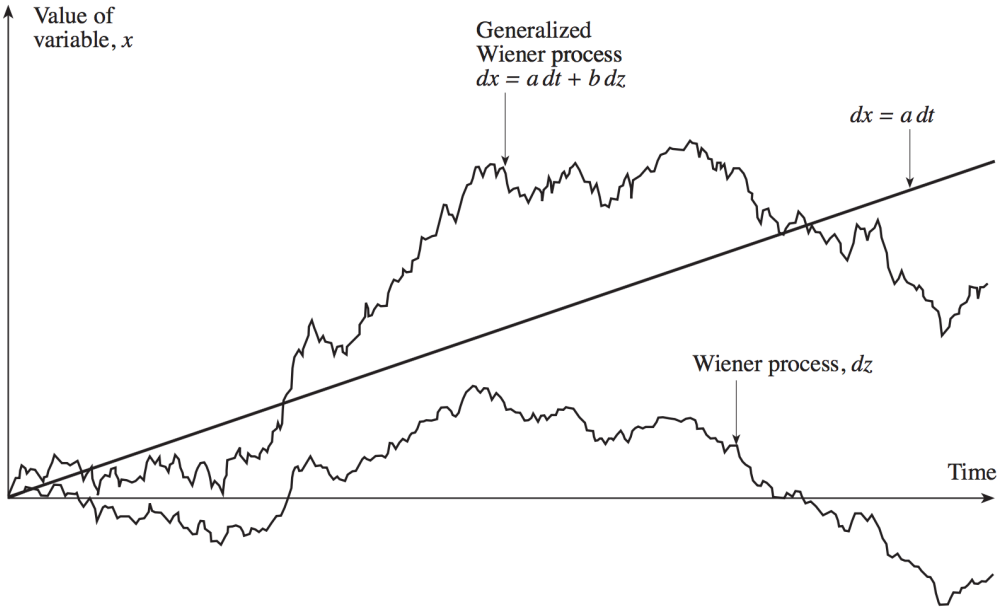



Figure 3.2: Generalized wiener process [19]

A Wiener process is used to represent the value of a stock $S$ with volatility $\sigma$ as

$$dS = \mu Sdt + \sigma Sdz \tag{3.4}$$

$\mu Sdt$ represents the risk-less rate of return of the stock and $\sigma Sdz$ represents the uncertainty of the future value due to the stock's volatility.

An Ito Process is a type of generalised Wiener process where a and b are functions of $x$ and $t$

$$dx = a(x,t)dt + b(x,t)dz \tag{3.5}$$

Ito's Lemma [20] is an important result of an Ito Process. It shows that a function $G$ of $x$ and $t$ follows the process

$$dG = (\frac{\delta G}{\delta x}a + \frac{\delta G}{\delta t} + \frac{\delta^2 G}{\delta x^2})dt + \frac{\delta G}{\delta x}bdz \tag{3.6}$$

Consequently, G is also a Ito process with a drift rate of

$$(\frac{\delta G}{\delta x}a + \frac{\delta G}{\delta t} + \frac{\delta^2 G}{\delta x^2})dt \tag{3.7}$$

and variance rate of

$$\frac{\delta G}{\delta x}bdz \tag{3.8}$$

From Ito's lemma is follows that a process followed by a function $G$ of $S$ and $t$ is

$$(\frac{\delta G}{\delta S}\mu S + \frac{\delta G}{\delta t} + \frac{1}{2}\frac{\delta^2 G}{\delta S^2}\sigma^2 S^2) \ + \frac{\delta G}{\delta S}\sigma S dz \tag{3.9}$$

The log-normal property is an important property of stock prices that states that the stock price at time T, given its price today, is lognormally distributed. The standard deviation is proportional to the square root of how far ahead the look ahead is. It is derived through the use of Ito's Lemma

$$G = lnS \tag{3.10}$$

$$dG = (\mu - \frac{\sigma^2}{2})dt + \sigma dz \tag{3.11}$$

$$lnS_T - lnS_0 \sim \phi[(\mu - \frac{\sigma^2}{2})T + \sigma^2 T] \tag{3.12}$$

Using the log-normal returns property of stocks, the expected return of the stock can be written as

$$S_T = S_0 e^{(r - \frac{\sigma^2}{2})dt + \sigma\phi(0,1)dt} \tag{3.13}$$

This would be appropriate for calculating the price of the option however it needs to reflect the discount rate associated with the future pay off hence the option price can be calculated using the value

$$e^{-rT} \times E[S(T)] \tag{3.14}$$

where $E[S(T)]$ is the expected future pay off.

Gaussian walks can be used as part of the Monte Carlo approach is applied in situations where the Black-Scholes method cannot be used such as in Asian Options. Gaussian walks is addressed in more detail later in the background research.

## 3.3 Binomial Trees

Binomial trees for pricing options is a popular method due to its simplicity as well as its ability to handle more complex option properties (in particular American options). The model breaks down the time of the option into discrete steps as well as breaking the values the stock can take into discrete values. As the time step tends to zero it can be shown that the value calculated will tend to the Black-Scholes model.

The initial scenario considered is a stock that will either have gone up or down at the end of a time period as shown in Figure 3.3. $f$ is the option price at that point in the tree, $S_0$ is the stock price now, $u$ is an upward movement in price ($u > 1$), and $d$ is a downward movement in price ($0 < d < 1$).

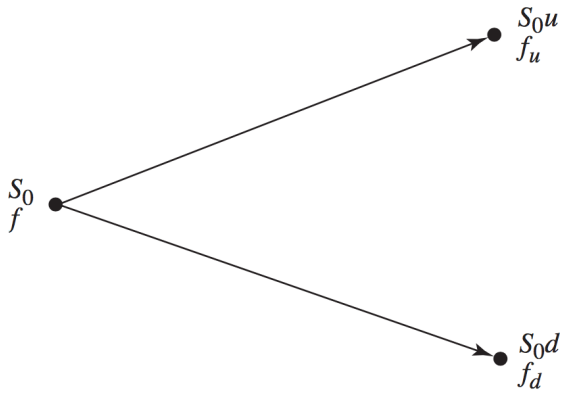



Figure 3.3: Stock and option prices in a general one-step tree [19]

The assumption is made that there are no arbitrage opportunities in pricing the option. To create a risk-less portfolio, the pay-off of an option whether it goes up or down must be equal, hence

$$S_0u\Delta - f_u = S_0d\Delta - f_d \tag{3.15}$$

$$\therefore \Delta = \frac{f_u - f_d}{S_0u - S_0d} \tag{3.16}$$

This enables option price, f to be determined as

$$S_0\Delta - f = (S_0u\Delta - f_u)e^{-rT} \tag{3.17}$$

$$f = S_0\Delta(1 - ue^{-rT}) + f_ue^{-rT} \tag{3.18}$$

$$f = e^{-rT}[pf_u + (1-p)f_d] \tag{3.19}$$

where

$$p = \frac{e^{rT} - d}{u - d} \tag{3.20}$$

Risk-neutral valuation is the assumption that investors do not increase the expected return they require from an investment to compensate for increased risk. Two features come from this assumption: The expected return on a stock is the risk-free rate, the discount rate used for the expected payoff of an option is the risk-free rate. This in turn allows the interpretation of the variable $p$ (in the equation above) to be seen as a probability of an upward movement in a risk neutral world.

The concept behind the single step binomial tree can be extended to multiple steps. To better understand this a two step binomial tree has been considered.

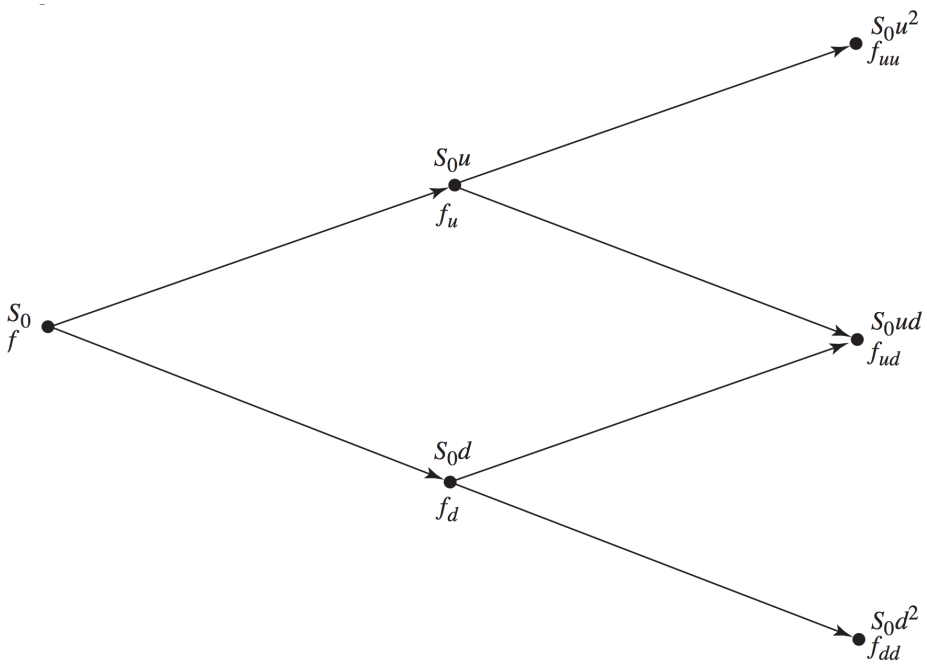



Figure 3.4: Stock and option prices in general two-step tree [19]

The change to two steps means that $\Delta T$ is now considered rather than $T$

$$p = \frac{e^{r\Delta T} - d}{u - d} \tag{3.21}$$

From the previous equations derived from the single tree it can be shown that

$$f_u = e^{-r\Delta T}[pf_{uu} + (1-p)f_{ud}] \tag{3.22}$$

$$f_d = e^{-r\Delta T}[pf_{ud} + (1-p)f_{dd}] \tag{3.23}$$

$$f = e^{-r\Delta T}[pf_u + (1-p)f_d] \tag{3.24}$$

hence

$$f = e^{-2r\Delta T}[p^2 f_{uu} + 2p(1-p) + (1-p)^2 f_{dd}] \tag{3.25}$$

The determination of appropriate values for $u$, $d$, and $p$ is an important part of ensuring the model is representative of the underlying stock. As the time step tends to zero, the model should have expected value and variance that matches the Wiener process representing the stock price.

The standard model is the Cox-Ross-Rubinstein model [7]. This model is created by setting the up and down values to the inverse of each other.

$$u = e^{\sigma\sqrt{\Delta T}} \tag{3.26}$$

$$d = e^{-\sigma\sqrt{\Delta T}} = u^{-1} \tag{3.27}$$

$$p = \frac{e^{r\sqrt{\Delta T}} - d}{u - d} \tag{3.28}$$

An alternative model is the Jarrow-Rudd model [21]. This model is obtained by setting probabilities of an up an down movement to be equal.

$$u = e^{(r-\frac{\sigma^2}{2})\Delta T+\sigma\sqrt{\Delta T}} \tag{3.29}$$

$$d = e^{(r-\frac{\sigma^2}{2})\Delta T-\sigma\sqrt{\Delta T}} \tag{3.30}$$

$$p = 0.5 \tag{3.31}$$

The two step binomial tree model can be expanded by further decreasing the step size which increases the number of steps required to reach the final time. As the number of steps increases the model becomes a better approximation of a Gaussian walk.

Backwards induction walks are used to determine the value of the option by propagating the value of the leaf nodes back to the head of the tree. This method however is not applicable for path-dependent options, such as Asian options. When backwards induction walks cannot be computed, the preferred method of computation is Monte Carlo random walks.

## 3.4 Monte Carlo Random Walks

Monte Carlo simulations use random sampling and statistical modelling to estimate functions and mimic the operations of complex systems that are otherwise difficult to simulate entirely [17]. In the context of computational finance, Monte Carlo simulations can be used to simulate the movement of a stock.

The standard error of a Monte Carlo simulation is inversely proportional to the square of the number of iterations [14] due to the central limit theorem. This property makes it computationally intensive to get a value that has a small confidence interval.

A confidence interval is a measure that gives an estimated range of values which are likely to include an unknown population parameter. The confidence level is the probability that the value falls within the estimated range. The width of the confidence interval gives a measure of the uncertainty of the unknown parameter. The factors that determine the width of the confidence interval are the size of the sample, the confidence level, and the measured variance of the sample. The confidence width is a useful measure of the validity of the result of a Monte Carlo simulation. Below is a table of the confidence levels with their corresponding width scaling coefficients, and the general form for a confidence interval with a confidence level of 99%.

| Confidence Level | 90% | 95% | 99% | 99.5% |
|---|---|---|---|---|
| Coefficient | 1.645 | 1.960 | 2.575 | 2.807 |

Table 3.1: Probability confidence coefficients

$$\hat{c} - \frac{2.575\hat{\sigma}}{\sqrt{N}} < c < \hat{c} + \frac{2.575\hat{\sigma}}{\sqrt{N}} \tag{3.32}$$

All the implementations in this project are based on Monte Carlo simulations. The methods can be broken down into two primary groups: Gaussian Walks and Tree-Based Walks.

### 3.4.1 Monte Carlo Gaussian Walks

This is a popular approach for calculating the value of Asian options [2]. The underlying walk simulates Brownian motion with a drift rate equal to the rate of interest and noise proportional to the square of the stock volatility. Figure 3.5 shows how the simulation becomes a more accurate

representation of a stock price as the step size decreases seen by the the increasingly more continuous appearance of the lines. Additionally the figure displays the resulting log-normal distribution of final prices of the stock, for the Gaussian walk the resulting distribution is not dependent on the number of steps taken in the walk.

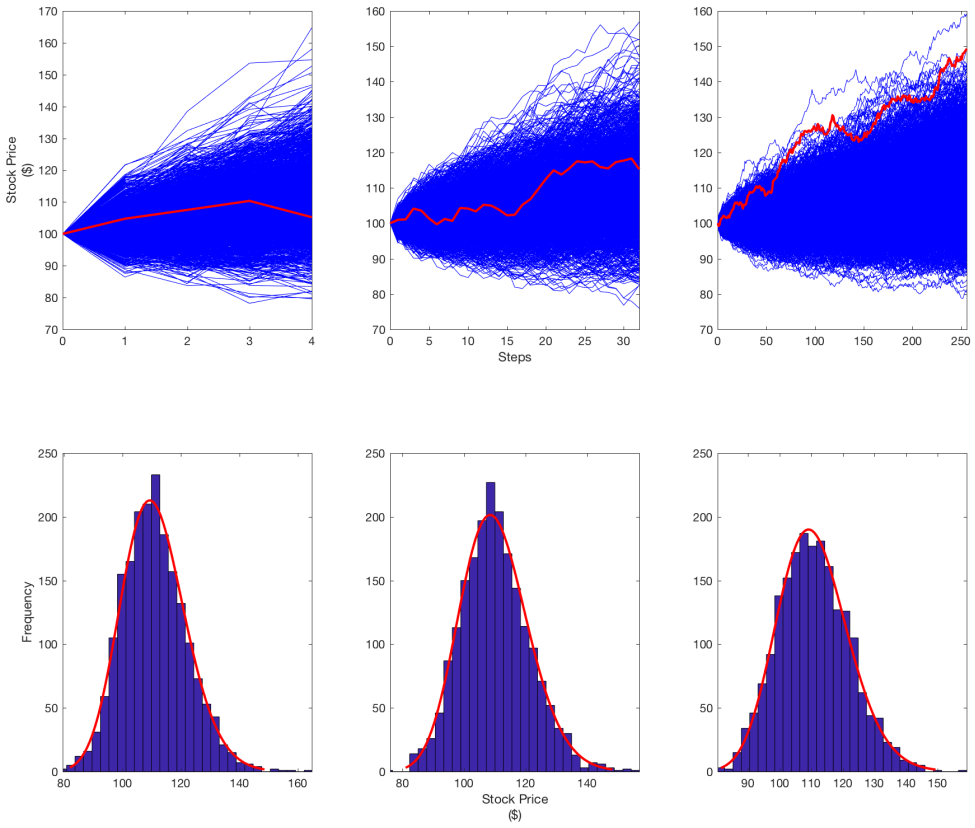



Figure 3.5: Monte Carlo gaussian walks for varying step sizes

The algorithm for Monte Carlo Gaussian pricing requires the properties of the stock such that the Gaussian Walk can be simulated. The average value of the stock is accumulated through the walk. This value is then used to calculate the value of the call option which is also accumulated. Once this process has been completed for all iterations the mean value of the call option is calculated which is then discounted to get the price of the option.

An alternative method for generating a Gaussian walk makes use of Brownian bridges to generate a random walk. The basis of a Brownian bridge is to calculate a coarse estimation of the walk then to generate a better estimation of the path by generating additional finer points in between the known points [14].

### 3.4.2 Monte Carlo Tree-Based Walks

This family of Monte Carlo simulations use a tree based model with discrete values that emulate Brownian motion. The tree can have multiple branches however standard model is a binomial tree. Figure 3.6 shows how the simulation becomes a more representative of the log-normal distribution of the stock price as the step size decreases. Notice that the range of end values is dependent on the number of steps taken. This is a result of the coarser approximation of Brownian motion by the underlying binomial model (Cox-Ross-Rubinstein).

**Algorithm 1** Gaussian Pricing Algorithm

**procedure** GAUSSIANMONTECARLO($S_0, K, r, \sigma, \Delta t, T, Steps, Iterations$)
  $payOffSum = 0$
  **for** `i=1 to Iterations` **do**
    $sum = S_0$
    $S = S_0$
    **for** `j=1 to Steps` **do**
      $W \leftarrow N(0,1)$
      $D \leftarrow (r - 0.5\sigma^2)\Delta t$
      $V \leftarrow \sigma\sqrt{\Delta t}b$
      $X \leftarrow D + W \times V$
      $S \leftarrow Se^X b$
      $sum \leftarrow sum + S$
    **end for**
    $L \leftarrow sum \div (Steps + 1) - K$
    $payoffSum+ = max(L, 0)$
  **end for**
  $P \leftarrow e^{-rT} \times payoffSum \div Iterations$
**end procedure**

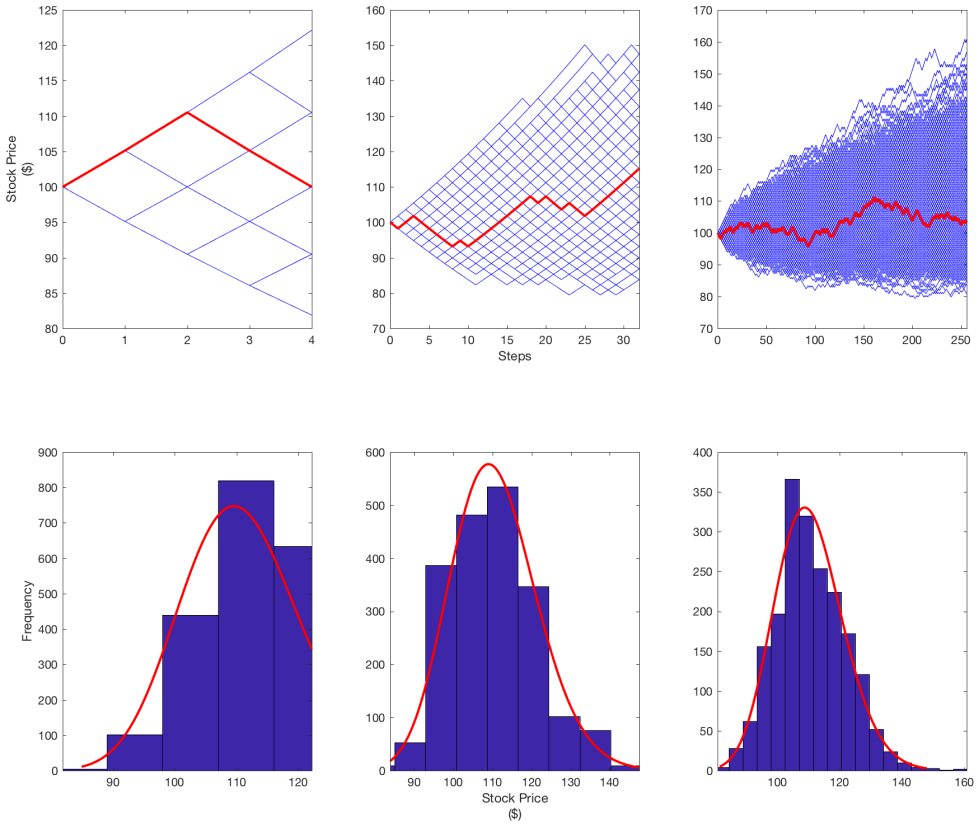



Figure 3.6: Monte Carlo binomial walks for varying step sizes

The algorithm for Monte Carlo Binomial pricing requires the up and down values of the tree along with the probability of going up for the appropriate step size. Like the Gaussian walk the average value of the stock is accumulated through the walk. This value is then used to calculate the value of

the call option which is also accumulated. Once this process has been completed for all iterations the mean value of the call option is calculated which is then discounted to get the price of the option.

**Algorithm 2** Binomial Pricing Algorithm

**procedure** BINOMIALMONTECARLO($S_0, K, u, d, p, T, Steps, Iterations$)
  $payOffSum = 0$
  **for** `i=1 to Iterations` **do**
    $sum = S_0$
    $S = S_0$
    **for** `j=1 to Steps` **do**
      $W \leftarrow U(0,1)$
      **if** `W` $\leq$ `p` **then**
        $S \leftarrow S \times u$
      **else**
        $S \leftarrow S \times d$
      **end if**
      $sum \leftarrow sum + S$
    **end for**
    $L \leftarrow sum \div (Steps + 1) - K$
    $payoffSum+ = max(L, 0)$
  **end for**
  $P \leftarrow e^{-rT} \times payoffSum \div Iterations$
**end procedure**

## 3.5 Multinomial Trees

Binomial trees are considered the *standard* tree based approach for calculating option prices however it is perfectly plausible to consider trees with more branches. The more branches the better a representation of the Gaussian walk the tree becomes however computationally this comes with an associated cost, typically an increase in memory and/or execution time.

A simple approach for generating a multinomial where the stock price distribution is available at every time is to generate the binomial lattice to match the distribution of the stock at every time step and then use the known probabilities to create a multinomial lattice based on the binomial lattice. [42]. For example consider taking two steps in a binomial tree, this has 4 potential routes that lead to 3 discrete end values. This information allows a trinomial tree to be expressed in terms of the half step size binomial tree coefficients: branch probabilities $p^2, 2p(1-p)$, and $(1-p)^2$ and branch multipliers $u^2, ud$, and $d^2$. A drawback of this approach for Asian options is that information about the path is lost in the process.

An example Trinomial tree that behaves similarly to two steps of Cox Ross Rubinstein model is [19]:

$$u = \sqrt{3\Delta t} \tag{3.33}$$

$$d = \frac{1}{u} \tag{3.34}$$

$$p_u = \sqrt{\frac{\Delta t}{12\sigma^2}}(r - \frac{\sigma^2}{2}) + \frac{1}{6} \tag{3.35}$$

$$p_m = \frac{2}{3}] \tag{3.36}$$

$$p_d = -\sqrt{\frac{\Delta t}{12\sigma^2}}(r - \frac{\sigma^2}{2}) + \frac{1}{6} \tag{3.37}$$

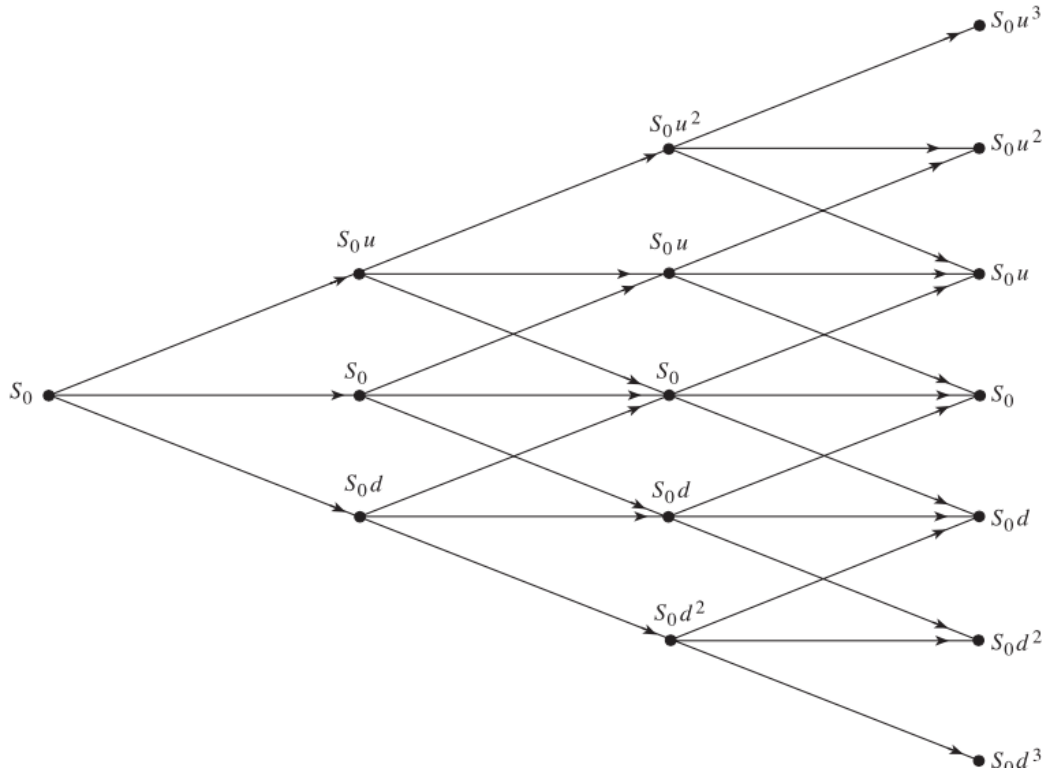



Figure 3.7: Trinomial tree [19]

It is possible to extend the number of branches further by using moment matching [35]. Moment matching is process of using specific sections from a standardized normal distribution such that the first, second, and potentially higher moments are matched.

Stratified sampling is a method of sampling representative values from an inverse cumulative normal distribution. It considers equal sized branches and sets the multipliers based on either mean or median of the associated probability distribution [19]. If the mean and variance of the model is slightly different to the actual expected probability distribution this can be fixed by multiplying through by a constant.

For a stratified normal sampled multinomial tree with $n$ branches, the $i^{th}$ branch multiplier would be calculated using the value:

$$N^{-1}(\frac{i-0.5}{n}) \tag{3.38}$$

The resulting value for 2 branches is in fact the Jarrow-Rudd binomial model.

## 3.6 Variance Reduction Techniques

In Monte Carlo simulations increasing the precision of the results by increasing the number of iterations is costly. An alternative approach is to consider methods that reduce the variance of the simulation. There exist a series of methods that enable simulations to have lower variances without the need to vastly increase the number of simulations.

### 3.6.1 Antithetic Variates

The antithetic variates method is a variance reduction technique dating back to 1956 [16]. The basis of the method is to generate a pair of random numbers simultaneously that result in random walks that are negatively correlated to one another [36]. For walks based on a random sample from a standard normal distribution an antithetic pair is $N(0,1)$ and $-N(0,1)$. For walks based on a uniform random number, an antithetic pair is $U(0,1)$ and $1-U(0,1)$. Figure 3.8 shows a antithetic pair of Gaussian walks.

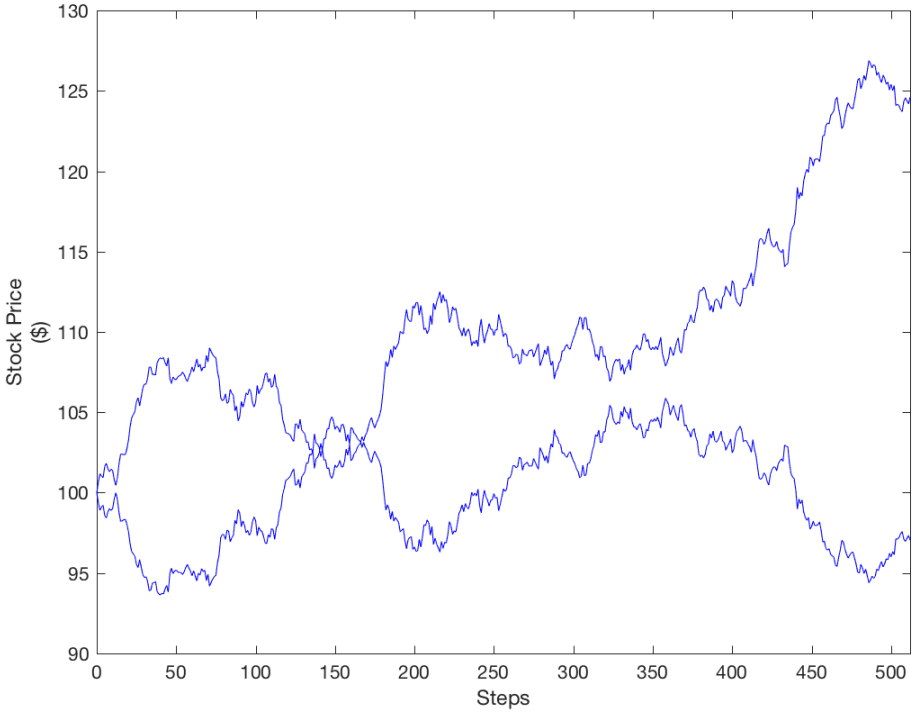



Figure 3.8: Pair of antithetic paths

Consider $X_i$, the distribution of average walk values. Assume that each path is independent. Let $n = 2m$, for some even $n \geq 2$

$$\bar{X}(n) = \frac{1}{2m}\sum_{i=1}^{m} X_i = \frac{1}{m}\sum_{i=1}^{m} Y_i = \bar{Y}(m) \tag{3.39}$$

where

$$Y_1 = \frac{X_1 + X_2}{2} \tag{3.40}$$

$$Y_2 = \frac{X_3 + X_4}{2} \tag{3.41}$$

$$\ddots$$

$$Y_N = \frac{X_{n-1} + X_n}{2} \tag{3.42}$$

Hence

$$E(Y) = E(x) = \mu \tag{3.43}$$

$$Var(Y) = \frac{1}{2}(\sigma^2 + Cov(X_1, X_2)) \tag{3.44}$$

In the case that pairs are uncorrelated the variance of Y is identical to that of X. However when a pair are negatively correlated the variance of the simulation will be reduced.

### 3.6.2 Control Variates

A control variate works using the opposite basis of correlation to that of the antithetic variate technique. It uses positive correlation to reduce the variance of a Monte Carlo simulation by using a correlated estimator that can be solved analytically.

Consider there are two similar derivatives but one has an analytical solution and the other does not; the two derivatives can be simulated using the same stochastic sequence and underlying model to get the Monte Carlo estimates for the two derivatives. The difference between the analytical result and the estimated result can be used to improve the accuracy of the derivative without an analytical solution. In the context of Asian options a similar derivative is a European option that can be calculated using the Black-Scholes equation or a Geometric Asian options [24].

The control variate estimate is defined as:

$$x_c = x + c(y - E(y)) \tag{3.45}$$

$$E[x_c] = E[x] \tag{3.46}$$

where $x_c$ is the control variate estimator, $x$ is the origianl estimator, $y$ is the correlated estimator, $E(y)$ is the analytical solution.

The coefficient c minimises the variance when it is equal to:

$$c = -\frac{Cov(x, y)}{Var(y)} \tag{3.47}$$

This allows for the calculation of the variance of the control variate technique as:

$$Var(x_c) = Var(x) - \frac{Cov(x, y)^2}{Var(y)} \tag{3.48}$$

The effectiveness of control variates in improving the rate of convergence of Monte Carlo Asian option pricing is quoted at 24x [2]. This can lead to a noticeably large speed up. The key factor determining the variance reduction of the control variance technique is the correlation between the key estimator and the control variate.

### 3.6.3 Multi-level Monte Carlo Simulations

Multi-level Monte Carlo (MLMC) variance reduction is a technique that is similar to the control variate approach. The method was first introduced by Heinrich in 2001 [18] and then generalised by Giles in 2008 [11] for stochastic differential equations. Rather than considering some similar derivative it considers an approximation that differs in accuracy and computational cost eg. different step size. The basis of the approach is to run a high number of cheap simulations and run more expensive simulations to counteract inherent biases of a smaller [12].

A two-level MLMC is very similar to the control variate method. Consider two estimators $P_0$ and $P_1$, where $P_0$ is a cheaper and less accurate estimator

$$E[P_1] = E[P_0] + E[P_1 - P_0] \tag{3.49}$$

which gives the general form for a two-level unbiased estimator for Monte Carlo methods,

$$\frac{1}{N_0}\sum_{n=1}^{N_0} P_0^{(n)} + \frac{1}{N_1}\sum_{n=1}^{N_1}(P_1^{(n)} - P_0^{(n)}) \tag{3.50}$$

where $P_1^{(n)} - P_0^{(n)}$ represents the difference between the two methods for the same stochastic sample.

According to literature [13] the ratio of $N_1$ to $N_0$ is :

$$\frac{N_1}{N_0} = \frac{\sqrt{\frac{V_1}{C_1}}}{\sqrt{\frac{V_0}{C_0}}} = \sqrt{\frac{V_1 C_0}{V_0 C_1}} \tag{3.51}$$

where $V$ is the variance and $C$ is the cost of computing.

The idea of two-level MLMC can be extended to multiple levels which takes the general form:

$$E[P_l] = E[P_0] + \sum_{l=1}^{L}(E[P_l - P_{l-1}]) \tag{3.52}$$

$$\frac{1}{N_0}\sum_{n=1}^{N_0} P_0^{(n)} + \sum_{l=1}^{L}[\frac{1}{N_l}\sum_{n=1}^{N_l}(P_l^{(l,n)} - P_{l-1}^{(l,n)})] \tag{3.53}$$

let

$$Y_L = \frac{1}{N_l}\sum_{n=1}^{N_l}(P_l^{(n)} - P_{l-1}^{(n)}) \tag{3.54}$$

For a fixed cost the variance is minimised at the point

$$N_l = \lambda \times \sqrt{\frac{V_l}{C_l}} \tag{3.55}$$

In order to achieve an overall variance of $\epsilon^2$ then

$$\lambda = \epsilon^{-2}\sum_{l=1}^{L}\sqrt{V_l C_l} \tag{3.56}$$

Giles puts forward this general algorithm for multilevel simulations:

1. Start with $L = 0$
2. Estimate $V_L$ using an initial $N_L = 10^4$ iterations
3. Define optimal $N_l$ , $l = 0, ..., L$
4. Evaluate extra samples as needed for new $N_l$

5. If $L \geq 2$, test for convergence using $max(S^{-1}|Y_{L-1}|, |Y_L|) < (S-1)\frac{\epsilon}{\sqrt{2}}$, with $S = 4$

6. If $L < 2$ or not converged, set $l := l + 1$ and go to 2

For a normal Monte Carlo method has complexity based on $\mathcal{O}(\epsilon^{-2})$ independent paths, each with $O(\epsilon^{-1})$ timesteps, giving a computational complexity which is $\mathcal{O}(\epsilon^{-3})$. Giles approach making use of Euler discretisations results in complexity $\mathcal{O}(\epsilon^{-2}\log 2)$, however when making use of Milstein Scheme he further reduces the complexity of the Monte Carlo simulation to $\mathcal{O}(\epsilon^{-2})$ [10].

## 3.7 Graphics Processing Units

### 3.7.1 Hardware

In the past performance gains in computing have been predominantly associated with the increasing number of transistors able to be fitted on an integrated circuit as well as their clock frequency. With a slowing of Moore's law and the era of peak frequency, hardware and software developers are increasingly looking to parallelism as a means of improving performance.

Graphics Processing Units (GPUs) were originally envisioned as a device to render graphics for computers. Due to the "embarrassingly" parallel nature of graphics based computation, the GPU has evolved with parallelism at the heart of its architecture. The organisation of each chip differs on a case by case basis however the general organisation of the cores is highly parallel in order to enable it to process large blocks of data in parallel.

Field Programmable Gate Arrays (FPGAs) are a popular alternative to GPUs in the world of High Performance Computing (HPC). FPGAs are a type of integrated circuit that allows developers to configure their purpose after their manufacture. The basic building block of an FPGA is programmable logic blocks that are highly interconnected that enables specialised processes to be built from them.

Fabry's project considered the finer points of applying his algortihm on an FPGA, however given that the GPU has different performance characteristics, different changes will need to be made. As such understanding what the nature of these differences is of paramount importance. Minhas considers comparing the performance characteristics of matrix multiplication for non-standard GPUs and FPGAs, he states that "the results showed that GPU implementations outperform FPGAs for larger data sizes but under-perform for smaller sizes where the memory latency and kernel start overhead become significant" [33]. Another takeaway from the paper is the better support on FPGAs for custom floating-point formats; this is a component of Fabry's implementation that could prove difficult to replicate efficiently on a GPU. A juxtaposition presented by Jones in 2010 [22] suggests GPUs are more *productive* than FPGA architectures for most benchmarks. It found that pseudo-random number generation (Mersenne Twister) was comparable in terms of performance increase over a CPU for the FPGA and GPU. This is a reassuring finding given the highly critical nature of pseudo-random number generation in Monte Carlo simulations. Furthermore it presents evidence suggesting the performance of floating point multiplication and scalar sums of vectors are greater than 3× faster on the GPU than the FPGA. In 2010, Betkaoui [3] found that GPUs usually enjoy a higher floating-point performance and memory bandwidth than FPGA-based HPC systems. He states that GPUs use GDDR memories which are optimized for sequential memory access operations, incurring a higher performance penalty for non-contiguous memory block transfers. This is unlikely to be a problem given the memory block transfers are entirely contiguous. The final conclusion of the paper is that the Monte Carlo benchmark for pricing Asian option is more efficient and demonstrates superior performance on the FPGA.

A benefit of using GPUs for High Performance Computing is that they're ready availability on cloud based platforms such as *Amazon Web Services*. These services allow users to make use of high performance GPUs based on a pay per use model that reduces the need for large overheads investing

in specialised hardware. The instances that will be used will be the P2 instances, designed for general-purpose GPU compute applications using CUDA and OpenCL, and G2 instances, optimized for graphics-intensive applications.

### 3.7.2 Software

The two leading options for writing software for GPUs are Khronos' OpenCL and NVidia's CUDA. Due to the more limited hardware options for CUDA, OpenCL was chosen for this project.

The C OpenCL API is a C with a C++ Wrapped API that is defined in terms of the C API. The OpenCL model portable, vendor and device-independent programs that can be accelerated on many difference hardware platforms such as GPUs and FPGAs. OpenCL begun as a graphics API however it has become particular popular for high performance computing .

OpenCL is defined in four segments called models:

1. Platform Model: Specifies there is one host and one or more processors (devices) able to execute the OpenCL Code.
2. Execution Model: How the OpenCL environment is configured on the host and how kernels are executed on the device
3. Memory Model: Defines the abstract memory hierarchy that the kernel uses. It closely resembles GPU memory hierarchy as shown in Figure 3.9
4. Programming Model: Defines how the concurrency model is mapped to physical hardware.

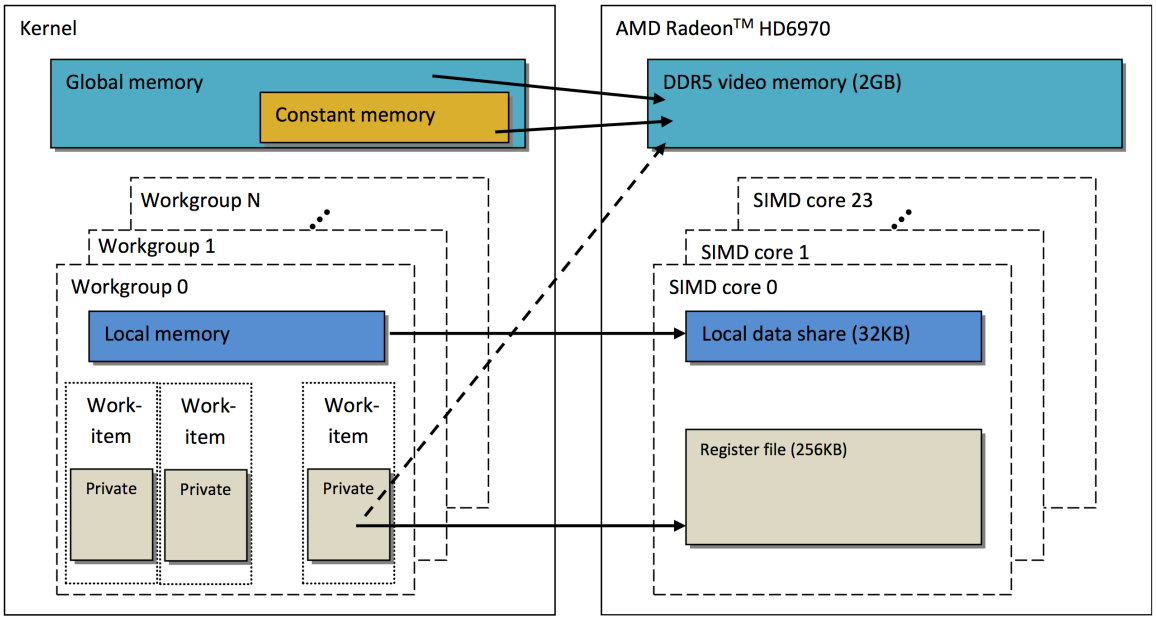



Figure 3.9: OpenCL memory model mapped to AMD Radeon 6970 GPU [23]

Compute-intensive and data-intensive portions of a given application, called kernels, are offloaded to the device. In order to execute a kernel on a device certain components need to be initialised on the host. A context is an abstract container that co-ordinates the interactions between the host and the device. A command queue is created for each device and the host calls on it when it needs an action be performed on by a device. Events are used to represent dependencies and provide a mechanism for profiling. The primary memory object in OpenCL is buffers, these enable the passing of data from the host context to the device.

Memory systems differ for each type of device however there is a singular abstract memory model that is used in OpenCL. Global memory is visible to all of the compute units on the device, it is used to pass memory between the host and the device; typically it is large in size however it is slower than other memory types available. Constant memory is used for data that needs to be accessed simultaneously by all work-items. Local memory is unique to each compute device, hence it is shared by a work-group. It is regularly smaller than global memory however it generally has less latency. Private memory is unique to each work-item, it is generally mapped to registers, a common use case is local variables [23].

## 3.8 Market Properties

In order to better understand the Monte Carlo random walks it is important to consider the range of values associated with the underlying stock is likely to take based on market conditions.

The two parameters that determine the nature of the path of a stock option are the interest rate and the implied volatility. These two parameters are beyond the control of the investor and are determined by the market and as such for the project a range of values for both are considered.

Calculating a reasonable range to for both can be deemed difficult due to the wide variety of stocks available. To reduce the range of values that the parameters could take the S&P 500 was considered as the basis for the values. The S&P 500 is a stock market index that represents approximately 80% coverage of available market capitalization in the United States of America [15]. Interest rates were considered as greater than 0% due to the issues associated with negative interest rates, and based on the on the best performer in 2016, Nvidia, which grew by 224% [37], a reasonable upper limit for the interest rate was calculated to be 300%. Easterly states than most period have volatility that falls in the bands of 10% to 20% [8]. When considering the worst and best performers over the past 20 years the range of volatilities the safely encompassed them were 0% to 250%.

$$0.05\% \leq r \leq 300\% \tag{3.57}$$

$$0.00\% \leq \sigma \leq 250\% \tag{3.58}$$

## 3.9 Random Number Generation

A Monte Carlo simulation needs to be able to generate sufficiently random numbers in order to simulate a large number of independent samples. In the field of computing it is difficult to calculate perfectly random numbers, consequently most random numbers are pseudo-random numbers. They share similar properties to true random numbers but are generated deterministically, where the next number is a mathematical function of the current number. A consequence of this determinism is the ability to generate the same sequence of results based on the same seed value. Due to the high number of random numbers required in a Monte Carlo simulation it is important to choose pseudo-random number generators that optimise the trade off between randomness and execution speed.

There are two types of random number generators that need to be used to implement the two variants of random walks. The Gaussian walk needs a normally distributed random number for each step, the discrete methods need a uniform random number generator.

There are a variety of uniform random number generators that are available to generate pseudo-random numbers each with different levels of randomness and execution characteristics. In order to test the randomness of numbers there exist a variety of tests, with popular test suites including Diehard and TestU01 [27, 30]. The two types of uniform random number generators considered for the scope of this project are XORShifts and Multiply-With-Carry (MWCs). XORShift generators are a type linear-feedback shift registers. They generate the next number by repeatedly taking the

exclusive or of a number with a bit-shifted version of itself [29]. The period of a XORShift generator is dependent on the number of bits in the state, the 128-bit state size XORShift generator passes all diehard tests, however, it fails the MatrixRank and LinearComp tests of the BigCrush test suite from the TestU01 framework. MWCs are based on arithmetic modulo mathematics [31]. For this project in particular the generator considered is Thomas' MWC64X generator due to its reported speed and statistical randomness [38].

In order to generate Gaussian random numbers Box and Muller method is considered as a reasonable approach [5, 39]. Although a dated approach to generating random normal numbers due to the increasing speed of approximate trigonometric and logarithmic functions on GPUs it remains a popular method. The basic form as given by Box and Muller takes two samples from the uniform distribution and maps them to two standard normally distributed samples.

**Algorithm 3** Box-Muller Algorithm

1: **procedure** BoxMuller( )
2: $u_1 \leftarrow U(0,1)$
3: $u_2 \leftarrow U(0,1)$
4: $a \leftarrow \sqrt{-2\log(u_1)}$
5: $b \leftarrow 2\pi u_2$
6: $normal_1 = a\sin(b)$
7: $normal_2 = a\cos(b)$
8: **end procedure**

The Quasi-Random Monte Carlo method is a particular variant of the Monte Carlo simulation that do not use pseudo random numbers but rather considers using well-chosen deterministic points that represent a more balanced representation of the iteration space [34]. The use of Quasi-Random Monte Carlo methods helps to improve the convergence rates of simulations Monte Carlo simulations in particular for Asian option pricing [6, 26].

# 4 Requirements Capture

As stated in the Project Specification, the primary objective of the project is to investigate the properties associated with discrete time and discrete space pricing models for Asian Options on a GPU. The background presented allows for further clarification of the approach to answering the guiding questions of the project as well providing clearer guidelines on the expectations and deliverables of the project.

1. How do the convergence properties of discrete space models differ from the Gaussian continuous space model?

   A basic level of understanding needs to be acquired in the difference and similarities in convergence of the discrete and continuous space methods in order to present an appropriate basis that allows for the comparison of the two. The convergence properties of the models will be examined by considering the Root-Mean-Square Error for different Steps and Iteration spaces, *NM Space*. This will inform the basis of the precision and accuracy of the model which in turn will provide a basis for the comparison of different models, execution time for equal confidence interval sizes.

2. How should the algorithms be parallelised and split across compute units?

   Each pricing method has different performance characteristics which in turn will result in different approaches to splitting the computation of the simulation. Determining the answer to this question will require side-by-side comparisons of similar implementations with singular differences to determine the impact of the changes.

3. What advantages of Fabry's discrete time and discrete space model on an FPGA are directly applicable to GPUs?

   This question provides an important entry point to understand the performance characteristics of GPUs and informs the subsequent sections of the project. Particular elements of Fabry's approach such as using the fixed point calculations, pre-calculated stock values, and others must be justified for the GPU architecture otherwise must be removed in the pursuit of an optimised solution.

4. What algorithmic changes should be made when targeting a GPU?

   Following on from the previous question, understanding how to alter the models to increase their throughput on GPUs is critical in maximising their performance. Considerations that can be made include size of numbers, random number generation, maximising caching, memory management, etc. On the basis of the background research the generation of random numbers will be beyond the scope of the project, some fundamental analysis will be carried out however no attempts to create a new random number generator will be considered.

5. What improvements in performance and accuracy can be obtained using a non-recombinant models on a GPU?

   An essential component of Fabry's optimisations for FPGAs was using the LUT to store the discrete space the stock path could take. It is as such of interest to consider whether dynamically generating discrete space models rather than storing it in memory is an appropriate approach

to optimise the pricing methods. The initial entry point of this investigation is a comparison for the Cox-Ross-Rubinstein binomial model to determine whether multiplicative as opposed memory based operations are advantageous, this will characterise the focus of the subsequent variance reduction techniques.

6. What improvements in performance and accuracy can be obtained using antithetic sampling, control variates, multi-level, and Quasi-Monte Carlo method on a GPU?

   (a) The use of antithetic sampling appears to have strong foundations in the continuous model however it needs to be seen whether the negative correlation of this is applicable to the discrete space models.

   (b) The use of control variates is a highly popular variance reduciton technique for Monte Carlo simulations. It needs to be seen the difference in computational complexity and improvements in performance for European and Geometric control variates.

   (c) Multi-level simulations fall within the remit of the project however it is not clear as to whether there is an appropriate approach for implementing such an approach in discrete space due to the dependence of shared stochastic paths between simulation levels.

   (d) Quasi-Monte Carlo methods are beyond the scope of this project. It is noted that their improvements in performance and accuracy are significant however due to their dependence on Gaussian bridges, developing an alternative approach for discrete models is deemed too complex given the time constraints of the project.

7. Of the methods examined which has the best execution properties for a GPU and how does this compare to Fabry's FPGA approach?

   A comparison between the different Monte Carlo simulation can be carried out by considering execution time for like accuracies, and step throughput. These figures can then be used to determine the most appropriate solution for GPUs, as well as providing substantial evidence as to whether or not it is an appropriate model when comparing its performance with existing FPGA solutions.

In order to answer these questions, the following deliverables are presented:

1. Development of Test Framework
2. NM Space Convergence Models
3. Variance Reduction Analysis
4. GPU Optimised Monte Carlo Methods for:
   (a) Cox-Ross-Rubinstein Binomial Trees
   (b) Gaussian Walks
   (c) Jarrow-Rudd Binomial Trees
   (d) Multinomial Trees
   (e) Antithetic Variates
   (f) Control Variates
   (g) Multi-level Approach

# 5 Model Convergence Analysis

In this chapter the convergence properties of the models are examined through the scope of Step-Iteration space (*NM Space*). This analysis provides an understanding of the required number of steps and iterations that have to be executed to achieve a given level of expected accuracy. The option pricing convergence models considered in this chapter are:

1. Cox-Ross-Rubinstein Binomial Trees
2. Gaussian Walks
3. Jarrow-Rudd Binomial Trees
4. Cox-Ross-Rubinstein Compressed Multinomial Trees
5. Jarrow-Rudd Compressed Multinomial Trees
6. Normal Sampled Multinomial Trees

The generation and execution properties of these models for GPUs is considered in more depth in Chapter 8.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{K}\sum_{m=1}^{K}(c - T_r)^2} \tag{5.1}$$

To assess the quality of a simulation across a range of iteration and step sizes the Root-Mean-Square Error (RMSE) is used. It is a standard statistical measure applied to Monte Carlo simulations; it measures the differences between values predicted by a model and the values actually observed. This measure is preferable to the Mean-Square Error (MSE) as the units are common with the Call Price improving the interpretability of the result. Defining the true value for an Asian option is difficult due to the absence of a analytical formula for calculating its value. To determine the true value of an Asian option a Gaussian Monte Carlo simulation was used with 500,000 steps and 500,000,000 iterations. For the *NM spaces* considered, each point was generated by calculating 100 residuals and taking their mean to minimise the sampling noise. To enable direct comparisons to be drawn across NM Spaces for different models the same option parameters were considered, $S_0 = 100, K = 110, \sigma = 0.3, r = 0.15$, and $T = 1$. From the massive simulation carried out the true value of the call price of this option is \$5.73007.

## 5.1 Cox-Ross-Rubinstein Binomial Trees

This model was the basis of Fabry's FPGA simulation, it is important to consider its standalone properties in *NM Space*, as to better understand the limitations of the model.

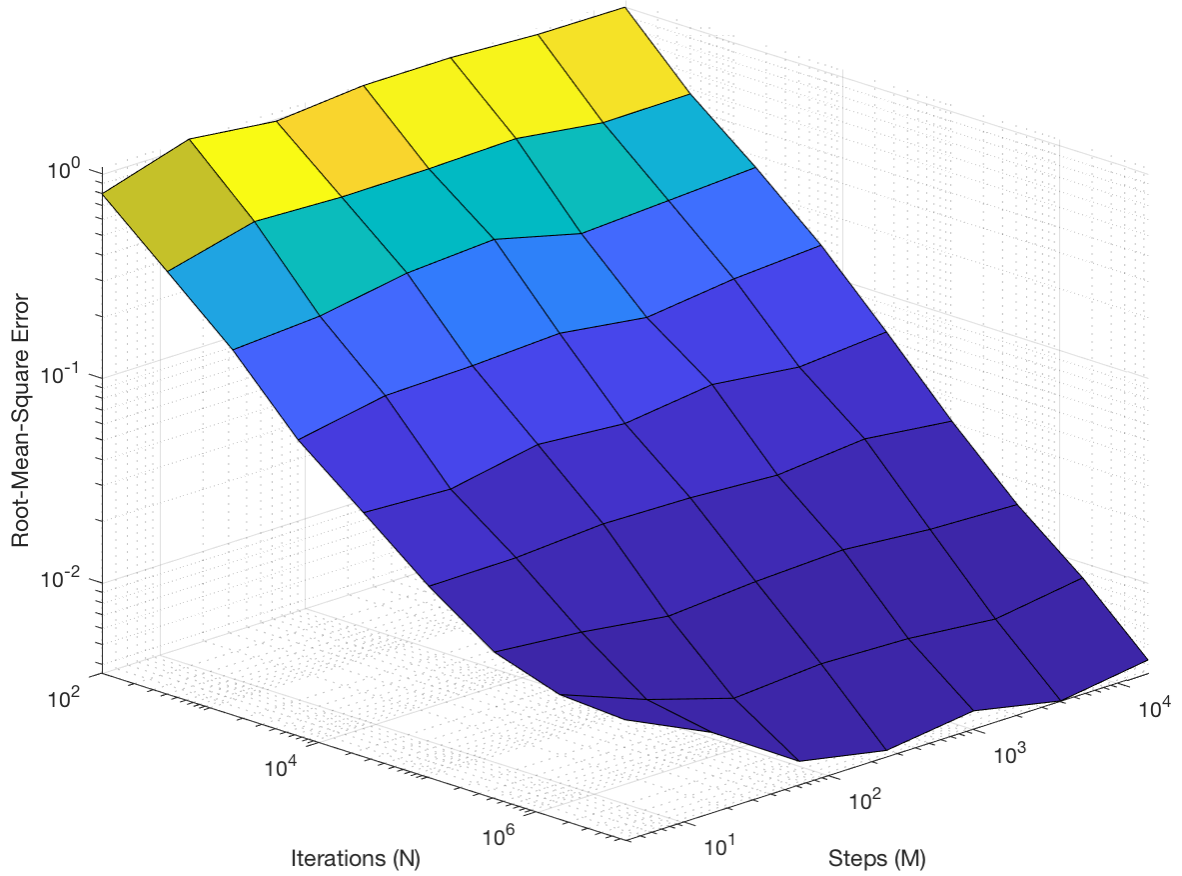
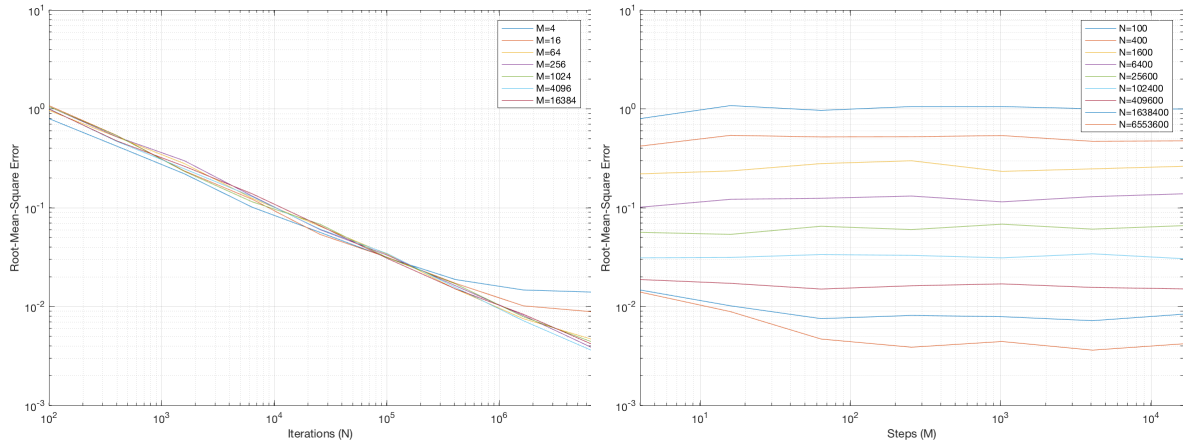



Figure 5.1: Root-Mean-Square Error of CRR model in *NM Space*

Figure 5.1 displays the expected inverse proportionality between the square root of iterations and the RMSE predicted by central limit theorem. It also shows the absence of accuracy gains from increasing the number of steps.




(a) RMSE against Iterations

(b) RMSE against Steps

Figure 5.2: Root-Mean-Square Error of CRR model in 2D

As seen from splitting the 3-dimensional model in Figure 5.1 in Figure 5.2 there are limited gains to increasing step sizes for improving the accuracy of pricing an Arithmetic Asian option. However it appears that simulations with a small number of steps and large iterations begin to peel off from the

trend line for for RMSE. This warrants further investigation into the the absolute convergence of the Cox-Ross-Rubinstein model to determine whether there is an inherent bias for small binomial trees that limits their ability to accurately represent the continuous model of a stock.

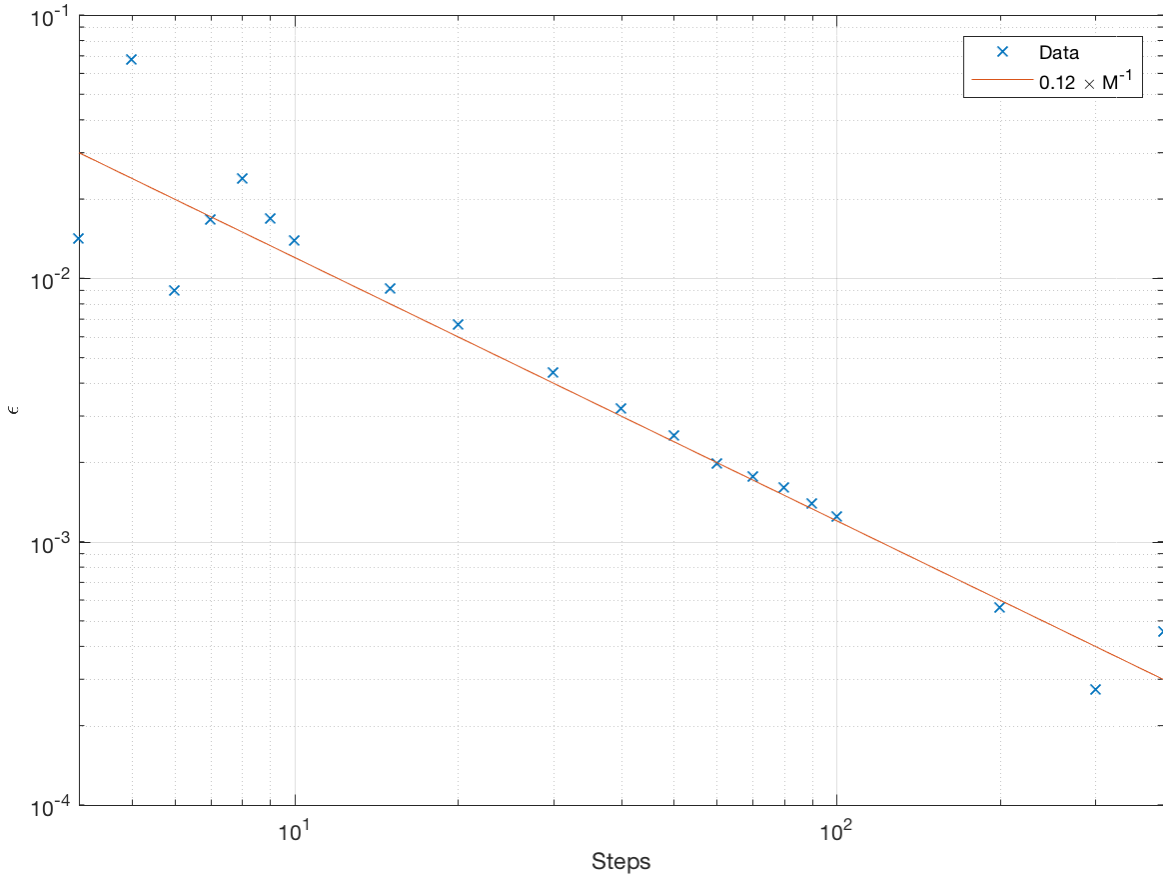



Figure 5.3: Absolute error convergence of CRR model

Figure 5.3 was generated using 10,000,000,000 iterations to see the inherent bias of the model for small tree sizes. As the number of steps increases, the absolute error of the model is reduced. It is determined that for the Cox-Ross-Rubinstein binomial model the absolute error of the tree is:

$$\epsilon = 0.12 \times M^{-1} \tag{5.2}$$

From Figure 5.2 it is seen that the bias of the model with $4 \times 10^0$ steps begins to break the expected convergence of increasing iterations at the point $2 \times 10^5$. To ensure that the bias remains negligible beyond this point the absolute error must decrease at the same rate as the RMSE from a point at which the model behaves correctly. This ensures that the bias will not have a noticeable effect on the confidence interval of the model.

The line of best fit for the iterations convergence is:

$$\epsilon = 10N^{-\frac{1}{2}} \tag{5.3}$$

Equating epsilons gives:

$$M \approx 1.2 \times 10^{-2} N^{\frac{1}{2}} \tag{5.4}$$

Hence the CRR Binomial Monte Carlo model has complexity:

$$\mathcal{O}(NM) = \mathcal{O}(NN^{\frac{1}{2}}) = \mathcal{O}(\epsilon^{-2}(\epsilon^{-2})^{\frac{1}{2}}) = \mathcal{O}(\epsilon^{-3}) \tag{5.5}$$

## 5.2 Gaussian Walks

Understanding the performance of the discrete models requires an understanding of the performance of the standard continuous Gaussian model used to price Asian options.

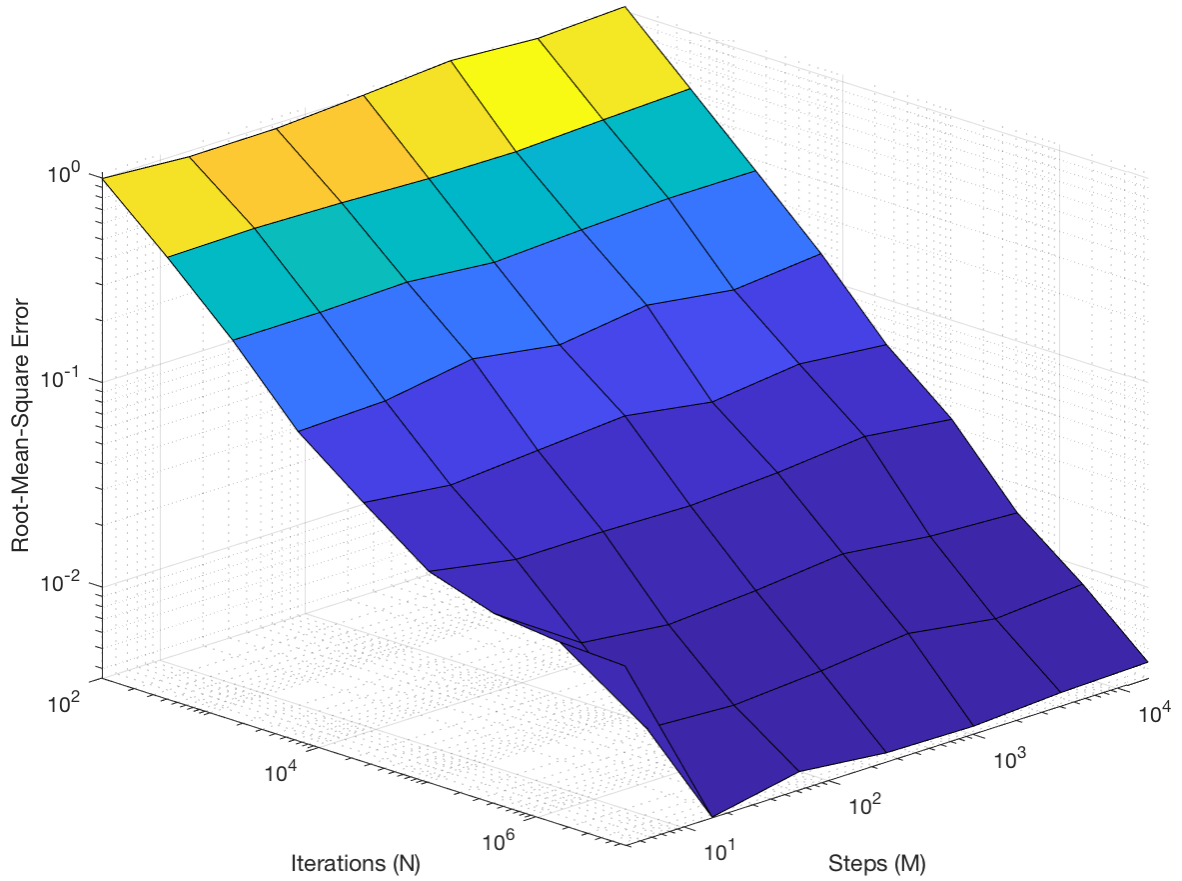
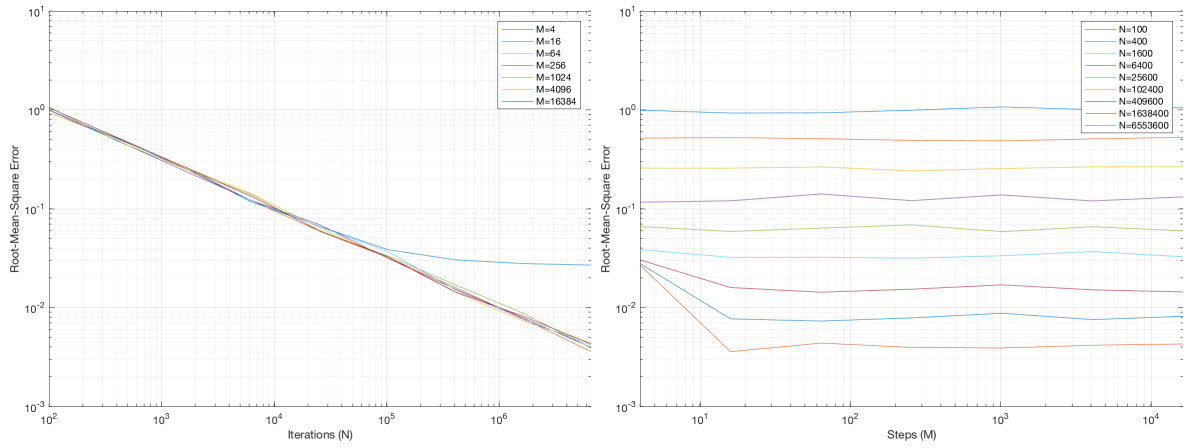



Figure 5.4: Root-Mean-Square Error of Gaussian model in *NM Space*




(a) RMSE against Iterations

(b) RMSE against Steps

Figure 5.5: Root-Mean-Square Error of Gaussian model in 2D

As seen from splitting the Figure 5.4 into Figure 5.5 the similarities to the Cox-Ross-Rubinstein model

are apparent with the same anticipated relationship between increasing iterations and decreasing root-mean-square error. It is noted that for the smallest number of steps considered, 4, the Gaussian begins to break from the standard convergence for increasing iterations; the break in the relationship occurs at smaller number of iterations than the CRR model; however the subsequent number of steps, 16 does not break from the convergence trend suggesting that the model is less dependent on the number of steps than the binomial model.

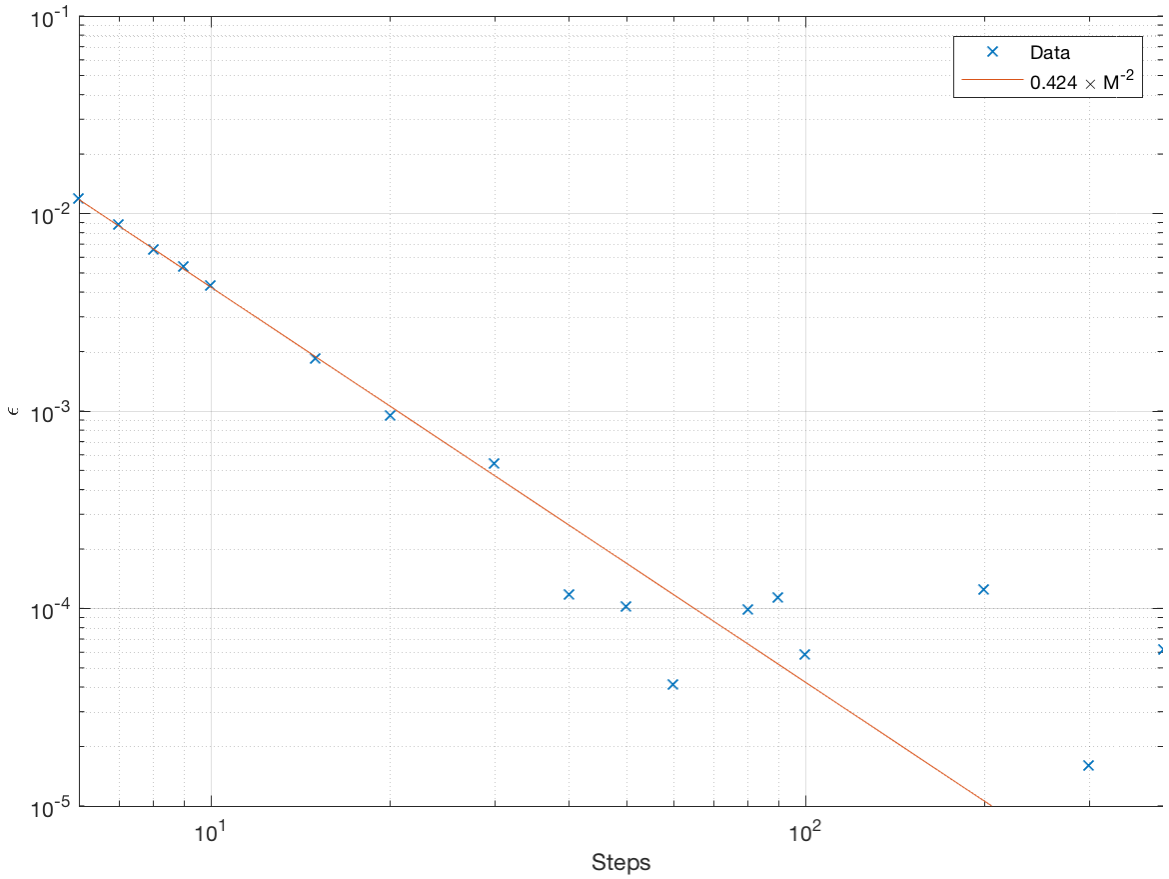



Figure 5.6: Absolute error convergence of Gaussian model

From Figure 5.5 it is seen that the model starts to break the convergence trend with $4 \times 10^0$ steps and $10^5$ iterations. The same methodology for evaluating the bias for the Cox-Ross-Rubinstein model is considered for the Gaussian model to determine the required number of steps to limit the impact the bias on the confidence level as shown in Figure 5.6. The distance between the data points between the line of best fit grow with the number of steps taken however this is due to the absolute error being comparable to the confidence interval size of the simulation as well as floating point error becoming substantial.

$$\epsilon = 0.424 \times M^{-2} \tag{5.6}$$

The line of best fit for the iterations convergence is:

$$\epsilon = 10N^{-\frac{1}{2}} \tag{5.7}$$

Equating epsilons gives:

$$M \approx 2.059 \times 10^{-1} N^{\frac{1}{4}} \tag{5.8}$$

Hence the Gaussian model has complexity:

$$\mathcal{O}(NM) = \mathcal{O}(NN^{\frac{1}{4}}) = \mathcal{O}(\epsilon^{-2}(\epsilon^{-2})^{\frac{1}{4}}) = \mathcal{O}(\epsilon^{-\frac{5}{2}})$$

This complexity of the Gaussian and the Cox-Ross-Rubinstein models differ hence suggesting for small confidence sizes the Gaussian will outperform the Cox-Ross-Rubinstein model.

## 5.3 Jarrow-Rudd Binomial Trees

The Jarrow-Rudd model is the primary alternative to the Cox-Ross-Rubinstein model for binomial tree based pricing. It is a potentially less computationally intensive alternative, however before this performance difference cannot be considered before the difference in convergence properties.

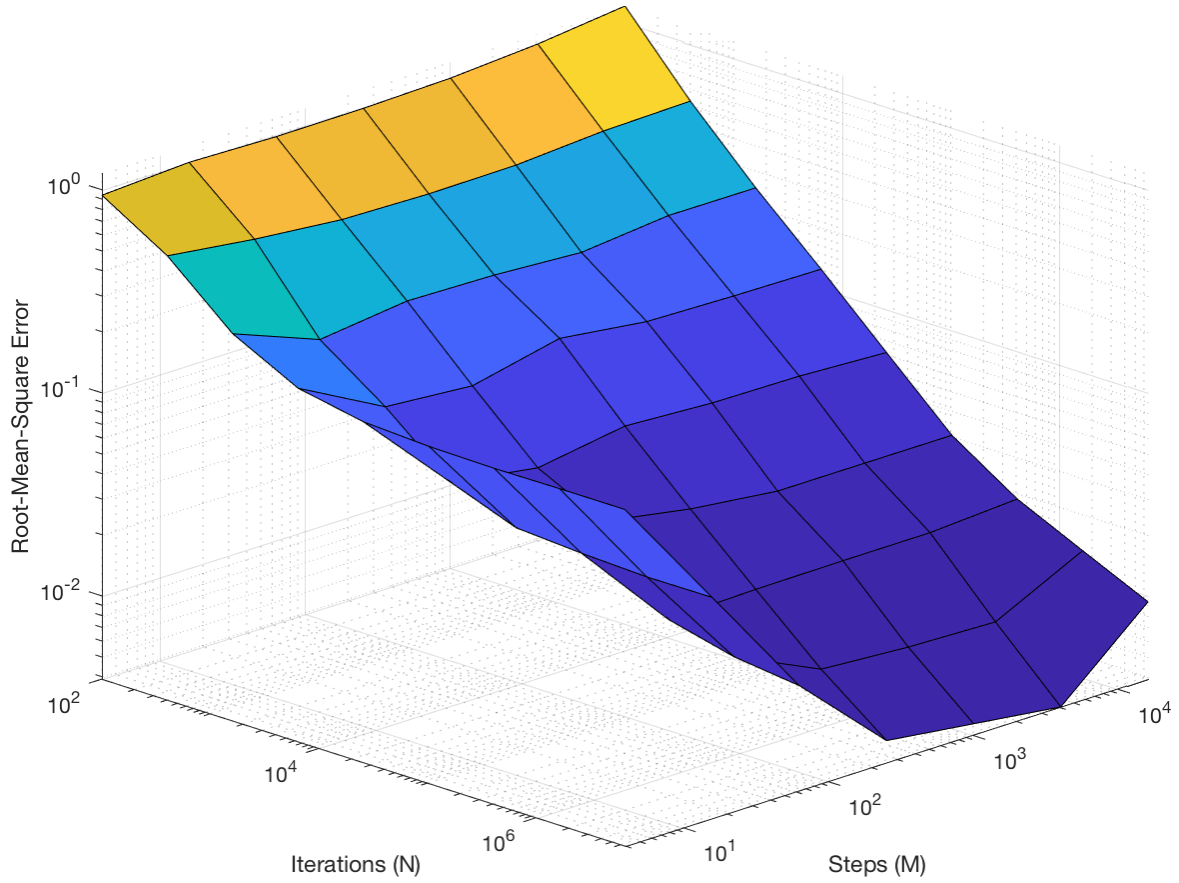
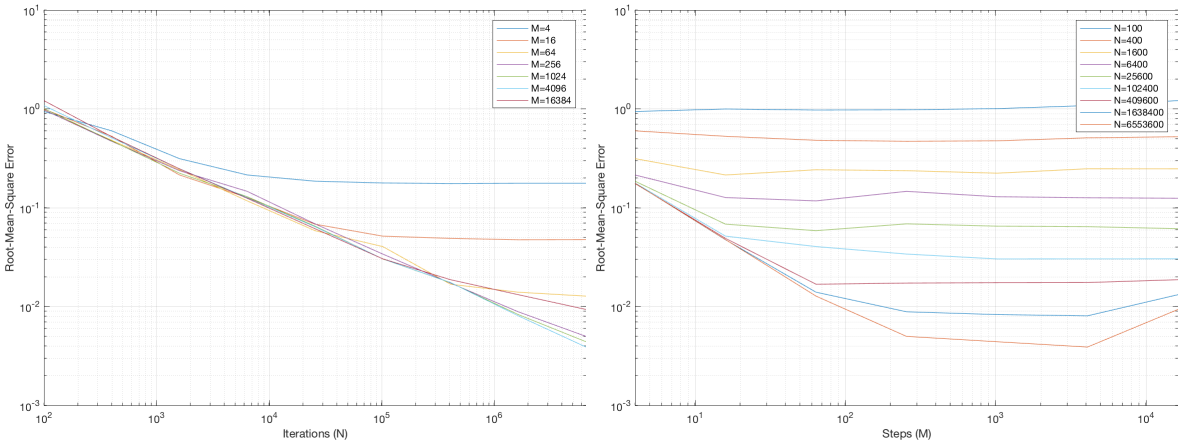



Figure 5.7: Root-Mean-Square Error of JR Model in *NM Space*

Figure 5.7 is similar in appearance to the Cox-Ross-Rubinstein model, however the levels or error caused by small number of steps is distinctly greater and occurs at a smaller number of iterations. There is a slight limitation of the convergence of the model at the maximum steps considered, this is due to the quantisation error of the floating point calculations that begin earlier than in the CRR model; this error is discussed in more detail in Chapter 8.




(a) RMSE against Iterations

(b) RMSE against Steps

Figure 5.8: Root-Mean-Square Error of JR model in 2D

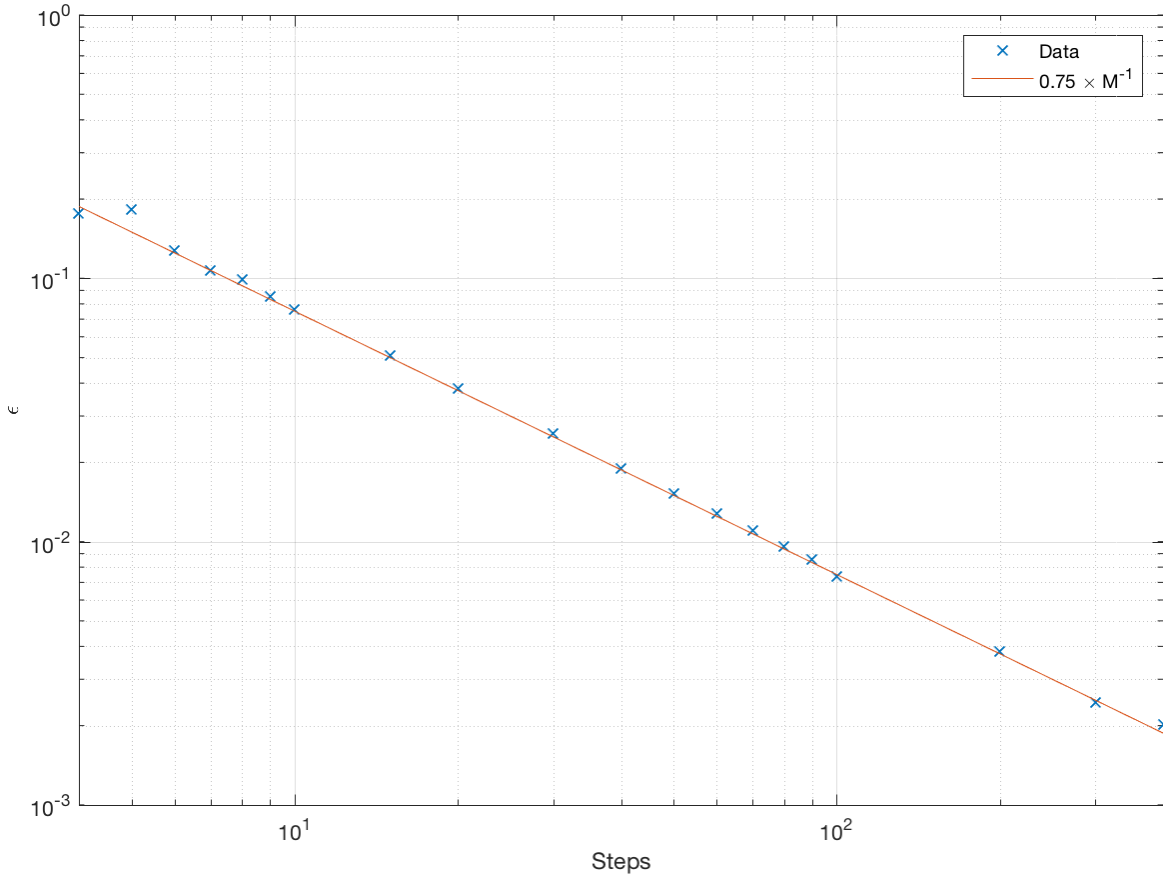



Figure 5.9: Convergence of JR model to true value

Figures 5.8 and displays the break in the iterations convergence with $4 \times 10^0$ Steps and $4 \times 10^3$ Iterations. The same methodology for evaluating the bias for the Cox-Ross-Rubinstein model is considered for the Jarrow-Rudd model to determine the required number of steps to limit the impact the bias on the confidence level as shown in Figure 5.9.

$$\epsilon = 0.75 \times M^{-1} \tag{5.9}$$

The line of best fit for the iterations convergence is:

$$\epsilon = 10N^{-\frac{1}{2}} \tag{5.10}$$

Equating epsilons gives:

$$M \approx 7.5 \times 10^{-2} N^{\frac{1}{2}} \tag{5.11}$$

Hence the CRR Binomial Monte Carlo model has complexity:

$$\mathcal{O}(NM) = \mathcal{O}(NN^{\frac{1}{2}}) = \mathcal{O}(\epsilon^{-2}(\epsilon^{-2})^{\frac{1}{2}}) = \mathcal{O}(\epsilon^{-3}) \tag{5.12}$$

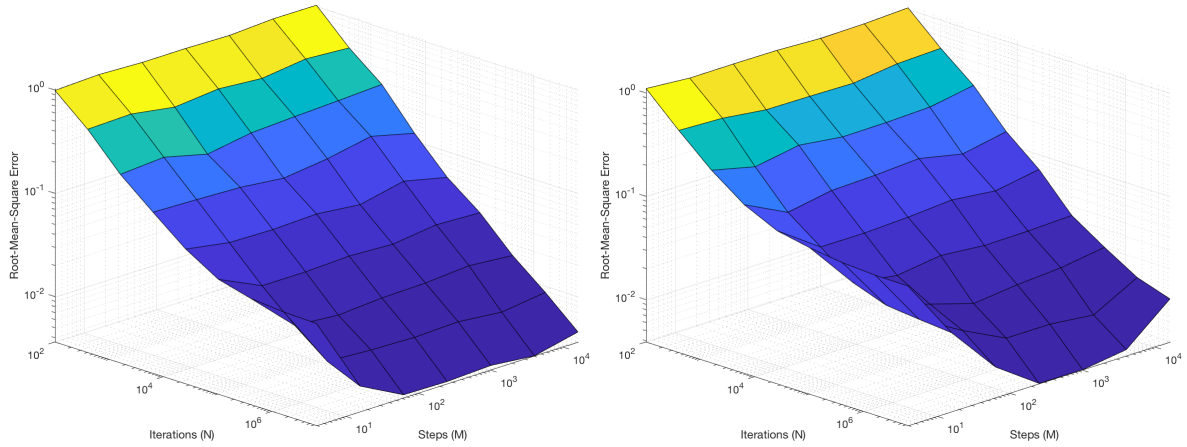
This suggests that the overall complexity of the binomial Jarrow-Rudd model is equivalent to that of the Cox-Ross-Rubinstein model, however it is noted that the number of steps required to achieve convergence is scaled to a greater value.

## 5.4 Multinomial Trees

This section is broken into two sub-variants of multinomial trees. Compressed multinomial trees represent the category of trees generated by considering the implied multipliers and probabilities of taking several smaller binomial steps using Cox-Ross-Rubinstein or Jarrow-Rudd models. Normal sampled multinomial trees represent those generated from using the stratified sampling method to generate the discrete set of multipliers.

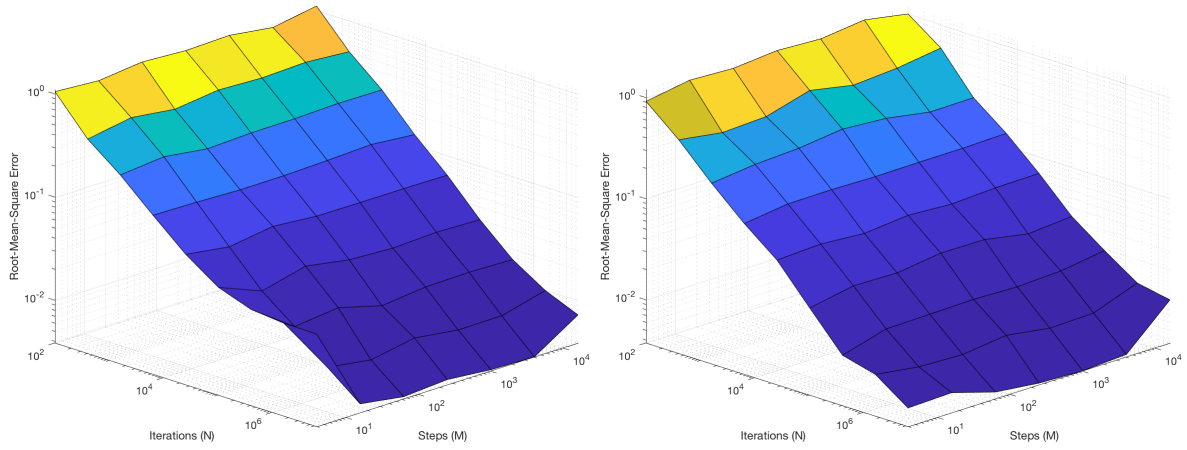
### 5.4.1 Compressed Multinomial Trees

To initiate the investigation into the improvements resulting from compressed multinomial trees, four variants of the tree are presented: two Cox-Ross Rubinstein based models, and two Jarrow-Rudd based models.




(a) Trinomial CRR (b) Trinomial JR

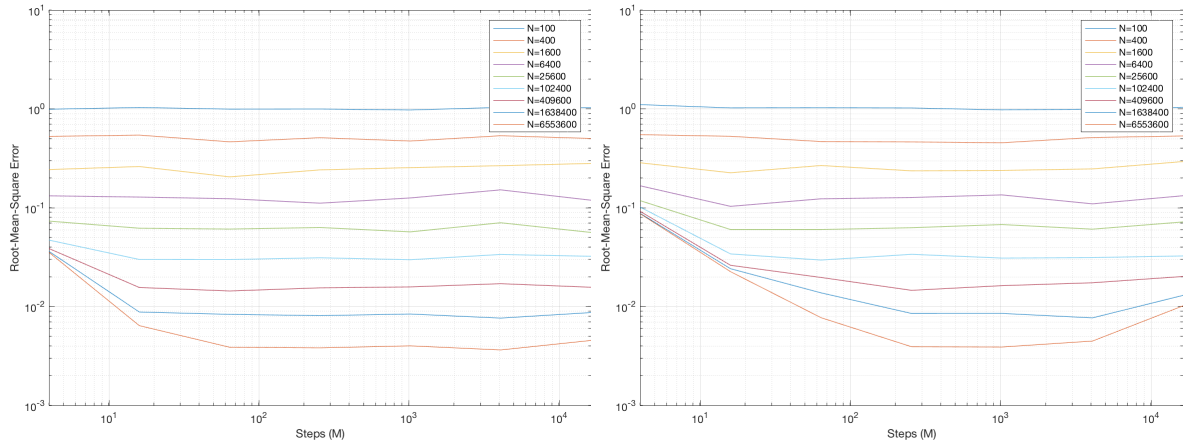
Figure 5.10: Root-Mean-Square Error of trinomial models in *NM Space*




(a) Octanomial CRR (b) Nonanomial JR

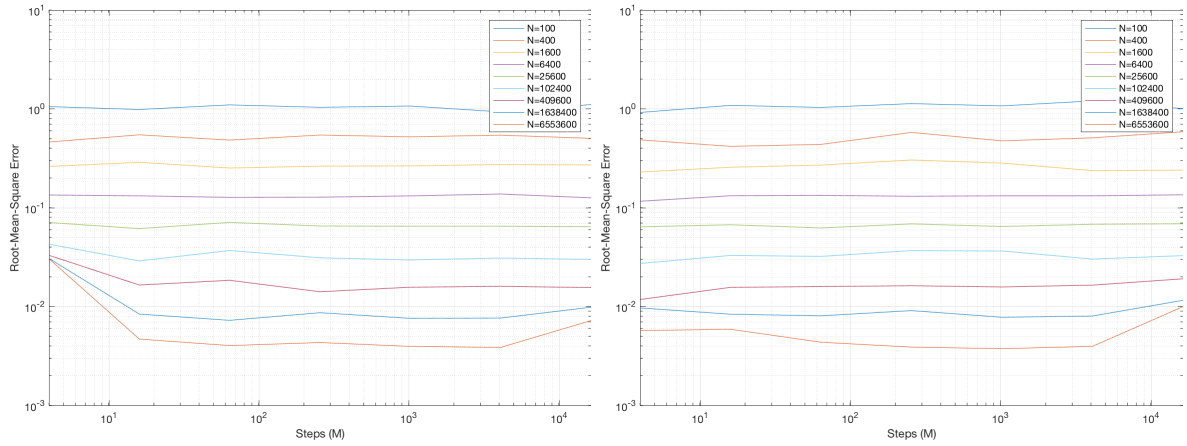
Figure 5.11: Root-Mean-Square Error of multinomial models in *NM Space* showing the improvement of convergence for greater number of branches

Figures 5.10 and 5.11 exhibit the overall convergence pattern of the compressed multinomial trees, which is resemblant of the underlying binomial models. The difference between the binomial and the trinomial models is minor however the larger octanomial and nonanomial models show markedly better convergence properties for fewer steps.



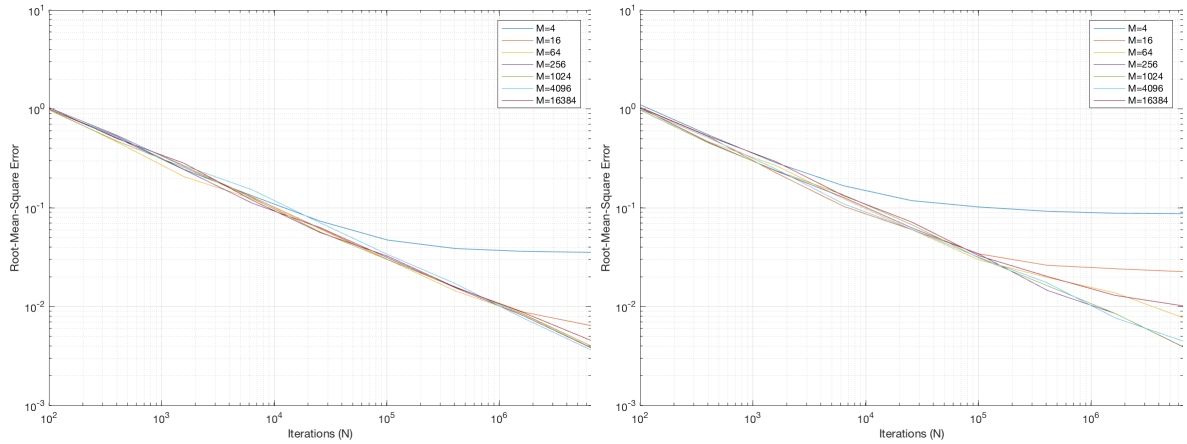

(a) Trinomial CRR (b) Trinomial JR

Figure 5.12: Root-Mean-Square Error of multinomial models against steps



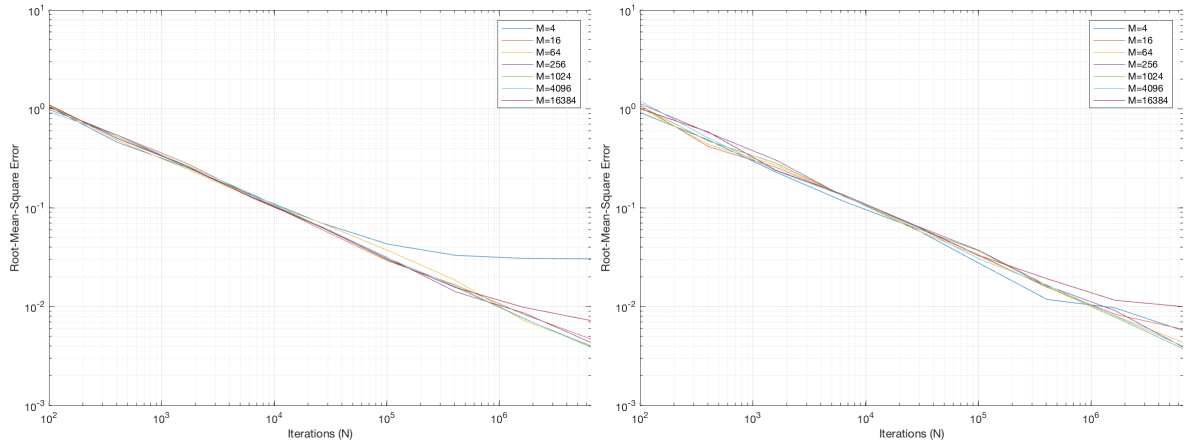

(a) Octanomial CRR (b) Nonanomial JR

Figure 5.13: Root-Mean-Square Error of multinomial models against steps




(a) Trinomial CRR (b) Trinomial JR

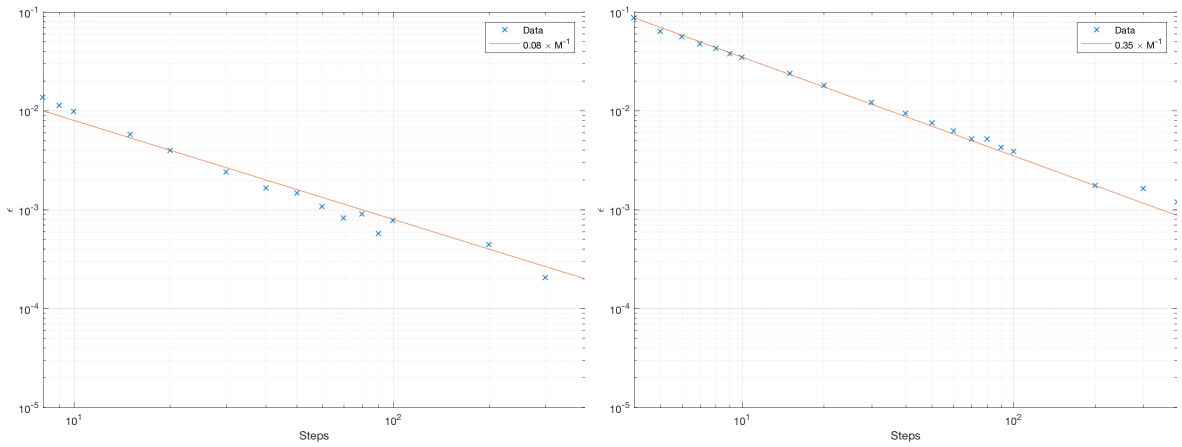
Figure 5.14: Root-Mean-Square Error of trinomial models against iterations




(a) Octanomial CRR (b) Nonanomial JR

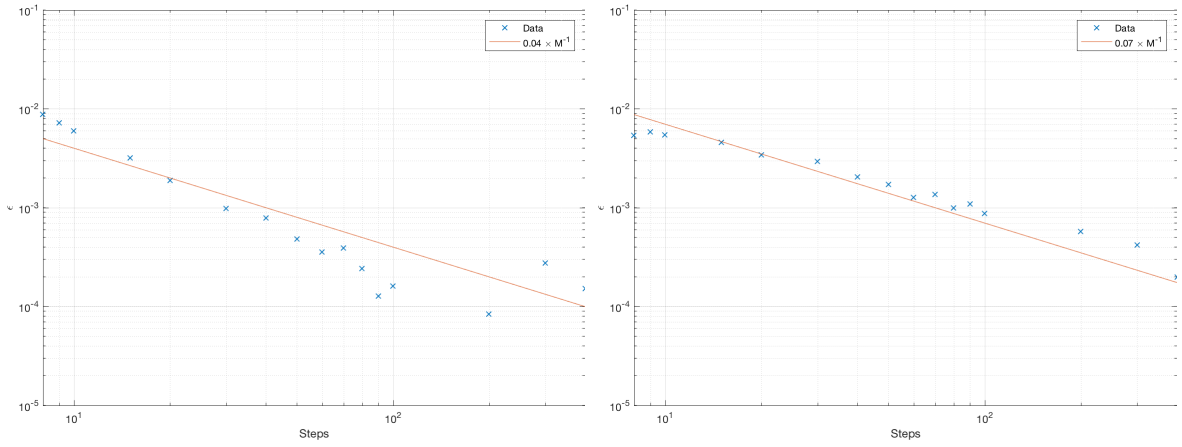
Figure 5.15: Root-Mean-Square Error of multinomial models against iterations

Figures 5.12 , 5.13 ,5.14 , and5.15 display the iterative improvements of increasing the number of branches of the compressed multinomial model. This is seen by the better convergence for fewer steps as highlighted by the delayed point at which the *NM Space* breaks the iterations relationship.




(a) Trinomial CRR (b) Trinomial JR

Figure 5.16: Convergence of trinomial models to true value




(a) Octanomial CRR (b) Nonanomial JR

Figure 5.17: Convergence of multinomial models to true value

Figures 5.16 and 5.17 present that as the number of branches increases in the multinomial tree, the tree becomes a better representation of the Gaussian movement of the underlying stock for each step. The rate of convergence to the true value is the same as the binomial models however the scaling of the number of steps required is fewer. The Octanomial CRR approach to the true value against steps is modelled as an inverse relationship for simplicity, but if the two largest step values are ignored the relationship could potentially represent a change in the order of the convergence.

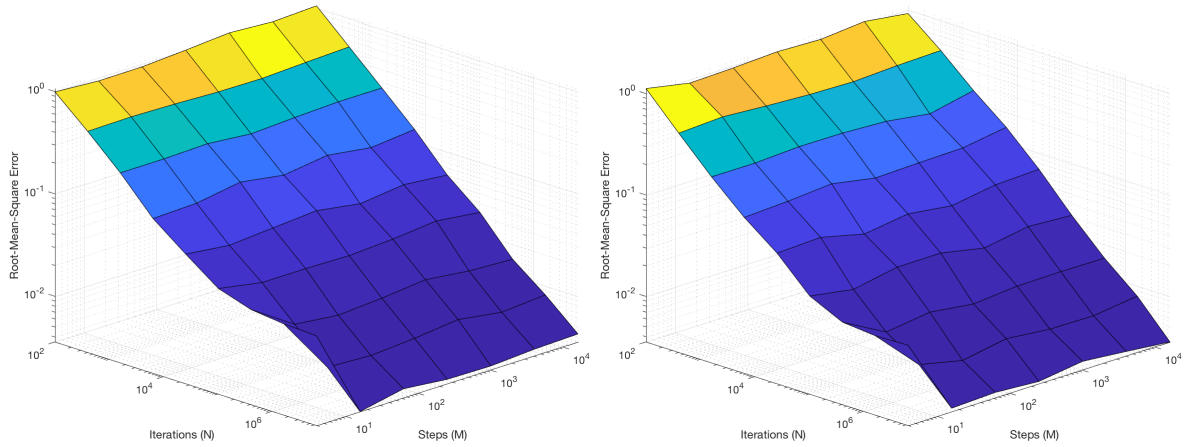
Table 5.1 follows approach used in previous sections to calculate the *NMSpace* properties for the compressed multinomials.

| Trinomial CRR | Trinomial JR | Octanomial CRR | Nonanomial JR |
|---|---|---|---|
| $\epsilon = 0.08 \times M^{-1}$ | $\epsilon = 0.35 \times M^{-1}$ | $\epsilon = 0.04 \times M^{-1}$ | $\epsilon = 0.07 \times M^{-1}$ |
| $M \approx 8 \times 10^{-3} N^{\frac{1}{2}}$ | $M \approx 3.5 \times 10^{-2} N^{\frac{1}{2}}$ | $M \approx 4 \times 10^{-3} N^{\frac{1}{2}}$ | $M \approx 7 \times 10^{-3} N^{\frac{1}{2}}$ |
| $\mathcal{O}(\epsilon^{-3})$ | $\mathcal{O}(\epsilon^{-3})$ | $\mathcal{O}(\epsilon^{-3})$ | $\mathcal{O}(\epsilon^{-3})$ |

Table 5.1: Multinomial *NM Space* properties

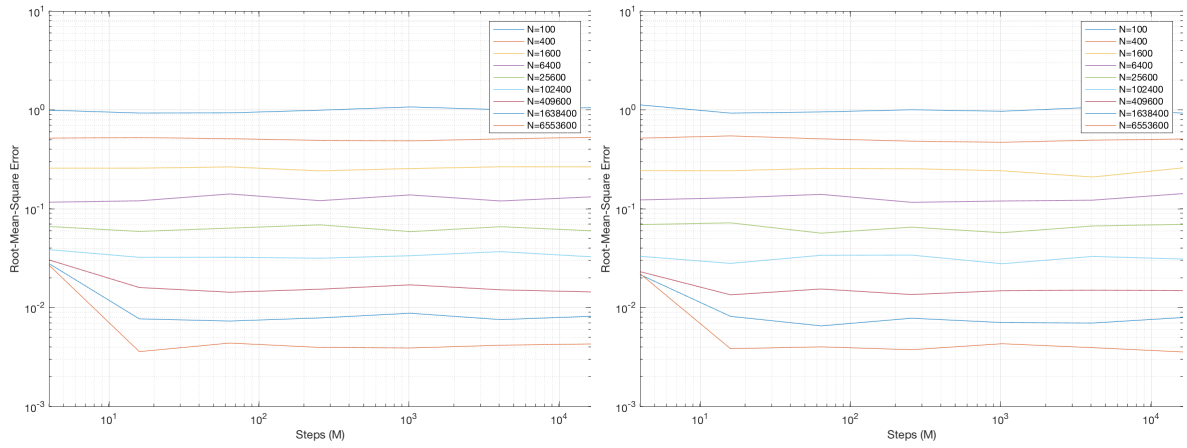
### 5.4.2 Normal Sampled Multinomial Trees

This method provides the closest likeness to the Gaussian walk with a generation method that allows for simplistic scaling of the number of branches through increasing the number of samples of the inverse cumulative normal taken. The convergence analysis considers a multinomial approximation created with 64 branches.



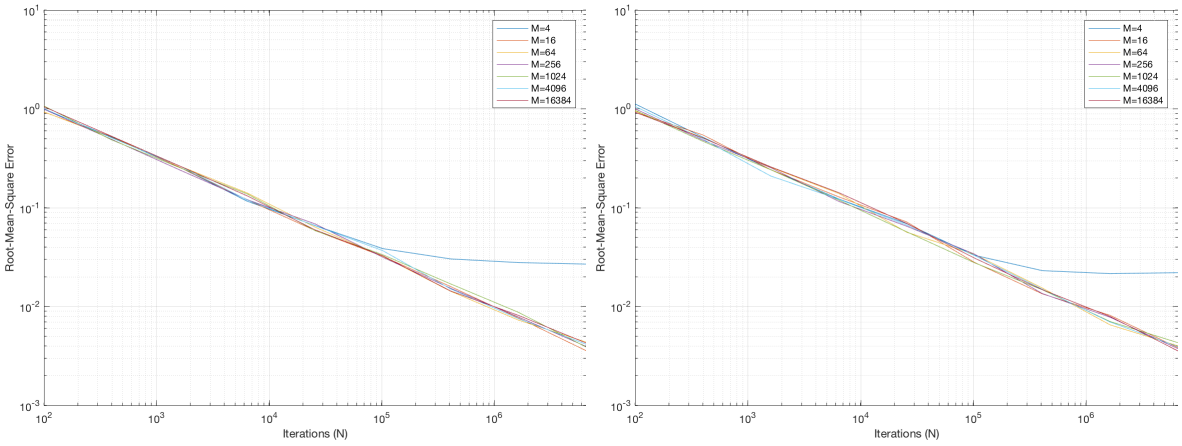

(a) Gaussian (b) Sampled normal multinomial

Figure 5.18: Root-Mean-Square Error of Gaussian and sampled normal multinomial in *NM Space*




(a) Gaussian (b) Sampled normal multinomial

Figure 5.19: Root-Mean-Square Error of Gaussian and sampled normal multinomial pproximation against steps




(a) Gaussian (b) 64-Nomial

Figure 5.20: Root-Mean-Square Error of Gaussian and 64-nomial Approximation against Iterations

The absolute error convergence of the normal sampled multinomial model is not presented as the convergence properties of the Gaussian and the 64 branch sampled normal multinomial are indistinguishable. Consequently the *NM Space* properties for selecting an appropriate number of steps to keep bias negligible for the multinomial will be the same as that of the Gaussian.

## 5.5 Summary

The Cox-Ross-Rubinstein binomial model has better convergence properties than the Jarrow-Rudd binomial model. The Jarrow-Rudd model requires 6.25× as many steps to achieve the same level of bias. This improvement does not result in a change in the intrinsic relationship between bias and steps, as the absolute error is proportional to the inverse number of steps in both. Hence the binomial model's complexity is $\mathcal{O}(\epsilon^{-3})$.

The Gaussian walk has a faster rate of true value convergence for steps than the binomial models. The absolute error is proportional to the inverse square of the number of steps. This results in an *NM Space* model that has a better complexity, $\mathcal{O}(\epsilon^{-\frac{5}{2}})$, than the binomial models'. This suggests that the binomial models will only outperform the Gaussian model for low accuracy simulations; unless the inherent complexity of walk generation can be changed in the GPU implementation.

Compressed multinomial trees have better convergence properties than their binomial counterparts. The greater the number of branches, the better the scaling. The Octanomial Cox-Ross-Rubinstein and Nonanomial Jarrow-Rudd models require 3× and 10.71× fewer steps than their binomial counterparts respectively. The complexity of the implied models is the same as the binomial models.

The normal sampled multinomial tree with 64 branches converges at the same rate as the pure Gaussian model offering strong evidence that if the throughput of the implementation is higher than the continuous model, it will prove itself a credible optimisation technique for pricing Asian options.

# 6 Variance Reduction Analysis

In this chapter, variance reduction techniques are examined to determine if the *NM Space* changes due to these methods being implemented on top of existing methods. The investigation begins by considering the difference in variance reduction for the different underlying models and for different option parameters for antithetic variates. It then explores control variate techniques by first examining the different control variates available, and then presenting the new *NM Space* model. The chapter closes with a brief discussion on the use of the multi-level technique.

## 6.1 Antithetic Variates

Section 3.6 proposes a straightforward method for reducing the variance of a predictor, the antithetic variate approach. Introducing this method for Gaussian walks is well documented, however its effectiveness for discrete-space and discrete-time Monte Carlo simulations is not known. It has to be determined if the antithetic walk for a discrete-space implementation is negatively correlated. Compressed multinomial trees are not presented in this investigation as their antithetic convergence properties will be implicitly defined by their underlying models.

Based on the option with parameters $S_0 = 100, K = 110, \sigma = 0.3, r = 0.15$, and $T = 1$, the simulations are run with 1,000 steps and 5,000,000 iterations with the same seed with regular then antithetic sampling. The changes in option parameters are considered one at a time to see the effect each individual component has on the level of variance reduction achieved by the method.

Figure 6.1 shows the reduction of variance across the four models however the binomial Cox-Ross-Rubinstein model appears to break the observed reduction in variance reduction when interest rates become large. This is due to the increase in the dominance of the upper branch probability meaning that when the inverse probability at each step is taken the probability that the upper branch is still selected is high. Equation 6.1 displays this effect by considering a large interest rate resulting in the up branch being $4\times$ more probable than the down branch.

$$\begin{aligned} p_u = 0.8 &\implies p_d = 0.2 \\ RegularRandom = p &\implies AntitheticRandom = 1 - p \\ P(AntitheticDown|RegularUp) &= 0.25 \\ P(AntitheticUp|RegularDown) &= 1.0 \end{aligned} \tag{6.1}$$

This property is not present in the Jarrow-Rudd model as the inverse probability always results in the opposite branch being taken. The reason for the larger reduction in variance for the greater values of interest rate is the effective shifting of the average value of the average walk from the strike price. This shift reduces the number of zero value payoffs generated, which results in higher levels of anti-correlation between the pairs.

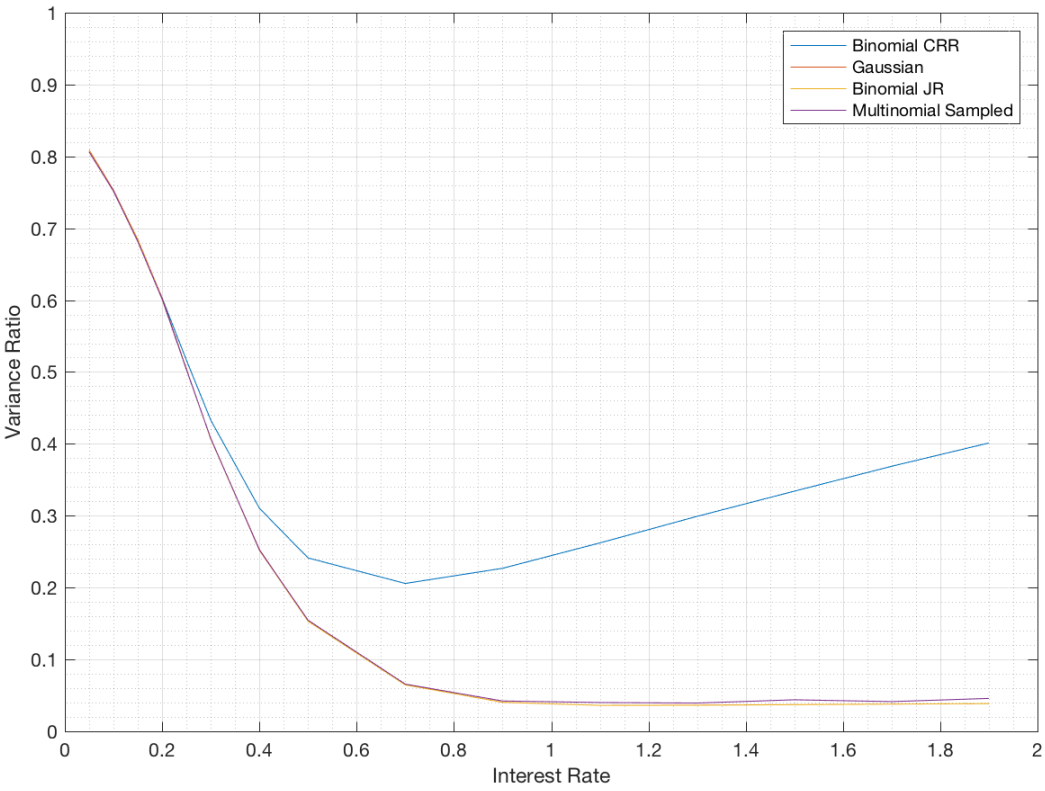



Figure 6.1: Antithetic variance ratio against interest rate

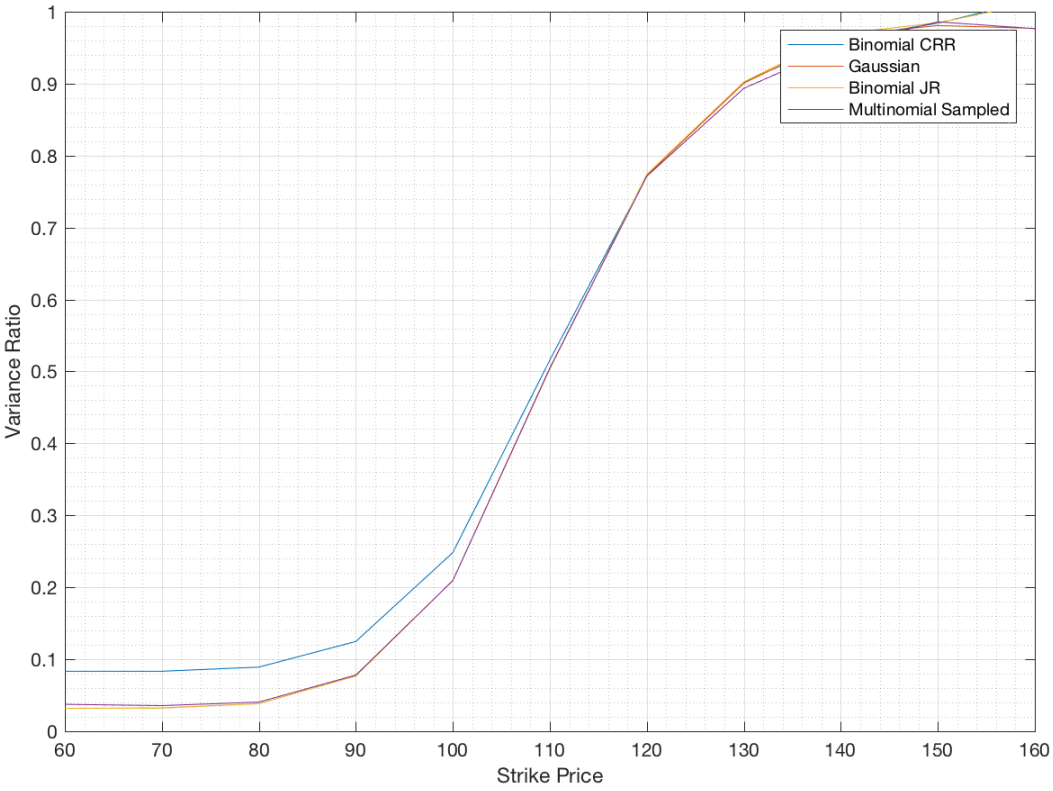



Figure 6.2: Antithetic variance ratio against strike price

Figure 6.2 supports the conclusions reached from considering changes in interest rates, that a movement between the expected value of the average walk and the strike price will result in a change in the level of variance reduction. This makes clear the relationship as there is little difference between the underlying models as the the strike price does not affect the path generation. There is a mildly less variance reduction in the Cox-Ross-Rubinstein model, however this was explored in the previous diagram. This variance reduction relationship in call prices implies that for put prices the relationship will occur in the opposite direction of the strike price due to the opposite sign of the payoff calculation.

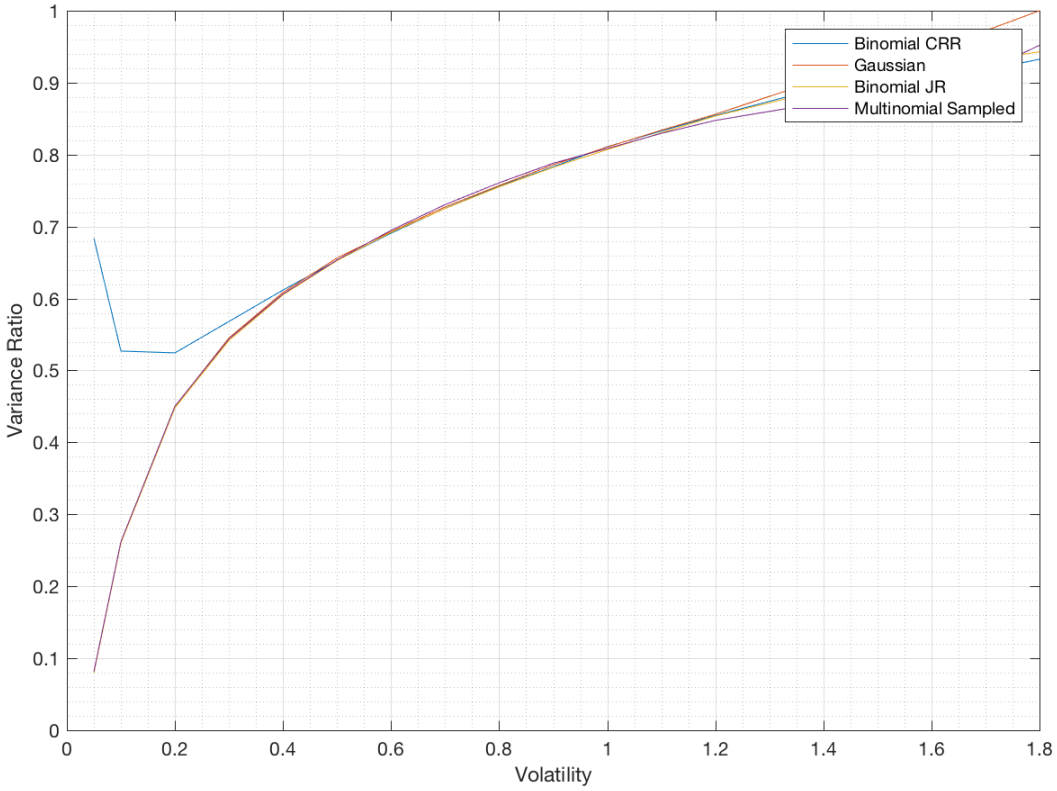



Figure 6.3: Antithetic variance ratio against volatility

Increasing the volatility of a simulation increases the value of call price due to the arithmetic bias of large outliers at the upper end of the average walk distribution. This explains why in Figure 6.3 there is less variance reduction for higher volatility options, as it results in an effective increase in the average walk value. The binomial model Cox-Ross-Rubinstein breaks the pattern for small volatilities as the value of the upper and lower branch of the model become increasingly similar resulting in a minimal impact of taking the alternative path.

The level of the variance reduction differs across the input parameters similarly for all models, with the exception of the Cox, Ross and Rubinstein model which diverges for low interest rates and high volatilities. For the option considered as the basis of the *NM Space* comparative analysis, the variance ratio is 0.7. Based on this fact a new *NM Space* will not be recalculated as the resulting reduction does not change the convergence of the bias. The variance reduction will impact the bias to confidence interval size ratio however this is deemed a manageable difference.

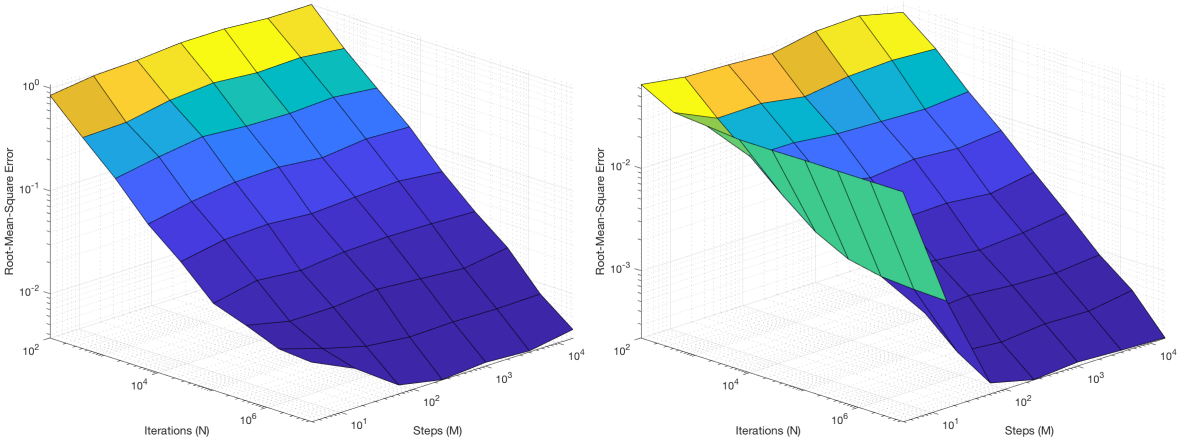
## 6.2 Control Variates

This section lays out the changes required in *NM Space* when using control variates. The first stage of the investigation is considering the difference between European and Geometric options as control

variates for pricing Arithmetic Asian options. The section continues by examining the differences between the variance reduction of continuous and discrete space models using Geometric control variates.

### 6.2.1 Gaussian Geometric and European

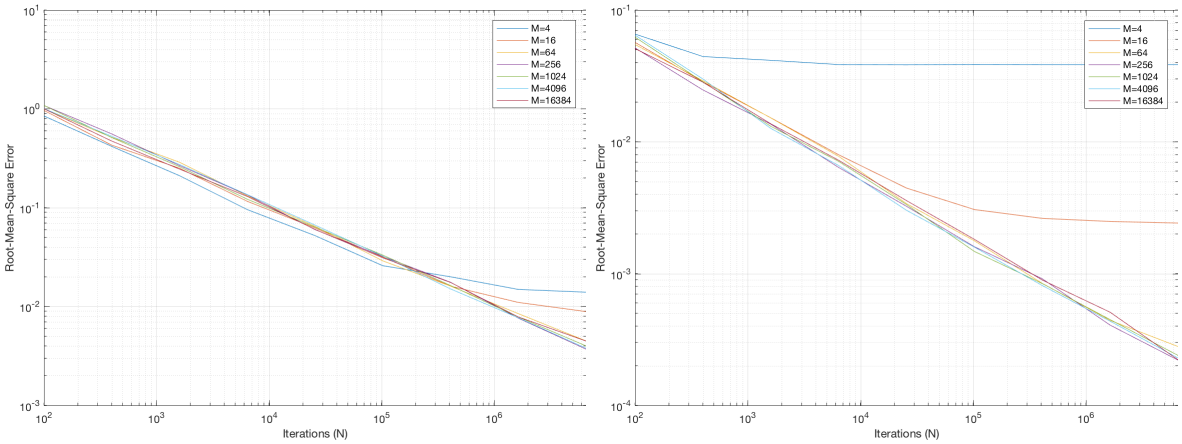
The two primary control variate candidates are European and Geometric Asian options; both have an analytical solution and both have prices correlated to the Arithmetic Asian option price. Introducing either control variate will result in a increase computational workload however the *NM Space* model must be considered to understand whether the trade off is significant.




(a) European (b) Geometric

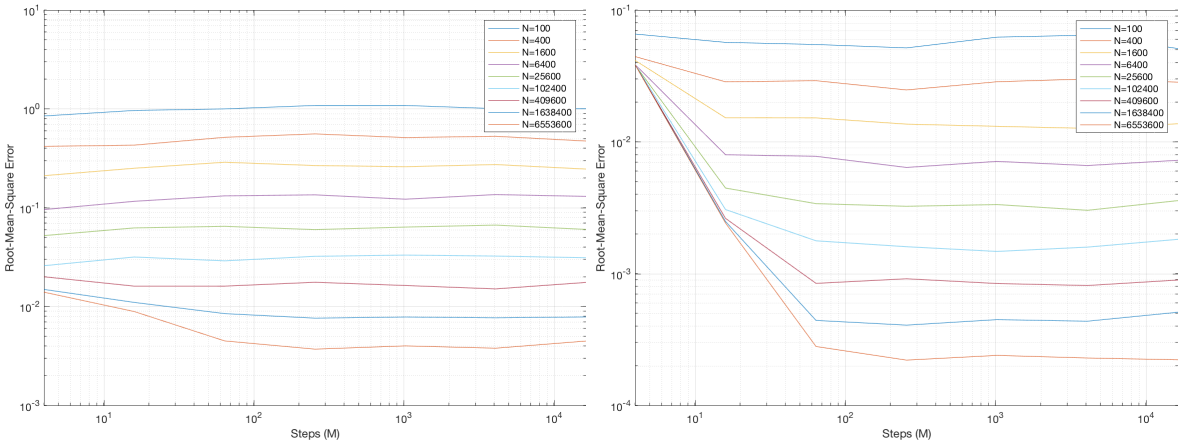
Figure 6.4: Root-Mean-Square Error of Gaussian control variates *NM Space*

Figure 6.4 displays the RMSE of the two control variate techniques. The first observation is that the European control variate does not result in a strikingly different result to the regular Gaussian model. There is a distinct difference in the geometric model that introduces a dramatic relationship between increasing the number of steps taken. The source of this increased dependence on steps is a consequence of the Geometric Asian Monte Carlo simulation having an *NM Space* that is more dependent on the number of steps executed.




(a) European (b) Geometric

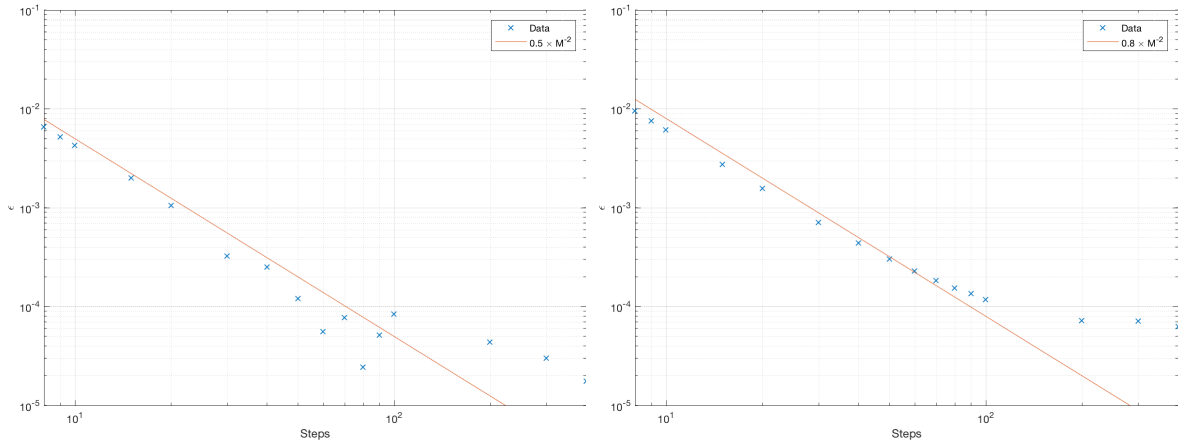
Figure 6.5: Root-Mean-Square-Error Gaussian control variates against iterations




(a) European (b) Geometric

Figure 6.6: Root-Mean-Square-Error Gaussian control variates against steps

Figures 6.5 and 6.6 clarify the nature of the convergence properties. It is noted that the point at which the step level breaks off from the iteration trend is immediate for the Geometric control variate suggesting that the *NM Space* of pricing a Geometric Asian option has a higher initial number of steps required than the European control variate method and the plain Monte Carlo methods.




(a) European (b) Geometric

Figure 6.7: Convergence of Gaussian control variates to true value

For Geometric control variates the iterations convergence equation is:

$$\epsilon = 0.5N^{-\frac{1}{2}} \tag{6.2}$$

suggesting a reduction in variance from the non-control variate method of 20. This allows the *NM Space* to be calculated using the same methods presented in Chapter 5 as shown in Table 6.1

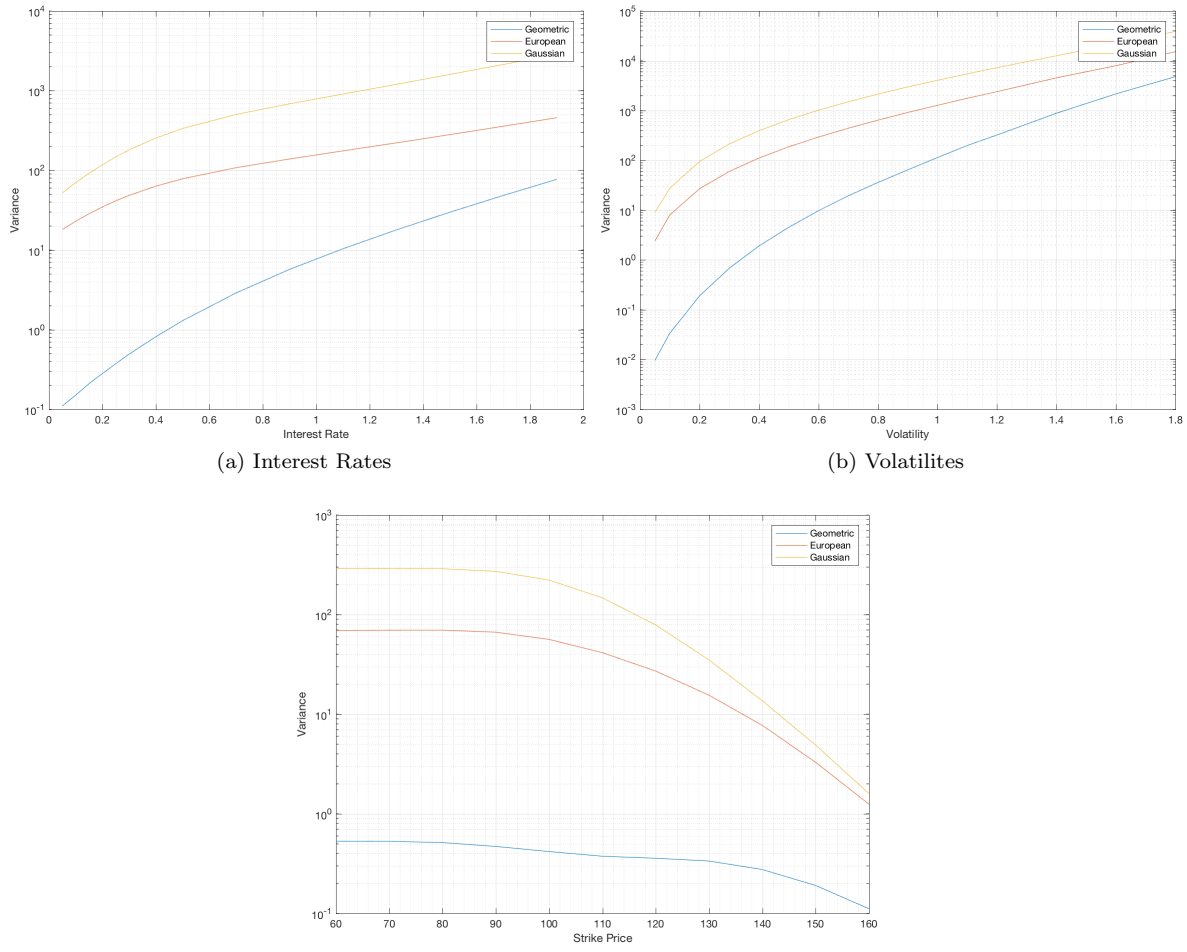
| European Control Variate | Geometric Control Variate |
|---|---|
| $\epsilon = 0.5 \times M^{-2}$ | $\epsilon = 0.8 \times M^{-2}$ |
| $M \approx 2.23 \times 10^{-1} N^{\frac{1}{4}}$ | $M \approx 1.2649 N^{\frac{1}{4}}$ |
| $\mathcal{O}(\epsilon^{-\frac{5}{2}})$ | $\mathcal{O}(\epsilon^{-\frac{5}{2}})$ |

Table 6.1: Gaussian control variate *NM Space* properties

The differences between the Geometric and European control variates are analysed through measuring the changes in the variance and coefficient values. The methodology for examining their properties is the same as that presented for Antithetic analysis in the previous section. The variance and coefficient derivations are calculated using:

$$c = \frac{Cov(x, y)}{Var(y)} \tag{6.3}$$

$$Var(x_c) = Var(x) - \frac{Cov(x, y)^2}{Var(y)} \tag{6.4}$$




(a) Interest Rates

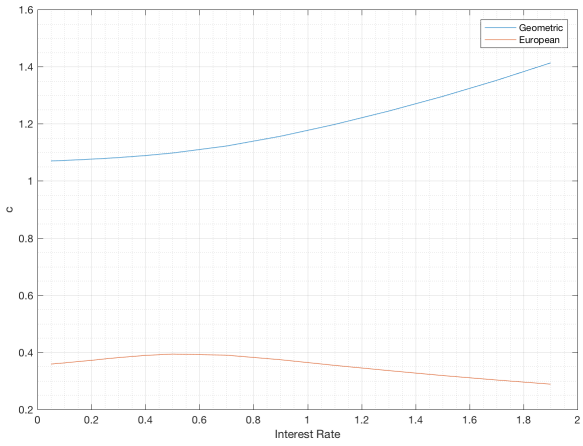
(b) Volatilites

(c) Strike Prices

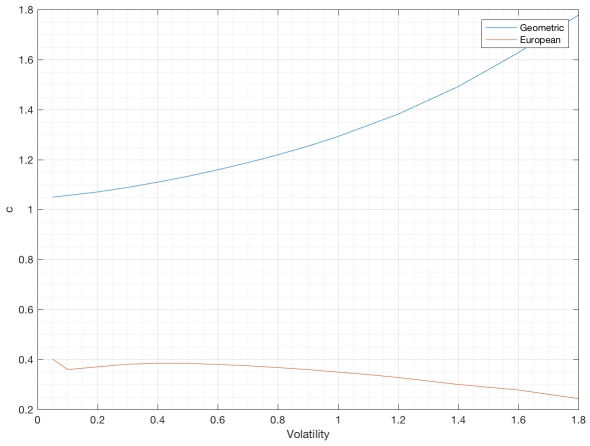
Figure 6.7: Gaussian and European control variate variances

Figure 6.7 displays the variances of the call price of a standard Gaussian walk, a European control variate method, and a Geometric control variate method. It is apparent that both control variate

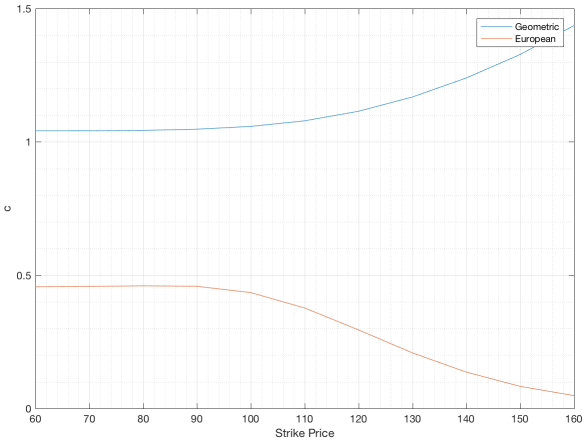
methods result in a reduction in variance however the reduction is far greater in the Geometric Gaussian than in the European case for all values considered.




(a) Interest Rates




(b) Volatilities




(c) Strike Prices

Figure 6.7: Gaussian and European control variate coefficients

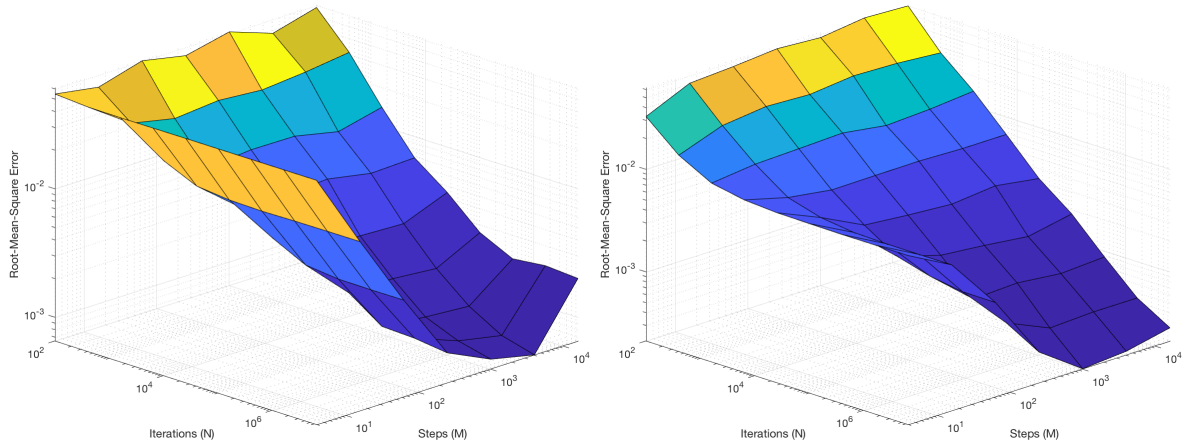
Figure 6.7 provides an empirical case as to why their is more variance reduction in the Geometric case when juxtaposed with the European. It is seen that in all cases the covariance is higher in the geometric case, hence the reduction it makes is greater.

### 6.2.2 Geometric Binomial

It known that for a Gaussian walk the uses Geometric control variates results in $20x$ smaller variance. It is not known whether this holds for the discrete models proposed. Determining whether the same holds is implicitly dependent on the ability of the discrete-space models to price Gaussian Geometric options.

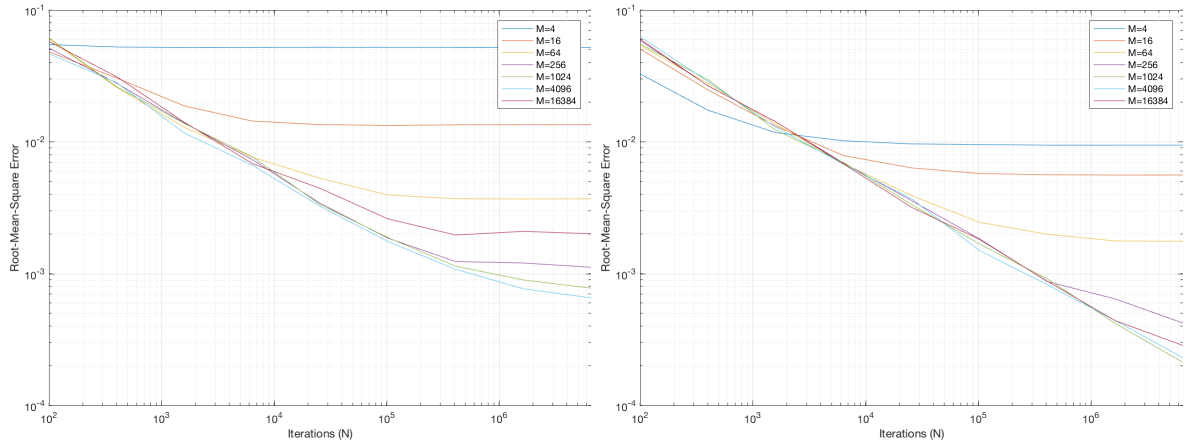
Figure 6.8 reveals that Geometric control variates are still applicable for discrete space models, but the *NM Space* convergence properties of the models differ mildly from each other as well as from the Gaussian control variate technique considered in the previous section.

Figures 6.9 and 6.10 evidence the differences between the two fundamental binomial tree based models. The Cox-Ross-Rubinstein model has a higher initial requirement of steps, but once both models have passed the initial steps threshold both follow the standard expected inverse square-root model anticipated by central limit theorem. It is seen that the convergence of the Cox-Ross-Rubinstein



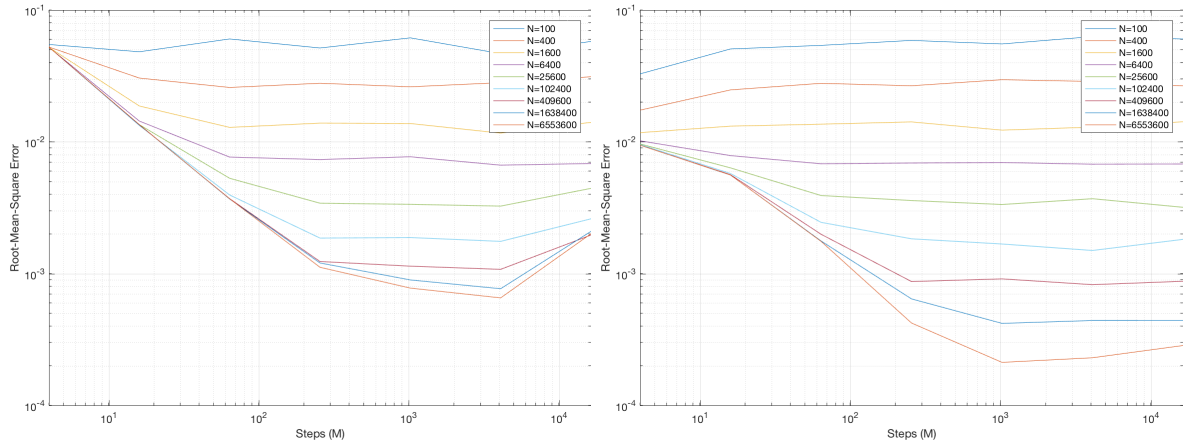

(a) Cox-Ross-Rubinstein (b) Jarrow-Rudd

Figure 6.8: Root-Mean-Square Error of binomial control variates *NM Space*




(a) Cox-Ross-Rubinstein (b) Jarrow-Rudd

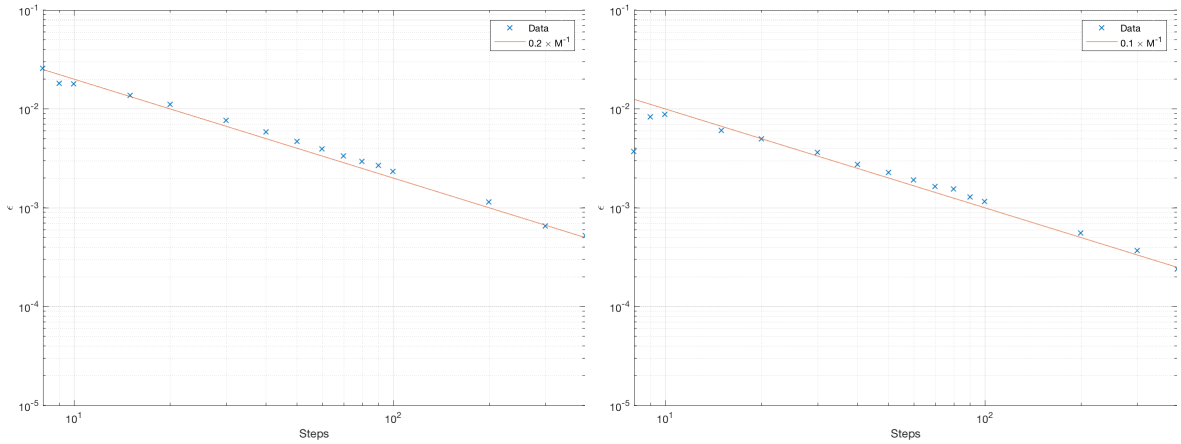
Figure 6.9: Root-Mean-Square Error of binomial control variates against iterations




(a) Cox-Ross-Rubinstein (b) Jarrow-Rudd

Figure 6.10: Root-Mean-Square Error of binomial control variates against steps

model stalls for a large number of steps, the slight increase at the end of the steps range points that this is likely down to floating point error of the calculation.




(a) Cox-Ross-Rubinstein (b) Jarrow-Rudd

Figure 6.11: Convergence of binomial control variates to true value

| CRR Control Variate | JR Control Variate |
|---|---|
| $Bias\epsilon = 0.2 \times M^{-2}$ | $\epsilon = 0.1 \times M^{-2}$ |
| $M \approx 6.324 \times 10^{-1} N^{\frac{1}{2}}$ | $M \approx 4.472 \times 10^{-1} N^{\frac{1}{2}}$ |
| $\mathcal{O}(\epsilon^{-3})$ | $\mathcal{O}(\epsilon^{-3})$ |

Table 6.2: Binomial control variate *NMSpace* properties

The overall complexity of the model is the same to that of the non control variate method however the *NM Space* hides the scaling of epsilon based on the variance reduction achieved by the control variate method.

### 6.2.3 Geometric Multinomial

In the interest of avoiding repetition it is stated that the Geometric multinomial tree with 64 branches results in a model indistinguishable from the true Gaussian approximation got control variate techniques too. The location of the results used to determine this are found in the repository referenced in Chapter 13.

### 6.2.4 Multi-level

The basis of the Multi-level Monte Carlo simulation is enticing with the promise of active variance reduction using a low-cost baseline estimates that could then have its inherent error eliminated through the use of correlated higher-cost simulations, changing the very complexity of the simulation.

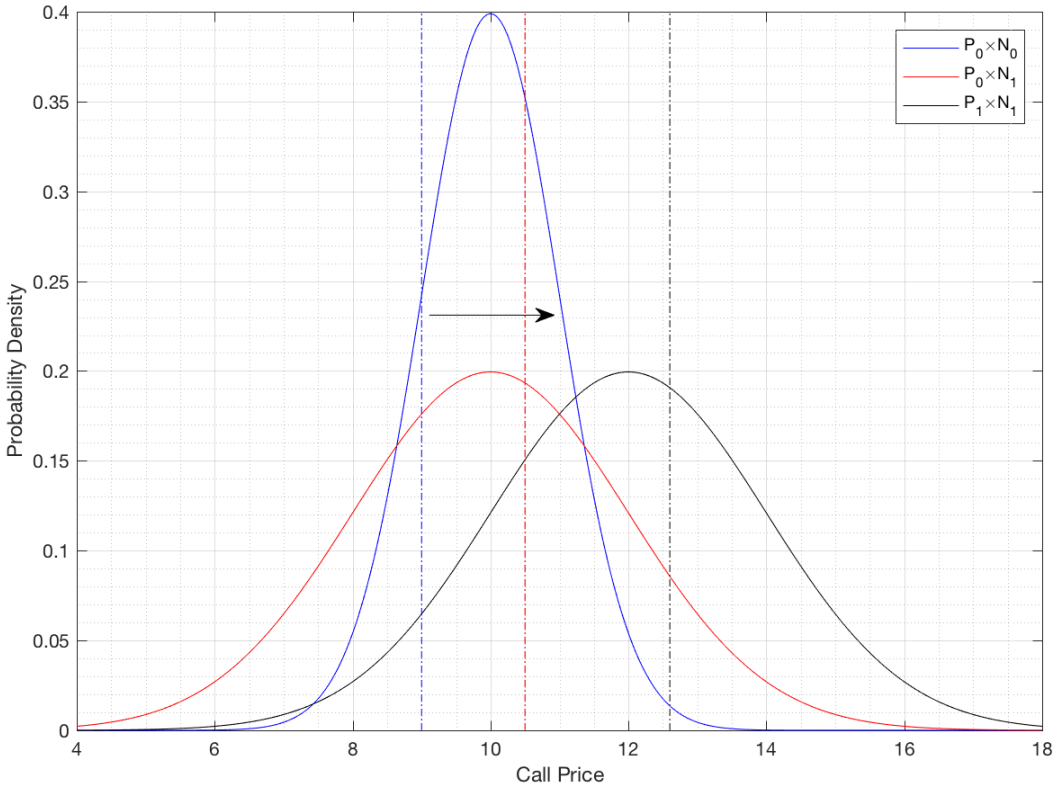



Figure 6.12: Expected behaviour of multi-level approach

Figure 6.12 displays the anticipated behaviour of the multi-level approach. The granular simulation gives a tight confidence interval, however it is a biased result due to the inaccuracy of the underlying estimation. The granular and fine simulation are generated with the same underlying stochastic process, the difference between the two provides the bias difference between the two simulations.

The implementation considered in this project made use of Gaussian bridges as a means of increasing the number of intermediary steps. This allowed for generation of a more accurate path representation with a shared stochastic basis to a lower level simulation. The initial findings were unsatisfactory with the implementation showing no distinct improvements over the standard Gaussian method.

The primary issue encountered with the model was that the convergence basis provided by Giles [11] results in convergence after a single adjustment. The adjustment made by this two-step multilevel approach was several orders of magnitude smaller than the lowest level estimation's confidence interval irrespective of the magnitude of the bias, eg. $P_0 = 5.624, P_1 - P_0 = 0.00033, T_r \approx 5.73007$. This result does not match those presented by Giles, it is assumed that the basis of the error is in the interpretation of the point at which the variance of levels is taken.

Rather than investing further resources into troubleshooting this method, the decision was taken to remove the multi-level methods from the scope of the project. This decision was made in part down to the poor initial findings but primarily due to the lack of documented approaches to apply the method to the discrete-space models.

## 6.3 Summary

The antithetic variate method showed reduction in variance for all methods although the Cox-Ross-Rubinstein model broke the trends due to the manner in which its $u, d$, and $p$ values are calculated. The decision was made that the *NM Space* would not be altered due to lack of impact on the

convergence of the bias.

It was found that Geometric Asian options are better than European options as control variates at reducing the level of variance. The reductions in variance caused by the Geometric Asian control variate method were applicable for the discrete models. Contrastingly the Jarrow-Rudd model outperformed the Cox-Ross and Rubinstein model, requiring fewer steps. For all the control variate methods due to the nature of the variance reduction, the *NM Space* was recalculated for all models applied to.

Due to the inability of the multi-level method to be applied to the discrete models it was decided to be expunged from the project.

# 7 Framework Design Choices

To test a variety of different Monte Carlo based option pricing simulations on the GPU, a framework was devised to enable reliable comparisons. The framework was designed such that it minimised the amount of initial set-up required to generate a new method.

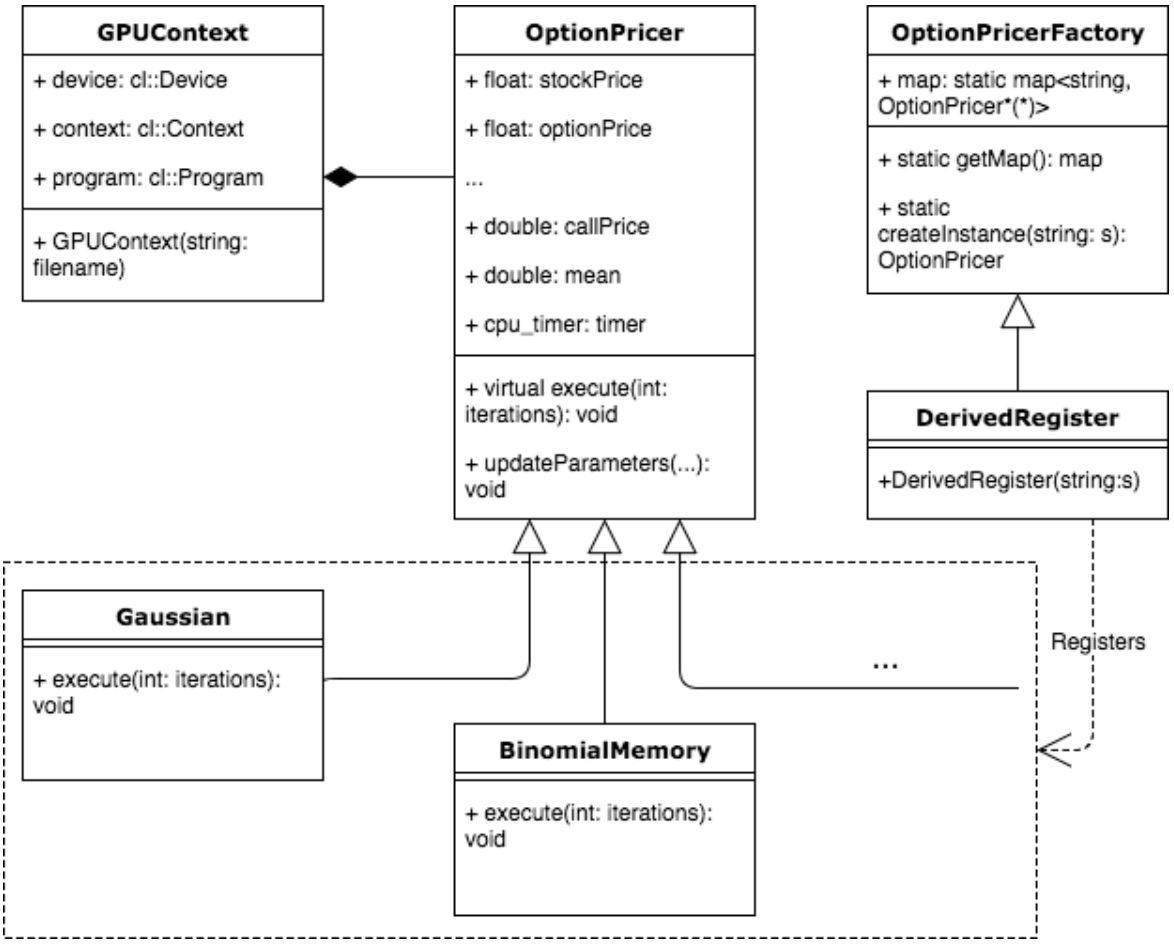



Figure 7.1: Overall Code Structure

The framework was built around an abstract class called *OptionPricer*. This had children class for each type of Monte Carlo simulation and consisted of a *GPUContext* that handled the OpenCL setup for the GPU. To access all of these different types of option pricing simulators, a static map was created which contained the constructor for each option pricer to enable creation using the an identifier string. Figure 7.1 displays the principal methods, variables, and interactions between classes.

The *OptionPricer* contains all of the common elements of each Monte Carlo option pricing variant. This includes things such as the underlying stock variables, execution options, and statistics. Another

important component of the parent class is that it contains utility methods that are common across all of the classes such as output formatting and calculating the average value of a buffer. *Boost* timers are included to ensure that there is a consistent method for timing each Monte Carlo simulation, which does not take into account the overhead of the framework when generating execution times.

The *GPUContext* handles initialising the main OpenCL objects that are common across the different variants of the engines. It selects the platform and device based on environment variables from the terminal; this is done as the development integrated graphics card does not support double precision execution hence the using environment variables allowed to quickly change without recompilation of the code to use the CPU to test the functionality of the double precision kernels. Loading the kernel is part of the initialisation process, rather than simply loading in the kernel for the specific Monte Carlo walk it also loads in an OpenCL file that contains common functions such as random number generation, payoff calculation, etc.

The *OptionPricerFactory* is a singular static map that uses the child class *DerivedRegister* to store the constructors for each of the variants of Option Pricers. This enables the use of the *OptionPricer* name to initialise the class which helps with the automation of testing as it increases the ease of running the same tests for different devices.

The executables generated based on the framework are explained in Chapter 12.

# 8 GPU Optimisations

This chapter presents how to implement continuous and discrete-space Monte Carlo simulations for GPUs. This goals is achieved by exploring particular design choices that can be made for each variant of the Monte Carlo simulation and considering what potential trade-offs are made.

## 8.1 Hardware Specifications

In this chapter and subsequent chapters there will be various execution times quoted, unless explicitly stated it should be assumed that the tests were run using the NVIDIA Tesla K80 on an *Amazon Web Services* p2.xlarge instance. Below is a table of the primary specifications of the cards used in this project.

| **GPU Name** | **NVIDIA Tesla K80** | **NVIDIA K520** | **Intel Iris 5100** |
|---|---|---|---|
| Memory | 24GB | 8GB | 2GB |
| Base Clock Frequency | 562MHz | 797MHz | 200MHz |
| Peak Frequency | 876MHz | 1250MHz | 1200MHz |
| Maximum Input Power Consumption | 300W | 225W | 30W |
| Release Date | 2015 | 2014 | 2013 |

Table 8.1: Graphics Processing Units

## 8.2 Uniform Random Number Generation

An aforementioned critical component of the Monte Carlo simulation in section 3.9 is the random number generation. There is a trade off between the *randomness* of a number and the time taken to generate it. The two primary types of random number generators considered are XORShifts proposed by George Marsagalia [29] and a Multiply-with-Carry based algorithm presented by David Thomas [38]. The transformation of the methods into OpenCL implementations are considered below:

```
uint MWC64X(uint2 *state)
{
    uint x=(*state).x;
    uint c=(*state).y;
    uint res=x^c;
    uint hi=mul_hi(x,4294883355U);
    x=x*4294883355U+c;
```

```
    c=hi+(x<c);
    *state=(uint2)(x,c)
    return res;
}


//Bit Sampled MWC64X
for (int i = 0; i < 32; i++)
{
   random = (random >> 1) & 1;
}
random = MWC64X(state);}


uint xorshift32(uint *state) {
    uint x = *state;
    x ^= x << 13;
    x ^= x >> 17;
    x ^= x << 5;
    *state = x;
    return x;
}


uint xorshift128(uint4 *state)
{
    uint t = (*state).w;
    t ^= t << 11;
    t ^= t >> 8;
    (*state).w = (*state).z;
    (*state).z = (*state).y;
    (*state).y = (*state).x;
    t ^= (*state).x;
    t ^= (*state).x >> 19;
    (*state).x = t;
    return t;
}
```

To test the speed of the random number generator a kernel comparison was devised. Each kernel generates a sequence of random numbers and stores the last number reached in a buffer, to ensure the sequence code is not optimised away by the OpenCL compiler. To minimise the impact of the buffer read and write times each kernel generates 134,217,728 sequential random numbers.

| **Random Numbers Generated** | **XORShift32** | **XORShift128** | **MWC64X** | **MWC64X Bit Sampled** |
|---|---|---|---|---|
| 1,342,177,280,000 | 8.884618 | 7.743461 | 10.590659 | 0.434366 |
| 13,421,772,800,000 | 66.854407 | 60.504456 | 83.987211 | 2.724632 |

Table 8.2: Random number execution times

The execution times of the XORShift32, XORShift128, and MWC64X are similar. Given the smaller state size of the XORShift32 and its slower execution time relative to the XORShift128 it is discarded as a credible random number generator. The XORShift128 is approximately 1.37x faster than the MWC64X generator, however the numbers generated are *less random*. Given the minimal difference between the speed of generation the decision is made to use the MWC64X over the XORShift128 as the standard uniform random number generation. When a binary random number is required it is noted that bit sampling the MWC64X has a 31x speed up over the equivalent uniform MWC64X generator, this is a benefit of the high quality of randomness of the number, as typically *fast* random number generators often have less random low order bits.

## 8.3 Option Price Accumulation

A bottleneck that is present in all the variants of the option pricer is getting the values of the call prices from the GPU device to the CPU host and taking its mean. This is an operation that is intrinsically opposed to the GPU architecture as accumulating to a single location would require the use of atomic properties that would result in performance far slower than a sequential accumulation. An alternative approach would be to copy the data to the CPU and then process it there, however this too is limited by the bandwidth of transferring the GPU memory to the CPU. To circumnavigate these two problems a partial accumulator was introduced to compress the size of data transferred as well as reducing the number of sequential operations that the CPU needs to execute to calculate the result.

```
__kernel void averageKernel(__global float *input, __global float *output,
                            uint const divisionSize, uint const inputSize) {
  uint n = get_global_id(0);
  float outputInternal = 0;
  if ((n + 1) * divisionSize ≤ inputSize) {
    for (uint i = 0; i < divisionSize; i++) {
      outputInternal += input[n * divisionSize + i];
    }
  } else {
    for (uint i = 0; i < (n + 1) * divisionSize − inputSize; i++) {
      outputInternal += input[n * divisionSize + i];
    }
  }
  output[n] = outputInternal;
}
```

The code shows the basis of condensing the data into partial sums. An important consideration was taking into account the case in which the bin size does not fit perfectly into the total number of iterations, hence modulo arithmetic was used to avoid reading from unassigned memory.

## 8.4 Memory Based Model

Fabry's model used a lattice based model on an FPGA in which the values of the lattice were pre-calculated. This design choice makes sense in the context of the FPGA which uses Lookup Tables that have very rapid lookup speed. This raises the question as to whether or not this design choice can be replicated when considering implementing it in GPU memory.

The algorithm is similar to the multiplicative model presented in the Monte Carlo subsection of the background research. The difference between the two is pre-calculating the values the lattice, and stepping between memory locations to discover the current value of the walk as opposed to multiplicatively calculating it based on the value at the previous point.

The pre-calculated values are generated on the CPU and then transferred to the GPU using a buffer. Storing the pre-calculated values in global memory is not an effective approach for implementing this model as global memory is inefficient and slow. Accessing local memory is faster than global memory however it comes with the cost of having to copy the global memory to the local memory. This is only worthwhile if the time saved by using local memory in the walk is greater than the amount of time required to copy the memory across. This presents the need to consider doing multiple walks per kernel as opposed to a single walk per kernel. This should result in a speed up for the global memory variant as well as it will reduce the number of times the call option price needs to be written out to memory.

Figure 8.1 brings to light the difference in execution speed for global and local memory when executing the same simulation with 2,000 steps and 10,000,000 iterations. Contrary to the initial belief that

**Algorithm 4** Binomial Pricing Algorithm (Lattice)

1: **procedure** BINOMIALMONTECARLO($S_0, K, u, d, p, T, Steps, Iterations$)
2: $payOffSum = 0$
3: $LatticeSize = 2 * steps + 1$
4: $Lattice[Steps] = S_0$
5: **for** `i=0 to Steps-1` **do**
6: $Lattice[i] = S_0 * u^i$
7: $Lattice[i + Steps] = S_0 * d^i$
8: **end for**
9: **for** `i=1 to Iterations` **do**
10: $sum = S_0$
11: $SIndex = steps$
12: **for** `j=1 to Steps` **do**
13: $W \leftarrow U(0, 1)$
14: **if** `W` $\leq$ `p` **then**
15: $SIndex \leftarrow SIndex + 1$
16: **else**
17: $SIndex \leftarrow SIndex - 1$
18: **end if**
19: $sum \leftarrow sum + Lattice[SIndex]$
20: **end for**
21: $L \leftarrow sum \div (Steps + 1) - K$
22: $payoffSum+ = max(L, 0)$
23: **end for**
24: $P \leftarrow e^{-rT} \times payoffSum \div Iterations$
25: **end procedure**

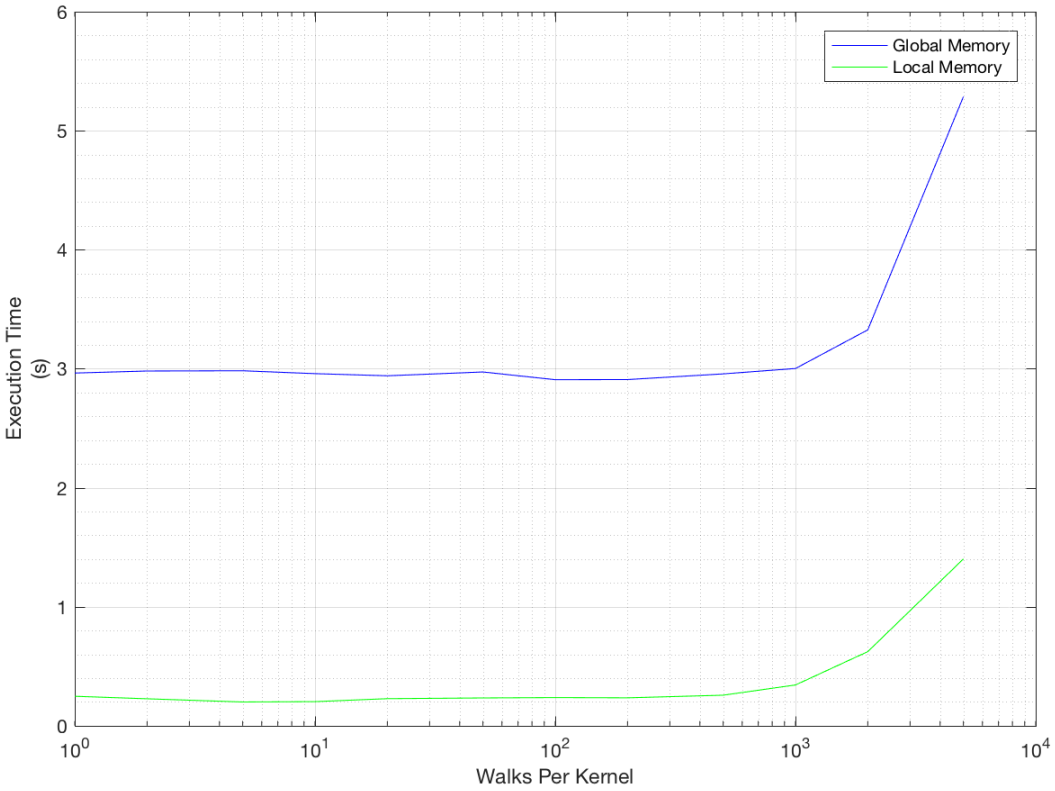



Figure 8.1: Execution time against walks per kernel

each thread would need to copy global memory into local memory at the beginning of execution, the copy is completed for all work groups in parallel. This suggests that the need for multiple kernels per thread is unnecessary to make the copy effective. This experiment does however raise an interesting point that when the walks per kernel reaches 200 there is a noticeable slowdown; this is a consequence of the underutilisation of the execution cores available.

The walks per kernel could be increased for the benefit of reducing the amount of memory required to store the walk information. This decision was decided against as the partial accumulation removed the majority of the slowdown associated with the copying of the memory, and reducing the information stored would limit the ability to calculate additional statistical tests on the call prices such as calculating moments.

It is noted that the amount of local memory available is far smaller than the amount of global memory available. This limits the number of steps than can be taken based on the size of the available local memory. This value differs between hardware however issues begun appearing when the number of steps was of the order of $10^4$. Hence any solution based on this memory based solution will as such have an upper limit based on the number of steps it can take, based on the *NM Space* model this is unlikely to be an issue for confidence intervals considered in this project for this model.

## 8.5 Multiplication Based Models

The alternative approach to the memory based model is the use of an active multiplication approach. Instead of pre-calculating the stock values, each stock value is calculated based on the previous stock value. The basis for this decision is that reading to memory and computing a multiplication on a GPU is comparable in terms of execution time.

A design decision for the multiplicative model is choosing whether to consider single precision or double precision numbers. The execution speed of double precision numbers on GPUs is considerably slower than floating precisions, the exact factor is dependent on specific optimisations made by the hardware. The basis of this faster floating point execution is a consequence of the original use case of the GPU, which is optimised for graphics processes which primarily make use of floating point values.

Figure 8.2 shows that the error associated with floating point and double is correlated to the number of steps taken, this is due to the convergence of $dt \to 0$, which leads to small quantisation errors in the values of $u$,$d$ and $p$ which in turn results in errors that propagate through the walk. The convergence of these values can be seen in Figure 8.3. This is an issue that is less noticeable in the lattice based model as all of the calculations used to fill the lattice were done with double precision and the maximum step size is limited due to the constraints on local memory. The larger the value of the volatility and the interest rate the fewer steps required before this error becomes significant due to its involvement in the calculation of the values of $u$ and $d$. The decision is made that the magnitude of the error is small enough to ignore as the Cox-Ross-Rubinstein *NM Space* considered is below $10^3$ steps in this project which results in error less than $10^{-4}$.

The use of a multiplicative model allows for the use of a different binomial model, the Jarrow-Rudd model. The Jarrow-Rudd model is examined to determine the differences between the use of of double precision and single precision to determine whether it is also a viable choice to use floating point numbers.

From Figure 8.4 is is seen that the error associated with single precision and double precision follows the same general pattern seen in the Cox-Ross-Rubinstein model however it is scaled to greater level of magnitude. The decision is made that it is still advantageous to use single precision calculations as the Jarrow-Rudd *NM Space* considered is below $\times 10^4$ steps in this project which results in error less than $10^{-3}$. For greater levels of accuracy the Jarrow-Rudd model would likely need to use double precision.

Conditional statements are a layer of complexity that can slow down stepping a binomial tree model.

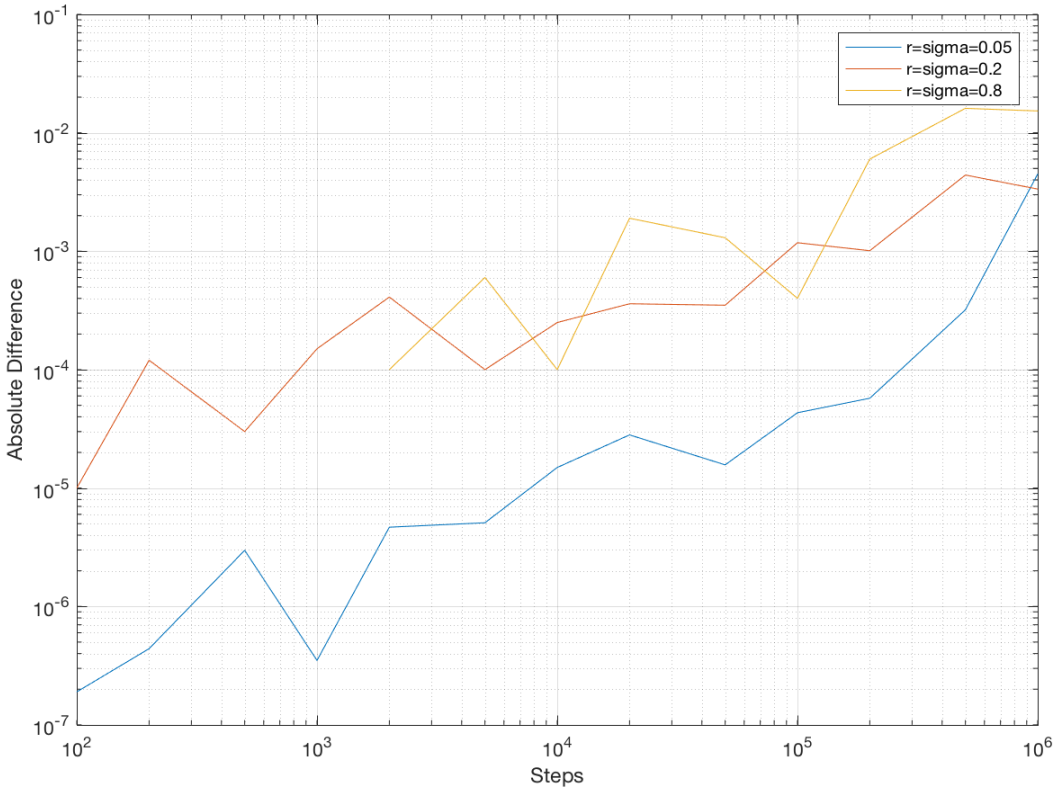



Figure 8.2: Absolute difference between single precision and double precision for Cox-Ross-Rubinstein

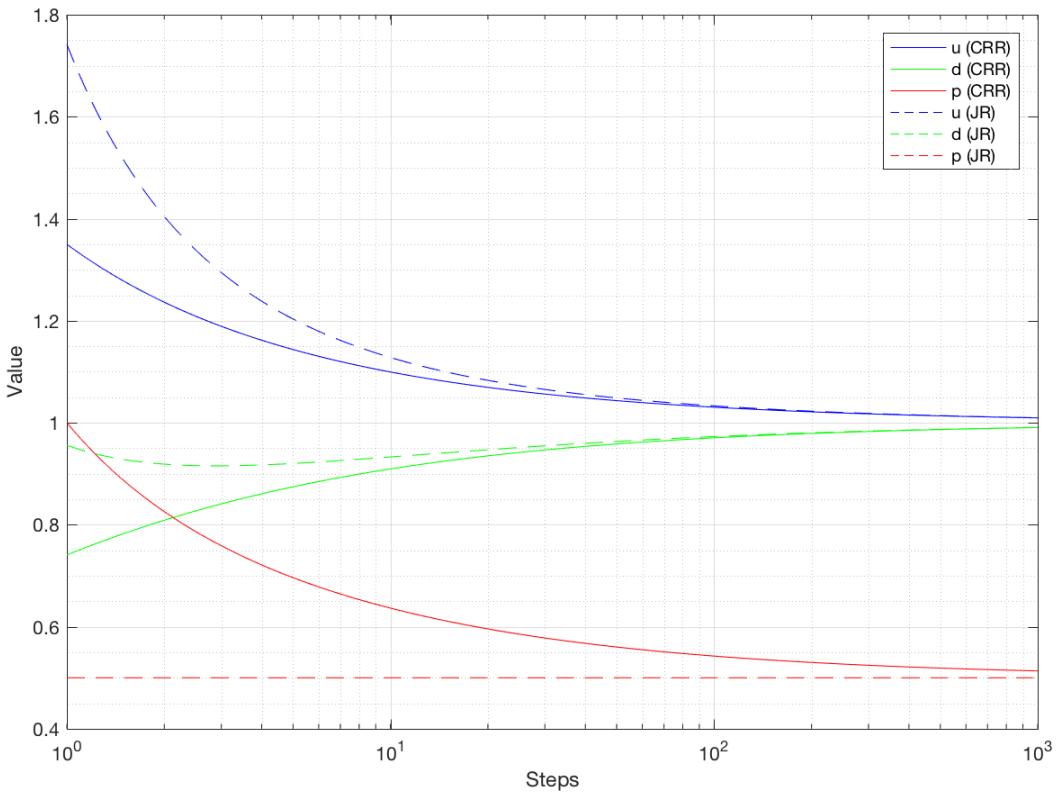



Figure 8.3: Convergence of u,d, and p for binomial models

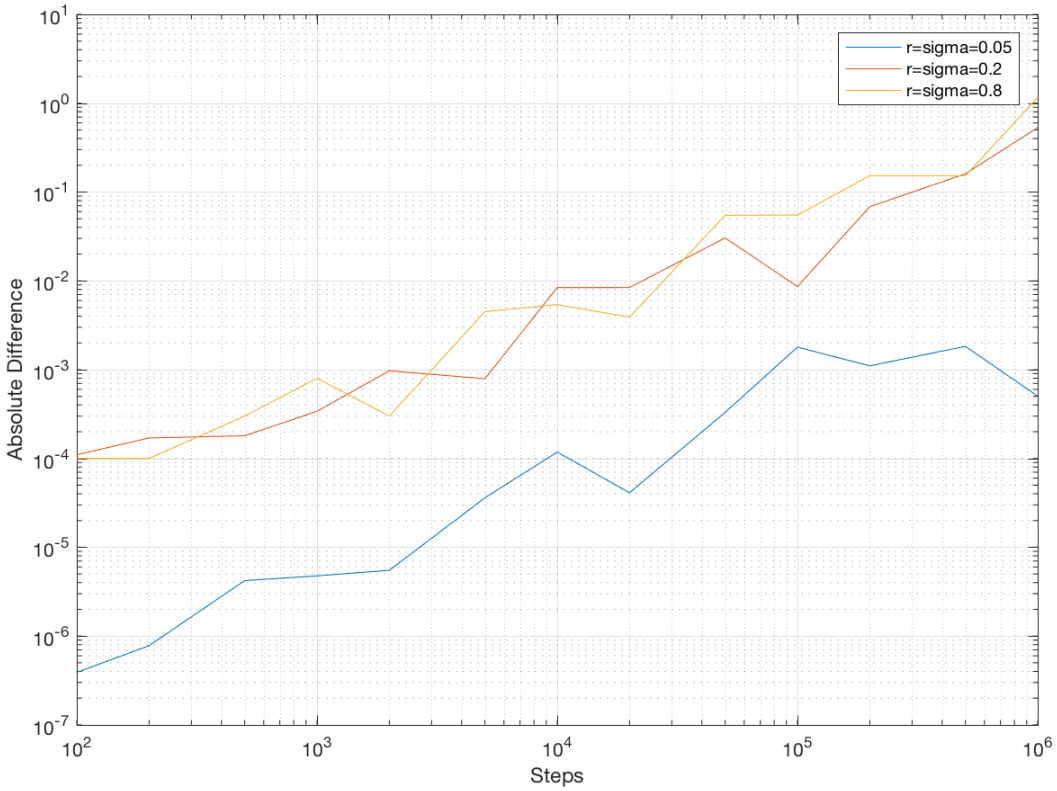



Figure 8.4: Absolute difference between single precision and double precision for Jarrow-Rudd

The Jarrow-Rudd model has the advantageous property of both branches being equally probable which enables a single boolean value to be used to determine whether to go up or down. This can be take advantage of either making use of an array indexed by the boolean or considering a multiplication in which one branch is multiplied the bit and the other is multiplied by the NOT of the bit. Due to the ability of the method to be scaled to multinomial trees and faster execution speed the array based approach is followed.

```
//Conditional Approach
if(random&0b1)
{
    currentValue*=u;
}
else
{
    currentValue*=d;
}


//Array Approach
float multipliers[2] = {u,d};
currentValue*=multipliers[random&0b1];
```

## 8.6 Normal Random Number Generator

Normal Random samples are required to complete Gaussian walks. In section 3.9 it is stated that the normally distributed random number generator being used is Box-Muller method. Below is the implementation, making use of the MWC64X random number generator.

```
float2 generateNormals(uint2 *seed, uint2 *seed2) {
  uint random1 = MWC64X(seed);
  if (random1 == 0) {
    random1 = MWC64X(seed);
  }
  uint random2 = MWC64X(seed2);

  float u1 = (float)random1 / (float)0xffffffff;
  float u2 = (float)random2 / (float)0xffffffff;

  float a = native_sqrt(-2 * native_log(u1));
  float b = 2 * M_PI * u2;

  float normal1 = a * native_sin(b);
  float normal2 = a * native_cos(b);

  return (float2)(normal1, normal2);
}
```

The decision was made to use the native functions as opposed to the standard functions due to the significant speed ups for all the GPUs tested on, as shown in Table 8.3. The difference in the results from introducing the native functions will differ depending on the hardware optimisations for native execution however the differences found on the Tesla K80 were negligible with simulation difference of the order $10^{-5}$

| **Random Numbers Generated** | **Native** | **Non-Native** |
|---|---|---|
| 1,342,177,280,00 | 73.583048 | 11.793961 |

Table 8.3: Normal Random Number Execution Times

A problem that should be considered is that using native functions limits the user to floating precision numbers as native functions are often not supported on GPUs in double precision; this was the case for the three graphics cards used in this project.

## 8.7 Gaussian Model

It is necessary to produce an optimised Gaussian model to compare the performance of the discrete-space models against. Similar to the binomial models it is necessary to consider the differences between using single precision and double precision. Incorporated into this single precision and double precision comparison is the use of native functions for the single precision simulation.

Figure 8.5 makes it evident that the size of the error between floating precision and double precision for Gaussian walks follows the relationship found in the binomial models of increasing errors for decreasing step sizes. This further supports the claim that the basis of the error is due to the quantisation error of $dt \to 0$, which in turn is propagated based on the parameters of the underlying stock.

Hence it is advantageous to use single precision calculations as the Gaussian *NM Space* considered is below $10^2$ steps in this project which results in error less than $10^{-4}$.

## 8.8 Multinomial Based Model

Based on the results found for the speed of local memory and the improved convergence properties of multinomial models over binomial models a well designed optimisation method with small reductions

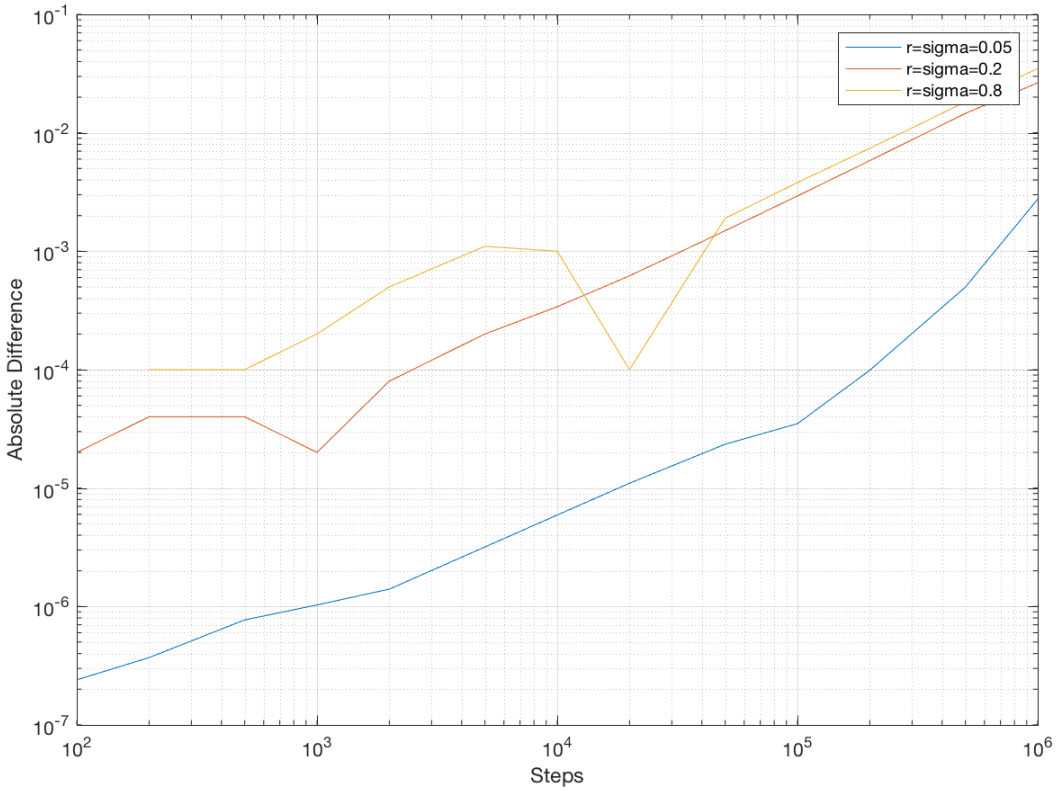



Figure 8.5: Absolute difference between single precision and double precision for Gaussian

in performance is likely to result in a faster option pricer.

### 8.8.1 Compressed Cox-Ross-Rubinstein Multinomial Model

The Cox-Ross-Rubinstein multinomial model requires an array of multipliers and an array of probabilities. The arrays are initialised using the $u, d$, and $p$ values from a binomial model with small steps size. The OpenCL code for initialising these arrays is:

```
float multipliers[4];
uint probabilities[4];

for (uint j = 0; j < 4; j++) {
   probabilities[j] = binomial(3, j) * std::pow(p, j) * std::pow((1 - p), 3- j) ...
       *std::numeric_limits<uint>::max();
   multipliers[j] = std::pow(u, j) * std::pow(d, 3 - j);
}

for (uint j = 1; j < 3; j++) {
   probabilities[j] += probabilities[j - 1];
}
```

The difference between single precision and double precision is not considered for the implied multinomial models as all branch multipliers are generated in double precision on the CPU. The *NM Space* calculated shows that the required number of steps is less than the equivalent binomial model implying that the precision error will be smaller than it was for the underlying binomial model.

The additional branches in the model introduce additional complexity in selecting the multiplier based on the random number generated. The OpenCL implementation for selecting the appropriate branch multiplier for a quadnomial tree is detailed below:

```
if(random<probabilities[1])
{
    if(random<probabilities[0])
        {
        currentValue=multipliers[0];
        }
    else
        {
        currentValue=multipliers[1];
        }
}
else
{
    if(random<probabilities[2])
        {
        currentValue=multipliers[2];
        }
    else
        {
        currentValue=multipliers[3];
        }
}
```

For the implied Cox-Ross-Rubinstein multinomial models the number of conditional statements will grow logarithmically, and the memory will grow linearly with the number of branches in the multinomial tree.

### 8.8.2 Compressed Jarrow-Rudd Multinomial Model

The implied Jarrow-Rudd multinomial model offers more interesting approaches to storing the multipliers in memory. An array with size equal to the total number of possible walks that can be taken is a plausible model as the number of occurrences of the each multiplier relative to the number of arrays will match their probability distribution. Alternatively a smaller array could be used in which all of the end branch multipliers. The two methods are outlined below.

```
//Small Array
float multipliers[3] = {u*u,u*d,d*d};
currentValue=currentValue*multipliers[popcount(random&0b11)];


//Large Array
float multipliers[4] = {u*u,u*d,u*d,d*d};
currentValue=multipliers*[random&0b11];
```

The small array method is slower than the larger array method as the popcount function on the GPU is slower than directly accessing local memory based using the bit selected random number. This increase in speed comes at the cost of exponential memory growth for the large array method making it infeasible for large trees.

For multinomial trees with equally probable branches the number of bits that must be sampled grows logarithmically based on the number of branches. There is as such a trade off between a better representation of a continuous space in return for slower implied random number generation speed.

### 8.8.3 Normal Sampled Multinomial Model

The normal sampled multinomial method considers equal spaced samples of the inverse normal cumulative distribution. This selection ensures that the average of the samples is zero. Then from this point the standard deviation can be calculated and the discrete normal values scaled appropriately

to match the the expected standard deviation. These discretised normal values are then used to calculate the branch multipliers of the multinomial tree. The C++ CPU code for the generation of these multipliers is show below:

```
uint branches = 64;
float discreteNormals[64];
float multipliers[64];
double mean = 0;
double standardDeviation = 0;
for (uint i = 0; i < 64; i++) {
   double probabilityValue =(i * (1.0 / branches) + (i + 1) * (1.0 / branches)) / 2.0;
   discreteNormals[i] = NormalCDFInverse(probabilityValue);
   mean += discreteNormals[i];
}

mean/= 64.0;
for (uint i = 0; i < 64; i++) {
   standardDeviation += (discreteNormals[i] - mean) * (discreteNormals[i] - mean);
}
standardDeviation = sqrt(shiftsStandardDeviation / 64.0);

for (uint i = 0; i < 64; i++) {
   discreteNormals[i] = discreteNormals[i] / shiftsStandardDeviation;
   multipliers[i] = exp((interestRate - 0.5 * sigma * sigma) * dt + sigma * sqrt(dt) * ...
       discreteNormals[i]);
}
```

The multinomial sampled method takes the best of both the Jarrow-Rudd multinomial memory storage models, as the number of branches is the size of the array required and the equal probability allows for direct access using the bit sampled random number as the index.

The primary trade off that needs to be considered for the multinomial sampled method is that as the number of branches increase the approximation of the Gaussian model improves, but this comes at the cost of more expensive random number generation and increase usage requirement. When the convergence of the models was analysed it was found that 64 was appropriate discretisation size for the *NM Space* considered in this project.

## 8.9 Antithetic Variates

The generation of antithetic pairs is a straightforward process for all of the approaches considered however there are slight differences in how the pair value is generated.

```
// Gaussian
currentValue*= native_exp(alpha + beta * normal);
currentValueAntithetic*= native_exp(alpha - beta * normal);


// Equal Probability
currentValue*= shifts[random & 0b11];
currentValueAntithetic*= shifts[(0xffffffff - random & 0b11)];


// Cox-Ross-Rubinstein
uint pinv = 0xffffffff - p;
currentValue = random < p ? currentValue*u : currentValue*d;
currentValueAntithetic = pinv < random ? currentValueAntithetic*u : ...
    currentValueAntithetic*d;
```

The antithetic method reduces the number of random numbers that need to be generated so immediately it might seem as if there will always be a speed up; however this is not strictly the case. For

models based on local memory, the antithetic path will be accessing locations on the opposite side of the local memory and due to the nature of the memory hierarchy there is potential that antithetic sampling will counteract caching. To mitigate this issue antithetic multipliers can be placed next to once another resulting in a better cache hit rate. The OpenCL code below outlines how this could be implemented:

```
//Antithetic friendly memory solution
shifts[4] = {u*u, d*d, u*d, u*d}
uint index = random & 0b11
currentValue*= shifts[index];
currentValueAntithetic*= shifts[index-1+(2*index%2)];
```

## 8.10 Control Variates

The primary components of the control variate technique are the same as the previously mentioned methodologies however in order to calculate the coefficient for the difference between the control variate and the Asian arithmetic option the covariance and variance of the samples needs to be calculated rapidly.

Below is the OpenCL code used to calculate the variance of the call prices buffer. It is similar to the accumulation of the mean and reduces the amount of code that needs to be run on the CPU if it was run sequentially.

```
__kernel void varianceKernel(__global float *input, __global float *output,
                             uint const divisionSize, uint const inputSize,
                             float const mean) {
  uint n = get_global_id(0);
  float outputInternal = 0;
  if ((n + 1) * divisionSize <= inputSize) {
    for (uint i = 0; i < divisionSize; i++) {
      outputInternal += (input[n * divisionSize + i] - mean) *
                        (input[n * divisionSize + i] - mean);
    }
  } else {
    for (uint i = 0; i < (n + 1) * divisionSize - inputSize; i++) {
      outputInternal += (input[n * divisionSize + i] - mean) *
                        (input[n * divisionSize + i] - mean);
    }
  }
  output[n] = outputInternal;
}
```

The same approach was adopted for generating the covariance between two inputs.

There is a considerable difference in the complexity of calculating the geometric mean of a walk as opposed to the arithmetic mean. In order to calculate the geometric mean the sum of the logarithms is calculated at each step of the walk, this is an expensive operation and as such the native functionality is used to minimise the impact on execution time. The control variate walk is generated at the same time as the Arithmetic Asian option to avoid having to generate the stock price movement and random numbers twice as this would be unnecessary computational repetition.

# 9 Results and Evaluation

This chapter explores the performance and accuracy trade off characteristics of the GPU optimised solutions considered in the previous chapter. The models are compared with similar models and against the base model, the Gaussian walk with regular random number generation. The chapter proceeds to explore the larger relationships found with discrete-space results. Then the validity of the results is explored. Finally the results and performance are compared against existing technologies.

In order to assess the performance of the models the following assessments are considered:

- **Steps Throughput**: A measure of how many time steps the model can calculate per second. A time step is defined at the atomic discrete movement in time for a stochastically distinct models. In the control variate methods the arithmetic walk and the control variate walk are being stepped simultaneously, this is considered a single time step. In the antithetic variate method, stepping a pair of correlated walks counts as two time steps. The time steps throughput is calculated by taking the number of time steps taken, $NM$, and dividing through by the execution time. To produce a numerical value representative of the model, the mean of the step throughputs is calculated for three large simulations with varied step sizes.
- **Time Taken for to reach 99% Confidence Level Size**: Simulations are run with iterations in the range 10,000 to 100,000,000 and steps determined by the *NM Space* models devised in Chapters 5 and 6. This will enable generation of graphs that give a high level visual interpretation of the differences between models for different simulation sizes. These graphs all include a standard sampled Gaussian as a baseline performance reference for continuous models.
- **Execution Time for Given Accuracy**: Using the *NM Space* models devised in Chapters 5 and 6 the iterations and steps required to achieve a 99% confidence interval equal to a certain size are calculated. The simulations are run and then this enables comparison of execution times at specific accuracies which enables determination of speed ups.

An important note is that rather than taking the literal value returned by the *NM Space* model there is an arbitrary minimum number of steps set at four; this is as below this region the convergence of the model was not considered in the investigations.

When considering the execution time it is assumed that it is made up of:

- Loading the kernel $= f$
- Executing the kernel $\propto N \times M$
- Accumulating the results $\propto N$

Consequently on this basis, the approximation is made that for large simulations the execution time is dominated by the execution of the kernel and the other components become negligible.

## 9.1 Binomial Models

The binomial models are the appropriate entry point when considering the performance and accuracy characteristics of the discrete-space models, as they represent the coarsest and least computationally intensive models covered in this project. The first examination considers the difference between the Cox-Ross-Rubinstein models multiplication and memory based, the Jarrow-Rudd model and the base level Gaussian; this is followed by the same set but with antithetic sampling for the binomial models.

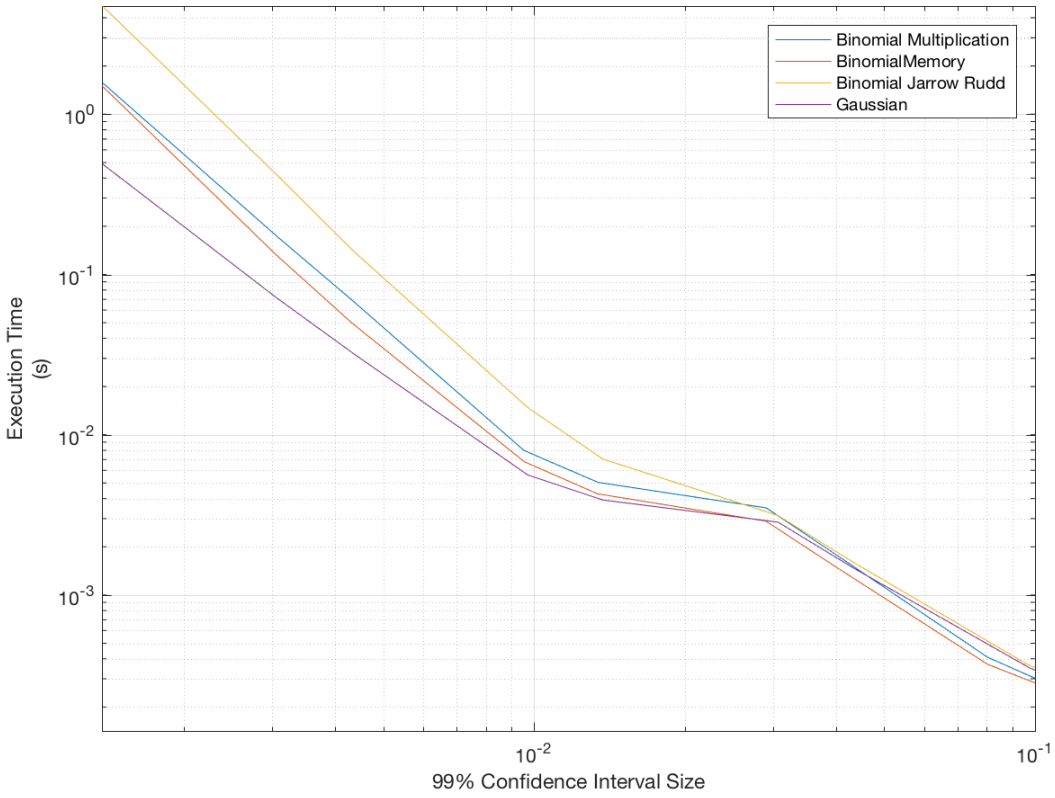



Figure 9.1: Binomials' execution times against confidence interval size

Figure 9.1 can be broken into three areas of interest. For the low accuracy results the execution time between the results is difficult to distinguish the performance as all models have a high proportion of execution time being dominated by the fixed kernel cost and are not making full use of the graphics card in accumulation and stepping. The first kink in the graph represents the point at which the iteration space is large enough to use all available cores of the graphics card. The second kink is when the accumulation step size changes to handle the larger number of iterations. Beyond this point the growth of the execution time for the discrete models is defined by the complexity of the *NMSpace* models. It is noticed that the advantage the binomial memory based model has over the multiplication model fades at a confidence interval size of $2 \times 10^{-3}$, as the size of the discrete model becomes large resulting in more probable cache misses for local memory.

Figure 9.2 shows the same models as Figure 9.1 however the binomial models now make use of antithetic sampling. There is a speed up over the non-antithetic model, however the majority of the comparative characteristics remain the same; however the point which the multiplication model becomes faster than the memory based model is earlier as the antithetic paths result in worse caching performance of the local memory model.

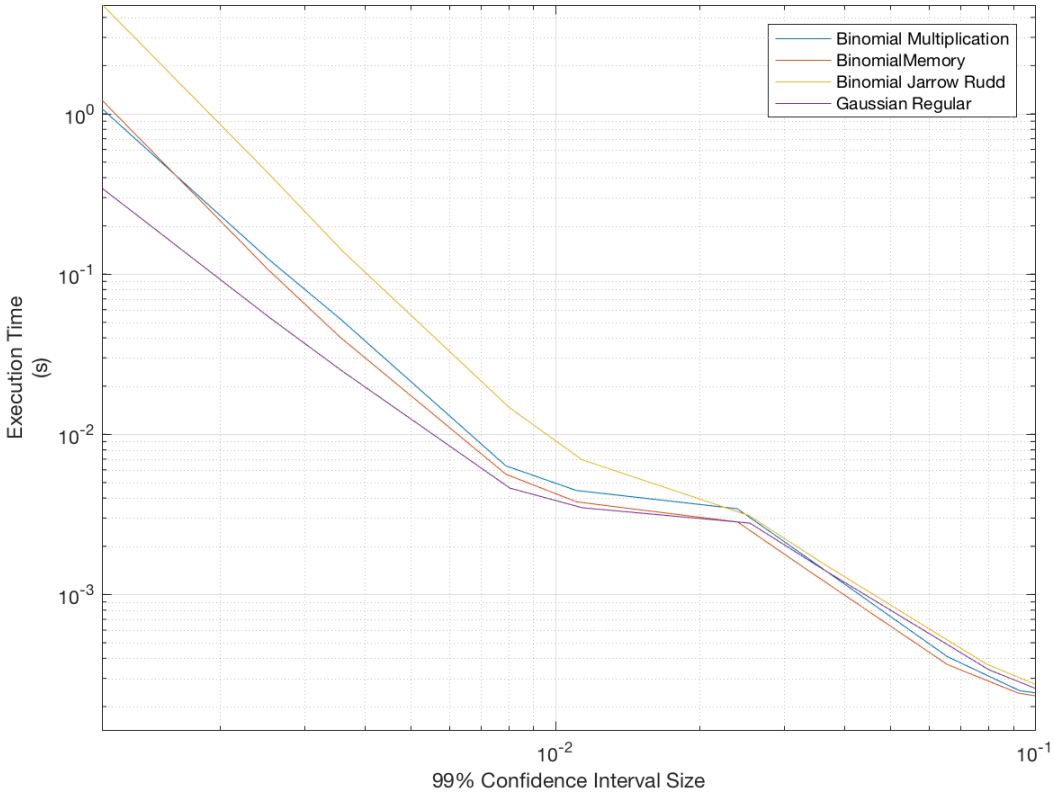



Figure 9.2: Antithetic binomials' execution times against confidence interval size

## 9.2 Multinomial Models

The multinomial models are of interest due to their convergence properties being better than their binomial counterparts suggesting they could potentially be faster for smaller confidences. It is of interest to determine whether these better convergence properties outweigh the deductions in performance.

Figures 9.3 and 9.5 display the similarities between the base continuous model and the larger multinomial models. As the number of branches of the models increases it is seen that they become more competitive with the Gaussian model for larger convergence size. The nonanomial Jarrow-Rudd model and the octanomial Cox-Ross-Rubinstein model are able to execute comparably with the Gaussian until an accuracy level of $5 \times 10^{-3}$ is reached, where the inherent complexity of the models outweighs their improvements in throughput. The sampled multinomial model exceeds the performance of the Gaussian model, however when antithetic sampling is introduced the difference between the two is reduced as the relative cost of generating the random normal sequence is more expensive hence the antithetic sampling has a bigger impact on the overall execution time of the Gaussian.

It is of interest to better understand the nature of the trade off for increasing the number of branches in the tree on the throughput of the execution. Figure 9.4 shows the difference in throughputs for difference sizes of compressed Jarrow-Rudd multinomial trees (Large Array) and normal sampled trees. The strength of the linear relationship between the multinomial trees and the overall throughput is clear to see. The normal sampled multinomial tree under-performs as relative to the line of best fit because it does not have an index size that is a factor of 32-bits, hence the random numbers are not used as efficiently as in the Jarrow-Rudd cases.

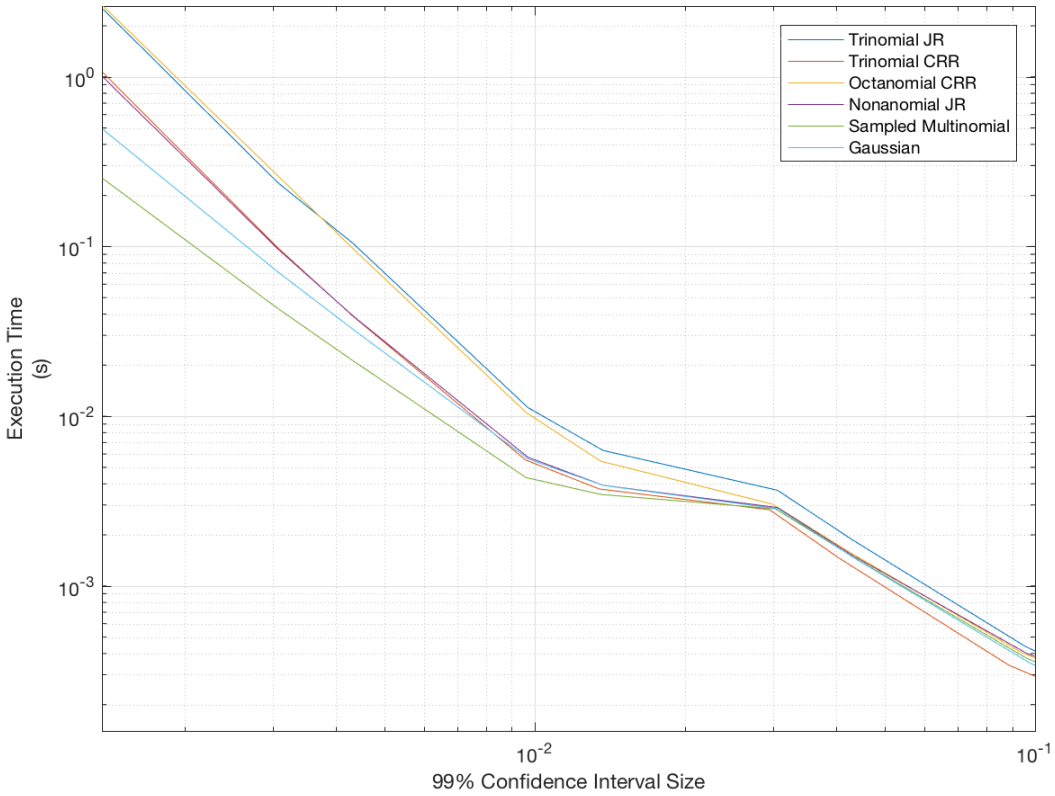



Figure 9.3: Multinomials' execution times against confidence interval size

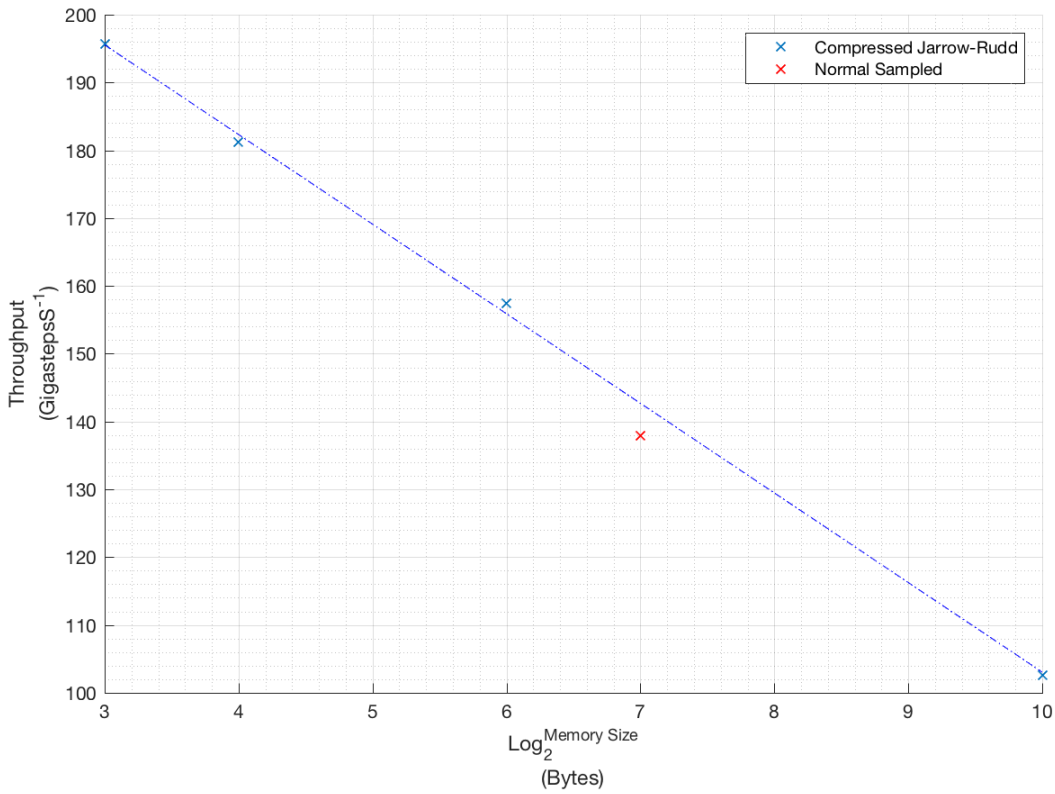



Figure 9.4: Time steps throughput against number of branches

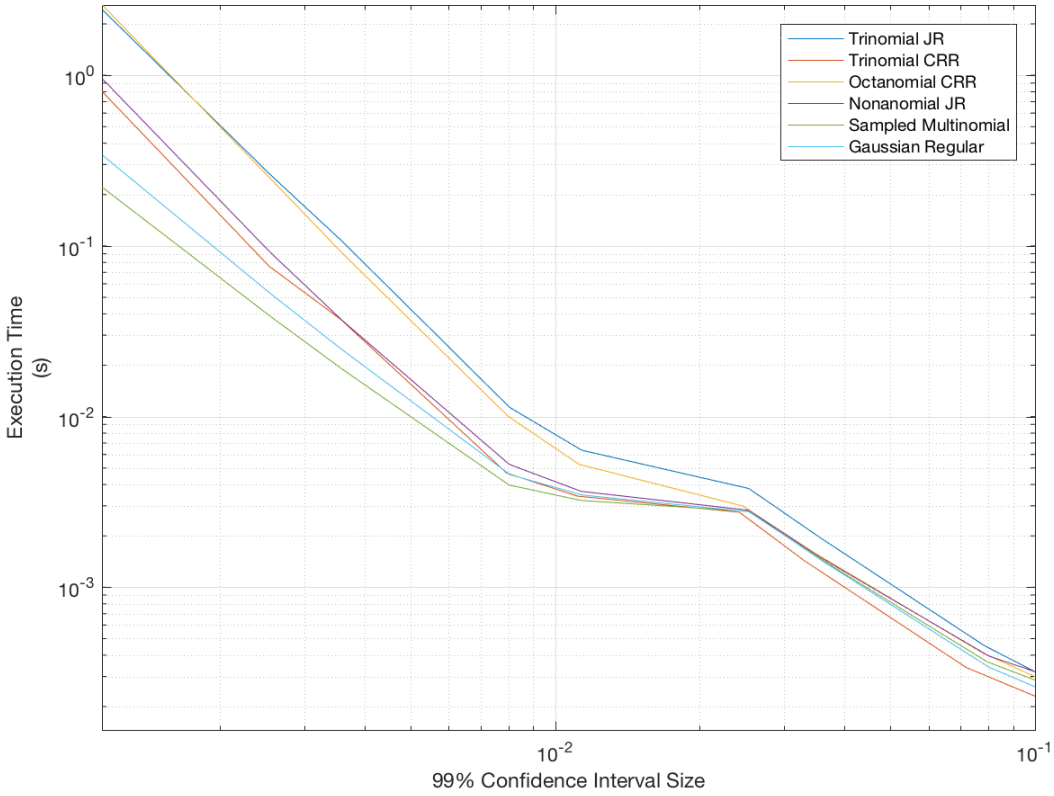



Figure 9.5: Antithetic multinomials' execution times against confidence interval size

## 9.3 Control Variate Techniques

The control variate methods were applied to methods that were deemed most representative of the underlying discrete models and the base model. The variance was distinctly lower than that of the regular models and as such it needs to be seen whether the additional complexity required to calculate the Monte Carlo Geometric walk doesn't result in a slowdown greater than the improvements brought about by the reduction in variance.

It can be seen in Figures 9.6 and 9.7 that the performance the geometric control variates is as much a 30x faster than the standard Gaussian approach. This highlights the effectiveness of using the geometric control variate method. The European control variate does not outperform the the regular Gaussian method for small simulations, however it starts to become faster for larger simulations. The speed up due to antithetic appears similar across all simulations, this is assumed that because the random generation takes up a small proportion of the time step complexity. The primary conclusion taken from this graph is that the geometric control variates perform better than European control variates and the regular based method. Of the four geometric control variate methods the performance for large confidence interval sizes is split into two groups, the binomial models both have the same complexity however the ratio of their steps is constant hence the Jarrow-Rudd model executes $2 \times faster$ as the *NM Space* grows. The overall strongest performer of the geometric control variate methods is the sampled multinomial approach which has a higher throughput for the same *NM Space* model as the continuous geometric control variate model. It is important to recognise that for larger confidence interval sizes the difference in the speeds of the geometric models is minimal suggesting that for GPUs where the native exponential functions and local memory are slow, it could be viable to consider using a binomial Geometric approach.

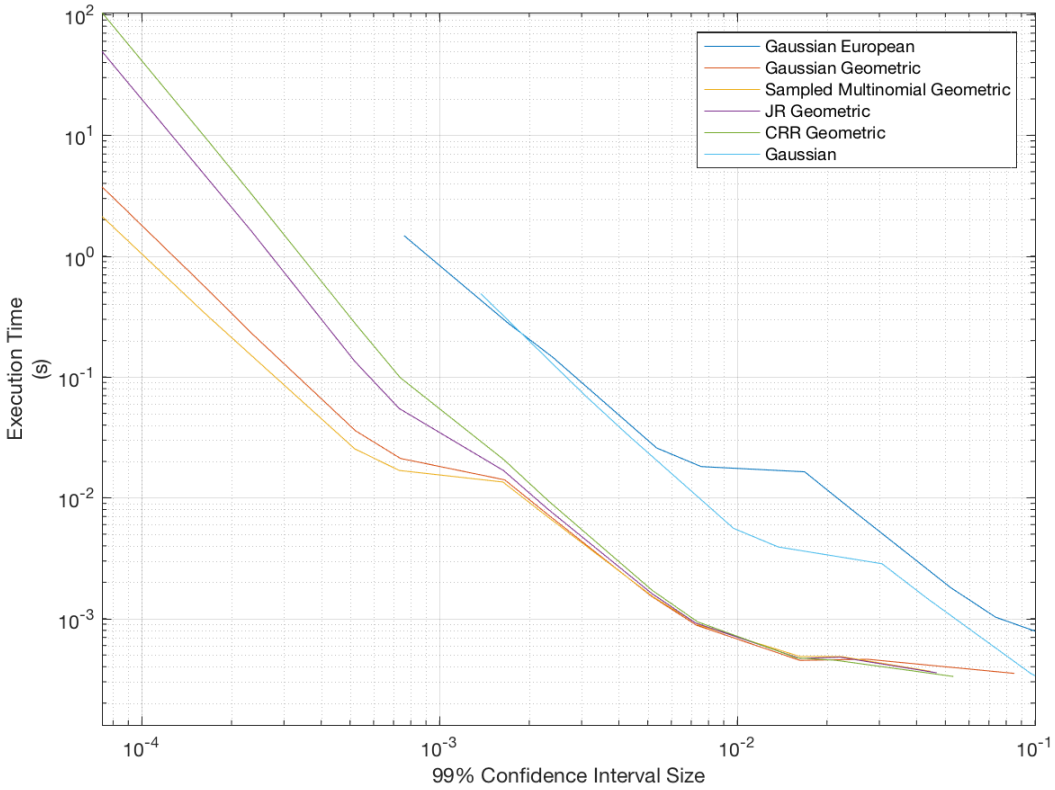



Figure 9.6: Control variate models' execution times against confidence interval size

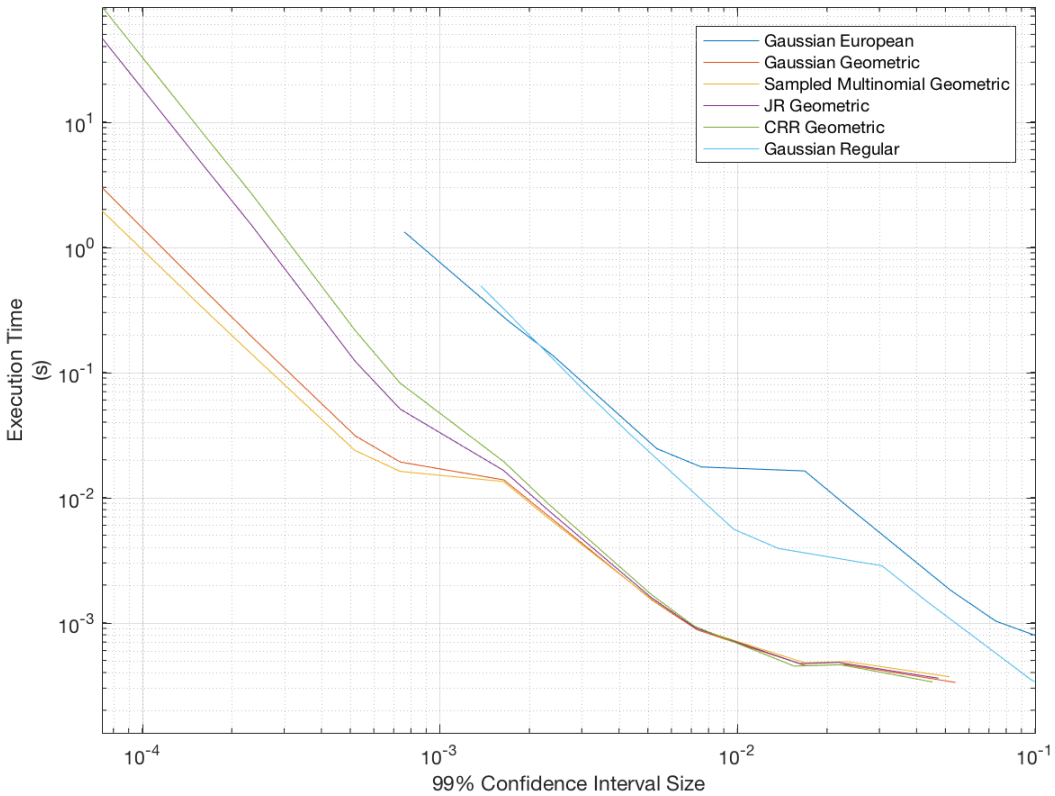



Figure 9.7: Antithetic control variate models' execution times against confidence interval size

## 9.4 Complete Comparison

The graphs in the previous section gave a good general overview of the properties of the models, this section now looks more closely at the numerical values of the speed ups relative to the Gaussian walk, and the antithetic speed ups relative to the regularly sampled Gaussian walk.

### 9.4.1 NVIDIA Tesla K80

| | **Regular** | | **Antithetic** | | |
|---|---|---|---|---|---|
| **Name** | **Throughput** ($GSteps/s$) | **Base Speed Up** | **Throughput** ($GSteps/s$) | **Antithetic Speed Up** | **Base Speed Up** |
| Binomial JR | 195.641 | 4.456 | 195.189 | 0.998 | 4.446 |
| Trinomial JR | 181.204 | 4.127 | 188.696 | 1.041 | 4.298 |
| Sampled Normal | 137.855 | 3.140 | 160.174 | 1.162 | 3.648 |
| Binomial JR Double | 120.055 | 2.735 | 119.009 | 0.991 | 2.711 |
| Binomial JR Geometric CV | 104.011 | 2.369 | 111.145 | 1.069 | 2.532 |
| Binomial CRR | 103.955 | 2.368 | 161.357 | 1.552 | 3.675 |
| Trinomial CRR | 102.797 | 2.341 | 140.984 | 1.371 | 3.211 |
| Nonanomial JR | 102.558 | 2.336 | 101.786 | 0.992 | 2.318 |
| Binomial CRR Memory Local | 99.271 | 2.261 | 112.912 | 1.137 | 2.572 |
| Sampled Normal Geometric CV | 82.898 | 1.888 | 96.315 | 1.162 | 2.194 |
| Binomial CRR Double | 77.988 | 1.776 | 97.797 | 1.254 | 2.228 |
| Binomial CRR Geometric CV | 71.726 | 1.634 | 88.753 | 1.237 | 2.022 |
| **Gaussian** | 43.902 | 1.000 | 69.785 | 1.590 | 1.590 |
| Gaussian European CV | 42.591 | 0.970 | 67.218 | 1.578 | 1.531 |
| Gaussian Geometric CV | 35.428 | 0.807 | 48.367 | 1.365 | 1.102 |
| Octanomial CRR | 17.968 | 0.409 | 18.028 | 1.003 | 0.411 |
| Binomial CRR Memory Global | 8.076 | 0.184 | 8.349 | 1.034 | 0.190 |
| Gaussian Double | 6.141 | 0.140 | 8.167 | 1.330 | 0.186 |

Table 9.1: NVIDIA Tesla K80 throughputs

Examining the throughputs shown in Table 9.1, it is evident that all of the discrete models have

greater throughput than the base Gaussian with the exception of the Binomial Cox-Ross-Rubinstein model using global memory and the implied Cox-Ross-Rubinstein octanomial model. The first is self explanatory based on the results of the previous chapter comparing global memory performance to that of local memory. The octanomial is much slower than the binomial and trinomial models also built on the Cox-Ross-Rubinstein models as those two models both have probabilities and branch multipliers passed in as kernel arguments whereas the octanomial uses two local memory stores as well as having three layers of conditional statements.

Unsurprisingly the Gaussian based control variate models and the double precision Gaussian model have lower step throughputs, as the operations involved in calculating each step are greater. Interestingly the discrete binomial control variate models still have a higher throughput than the base model even with the additional complexity of calculating the Geometric control variate path.

There is as expected a speed up for or equal, *within margin of error*, throughput performance for antithetic sampling. The throughput increase is greater in models where the random number generation makes up a larger proportion of the execution time, as highlighted by the $\approx 1.5\times$ speed up for both the Gaussian and binomial Cox,Ross and Rubinstein model which both require one random number to be generated for each time step.

Table 9.2 and 9.3 show the effectiveness of the models trade off between convergence properties and throughput by considering the execution time for given confidence levels. The first table considers an accuracy level that enables a comparison of all the methods however it doesn't provide enough differentiation between the geometric control variate methods hence a finer accuracy is considered in a second table.

The only discrete method with a speed up over the Gaussian model is the sampled normal multinomial tree, with a speed up of $1.8\times$. The other discrete models are considerably slower, although the larger the number of branches in the Jarrow-Rudd based multinomial models, the closer it comes to matching the performance of the base Gaussian. The Cox-Ross-Rubinstein models do not have the such a distinct pattern with the trinomial model outperforming the binomial but the octanomial coming between the two. This is a consequence of the differences in the trade offs between the steeper reduction in throughput in comparison to the improvements in model convergence in *NM Space*.

The antithetic execution time is less for all models as the variance reduction of the model is combined with the faster random number generation leading to two factors reducing the execution time required reach a confidence level size.

According to the results in Table 9.3 for high levels of accuracy the geometrics, the normal sampled multinomial model outperforms the Gaussian with $1.7\times$ speed up. Due to the larger number of steps required to get convergence for the Cox-Ross-Rubinstein model when using a Geometric control variate than the Jarrow-Rudd model there is a noticeable $2\times$ difference in execution times.

## 9.5 Trade Off between Throughput and Steps Required

Figure 9.8 displays the overall properties of attempting to solve the pricing of an Asian option price using discrete-time Monte Carlo methods. There is a clear separation between the control variate methods on the left and the regular methods on the right hand side of the figure. The model shows how in order to reduce the number of time steps required the cost of generating a time step must increase.

| | Regular | | Antithetic | | |
|---|---|---|---|---|---|
| **Name** | **Execution Time for $\epsilon = 2 \times 10^{-3}$** | **Base Speed Up** | **Execution Time for $\epsilon = 2 \times 10^{-3}$** | **Antithetic Speed Up** | **Base Speed Up** |
| Sampled Normal Geometric CV | 0.010 | 18.684 | 0.010 | 1.007 | 18.822 |
| Gaussian Geometric CV | 0.011 | 17.592 | 0.010 | 1.044 | 18.364 |
| Binomial JR Geometric CV | 0.012 | 15.727 | 0.012 | 1.029 | 16.177 |
| Binomial CRR Geometric CV | 0.019 | 9.967 | 0.014 | 1.369 | 13.645 |
| Sampled Normal | 0.106 | 1.794 | 0.063 | 1.671 | 2.997 |
| Gaussian European CV | 0.116 | 1.633 | 0.103 | 1.127 | 1.840 |
| **Gaussian** | 0.190 | 1.000 | 0.088 | 2.147 | 2.147 |
| Nonanomial JR | 0.307 | 0.618 | 0.174 | 1.768 | 1.093 |
| Trinomial CRR | 0.330 | 0.575 | 0.150 | 2.204 | 1.268 |
| Binomial CRR | 0.480 | 0.396 | 0.178 | 2.692 | 1.066 |
| Binomial CRR Memory Local | 0.485 | 0.392 | 0.208 | 2.332 | 0.914 |
| Binomial CRR Double | 0.656 | 0.289 | 0.288 | 2.277 | 0.659 |
| Trinomial JR | 0.776 | 0.245 | 0.429 | 1.808 | 0.443 |
| Octanomial CRR | 0.866 | 0.219 | 0.510 | 1.700 | 0.373 |
| Gaussian Double | 1.075 | 0.177 | 0.528 | 2.035 | 0.360 |
| Binomial JR | 1.573 | 0.121 | 0.881 | 1.785 | 0.216 |
| Binomial JR Double | 2.447 | 0.078 | 1.413 | 1.732 | 0.134 |
| Binomial CRR Memory Global | 5.351 | 0.036 | 3.010 | 1.778 | 0.063 |

Table 9.2: NVIDIA Tesla K80 execution times

| | Regular | | Antithetic | | |
|---|---|---|---|---|---|
| **Name** | **Execution Time for $\epsilon = 1 \times 10^{-4}$** | **Base Speed Up** | **Execution Time for $\epsilon = 1 \times 10^{-4}$** | **Antithetic Speed Up** | **Base Speed Up** |
| Sampled Normal Geometric CV | 1.082 | 1.673 | 0.994 | 1.088 | 1.821 |
| **Gaussian Geometric CV** | 1.810 | 1.000 | 1.447 | 1.251 | 1.251 |
| Binomial JR Geometric CV | 19.949 | 0.091 | 18.929 | 1.054 | 0.096 |
| Binomial CRR Geometric CV | 42.351 | 0.043 | 33.118 | 1.279 | 0.055 |

Table 9.3: NVIDIA Tesla K80 execution times for finer accuracies

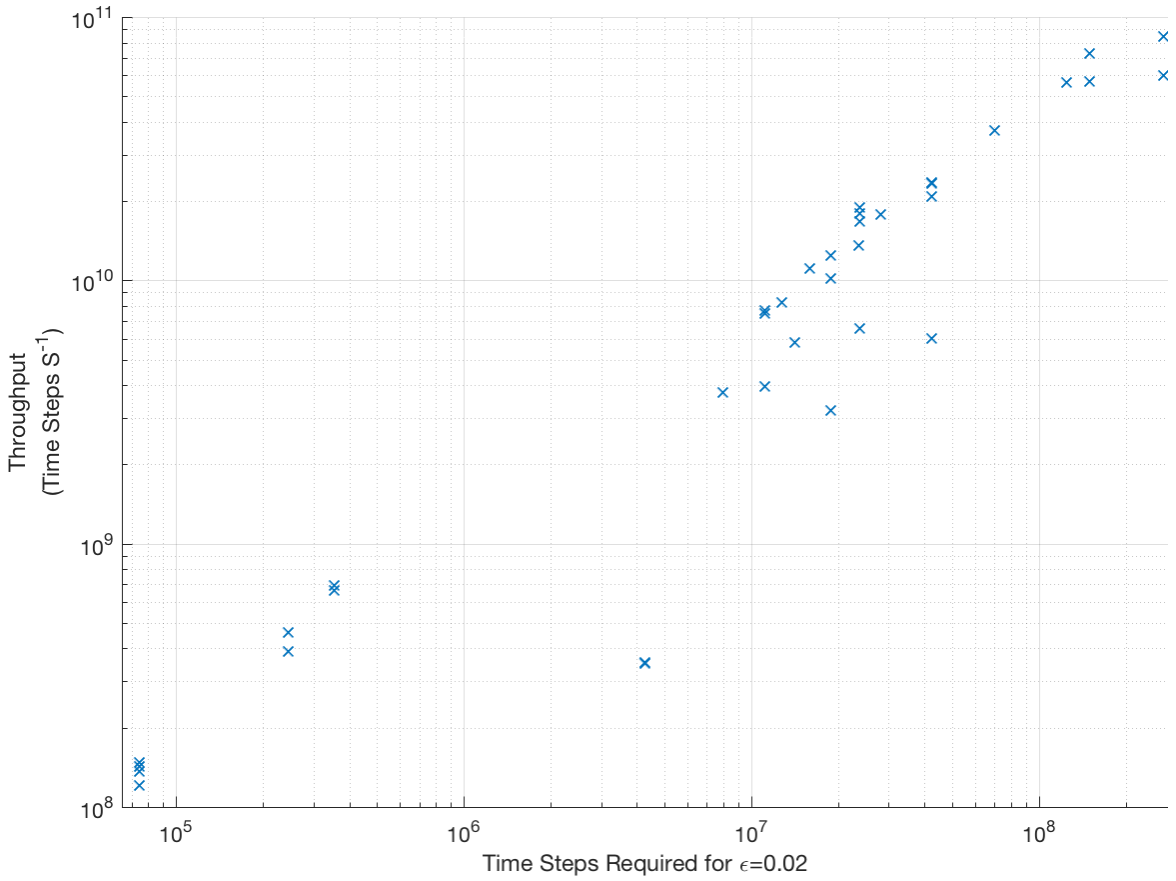



Figure 9.8: Throughput against steps required

This numerical comparison of models leads to the following intermediary conclusions, it is always beneficial to use antithetic variates over regular random number generation. The fastest non-control variate based method was the sampled multinomial tree, and the fastest control variate method was the sampled multinomial tree with a geometric control variate.

## 9.6 Verification of the Sufficient *Randomness* of Random Number Generator

Confidence that the stochastic sequence generated by the pseudo-random number generator are sufficiently random is required to avoid any bias which could consequently lead to false conclusions being drawn about the models convergence properties.

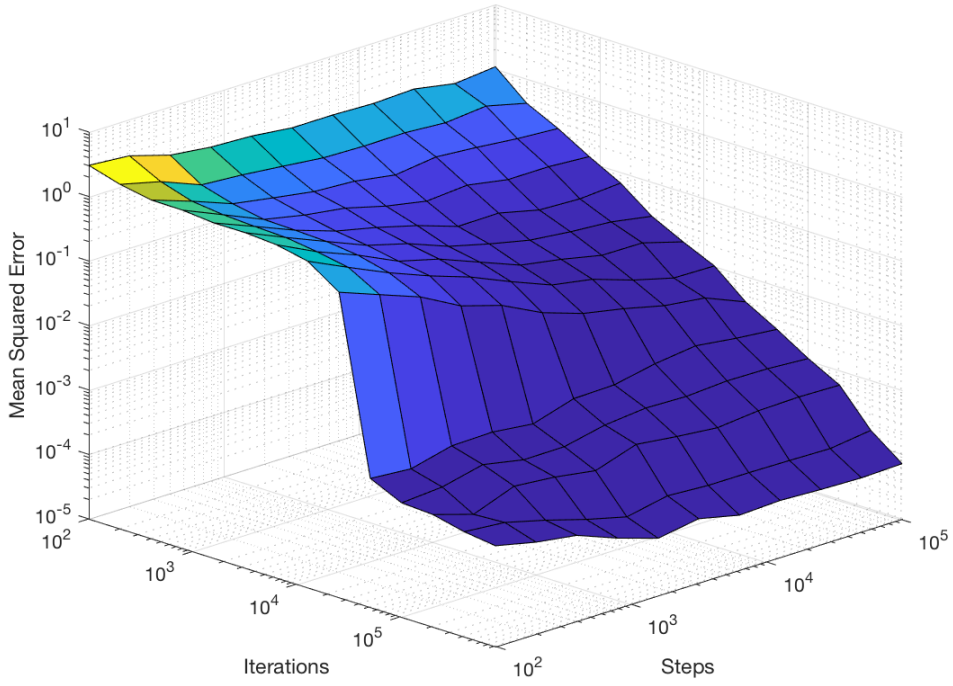



Figure 9.9: Inadequately Seeded Monte Carlo Simulation in NM Space

Initially the random numbers were seeded using the kernel identifier as the basis of the seed. Figure 9.9 shows the effect of this on the *NM Space*. It shows a false correlation between increasing steps for low iterations and reducing RMSE. Increasing the number of steps is removing the impact of the closely correlated initial steps of the poorly seeded random number generator. The key piece of information that gives away the failure of this model is the break from the central limit theorem convergence of iterations. This break in convergence represents the point at which the kernel numbers being used are large enough that their most significant bits are no longer zero and as such the random numbers correlation dissipates more rapidly. Hence for the models executed the convergence tightly coupled with the central limit theorem convergence is indicative of the sufficient randomness of the random numbers.

## 9.7 Verification of Convergence

An important part of the project is being able to prove that the call prices generated by the Monte Carlo simulations are the correct values for Asian options.

The initial step to determine the correctness of the model is when writing a pricing engine the kernel, being by pricing European options instead of Asian options. This could then be tested using a range of values within the range considered for the scope of this project to determine if the confidence intervals of large step iterations matched the expected values produced by the Black-Scholes model. It is assumed that in moving from pricing a European option to pricing an Asian option no error is introduced, however in order to further confirm the validity of the results produced by others were examined to determine whether call prices calculated matched those found externally.

In Many-Core Programming with Asian Option Pricing [28], Li considers the Asian option $S_0 = 100, K = 95, \sigma = 0.03, r = 0.2$, and $T = 2$, and calculates the value of the Call Price to be \$10.6022. All of the models considered in this paper were run with at least 500 steps and 4,000,000 iterations and all resulted in a call price value with \$10.6022 within its confidence interval.

An additional comparison that was used, is the *MATLAB* built in Asian option pricing library [25]. This enabled multiple options to be tested for the models to determine correctness of call price values generated by the GPU solutions using at least 500 steps and 4,000,000 iterations.

## 9.8 Comparison with Existing Technologies

The results of this paper have standalone interest but their relative performance is based on a Gaussian model developed for this report. To display the effectiveness of the models, there needs to be comparisons to existing implementations. This section considers comparative performance against implementations executed by third parties on GPUs and FPGAs. The comparisons will consider the simulation most similar to that in the paper be on execution times and step throughputs however further arguments will be made based on these findings and paper context.

### 9.8.1 Altera

The results found in *Monte Carlo Pricing of Asian Options on FPGAs using OpenCL* [1] found that using an Altera Stratix V D8 could result in $1.2 \times 10^{10}$ steps per second, and that a leading 26nm GPU could only generate $1.01 \times 10^{10}$ steps per second. This paper's optimised Gaussian can generate $1.686 \times 10^{10}$ steps per second. Assuming that the Gaussian Monte Carlo walks are both optimised equally well, this suggests a performance gap between the graphics card used by Altera and the NVIDIA K560 of approximately 0.599. All results are hence scaled by this factor in order to allow for a justifiable comparison between the K560 and the FPGA. Given the method presented by Altera does not make use of control variates the comparisons put forward are the regular and antithetic sampled multinomial models; this is shown in Table 9.4. After scaling the GPU figures, it is calculated that the improvement in performance for antithetic multinomial models allows for a 17% increase in step power efficiency compared to the FPGA Gaussian walk implementation.

### 9.8.2 Anson H. Tse FPGA Implementation

Tse [2, 40] implements a Gaussian simulation on an FPGA with 3650 steps and 10,000,000 iterations and a Gaussian Monte Carlo control variate simulation using 365 Steps and 1,000,000 iterations. A limitation of this comparison is the lack of definitive guidelines for comparing two distinctly different pieces of hardware, an FPGA and an integrated GPU. Table 9.5 shows the comparison between the

| **Hardware** | **Model** | **Performance** ($Steps/s$) | **Power** ($W$) | **Performance / Power** ($Steps/s/W$) |
|---|---|---|---|---|
| Altera Stratix V D8 | Gaussian | $1.2 \times 10^{10}$ | 45 | $2.66 \times 10^{8}$ |
| Altera 26nm GPU | Gaussian | $1.01 \times 10^{10}$ | 212 | $4.88 \times 10^{7}$ |
| NVIDIA K560 | Gaussian | $1.686 \times 10^{10}$ | 225 | $7.49 \times 10^{7}$ |
| NVIDIA K560 | Sampled Multinomial | $6.801 \times 10^{10}$ | 225 | $3.02. \times 10^{8}$ |
| NVIDIA K560 | Sampled Multinomial Antithetic | $9.893 \times 10^{10}$ | 225 | $4.40 \times 10^{8}$ |
| NVIDIA K560 Scaled | Sampled Multinomial | $4.08 \times 10^{10}$ | 225 | $1.81 \times 10^{8}$ |
| NVIDIA K560 Scaled | Sampled Multinomial Antithetic | $5.93 \times 10^{10}$ | 225 | $3.11 \times 10^{8}$ |

Table 9.4: Altera performance comparison with sampled multinomial methods

FPGA and the integrated Intel Iris 5100 GPU. The antithetic sampled multinomial results in a speed up of 6.17× and an energy reduction of 86% over Anson's Gaussian simulation; when considering control variates there is 4.6× speed up and an energy reduction of 60% over Anson's Gaussian control variate simulation.

| **Hardware** | **Model** | **Execution Time** ($s$) | **Power** ($W$) | **Execution Time** × **Power** ($Ws$) |
|---|---|---|---|---|
| Virtex-5 xc5vlx330t | Gaussian | 11.4 | 29 | 330.6 |
| Virtex-5 xc5vlx330t | Gaussian Control Variate | 0.184 | 29 | 5.336 |
| Intel Iris 5100 | Gaussian | 4.36 | 30 | 130.8 |
| Intel Iris 5100 | Gaussian Control Variate | 0.10 | 30 | 3 |
| Intel Iris 5100 | Sampled Multinomial | 1.83 | 30 | 54.9 |
| Intel Iris 5100 | Sampled Multinomial Antithetic | 1.47 | 30 | 44.1 |
| Intel Iris 5100 | Sampled Multinomial Control Variate | 0.055 | 30 | 1.65 |
| Intel Iris 5100 | Sampled Multinomial Control Variate Antithetic | 0.040 | 30 | 1.2 |

Table 9.5: Comparison of models with Tse's FPGA implementations

### 9.8.3 FPGA Discrete Monte Carlo Implementation

The basis of this paper was Fabry's paper [9] on discrete Cox-Ross-Rubinstein Monte Carlo methods for FPGAs. As such it is relevant to determine the comparative performance of discrete-space algorithms on GPUs against FPGAs to determine if it's an appropriate approach for the architecture.

The approach for generating this comparison is outlined below, the results can be found in Tables 9.6 and **??**.

Fabry displays empirical results for his discrete Cox-Ross-Rubinstein and Gaussian values with:

$$10^0 < RMSE < 10^{-2} \tag{9.1}$$

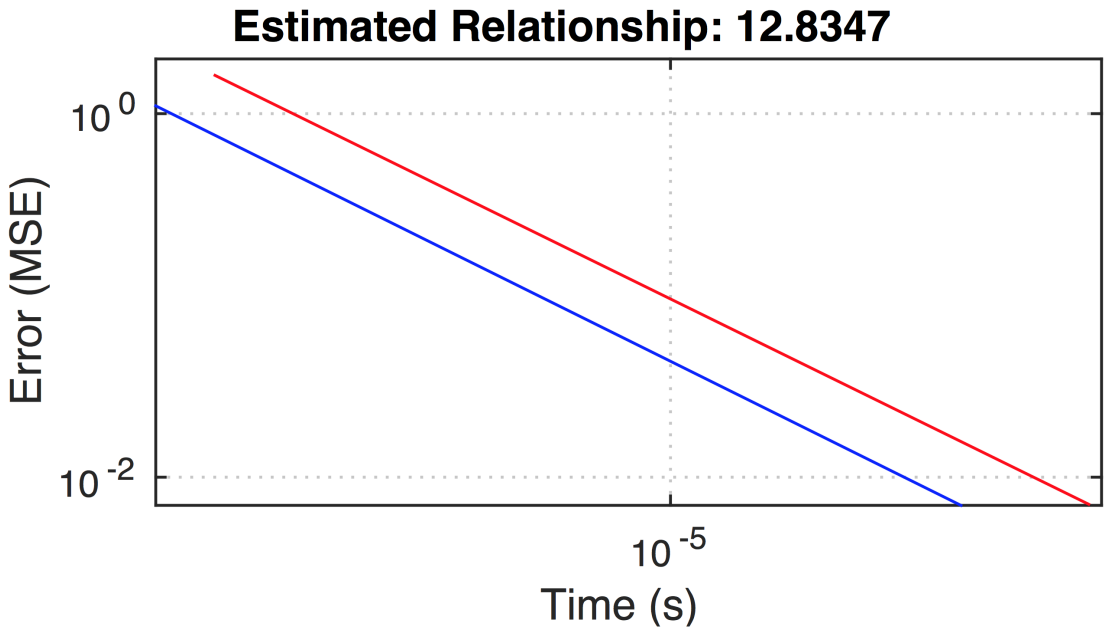



Figure 9.10: Relationship between Gaussian and Binomial presented by Fabry [9]

He does not provide the option properties used to obtain these results and sadly this range of execution times cannot be beaten irrespective of the option's parameter as the fixed cost of the kernel load time is greater than this magnitude of execution times. To compare the sampled multinomial method presented in this paper with the discrete space model presented by Fabry, it is necessary to make a range of assumptions about the methodologies he used in order to formulate an appropriate comparison.

It is known that Fabry defines his *NM Space* as:

$$BinomialRatio = e^{-2.1911}(NM)^{-0.22619} \tag{9.2}$$

$$GaussianRatio = e^{0.3933}(NM)^{-0.38247} \tag{9.3}$$

As such throughout this section all of implementations considered *NM Space*, including those from this paper, will be assumed to use Fabry's method.

To determine more information about the performance characteristics of Fabry's Gaussian model the assumption is made that the Gaussian model executed by Fabry has the same performance characteristics of that presented by Tse [40] as it is run on the same FPGA with the same number of Gaussian kernels and frequency.

From Figure 9.10 it is difficult to ascertain the scale of the time values due to the absence of markers on the time axis aside from the marker $10^{-5}$, hence this execution time is used to calculate the performance properties of his model. To add to further confusion in the background research Fabry defines MSE as RMSE, hence the y-axis of the graph is interpreted as RMSE and not MSE.

It is known that Tse's Gaussian Walk had throughput:

$$\frac{3650 \times 1000000}{11.4} = 3.2 \times 10^9 \tag{9.4}$$

From the throughput information generated from Tse's paper and the execution time provided by Fabry it is determined that the time steps taken were:

$$Throughput = \frac{TimeSteps}{ExecutionTime} \implies 3.2 \times 10^9 = \frac{NM}{10^{-5}} \implies NM = 3.2 \times 10^4 \tag{9.5}$$

Using the Gaussian Ratio presented by Fabry gives:

$$GaussianRatio = e^{0.3933}(3.2 \times 10^4)^{-0.38247} = 0.028 \tag{9.6}$$

$$N = 1070 \tag{9.7}$$

$$M = 30 \tag{9.8}$$

Equating the standard deviation of the normal distribution implied by central limit theorem and the RMSE from Fabry's graph allows for the approximate estimation of the standard deviation of the call price considered by Fabry.

$$RMSE = \frac{\sigma}{\sqrt{N}} \implies 10^{-1} = \frac{\sigma}{\sqrt{1070}} \implies \sigma = 3.271 \tag{9.9}$$

The standard deviation of the call price for the Gaussian model and the Binomial model are assumed to be the same such that the iterations required by the binomial model can be calculated as:

$$RMSE = \frac{\sigma}{\sqrt{N}} \implies 4 \times 10^{-2} = \frac{3.271}{\sqrt{N}} \implies N = 6688 \tag{9.10}$$

Using the Binomial Ratio presented by Fabry gives:

$$BinomialRatio = e^{-2.1911}(BinomialRatio \times N^2)^{-0.22619} \implies BinomialRatio = 0.00649 \implies M = 44 \tag{9.11}$$

Hence:

$$Throughput = \frac{TimeSteps}{ExecutionTime} \implies Throughput = \frac{6688 \times 44}{10^{-5}} \implies Throughput = 2.943 \times 10^{10} \tag{9.12}$$

| **Hardware** | **Model** | **Performance** ($Steps/s$) | **Power** ($W$) | **Performance / Power** ($Steps/s/W$) |
|---|---|---|---|---|
| Virtex-5 xc5vlx330t | Binomial CRR | $2.943 \times 10^{10}$ | 29 | $1.015 \times 10^9$ |
| NVIDIA K80 | Binomial CRR | $1.03 \times 10^{11}$ | 300 | $3.43 \times 10^9$ |

Table 9.6: Comparison of GPU binomial implementation with Fabry's FPGA implementations

Consequently it is determined that for the binomial Cox-Ross-Rubinstein model executed on the NVIDIA K80 has $\approx 3\times$ greater throughput but $\approx 10\times$ greater power consumption, resulting in a $\approx 3\times$ reduction in the performance per watt for the binomial Cox,Ross, and Rubinstein model.

To compare the sampled multinomial model to the binomial model the convergence differences have to be considered. In order to present a fair comparison in which the kernel load time is not the limiting factor the implied execution times based on throughput are calculated based on the convergence models presented in Fabry's paper, for RMSE size $10^{-3}$

$$RMSE \approx \frac{\sigma}{\sqrt{N}} \implies 10^{-3} = \frac{3.271}{\sqrt{N}} \implies N = 1.07 \times 10^{7} \tag{9.13}$$

Hence the steps implied by the ratios for the Gaussian and binomial models are:

$$M_{Binomial} = 4570, M_{Gaussian} = 1820 \tag{9.14}$$

| **Hardware** | Virtex-5 xc5vlx330t | NVIDIA K80 | NVIDIA K80 |
|---|---|---|---|
| **Model** | Binomial CRR | Sampled Multinomial | Sampled Multinomial Antithetic |
| **Performance** ($Steps/s$) | $2.943 \times 10^{10}$ | $1.38 \times 10^{11}$ | $1.61 \times 10^{11}$ |
| **Time Steps Required** ($Steps$) | $4.89 \times 10^{10}$ | $1.9474 \times 10^{10}$ | $1.9474 \times 10^{10}$ |
| **Power** ($W$) | 29 | 300 | 300 |
| **Performance / Power** ($Steps/s/W$) | $1.01 \times 10^{9}$ | $4.60 \times 10^{8}$ | $5.36 \times 10^{8}$ |
| **Time Steps Required / Performance** ($s$) | 1.662 | 0.1411 | 0.1209 |
| **Execution Time $\times$ Power** ($Ws$) | 48.198 | 42.33 | 36.27 |

Table 9.7: Tesla K80 sampled multinomial against Fabry's FPGA binomial model

From Table 9.7 it is seen that for RMSE $10^{-3}$ the Tesla K80 using an antithetic sampled multinomial model uses 24.7% less energy than Fabry's binomial method. It has $\approx 2\times$ less step throughput per watt however due to the multinomial models better convergence properties the execution energy is less. It is also noted that this is likely a conservative figure as the antithetic sampled multinomial considered the same iteration space when in actuality the RMSE would have been reduced by the antithetic variate technique.

### 9.8.4 Requirements Capture Revisited

In Chapter 4, questions were posed as a means of measuring the success of the project. These are revisited here:

1. How do the convergence properties of discrete space models differ from the Gaussian continuous space model?

   The continuous models have a better convergence complexity of $\mathcal{O}(\epsilon^{-\frac{5}{2}})$ opposed to the discrete models' $\mathcal{O}(\epsilon^{-3})$. The sampled multinomial model is a discrete model that is however a

close enough approximation of the Gaussian model that it has indistinguishable convergence properties for the *NM Space* considered in this project.

2. How should the algorithms be parallelised and split across compute units?

   The exact changes that were made for each model are outlined in detail in Chapter 8.

3. What advantages of Fabry's discrete time and discrete space model on an FPGA are directly applicable to GPUs?

   The use of the Cox-Ross-Rubinstein model on the GPU led to a 2.37× greater throughput than the Gaussian model. The FPGA saw a 13× increase in throughput over the Gaussian model. Hence the model has directly applicable benefits on the GPU, however they are markedly less pronounced.

4. What algorithmic changes should be made when targeting a GPU?

   There are minimal gains from placing the entire potential stock movement space on the GPU. Consequently the use of alternative multiplicative models, with better convergence properties become an option. The exploration of the optimisations that can be made for each alternative model is outlined in full in Chapter 8.

5. What improvements in performance and accuracy can be obtained using a non-recombinant models on a GPU?

   It was found that increases in throughput can be achieved through changing to the Jarrow-Rudd binomial model at the cost of convergence. It was found for this trade off, the execution time was $\approx 3\times$ slower than the Cox-Ross-Rubinstein model at pricing with an accuracy of $2 \times 10^{-3}$. Improvements in the convergence properties were able to be introduced through the use of multinomial trees at the cost of throughput. The sampled multinomial trees had the best convergence properties, matching the Gaussian, and a 3.14× greater throughput for tree with sixty-four branches.

6. What improvements in performance and accuracy can be obtained using antithetic sampling, control variates, multi-level on a GPU?

   (a) The antithetic method resulted in greater than or equal throughput and a reduction in variance, hence it was always beneficial to use it over a regular random number generation.

   (b) The European control variate method had reductions in variance however the additional computation means its performance was worse than without it for larger confidence interval sizes. The geometric control variate saw large variance reductions with magnitude as great as $10^2$, this was balanced with reductions in throughput of between 25-50% depending on the base implementation.

   (c) Multi-level was not implemented successfully

7. Of the methods examined which has the best execution properties for a GPU and how does this compare to Fabry's FPGA approach?

   Of the non-control variate methods considered, using the Cox-Ross-Rubinstein model is not a worthwhile endeavour on the the GPU as its power consumption is 3.38× greater than the FPGA implementation's. Using the antithetic sampled multinomial results in 24.7% less energy required to compute a simulation with $RMSE = 10^{-3}$ compared to the FPGA binomial model. The introduction of a geometric control variate with antithetic variates results in a further speed up of 18.82× over the regular method offering a 25× more efficient method than the binomial FPGA solution.

# 10 Further Work

A multi-level implementation was established as a feasible approach at the beginning of this report, however due to unexpected convergence issues and time limitations, a full working implementation was not completed. The key advantage of the method is the removal of the need to calculate the *NM Space* of the model; as the multi-level method dynamically determines the bias and confidence intervals of the option. Given the strong results presented in this paper, based on the geometric control variate method, it would be an propitious proposition to investigate if it is possible to combine the multi-level and control variate methods such that the geometric control variate makes use of Gaussian bridges to generate a stochastically associated multi-level geometric walk. Equation 10.1 shows the assumed form of such a method.

$$x_c = x + c(\frac{1}{N_0}\sum_{n=1}^{N_0} y_0^{(n)} + \sum_{l=1}^{L}[\frac{1}{N_l}\sum_{n=1}^{N_l}(y_l^{(l,n)} - y_{l-1}^{(l,n)})] - E(y)) \tag{10.1}$$

It would be of significant interest to determine whether or not a discrete multi-level method can be formulated. This would require a method to generate stochastically linked samples for different levels of model accuracy. Attempts were made to create a discrete multi-level method using larger multinomial tree sizes as the higher levels of accuracy. The basis of this idea was rather than the level accuracy growing through expansion in time, as it does in the Gaussian model, it would instead grow in discrete-space. The initial findings for this approach were inconclusive due to mixed correlations between different multinomial tree sizes using the same uniform random sequence. An alternative approach to this method would be formulating a discrete-space equivalent to the Gaussian bridge to generate intermediary steps in a walk.

The project provided a utilitarian framework for building and testing the Monte Carlo simulations. Improvements in the functionality of the framework can and should be made. An addition that would prove advantageous is executables being run on the basis of error size as opposed to having to provide a specified *NM Space*. This would require the calculation of the required *NM Space* inside the framework, however this could be entirely circumnavigated if a multi-level implementation was successfully implemented. Originally the framework had the functionality to calculate a call, put, or both prices in the most effective way possible, but this hampered the ability to prototype models due to the additional complexity; hence it was was discontinued. The reintroduction of this functionality would improve the value of the framework as a tool to price options. The fundamental concept behind the framework was providing a basis to compare models consistently, and rapidly prototype methods. It would likely be beneficial to build a new framework that is heavily optimised for performance as opposed to ease of development. Such a framework would allow for the introduction of handling live input processing, handling multiple OpenCL devices, and introducing a method to interrupt simulations in progress rapidly.

In the project the arithmetic average of a path was calculated using a simplistic Euler approximation, the average of a discrete segment is equal to the mean of the two end points. The Milstein scheme is a better approximation of the numerical solution to a stochastic differential equation [32], as such it is of interest to determine if its convergence benefits outweigh the additional computational requirements

of implementing it.

The effect of combining antithetic and control variate techniques was not analysed in this project. Antithetic variates were applied with control variates, however they were used as a method to improve the throughput of the simulation. The alterations to variance were disregarded. It needs to be investigated further through RMSE convergence analysis to determine if the *NM Space* can be further optimised and squeeze further performance gains.

The principal idea supporting the project was investigating whether the benefits of an FPGA optimised binomial Monte Carlo simulation were applicable when considering a GPU's architecture. The optimisations made resulted in a sampled multinomial tree being recognised as the optimal solution for the scope of the project. The question is now posed whether this model could be optimised for an FPGA's architecture?

# 11 Conclusion

This project considers whether discrete-space algorithms devised for FPGAs as an alternative to Gaussian Monte Carlo pricing algorithms are applicable for GPU architectures. It was found that the *NM Space* of the binomial models converged at a slower rate compared to the Gaussian model. This difference meant that for large accuracies, the performance characteristics were similar due to the increased throughput of the binomial model; however as the *NM Space* grew, the Gaussian model became comparatively faster. It was then determined that expanding the number of branches of the discrete model improved its convergence properties. A multinomial tree based on sampling an inverse normal cumulative distribution resulted in like convergence properties to the Gaussian model but with a 3.14× increase in throughput for a branch size of sixty-four.

It was found that this sampled multinomial's performance could be enhanced through the introduction of variance reduction methods. Antithetic variates reduced the variance of the results whilst simultaneously improving the throughput by halving the number of random numbers needed to be generated. This resulted in a 1.16× increase in throughput over the regular random number generation.

The use of geometric control variates vastly reduced the variance of the simulation, this came at the cost of throughput. The trade off resulted in an implementation that was 25× more power efficient than the original discrete space model proposed for the FPGA. Hence conclusively answering the question that discrete space models are applicable for GPUs.

# 12 User Guide

The source files for this project can be found at `https://github.com/nar213/fyp-nar213`

## 12.1 Binaries

All of the relevant binary executables can be built by typing 'make all' within the subdirectory Workspace/Testsuite.

The environment variables MONTE_CARLO_DEVICE and MONTE_CARLO_PLATFORM are used to select the OpenCL device and platform being used respectively. Information regarding which platform and device has been selected is displayed via standard error channel.

**specificSimulation** is a simplistic program that runs an specific simulation. It uses command line arguments to determine the parameters of the execution. It takes these arguments:

1. Stock Price: The initial price of the underlying stock
2. Strike Price: The strike price of the option
3. Interest Rate: The interest rate at which the stock grows and will be discounted against
4. Final Time: Time to expiration
5. Volatility: The volatility of the underlying stock
6. Steps: The number of steps the final time is broken into
7. Iterations: The number of iterations execute by the simulation
8. Random Seed: Boolean determining whether to start simulation from a random point or to start from a static seed
9. Statistics: Boolean determining whether additional outputs are calculated such as skewness, and kurtosis
10. Antithetic: Boolean determining whether or not antithetic random number generation is used
11. Engine Name : A string that determines which engine to run

eg.

```
./bin/specificSimulation 100 105 0.1 1 0.25 1000 1000000 1 0 1 BinomialMultiplication
```

**advancedSimulation** is similar to *specificSimulation* however is read in the input from a CSV file and is more capable of running large numbers of simulations. The primary use case of this file is for large computations. The CSV must have these arguments on each line:

1. Stock Price: The initial price of the underlying stock
2. Strike Price: The strike price of the option
3. Interest Rate: The interest rate at which the stock grows and will be discounted against
4. Final Time: Time to expiration
5. Volatility: The volatility of the underlying stock
6. Steps: The number of steps the final time is broken into
7. Iterations: The number of iterations execute by the simulation
8. Random Seed: Boolean determining whether to start simulation from a random point or to start from a static seed
9. Statistics: Boolean determining whether additional outputs are calculated such as skewness, and kurtosis
10. Antithetic: Boolean determining whether or not antithetic random number generation is used
11. Engine Name : A string that determines which engine to run
12. Repetitions: The number of times to run this specific simulation

eg.

```
./bin/advancedSimulation inputs.csv > output.csv
```

where inputs.csv contains

```
100,115,0.92,1,0.25,1000,5000000,1,0,0,GaussianControlVariatesGeometric,42
100,115,0.92,1,0.25,1000,5000000,1,0,0,JarrowRuddControlVariatesGeometric,42
```

**batchComparison** provides a basic level of confidence that the build was successful by running all of the GPU based engines to see that they are producing *correct* results. This a good first point of call to determine whether there are any issues with any of the engines.

eg.

```
./bin/batchComparison
```

## 12.2 Creating a new engine

In order to create a new engine a simplistic skeleton has been provided below to shows the fundamental components that are required to add a new simulation engine to the register. Note that the makefile is not entirely automated and the CPP file needs to be added manually.

**/include/EngineName.hpp**

```
#ifndef ENGINE_NAME_HEADER
#define ENGINE_NAME_HEADER

#include "GPUContext.hpp"
#include "optionPricer.hpp"
#include "pricerRegistrar.hpp"

class EngineName : public OptionPricer {
  static DerivedRegister<EngineName> reg;

public:
  EngineName() {
    gpu = GPUContext("src/GPU/EngineName.cl");
    pricerName = "EngineName";
  }
  bool execute(int iterations);

protected:
};

#endif // ENGINE_NAME_HEADER
```

### /src/EngineName.cpp

```
#include "GPU/EngineName.hpp"

DerivedRegister<EngineName>
    EngineName::reg("EngineName");

bool EngineName::execute(int iterations) {
  callPrice = 42;
  return true;
}
```

# 13 Appendix

## 13.1 Guide to Results stored on Github

Due to the size of the results collected only the K560 results have been included in the appendices as they are reference in the Evaulation section. All of the data used to generate the figures and draw conclusions from in this paper can be found at `https://github.com/nar213/fyp-nar213`. The output files can be found in the subdirectory `Workspace/TestSuite/outputs`. These files were generated using the advancedSimulation script and the files found in `Workspace/TestSuite/inputs`. Below is a summary of the subdirectory contents found in the outputs.

- Antithetics: Comparison of the effects of interest rates, strike price, and volatilites for Cox-Ross-Rubinstein, Jarrow-Rudd, Gaussian, and sampled multinomial files
- ConvergenceProperties: Data used to generate *NMSpace* models for a variety of models.
- CorrelationsCV: Data used to generate *NMSpace* models for a variety of control variate based models.
- ExecutionConfidence: Using a python script found in `Workspace/TestSuite/scripts` the optimal *NMSpace* ratios were generated to generate results providing appropriate confidence level interval size against execution time
- FloatingDouble: Comparison of the output values for the floating point and double implementations
- GivenAccuracy: *NM Space* provided for each model to achieve the given the confidence level.
- LargeConvergenceProperties: Data used to generate *NMSpace* models for a variety of models. Similar to ConvergenceProperties however it has a finer difference between the discretisations of the space. This data was used to generate an alternative *NM Space* model however it was not included in the report due to the inconsistency of model.
- Model Validation: The parameters used to test against the *MATLAB* Asian option pricer and the results found in the paper.
- SmallConvergenceProperties: Data used to generate *NMSpace* models for a variety of models, courser graph, quicker to generate and hence easier to generate results allowing for faster generation of approximate relationships earlier on in the project.
- SmallConvergencePropertiesCV: Data used to generate *NMSpace* models for a variety of control variate based models, courser graph, quicker to generate and hence easier to generate results allowing for faster generation of approximate relationships earlier on in the project.
- StepsThroughput: The values used to generate the time step throughputs
- TrueValueApproach: Basis of the absolute error analysis for very large iteration spaces for a variety of models.

- TrueValueApproachCV: Basis of the absolute error analysis for very large iteration spaces for a variety of control variate models.
- TrueValues: Several true value estimates to check that for alternative option prices the models held
- WalksPerKernel: Data used to determine the diference in execution time for the binomial memory and multiplication models using different number of walks per kernel.

## 13.2 NVIDIA K560 Results

| | **Regular** | | **Antithetic** | | |
|---|---|---|---|---|---|
| **Name** | **Throughput** ($G.Steps/s$) | **Base Speed Up** | **Throughput** ($GSteps/s$) | **Antithetic Speed Up** | **Base Speed Up** |
| Binomial JR | 85.43 | 5.07 | 98.93 | 1.16 | 5.87 |
| Trinomial JR | 84.24 | 5.00 | 98.08 | 1.16 | 5.82 |
| Sampled Normal | 68.01 | 4.03 | 90.53 | 1.33 | 5.37 |
| Trinomial CRR | 64.85 | 3.85 | 93.77 | 1.45 | 5.56 |
| Binomial CRR | 64.16 | 3.80 | 108.59 | 1.69 | 6.44 |
| Nonanomial JR | 59.13 | 3.51 | 63.94 | 1.08 | 3.79 |
| Binomial JR Geometric CV | 58.86 | 3.49 | 67.93 | 1.15 | 4.03 |
| Binomial CRR Memory Local | 52.94 | 3.14 | 65.65 | 1.24 | 3.89 |
| Binomial CRR Geometric CV | 45.64 | 2.71 | 59.37 | 1.30 | 3.52 |
| Sampled Normal Geometric CV | 41.04 | 2.43 | 57.52 | 1.40 | 3.41 |
| Gaussian | 16.86 | 1.00 | 29.43 | 1.75 | 1.75 |
| Gaussian European CV | 16.35 | 0.97 | 28.72 | 1.76 | 1.70 |
| Binomial JR Double | 15.91 | 0.94 | 16.04 | 1.01 | 0.95 |
| Gaussian Geometric CV | 15.82 | 0.94 | 24.71 | 1.56 | 1.47 |
| Binomial CRR Double | 14.08 | 0.84 | 15.18 | 1.08 | 0.90 |
| Octanomial CRR | 10.37 | 0.61 | 10.50 | 1.01 | 0.62 |
| Binomial CRR Memory Global | 6.68 | 0.40 | 6.66 | 1.00 | 0.39 |
| Gaussian Double | 0.65 | 0.04 | 0.90 | 1.38 | 0.05 |

Table 13.1: NVIDIA Grid K560 Throughputs

| | **Regular** | | **Antithetic** | | |
|---|---|---|---|---|---|
| **Name** | **Execution Time for $\epsilon = 2 \times 10^{-3}$** | **Base Speed Up** | **Execution Time for $\epsilon = 2 \times 10^{-3}$** | **Antithetic Speed Up** | **Base Speed Up** |
| Gaussian Geometric CV | 0.012 | 46.614 | 0.011 | 1.046 | 48.744 |
| Sampled Normal Geometric CV | 0.013 | 41.699 | 0.012 | 1.104 | 46.057 |
| Binomial JR Geometric CV | 0.017 | 33.765 | 0.015 | 1.104 | 37.282 |
| Binomial CRR Geometric CV | 0.018 | 31.613 | 0.016 | 1.098 | 34.711 |
| Gaussian European CV | 0.226 | 2.468 | 0.182 | 1.248 | 3.079 |
| Gaussian | 0.559 | 1.000 | 0.255 | 2.191 | 2.191 |
| Trinomial CRR | 0.628 | 0.889 | 0.295 | 2.131 | 1.896 |
| Binomial CRR | 0.841 | 0.665 | 0.350 | 2.401 | 1.596 |
| Sampled Normal | 0.856 | 0.653 | 0.432 | 1.981 | 1.294 |
| Nonanomial JR | 1.530 | 0.365 | 0.659 | 2.322 | 0.848 |
| Binomial CRR Memory Local | 1.673 | 0.334 | 0.679 | 2.463 | 0.823 |
| Trinomial JR | 2.317 | 0.241 | 1.105 | 2.096 | 0.506 |
| Octanomial CRR | 3.152 | 0.177 | 1.496 | 2.107 | 0.374 |
| Binomial CRR Double | 3.252 | 0.172 | 1.758 | 1.850 | 0.318 |
| Binomial JR | 3.799 | 0.147 | 1.852 | 2.051 | 0.302 |
| Binomial CRR Memory Global | 5.751 | 0.097 | 3.250 | 1.769 | 0.172 |
| Gaussian Double | 9.865 | 0.057 | 4.277 | 2.306 | 0.131 |
| Binomial JR Double | 17.278 | 0.032 | 9.572 | 1.805 | 0.058 |

Table 13.2: NVIDIA Grid K560 Execution Times

| | **Regular** | | **Antithetic** | | |
|---|---|---|---|---|---|
| **Name** | **Execution Time for $\epsilon = 1 \times 10^{-4}$** | **Base Speed Up** | **Execution Time for $\epsilon = 1 \times 10^{-4}$** | **Antithetic Speed Up** | **Base Speed Up** |
| Sampled Normal Geometric CV | 1.082 | 1.673 | 0.994 | 1.088 | 1.821 |
| Gaussian Geometric CV | 1.810 | 1.000 | 1.447 | 1.251 | 1.251 |
| Binomial JR Geometric CV | 19.949 | 0.091 | 18.929 | 1.054 | 0.096 |
| Binomial CRR Geometric CV | 42.351 | 0.043 | 33.118 | 1.279 | 0.055 |

Table 13.3: NVIDIA Grid K560 Execution Times for Finer Accuracies

# Bibliography


[1] Altera. Monte carlo pricing of asian options on fpgas using opencl, 2013.

[2] HT Anson, David B Thomas, Kuen Hung Tsoi, and Wayne Luk. Reconfigurable control variate monte-carlo designs for pricing exotic options. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 364–367. IEEE, 2010.

[3] Brahim Betkaoui, David B Thomas, and Wayne Luk. Comparing performance and energy efficiency of fpgas and gpus for high productivity computing. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 94–101. IEEE, 2010.

[4] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–54, 1973.

[5] George EP Box, Mervin E Muller, et al. A note on the generation of random normal deviates. *The annals of mathematical statistics*, 29(2):610–611, 1958.

[6] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.

[7] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263, 1979.

[8] Ed Easterling. Volatility in perspective. *Crestmont Research*, 2017.

[9] Pieter Fabry. Option-pricing using monte-carlo over lattices. Master's thesis, Imperial College London, 2016.

[10] Michael B Giles. Improved multilevel monte carlo convergence using the milstein scheme. In *Monte Carlo and quasi-Monte Carlo methods 2006*, pages 343–358. Springer, 2008.

[11] Michael B Giles. Multilevel monte carlo path simulation. *Operations Research*, 56(3):607–617, 2008.

[12] Michael B Giles. Multilevel monte carlo methods. Online, 2012.

[13] Michael B Giles. Multilevel monte carlo methods. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 83–103. Springer, 2013.

[14] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2013.

[15] S&P Global. S&p 500, 2017.

[16] JM Hammersley and KW Morton. A new monte carlo technique: antithetic variates. In *Mathematical proceedings of the Cambridge philosophical society*, volume 52, pages 449–475. Cambridge Univ Press, 1956.

[17] Robert L Harrison, Carlos Granja, and Claude Leroy. Introduction to monte carlo simulation. In *AIP conference proceedings*, volume 1204, pages 17–21. AIP, 2010.

[18] Stefan Heinrich. Multilevel monte carlo methods. In *International Conference on Large-Scale Scientific Computing*, pages 58–67. Springer, 2001.

[19] John C. Hull. *Options, Futures, and Other Derivatives*. Pearson, Prentice Hall, Upper Saddle River (N.J.), 2006.

[20] Kiyosi Itô. Stochastic differential equations in a differentiable manifold. *Nagoya Math. J.*, 1:35–47, 1950.

[21] Robert Jarrow and Andrew Rudd. Approximate option valuation for arbitrary stochastic processes. *Journal of financial Economics*, 10(3):347–369, 1982.

[22] David H Jones, Adam Powell, Christos-Savvas Bouganis, and Peter YK Cheung. Gpu versus fpga for high productivity computing. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 119–124. IEEE, 2010.

[23] David R Kaeli, Perhaad Mistry, Dana Schaa, and Dong Ping Zhang. *Heterogeneous Computing with OpenCL 2.0*. Morgan Kaufmann, 2015.

[24] Angelien GZ Kemna and ACF Vorst. A pricing method for options based on average asset values. *Journal of Banking & Finance*, 14(1):113–129, 1990.

[25] MathWorks United Kingdom. Pricing asian options, 2017.

[26] N Kubendran, Laxmi Sowmya Rachiraju, Aswini Kumar Mishra, and Aruna Bommareddy. Quasi-monte carlo approach to asian options pricing. *Asia-Pacific Journal of Management Research and Innovation*, 10(1):67–78, 2014.

[27] Pierre L'Ecuyer and Richard Simard. Testu01: A c library for empirical testing of random number generators. *ACM Trans. Math. Softw.*, 33(4):22:1–22:40, August 2007.

[28] Shuo Li and James Lin. Many-core programming with asian option pricing. In *High Performance Computational Finance (WHPCF), 2014 Seventh Workshop on*, pages 45–52. IEEE, 2014.

[29] George Marsaglia et al. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003.

[30] George Marsaglia, Wai Wan Tsang, et al. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 2002.

[31] George Marsaglia and Arif Zaman. A new class of random number generators. *The Annals of Applied Probability*, pages 462–480, 1991.

[32] GN Mil'shtejn. Approximate integration of stochastic differential equations. *Theory of Probability & Its Applications*, 19(3):557–562, 1975.

[33] Umar Ibrahim Minhas, Samuel Bayliss, and George A Constantinides. Gpu vs fpga: A comparative analysis for non-standard precision. In *International Symposium on Applied Reconfigurable Computing*, pages 298–305. Springer, 2014.

[34] Harald Niederreiter. *Random Number Generation and quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[35] James Primbs, Muruhan Rathinam, and Yuji Yamada. Option pricing with a pentanomial lattice model that incorporates skewness and kurtosis. *Applied Mathematical Finance*, 14(1):1–17, 2007.

[36] K Sigman. Introduction to reducing variance in monte-carlo simulations, 2007.

[37] Trefis Team. What led to the surge in nvidia's stock price this year; why we believe the market over-reacted, Jul 2016.

[38] David B Thomas. The mwc64x random number generator, 2011.

[39] David B Thomas, Wayne Luk, Philip HW Leong, and John D Villasenor. Gaussian random number generators. *ACM Computing Surveys (CSUR)*, 39(4):11, 2007.

[40] Anson HT Tse, David B Thomas, Kuen Hung Tsoi, and Wayne Luk. Efficient reconfigurable design for pricing asian options. *ACM SIGARCH Computer Architecture News*, 38(4):14–20, 2010.

[41] Paul Wilmott. *Paul Wilmott on Quantitative Finance*. Wiley, Hoboken, N.J, 2nd edition, 2014.

[42] Yuji Yamada and James A. Primbs. *Construction of Multinomial Lattice Random Walks for Optimal Hedges*, pages 579–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.