Application of Multi Layer Monte Carlo in Solving Partial Differential Equations for Bond Option Pricing

by

Eric Li

Supervisor: Professor Chi-Guhn Lee
April 2022

Application of Multi Layer Monte Carlo in Solving Partial Differential Equations for Bond Option Pricing

Eric Li

Abstract

Presently, Partial Differential Equations (PDEs) are prevalent in modelling quantitative financial derivatives. However, the Curse of Dimensionality (CoD) makes modelling these PDEs in high dimensions difficult due to requiring more training data and training time. Deep neural networks have shown to be effective in mitigating the CoD in options pricing, but are limited by time inefficiencies with underlying Monte Carlo (MC) path sampling for data generation and modelled PDE simplicity. The goal of this thesis is to increase the time efficiency of path simulation and to increase the complexity of PDE modelled with stochastic interest rates, which have not been done in present literature. This is approached by applying a path sampling method previously only used for numerical PDE estimation- Multilevel Monte Carlo (MLMC)- in place of MC in deep neural network data generation for option pricing with the Black Scholes Barenblatt equation and modelling stochastic interest rates with zero coupon bonds. Key results include MLMC requiring 1.4-1.6 and 5.1-7.2 times less training time and time steps to reach various specified validation relative errors respectively. No results for pricing zero coupon bonds with deep neural networks have been provided at this time. The results show initial promise for MLMC replacing previous MC methods for increased time efficiency in deep neural network data generation, with further investigation needed for experiments with modelling zero coupon bonds and more complex PDEs in bond options and swaptions, and further improving MLMC's time efficiency through methods such as adaptive sampling.

Acknowledgments

I would like to express my sincere thanks to my supervisor Prof. Chi-Guhn Lee for receiving me under his research group and advising through key components of this thesis. I would also like to express my gratitude for Raj Patel and Amine Aboussalah for framing this project, offering supporting background material, and providing guidance over the course of this thesis.

Contents

1 Introduction **1**

2 Literature Review **3**

2.1 Modelling PDEs and SDEs with Deep Learning 3

2.2 Time Efficiency 6

2.3 Stochastic Interest Rates 8

3 Methods **10**

3.1 Comparing MLMC and MC 10

3.2 Optimizing MLMC Hyperparameters 12

3.3 Stochastic Interest Rates: Bond Options 13

4 Results and Discussion **15**

4.1 Comparing MLMC and MC Results 15

4.2 MLMC Hyperparameter Optimization 18

5 Conclusion **21**

Bibliography **23**

A Other MLMC and MC Comparisons **25**

B MLMC Run Graphs **28**

C MLMC Hyperparameter Exploration **30**

List of Tables

A.1 The Seconds per Time Step Required by the Best Model of MLMC and MC for Each Respective Relative Error 26

List of Figures

2.1 MLMC path simulation for $l = 1, 2, 3$, $M = 2$, and $T = 1$ 7

3.1 The prediction of the option price Y_t and calculation of loss at each time step t in one path realization 11

4.1 The total time in seconds required for MLMC and MC to reach a certain validation relative error on a log scale. 16

4.2 The total number of time steps required for MLMC and MC to reach a certain validation relative error on a log scale. 17

4.3 The log normalized of the total number of time steps required for an H Factor and N_l MLMC hyperparameter combination to reach 0.02 validation relative error. 18

A.1 The seconds per time step required for the best models of both MLMC and MC at each respective validation relative error. 25

A.2 The factor of improvement for the total training time the best model of MLMC displays over the best model of MC across varying validation relative error tested. 26

A.3 The factor of improvement for the number of time steps the best model of MLMC displays over the best model of MC across varying validation relative error tested. 27

B.1 5 sample paths generated from the MLMC deep neural network predicting option prices between initial to terminal time compared to the option prices obtained from the explicit solution with a validation relative error of 0.02. 28

B.2 The mean and two standard deviations of a MLMC deep neural network across a batch of 100 validation sampling paths. The deep neural network's is trained when a validation relative error of 0.02 is reached. 29

C.1 The log normalized total time steps required for MLMC with various h values to reach respective validation relative errors. 30
C.2 The log normalized total time required for MLMC with various N_l values to reach respective validation relative errors. 31

Chapter 1

Introduction

Within mathematics, Partial Differential Equations (PDEs) are regularly used to describe real world phenomenons through modelling the relationship of underlying variables and functions. Despite the prevalent usage of PDEs, the "Curse of Dimensionality" (CoD) makes developing methods to represent and to solve such PDEs difficult in high dimensions due to the increasing data sparsity- a consequence of increasing the number of dimensions- and the resulting exponential growth of data quantity needed to compensate such phenomena [4]. Given that there is a diverse range of PDE types and classes, the CoD is prevalent in many applications, one of which being quantitative finance where option pricing is modelled over time with high dimensional PDEs.

Currently, deep learning has been shown to mitigate the CoD problem when modelling high dimensional PDEs through various experiments in quantitative finance despite no theoretical backing [5]. A common method to apply deep learning in modelling PDEs is to reformulate the PDEs as a system of equivalent Backward Stochastic Differential Equations (BSDEs) through using the Feynman-Kac Formalism and Ito's formula [8, 18]. Then, the equivalent BSDEs can be discretized into a sequence of time steps using the Euler-Maruyama scheme [13], which is a commonly used method for discretizing basic Stochastic Differential Equations (SDEs) [12]. After discretization, Monte Carlo sampling methods are used for path simulation to generate training, validation, and testing data representing underlying BSDEs.

Through this method, European option prices and other simplistic European financial derivatives have been modelled through deep learning application on the Black-Scholes-Barenblatt (BSB) PDE equation in high dimensions [18, 8, 13, 5]. Despite promising initial results shown for upwards to 100 dimensions [13], present methods are largely limited from realistic application due to the time complexity associated with iterative Monte Carlo sampling method application and the simplicity of the

underlying financial derivative PDE modelled. Consequentially, there are two objectives of my thesis: to increase the time efficiency of path sampling for data generation when applying deep neural networks to model high dimensional BSB PDEs and to extend the complexity of the modelled financial derivative PDEs to include stochastic interest rates, which better reflect real world applications.

Chapter 2

Literature Review

The purpose of this literature review is to perform a deeper dive into the current common methods to reformulate PDEs into equivalent SDE expressions for deep neural network application. Experimental results from this formulation applied on PDEs in quantitative finance will further be highlighted along with associated advantages and disadvantages. Additional exploration will be performed on possible solutions for the aforementioned observed disadvantages with a focus on decreasing path sampling time complexity and increasing the complexity of the modelled PDE.

2.1 Modelling PDEs and SDEs with Deep Learning

The formulation of PDEs into SDEs and the subsequent modelling with deep neural networks in this thesis is based on a paper by Raissi. The PDE formulation begins with the general quasi-linear PDE:

$$u_t = f(t, x, u, Du, D^2u) \tag{2.1}$$

with $u(t, x)$ as the solution with terminal condition $u(T, x) = g(x)$. Through the application of Ito's formula, which is used to calculate the differentials of stochastic functions [15], $u(t, x)$ and its derivative $D(t, x)$ are related to a pair of stochastic processes in a forward backward stochastic differential equation (FBSDE) system:

$$\begin{aligned} dX_t &= \mu(t, X_t, Y_t, Z_t)dt + \sigma(t, X_t, Y_t)dW_t, \quad t \in [0, T], \\ X_0 &= \xi, \\ dY_t &= \varphi(t, X_t, Y_t, Z_t)dt + Z_t'\sigma(t, X_t, Y_t)dW_t, \quad t \in [0, T), \\ Y_t &= g(X_T) \end{aligned} \tag{2.2}$$

through setting

$$\begin{aligned} u(t, X_t) &= Y_t, \\ Du(t, X_t) &= Z_t. \end{aligned} \tag{2.3}$$

To apply deep learning on the related PDE and FBSDE system, $u(t, x)$ and $Du(t, x)$ are modelled with a deep learning neural network and automatic differentiation on $u(t, x)$ respectively. In this case, automatic differentiation is favored over other manual, numerical, and symbolic differentiation methods due to its use of chain rule in computing and propagating partial derivatives, which results in higher time efficiency and accuracy in approximation [1].

To model $u(t, x)$ and $Du(t, x)$ with a neural network, the Euler-Maryuama scheme is used to discretize FBSDE 2.2. An approximation for X^n and Y^n at each time step t^n for $n = 0, 1, ..., N-1$ is thus

$$\begin{aligned} X^{n+1} &\approx X^n + \mu(t^n, X^n, Y^n, Z^n)\Delta t^n + \sigma(t^n, X^n, Y^n)\Delta W^n, \\ Y^{n+1} &\approx Y^n + \varphi(t^n, X^n, Y^n, Z^n)\Delta t^n + (Z^n)'\sigma(t^n, X^n, Y^n)\Delta W^n \end{aligned} \tag{2.4}$$

where $\Delta t^n = T/N$ is the size of a single time step and ΔW^n is the incremental change of Brownian Motion driven by a normal distribution $\mathcal{N}(0, \Delta t^n)$. The initial value of X is set as $X^0 = \xi$. The neural network's loss function is given as

$$\sum_{m=1}^{M}\sum_{n=0}^{N} |Y_m^{n+1} - Y_m^n - \varphi(t^n, X_m^n, Y_m^n, Z_m^n)\Delta t^n - (Z_m^n)'\sigma(t^n, X_m^n, Y_m^n)\Delta W_m^n|^2 + \sum_{m=1}^{M} |Y_m^N - g(X_m^N)|^2 \tag{2.5}$$

where M is the batch size (number of stochastic path simulations in one iteration). In 2.5, the former term with a summation over each M and N can be inferred as the loss accumulated over time steps $n = 0, 1, ..., N-1$, while the latter summation term can be inferred as the estimation error between the approximated Y_m^N and the actual $g(X_m^M)$ stochastic process Y values at terminal time N.

This generalized method of formulating a PDE 2.1 into a system of FBSDEs 2.2 and modeling with a deep neural network with loss 2.5 is applied to quantitative finance by using the BSB equation for European Option pricing. This is done by setting u_t in 2.1 as

$$u_t = -\frac{1}{2}Tr[\sigma^2 diag(X_t^2)D^2 u] + r(u - (Du)'x) \tag{2.6}$$

with $u(T, x) = g(x)$ as the terminal condition, X_t as the underlying stock price at

time t, and u_t as the overlying option prices at time t. Through Ito's formula, 2.6 is related to the FBSDE system

$$\begin{aligned} dX_t &= \sigma diag(X_t) dW_t, \quad t \in [0, T], \\ X_0 &= \xi, \\ dY_t &= r(Y_t, Z_t' X_t) dt + \sigma Z_t' diag(X_t) dW_t, \quad t \in [0, T), \\ Y_t &= g(X_T) \end{aligned} \tag{2.7}$$

where $g(X) = \|x^2\|$, X_t is the underlying stock price at time t, constants scalars T, σ, and r represent the total time period, underlying volatility, and interest rate respectively, and constant vector ξ represents the initial underlying stock price at $t = 0$. MC path simulations can now be used to generate training, validation, and test data through approximation of Y_t, which are used during deep neural network training.

This approach by Raissi has several significant advantages over previous approaches, namely the requirement of only one deep neural network and one complete training cycle to model 2.1 and 2.2 at any given point in time. This is a significant advancement in comparison to the approach proposed by Han et al., which only models a subset of FBSDE cases from 2.2 and calls for the use of separate deep neural networks at each time step t_n [8]. Only being able to predict terminal value Y_0, Han et al.'s approach also requires retraining to predict Y_t values at intermediate time steps $0 < t < N$ [8]. Overall, Raissi approach reduces the number of parameters required to train the deep neural network and enables the modelling of stochastic process values at intermediate time steps.

Experimentally, Raissi's approach has shown success in 100 dimensions when applied to European option pricing. With the explicit solution of 2.6 as

$$u(t, x) = exp((r + \sigma^2)(T - t))g(x), \tag{2.8}$$

the relative error of the predicted option price is compared to the explicit option price at terminal time. By setting the total number of time steps between $[0, T]$ as $N = 50$ and the batch size as $M = 100$, Raissi applies a 5 layer deep neural network with 256 neurons in each hidden layer to achieve a relative error of about 2.3×10^{-3}, after approximately 500 steps of training [13]. This is in line with the results produced by Han et al. [13, 8].

2.2 Time Efficiency

While Raissi's approach depicts initial promising results and advances upon a previous approach by Han et al. by reducing the number of training parameters, there are several apparent limitations, one of which being the use of Monte Carlo (MC) sampling for path simulations. With additional sampling and neural network iterations required to obtain better Y_t at intermediate time steps [13], Raissi's approach is limited by MC sampling's slow convergence rate of $O(N^{-1/2})$ [10], deceleration over iterations [3], and sub optimal time complexity of $O(\epsilon^{-3})$ to achieve a user specified root mean squared error of ϵ [7]. For small ϵ values, MC sampling becomes costly and even small time efficiency improvements in sampling techniques can lead to substantial improvement across many iterations [3].

One sampling method improvement over MC sampling is Quasi-Monte Carlo (QMC) path sampling, which exhibits more uniform path sampling distributions in comparison to MC sampling. Due to this difference, QMC has a faster convergence rate of $O(N^{-1})$, which is the best possible convergence rate for equally weighted sampling techniques [10]. Despite usage for numerical option pricing estimations, QMC has not been largely applied in the application of data generation for training and validating machine learning models in high dimensions.

Another more recent sampling method improvement over MC path sampling is Multilevel Monte Carlo (MLMC) path sampling. Whereas MC and QMC use a fixed number of discretized time steps between initial and terminal times, MLMC uses various MC layers, which have varying number of discretized time steps, over the total number of path iterations. At initialization, MLMC starts with a MC layer of a hyperparameter M time steps. Then, MLMC computes the number of iterations N_1 to run at the current layer. Once N_1 iterations have been run, MLMC proceeds to increase the number of discretized time steps by a factor of M to achieve M^2 time steps. This process is repeated for a total of hyperperamater L layers. At general layer l, an optimal number of time steps N_l is calculated by

$$N_l = \left\lceil 2\epsilon^{-2}\sqrt{V_l h_l}\left(\sum_{l=0}^{L}\sqrt{V_l/h_l}\right)\right\rceil \tag{2.9}$$

where V_l is the variance at level l estimated with initial 10^4 samples, h_l is the size of each discretized time step calculated by $h_l = T/M$, and L is the total number of MC layers in the MLMC instance. Early termination at a given layer may also occur if

the convergence condition

$$|\hat{Y}_L - M^{-1}\hat{Y}_L - 1| < \frac{1}{\sqrt{2}}(M^2 - 1)\epsilon \tag{2.10}$$

is satisfied [7]. An illustration of the algorithm with $M = 2$ and $T = 1$ is shown in 2.1.

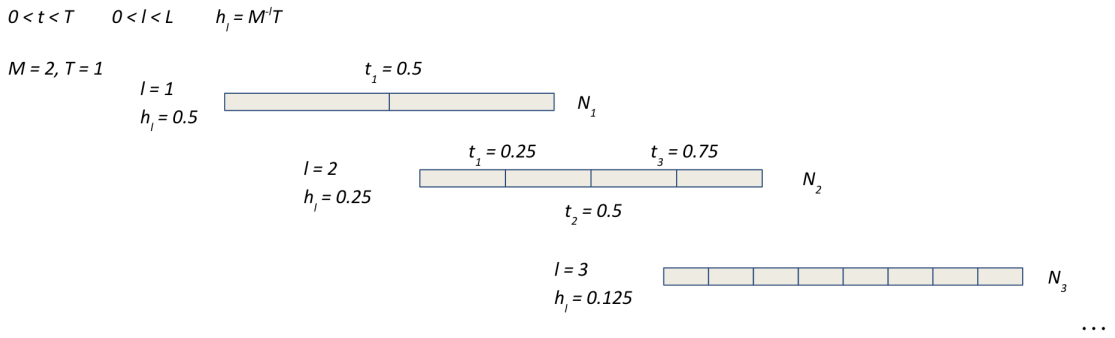

Figure 2.1: MLMC path simulation for $l = 1, 2, 3$, $M = 2$, and $T = 1$

Through the use of a variable number of time discretization at each layer, MLMC has a time complexity of $O(\epsilon^{-2}(log\epsilon)^2)$ to achieve an user specified root squared mean error of ϵ. This is an improvement over the $O(\epsilon^{-3})$ time complexity of MC to achieve the same ϵ. This time improvement by MLMC is reflected through experiments, which have shown a time decrease by a factor of 30-65 relative to MC when modelling European, Asian, Digital, and Lookback option prices. Furthermore, MLMC's time factor improvement is greater for smaller ϵ values [7]. Despite these initial promising results, MLMC is limited by low dimensions used during experimentation and simplicity of the modelled financial derivatives in which the time improvements are observed. MLMC also has not been applied for data generation in deep neural networks. Finally, MLMC's additional time improvement opportunities through combined usage with QMC and other sampling methods have not been observed.

In addition to the different sampling method improvements over the MC sampling method, other SDE discretization methods can be used to improve upon the Euler-Maruyama scheme. One such method is the Milstein discretization scheme, which has been prevalent in discretizing SDEs describing financial derivatives. The Milstein scheme is generally known to be more accurate in comparison to the Euler-Maruyama scheme due to increased expansion of mean and standard deviation coefficients through Ito's lemma [14]. Presently, the combination of path sampling improvements, such as with QMC or MLMC, and time discretization improvements, such as with the Milstein discretization scheme, has been unexplored and its potential

speed and accuracy improvements are unclear.

2.3 Stochastic Interest Rates

To realistically model the development of a financial derivative over time, the stochastic properties of short interest rates are considered. The Ho-Lee model is one of the first models used to model short term interest rates with the SDE

$$dr_t = \theta(t)dt + \sigma dW_t, \tag{2.11}$$

where σ is a constant, θ is some time dependent function, and W_t is Brownian motion [17]. Though simple, a limitation of the model is the exclusion of the mean reversion property, which is the property in which assets converge to an average price over time, commonly seen with interest rates [11]. To address this limitation, the One-Factor Hull White model builds on the Ho Lee model with

$$dr(t) = [\theta(t) - ar(t)]dt + \sigma dW_t, \tag{2.12}$$

where W_t is Brownian motion, $\theta(t)$ is some deterministic function of time, and a and σ are constants [2]. Modelling $r(t)$ as a normal distribution $\mathcal{N}(\frac{\theta}{a}, \frac{\sigma^2}{2a})$ when $t \to \infty$, the One-Factor Hull White SDE sees $r(t)$ approach the normal distribution faster with larger a mean reversion values [2]. Despite an improvement upon the Ho Lee model, this model is limited by the constant mean reversion factor a, which results in a poor representation of yield curves. This limitation is addressed by the Two-Factor Hull White model, which is often used to model short rate of interest development. The Two-Factor Hull White model introduces an additional stochastic process $u(t)$ to 2.12 to obtain SDE

$$\begin{aligned} dr(t) &= [\theta(t) + u(t) - ar(t)]dt + \sigma_1(t)dW_t^1 \\ du(t) &= -bu(t)\,dt + \sigma_2(t)dW_t^2, \end{aligned} \tag{2.13}$$

where $u(0) = 0$, a, b, σ_1, σ_2 are positive constants and W_t^2 and W_t^2 are two separate Brownian Motion processes with correlation ρ [2]. Currently, these developments of modelling stochastic interest rates have not largely been applied in deep learning approaches, which offer an area of exploration.

In Raissi's approach, a constant interest rate is used in the deep neural network model of the BSB equation. Due to the more accurate representation of financial derivatives through stochastic interest rates and the lack of machine learning to predict stochastic interest rates in present literature, the incorporation of stochastic

interest rate properties in 2.6 and 2.7 for modelling by a deep neural network is still to be explored.

Chapter 3

Methods

To meet the two objectives of this thesis- to increase the path sampling time efficiency when generating data for training a deep neural network in modelling high dimensional BSB PDEs and to extend the complexity of modelled quantitative finance PDEs to include stochastic interest rates- three separate methods are described in this section. The first is the method associated with comparing MLMC and MC time efficiency for neural network data generation. The second is the method for MLMC hyper parameter optimization. The third is the method of representing stochastic interest rates as a zero coupon bond and formulating the corresponding PDE into a SDE for neural network modelling.

3.1 Comparing MLMC and MC

Given the first goal to increase the time efficiency of path sampling, the time efficiency of MLMC is compared to that of MC. To run experiments with MLMC, MLMC is "swapped" in place of MC. This means that both mathematical foundation and neural network implementation is consistent between the two sampling methods.

In more descriptive terms, the same general PDE to SDE formulation via Ito's and subsequent SDE discretization for deep neural network training and data generation as described in 2.1 is used. The same BSB PDE 2.6 and BSB SDE 2.7 are used to represent the underlying option price as well. For the overarching deep neural network model, both MLMC and MC path sampling methods generate data for a 6 layer fully connected deep neural network with $D+1$ neurons in the input layer, where D is the number of dimensions, 256 neurons in each of the 4 hidden layers, and 1 neuron in the output layer. ReLU activation is used between each hidden layer. For training, $D = 10$ dimensions and $M = 100$ batch size are used, chosen respectively due to the high number of dimensions realistically applied in the quantitative finance industry

and usage in Raissi's approach [13]. With each batch representing 100 separate path realizations, N time steps are discretized between initial time $t = 0$ and terminal time $T = 1$ based on MLMC or MC sampling policy for each path realization in each iteration. At each intermediate time step t in path realization m, X_t is determined from $X_{t-1}+dX_t$ as per 2.7, with Brownian motion dW_t simulated by $(\sqrt{dt})\mathcal{N}(0,1)$ and initial underlying stock price is set as $X_0 = (0, 1, 0, 1, ...) \in \mathbb{R}^D$ as done in [13]. Scalar t and vector X_t are then concatenated to become a $D + 1$ vector and inputted into the deep neural network model to obtain Y_m^{n+1}, which can then be used to calculate loss across all path time steps and path realizations with loss equation 2.5 and 2.7 and Z_m^{n+1} with automatic differentiation. The loss is optimized through the Adam optimizer with a learning rate of 0.001. This training process in one time step for one given path realization is illustrated in 3.1.

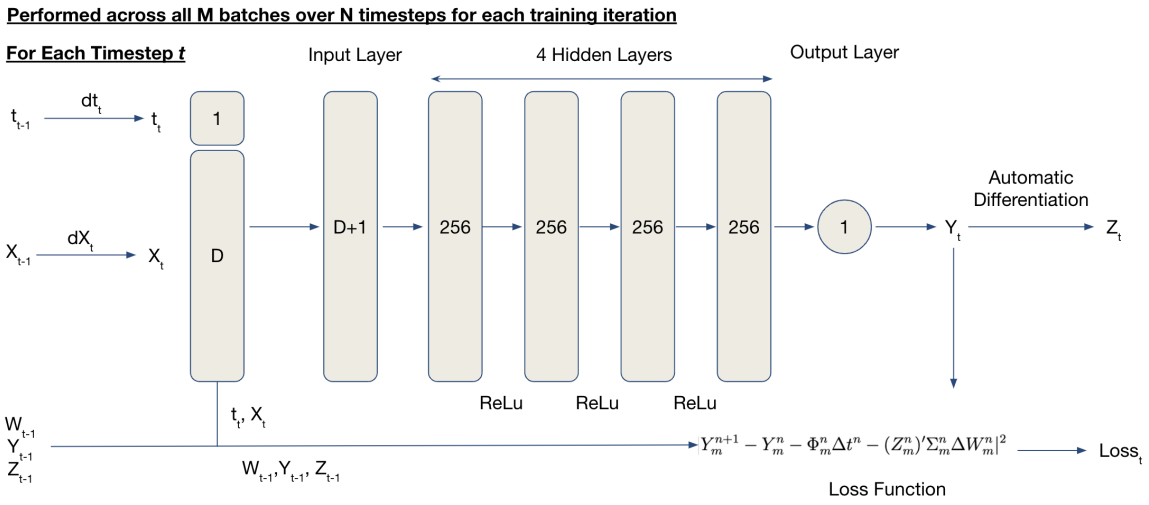

Figure 3.1: The prediction of the option price Y_t and calculation of loss at each time step t in one path realization

Between MLMC and MC, the only difference is the number of discretized time steps N. For MC, N is set as a constant user set hyperparameter maintained throughout training. For MLMC, N is a variable which increases at a factor of h for each subsequent MLMC layer. Whereas [7] used 2.9 to calculate a varying number of samples at each layer before layer progression, a constant N_l value is used instead and tuned as an user set hyperparameter for simplification. Specifically, N_l training iterations are ran before subsequent layer progression and number of time step increase by factor h.

At each training iteration, validation relative error ϵ is found by comparing the deep neural network predicted option price $\hat{Y}$ to the actual option price Y at each of a set 50 discretized time steps between initial and terminal times across 10 path

realizations. The validation relative error calculated by

$$\epsilon_n^m = \frac{|\hat{Y_n^m} - Y_n^m|}{Y_n^m} \tag{3.1}$$

at each time step. Path realization is then averaged, first across all path realizations and then across all time steps to obtain one single validation relative error ϵ for each training iteration. Training for both MLMC and MC deep neural networks continue until ϵ reaches an user set validation relative error value as done to bench mark MLMC's numerical modelling performance to MC in [7]. To compare the two sampling methods, total training time and the total number of time steps required for the deep neural network model to reach ϵ are used. The total number of time steps is calculated through taking a summation across all the discretized time steps for all iterations. As an example, if the MLMC sampled deep neural network model required 3 total training iterations and trained across 3 layers of MLMC with 2, 4, and 8 discretized time steps respectively, the total number of time steps is $2+4+8=14$. For MC, this summation of the total number of time steps simplifies to the product between the total number of training iterations and the constant number of time steps for each path realization. MLMC and MC neural network models with different path sampling hyperparameters (N_l and h for MLMC and N for MC) are trained until reaching various ϵ and the total time and total time steps are recorded. ϵ values are chosen based on industry preference for model accuracies better than 5%.

3.2 Optimizing MLMC Hyperparameters

To optimize MLMC hyperparameters, the same formulation of MLMC and the deep neural network as described in 3.1 is made. For hyperparameter exploration, two MLMC hyperparameters are considered: the factor h in which subsequent MLMC layers increase time steps and the number of training iterations run at each layer N_l. To explore hyperparameters, a grid search of the combination of both hyperparameters is performed, given the limited possible search space for both hyperparameters. The search space for hyperparameter exploration in this thesis is guided by Gile's h hyperparameter exploration, which found an optimum value of $h = 7$ [7], and limited by the performance capabilities of the device used for training.

3.3 Stochastic Interest Rates: Bond Options

To formulate the modelling of stochastic interest rates with a deep neural network, an approach similar to the relation of PDEs and SDEs and subsequent discretization from [13] is used. There are slight differences with the previous method, starting with the underlying asset being measured. Instead of the prediction of an option price at a given point of time as derived from an underlying stock price, the price of a zero coupon bond at a given time t, which is a derivative that pays a value of 1 at a maturity time in the future T, is predicted. A zero coupon bond is chosen to model stochastic interest rates due to its usage as a building block in interest rate derivative theory.

The approach starts with the stochastic interest rate initially defined through the Two-Factor Hull White model under risk neutral conditions:

$$\begin{aligned} dr_t &= \alpha(\theta_t - r_t)dt + \sigma dW_t^1, \\ d\theta_t &= \beta(\phi - \theta_t)dt + \eta dW_t^2, \end{aligned} \tag{3.2}$$

where r_t is short run interest rate at t, θ_t is long run interest rate at t, W_t^1 and W_t^2 are two components of two-dimensional Brownian motion $W = (W^1, W^2)$, and α, β, σ, and η are positive constants. With the application of the Two-Factor Hull White model, it is known that the price of a zero coupon bond at time t with a payoff of 1 at maturity T is given by the expected value:

$$P_t(T) = \mathbb{E}^{\mathbb{Q}}[e^{-\int_t^T r_u du} | \mathcal{F}_t]. \tag{3.3}$$

where $\mathcal{F}_t$ is the filtration of information available at t and r is short run interest rate. Though 3.3 can be modelled with various numerical estimation methods, the methods are largely inefficient. This inefficiency can be bypassed through relating it to a PDE with the Feynman-Kac theorem, which allows the representation of the expectation of a stochastic process as a PDE [6]. To set up the Feynman-Kac theorem application, stochastic processes r and θ are combined to be a two dimensional stochastic process X, where $X_t = (r_t, \theta_t)$, such that

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t \tag{3.4}$$

and μ and σ are two dimensional functions

$$\mu(t,x,y) = \begin{bmatrix} \alpha(y-x) \\ \beta(\phi - y) \end{bmatrix} \qquad \sigma(t,x,y) = \begin{bmatrix} \sigma & 0 \\ 0 & \eta \end{bmatrix}, \tag{3.5}$$

which include components from 3.2. After defining functions $V(t,x,y) = x$ and $f(x,y) = 1$ and a generator A required for the Feynman-Kac theorem, the theorem can be applied and simplification can be performed to relate 3.3 to function $u(t, r_t, \theta_t)$:

$$P_t(T) = u(t, r_t, \theta_t), \tag{3.6}$$

where $r_t = x$ and $\theta_t = y$. Here, $u(t, r_t, \theta_t)$ (or equivalently $u(t,x,y)$) is a function satisfying complex PDE:

$$\begin{aligned} \frac{\partial u}{\partial t} + \alpha(y-x)\frac{\partial u}{\partial x} + \beta(\phi - y)\frac{\partial u}{\partial y} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\eta^2\frac{\partial^2 u}{\partial y^2} &= xu \\ u(T,x,y) &= 1. \end{aligned} \tag{3.7}$$

At this stage, the Euler-Maruyama scheme can be applied to discretize 3.4 and the deep neural network can be set up to predict $P_t(T)$ at given time t. The predicted $P_t(T)$ from the neural network can be compared with the known explicit solution to 3.7.

Results for deep neural network accuracy in predicting $P_t(T)$ have not yet been attained at the time of this report and is outlined in next steps under section 5.

Chapter 4

Results and Discussion

Results for this thesis is attained for comparing MLMC and MC in data generation for deep neural network training outlined in 3.1 and for exploring MLMC hyperparameters when optimizing time taken to reach validation relative error ϵ outlined in 3.2. Results have not yet been attained for modelling complex PDEs via zero coupon bonds as outlined in 3.3 and are left for next steps outlined in section 5.

4.1 Comparing MLMC and MC Results

To compare the time efficiency of MLMC and MC sampling methods for deep neural network data generation, the total training time in seconds and the total time steps required to reach a certain ϵ are measured. The results are in 4.1 and 4.2 respectively. In both 4.1 and 4.2, each data point represents the average total time or total number of time steps across 3 separate seed runs for various hyperparameter combinations. For MC, the total number of discretized time steps N is varied. For MLMC, the factor h in which the number of time steps in subsequent layers increases by and the number of training iterations for each layer N_l are varied.

From 4.1 and 4.2, it is observed that MLMC reaches ϵ values with a smaller amount of total run time and total time steps as compared to those required by MC to reach the same ϵ value. Specifically, when comparing the best hyperparameter combinations for both MLMC and MC such that the total run time and the number of time steps are minimized, MLMC requires 1.4-1.6 times less training time and 5.1-7.2 times less total time steps when compared to MC across all ϵ values. The combination of MLMC's time step and total training time improvements over MC can be seen in table A.1 and plotted in graph A.1, which display MLMC requiring an average of 23.58 times less seconds per time step in comparison to MC at a given validation relative error. These time improvements from MLMC compared to MC

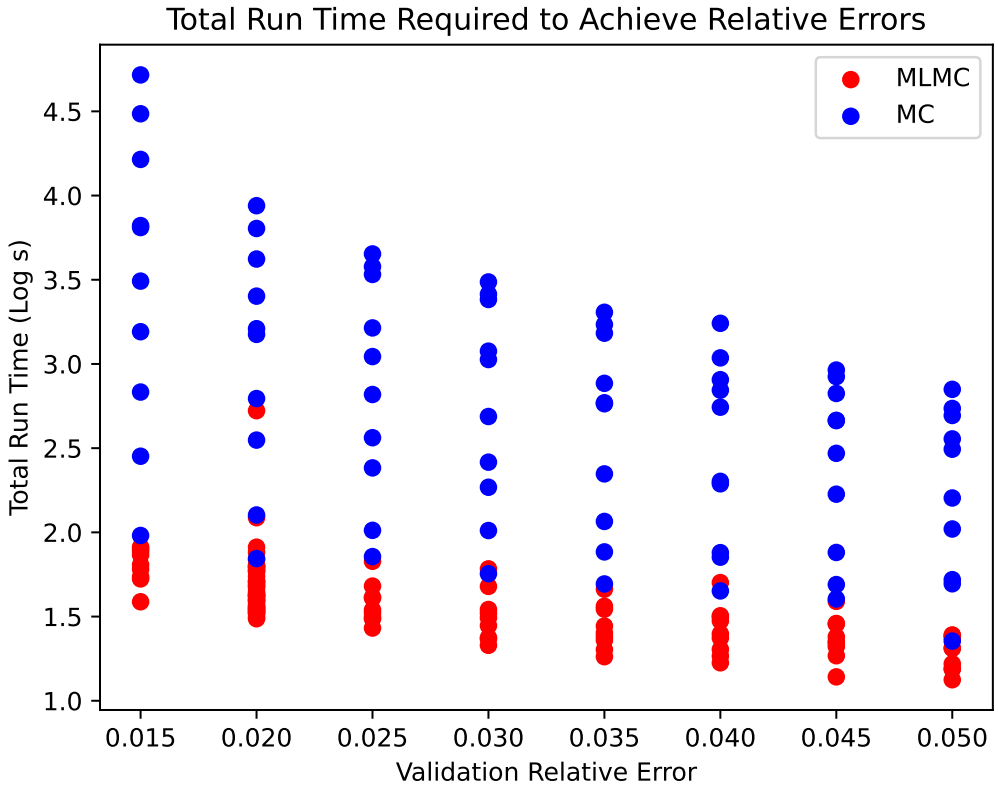

Figure 4.1: The total time in seconds required for MLMC and MC to reach a certain validation relative error on a log scale.

may be attributed to the low to high number of time steps discretized by MLMC at increasing layers. By starting at a low number of time steps, MLMC allows the deep neural network to first learn general features of the PDE solution. Then after progressive iterations, the deep neural network learns progressively more specific and complex features through more time steps. This is analogous to contemporary deep learning methods which aim to learn feature hierarchies, with lower general features learning used to support higher and more specific feature learning [9].

Further insight on MLMC's performance in comparison to that of MC can be drawn from A.2 and A.3, which respectively plot MLMC's factor of improvement for total training time and total time steps over MC across validation relative errors. From A.3, it is observed that the improvement for the total number of time steps generally increases as validation relative error decreases, with a significant drop for when validation relative error is 0.015. Despite this trend, A.2 does not display a similar pattern for the total training time improvement of MLMC over MC as relative error decreases. The source of these inconsistency of results between time step and

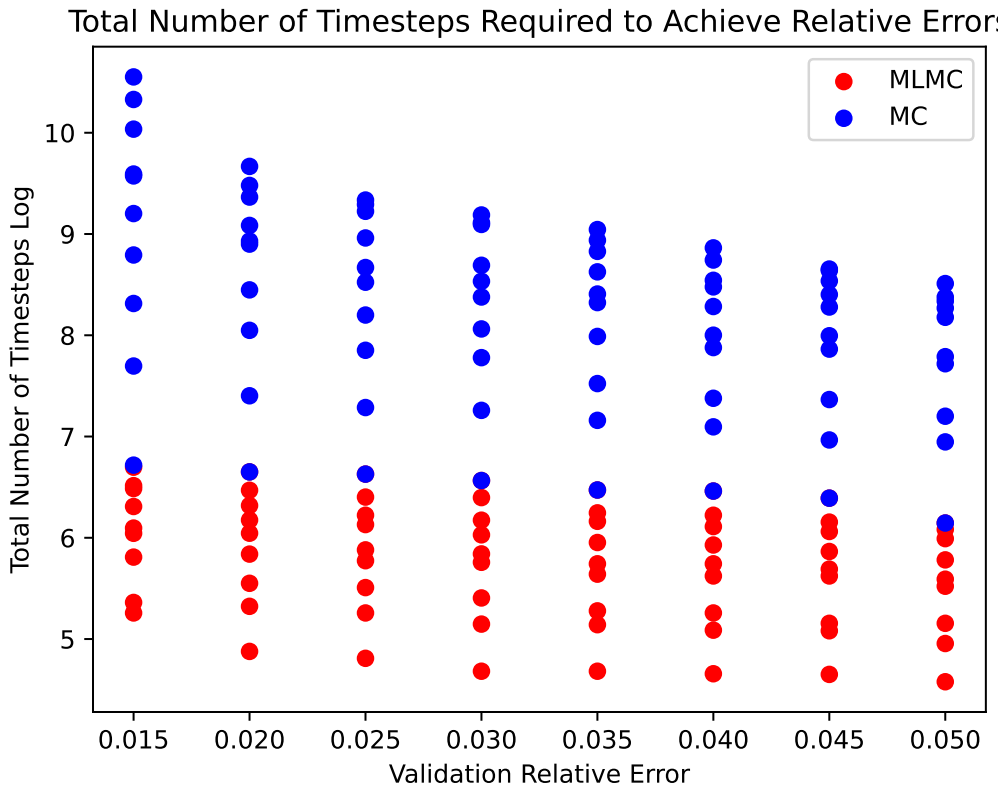

Figure 4.2: The total number of time steps required for MLMC and MC to reach a certain validation relative error on a log scale.

overall time improvement is currently unclear and require more seed runs to clarify possible trends.

In 4.1 and 4.2, it is also observed that the spread of MLMC's points increase slower in comparison to the spread of MC's points as relative error decreases. This indicates that hyperparameter selection and optimization is more important for MC than it is for MLMC as different hyperparamter combinations seem to vary MC's performance much more in comparison to that of MLMC.

From these aforementioned figures, it is initially concluded that MLMC path sampling exhibits a significant time improvement over MC path sampling for deep neural network data generation. While these are promising results, there are several limitations to the overall analysis. The first limitation is the limited number of seed runs. Given that only 3 seeds runs are selected for each MLMC and MC model and corresponding data point on 4.1 and 4.2, both abnormally high or low performing seeds can substantially skew results. This can be mitigated through performing and averaging the data across more seed runs. Another limitation is the implementation of both MC

and MLMC path sampling methods. In this current comparison, the most simplistic form of MC is implemented without consideration for possible performance boosting methods, such as Richardson extrapolation as seen in [7]. MLMC's implementation is also simplified for the purpose of this thesis. A fixed number of training iterations per layer N_l is used in place of a variable N_l calculated on the fly as seen in [7]. Further performance boosting methods with MLMC is also left unexplored. This indicates that the benefits of MLMC over MC may vary in realistic applications where either sampling method is optimized for its use case. A third limitation is the simplicity of the PDE modelled. Despite MLMC being likely to perform better than MC in settings involving more complex PDEs considering the substantial performance increase for simplistic PDEs, experiments are still to be performed to show that this is the case.

4.2 MLMC Hyperparameter Optimization

To explore MLMC hyperparameters, a grid search is applied for the following combinations of $h = 2, 3, 4, 5, 6, 7, 8, 9, 10$ and $N_l = 50, 100, 150, 200, 250$. The log normalized time steps required to reach a validation relative error of 0.02 is seen in heatmap 4.3. Time steps for $h = 7, 8, 9, 10$ and $N_l = 50$ are not measured due to very large run times. In 4.3, it is seen that the total number of time steps increases as h increases,

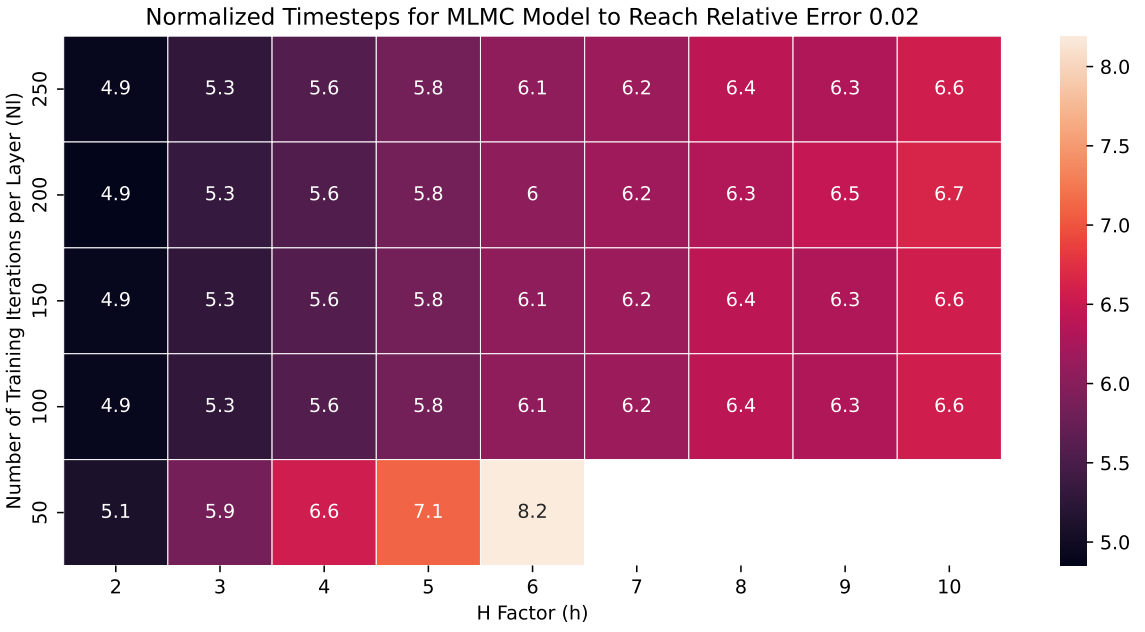

Figure 4.3: The log normalized of the total number of time steps required for an H Factor and N_l MLMC hyperparameter combination to reach 0.02 validation relative error.

but seem largely independent to changes in N_l. Despite [7] citing fastest numerical convergence of MLMC modelling a PDE when $h = 7$, the increase of required time steps with the increase in h is plausible given the differing use cases of MLMC. Whereas [7] uses MLMC for numerical modelling of PDEs, this project uses MLMC for training deep neural networks, which are then modelling underlying PDEs. As a result, larger h values may result in larger rates of time step accumulation, preventing the neural network from capturing more general PDE features during a smaller number of discretized time steps before moving on to capture more specific features during a larger number of discretized time steps. This may result in time steps increasing at a faster rate without yielding a faster reduction in validation relative error. The result for N_l is unexpected and may be attributed to the deviation of MLMC's implementation in this thesis when compared to the implementation by Giles in [7]. Given that [7] calculates a variable N_l for each layer, a fixed N_l for all MLMC layers as implemented here may possibly average out the optimal and sub optimal N_l values in various layers, thus obtaining an average that performs moderately across all layers. These trends for h and N_l are also exhibited through varying validation relative errors in C.1 and C.2. As validation relative error decreases, the total number of time steps required increases non linearly: the increase in the number of time steps required to improve validation relative error from 0.015 to 0.01 is far more than that required to improve validation relative error from 0.05 to 0.02. This is generally expected, since progressively more training iterations is required by the neural network to reach smaller validation relative errors, which results in more data and time steps needed to be generated.

When observing one singular training and validation run for a MLMC deep neural network, B.1 and B.2 are plotted for the MLMC model with the optimal set of hyperparamters found $h = 2$ and $N_l = 200$. Training is terminated when validation relative error is 0.02. In B.1, it is observed that the learned option price generally follows the exact option price with some deviations when sudden movements from the exact option price occur. This points towards a weakness of the deep neural network model and path sampling policy of being vulnerable against sudden changes in the underlying stochastic process. In B.2, both the mean and standard deviation of the relative error across a validation batch is seen to increase as time steps progress towards terminal time. This indicates that while the current uniform sampling policy demonstrated by MLMC (and MC) may be effective for generally training the deep neural network across all time steps, more training iterations will need to be done on specific segements of time for the deep neural network to reliably predict all time steps offered. These validation relative error paths differ from the MC results obtained by

[13], which demonstrated MC's convergence at initial and terminal time. However, deviations in training approaches warrant these differences. While [13] terminated training when a certain number of iterations are met, the approach outlined in this thesis terminated training when a certain validation relative error is met. Furthermore, the threshold validation relative errors set in this paper, which are based on the accuracy expectations of option pricing models in industry, are far greater than the validation relative errors achieved in [13], which leaves room for more error and path deviations seen in this thesis.

Overall, it is concluded that an optimal hyperparameter set for MLMC include a small h value, the best being $h = 2$ in these experiments. On the other hand, MLMC performance does not seem to very by N_l. There are several limitations to these results. The first limitation is similar to that mentioned in 4.1: the small number of seed runs (3) for each hyperparameter combination allows for very effective or poor training runs to largely skew the averaged result for each result. This can be mitigated with more seed runs (ie: 10). Another limitation is the deviation from the MLMC model in [7], which uses variable N_l compared to the fixed N_l iterations at each MLMC layer for each hyperparameter combination. This may effect overall results for h, since the fixed N_l run with each h may have been more optimal for some h compared to others. Note that if a calculated N_l is to be implemented, the MLMC model will only have h as a hyperparameter. A third limitation is the simplicity of the MLMC model used in this approach; no performance boosting methods are used for this thesis, which may have offered more diverging results for hyperparameter tuning. This can be mitigated and explored through implementing performance boosting methods such as adaptive sampling, which samples and trains more iterations at high loss time steps [16].

Chapter 5

Conclusion

With PDEs' prevalent use in quantitative finance, deep neural networks have been shown to be a promising solution in mitigating the CoD. Due to the slow time complexity of current MC methods to generate deep neural network training data and the simplicity of PDEs modelled, the goals of this thesis are to increase time efficiency of path sampling and to apply deep neural networks to more complex PDEs involving stochastic interest rates. Accomplishment of these goals reveals more of the potential deep neural networks have in realistic quantitative finance applications. To address the former goal of reducing path sampling time complexity, MLMC based on [7] is substituted as the path sampling method in place of MC following a formulation in [13]. To compare the performance of MLMC and MC, total training time and total number of time steps required to reach various validation relative errors are measured. To address the latter goal of modelling more complex PDEs involving stochastic interest rates with deep neural networks, a formulation resembling the approach in [13] is performed. By introducing the Two-Factor Hull White model and the known payoff of zero coupon bonds, a SDE is modelled and related to an accompanying PDE with the Feynman-Kac theorem, to which discretization by the Euler-Marayama schema can then be applied for deep neural network formulation.

The results of this thesis met the former goal for time complexity reduction with MLMC requiring 1.4-1.6 less training time and 5.1-7.2 less total time steps as compared to MC across all validation relative error values. This shows that MLMC is a promising replacement for MC in speeding up the deep neural network data generation, which allows neural networks to more effectively and efficiently address the CoD when modelling PDEs. For the second goal, results for the combined MLMC and deep neural network model currently are not available for more complex PDEs involving stochastic interest rates as experiments are still be to run. While these initial results of applying MLMC to deep neural networks are promising, the several limitations of

the limited number of seed runs (3), the simplification of MLMC from [7] with fixed N_l, and the lack of other performance boosting methods run for both MLMC and MC warrants further exploration into the effectiveness. Next steps of this thesis include first and foremost the experiments of stochastic interest rates with zero coupon bonds to be run as outlined in 3.3. Increasingly complex PDEs involving stochastic interest rates implemented through bond options and swaptions can then be explored to examine the combined MLMC and deep neural network model's capabilities for solving more complex PDEs. To mitigate the aforementioned limitations, next steps also involve more seed runs to reveal more reliable MLMC-MC comparisons and hyperparameter trends, MLMC implementation using variable N_l to allow benchmarking of MLMC with performance in literature, and other performance boosting methods, such as adaptive sampling, in which additional paths and training are completed for highly evaluated loss time steps [16]. These next steps comprehensively reveal more and more about MLMC's potential in being used in conjunction with deep neural networks to model PDEs and to mitigate the CoD.

Bibliography

[1] Atilim Gunes Baydin et al. *Automatic differentiation in machine learning: a survey*. 2018. arXiv: 1502.05767 [cs.SC].

[2] Arnaud Blanchard. *The Two-Factor Hull-White Model : Pricing and Calibration of Interest Rates Derivatives*. Accessed: 2022-01-21. 2012. URL: https://www.math.kth.se/matstat/seminarier/reports/M-exjobb12/120220b.pdf.

[3] Russel E. Caflisch. "Monte Carlo and quasi-Monte Carlo methods". In: *Acta Numerica* 7 (1998), pp. 1–49. DOI: 10.1017/S0962492900002804.

[4] *Curse of Dimensionality*. https://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality. Accessed: 2022-03-14.

[5] Weinan E, Jiequn Han, and Arnulf Jentzen. "Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations". In: *Communications in Mathematics and Statistics* 5.4 (Nov. 2017), pp. 349–380. ISSN: 2194-671X. DOI: 10.1007/s40304-017-0117-6. URL: http://dx.doi.org/10.1007/s40304-017-0117-6.

[6] Forrest Flesher. *Stochastic Processes and the Feynman-Kac Theorem*. URL: https://scholar.harvard.edu/files/forrestgflesher/files/final_paper_final.pdf.

[7] Michael B. Giles. "Multilevel Monte Carlo Path Simulation". In: *Operations Research* 56.3 (2008), pp. 607–617. DOI: 10.1287/opre.1070.0496. eprint: https://doi.org/10.1287/opre.1070.0496. URL: https://doi.org/10.1287/opre.1070.0496.

[8] Jiequn Han, Arnulf Jentzen, and Weinan E. "Solving high-dimensional partial differential equations using deep learning". In: *Proceedings of the National Academy of Sciences* 115.34 (2018), pp. 8505–8510. ISSN: 0027-8424. DOI: 10.1073/pnas.1718942115. eprint: https://www.pnas.org/content/115/34/8505.full.pdf. URL: https://www.pnas.org/content/115/34/8505.

[9] S. Jothilakshmi and V.N. Gudivada. "Chapter 10 - Large Scale Data Enabled Evolution of Spoken Language Research and Applications". In: *Cognitive Computing: Theory and Applications*. Ed. by Venkat N. Gudivada et al. Vol. 35. Handbook of Statistics. Elsevier, 2016, pp. 301–340. DOI: https://doi.org/10.1016/bs.host.2016.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0169716116300463.

[10] Frances Y. Kuo and Ian H. Sloan. "Lifting the Curse of Dimensionality". In: 2005.

[11] *Mean reversion*. https://www.nasdaq.com/glossary/m/mean-reversion. Accessed: 2022-01-21. 2018.

[12] Jan Palczewski. *Numerical schemes for SDEs*. https://www.mimuw.edu.pl/~apalczew/CFP_lecture5.pdf. 2021.

[13] Maziar Raissi. *Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations*. 2018. arXiv: 1804.07010 [stat.ML].

[14] Fabrice Douglas Rouah. *Euler and Milstein Discretization*. Accessed: 2022-01-12.

[15] Mike Stecher. *Brownian Motion and Stochastic Differential Equations*. 2012.

[16] Kejun Tang, Xiaoliang Wan, and Chao Yang. *DAS: A deep adaptive sampling method for solving partial differential equations*. 2021. DOI: 10.48550/ARXIV.2112.14038. URL: https://arxiv.org/abs/2112.14038.

[17] Pietro Veronesi. *Fixed income securities: Valuation, risk, and risk management*. J. Wiley amp; Sons, Inc., 2011.

[18] Wenzhong Zhang and Wei Cai. *FBSDE based Neural Network Algorithms for High-Dimensional Quasilinear Parabolic PDEs*. 2021. arXiv: 2012.07924 [math.NA].

Appendix A

Other MLMC and MC Comparisons

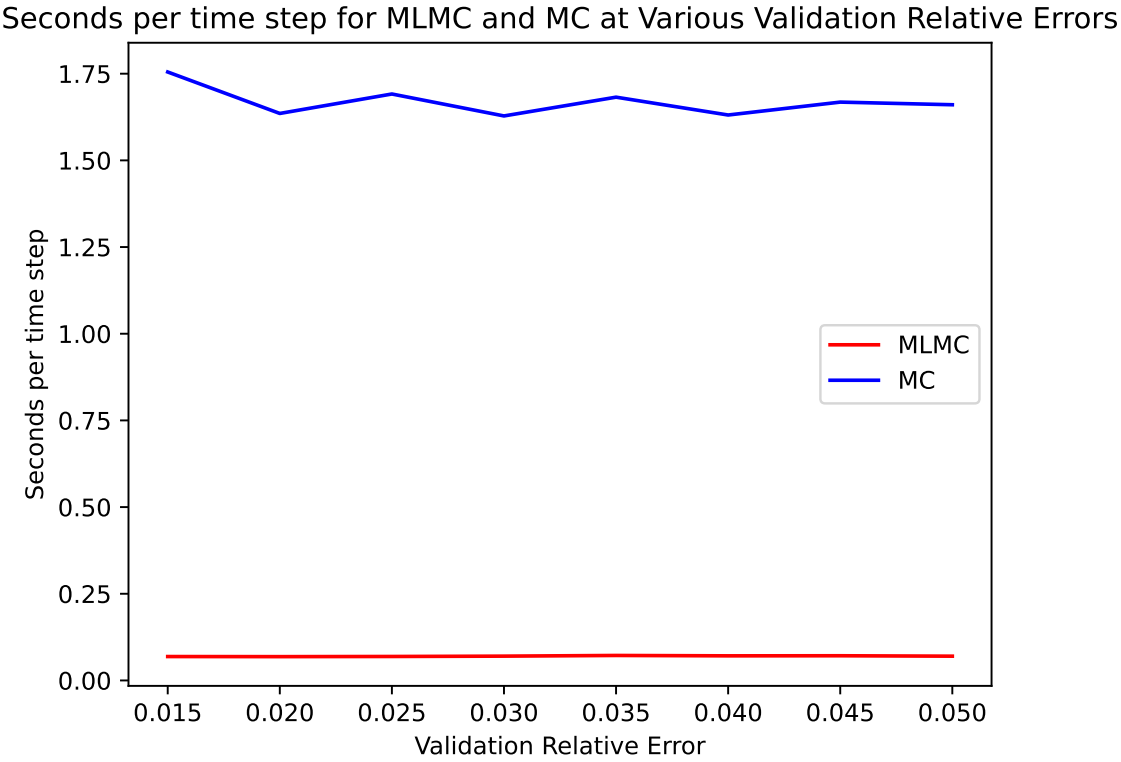

Figure A.1: The seconds per time step required for the best models of both MLMC and MC at each respective validation relative error.

Table A.1: The Seconds per Time Step Required by the Best Model of MLMC and MC for Each Respective Relative Error

Validation Relative Error	MLMC(s)	MC(s)	MC/MLMC
0.015	0.068922	1.75491	25.46226169
0.02	0.068647	1.635333	23.82235203
0.025	0.069117	1.691431	24.47199676
0.03	0.070092	1.628132	23.22849969
0.035	0.0782062	1.68228	21.51082651
0.04	0.070955	1.630835	22.98407441
0.045	0.0712	1.668044	23.42758427
0.05	0.069992	1.660436	23.72322551

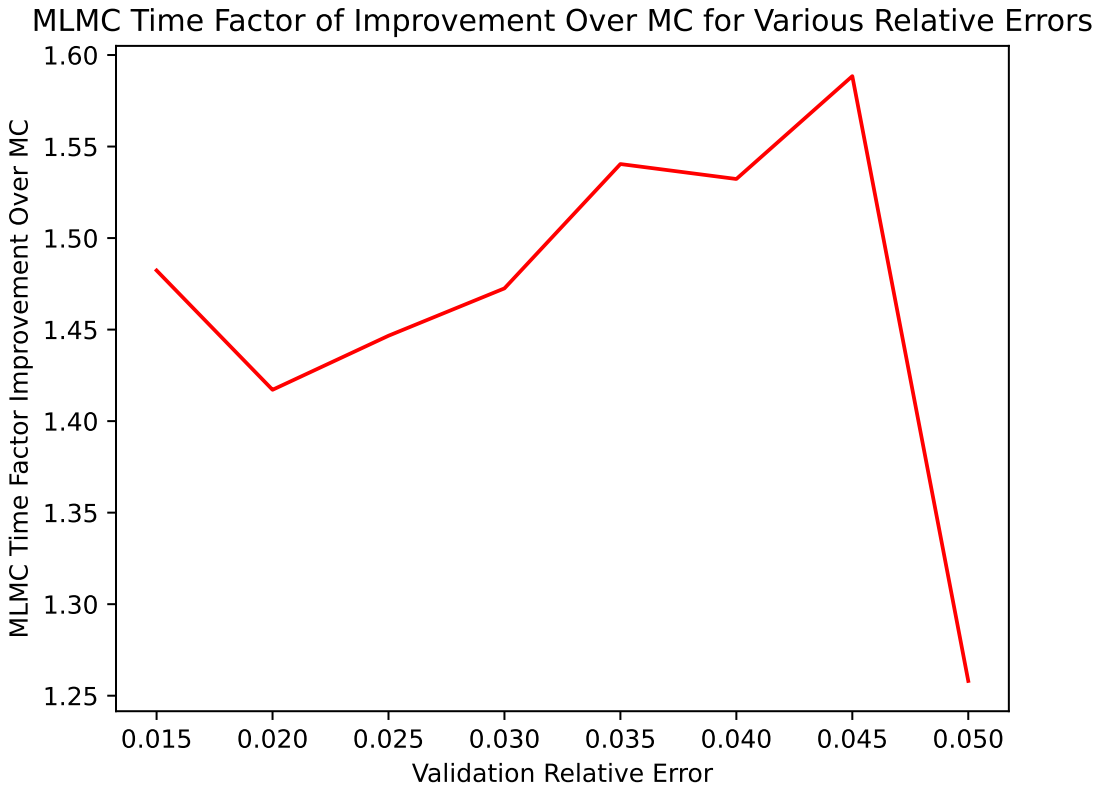

Figure A.2: The factor of improvement for the total training time the best model of MLMC displays over the best model of MC across varying validation relative error tested.

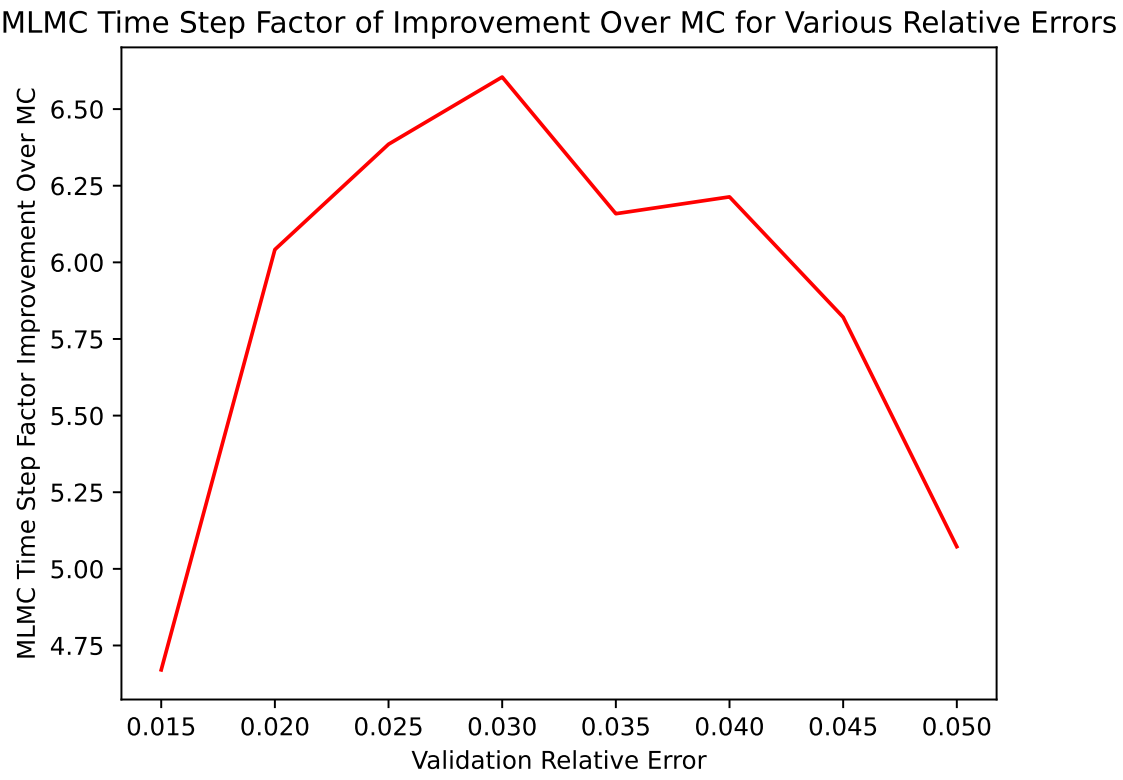

Figure A.3: The factor of improvement for the number of time steps the best model of MLMC displays over the best model of MC across varying validation relative error tested.

Appendix B

MLMC Run Graphs

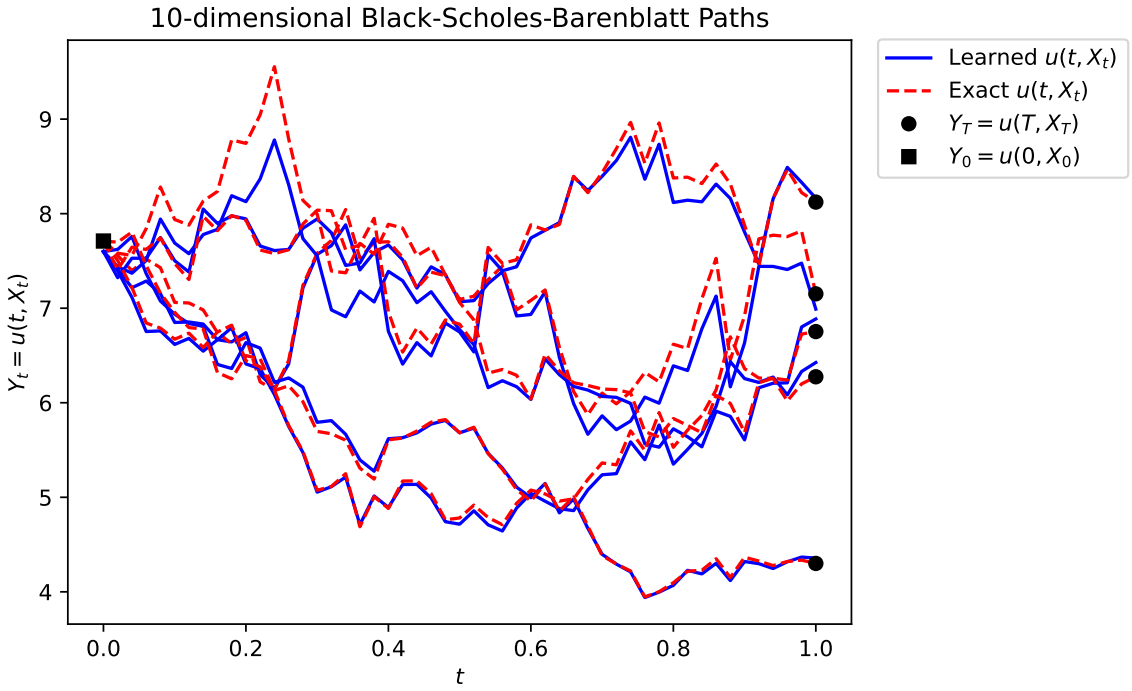

Figure B.1: 5 sample paths generated from the MLMC deep neural network predicting option prices between initial to terminal time compared to the option prices obtained from the explicit solution with a validation relative error of 0.02.

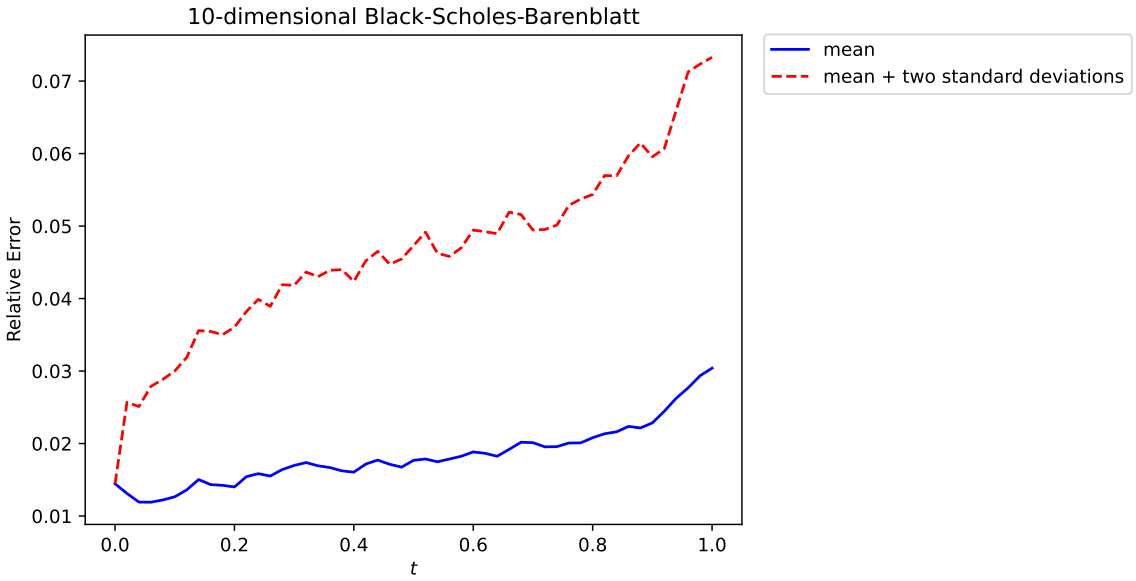

Figure B.2: The mean and two standard deviations of a MLMC deep neural network across a batch of 100 validation sampling paths. The deep neural network's is trained when a validation relative error of 0.02 is reached.

Appendix C

MLMC Hyperparameter Exploration

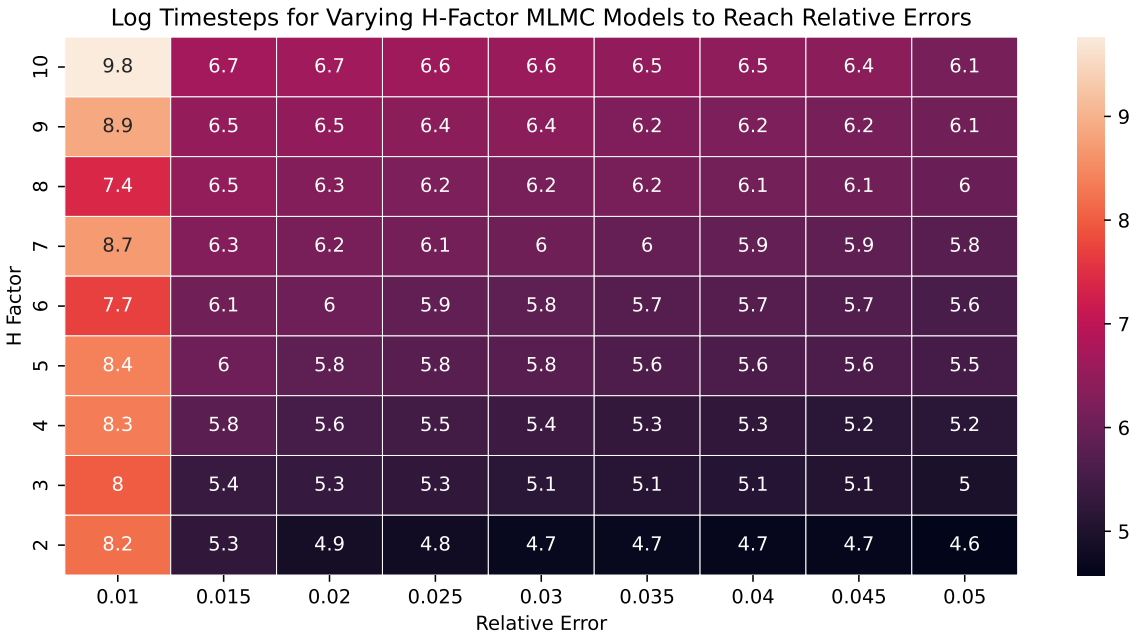

Figure C.1: The log normalized total time steps required for MLMC with various h values to reach respective validation relative errors.

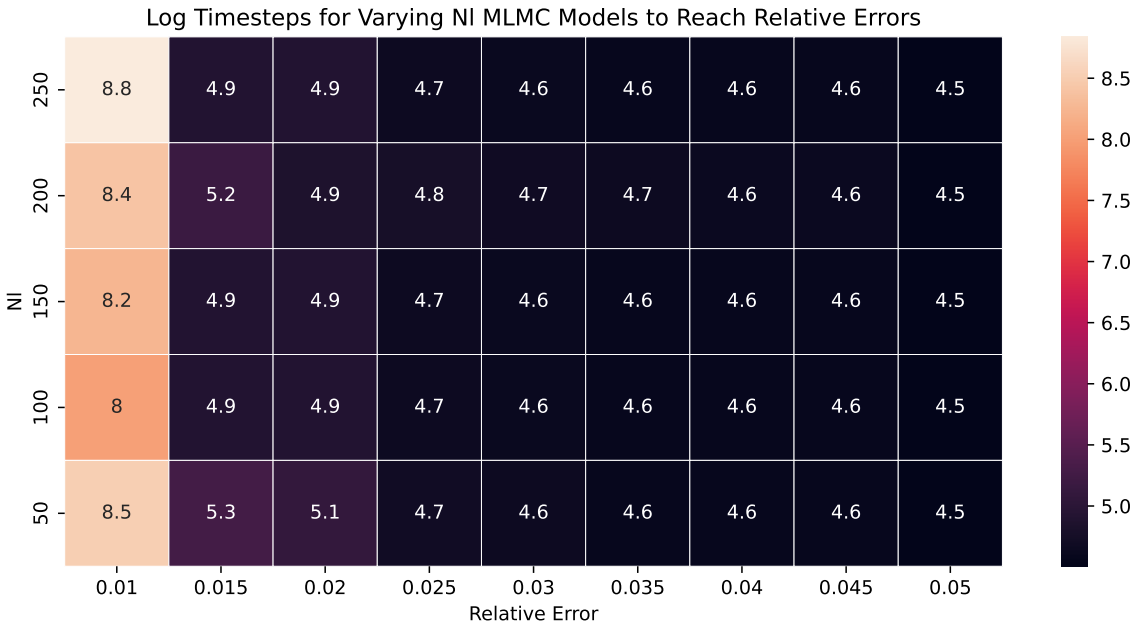

Figure C.2: The log normalized total time required for MLMC with various N_l values to reach respective validation relative errors.