Comparative Analysis of Deep Galerkin Method and Finite Difference Method for Solving PDEs in Portfolio Optimization

W.M. Manamela

orcid.org/0009-0006-0879-7415

Dissertation accepted in fulfillment of the requirements for the degree Master of Science in Applied Mathematics at the North-West University

Supervisor: Dr I Takaidza
Co-Supervisor: Dr ME Sonono
Co-Supervisor: Dr TCM Jele

DECLARATION

I hereby declare that the research work Comparative Analysis of Deep Galerkin Method and Finite Difference Method for Solving PDEs in Portfolio Optimization is of my own originality. All of the sources used in this study are acknowledged and included in the bibliography. The research work is submitted in fulfillment of the requirements for the degree Master of Science in Applied Mathematics at the North-West University, Department of Mathematics and Applied Mathematics.

ACKNOWLEDGEMENTS

First and foremost, let me take this opportunity to thank my esteemed supervisors, Dr. Takaidza, Dr. Sonono, and Dr. Jele, for their guidance, support, comments, and efforts in making me feel warmly welcomed in the academic sphere.

I would also like to express my appreciation to North-West University (NWU) for accepting me as their student.

A profound thank you is also due to my family, my CPUT colleagues, and Dr. Buzuzi for their unwavering support.

Lastly, I take this opportunity to express gratitude to God for the precious gifts of life, health, and knowledge that He has bestowed upon me.

ABSTRACT

Since its inception by Markowitz, mean-variance analyses have played a significant role in portfolio management, helping investors make wise and rational decisions. However, the mean variance has faced major drawbacks due to its inability to incorporate factors such as taxes and transaction costs. This has led to the framework being improved in many ways.

One of the well-known models for portfolio optimization that improves upon mean-variance analyses is the famous Merton portfolio problem. Merton himself solved the problem, utilizing the dynamical programming approach, which transformed the problem into a partial differential equation framework. It is worth noting that the resulting PDE generally lacks analytical solutions. In such cases, traditional numerical methods are employed to obtain the numerical solutions of the PDE. However, several studies have indicated that, at higher dimensions, these numerical methods present a significant computational challenge and tend to be slow. This issue is commonly referred to as the curse of dimensionality. To deal with the curse of dimensionality, deep learning algorithms are now being used.

This study aims to investigate the performance of the deep learning algorithm, the Deep Galerkin method (DGM), in comparison to the finite difference method (FDM). At first, both the mean-variance analysis framework and the Merton problem framework are presented. To solve the Merton problem, the HJB equation is utilized to transform the problem into the nonlinear partial differential equation (PDE) and the associated optimal controls. Furthermore, the resulting PDE and optimal control are solved by implementing Python code for both the DGM and FDM. The results demonstrate that, in general, the former outperforms the latter. We also observed that, at various time points, DGM consistently provided more accurate results compared to FDM.

Keywords: Deep Learning, Merton-Portfolio problem, Hamiltonian-Jacobi-Bellman equation, Deep Garlekin Method, Finite Difference Methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	:	Artificial neural networks
DGM	:	Deep Garleking method
DNN	:	Deep neural network
EFDM	:	Explicit finite difference method
FDM	:	Finite difference method
HJB	:	Hamilton-Jacobi-Bellman equation
IFDM	:	Explicit finite difference method
KdV	:	Korteweg-de Vries
MBGD	:	Mini batch gradient descent
MPT	:	Modern portfolio theory
MVA	:	Mean-variance-analyses
ODE	:	Ordinary differential equation
OVF	:	Optimal value function
PDE	:	Partial differential equation
ReLU	:	Rectified linear unit
SDE	:	Stochastic differential equation
SGD	:	Stochastic gradient descent

1. INTRODUCTION

1.1 Background

One of the most complex problems confronting both academic scholars and professionals is portfolio optimization. Portfolio optimization is a process used by investors to select the best portfolio from a list of eligible portfolios (Cartanyà Caro, 2022). It was initially put into practice by Markowitz (1952), who is regarded as the father of modern portfolio theory (MPT). Markowitz developed a mathematical framework based on the concept of diversification. This framework enables investors to manage their risk and return expectations by spreading their investments across a variety of assets with different risks and potential returns.

1.2 Problem statement

Even though the MPT theory is considered to be a major breakthrough in portfolios management, it has faced quite a number of criticisms listed as follows:

- The Higher the risk the higher the returns: The idea that investors will only take on higher risk if they are offered higher expected returns is often not supported by their actual behavior. In some cases, investment strategies require investors to take on risky investments, such as derivatives or futures, with no significant increase in expected returns, in order to reduce overall risk (McClure, 2010). Furthermore, investors may have other considerations that are more important than the distribution of returns, such as personal preferences or utility functions (Mangram, 2013).
- Investor irrationality: The assumption is that investors possess rational behaviour and make informed decisions on investment. However, investors are often influenced by group behaviours and make decisions based on emotions. This means that investors

tend to prioritize short-term gains over long-term investment strategies. These behaviours contradict the assumption of rationality among investors and often lead to sub-optimal investment outcomes (Mangram, 2013; Morien, 2005).

- Unlimited access to capital: The assumption is that investors are allowed to borrow any unlimited amount of money from the financial institutions at any risk-free interest rate. This assumption is not true because financial institutions do not allow investors to borrow money in any manner they prefer. Thus, the limitations of investors' borrowing capacity and their access to risk-free assets affect the practicality of MPT.
- Data requirements: MPT requires a large amount of data to effectively optimize a portfolio, including estimates of asset returns, variances, and covariances. However, obtaining accurate data can be challenging, and errors in the data can lead to errors in portfolio optimization (Mangram, 2013; Morien, 2005).
- No taxes or transaction costs: MPT does not consider taxes or transaction costs. However, in practice, investment products are subject to both taxes and transaction costs, such as brokers' fees and administrative expenses, which can significantly impact portfolio returns. Therefore, considering these costs when constructing portfolios is crucial for investors who aim to optimize their returns in the real world. Ignoring these costs can lead to sub-optimal portfolio selection and negatively affect investment outcomes (Mangram, 2013; Modigliani & Miller, 1963).
- Market efficiency: MPT assumes that markets are efficient and that asset prices reflect all available information. However, some argue that markets are not always efficient, and that certain assets may be mispriced, leading to sub-optimal portfolio performance (Morien, 2005).

Several authors have contributed to improving MPT beyond the original work of Harry Markowitz. Some of the most notable include: Modigliani and Miller (1963) who invented a mathematical technique for assessing how taxes and inflation can affect the amount of money earned from investing. They discovered that the amount of money you make after paying taxes depends on how different investments are taxed.

Several articles on investor overreaction were co-authored by Richard Thaler and Werner De Bondt (1985, 1987). Thaler and De Bondt questioned the efficient market hypothesis, which states that financial markets are always fully efficient and that asset prices always reflect all available information.. According to their findings, investors tend to overreact to new information, resulting in financial market mispricing. They also discovered that, companies that fared poorly in the past tended to outperform in the future, and equities that performed well tended to underperform.

Robert Merton developed a continuous-time finance framework, which uses stochastic calculus and dynamic programming to analyze the best portfolios of risky and risk-free assets. Merton's model considers the investor's preferences for risk and reward and seeks to maximize the expected utility of the investor's terminal wealth (Merton, 1969).

One of the key advantages of Merton's approach is that it allows for the incorporation of various types of uncertainty, such as changes in interest rates, market volatility, and other macroeconomic factors. This makes the model more robust and better able to account for unexpected events that may impact portfolio performance. Many researchers and academics improved Merton's problem under different factors. For example: **(a)** Benth et al. (2003) used the Ornstein-Uhlenbeck (OU) type stochastic volatility model to study the Merton problem in the context of the Black-Scholes model. **(b)** Pedersen and Peskir (2017) solved the nonlinear mean-variance optimal portfolio selection problem using the Lagrange multiplier. **(c)** B. Han and Wong (2021) conducted an extensive investigation of Merton's portfolio problem, employing the Volterra Heston model as the basis of their study. **(d)** Lastly, Farhadi et al. (2017) present a derivation of a novel formulation for Merton's optimal problem, incorporating a fractional stochastic stock price. Their work also explores the practical applications of this new framework.

Addressing complex dynamics behind Merton's work and portfolio optimization often involves resorting to Partial Differential Equations (PDEs), given their remarkable versatility in modelling intricate financial systems (Al-Aradi et al., 2019; Al-Aradi et al., 2018; Li et al., 2021; Sirignano & Spiliopoulos, 2018). In this research, we seek to embark on a

comparative study of two numerical methods, namely the Deep Galerkin Method (DGM) and the Finite Difference Method (FDM), both instrumental in solving PDE-based portfolio optimization problems.

The DGM is a novel approach that employs deep learning techniques, such as neural networks, to approximate solutions to PDEs (Li et al., 2022). Its adaptability and potential to handle high-dimensional problems make it an intriguing candidate for portfolio optimization. On the other hand, the FDM is a traditional numerical technique known for its simplicity and robustness in solving PDEs by discretizing the domain into smaller intervals. By scrutinizing the performance of these methods on portfolio optimization, we aim to shed light on their effectiveness in generating optimal asset allocations, fostering well-informed financial decision-making, and advancing the field of portfolio management. This research has the potential to significantly impact investment strategies, risk assessment, and overall financial stability.

1.3 Machine learning and deep learning

The original Markowitz MPT has undergone several changes over the years because it was shown to have many flaws. As a result, machine learning and deep learning algorithms are now being used in the field of study to address these flaws.

Machine learning is defined as a scientific discipline within the field of artificial intelligence that enables computers to learn and improve their performance without being explicitly programmed (El Naqa & Murphy, 2015; Wiederhold & McCarthy, 1992). Machine learning algorithms use previous data as input to create predictions or detect patterns in new data. (Krishnamachari, 2017). On the other hand, deep learning is defined as a sub-field of machine learning that is designed to replicate the capabilities of the human brain. It achieves this by utilizing neural networks with multiple layers, known as deep neural networks. These computational structures operate autonomously and have the ability to predict or solve complex problems.

One of the key advantages of machine learning and deep learning in finance is their ability to optimize portfolios. By analyzing large amounts of data, machine or deep learning algorithms can identify the most efficient portfolio allocation based on risk and return objectives. This can lead to better performance and lower risks for investors(Cartanyà Caro, 2022).

To demonstrate the strength and importance of deep learning, several studies involving using deep neural networks to solve PDEs have been conducted concurrently. Lee & Kang (1990) and Lagaris et al. (2000) proposed a deep learning algorithm on a fixed mesh. Malek and Beidokhti (2006) also advocated using a numerical hybrid Deep Neural Network (DNN) optimization technique to solve numerical solution for high-order differential equations. However, these grid-based approaches would be computationally inefficient for PDEs of higher dimension: the curse of dimensionality.

Recently, multiple studies have been conducted to employ deep learning approaches to overcome the curse of dimensionality. Pedersen & Peskir (2017) and J. Han et al. (2018) suggested a Feynman-Kac formula for deep-learning backwards stochastic differential equation.

The Feynman-Kac formula:

If heat equation with a cooling term is given as:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\frac{\partial^2 u}{\partial x^2} - K(x)u,$$

where $K > 0$ and relates to the amount of external cooling at a specific location x in the space domain R. Then the Feynman-Kac formula reads:

$$u(t,x) = \mathbb{E}^x \exp\left(-\int_0^t K(B_\tau)d\tau\right) g(B_\tau), \tag{1.1}$$

where we consider the probability measure $\mathbb{P}$, the process B represents a d-dimensional Brownian motion initiated at x, and the function u is the only solution to the heat equation.

Sirignano and Spiliopoulos (2018) suggested a deep learning algorithm: Deep Garlekin

Method to approximate solutions of higher order PDEs without mesh points. Also, Al-Aradi et al. (2019) extended the DGM algorithm to solve higher-order partial differential equations. They introduced a new method for solving Fokker-Planck equations and expanded the DGM algorithm to solve for the value function and optimal control simultaneously employing the Hamilton-Jacobi-Bellman (HJB) equations. Furthermore, they trained the newly proposed deep neural networks employing alternating stochastic gradient descent steps and demonstrated the efficiency of the approaches through numerical tests.
The general Fokker-Planck equation is given as follow:

$$\frac{\partial F}{\partial t} = \left[\frac{\partial}{\partial x}P^{(1)}(x) + \frac{\partial^2}{\partial x^2}P^{(2)}(x)\right]F, \tag{1.2}$$

where $P^{(2)}(x)$ is referred to as diffusion coefficient and $P^{(1)}(x)$ is the drift coefficient.

In adddition, Li et al. (2022) applied the DGM to solve the general Stokes equation. The authors of the article have attested to the potential advantages of DGM over traditional numerical methods in reducing computational complexity and achieving competitive results in a wide range of high-dimensional problems.
The general Stokes equation is given as follows:

$$\begin{aligned} \kappa u - v\nabla^2 u + \nabla\rho &= f, \text{in } D \\ \nabla.u &= 0, \text{in } D \\ u &= g, \text{on } \partial D, \end{aligned}$$

where $D \subset \mathbb{R}$, $\partial D \subset \mathbb{R}$ is the Dirichlet boundary condition, κ is a positive constant, v is the viscosity coefficient, u and ρ denote the velocity and pressure, respectively, and f and g represent the source terms.

Also, Li et al. (2021) developed a DGM that uses neural networks to approximate the solution of a second-order linear elliptic equation. The method combines the strengths of the Galerkin method and deep learning to achieve high accuracy and computational efficiency. Yang and Zhu (2021), on the other hand, proposed a local deep learning method

that uses a small neural network to approximate the solution of high order partial differential equations. This method reduces the computational cost of traditional deep learning methods and can handle complex boundary conditions and geometries.

1.4 Research aim and objectives

1.4.1 Research aim

The aim of this study is to investigate the performance of the Deep Galerkin method and Finite difference methods in approximating solutions, specifically the optimal value function and optimal control derived from the partial differential equation linked to a Merton portfolio problem. The performance of the DGM will be assessed in comparison to the finite difference method.

1.4.2 Research Objectives

The overall research objectives for the study are characterized as follows:

- To review and understand the theoretical foundations of portfolio optimization and the formulation of PDEs in this context.
- To investigate the Finite Difference Method and Deep Galerkin Method and gain insights into their numerical principles.
- To implement both methods for solving PDEs arising in portfolio optimization.
- To compare and evaluate the computational efficiency, accuracy, and robustness of FDM and DGM.
- To draw conclusions and provide recommendations on the most suitable method for portfolio optimization PDEs.

1.5 Delimitations of the study

This research will only look at the finite difference method as the numerical method for comparing the Deep Learning algorithm. Other numerical approaches, such as finite elements, finite volume, Monte Carlo methods, multigrid methods, and domain decomposition methods, will not be investigated.

Furthermore, due to time constraints, it is important to note that while the DGM has the potential to handle higher-dimensional PDEs, the implementation of the algorithm in the study is restricted to handling data up to one dimension.

1.6 Dissertation overview

The overall research is organised as follows:

Chapter 1: Provides a valuable introduction to the dissertation specifically, the Modern portfolio Theory. It explains the problem statement, some literature, the research aim and the objectives. It also explains the delimitation of the study.

Chapter 2: This chapter presents definitions and mathematical preliminaries of stochastic calculus, dynamic programming and utility theory.

Chapter 3: This chapter provides a detailed overview of PDEs and computational methods used to solve them.

Chapter 4: This chapter provides a detailed overview of the mean-variance analyses and the Merton portfolio problem. Furthermore, this chapter provides theoretical foundation, mathematical representation and implementation of deep Garlekin method and finite difference framework.

Chapter 5: This chapter discusses and compares various numerical findings achieved by implementing deep learning and finite differences in Python as stated in Chapter

4. Furthermore, draw conclusions about which computational method provide the most accurate approximation of the results.

Chapter 6: Concludes the study and provide possible suggestions on future research.

2. STOCHASTICS CALCULUS, OPTIMAL CONTROL AND UTILITY THEORY

This chapter introduces mathematical preliminary of stochastic calculus, dynamic programming and utility theory. It is divided into 3 sections. Section 2.1 aims to presents basic concept such as random process, probability space, Brownian motion and the formulations of stochastic differential equation and Itôs' lemma. Section 2.2 is concerned with optimal control and the formulation of the HJB equation. While, section 2.3 presents the utility theory framework.

2.1 Stochastics calculus

2.1.1 Probability space

Definition 2.1.1. *(Shreve et al., 2004; Wang, 2019) Probability space is a set of tripled* $(\Omega, \mathcal{F}, \mathbb{P})$*, where:*

- Ω *is a sample space which is the set of all possible results of an experiment.*
- σ*-algebra* $\mathcal{F}$*. The set* $\mathcal{F}$ *is called* $\sigma-$*algebra if the following conditions are satisfied:*
  - $\Omega \in \mathcal{F}$.
  - $\emptyset \in \mathcal{F}$.
  - *complements:if* $A_n \in \mathcal{F}\ \forall n \in \mathbb{N}$ *then* $A_n^c \in \mathcal{F}$.
  - *countable intersections :if* $A_n \in \mathcal{F}\ \forall n \in \mathbb{N}$ *then* $\cap_{n\in\mathbb{N}} A_n \in \mathcal{F}$.
  - *countable unions: if* $A_n \in \mathcal{F}\ \forall n \in \mathbb{N}$ *then* $\cup_{n\in\mathbb{N}} A_n \in \mathcal{F}$.
- $\mathbb{P}$ *is the probability measure, which assigns a probability value to each event in the sample space. The probability values belong to the interval* $[0, 1]$.

2.1.2 Filtration

Definition 2.1.2. *(Shreve et al., 2004; Wang, 2019) If $\mathcal{F}_t$ satisfy the property $\mathcal{F}_t \subset \mathcal{F}$ and is a family of σ-algebras for any $s \leq t$ then $\{\mathcal{F}_t\}_{t\geq 0}$ is a filtration on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$.*

2.1.3 Brownian motion

Definition 2.1.3. *(Shreve et al., 2004; Wang, 2019) A stochastic process, $\{W_t : 0 \leq t \leq \infty\}$, is called a standard Brownian motion if the following holds:*

- $W_0 = 0$.
- *The sample paths of W_t are almost surely continuous.*
- *Has a stationary increment, meaning $W_t - W_s$ are independent of the values of W_t whenever $s < t$.*
- *It is normally distributed with mean zero and standard deviation t or has Gaussian distribution, $W_t \sim N(0, t)$.*

2.1.4 Stochastic integral

Definition 2.1.4. *Consider an m-dimensional Brownian motion $W = \left(W_t^1 \cdots W_t^m\right)_{t\in T}$ and d-dimensional process $X = \left(X_t^1 \cdots X_t^d\right)_{t\in T}$ then the integral*

$$\int_0^T X_t(\omega) dW_t(\omega) := \lim_{n\to\infty} \int_0^T X_t^n(\omega) dW_t(\omega), \tag{2.1}$$

is said to be stochastic integral.

2.1.5 Stochastic differential equations

Definition 2.1.5. *(Benth, 2003; Shreve et al., 2004) An n-dimensional process, X_t, is a process that can be represented as:*

$$X_t = X_0 + \int_0^t \mu_s ds + \int_0^t \sigma_s dW_s, \tag{2.2}$$

where W is an m-dimensional standard Brownian motion, and μ and σ are n-dimensional and $n \times m$-dimensional $\mathcal{F}_t$-adapted processes, respectively.

2.1.6 Itô's lemma

Theorem 2.1.1. *(Benth, 2003; Hull, 2000; Shreve et al., 2004) Assume that the process X has a stochastic differential given by:*

$$dX_t = \mu_t\left(t, X_t\right) dt + \sigma_t\left(t, X_t\right) dW_t, \tag{2.3}$$

where μ_t and σ_t are functions of t and X_t. If $G(t, X)$ is a twice-differentiable function then its stochastic differential equation is given by:

$$dG = \left(\frac{\partial G}{\partial t} + \mu_t \frac{\partial G}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 G}{\partial X^2}\right) dt + \sigma_t \frac{\partial G}{\partial X} dW_t \tag{2.4}$$

Below we provide an informal proof of the Itô's lemma:

Proof.

$$G(t + \Delta t, X_t + \Delta X_t) = G(t, X_t) + \frac{\partial G}{\partial t}\Delta t + \frac{\partial G}{\partial X}\Delta X_t + \frac{1}{2}\frac{\partial^2 G}{\partial X^2}\left(\Delta X_t\right)^2 + R$$

and since $\Delta X_t = X_{t+\Delta t} - X_t = \mu_t \Delta t + \sigma_t \Delta W_t$ we see that:

$$G\left(t + \Delta t, X_t + \Delta X_t\right) \approx G\left(t, X_t\right) + \frac{\partial G}{\partial t}\Delta t + \frac{\partial G}{\partial X}\left(\mu_t \Delta t + \sigma_t \Delta W_t\right) + \frac{1}{2}\frac{\partial^2 G}{\partial X^2}\left(\mu_t \left(\Delta t\right)^2 + 2\sigma_t \mu_t \Delta t \Delta W_t + \sigma_t^2 \left(\Delta W_t\right)^2\right) + R \tag{2.5}$$

Recall the following properties from the definition of the standard Brownian motion:

$$\begin{aligned} \mathbb{E}\left[\Delta W\right] &= 0, \\ \mathbb{E}\left[\left(\Delta W\right)^2\right] &= Var\left[\Delta W\right] = \Delta t, \\ Var\left[\left(\Delta W\right)^2\right] &= 2\left(\Delta t\right)^2. \end{aligned}$$

Notice that variance of (ΔW) approaches zero in a similar manner as $\Delta t \to 0$, but converges much faster to zero than the latter. As a result, the expectation is that $(\Delta W)^2$ will exhibit increasingly deterministic behavior as Δt approaches zero.

The values derived from the informal logic presented above are as follows:

$$\left(\Delta t\right)^2 = 0, \quad \Delta t \Delta W = 0, \quad \left(\Delta W\right)^2 = \Delta t. \tag{2.6}$$

Substituting back (2.6) into Equation (2.5) yields:

$$G\left(t + \Delta t, X_t + \Delta X_t\right) \approx G\left(t, X_t\right) + \frac{\partial G}{\partial t}\Delta t + \frac{\partial G}{\partial X}\left(\Delta t + \sigma_t \Delta W_t\right) + \frac{\sigma_t^2}{2}\frac{\partial^2 f}{\partial X^2}\Delta t$$

Thus, the results yields the Itô's lemma:

$$dG \approx \left(\frac{\partial G}{\partial t} + \mu_t \frac{\partial G}{\partial X} + \frac{\sigma^2}{2}\frac{\partial^2 G}{\partial X^2}\right)dt + \sigma_t \frac{\partial G}{\partial X} dW_t$$

□

Remark 2.1.1. *For the multidimensional case. Let X_t be a n-dimensional process with the*

dynamics:

$$dX_t = \mu_t\left(t, X_t\right) dt + \sigma_t\left(t, X_t\right) dW_t, \tag{2.7}$$

then the function $G(t, X_t)$ has the differential:

$$dG \approx \left(\frac{\partial G}{\partial t} + \sum_{i=1}^{n} \mu_t^i \frac{\partial G}{\partial X^i} + \frac{1}{2}\sum_{i=1}^{n} C^{i,j} \frac{\partial^2 G}{\partial X^i \partial X^j}\right) dt + \sum_{i=1}^{n} \sigma_t \frac{\partial G}{\partial X} dW_t$$

2.2 Optimal control

Definition 2.2.1. *The optimization criterion is:*

$$J(t, x, \pi) = \mathbb{E}^{t,x,\pi}\left[\int t^T G\left(s, X_s^\pi, \pi_s\right) + F\left(X_T^\pi\right)\right], \tag{2.8}$$

where X_s^t is the state process with the dynamics.

$$dX_s^\pi = \mu\left(s, X_s^\pi, \pi_s\right) dt + \sigma\left(s, X_s^\pi, \pi_s\right) dW_t \tag{2.9}$$

Definition 2.2.2. *(Benth et al., 2003). An investment strategy (control) $\pi = \Pi(s) : t \leq s \leq T$ is considered admissible, denoted as $\pi \in \mathcal{A}_t$, where $\mathcal{A}_t$ is the control strategy , if the following conditions are satisfied:*

- *π is progressively measurable,*
- *$\pi(\rho)$ takes values between 0 and 1 (inclusive) with almost certain probability for all $t \leq s \leq T$,*
- *There exists a single solution $X_t^\pi = x$ for equation* (2.9).

Definition 2.2.3. *The value function is:*

$$\begin{aligned} V(t,x,\pi) &= \sup_{\pi\in\mathcal{A}_t} J(t,x,\pi) \\ &= \sup_{\pi\in\mathcal{A}_t} \mathbb{E}^{t,x,\pi}\left[\int_t^T G\left(s, X_s^\pi, \pi_s\right) ds + F\left(X_T^\pi\right)\right], \end{aligned}$$

Dynamic programming

The dynamic programming principle states that we can divide the optimization problem into two sub-intervals $[t, t_1]$ in a controlled Markov process where the process is independent of previous movements. How to accomplish this is explained by Bellman's principle of optimality given as:

$$V(t,x) = \sup_{\pi\in\mathcal{A}_t} \mathbb{E}\left[\int_t^{t_1} G\left(s, X_s^\pi, \pi_s\right) + V\left(t_1, X_{t_1}^\pi\right)\right]$$

Applying the Itô's lemma to V results in:

$$V(t,x) = v(t,x) + \int_t^{t_1}\left(\frac{\partial v}{\partial t} + \mu\frac{\partial v}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 v}{\partial x^2}\right) ds + \int_t^{t_1}\frac{\partial v}{\partial x}\sigma\left(s, X_s, \pi_s\right) dW_s \tag{2.10}$$

The stochastic integral part is a martingale meaning that its expectation become zero, $\mathbb{E}\left[\int_t^{t_1}\frac{\partial v}{\partial x}\sigma\left(s, X_s, \pi_s\right) dW_s\right] = 0$. Substituting back produces:

$$V(t,x) = \sup_{\pi\in\mathcal{A}_t} \mathbb{E}\left[\int_t^{t_1} G\left(s, X_s^\pi, \pi_s\right) ds + v(t,x) + \int_t^{t_1}\left(\frac{\partial v}{\partial t} + \mu\frac{\partial v}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 v}{\partial x^2}\right) ds\right] \tag{2.11}$$

If we divide both sides with $t - t_1$ and let $t \to t_1$ the following results is obtained:

$$0 = \lim_{t\to t_1} \frac{\sup_{\pi\in\mathcal{A}_t} \mathbb{E}\left[\int_t^{t_1}\left(G\left(s, X_s^\pi, \pi_s\right) + \frac{\partial v}{\partial t} + \mathcal{L}^a v\right) ds\right]}{t}, \tag{2.12}$$

where $\mathcal{L}^a v = \mu \frac{\partial v}{\partial x} + \frac{\sigma^2}{2} \frac{\partial^2 v}{\partial x^2}$. By utilizing the Mean Value Theorem the following results is established:

$$\frac{\partial v}{\partial t} + \sup_{\pi \in \mathcal{A}_t} \{G\left(s, X_s^\pi, \pi_s\right) + \mathcal{L}^a v\} = 0. \tag{2.13}$$

The Equation (2.13) obtained above is the Hamilton-Jacobi-Bellman equation.

2.3 Utility theory

Utility is an economic term used to measure how well a person is satisfied with something or product. This product can be anything from food, diversified portfolio, or sporting activities. The utility function assigns a value to every possible outcome. It is advisable for an investor to make tough financial decisions whenever the utility function value is higher.

2.3.1 Preference ordering

Consider the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and a random variable $X_1 \in \Omega$. The satisfaction level of an investor depends on a preference ordering O.

Let U denote a utility function that captures the preference ordering in a risk-free environment. If specific assumptions are met, the preference ordering can be represented using expected utility. Below are the assumptions that need to be fulfilled for this result to hold true. These assumptions can be found in (Fishburn, 1970; Milanesi, 2020).

Assumption 1 (Rationality): The preference ordering O as rational when it fulfills the following criteria.

- Reflexivity: For all X_1 in the set Ω, $X_1 O X_1$ holds true.
- Completeness: For all X_1 and X_2 in the set Ω, either $X_1 O X_2$ or $X_2 O X_1$ holds true.

- Transitivity: For all X_1, X_2 and X_3 in the set Ω, if $X_1 O X_2$ and $X_2 O X_3$ then $X_1 O X_3$ holds true.
- Continuity: For all X_2 in the set Ω, $\{X_1 \in \Omega \mid X_1 O X_2\}$ and $\{X_1 \in \Omega \mid X_2 O X_1\}$ are closed set.

Assumption 2 (Continuity): The preference ordering O is continuous if for all X_1, X_2, X_3 in Ω such that $X_1 O X_3$ and $X_2 O X_3$, there exists a in the domain $[0, 1]$ such that $(aX_1 + (1 - a)\, X_3)$ indifference to X_2.

Assumption 3 (Independence): The preference ordering O is independent if for all X_1, X_2, X_3 in Ω and there exists a in the domain $[0, 1]$, $X_1 O X_3 \iff (aX_1 + (1 - a)\, X_3) O (aX_2 + (1 - a)\, X_3)$.

2.3.2 Money lotteries

For more information see (Alexander, 2021; Fishburn, 1970).

- **Today:** A specific scenario within the context of decision-making under conditions of uncertainty, where the resultant outcomes are evaluated in terms of monetary values.
- **Lottery:** A lottery can be represented by a cumulative distribution function, denoted as $\mathbf{F}$, which operates on the real number line $\mathbb{R}$.
- **Preference ordering utility function**: The utility function formula that captures the preference ordering in a risk-free environment is represented as:

$$U(\mathbf{F}) = \mathbb{E}_{\mathbf{F}}[u(X_1)] = \int u(X_1) d\mathbf{F}(X_1). \tag{2.14}$$

2.3.3 Expected value versus expected utility

Expected value of lottery **F** is:

$$\mathbb{E}_{\mathbf{F}}\left[X_1\right] = \int X_1 d\mathbf{F}(X_1). \tag{2.15}$$

Expected utility of lottery **F** is:

$$\mathbb{E}_{\mathbf{F}}\left[u(X_1)\right] = \int u(X_1) d\mathbf{F}(X_1). \tag{2.16}$$

By comparing the expected value and expected utility, we can gain greater insight into an investor risk attitudes.

2.3.4 Risk appetite

Definition 2.3.1. *An investor is considered to be (Alexander, 2021; Fishburn, 1970; Milanesi, 2020):*

- *Risk-averse if he consistently favors the certain wealth amount* $\mathbb{E}\left[X_1\right]$ *over the* **F** *lottery,*

$$\int u(X_1) d\mathbf{F}(X_1) \leq u\left(\int X_1 d\mathbf{F}(X_1)\right) \quad \forall \mathbf{F}. \tag{2.17}$$

- *Risk-neutral if he is indifferent to risk,*

$$\int u(X_1) d\mathbf{F}(X_1) = u\left(\int X_1 d\mathbf{F}(X_1)\right) \quad \forall \mathbf{F}. \tag{2.18}$$

- *Risk loving if he approves any lottery that is currently fair,*

$$\int u(X_1) d\mathbf{F}(X_1) \geq u\left(\int X_1 d\mathbf{F}(X_1)\right) \quad \forall \mathbf{F}. \tag{2.19}$$

- *Strictly Risk-averse if he declines any lottery that is currently fair.*

2.3.5 Risk aversion and concavity

Please note that the statement in equation (2.18) is called the Jensen's inequality, named after the Danish mathematician Johan Jensen. Infact, the Jensen's inequality holds true if and only if U is concave (Alexander, 2021; Milanesi, 2020). Which implies that an investor is:

- Strictly risk-loving $\iff$ U is convex.
- Risk-neutral $\iff$ U is linear.
- Risk-averse $\iff$ U is concave.

2.3.6 Certainty equivalent

The certainty equivalent of a lottery $\mathbf{F}$ is the definite wealth amount that produces an equivalent expected utility as $\mathbf{F}$, that is:

$$C\mathbb{E}\left(\mathbf{F}, u\right) = u^{-1}\left(\int u(X_1) d\mathbf{F}(X_1)\right) \quad \forall \mathbf{F}. \tag{2.20}$$

From the above the following conditions holds true (See Alexander (2021)):

- An investor is strictly risk-averse $\iff$ $C\mathbb{E}\left(\mathbf{F}, u\right) \leq C\mathbb{E}_{\mathbf{F}} \quad \forall \mathbf{F}$.
- An investor is risk-neutral $\iff$ $C\mathbb{E}\left(\mathbf{F}, u\right) = C\mathbb{E}_{\mathbf{F}} \quad \forall \mathbf{F}$.
- An investor is risk-loving $\iff$ $C\mathbb{E}\left(\mathbf{F}, u\right) \leq C\mathbb{E}_{\mathbf{F}} \quad \forall \mathbf{F}$.

2.3.7 Quantifying risk attitude

The risk premium (P_u) formula is given as follows (Milanesi, 2020):

$$\begin{aligned} P_u &= \text{the expected value minus the certainty equivalent} \\ &= \mathbb{E}_{\mathbf{F}}\left[X_1\right] - C\mathbb{E}\left(\mathbf{F}, u\right). \end{aligned} \tag{2.21}$$

To get the expected utility of lottery from equation (2.21) above we perform some algebraic operation below:

$$\begin{aligned} P_u &= \mathbb{E}_{\mathbf{F}}\left[X_1\right] - u^{-1}\left(\int u(X_1) d\mathbf{F}(X_1)\right) \\ \Longrightarrow u^{-1}&\left(\int u(X_1) d\mathbf{F}(X_1)\right) = \mathbb{E}_{\mathbf{F}}\left[X_1\right] - P_u \\ \Longrightarrow &\int u(X_1) d\mathbf{F}(X_1) = u\left(\mathbb{E}_{\mathbf{F}}\left[X_1\right] - P_u\right) \\ \Longrightarrow &\mathbb{E}_{\mathbf{F}}\left[u(X_1)\right] = u\left(\mathbb{E}_{\mathbf{F}}\left[X_1\right] - P_u\right). \end{aligned}$$

Let $X_1 = y + \epsilon$ with $y \in \mathbb{R}$, ϵ belong to the set Ω is a random variable with a mean of zero and a variance of σ^2. If u possesses second-order differentiability, we can formulate the Taylor series in relation to ϵ as :

$$u(y+\epsilon) = u(y) + \epsilon u'(y) + \frac{\epsilon^2}{2} u''(y) + o(\epsilon^2).$$

Applying the knowledge of expected value throughout yields:

$$\begin{aligned} \mathbb{E}_{\mathbf{F}}\left[u(X_1)\right] = \mathbb{E}_{\mathbf{F}}\left[u(y+\epsilon)\right] &= \mathbb{E}_{\mathbf{F}}\left[u(y) + \epsilon u'(y) + \frac{\epsilon^2}{2} u''(y) + o(\epsilon^2)\right] \\ &= u(y) + \frac{\sigma^2}{2} u''(y). \end{aligned} \tag{2.22}$$

A construction of the first order condition for certainty equivalence is given as follows:

$$u(y - P_u(X_1)) = u(y) - P_u(y) u'(X_1) + o(P_u(X_1)).$$

Also, applying the knowledge of expected value throughout yields:

$$\mathbb{E}_{\mathbf{F}}\left[u(y - P_u(X_1))\right] = u(y) - P_u(X_1)u'(y). \tag{2.23}$$

By equating the relevant components of (2.21) with those of both (2.22) and (2.23), we derive an expression for the risk premium:

$$u(y) + \frac{\sigma^2}{2}u''(y) = u(y) - P_u(X_1)u'(y)$$
$$P_u(y) = -\frac{1}{2}\frac{u''(y)}{u'(y)}\sigma^2$$

From the above one obtains the *coefficient of absolute risk aversion* that defines various categories of utility functions:

$$R_A = -\frac{u''(y)}{u'(y)}.$$

Also, the *relative risk aversion* is another tool that characterizes different classes of utility functions.

$$R_R = -\frac{u''(y)}{u'(y)}.$$

Note that the *relative risk aversion* depends directly on wealth y.

2.3.8 Classification of utility functions

The utility function u is considered to be (Milanesi, 2020; Pandi, 2020):

- DARA (Decreasing Relative Risk Aversion) if for all $y \in \mathbb{R}^+$ and $R'_R < 0$.
- CARA (Constant Absolute Risk Aversion) if for all $y \in \mathbb{R}^+$ and $R'_R = 0$.
- IARA (Increasing Absolute Risk Aversion) if for all $y \in \mathbb{R}^+$ and $R'_R > 0$.

2.3.9 Families of utility functions

More information on utility functions can be found in (Milanesi, 2020; Pandi, 2020).

- Quadratic utility: $u(y) = y - \frac{a}{2}y^2$ with $a > 0$, it belongs to IARA utility family.
- The exponential utility: $u(y) = -e^{-ay}$ with $a > 0$, it belongs to CARA utility family.
- The logarithmic utility: $u(y) = \ln(y)$, it belongs to DARA utility function.
- Power utility: $u(y) = \frac{1}{1-p}y^{1-p}$, it belongs to DARA utility family.

3. COMPUTATIONAL APPROACHES FOR SOLVING PDEs

This chapter presents partial differential equations and computational approaches in details. It consists of 3 sections. Section 3.1 deals with the definition of PDE, different classes and forms of PDEs. Section 3.2 provides a solid foundation for various traditional numerical methods such as explicit finite differences, implicit finite differences, and Crank-Nicolson method. Section 3.3 details the theoretical foundations and mathematical formulation of Deep Learning. For more details refer to (Al-Aradi et al., 2018; Asaad, 2019; Byjus, 2023; Strauss, 2007)

3.1 Partial differential equations

3.1.1 Definition

An ordinary differential equation (ODE) comprises an unknown function $u(x)$ that is purely dependent on the equation's independent variable x. An ordinary differential equation, $\frac{du}{dx}$, is defined by the relationship between the derivative of an unknown function and the independent variable x. In general the ODE takes the form:

$$F(x; u; u_x; u_{xx}) = 0.$$

A partial differential equation, as opposed to an ordinary differential equation, involves an unknown function $u(x, y, z)$ that is dependent on many independent variables. A PDE is the partial derivatives of this unknown function with respect to the many independent variables x, y, z. (Al-Aradi, 2018; Byjus, 2023). The general form of this is:

$$F(x; y; u(x; y); u_x(x; y); u_y(x; y); u_{xx}(x; y); u_{xy}(x; y); u_{yy}(x; y)) = 0.$$

Some examples of PDEs (See Strauss, 2007):

$$\text{Transport equation: } u_x + u_y = 0 \tag{3.1}$$

$$\text{Wave equation: } u_{tt} - u_{xx} = 0 \tag{3.2}$$

$$\text{Laplace's equation: } u_{xx} + u_{yy} = 0 \tag{3.3}$$

$$\text{Heat equation: } u_t - u_{xx} = 0 \tag{3.4}$$

$$\text{Schrödinger equation: } iu_t - u_{xx} = 0 \tag{3.5}$$

$$\text{Burgers equation: } u_t + uu_x = 0 \tag{3.6}$$

$$\text{KdV's equation: } u_t + uu_x + u_{xxx} = 0 \tag{3.7}$$

3.1.2 Classification of PDEs

Order. The order of a partial differential equation is determined by the highest order derivative included in the equation (Al-Aradi, 2018; Strauss, 2007), . In the provided examples above Equations (3.1) and (3.6) are of first order. Equations (3.2), (3.3) and (3.5) are of second order while Equation (3.7) is of third order.

Linearity. In this instance both the unknown function and its derivatives appear in the equation in a linear manner (Strauss, 2007). For a simpler mathematical explanation of this property, we can express the equation in the following way:

$$\mathbf{D}u = 0, \tag{3.8}$$

where $\mathbf{D}$ is an operator that transforms u into a new function denoted as $\mathbf{D}u$.

The following property must hold true for the operator $\mathbf{D}$ to be considered linear:

$$\mathbf{D}(u + v) = \mathbf{D}u + \mathbf{D}v \quad \text{and} \quad \mathbf{D}(au) = a\mathbf{Du}, \forall \text{ functions } u, v \text{ and constant } a.$$

The Equation (3.8) is called linear if and only if the operator $\mathbf{D}$ is linear. In the previously

provided examples of PDEs, Equations (3.1) (3.2), (3.3), (3.4), (3.5) are all linear while Equations (3.6) and (3.7) are non-linear. For example if the operator **D** is applied to Equation (3.2) it produces:

$$\mathbf{D}(u+v) = (u+v)_{tt} - (u+v)_{xx} = u_{tt} + v_{tt} - u_{xx} + v_{xx} = (u_{tt} - u_{xx}) + (v_{tt} - v_{xx}) = \mathbf{D}u + \mathbf{D}v,$$

and

$$\mathbf{D}(au) = (au)_{tt} - (au)_{xx} = au_{tt} - au_{xx} = a\,(u_{tt} - u_{xx}) = a\mathbf{D}u.$$

To explore the case of non-linearity, attention is given to the issues that arises with Equation (3.7) as follows:

$$\begin{aligned}
\mathbf{D}(u+v) &= (u+v)_t + (u+v)(u+v)_x + (u+v)_{xxx} \\
&= u_t + v_t + (u+v)(u_x + v_x) + u_{xxx} + v_{xxx} \\
&= u_t + v_t + uu_x + uv_x + vu_x + vv_x + u_{xxx} + v_{xxx} \\
&= (u_t + uu_x + u_{xxx}) + (v_t + v_x + vv_x + v_{xxx}) + uv_x + vu_x \\
&\neq \mathbf{D}u + \mathbf{D}v
\end{aligned}$$

3.1.3 Forms of PDEs

PDEs can take one of the following forms:

- When an m order PDE exhibits linearity in its derivatives, it is considered quasi-linear. Both the independent variables and the derivatives of the unknown function determine the coefficients in such a PDE, with the condition that the order of these derivatives is strictly less than m (Al-Aradi, 2018; Byjus, 2023; Ganesh, 2021). The mathematical expression is given as follows:

$$A(x, y, u)u_x + B(x, y, u)u_y = F(x, y, u)$$

- A Semi-linear PDE is a quasi-linear PDE in which the coefficients of derivatives of order m are functions of the independent variables alone (x, y) (Al-Aradi, 2018; Byjus, 2023; Ganesh, 2021). The mathematical expression is given as follows:

$$A(x, y)u_x + B(x, y)u_y = F(x, y, u)$$

- A Linear PDE is one that is linear in the unknown function and all of its derivatives with coefficients that depend solely on the independent variables (Al-Aradi, 2018; Byjus, 2023; Ganesh, 2021). The mathematical expression is given as follows:

$$A(x, y)u_x + B(x, y)u_y + C(x, y) = F(x, y)$$

- A PDE which is not Quasi-linear is called a fully nonlinear PDE.

3.2 Numerical methods for PDEs

In general, it is often challenging to find analytical solutions by hands. In this instance, we must rely on numerical methods (Gim & Park, 2021). The finite difference technique is one of the most commonly used numerical techniques for solving PDEs. They accomplish this by using discretization to approximate PDEs.

This section presents mathematical properties governing FDM and describes three various FDM approaches and their mathematical features.

3.2.1 Finite difference approximation of derivatives

Forward difference

In order to find the approximate derivative of function u, it is necessary to consider the nearby grid point (j, n) (Al-Aradi, 2018; Asaad, 2019), the Taylor series expansion is expressed as follows:

$$u(x + \Delta x, t) = u(x, t) + hu'(x, t) + \frac{(\Delta x)^2}{2!}u''(x, t) + \frac{(\Delta x)^3}{3!}u'''(x, t) \tag{3.9}$$

$$u(x + \Delta x, t) = u(x, t) + \Delta x u'(x, t) + O((\Delta x)^2). \tag{3.10}$$

Hence, the forward difference approximation for $\frac{\partial u}{\partial x}$ is obtained:

$$\frac{\partial u}{\partial x} = \frac{u(x + \Delta x, t) - u(x, t)}{\Delta x}, \tag{3.11}$$

with a leading error of $O(\Delta x)$. Similarly the forward difference approximation for $\frac{\partial u}{\partial t}$ is:

$$\frac{\partial u}{\partial t} = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t}. \tag{3.12}$$

Backwards difference approximation

The Taylor series for $u(j - 1, n)$ about (j, n) is (Al-Aradi, 2018; Asaad, 2019):

$$u(x - \Delta x, t) = u(x, t) - \Delta x u'(x, t) + \frac{(\Delta x)^2}{2!}u''(x, t) - \frac{(\Delta x)^3}{3!}u'''(x, t) \tag{3.13}$$

$$u(x - \Delta x, t) = u(x, t) - \Delta x u'(x, t) + O((\Delta x)^2). \tag{3.14}$$

Hence, the backwards difference approximation for $\frac{\partial u}{\partial x}$ is obtained:

$$\frac{\partial u}{\partial x} = \frac{u(x,t) - u(x-\Delta x, t)}{\Delta x} + O(\Delta x).$$

Similarly, the backwards difference approximation for $\frac{\partial u}{\partial t}$ is obtained:

$$\frac{\partial u}{\partial t} = \frac{u(x,\Delta x) - u(x, t-\Delta t)}{\Delta x} + O(\Delta t).$$

Central difference approximation

To obtain the second order central difference add Equations (3.9) and (3.13) to get (Al-Aradi, 2018; Asaad, 2019):

$$\begin{aligned} u(x+\Delta x,t) + u(x-\Delta x,t) = u(x,t) + \Delta x u'(x,t) + \frac{(\Delta x)^2}{2!}u''(x,t) + \frac{(\Delta x)^3}{3!}u'''(x,t) \\ + u(x,t) - \Delta x u'(x,t) + \frac{(\Delta x)^2}{2!}u''(x,t) - \frac{(\Delta x)^3}{3!}u'''(x,t). \end{aligned}$$

The above lead to the central difference approximation for $\frac{\partial^2 u}{\partial x^2}$:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{(\Delta x)^2} + O((\Delta x)^2),$$

with leading error $O((\Delta x)^2)$. Similarly the central difference approximation of $\frac{\partial^2 u}{\partial t^2}$ is obtained:

$$\frac{\partial^2 u}{\partial t^2} = \frac{u(x,t+\Delta t) - 2u(x,t) + u(x,t-\Delta t)}{(\Delta t)^2} + O((\Delta t)^2).$$

To derive the first order central difference, subtract Equations (3.14) to (3.10), which yields:

$$u(x+\Delta x,t) - u(x-\Delta x,t) = u(x,t) + \Delta x u'(x,t) + O((\Delta x)^2) - u(x,t) + \Delta x u'(x,t) - O((\Delta x)^2),$$

leading to central difference approximation for $\frac{\partial u}{\partial x}$:

$$\frac{\partial u}{\partial x} = \frac{u(x + \Delta x, t) - u(x - \Delta x, t)}{2\Delta x}.$$

Similarly, the central difference approximation for $\frac{\partial u}{\partial t}$ is obtained:

$$\frac{\partial u}{\partial t} = \frac{u(x, t + \Delta x) - u(x, t - \Delta t)}{2\Delta t}.$$

3.2.2 Explicit finite difference method

The finite difference methods is demonstrated, by considering the Heat equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}. \tag{3.15}$$

In the explicit finite difference approach the temperature u_j^{n+1} at time $(n + 1)$, depend on the temperatures $(u_{j+1}^n, u_{j-1}^n, u_j^n)$ at time n (See figure 3.1). Using a forward difference at time t and a second order central difference for the space derivatives we discretize the heat equation as follows (Al-Aradi, 2018; Asaad, 2019):

$$\begin{aligned} \frac{u_j^{n+1} - u_j^n}{\Delta t} &= \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} \\ \implies u_j^{n+1} &= ru_{j+1}^n + (1 - 2r)u_j^n + ru_{j-1}^n, \end{aligned}$$

with $r = \frac{(\Delta t}{(\Delta x)^2}$. In analyzing the stability of the explicit method we look at the value of r. The finite difference method is stable if $r \leq 0.5$ and unstable if $r > 0.5$.

3.2.3 Implicit finite difference method

In the implicit finite difference approach, we are working backwards in time (Al-Aradi, 2018; Asaad, 2019). The temperatures $(u_{j+1}^{n+1}, u_{j-1}^{n+1}, u_j^{n+1})$ at time $n + 1$ depends on the

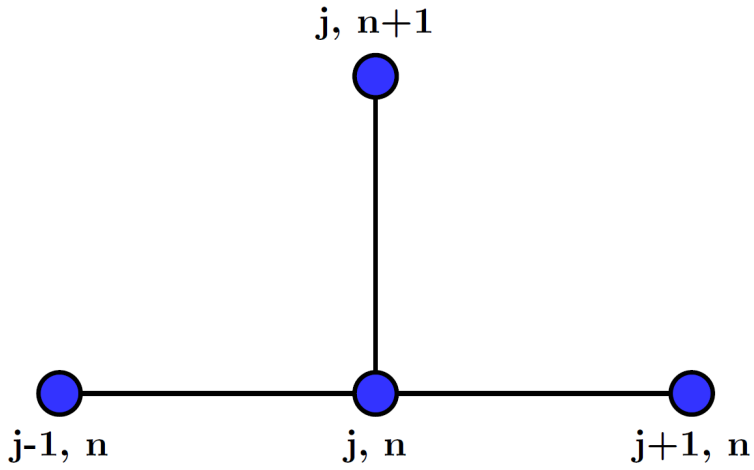


Figure 3.1: Explicit Method-stencil.

temperature u_j^n at time n (See figure 3.2).

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{(\Delta x)^2},$$

$$u_{j-1}^{n+1} - (1+2r)u_j^{n+1} + ru_{j+1}^{n+1} = -u_j^n. \tag{3.16}$$

From equation (3.16) observe that the left-hand side contains three unknown and the right-hand side only one known function.

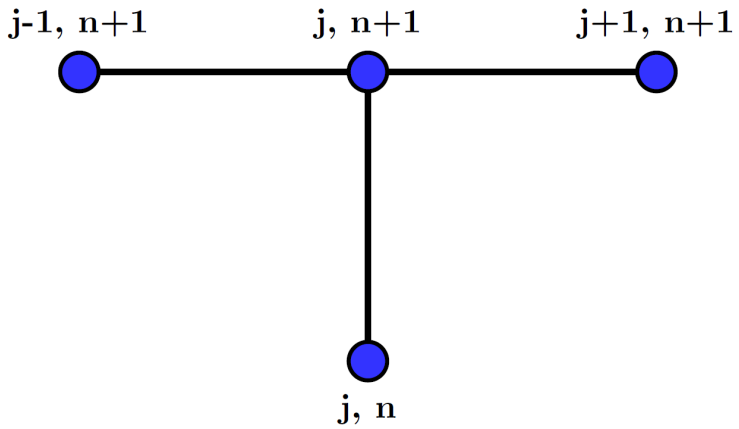


Figure 3.2: Implicit_method-stencil

3.2.4 Crank–Nicolson implicit finite difference method

To dicretize the heat Equation (3.15), this approach, consider the average of second order central difference approximation for space derivative and also take the central difference at time $t_{n+1/2}$ which gives (Al-Aradi, 2018; Asaad, 2019):

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{1}{2}\left(\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{(\Delta x)^2} + \frac{u_{j-1}^{n} - 2u_j^{n} + u_{j+1}^{n}}{(\Delta x)^2}\right),$$

$$\implies -ru_{j+1}^{n+1} + (2+2r)u_j^{n+1} - ru_{j-1}^{n+1} = ru_{j+1}^{n} - (2-2r)u_j^{n} + ru_{j-1}^{n}.$$

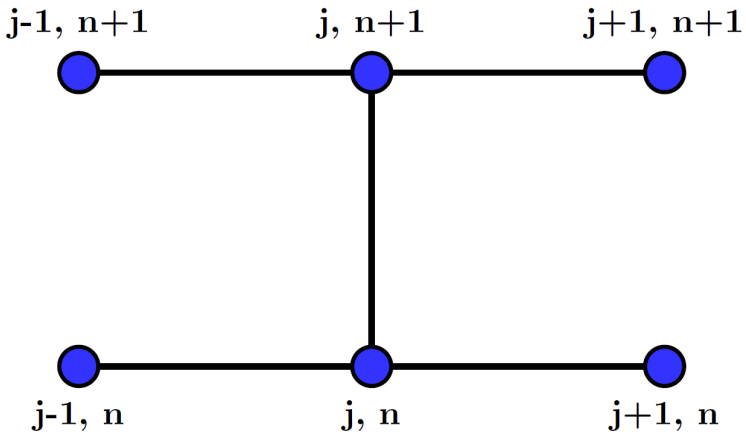


Figure 3.3: Crank–Nicolson Implicit Method-stencil.

3.3 Deep learning

The use of deep learning has attracted a great deal of attention in recent years. Several authors (Al-Aradi et al., 2019; Lee & Kang, 1990; Li et al., 2021; Pedersen & Peskir, 2017; Sirignano & Spiliopoulos, 2018; Yang & Zhu, 2021), have developed various deep learning algorithms to solve PDE. This section aims to provide an adequate background on deep learning.

3.3.1 Neural networks

Artificial neural network (ANN) is the branch of machine learning that aims to mimic the functionality of the human brain. Similar to the human brain, ANN has building blocks called neurons. These building blocks are interconnected networks that communicate with each other through nodes. A simple neural network consists of three layers: the input layer, the hidden layer and an output layer. This kind of neural network is called Feed forward artificial neural network as all the information flow in a forward manner (See Figure 3.4). All features are transferred through the nodes of the input layer, which functions as a

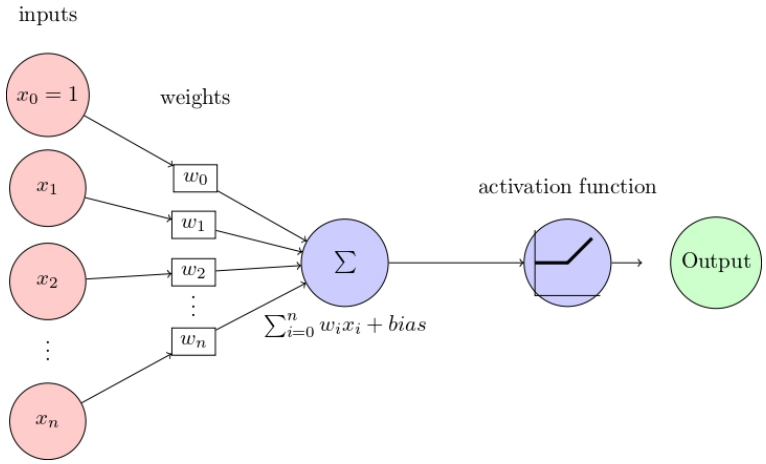


Figure 3.4: Artificial Neural Network with three layers.

receiver. The hidden layer performs all critical calculations, such as the weighted sum of the input data and the bias. The activation function at this layer converts the weighted sum

plus the bias value and then sends it to the output node. This ensures that the information delivered to the output layer is non-linear in nature (Dongare et al., 2012). A mathematical representation of the artificial neural network displayed in Figure 3.4 is given by:

$$f_j = \sigma_j \left(B_j + \sum_{i=1}^{d} w_{i,j} x_i \right), \quad j = 1, \cdots, d,$$

where f_j is the output function, σ_j is the nonlinear activation function for each layer and the bracketed subscripts correspond to the layer in question, B_j is the bias term added to the weighted sum, $w_{i,j}$ represent the weight and x_i represent the input values (Al-Aradi et al., 2018).

The activation function mentioned above also decides whether the neuron should trigger the non-zero output value or not. There are two types of activation functions: linear activation functions and non-linear activation functions. Non-linear activation functions are the most popular used, as they are able to process data of different shapes. These type of activation functions allow deep learning models to adapt to any data type.

Non-linear activation functions are divided into several categories: Sigmoid, Tanh and ReLU (Karlsson Faronius, 2023). The main difference amongst these activation functions lies in their range and the shape of their graphs.

1. Sigmoid or Logistic activation function: The first activation function to be used in ANN. Its values exist between 0 and 1.

   Its mathematical equation is given as follows (Karlsson Faronius, 2023):

   $$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma(x) \in [0, 1]$$

   The derivative is:

   $$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2},$$

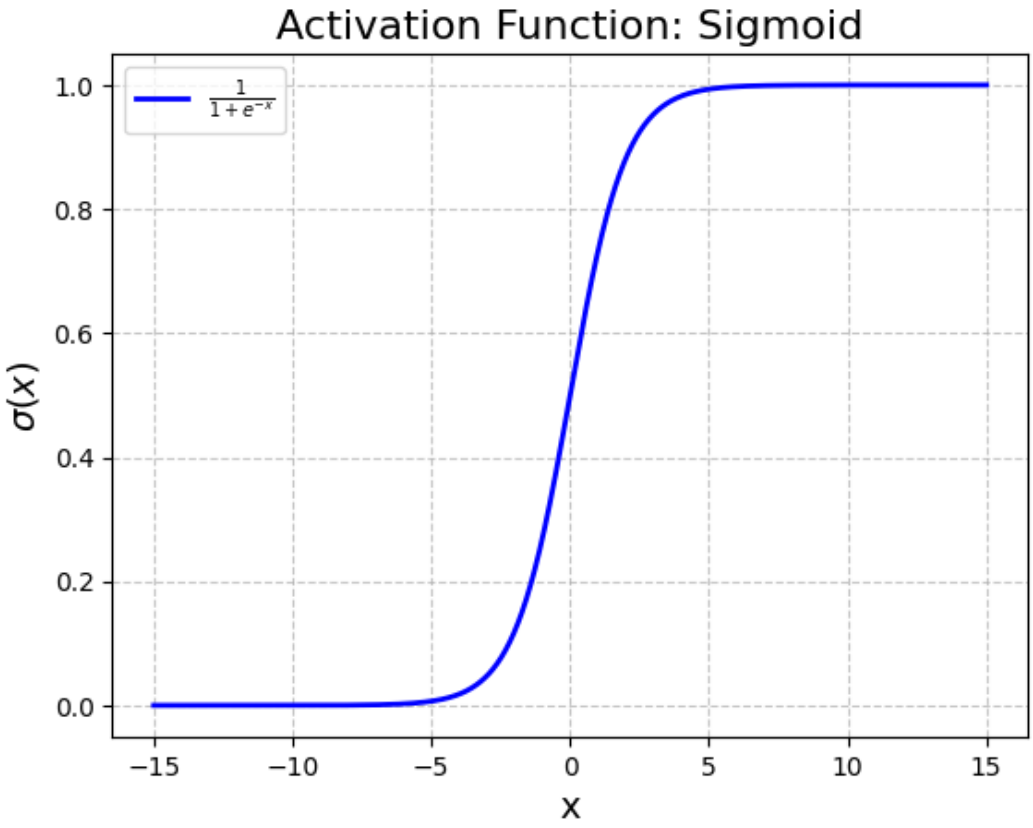


Figure 3.5: Sigmoid or Logistic activation function.

2. Tanh or hyperbolic tangent activation function: Its values exist between -1 and 1 .

   Its mathematical equation is given as follows (Karlsson Faronius, 2023):

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \sigma(x) \in [-1, 1]$$

   The derivative is:

$$\frac{d\sigma(x)}{dx} = \frac{4e^{2x}}{\left(e^{2x} + 1\right)^2}.$$

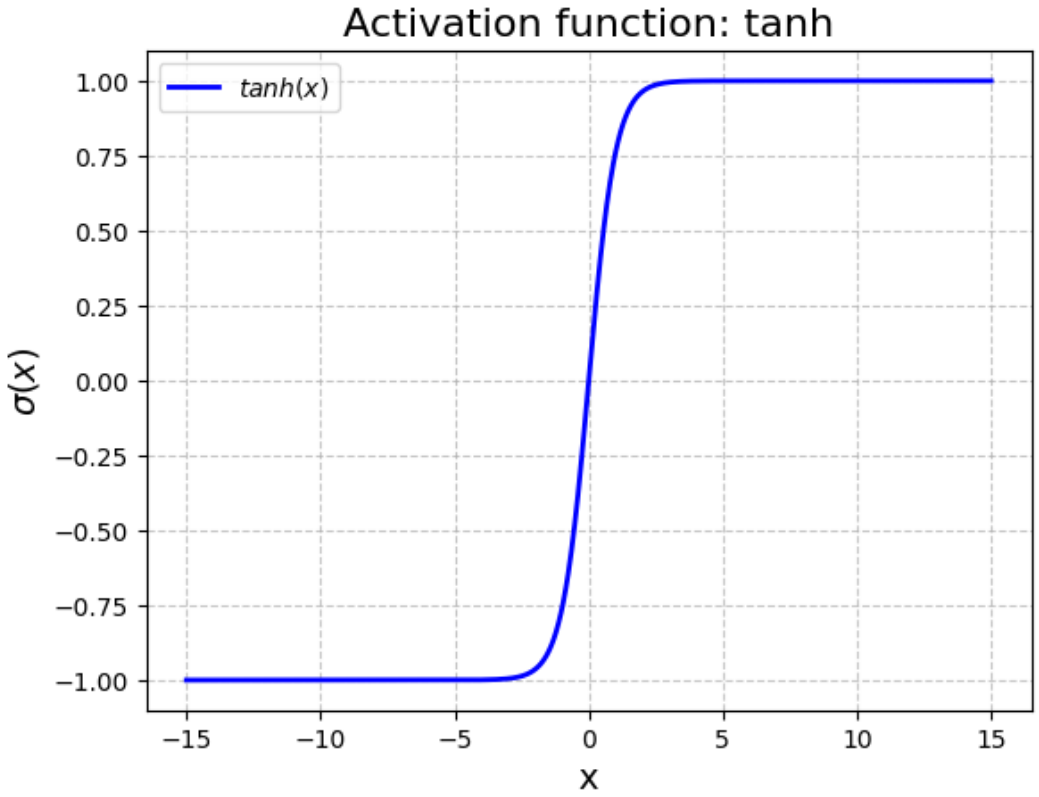


Figure 3.6: Tanh activation function.

3. ReLU (Rectified Linear Unit) activation function: It is one of the most implemented activated function in practice today. Its values exist between 0 and ∞ .

   Its mathematical equation is given as follows (Karlsson Faronius, 2023):

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise} \end{cases}$$

   The derivative is:

$$\frac{d\sigma(x)}{dx} = \begin{cases} 1, \text{if } x > 0, \\ 0, & \text{if } x < 0, \\ \text{undefined when } x = 0. \end{cases}$$

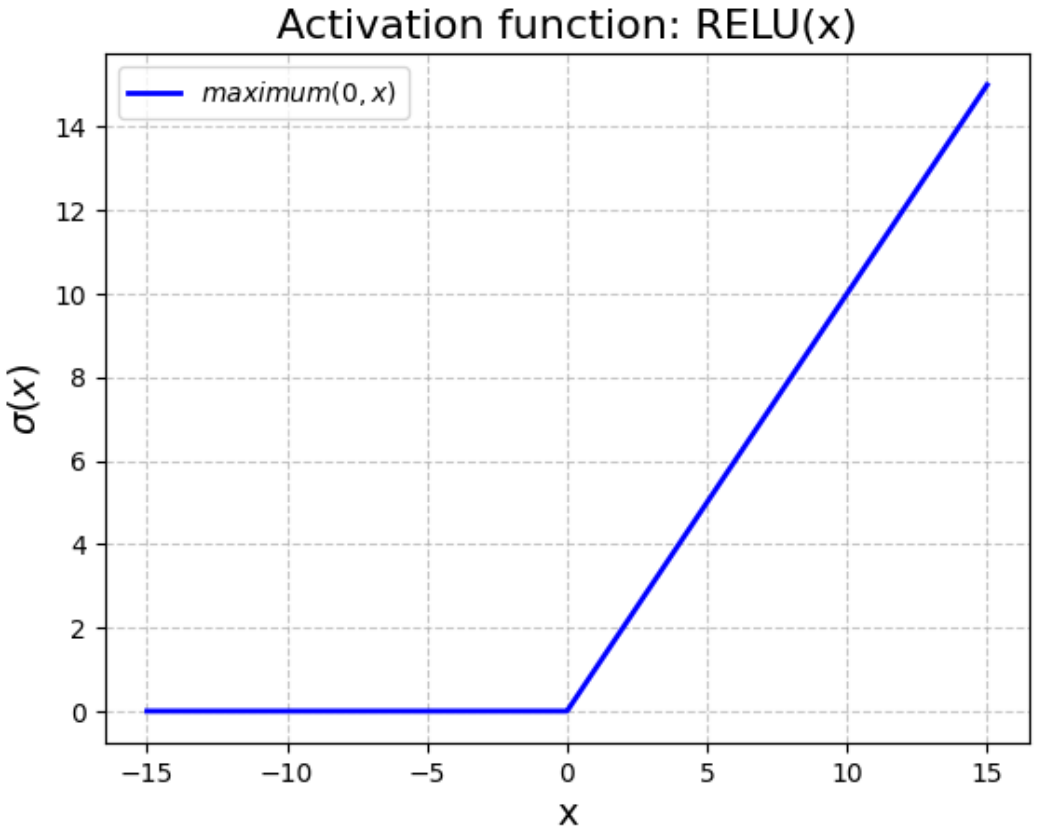


Figure 3.7: RELU activation function.

3.3.2 Deep neural network

The term deep learning comes from its complexity in the hidden layers. The complexity of the neural network increases with the number of hidden layers (Al-Aradi et al., 2018). A neural network is called deep neural network or multi-layer perception if it has more hidden layers or several layers. Figure 3.8, shows a neural network with two hidden layers. It is important to note that there are many types of neural networks, as shown in Figure 3.9.

The mathematical equation when incorporating additional hidden layer is given as follows:

$$f_k(x) = \sigma_j \left(B_k^{(2)} + \sum_{i=1}^{m2} w_{i,j} \phi_j \left(B_j^{(1)} + \sum_{i=1}^{m1} w_{i,j}^{(1)} x_i \right) \right),$$

where σ_j is the nonlinear activation function for each layer and the bracketed subscripts refer to the other previous layers, B_k bias of the new layer which is added to the weighted sum of the previous output, $w_{i,j}$ represent the weight, ϕ_j is the activation function of the previous output, $m2$ is the total number of previous hidden layers and $m1$ is the total number of current hidden layers.

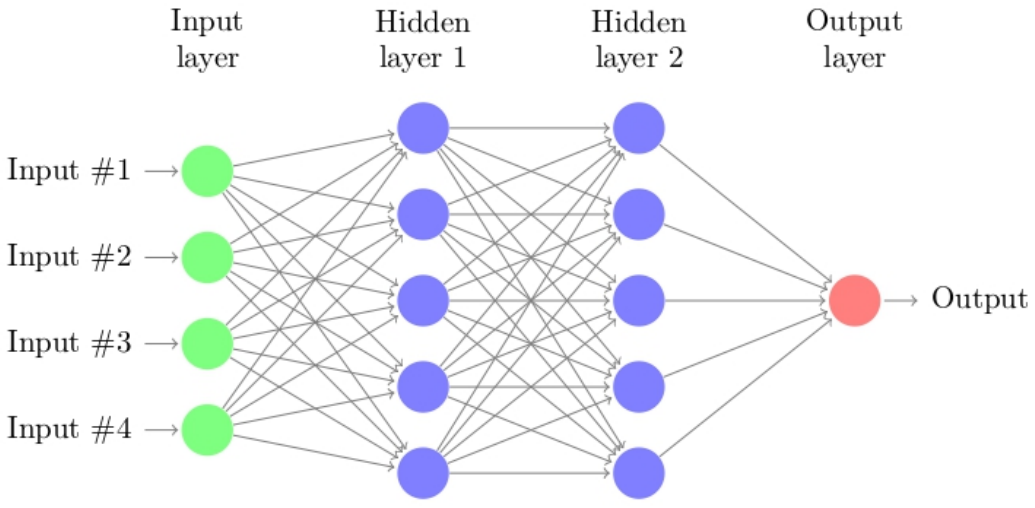


Figure 3.8: Deep Neural network.

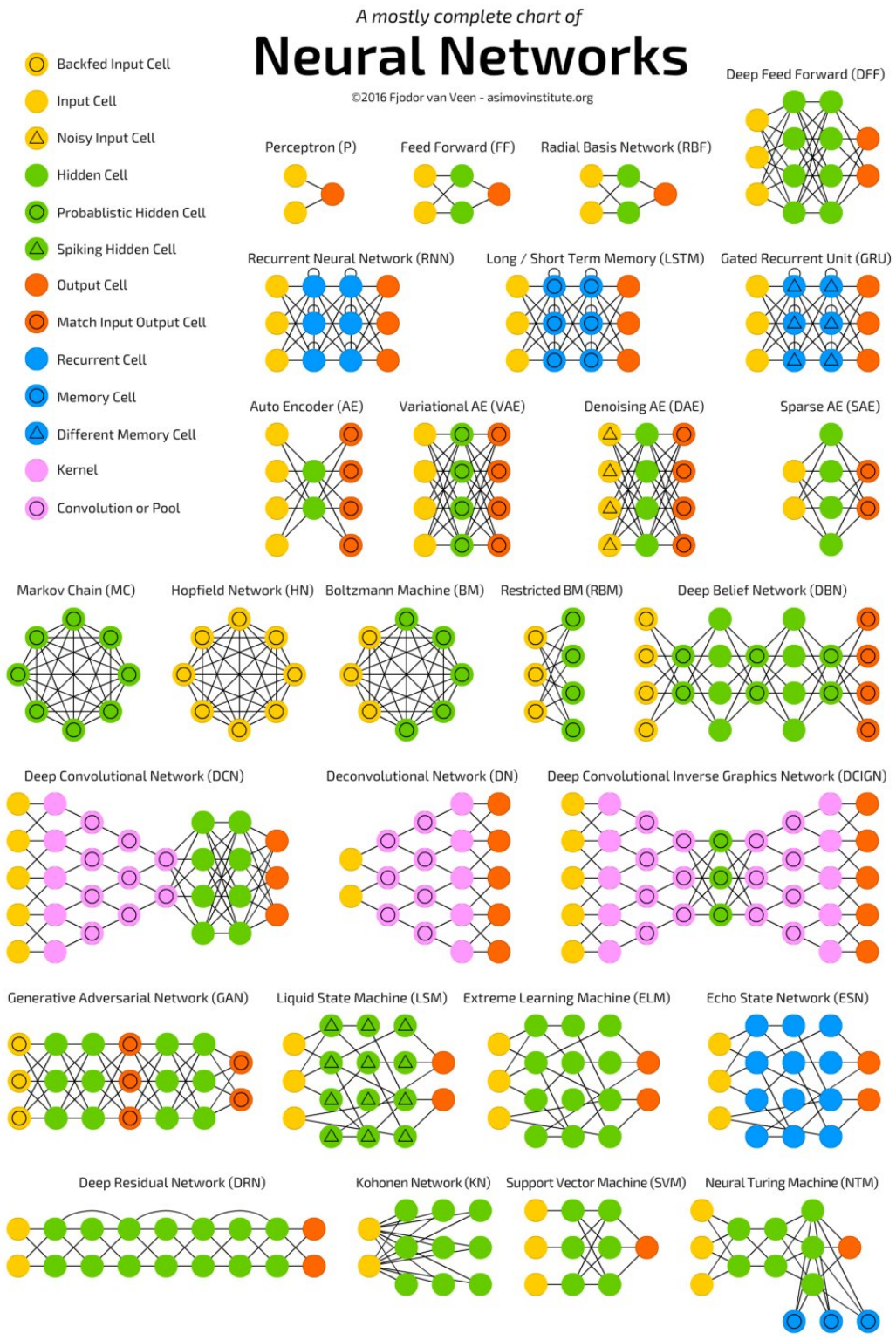


Figure 3.9: Various neural network architectures (Van Veen, 2016).

3.3.3 Stochastic gradient descent

Gradient descent

Let $L = L(\Psi; t, x)$ be the the loss function, with parameter set Ψ for the neural network consisting of the weights and bias terms in each layer.

Gradient descent is an optimization algorithm that is used to find the local minimum point of differentiable function L. It does this by taking iterative steps in the direction of the negative gradient of that function at the current location Ψ_i (Baeldung, 2023; javatpoint, 2023). The mathematical formulation of this algorithm is:

$$\Psi_{i+1} = \Psi_i - \beta * \nabla_{\Psi_{i+1}} L(\Psi_i; t, x),$$

where Ψ denotes the weights that need to be updated, β is the learning rate of the algorithm, a hyperparameter that controls how much to change the model in response to the estimated error.

This method has significant drawbacks since it is time consuming and computationally costly, as the complete data set is considered at each iteration to select the optimal solution during the training process.

Stochastic gradient descent

SGD is a slight modification of gradient descent. This approach does not take the entire dataset, but only take random samples during the training process. This saves time and is computationally more efficient when updating the parameters. However, the results are often influenced by outliers (Baeldung, 2023; javatpoint, 2023). The mathematical formulation

of this algorithm is:

$$\Psi_{i+1} = \Psi_i - \beta * \nabla_{w_i} L(\Psi_i; t^i, x_i)$$

Mini batch gradient descent

This method use a combination of gradient descent and stochastic gradient descent. The dataset is divided into smaller pieces called mini-batches, and the gradient is computed for each of these mini-batches (Baeldung, 2023; javatpoint, 2023).
The formula of Mini Batch Gradient Descent that updates the weights Ψ is:

$$\Psi_{i+1} = \Psi_i - \beta * \nabla_{\Psi_i} L(x^{i:i+b}, y^{i:i+b}; \Psi_i)$$

Backpropagation

Backpropagation is one of the most extensively used optimization techniques for training feedforward neural networks and fine-tuning neural network weights. It employs chain rule differentiation to computes the gradient of the loss function in relation to the network weights. To update the weights, backpropagation checks the error in the current layer and minimizes it by iterating backwards to the hidden layer (Baeldung, 2023; javatpoint, 2023).

4. RESEARCH METHODOLOGY

This chapter provides an in-depth exploration of the methodological content of the study. It is divided into four sections. Section 4.1 presents the mean-variance framework, section 4.2 presents a comprehensive mathematical framework for the Merton portfolio problem. While, the last two sections 4.3 & 4.4 presents the formulation and implementation of DGM and FDM.

4.1 The Mean variance-analyses

The mean-variance is a mathematical and economics framework used by investors to assess their expected returns based on the provided level of risk on different portfolios. It is one of the key cornerstones of MPT as it provides a solid theoretical foundation.

4.1.1 Duality problems

In this section two challenging problems of duality proposed by Markowitz, 1952 are presented.

Challenge I

The objective is to minimize portfolio risk based on the given level of expected return provided in the constraint(Pandi, 2020; Wojt, 2009).

The first optimization challenge reads:

$$\begin{aligned} \text{minimize: } & \mathbf{w}^T \Phi \mathbf{w} \\ \text{subject to: } & \mathbf{w}^T \mu = \mu_p \\ & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{w}_i \geq 0, \end{aligned}$$

where:

- $\mathbf{w}$ is n-column vector whose components $w_1, \ldots, w_n$ denote weight or proportion of the investor's wealth,
- $\mathbf{w}^T$ represents the transpose,
- Φ is the covariance matrix with entries $\sigma_{i,j}, i, j = 1, 2....n..$ And we assume that it is symmetry and positive definite $\mathbf{w}^T \Phi \mathbf{w} > 0$,
- $\mathbf{1}$ is n-column vector of ones, given as $\mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$,
- μ is an n-column vector of expected returns:

$$\mu_p = \sum_{i=1}^{n} w_i \mu_i,$$

- σ_p^2 is portfolio variance matrix:

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij} = \mathbf{w}^T \Phi \mathbf{w}.$$

To solve the optimization **Challenge I**, we employ the Lagrangian method:

$$L = \mathbf{w}^T \Phi \mathbf{w} + \lambda_1 \left(\mathbf{w}^T \mu - \mu_p \right) + \lambda_2 (\mathbf{w}^T \mathbf{1} - 1).$$

The first order conditions read as follows:

$$\frac{\partial L}{\partial \mathbf{w}} = 2\Phi\mathbf{w} - \lambda_1\mu - \lambda_2\mathbf{1} = 0 \tag{4.1}$$

$$\frac{\partial L}{\partial \lambda_1} = \mathbf{w}^T\mu - \mu_p = 0 \tag{4.2}$$

$$\frac{\partial L}{\partial \lambda_2} = \mathbf{w}^T\mathbf{1} - 1 = 0. \tag{4.3}$$

Solving for $\mathbf{w}$ from (4.1) yields:

$$\begin{aligned}\mathbf{w} &= \frac{1}{2}\lambda_1\Phi^{-1}\mu + \frac{1}{2}\lambda_2\Phi^{-1}\mathbf{1} \\ &= \frac{1}{2}\Phi^{-1}\left(\lambda_1\mu + \lambda_2\mathbf{1}\right)\end{aligned} \tag{4.4}$$

Plugging for $\mathbf{w}$ in Equations (4.2) and (4.3) produces:

$$\begin{cases} \frac{1}{2}\lambda_1\mu^T\Phi^{-1}\mu + \frac{1}{2}\lambda_2\mathbf{1}^T\Phi^{-1}\mathbf{1} = \mu_p \\ \frac{1}{2}\lambda_1\mathbf{1}^T\Phi^{-1}\mathbf{1} + \frac{1}{2}\lambda_2\mathbf{1}^T\Phi^{-1}\mathbf{1} = 1 \end{cases} \tag{4.5}$$

To solve for λ_1 and λ_2, let $A = \mathbf{1}^T\Phi^{-1}\mathbf{1}$, $B = \mathbf{1}^T\Phi^{-1}\mathbf{1}$ and $C = \mu^T\Phi^{-1}\mu$. Therefore, the Equations of (4.5) become:

$$\lambda_1 = \frac{2\left(A\mu_p - B\right)}{AC - B^2} \quad \text{and} \quad \lambda_2 = \frac{2\left(C - \mu_p B\right)}{AC - B^2}.$$

From our observation, we can see that λ_1 and λ_2 depends on μ_p. Since Φ is definite positive matrix so is Φ^{-1} and $A, C > 0$ and $AC - B^2 > 0$ now:

$$\sigma_p^2 = \mathbf{w}^T\Phi\mathbf{w} = \frac{A\mu_p^2 - 2B\mu_p + C}{AC - B^2}. \tag{4.6}$$

A parabolic shape is obtained when we vary the values of μ_p:

$$\frac{\partial \sigma_p^2}{\partial \mu_p} = \frac{2A\mu_p - 2B}{AC - B^2}. \tag{4.7}$$

The global minimum variance can be found by setting $\frac{\partial \sigma_p^2}{\partial \mu_p}$ to zero as follows:

$$\frac{2A\mu_p - 2B}{AC - B^2} = 0.$$

Therefore, the portfolio with minimum risk and global minimum vector of weight are given, respectively:

$$\begin{cases} \left(\mu_p^* = \frac{B}{A}, \sigma_p^{2*} = \frac{1}{A}\right) \\ \mathbf{w}_g = \frac{\Phi^{-1}\mathbf{1}}{\mathbf{1}^T\Phi^{-1}\mathbf{1}}. \end{cases}$$

Challenge II

The objective is to maximize the portfolio expected return for any given level of risk provided in the constraints (Pandi, 2020; Wojt, 2009)

The second optimization challenge reads:

$$\text{maximize: } \mathbf{w}^T\mu \tag{4.8}$$

$$\text{subject to: } \mathbf{w}^T\Phi\mathbf{w} = \sigma^2 \tag{4.9}$$

$$\mathbf{w}^T\mathbf{1} = 1. \tag{4.10}$$

To solve the optimization **Challenge II**, employ the Lagrangian method:

$$L = \mathbf{w}^T\mu + \lambda_1\left(\sigma^2 - \mathbf{w}^T\Phi\mathbf{w}\right) + \lambda_2\left(1 - \mathbf{w}^T\mathbf{1}\right).$$

The first order condition read as follows:

$$\frac{\partial L}{\partial \mathbf{w}} = \mu - 2\lambda_1\Phi\mathbf{w} - \lambda_2\mathbf{1} \tag{4.11}$$

Solving for $\mathbf{w}$ from Equation (4.11) produces:

$$\begin{aligned}\mathbf{w}^* &= \Phi^{-1}(\frac{1}{2\lambda_1}\mu - \frac{\lambda_2}{2\lambda_1}\mathbf{1}) \\ &= \Phi^{-1}\left(\frac{\mu - \lambda_2\mathbf{1}}{2\lambda_1}\right)\end{aligned}$$

The constraint of the optimization **Challenge II** is utilised to get the solutions of λ_1 and λ_2:

$$\begin{aligned}&\mathbf{w}^{*T}\mathbf{1} = 1 \\ \implies &\left[\Phi^{-1}\left(\frac{\mu - \lambda_2\mathbf{1}}{2\lambda_1}\right)\right]^T \mathbf{1} = 1 \\ \implies &(\mu - \lambda_2\mathbf{1})^T \left[\frac{\Phi^{-1}}{2\lambda_1}\right]^T \mathbf{1} = 1 \\ \implies &\left(\mu^T - \lambda_2\mathbf{1}^T\right)\left[\frac{\Phi^{-1}}{2\lambda_1}\right]\mathbf{1} = 1 \\ \implies &\frac{1}{2\lambda_1}\left(\mu^T\Phi^{-1}\mathbf{1} - \lambda_2\mathbf{1}^T\Phi^{-1}\mathbf{1}\right) = 1 \\ \implies &\frac{1}{2\lambda_1}(B - \lambda_2 A) = 1. \end{aligned} \tag{4.12}$$

where the values of A, B and C are still the same as in duality **Challenge I**.

Solving for λ_2 from (4.12) results in:

$$\lambda_2 = \frac{1}{A}(B - 2\lambda_1). \tag{4.13}$$

The expression for λ_1 is computed by employing the constraint provided in (4.9):

$$\begin{aligned}&\mathbf{w}^T\Phi\mathbf{w} = \sigma^2 \\ \implies &\left[\Phi^{-1}\left(\frac{\mu - \lambda_2\mathbf{1}}{2\lambda_1}\right)\right]^T \Phi\frac{\Phi^{-1}}{2\lambda_1}(\mu - \lambda_2\mathbf{1}) = \sigma^2 \\ \implies &\frac{(\mu - \lambda_2\mathbf{1})}{2\lambda_1}\left[\frac{\Phi^{-1}}{2\lambda_1}\right]^T (\mu - \lambda_2\mathbf{1}) = \sigma^2 \\ \implies &\left(\mu^T - \lambda_2\mathbf{1}^T\right)\frac{\Phi^{-1}}{4\lambda_1^2}(\mu - \lambda_2\mathbf{1}) = \sigma^2\end{aligned}$$

$$
\begin{aligned}
&\implies \frac{1}{4\lambda_1^2}\left(\mu^T\Phi^{-1}\mu - \lambda_2\mu^T\Phi^{-1}\mathbf{1} - \lambda_2\mathbf{1}^T\Phi^{-1}\mu + \lambda_2^2\mathbf{1}^T\Phi^{-1}\mathbf{1}\right) = \sigma^2 \\
&\implies \frac{1}{4\sigma^2}\left[C - 2B\left(\frac{1}{A}\left(B - 2\lambda_1\right)\right) + \left(\frac{1}{A}\left(B - 2\lambda_1\right)\right)^2 A\right] = \lambda_1^2 \\
&\implies \frac{1}{4\sigma^2}\left[C - \left(\frac{2B^2 - 4B\lambda_1}{A}\right) + \left(\frac{B^2 - 4B\lambda_1 + 4\lambda_1^2}{A}\right)\right] = \lambda_1^2 \\
&\implies \frac{1}{4\sigma^2}\left[C - \frac{B^2 + 4\lambda_1^2}{A}\right] = \lambda_1^2 \\
&\implies \lambda_1 = \sqrt{\frac{AC - B^2}{4(\sigma^2 A - 1)}}.
\end{aligned}
$$

Substituting λ_1 into Equation (4.13) produces:

$$
\begin{aligned}
\lambda_2 &= \frac{1}{A}(B - 2\sqrt{\frac{AC - B^2}{4(\sigma^2 A - 1)}}) \\
&= \frac{1}{A}\left[B - \sqrt{\frac{AC - B^2}{\sigma^2 A - 1}}\right],
\end{aligned}
$$

with A, B, and C constants given as in **Challenge I**. Finally substituting λ_1 and λ_2 into $\mathbf{w}$:

$$
\begin{aligned}
\mathbf{w}^* &= \Phi^{-1}\left[\frac{\mu - \frac{1}{A}\left(B - \sqrt{\frac{AC-B^2}{\sigma^2 A-1}}\right)\mathbf{1}}{2\sqrt{\frac{AC-B^2}{4(\sigma^2 A-1)}}}\right] \\
&= \Phi^{-1}\left[\mu\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}} - \frac{B}{A}\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}}\mathbf{1} + \frac{1}{A}\sqrt{\frac{AC - B^2}{\sigma^2 A - 1}}\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}}\right] \\
&= \Phi^{-1}\left[\mu\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}} + \frac{1}{A}\left(1 - B\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}}\mathbf{1}\right)\right].
\end{aligned}
$$

From Equation (4.4) of duality **Challenge I** we had:

$$
\mathbf{w}^* = \frac{1}{2}\Phi^{-1}\left(f_1\mu + f_2\mathbf{1}\right),
$$

from this the following are established:

$$f_1 = 2\left(\frac{A\mu_p - B}{AC - B^2}\right)$$

and

$$f_2 = 2\left(\frac{C - B\mu_p}{AC - B^2}\right).$$

Taking one of the two, the following result is obtained:

$$\begin{aligned}
\sqrt{\frac{\sigma^2 A - 1}{AC - B^2}} &= 2\left(\frac{A\mu_p - B}{AC - B^2}\right) \\
\implies \frac{\sigma^2 A - 1}{AC - B^2} &= \left(\frac{A\mu_p - B}{AC - B^2}\right)^2 \\
\implies \sigma_p^2 &= \frac{A^2\mu_p^2 - 2B\mu_p + B^2}{AC - B^2}.
\end{aligned}$$

The solution for σ_p^2 looks exactly the same as that of duality **Challenge I**.

4.1.2 The efficient frontier

It is a curve that represents a set of optimal portfolios that are expected to provide sufficient expected returns for any given level of risk. It is based on the concept of diversification. The idea here is to minimize risk while attaining the highest returns on the portfolio.

It is important to note that the portfolios above the curved line, as shown in Figure 4.1, are considered infeasible and impossible to achieve. Whereas, the portfolios below the curved line are considered suboptimal.

The optimal portfolio return typically resides on the efficient frontier. Risk-averse investors tend to favor the lower left-hand side of the efficient frontier, which offers lower risk and modest returns. On the other hand, investors who seek higher returns and are willing to accept higher risk would be inclined towards the upper right-hand side of the efficient

frontier.

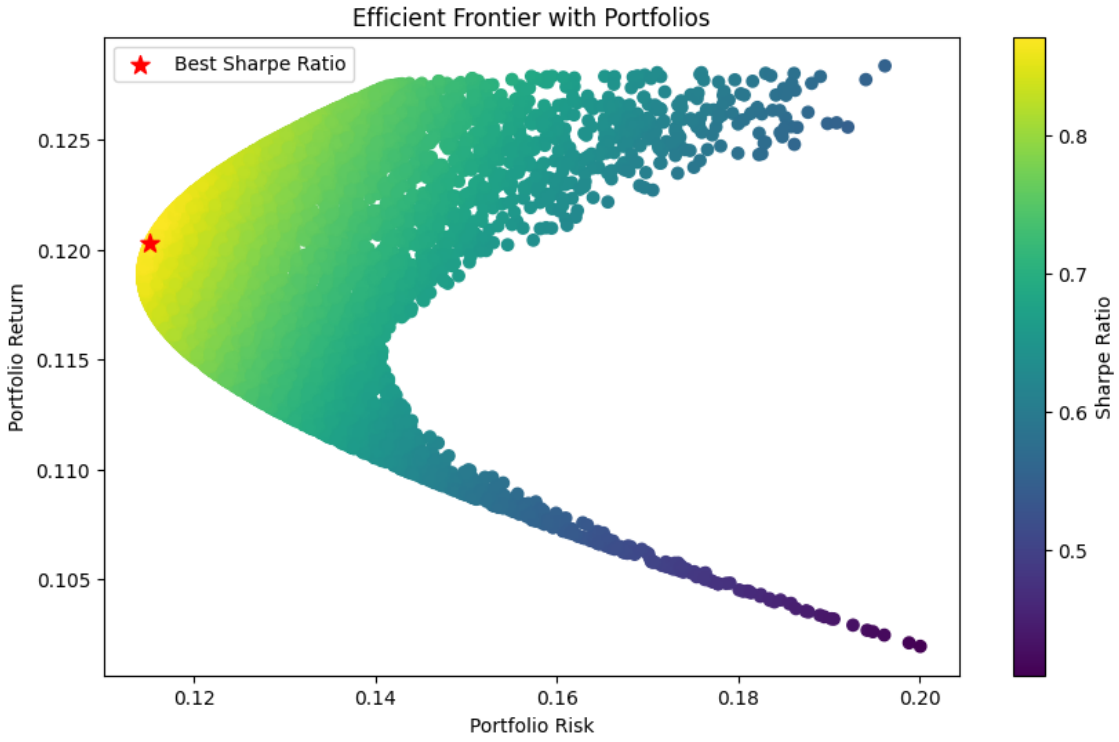


Figure 4.1: Efficient frontier.

4.1.3 One fund separation theorem

The previous duality problems did not include the risk free asset. However, in this section we will incorporate the risk-free asset, which has a risk-free rate denoted as r_f.

The mean-variance model, with the inclusion of a risk-free asset, can now be formulated as follows (Pandi, 2020):

$$\begin{aligned} &\text{minimize: } \mathbf{w}^T \Phi \mathbf{w} \\ &\text{subject to: } \mathbf{w}^T \mu + (1 - \mathbf{w}^T \mathbf{1}) r_f = \mu_p. \end{aligned}$$

To solve the optimization problem we employ the Lagrangian method:

$$\begin{aligned} L &= \mathbf{w}^T \Phi \mathbf{w} + \lambda \left(\mathbf{w}^T \mu + (1 - \mathbf{w}^T \mathbf{1}) r_f - \mu_p \right) \\ &= \mathbf{w}^T \Phi \mathbf{w} + \lambda \left(\mathbf{w}^T (\mu - r_f \mathbf{1}) + r_f - \mu_p \right) \end{aligned}$$

The first order derivatives are given as follows:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w}\Phi + \lambda \left(\mu - r_f \mathbf{1} \right) = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} = \mathbf{w}^T (\mu - r_f \mathbf{1}) + r_f - \mu_p = 0. \end{cases} \tag{4.14}$$

Solving for $\mathbf{w}^*$ from the first part of the above Equation (4.14):

$$w^* = \lambda \Phi^{-1} \left(\mu - r_f \mathbf{1} \right).$$

Inserting this into the second part of Equation (4.14) gives:

$$\begin{aligned} & \lambda \Phi^{-1} \left(\mu - r_f \mathbf{1} \right)^T \left(\mu - r_f \mathbf{1} \right) + r_f - \mu_p = 0 \\ \implies & \lambda \Phi^{-1} \left(\mu - r_f \mathbf{1} \right)^T \left(\mu - r_f \mathbf{1} \right) = \mu_p - r_f \\ \implies & \lambda \left(C - 2 r_f B + r_f^2 A \right) = \mu_p - r_f. \end{aligned} \tag{4.15}$$

Solving for λ from Equation (4.15) we have:

$$\lambda = \frac{\mu_p - r_f}{\left(C - 2 r_f B + r_f^2 A \right)}.$$

The portfolios risk expression is given as follows:

$$\begin{aligned} \sigma^2 &= \mathbf{w}^T \Phi \mathbf{w} \\ &= \lambda \left(\mathbf{w}^{T*} \mu - r_f \mathbf{w}^{T*} \right) \\ &= \lambda \left(\mu_p - r_f \right), \end{aligned}$$

substitution for λ into the above Equation, the variance expression yields:

$$\sigma^2 = \frac{\left(\mu_p - r_f\right)^2}{\left(C - 2r_f B + r_f^2 A\right)},$$

provided that:

$$\mu_p = r_f \pm \sigma_p \sqrt{C - 2r_f B + r_f^2 A}.$$

From the Equations (4.6) and (4.7) of the duality **Challenge I**, we observed that the set of minimum variance lies on a parabola. However, with the inclusion of the risk-free rate by the investors, the minimum variance will now lie on the two lines given below (Pandi, 2020):

$$\begin{aligned}\text{Upper-line } &= r_f + \sigma_p \sqrt{C - 2r_f B + r_f^2 A},\\ \text{Lower-line } &= r_f - \sigma_p \sqrt{C - 2r_f B + r_f^2 A}.\end{aligned}$$

Because investors are inclined towards the upper right-hand side of the efficient frontier, the the upper tangent line to the hyperbola will offer an optimal portfolio choice through the point $(0, r_f)$.

Thus solving the following equations at the same time:

$$\begin{aligned}\sigma^2 &= \frac{\left(\mu_p - r_f\right)^2}{\left(C - 2r_f B + r_f^2 A\right)}\\ \mu_p &= r_f + \sigma_p \sqrt{C - 2r_f B + r_f^2 A}.\end{aligned}$$

The coordinate $\left(\sigma^E, \mu^E\right)$ of the tangency portfolio E yields:

$$\begin{aligned}\sigma_p^{2E} &= \frac{C - 2r_f B + a r_f^2}{\left(B - r_f A\right)^2},\\ \mu_p^E &= \frac{C - r_f B}{B - r_f A}.\end{aligned}$$

Furthermore, the tangency optimal vector of weights yields:

$$\mathbf{w}_F^* = \frac{\Phi^{-1}\left(\mu - r_f \mathbf{1}\right)}{B - A r_f}.$$

4.2 Merton portfolio optimization problem

Problem formulation

The Merton portfolio optimization problem is one of the well-known problems in continuous-time finance, pioneered by the famous economist Robert C. Merton (1969). The main purpose of the problem is to establish the best investment strategy for maximizing the expected utility for the initial wealth (w_0), within a specified time interval [0, T]. At time t, investors need to decide how to consume ($C_t \geq 0$) their wealth by distributing and investing in a risky portfolio (π_t) or a risk-free portfolio ($1 - \pi_t$). The objective is to obtain (Merton, 1969):

$$V(t, w_t) = \max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[U\left(W_t^{\pi}\right) | W_t^{\pi} = w_t\right] \tag{4.16}$$

The price of risky asset S_t, is given by the stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma S_t dB_t \tag{4.17}$$

The stochastic differential equation for the wealth (w_t) by the stochastic differential equation:

$$dw_t = \left[\pi_t \mu + (1 - \pi_t) r\right] w_t dt + \pi_t w_t \sigma dB_t \tag{4.18}$$

where B_t is the Brownian motion, μ is the expected return and σ is the volatility of risky asset and r is the risk free-interest rate.

For this study the exponential utility function is used:

$$U(w_t) = -e^{-pw_t}.$$

The above exponential utility was chosen because of its simple mathematical representation, easy-to-interpret parameters, falling under constant absolute risk aversion (CARA) utility, exhibiting risk aversion, and also because its properties are consistent with observed market pricing.

Solving the Merton problem

4.2.1 HJB equation for the Merton problem

Consider the objective function:

$$V = \max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[-e^{-\beta T} U(W_T)\right],$$

where β is the subjective discount rate. For the partition $0 \leq t < t_1 < T$, V satisfies the following:

$$\begin{aligned} V &= \max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[e^{-\beta(t_1 - t)} V(t_1, w_{t_1})\right] \\ \implies e^{-\beta t} V &= \max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[e^{-\beta t_1} V(t_1, w_{t_1})\right]. \end{aligned}$$

Setting $\beta = 0$ and transforming the above expression into stochastic differential equation the following results is obtained:

$$\max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[dV(t, w_t)\right] = 0.$$

Applying the Itôs'lemma to V and setting the expectation of the dB_t term to zero since it is a martingale gives:

$$\begin{aligned}
&\max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[\left(\frac{\partial V}{\partial t} + \left[\pi_t \mu + (1-\pi_t)\, r\right] w_t \frac{\partial V}{\partial w_t} + \frac{\pi_t^2 \sigma^2 w_t^2}{2}\frac{\partial^2 V}{\partial w_t^2}\right)dt + \pi_t \sigma_t \frac{\partial V}{\partial w_t} dB_t\right] = 0 \\
\implies &\max_{\pi_t \in \mathcal{A}_t} \mathbb{E}\left[\left(\frac{\partial V}{\partial t} + \left[\pi_t \mu + (1-\pi_t)\, r\right] w_t \frac{\partial V}{\partial w_t} + \frac{\pi_t^2 \sigma^2 w_t^2}{2}\frac{\partial^2 V}{\partial w_t^2}\right)dt\right] = 0
\end{aligned}$$

Dividing both side by dt, the HJB equation for the Merton problem is obtained:

$$\frac{\partial V}{\partial t} + \max_{\pi_t \in \mathcal{A}_t} \left\{ \left[\pi_t \mu + (1-\pi_t)\, r\right] w_t \frac{\partial V}{\partial w_t} + \frac{\pi_t^2 \sigma^2 w_t^2}{2}\frac{\partial^2 V}{\partial w_t^2} \right\} = 0 \tag{4.19}$$

with boundary condition:$V(T, w) = \epsilon^p e^{-p w_t}$.

4.2.2 Optimal control (wealth) and optimal value function PDE

The first order condition of Equation (4.19) with respect to π_t is given as follows:

$$\begin{aligned}
&(\mu - r)\, w_t \frac{\partial V}{\partial w_t} + \frac{2\pi_t \sigma^2 w_t^2}{2}\frac{\partial^2 V}{\partial w_t^2} = 0 \\
\implies &\pi_t \sigma^2 w_t^2 \frac{\partial^2 V}{\partial w_t^2} = -(\mu - r)\, w_t \frac{\partial V}{\partial w_t} \\
\implies &\pi_t = -\frac{(\mu - r)}{\sigma^2 w_t}\frac{\partial V}{\partial w_t}\left(\frac{\partial^2 V}{\partial w_t^2}\right)^{-1}.
\end{aligned}$$

Inserting π_t into HJB equation (4.19) gives:

$$\begin{aligned}
&\frac{\partial V}{\partial t} - \frac{(\mu - r)^2 w_t}{\sigma^2 w_t}\left(\frac{\partial V}{\partial w_t}\right)^2\left(\frac{\partial^2 V}{\partial w_t^2}\right)^{-1} + r w_t \frac{\partial V}{\partial w_t} + \left(\frac{(\mu - r)}{\sigma^2 w_t}\frac{\partial V}{\partial w_t}\left(\frac{\partial^2 V}{\partial w_t^2}\right)^{-1}\right)^2 \frac{\sigma^2 w_t^2}{2}\frac{\partial^2 V}{\partial w_t^2} = 0 \\
&\implies \frac{\partial V}{\partial t} + r w_t \frac{\partial V}{\partial w_t} - \frac{(\mu - r)^2}{2\sigma^2}\frac{\left(\frac{\partial V}{\partial w_t}\right)^2}{\frac{\partial^2 V}{\partial w_t^2}} = 0.
\end{aligned} \tag{4.20}$$

The boundary condition is: $V(T, w_t) = -\epsilon^p e^{-p w_t}$.

4.2.3 Optimal value function

To evaluate the PDE (4.20), the solution is assumed to be of this form:

$$V(t, w_t) = -g(t)e^{-pw_t Z(t)}. \tag{4.21}$$

Now,

$$\begin{aligned}
\frac{\partial V}{\partial t} &= -\left[g'(t) + w_t p g(t) Z'(t)\right] e^{-pw_t Z(t)} \\
\frac{\partial V}{\partial w_t} &= g(t) p Z(t) e^{-pw_t Z(t)}, \\
\frac{\partial^2 V}{\partial w_t^2} &= -p^2 Z^2(t) g(t) e^{-pw_t Z(t)}.
\end{aligned}$$

and,

$$\pi_t^* = -\frac{\mu - r}{\sigma^2 w_t p}.$$

Substituting into the value function PDE:

$$-\left[g'(t) + w_t p g(t) Z'(t)\right] e^{-pw_t Z(t)} + r w_t g(t) p Z(t) e^{-pw_t Z(t)} - \frac{(\mu - r)^2}{2\sigma^2} \frac{\left(g(t) p Z(t) e^{-pw_t Z(t)}\right)^2}{-p^2 Z^2(t) g(t) e^{-pw_t Z(t)}} = 0$$

Rearranging the expressions and extracting common factors yields:

$$\begin{aligned}
& g'(t) + w_t p g(t) Z'(t) - r w_t g(t) p Z(t) - \frac{(\mu - r)^2}{2\sigma^2} g(t) = 0 \\
\implies & g'(t) - \frac{(\mu - r)^2}{2\sigma^2} g(t) + \left(Z'(t) - r Z(t)\right) w_t p g(t) = 0
\end{aligned}$$

We can write the above term in this way:

$$\left(\frac{dg(t)}{dt} - \frac{(\mu - r)^2}{2\sigma^2} g(t)\right) + \left(\frac{dZ(t)}{dt} - r Z(t)\right) w_t p g(t) = 0.$$

Since this must hold for every w_t and t, the terms in bracket must be equated to zero.

First term

The method of separation of variables gives:

$$\frac{dg(t)}{dt} = \frac{(\mu - r)^2}{2\sigma^2} g(t).$$

Integrating both side and simplifying leads to:

$$\ln(g(t)) = Qdt + c$$
$$\implies g(t) = g_0 e^{Qt},$$

where $Q(t) = \frac{(\mu - r)^2}{2\sigma^2}$.

Using the terminal condition $g(T) = 1$:

$$g_0 = e^{-QT} \implies g(t) = e^{-Q(T-t)}. \tag{4.22}$$

Second term

The method of separation of variables gives:

$$\frac{dZ(t)}{dt} = rZ(t).$$

Integrating both side and simplifying leads to:

$$\ln(Z(t)) = rdt$$
$$\implies Z(t) = Z_0 e^{r}.$$

Using the terminal condition $Z(T) = 1$:

$$Z_0 = e^{-rT} \implies Z(t) = e^{-r(T-t)}. \tag{4.23}$$

Thus, the optimal value function is obtained:

$$\begin{aligned} V(t, w_t) &= -g(t)e^{-pw_t Z(t)} \\ &= -e^{-Q(T-t)} \exp\left\{-pw_t e^{-r(T-t)}\right\} \\ &= -\exp\left\{-\frac{(\mu - r)^2}{2\sigma^2}(T-t) - pw_t e^{-r(T-t)}\right\} \end{aligned} \tag{4.24}$$

4.3 Deep Garlekin method

Let $u = u(t, w_t)$ be an unknown function which satisfies the non-linear PDE (Gim and Park (2021)):

$$\begin{cases} \frac{\partial u}{\partial t}(t, w_t) + \mathcal{L}u(t, w_t) = 0, & (t, w_t) \in [0, T] \times D \\ u(0, w_t) = u_0(w_t), & w_t \in D \quad \text{(Initial condition)} \\ u(t, w_t) = f(t, w_t), & (t, w_t) \in [0, T] \times \partial D \quad \text{(Boundary condition)} \end{cases}$$

where $D \in \mathbb{R}^d$. Now, the focus is to express the solution of (4.25) as a neural network function $g = g(t, w_t; \Psi)$ in replacement of u, where $\Psi = (\Psi^1, \Psi^2, .., ..., \Psi^\alpha)$ denotes a vector of network parameters.

Define a loss function as:

$$\begin{aligned} L(g) = \left\| \frac{\partial g}{\partial t}(t, w_t; \Psi) + \mathcal{L}g(t, w_t; \Psi) \right\|^2_{[0,T]\times D, \nu_1} &+ \|g(T, w_t; \Psi) - f(t, w_t)\|^2_{[0,T]\times D, \nu_2} \\ &+ \|g(T, w_t; \Psi) - V_0(w_t)\|^2_{[0,T]\times D, \nu_3} \end{aligned}$$

The loss function is composed of three portions given as follows:

- The first portion determines how accurate the neural network g approximates the PDE

differential operator:

$$\left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) g(t, w_t; \Psi) \right\|^2_{[0,T]\times D, v_1}$$

- The second portion determines how accurate the neural network g approximates the PDE boundary condition:

$$\left\| g(T, w_t; \Psi) - f(t, w_t) \right\|^2_{[0,T]\times D, v_2}$$

- The third portion determines how accurate the neural network g approximates the PDE initial condition:

$$\left\| g(0, w_t; \Psi) - V_0(w_t) \right\|^2_{[0,T]\times D, v_3}$$

The above portions are expressed in terms of $\mathbf{L}^2$ norm. The purpose is to minimize $L(g)$ while obtaining the value of the parameter Ψ:

$$\Psi = \arg\min_{\Psi} L(g(t, w_t; \Psi)) \tag{4.25}$$

As the error $L(g)$ decreases, the approximating function g would get closer to the solution u. An appropriate explanation and demonstration of the DGM algorithm is presented on the next page.

Algorithm 1: Deep Galerkin Method algorithm (Al-Aradi et al., 2018; Gim & Park, 2021; Sirignano & Spiliopoulos, 2018)

Input: Sample random points $S_n = (t_n; w_n)$, set learning rate β_n, set Maximum iterations N.(*input*)

Output: Ψ_{n+1} (*Initialize the parameter* Ψ)

while *iterations* $\leq N$ **do**

read current;

$$L(g) = \left(\frac{\partial g}{\partial t}(t, w_t; \Psi) + \mathcal{L}g(t, w_t; \Psi)\right)^2_{[0,T]\times D, v_1} + (g(T, w_t; \Psi) - f(t, w_t))^2_{[0,T]\times D, v_2} + (g(T, w_t; \Psi) - V_0(w_t))^2_{[0,T]\times D, v_3}$$

and

$$\Psi_{n+1} = \Psi_n - \beta_n \nabla_\Psi \mathbf{L}(\Psi, s_n)$$

; /* Use the stochastic gradient descent at s_n: */

if $\lim_{n\to\infty} \left\|\Psi_{n+1} - \Psi_n\right\| = 0$ **then**

return the parameters Ψ_{n+1};

else

go back to the beginning of current while loop ; /* Repeat until $\left\|\Psi_{n+1} - \Psi_n\right\|$ is small enough. */

end

end

4.3.1 The DGM architecture

In this section, we provide the deep learning architecture adopted by Sirignano and Spiliopoulos, 2018. This architecture consists of three layers, which we refer to as DGM layers: an input layer that accepts as input randomly sampled w, a hidden layer that performs some complex mathematical operations, and an output layer that passes out the value y to other DGM nodes. Although, depending on the task, this framework could be easily extended to allow for additional hidden layers. Figure 4.2 shows a visual representation of what the DGM looks like from afar.

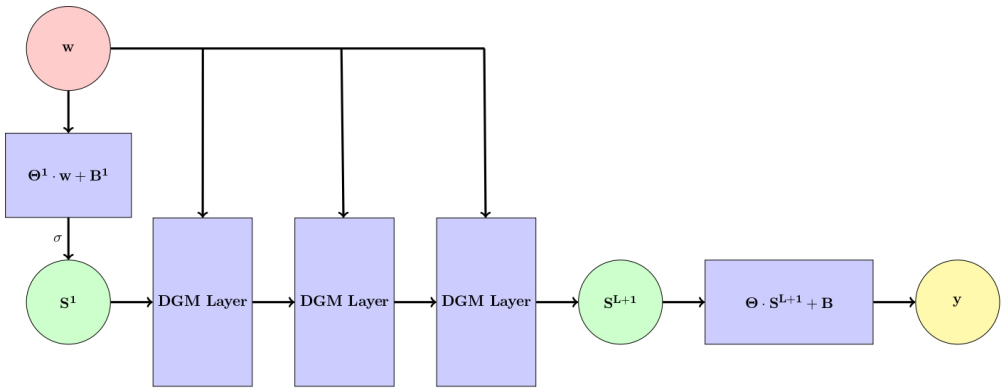


Figure 4.2: DGM from an eagles' eye perspective (Al-Aradi et al., 2018).

Equations (4.26) provide a mathematical representation of the DGM architecture, which was found to be effective by Sirignano and Spiliopoulos, 2018. This architecture closely resembles that of the architecture for Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) and highway networks (Srivastava et al., 2015). The architecture is considered complex because of the series of mathematical operations happening within each layer. In addition, the DGM architecture is composed of repeated element-wise multiplication features, enabling it to adapt easily to the nonlinear functions of the input, which are rapidly changing in certain time and space regions.

$$
\begin{cases}
S^1 = \sigma\left(\Theta^1 \cdot w + B^1\right) & \\
Z^l = \sigma\left(u^{z,l} \cdot w + \Theta^{z,l} \cdot S^l + B^{z,l}\right), & l = 1, \ldots, L \\
G^l = \sigma\left(u^{g,l} \cdot w + \Theta^{g,l} \cdot S^l + B^{g,l}\right), & l = 1, \ldots, L \\
R^l = \sigma\left(u^{r,l} \cdot w + \Theta^{r,l} \cdot S^l + B^{r,l}\right), & l = 1, \ldots, L \\
H^l = \sigma\left(u^{h,l} \cdot w + \Theta^{h,l} \cdot \left(S^l \odot R^l\right) + B^{h,l}\right), & l = 1, \ldots, L \\
S^{l+1} = \left(1 - G^l\right) \odot H^l + z^l \odot S^l, & l = 1, \ldots, L \\
f(t, w; \Psi) = \Theta \cdot S^{L+1} + B, &
\end{cases}
\tag{4.26}
$$

where $\odot$ denotes Hadamard (element-wise) multiplication, L number of layers, σ activation function, w are inputs, B is the bias term, u are model parameters and Θ are the weights.

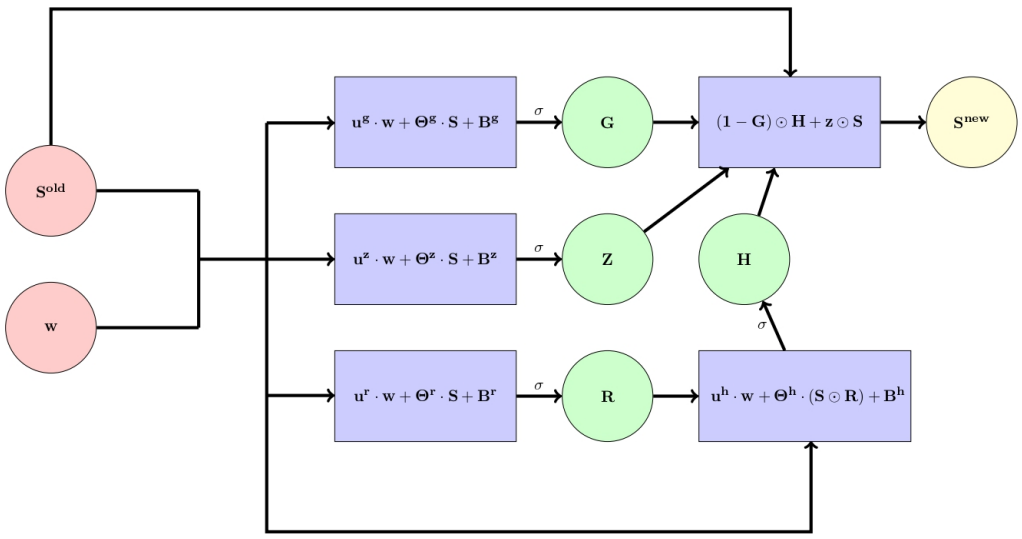


Figure 4.3: Operations within a single DGM layer (Al-Aradi et al., 2018).

4.3.2 Implementation details

For coding purposes Python was chosen as programming language. The integrated development environment (IDE) that was used is Jupyter notebook found in the Google

Collaboration. The reason for using Google Colab IDE is because it is well updated and has many features that are capable of running deep learning algorithms.

For implementing the DGM algorithm in Python a library called TensorFlow is employed. TensorFlow is an open-source library developed by Google used mostly for deep learning, machine learning and artificial intelligent. The automatic differentiation in reverse mode feature of TensorFlow enables the computation of derivatives for a wide variety of functions. TensorFlow, for instance, can be utilized to determine the gradient of the neural network (4.26) with respect to the network parameters.

The following line of code was used to import the TensorFlow package into Python.

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
```

Since most of the package are now updated to version 2,this line of code was run to allow working in version 1.

```
tf.compat.v1.disable_eager_execution()
```

Another important library is Numpy, a python library used for numerical computation. This library is usefull when dealing with random samples.

```
import numpy as np
```

For visualization purposes the matplotlib library was utilised.

```
import matplotlib.pyplot as plt
```

Below is an illustration of the short code implemented in Python to show how the algorithm works (Al-Aradi, 2018).

```
# Initialize parameters for the DGM neural network
init_op = tf.compat.v1.global_variables_initializer()

# Initialize the learning rate
```

```
learn_rt = tf.compat.v1.train.exponential_decay(start_learn_rt,
                                    global_step,100000, 0.96,staircase=
                                    True)

# Setting up Stochastics Gradient descent(AdamOptimizer)
optimizer = tf.compat.v1.train.AdamOptimizer(learn_rt=learn_rt).minimize
                                    (loss_tnsr)
# Random samples for the domain interior
t_int = np.random.uniform(low=t_low - 0.5*(T-t_low), high=T, size=[
                                    nSim_interior, 1])
w_int = np.random.uniform(low=w_low - 0.5*(w_high-w_low), high=w_high +
                                    0.5*(w_high-w_low), size=[nSim_int,
                                    1])

# random samples for initial/terminal condition
t_term = T * np.ones((nSim_term, 1))
w_term = np.random.uniform(low=w_low - 0.5*(w_high-w_low), high=w_high +
                                     0.5*(w_high-w_low), size = [
                                    nSim_terminal, 1])

# Compute derivatives at current samples
diff_V = -0.5 * theta**2 * V_w**2 + (V_t + r*w_int*V_w)*V_ww

# compute average L2-norm of differential operator
L1 = tf.reduce_mean(tf.square(diff_V))

# Loss 3 term for the initial/terminal condition
fitted_terminal = model(t_term, w_term)
L3 = tf.reduce_mean( tf.square(fitd_term - targ_term) )

# Calculate Gradient Descent Steps
for _ in range(steps_per_sample):
    loss,L1,L3,_ = sess.run([loss_tnsr, L1_tnsr, L3_tnsr, optimizer],
    feed_dict = {t_int_tnsr:  t_int,w_int_tnsr: w_int, t_term_tnsr:
                                        t_term,w_term_tnsr: w_term})
```

4.4 The finite difference method for the value function PDE

The previous section focused on the DGM. In the current section, attention is drawn to the finite difference method. To solve Equation (4.20) using FDM, the discrete shorthand notation is used. In addition, the forward difference and central difference formulas are employed to discretize the PDE. This allows the equation to be implemented in Python.

Now, the domain of the solution was choosen in the region $w \in [0, 1]$ with the goal of estimating $V(t, w_t)$ values at particular locations and times (Herman, 2015). The interval $[0, 1]$ is first divided into N subintervals of width, $h = 1/Nw$.

Then, the endpoints of the subintervals are given by:

$$w_j = jh, \quad j = 0, 1, \ldots, Nw.$$

Likewise, increment with time step K:

$$t_n = nK, \quad n = 0, 1, \ldots, Nt.$$

The results in the grid points $\left(w_j, t_n\right)$. It is crucial to note that the objective is to obtain a solution for $V_j^n \approx V\left(w_j, t_n\right)$.

Explicit finite difference method

The forward difference for PDE (4.20) in terms of time discretization is:

$$\frac{\partial V}{\partial t} = \frac{V_j^{n+1} - V_j^n}{K}. \tag{4.27}$$

The central difference for PDE (4.20) in terms of space discretization is:

$$\frac{\partial V}{\partial w_t} = \frac{V_{j+1}^n - V_{j-1}^n}{2h}. \tag{4.28}$$

The second-order central difference for PDE (4.20) of space discretization is:

$$\frac{\partial^2 V}{\partial w^2} = \frac{V_{j+1}^n - 2V_j^n + V_{j-1}^n}{h^2}. \tag{4.29}$$

Substituting the three terms (4.27), (4.28) and (4.29) into Equation (4.20) results in:

$$\frac{V_j^{n+1} - V_j^n}{K} - \frac{(\mu - r)^2}{2\sigma^2} \cdot \frac{\left(\frac{V_{j+1}^n - V_{j-1}^n}{2h}\right)^2}{\left(\frac{V_{j+1}^n - 2V_j^n + V_{j-1}^n}{h^2}\right)} + \left(\frac{V_{j+1}^n - V_{j-1}^n}{2h}\right) r w_t = 0.$$

Rearranging the above gives:

$$V_j^{n+1} = V_j^n - \frac{rKw_t}{2h}\left(V_{j+1}^n - V_{j-1}^n\right) + \frac{(\mu - r)^2}{8\sigma^2} \cdot K \cdot \frac{\left(V_{j+1}^n - V_{j-1}^n\right)^2}{\left(V_{j+1}^n - 2V_j^n + V_{j-1}^n\right)}. \tag{4.30}$$

5. RESULTS AND DISCUSSION

In this chapter, the insights gained from the application of the methods presented in chapter 4 are explored in greater depth. The focus of this chapter is on a comparison of the DGM with the FDM.

5.1 Data and numerical test

Quasi-linear PDE

Prior to implementing the PDE (4.20) in Python, the model was converted to Quasi-linear form:

$$-\frac{(\mu - r)^2}{2\sigma^2}\left(\frac{\partial V}{\partial w}\right)^2 + \frac{\partial^2 V}{\partial w^2}\left(\frac{\partial V}{\partial t} + r w_t \frac{\partial V}{\partial w_t}\right) = 0. \tag{5.1}$$

In this way it became numerically stable.

DGM numerical test

The region $(t; w) \in [0; 1]^2$, with oversampling of 50% in the space and time points, was selected for performing numerical tests on DGM. In addition, the following parameter values of the Merton problem were used for illustration: $r = 0.06$ (risk-free interest rate), $\mu = 0.3$ (mean value of assets), $\sigma = 0.35$ (volatility of assets), $p = 1$ (exponential utility) and $T = 1$ (terminal time).

Futhermore, for the DGM algorithm, 3 hidden layers were used and 50 neurons were selected for each layer, which meant that 8000 time-space points were sampled from the domain $(t, w) \in [0, 1]^2$. To ensure correct and efficient training of the neural network, a

learning rate of 0.001 was chosen.

FDM numerical test

For performing numerical test on FDM: The length of the spatial domain was set to $L = 1.0$, the terminal time $T = 1$ remained the same as in DGM, the number of spatial grid points Nw and the number of time steps Nt was set to 8000. Finally, the same Merton parameter values were used as in the DGM.

Duration

In terms of duration, it took 16 minutes to run the simulations and generate numerical results for DGM. On the other hand, it took only 3 minutes to run numerical simulations on the FDM. It is indeed surprising that while FDM saves time in execution, it does not provide accurate results compared to DGM (see Figure 5.5).

5.2 Comparing DGM and FDM: Value function

General observation

In analysing the Figures 5.1 and 5.2 of the value functions, a significant general observation comes to light: The DGM method is performing well in terms of predictions as compared to FDM which performs poorly. In both Figures, observe that at time t =0, both DGM and FDM value function figures have instabilities, but as time approaches $T = 1$, both numerical methods stabilise (i.e. both methods are improving in terms of estimation even though DGM is performing far better than FDM).

Time periods

Below, a detailed explanation for each time period is provided.

- At time t =0.

  FDM: In Figure 5.2, both lines of the estimated and analytical solution are far apart, indicating that FDM is imprecise in terms of estimations and has high instability.

  DGM: In Figure 5.1 both lines of the estimated and analytical solution, are little bit closer to each other indicating that DGM is slightly better in approximation as compared to FDM, but not giving accurate results at this time. This irregularity can be observed in the region [0, 0.3] and [0, -0.6], of the wealth and value function respectively. And, also in region [0.7, 1] and [-0.35, -0.6] of the wealth and value function respectively. But for values in the region [0.4, 0,6] of the wealth, the DGM estimated solution line touches the analytical solution line.

- At time t =0.25.

  FDM: In Figure 5.2 the estimated and analytical solution, are still far apart this indicates that FDM is still inaccurate and imprecise in terms of approximation and still has high instability.

  DGM: In Figure 5.1 DGM estimate line is getting closer to the analytical solution line this indicates that the DGM is slightly becoming better in terms of approximation as compared to FDM but is not yet giving accurate results. The DGM estimated solution line touches the Analytical solution line in all the regions, excluding only a portion of the range [0, 0.2] for the wealth.

- At time t =0.5.

  FDM: In Figure 5.2 both lines of the approximation solution and analytical solution, are slightly far apart. This still shows that FDM still possesses some instability.

  DGM: In Figure 5.1 the DGM estimate solution is getting closer to the analytical solution. This shows that DGM results have improved.

- At time t =1. Both methods stabilize, but with FDM still not giving accurate approximation while DGM fully giving accurate approximation.

5.3 Advising investors

In analysing the Figures 5.1 and 5.2 representing the DGM value function, a significant observation comes to light: the optimal time to offer advice to investors, facilitating well-informed decision-making in their investments, is when $t = 0.5$.

- For high level risk investors, the point of interest is (-0.4, 0.8). In this situation, the computed risk appetite is $(1 - 0.4 = 0.6)$ or 60%, with possible returns of 80%.
- For risk-averse investors considering portfolio investments, the point of interest is $(-0.52, 0.6)$. The risk appetite is $(1 - 0.52 = 0.48)$ or 48%, with a possible return of 60%.
- investors that value safety and have a low risk tolerance in their investment, the point of interest is (-0.75, 0.1). In this scenario, the risk appetite is $(1 - 0.75 = 0.25)$, or 25%, while the possible return on their investments is 10%.

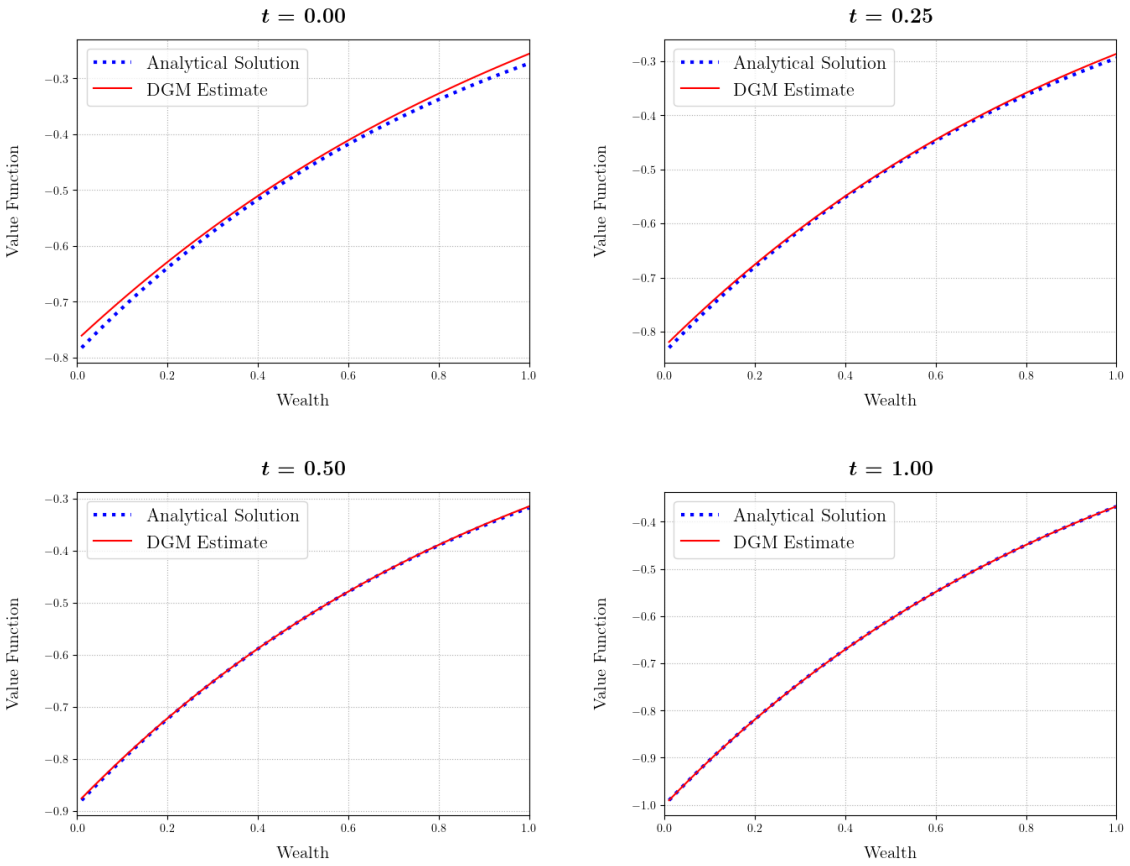


Figure 5.1: DGM value function

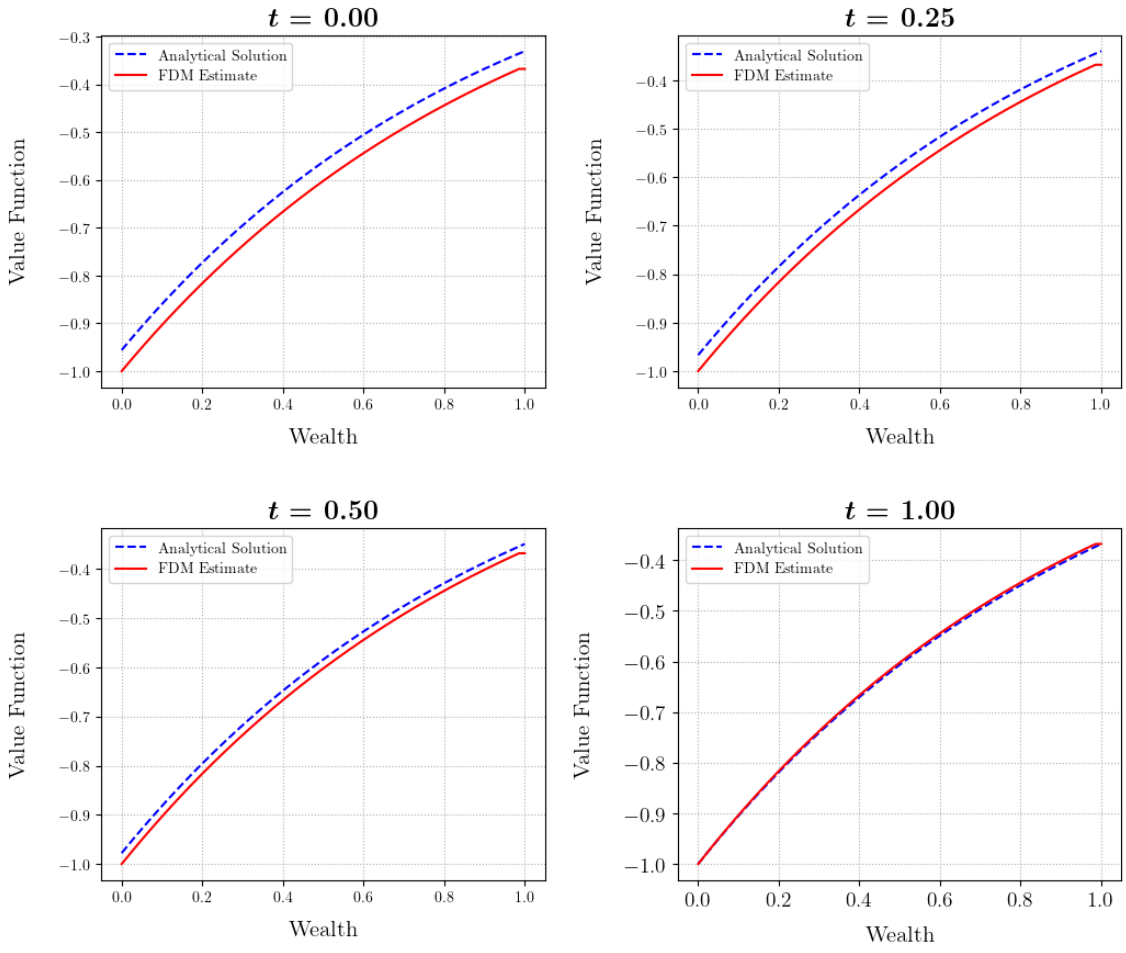


Figure 5.2: FDM value function

5.4 Comparing DGM and FDM: Optimal control

In analyzing Figures 5.3 and 5.4 illustrating the optimal control for DGM and FDM, a significant observation emerges: In general, the DGM method continues to demonstrate strong predictive performance compared to FDM, which appears to perform poorly. It is evident that DGM exhibits substantial improvement in approximating the optimal control analytical solution across all time periods: $t = 0$, $t = 0.25$, $t = 0.5$, and $t = 1$. Conversely, FDM, particularly noticeable at $t = 0$, performs well with only minor instability. However, as time progresses, FDM struggles to approximate the solution accurately.

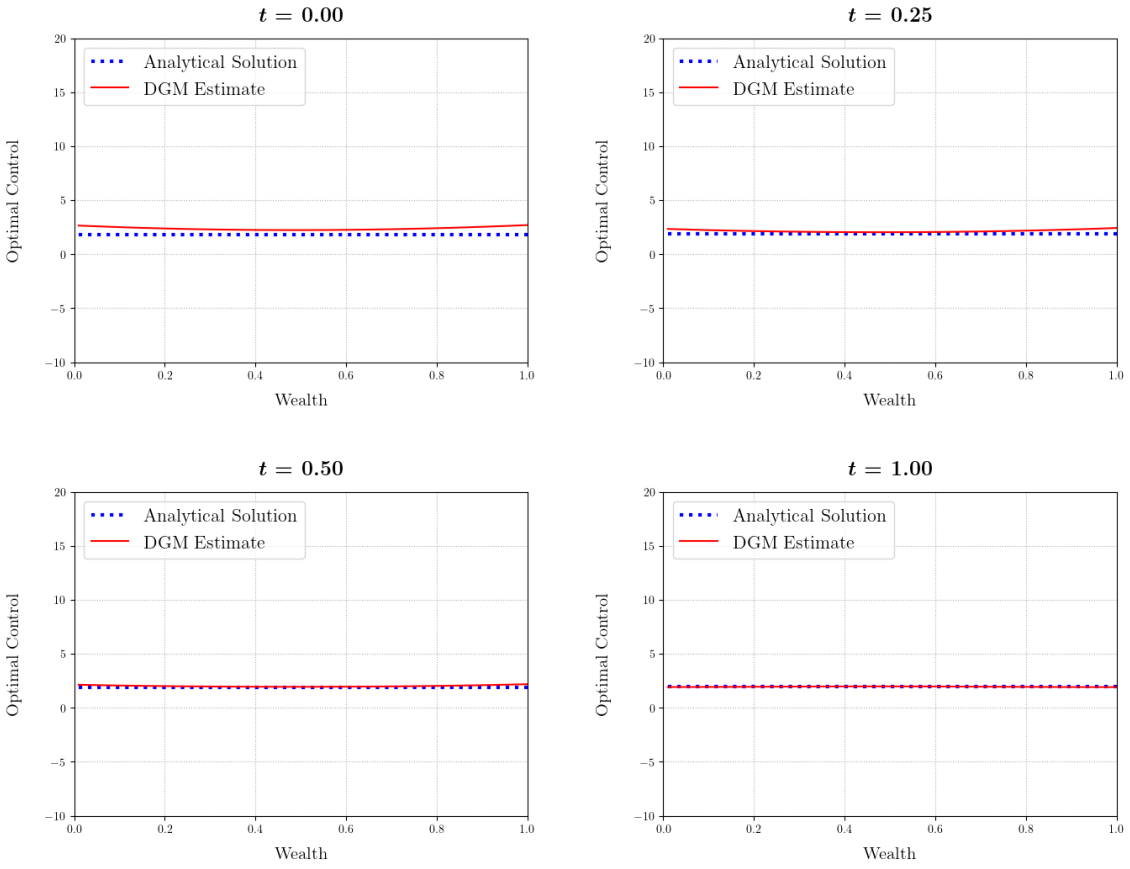


Figure 5.3: DGM optimal control

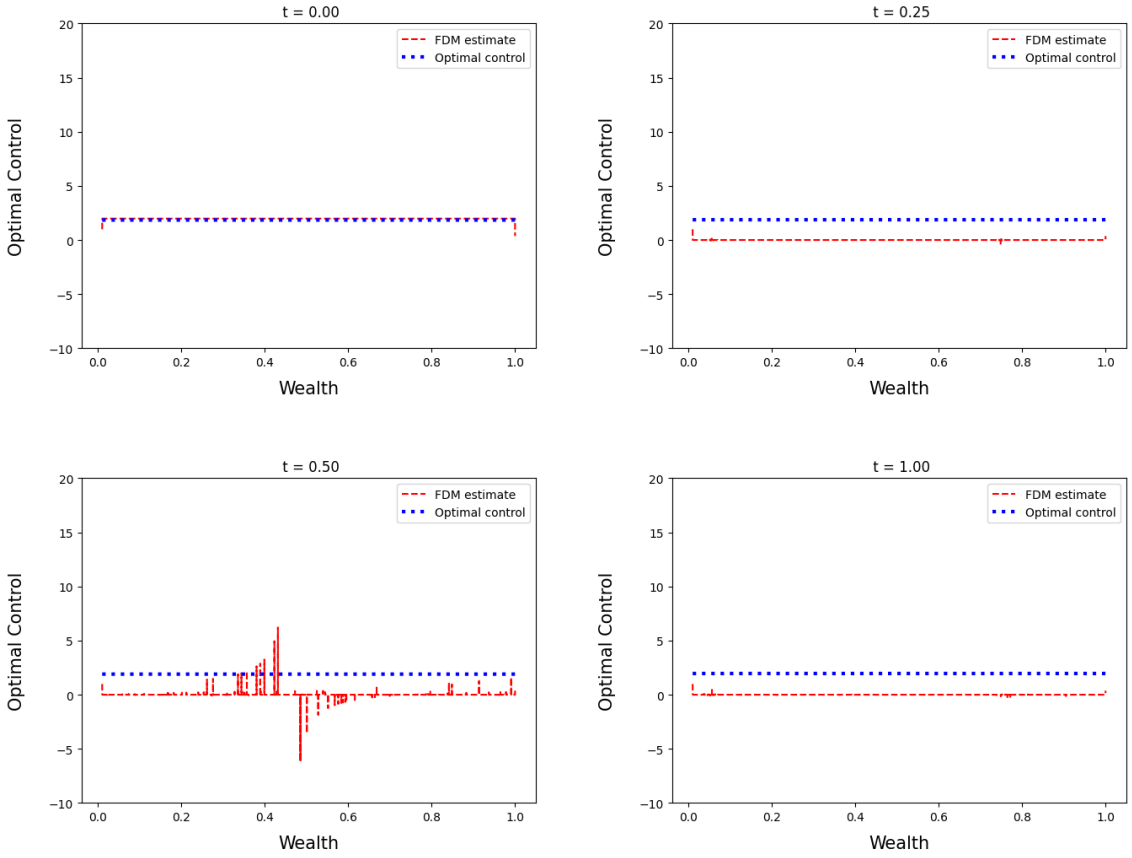
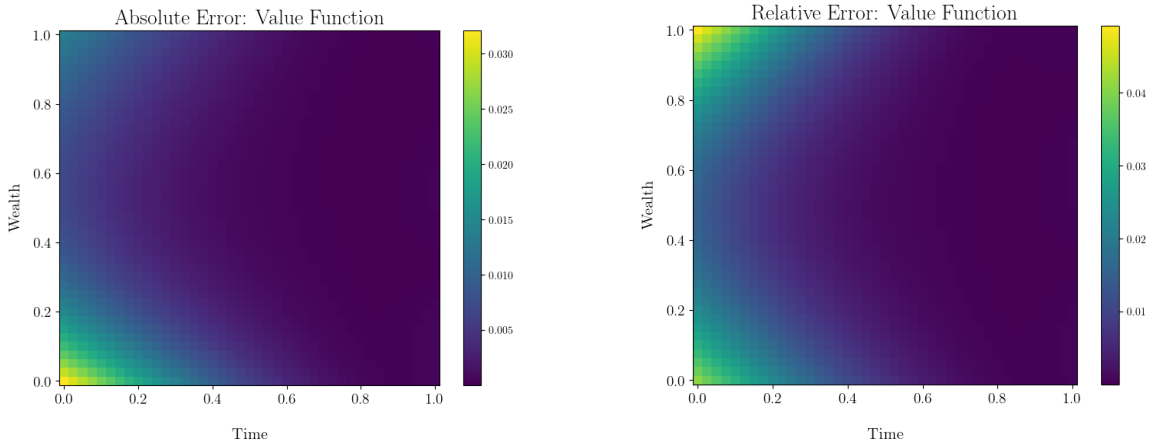


Figure 5.4: FDM optimal control

5.5 Comparing DGM and FDM: Errors value function and optimal control

In analyzing Figures 5.5 and 5.6, which illustrate the errors for the DGM and FDM value functions, a significant observation emerges: The absolute and relative errors in the region of (wealth, time) $\in [0, 1] \times [0, 0.6]$ are much higher for FDM but lower for DGM, as can be observed in the region of (wealth, time) $\in [0, 1] \times [0, 0.2]$. This supports the previously depicted Figure 5.1 of the value function, wherein DGM accurately predicted the value function.

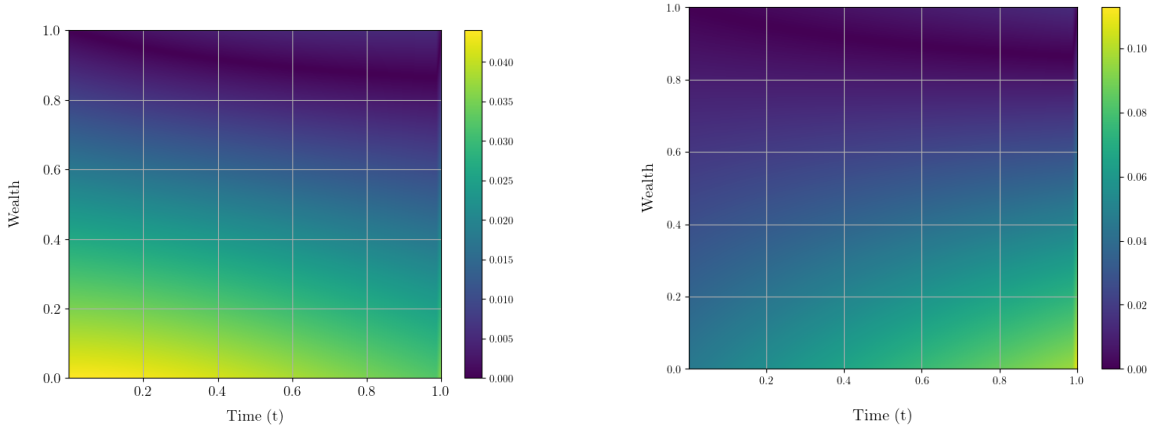
Similarly, in analyzing Figures 5.7 and 5.8, which illustrate the errors for the DGM and FDM value functions, a significant observation emerges: The absolute and relative errors in the region are widely spread out indicating a high level value error for FDM but lower for DGM, as it can be observed in the regions of (wealth, time) $\in [0, 0.2] \times [0, 0.2]$. This still supports the previously depicted Figure 5.1 of the value function, wherein DGM accurately predicted the value function.



(a) Absolute Error DGM value function

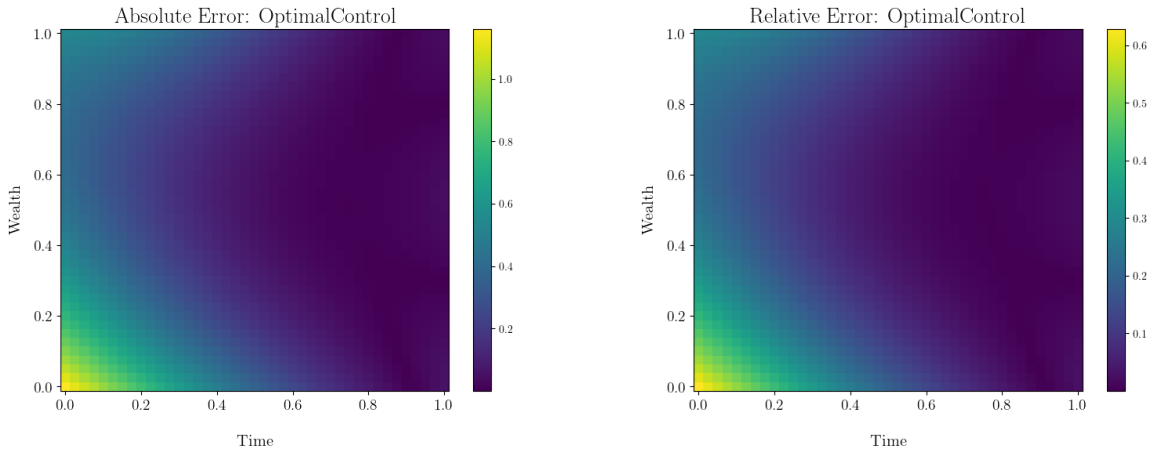
(b) Relative error DGM value function

Figure 5.5: Absolute and relative errors DGM value function



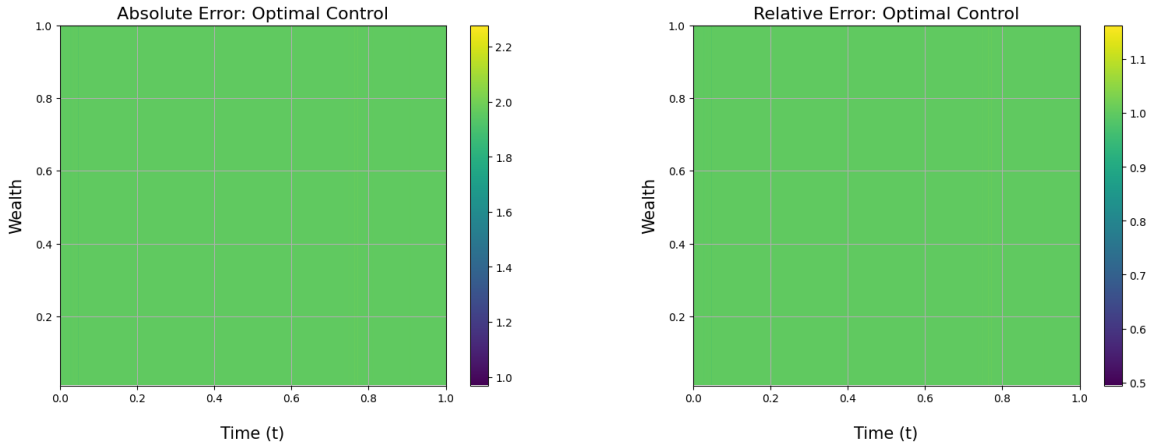
(a) Absolute error FDM value function (b) Relative Error FDM value function

Figure 5.6: Absolute and relative errors FDM value function



(a) Absolute error DGM optimal control (b) Relative error DGM optimal control

Figure 5.7: Absolute and relative errors DGM optimal control



(a) Absolute error FDM optimal control

(b) Relative error FDM optimal control

Figure 5.8: Absolute and relative errors FDM optimal control

Table 5.1: Error metrics for DGM and FDM

	DGM value function	DGM optimal control	FDM value function	FDM optimal control
Mean absolute error	0.00697	0.21799	0.13757	1.95912
Mean relative error	0.01262	0.21799	0.07199	1.00009

The values in Table 5.1 are the mean error values for both DGM and FDM optimal control and value function, respectively.

- For the corresponding value functions, DGM has a mean absolute error of 0.00697, while the FDM mean absolute error is 0.13757.
- For the corresponding value functions, DGM has a mean relative error of 0.01262, while FDM has a mean relative error of 0.07199.
- For a corresponding optimal control, DGM has a mean absolute error of 0.13757, while the FDM mean absolute error is 1.95912.
- For a corresponding optimal control, DGM has a mean relative error of 0.21799, while FDM has a mean relative error of 1.00009.

- In all the above cases, one can observe that the mean absolute error and mean relative error for FDM are greater than those of DGM. This is an indication that DGM performed well in terms of approximations.

6. CONCLUSIONS, RECOMMENDATIONS AND FUTURE WORK

The research delved into the theoretical underpinnings of portfolio optimization through a structured approach. Initially, the study addressed two duality problems aimed at maximizing anticipated returns while minimizing risk variance. Additionally, it explored the one-fund separation theorem, which integrates risk free assets, an aspect not covered by the initial duality problems.

In Chapter 4, attention shifted towards the Merton problem, where the transformation framework utilizing the Hamilton-Jacobi-Bellman equation was introduced. This transformation aimed to convert the problem into a partial differential equation (PDE) along with the corresponding optimal controls.

Moreover, a detailed examination of the mathematical frameworks underpinning computational approaches, namely the Deep Galerkin Method (DGM) and Finite Difference Method (FDM), was provided. The study meticulously described the discretization and implementation of the PDE across both methodologies, with Python serving as the primary computational tool.

In summary, the comparative analysis between the FDM and DGM provided valuable insights into their effectiveness in approximating solutions for PDEs. The DGM emerged as the superior choice, showcasing enhanced stability and robustness in handling complex PDE without succumbing to singularities, unlike FDM. Conversely, FDM exhibited notable flaws, particularly in encountering singularities arising from the nonlinearity of discretized PDE, compromising its accuracy and reliability.

In light of these noteworthy findings, the strong recommendation is for researchers and practitioners to favor the utilization of the DGM despite its relatively slower computational speed. The precision and reliability offered by DGM significantly outweigh its drawbacks, positioning it as the preferred choice for tasks prioritizing accuracy and stability. Caution

is particularly warranted when considering the FDM, given its propensity for higher mean error and susceptibility to singularities in specific scenarios. It is crucial for users to exercise discernment, especially in applications demanding high accuracy and robustness. This comparative analysis serves as a valuable guide for selecting the most appropriate numerical approximation method, tailored to the specific requirements of the task at hand.

Lastly, due to time constraints, the study only focused on the implementation of DGM and FDM for the one-dimensional Merton problem. However, for future research, the goal is to extend the investigation to a two-dimensional, three-dimensional, or even n-dimensional Merton problem. In addition, the research efforts will include the exploration of other computation method alternatives that are expected to be compared with the existing DGM.

Furthermore, it is imperative to note that the current study relied on randomly sampled data obtained from Python. The future study hopes to use real-world data from businesses. The purpose is to gauge the realistic portrayal of the algorithm's efficiency and to widen the research's practical sphere. Furthermore, the results should play a significant role in portfolio management by offering valuable and reliable investment instruments to investors to help them make informed decisions.

BIBLIOGRAPHY

Al-Aradi, A. (2018). *Deep galerkin method*. https://github.com/alialaradi/DeepGalerkinMethod/blob/master/MertonProblem.py

Al-Aradi, A., Correia, A., Naiff, D. d. F., Jardim, G., & Saporito, Y. (2019). Extensions of the deep galerkin method. *arXiv preprint arXiv:1912.01455*.

Al-Aradi, A., Correia, A., Naiff, D., Jardim, G., & Saporito, Y. (2018). Solving nonlinear and high-dimensional partial differential equations via deep learning. *arXiv preprint arXiv:1811.08782*.

Alexander, W. (2021). *Lecture 9: Attitudes toward risk* [Accessed: October 3, 2023]. MIT. https://ocw.mit.edu/courses/14-121-microeconomic-theory-i-fall-2015/07a559609869398fc6f98e550a9d2d80_MIT14_121F15_6S.pdf

Asaad, M. A. (2019). *A finite difference method (fdm)* (B.S. thesis). The College of Science Basrah. https://un.uobasrah.edu.iq/lectures/13717.pdf

Baeldung. (2023). *Differences between gradient, stochastic and mini batch gradient descent* [Accessed: October 3, 2023]. baeldung.com. https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch

Benth, F. E. (2003). *Option theory with stochastic analysis: An introduction to mathematical finance*. Springer Science & Business Media.

Benth, F. E., Karlsen, K. H., & Reikvam, K. (2003). Merton's portfolio optimization problem in a Black and Scholes market with non-gaussian stochastic volatility of Ornstein-Uhlenbeck type. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, *13*(2), 215–244.

Byjus, S. (2023). *Partial differential equations* [Accessed: October 3, 2023]. Byjus. https://byjus.com/maths/partial-differential-equation/

Cartanyà Caro, P. (2022). *A deep learning approach to portfolio optimization* (B.S. thesis). Universitat Politècnica de Catalunya.

De Bondt, W. F. M., & Thaler, R. H. (1985). Does the stock market overreact? *The Journal of Finance*, *40*(3), 793–805. https://doi.org/10.1111/j.1540-6261.1985.tb05004.x

De Bondt, W. F. M., & Thaler, R. H. (1987). Further evidence on investor overreaction and stock market seasonality. *The Journal of Finance*, *42*(3), 557–581. https://doi.org/10.1111/j.1540-6261.1987.tb02505.x

Dongare, A., Kharde, R., Kachare, A. D., Et al. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, *2*(1), 189–194.

El Naqa, I., & Murphy, M. J. (2015). *What is machine learning?* Springer.

Farhadi, A., Erjaee, G. H., & Salehi, M. (2017). Derivation of a new merton's optimal problem presented by fractional stochastic stock price and its applications. *Computers & Mathematics with Applications*, *73*(9), 2066–2075.

Fishburn, P. C. (1970). *Utility theory for decision making* (tech. rep.). Research analysis corp McLean VA.

Ganesh, S. (2021). *Partial differential equations* [Accessed : November 26, 2023]. https://www.math.iitb.ac.in/~siva/ma515152022/3MA515Notes.pdf

Gim, D., & Park, H. (2021). A deep learning algorithm for optimal investment strategies. *arXiv preprint arXiv:2101.12387*.

Han, B., & Wong, H. Y. (2021). Merton's portfolio problem under volterra heston model. *Finance Research Letters*, *39*, 101580.

Han, J., Jentzen, A., & Weinan, E. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, *115*(34), 8505–8510.

Herman, R. L. (2015). Introduction to partial differential equations. *North Carolina, NC, USA: RL Herman*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hull, J. C. (2000). Options, futures, and other derivatives, th ed. Prentice Hall, New Jersey.

javatpoint. (2023). *Gradient descent in machine learning* [Accessed: October 3, 2023]. javatpoint.com. https://www.javatpoint.com/gradient-descent-in-machine-learning

Karlsson Faronius, H. (2023). *Solving partial differential equations with neural networks* (Master's thesis). Uppsala University. https://www.diva-portal.org/smash/get/diva2:1746454/FULLTEXT02

Krishnamachari, R. T. (2017). Big data and AI strategies.

Lagaris, I. E., Likas, A. C., & Papageorgiou, D. G. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, *11*(5), 1041–1049.

Lee, H., & Kang, I. S. (1990). Neural algorithm for solving differential equations. *Journal of Computational Physics*, *91*(1), 110–131.

Li, J., Yue, J., Zhang, W., & Duan, W. (2022). The deep learning galerkin method for the general stokes equations. *Journal of Scientific Computing*, *93*(1), 5.

Li, J., Zhang, W., & Yue, J. (2021). A deep learning galerkin method for the second-order linear elliptic equations. *International Journal of Numerical Analysis & Modeling*, *18*(4).

Malek, A., & Beidokhti, R. S. (2006). Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied Mathematics and Computation*, *183*(1), 260–271.

Mangram, M. E. (2013). A simplified perspective of the markowitz portfolio theory. *Global journal of business research*, *7*(1), 59–70.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91.

McClure, B. (2010). Modern portfolio theory: Why it's still hip. *Investopedia. Retrieved on*, *16*(11), 2018.

Merton, R. C. (1969). Lifetime portfolio selection under uncertainty: The continuous-time case. *The review of Economics and Statistics*, 247–257.

Milanesi, P. (2020). *The merton problem with derivatives* (Masters thesis). The Polytechnic University of Milan.

Modigliani, F., & Miller, M. H. (1963). Corporate income taxes and the cost of capital: A correction. *The American economic review*, *53*(3), 433–443.

Morien, T. (2005). Travis morien financial advisors. *Retrieved on MPT Criticism*, *12*(18), 11.

Pandi, A. (2020). *Mean-semivariance approach for portfolio optimisation* (Masters thesis). North West University. https://repository.nwu.ac.za/bitstream/handle/10394/35692/Pandi_AN.pdf

Pedersen, J. L., & Peskir, G. (2017). Optimal mean-variance portfolio selection. *Mathematics and Financial Economics*, *11*(2), 137–160.

Shreve, S. E. Et al. (2004). *Stochastic calculus for finance ii: Continuous-time models* (Vol. 11). Springer.

Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, *375*, 1339–1364.

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Strauss, W. A. (2007). *Partial differential equations: An introduction*. John Wiley & Sons.

Van Veen, F. (2016). *Neural network zoo* [Posted on September 14, 2016. Accessed on < November 22, 2023>]. https://www.asimovinstitute.org/neural-network-zoo/

Wang, K. (2019). *Portfolio optimisation under rough stochastic volatility via machine learning* (Master's thesis).

Wiederhold, G., & McCarthy, J. (1992). Arthur samuel: Pioneer in machine learning. *IBM Journal of Research and Development*, *36*(3), 329–331.

Wojt, A. (2009). Portfolio selection and lower partial moments. *Stockholm: Royal Institute of Technology Working Paper*. http://www.%20math.%20kth.%20se/matstat/seminarier/reports/M-exjobb09/091214b.%20pdf

Yang, J., & Zhu, Q. (2021). A local deep learning method for solving high order partial differential equations. *NUMERICAL MATHEMATICS-THEORY METHODS AND APPLICATIONS*.