Deep Learning

for

Reflected Backwards Stochastic Differential Equations

A Major Qualifying Project Report Submitted to The Faculty of Worcester Polytechnic Institute In partial fulfillment of the requirements for the Degree of Bachelor of Science

Written By

Frederick "Forrest" Miller

Approved by:

Stephan Sturm

March 24, 2023

Abstract

In this work, we in investigate the theory and numerics of reflected backwards stochastic differential equations (RBSDEs). We review important concepts from stochastic calculus, as well as key theoretical properties of (R)BSDEs. We provide an overview of feedforward neural networks and their applications to functional approximation for numerical implementations. We also discuss the key application of RBSDEs to the field of mathematical finance, in particular indifference pricing of put options. Lastly, we present preliminary theoretical and numerical results of Risk Indifference pricing of American from both the Buyer's and Seller's perspectives.

Acknowledgements

I would like to thank Professor Stephan Sturm for advising me throughout the writing of this report. Additionally, I would like to thank Professor Rohini Kumar (Wayne State University), as well as Dr. Hussein Nasralah (University of Michigan Dearborn). Without them, I would not have been able to make nearly as much progress with the theory and numerics of this report.
I would like to thank my parents and sisters for their endless encouragement through my life. I am extremely grateful for all their suggestions balancing my courseload with healthy habits.
I would also like to thank all of my friends throughout the years for listening to my talk about whatever I was excited about at that moment. You are too many to list, but I am going to try anyway: Charlotte Clark, Noah Ziff, Jakob Misbach, Maggie Munroe, Guillermo Nunez Ponasso, Elisa Negrini, Giulio Farolfi, Tony Vuolo, Camille Williams, Avery Smith, Ben Gobler, Jessica Wang, Scar Clarke, Ethan Washock, Matt Dzwill, Dan Quackenbush, Neil Kale, Jackson Sypek, Daniil Volkov, Felix Liu, Antrim Lottick, Grace Tiddei, Siavash Raissi, and many others who have supported and listened to me about math along the way.
Thank you to Greg Audin for keeping my spirits up during late evenings of work in the Math Lounge.

Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

Differential equations can often be used to model deterministic systems, where at each time point t from 0 to T, the state of the system can be known with certainty. However, this is not always the case. A major example of this is the price of an asset, such as an option, in the stock market. Therefore, it is important to have a strong theoretical underpinning for random (also known as stochastic) systems. In particular, *stochastic* differential equations can be used to model options in the stock market. One of the most famous examples is the Nobel Prize winning Black Scholes model (Black and Scholes, 1973) that uses the following stochastic differential equation;

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{1.1}$$

where μ is the mean return and σ is the volatility of the asset. The Black Scholes option pricing framework they developed has formed the underpinnings of modern financial option pricing.

In addition to the Black Scholes model, there has been much work done to expand the model to handle a wider class of financial options. This has been done by expanding upon the SDE above as well as considering other pricing frameworks such as indifference pricing.

In this work, we provide the theoretical underpinnings for stochastic differential equations. We prove a key existence and uniqueness result for backwards stochastic differential equations (BSDEs) and reflected backwards stochastic differential equations (RBSDEs). While it is often useful to have theoretical results, analytical solutions to many BSDEs and RBSDEs are currently unknown. As such, it is important to be able to utilize numerical methods to find numerical solutions. For the case of financial mathematics, this is crucial as numerical techniques are the primary way that market players implement trading strategies. We utilize tools from deep learning, in particular feed forward neural networks and gradient descent algorithms, to find numerical solutions to (R)BSDEs to approximate the price of financial options.

We conclude by extending the work of Sircar and Sturm (2015) and provide preliminary numerical results to the problem of risk indifference pricing from the buyer's and seller's perspective of an American Put option under a stochastic volatility model. We use a deep learning framework from Gao et al. (2022) for computing the risk indifference price.

Chapter 2

Stochastic Calculus

In this chapter, we provide an overview of the tools from Probability Theory and Stochastic Calculus that we use to study the Reflected Backwards Stochastic Differential Equations.

2.1 Probability Spaces

In this section, we formalize the notion of what it means for an event to have a certain probability of occurring.

Definition 2.1 (Sample Space Ω)**.** *We call a set Ω the sample space of events $\omega \in \Omega$. ω is called a (potential) outcome.*

Example 2.1. *Consider a fair coin with 2 sides. If we flip the coin once, the sample space $\Omega = \{H, T\}$, where the coin can be heads (H) or tails (T).*

This example can then be expanded to flipping a coin n times. The sample space Ω becomes all the possible permutations of the resulting n flips. So, if the coin were flipped twice, we have that $\Omega = \{HH, HT, TH, TT\}$.

In order to study and quantify probabilities of more complex events, we need to utilize a σ-algebra to encode sets of possible events.

Definition 2.2 (σ-algebra $\mathcal{F}$)**.** *If Ω is a given set, then a σ-algebra $\mathcal{F}$ on Ω is a set of subsets of Ω such that the following properties hold:*

(i) $\emptyset \in \mathcal{F}$

(ii) $F \in \mathcal{F} \Rightarrow F^C \in \mathcal{F}$

(iii) $F_1, F_2, ... \in \mathcal{F} \Rightarrow F := \cup_{n=1}^{\infty} F_n \in \mathcal{F}$

(Øksendal, 2003)

We say that the pair $(\Omega, \mathcal{F})$ is called a measurable space. Furthermore, a measure space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where $\mathbb{P}$ is a measure (defined below) on $(\Omega, \mathcal{F})$.

Definition 2.3 (Probability Measure $\mathbb{P}$)**.** *A function $\mathbb{P} : \mathcal{F} \to [0, 1] \subset \mathbb{R}$ is a probability measure if $\mathbb{P}$ satisfies:*

(i) $\mathbb{P}(\Omega) = 1$

(ii) if the collection $\{F_i\}_{i \in \mathbb{N}} \in \mathcal{F}$ is disjoint, then

$$\mathbb{P}\left(\bigcup_{n=0}^{\infty} F_n\right) = \sum_{n=0}^{\infty} \mathbb{P}(F_n)$$

If $\{F_i\}_{i\in\mathbb{N}} \in \mathcal{F}$ is not disjoint, we have subadditivity, which is that $\mathbb{P}\left(\bigcup_{n=0}^{\infty} F_n\right) \leq \sum_{n=0}^{\infty} \mathbb{P}(F_n)$.

Example 2.2. *Consider the sample space from Example 2.1. A probability measure $\mathbb{P} : \Omega \to [0,1]$ can be defined as $\mathbb{P}(H) = \mathbb{P}(T) = \frac{1}{2}$. This is known as flipping a "fair" coin.*

From the definition of probability measure, we have that the measure in Example 2.2 is a probability measure.

Definition 2.4 (Probability Space)**.** *We call the triple $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space, with Ω as in Definition 2.1, $\mathcal{F}$ as in Definition 2.2, and $\mathbb{P}$ as in Definition 2.3.*

Additionally, we call $F \subset \Omega$ such that $F \in \mathcal{F}$ an *event*, and we can then say that $\mathbb{P}(F)$ represents the *probability* that F occurs. More formally, we say that F is $\mathcal{F}-$measurable.

Definition 2.5 (Random Variable X)**.** *A random variable X is an $\mathcal{F}-$measurable function $X : \Omega \to \mathbb{R}$*

Additionally, it is important to note that every random variable X induces a probability measure $\mathbb{P}_X$, where $\mathbb{P}_X$ is the distribution of X. Some popular examples for distributions include the normal distribution, the exponential distribution, binomial distribution, and many others.

Definition 2.6 (Expectation)**.** *The expectation, or expected value, of X is defined as*

$$\mathbb{E}[X] := \int_{\Omega} X(\omega) d\mathbb{P}(\omega)$$

The expectation requires that the integral above is well defined.

In other words, the expectation is the average value for a random variable. For example, $X \sim N(\mu, \sigma^2)$ has $\mathbb{E}[X] = \mu$.

2.2 Stochastic Processes

Using random variables, we can now define one of the central objects for this paper: the stochastic process.

Definition 2.7 (Stochastic Process X_t)**.** *A stochastic process is a collection of random random variables $(X_t, 0 \leq t \leq T < \infty)$ defined on $(\Omega, \mathcal{F}, \mathbb{P})$, valued in $\mathbb{R}^n$.*

T is known as the terminal time, as most applications involve a finite time horizon. Two major types of stochastic processes are discrete stochastic processes and continuous stochastic processes. For discrete stochastic processes, the index set between 0 and T is countable. For continuous stochastic processes, the index set is typically the interval $[0, T]$.

Just as we can use equations to describe deteriminstic functions, we can do the same with stochastic processes. Below, we provide an example of a stochastic process defined on $[0, 10]$, with discrete time steps.

Example 2.3. $X_{t+1} = X_t + \sigma X_t + t\mu, X_0 = 0$

This is known as a random walk with drift μ. If $\mu = 1$ and $\sigma = .2$, and we sample $X_t \sim \mathcal{N}(0, 1)$ distributions, we can obtain the following plot:

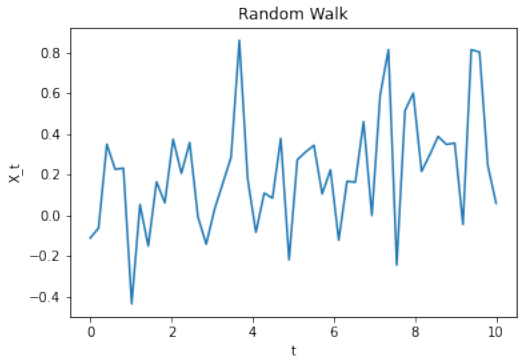


Figure 2.1: One possible realization of the random walk.

Stochastic processes are very useful for describing unpredictably phenomena. A particular example, which we will explore extensively in Chapter 6, is a stock price.

2.3 Conditional Expectation

From basic probability theory, we have that a conditional probability is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \text{ if } P(B) > 0, \tag{2.1}$$

for two events A and B. This then reads as the probability an event A occurs, given that B has already occurred. As a basic example, consider the probability of a die being rolled showing a 3 on top given that the value is odd. In this case, the *conditional* probability of this event is $\frac{1}{3}$, while the unconditional probability of this event would be $\frac{1}{6}$, as we have no prior information to work with.

Conditional expectations seek to apply this notion of applying known knowledge when computing an expected value of a random variable. Therefore, we can define the conditional expectation as follows

Definition 2.8 (Conditional Expectation)**.** *Given a random variable X with $\mathbb{E}[|X|] < \infty$, the conditional expectation of X given an event A has occurred:*

$$\mathbb{E}[X|A] = \frac{\mathbb{E}[I_A X]}{P(A)}, \text{ if } P(A) > 0$$

Where

$$I_A(\omega) = \begin{cases} 1 & \text{if } \omega \in A \\ 0 & \text{if } \omega \notin A \end{cases}$$

As a reminder, $\omega \in A$ means that that a possible realization ω is in the event A. It is important to note that the conditional expectation is no longer a singular number, but instead is now a random variable. In fact, it can be thought of as a rough approximation of the random variable X given the knowledge of A.

Before providing some theoretical properties of the conditional expectation, we first define what it means for events and variables to be *independent*.

Definition 2.9 (Independent Events and Variables)**.** *The events $A_1, ..., A_n$ are independent if for every choice of indices $i_1, ..., i_k$ where $1 \leq k \leq n$, we have that*

$$P(A_{i_1} \cap ... \cap A_{i_k}) = P(A_{i_k})...P(A_{i_k}) \tag{2.2}$$

The random variables $X_1, ..., X_n$ are independent if for every choice of indices $i_1, ..., i_k$ where $1 \leq k \leq n$ as well as subsets $B_1, ..., B_n$ of $\mathcal{F}$, we have that

$$P(X_{i_1} \in B_{i_1} \cap ... \cap X_{i_k} \in B_{i_k}) = P(X_{i_1} \in B_{i_1})...P(X_{i_k} \in B_{i_k}) \tag{2.3}$$

Remark 2.1. *The following properties can be derived for the conditional expectation:*

1. *Linearity. For random variables X_1, X_2, and an event $A \in \mathcal{F}$, and constants $c_1, c_2 \in \mathbb{R}$ we have*

$$\mathbb{E}[(c_1X_1 + c_2X_2)|A] = c_1\mathbb{E}[X_1|A] + c_2\mathbb{E}[X_2|A]$$

2. *If X and A are independent, $E[X|A] = E[X]$. X and A being independent means that the information contained in A does not restrict the domain of X.*

3. *$\mathbb{E}[X|A]$ only uses the information present in A.*

4. *$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|A]]$*

More can be found in (see Mikosch, 1998, section 1.4), (see Shreve, 2004, chapter 2.3)

2.4 Filtration

The concept of a filtration is the formalization of a system gaining information over time. Filtrations allow for the construction of a martingale, which is crucial to the Itô integral, which is a key piece of a stochastic differential equation. Thus, we provide the definition below for a filtration and a stochastic process that is adapted to a filtration.

Definition 2.10 (Filtration $\mathcal{F}_t$)**.** *Let T an interval as in Definition 2.7. The collection $(F_t, 0 \leq t \leq T)$ of $\sigma-$algebra's on Ω is a continuous time filtration if*

$$\mathcal{F}_s \subset \mathcal{F}_t, \quad \forall\ 0 \leq s \leq t \leq T$$

Alternatively, consider $(\mathcal{F}_t, t \leq T)$, where T is a countable set. This is is a sequence of sets defined in Ω and $\mathcal{F}_t \subset \mathcal{F}_{t+1}$ for all $t \in T$, $(\mathcal{F}_t, t \leq T)$ is a filtration.

A filtration is an increasing stream of information. As t increases, the amount of information present in the system grows. We can apply this stream of information to our stochastic processes to apply them to financial systems.

Definition 2.11 (Stochastic Process X_t Adapted to the Filtration $\mathcal{F}_t$)**.** *Let T be a set as defined in 2.7. Then, the stochastic process $X_t = (X_t, t \in T)$ is adapted to the filtration $(\mathcal{F}_t, t \leq T)$ if*

$$\sigma(X_t) \subset \mathcal{F}_t, \forall t \geq 0$$

Where $\sigma(X_t)$ is the $\sigma-$field generated by X_t.

It is important to note that a stochastic process $(X_t, 0 \leq t \leq T)$ is always adapted to what is known as the *natural* filtration. The natural filtration is the filtration associated with the process that records all past behavior at each time point. More formally, this is of the form

$$\mathcal{F}_t = \sigma\{X_s^{-1}(A)|0 \leq s \leq t \leq T\}, A \in \Omega\}$$

Which is the preimages of the random variable at each timepoint. Here, Ω is the $\sigma-$algebra used for this particular stochastic process.

2.5 Martingales

One of the most important properties that is studied in stochastic calculus is the *martingale*. Martingale's allow us to formalize many notions about the behavior of stochastic processes, and study classes of stochastic processes together at once. Here, we provide the definitions of a discrete and continuous time Martingale from (see Mikosch, 1998, 1.5).

Definition 2.12 (Discrete Time Martingale). *The stochastic process $X = (X_n, n = 0, 1, 2, ...)$ is a discrete time martingale with respect to the filtration $(\mathcal{F}_n, n = 0, 1, 2, ...)$, denoted by $(X, (\mathcal{F}_n))$, if the following properties hold*

(i) $\mathbb{E}[|X_n|] < \infty$ *for all* $n = 0, 1, 2, ...$

(ii) X *is adapted to the filtration* $\mathcal{F}_n$ *as in Definition 2.11*

(iii) $\mathbb{E}[X_{n+1}|\mathcal{F}_n] = X_n$, *for all* $n = 0, 1, 2, ...$

This last item is often called the Martingale Property. Informally, it states that the best guess for the value of the stochastic process in the future is the guess at the current time. This also has a continuous time analogue which we will show below.

Definition 2.13 (Continuous Time Martingale). *The stochastic process $X = (X_t, 0 \leq t \leq T)$ is a continuous time martingale with respect to the filtration $(\mathcal{F}_t, 0 \leq t \leq T)$, denoted by $(X, (\mathcal{F}_t))$, if the following properties hold*

(i) $\mathbb{E}[|X_t|] < \infty$ *for all* $0 \leq t \leq T$

(ii) X *is adapted to the filtration* $\mathcal{F}_t$ *as in Definition 2.11*

(iii) $\mathbb{E}[X_t|\mathcal{F}_s] = X_s$, *for all* $0 \leq s < t \leq T$

With these definitions, we can now define one of the most important objects in stochastic calculus: Brownian Motion.

2.6 Brownian Motion

Brownian motion is the one of the most important aspects of stochastic calculus. Brownian motion has allowed for many breakthroughs in physics, biology, and mathematical finance. Brownian motion was first mathematically studied by Louis Bachelier in his doctoral thesis (Bachelier, 1900), which in English is titled *Theory of Speculation*. This was the first work to use Brownian Motion to model asset prices. The problem of option pricing was later popularized by Black and Scholes (1973), in which the modern option pricing theory was born. Myron Scholes and Robert Merton were awarded the Nobel Prize in Economic Sciences in 1997 for this work (Fischer Black had passed away in 1995, and the Nobel Prize is not awarded posthumously). Brownian Motion did not become popularized until Einstein published his work (Einstein, 1956 - 1926). Brownian Motion is also sometimes referred to as a Wiener Process because of the work of work of Norbert Wiener in 1918 (Wiener, 1976).

Below, we provide two popular definitions for Brownian motion, from (see Mikosch, 1998, 1.3.1) and (see Shreve, 2004, 3.3.1) respectively. While they are quite similar, Definition 2.14 comes from a more elementary text than the other. For completeness, both are included.

Definition 2.14 (Brownian Motion W_t, Mikosch (1998)). *A one dimensional stochastic process $W_t, t \geq 0$ is called Brownian Motion if:*

(i) $W_0 = 0$

(ii) It has stationary, independent increments

(iii) For every $t > 0$, $W_t \sim \mathcal{N}(0, t)$ *distribution*

(iv) Every sample path is continuous

Note that $\mathcal{N}(0, t)$ is a normal distribution with mean 0 and variance t.

Stationary increments means that $W_t - W_s$ is equal in distribution to $W_{t+h} - W_{s+h}$ for all $t, s \geq 0$ such that $t + h, s + h \in [0, T]$. Independent increments means that for every choice $t_i \in [0, T]$, with $t_1 < ... < t_n$ and $n \geq 1$, we have that

$$W_{t_2} - W_{t_1}, ..., W_{t_n} - W_{t_{n-1}}$$

are independent random variables.

Definition 2.15 (Brownian Motion W_t, Shreve (2004)). *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. For each $\omega \in \Omega$, suppose there is a continuous function $W_t, t \geq 0$ that depends on ω and $W_0 = 0$. Then, $W_t, t \geq 0$ is a Brownian Motion if f for all $0 = t_0 < ... < t_m$ the increments*

$$W_{t_1} - W_{t_0}, ..., W_{t_m} - W_{t_{m-1}}$$

are independent and each is normally distributed with mean 0 and variance $t_{i+1} - t_i$

Additionally, we can define a filtration that embeds on Brownian Motion.

Definition 2.16 (Filtration for Brownian Motion). *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space on which we define W_t as a Brownian Motion, $t \geq 0$. A filtration for Brownian Motion $\mathcal{F}_t, t \geq 0$ is a collection of $\sigma-$algebras satisfying*

(i) For $0 \leq s < t$, $\mathcal{F}_s \subset \mathcal{F}_t$.

(ii) W_t is $\mathcal{F}_t$-measurable.

(iii) $W_u - W_t$ is independent of $\mathcal{F}_t$, if $0 \leq t < u$.

Brownian Motion produces several powerful results that we highlight we proof below.

Theorem 2.1. *Let W_t be a Brownian Motion as outlined in Definition 2.14. Then, W_t is a martingale with respect to the filtration $\mathcal{F}_t = \sigma(W_s, 0 \leq s \leq t \leq T)$, which is the filtration generated by Brownian motion.*

Proof. Fix $T > 0 \in \mathbb{R}$, and $s < t < T$. Note that

$$\mathbb{E}[W_t] = 0$$

as $W_t \sim \mathcal{N}(0, t)$ comes from a normal distribution of mean 0. Therefore, W_t has finite expectation. Additionally, since F_t is a filtration defined on the Brownian Motion (the natural filtration), Brownian motion is adapted to the natural filtration. Lastly, we have

$$\mathbb{E}[W_t|\mathcal{F}_s] = \mathbb{E}[W_t - W_s + W_s|\mathcal{F}_s],$$

using the fact that $W_s - W_s = 0$. Applying linearity of expectations, we have

$$\mathbb{E}[W_t|\mathcal{F}_s] = \mathbb{E}[W_t - W_s|\mathcal{F}_s] + \mathbb{E}[W_s|\mathcal{F}_s]$$

$$\mathbb{E}[W_t|\mathcal{F}_s] = \mathbb{E}[W_t - W_s|\mathcal{F}_s] + W_s$$

As the information at $\mathcal{F}_s$ is sufficient to evaluate W_s. The first term is 0 as $W_t - W_s$ are independent of each other, and have mean 0. Therefore, we have

$$\mathbb{E}[W_t|\mathcal{F}_s] = 0 + W_s = W_s$$

Therefore, Brownian Motion satisfies the requirements of Definition 2.13 and is Martingale. □

Now, we will show that Brownian Motion satisfies the property of Quadratic Variation, defined below.

Definition 2.17 (Quadratic Variation). *Let $f : [0, T] \to \mathbb{R}$. The quadratic variation of f up to T is*

$$[f, f](T) = \lim_{||\Pi|| \to 0} \sum_{j=0}^{n-1} (f(t_{j+1}) - f(t_j))^2$$

Where $\Pi = \{t_0, ..., t_n\}, 0 = t_0 < ... < t_n = T$.

The property of quadratic variation is that the series above $[f, f](T)$ is finite.

Theorem 2.2. *Let W be a Brownian Motion. Then $[W, W](T) = T, \forall\ T \geq 0$ almost surely.*

Proof. Let Π be a partition of T. Let

$$Q_\Pi = \sum_{j=0}^{n-1} (W_{t_{j+1}} - W_{t_j})^2$$

be the sample quadratic variation. We will show that its' mean converges to T as $||\Pi|| \to 0$, and variance converges to 0. First, we will show that the mean converges.

Note that Q_Π is a sum of independent random normal variables, which is then an independent random normal variable. Additionally, have that for each part of the sum,

$$\text{Var}(W_{t_{j+1}} - W_{t_j}) = \mathbb{E}[(W_{t_{j+1}} - W_{t_j})^2] - \mathbb{E}[(W_{t_{j+1}} - W_{t_j})]^2$$

Note that the second term is 0 from the difference being independent, linearity of expectations, and each having mean 0. Thus, we have that

$$\text{Var}(W_{t_{j+1}} - W_{t_j}) = t_{j+1} - t_j$$

from the definition of the variance of Brownian Motion, and the independence between $W_{t_{j+1}}$ and W_{t_j}. Thus,

$$\mathbb{E}[(W_{t_{j+1}} - W_{t_j})^2] = t_{j+1} - t_j$$

Therefore, we have that

$$\mathbb{E}[Q_\Pi] = \mathbb{E}[\sum_{j=0}^{n-1} (W_{t_{j+1}} - W_{t_j})^2]$$

Linearity of expectations allows us to switch the order of the summation and expectation.

$$= \sum_{j=0}^{n-1} \mathbb{E}[(W_{t_{j+1}} - W_{t_j})^2] = \sum_{j=0}^{n-1} (t_{j+1} - t_j) = T$$

Now, we will compute the variance. Note that

$$\text{Var}((W_{t_{j+1}} - W_{t_j})^2) = \mathbb{E}[(W_{t_{j+1}} - W_{t_j})^4] + \mathbb{E}[(W_{t_{j+1}} - W_{t_j})^2]^2$$

$$= 3(t_{j+1} - t_j)^2 - (t_{j+1} - t_j)^2$$

As the 4^{th} moment of a random normal variable (which the difference of two independent random normal variables is) is the Kurtosis, which for a random normal variable is $3\sigma^4$, and $\sigma^2 = (t_{j+1} - t_j)$ from above. Thus we have

$$\text{Var}((W_{t_{j+1}} - W_{t_j})^2) = 2(t_{j+1} - t_j)^2$$

Thus,

$$\text{Var}(Q_\Pi) = \sum_{j=1}^{n-1} 2(t_{j+1} - t_j)^2$$

From (see Shreve, 2004, Remark 3.4.2), we can bound this with $||\Pi||$ linearly, which then obtains

$$\sum_{j=1}^{n-1} 2(t_{j+1} - t_j)^2 \leq \sum_{j=1}^{n-1} 2||\Pi||(t_{j+1} - t_j) = 2||\Pi||T$$

Thus, as $||\Pi|| \to 0$, we have that the variance goes to 0. Therefore,

$$\text{Var}(Q_\Pi) \leq 2||\Pi||T \to 0, ||\Pi|| \to 0$$

Thus, we have that Brownian Motion has finite quadratic variation almost surely. □

Another important to item to note, which will not be stated with proof, is the following result:

Theorem 2.3. *Let W_t be a Brownian motion. No sample path of the Brownian Motion is differentiable.*

This result is important for defining integration with Brownian Motion. Additionally, it means that sample paths of Brownian motion are another example of a function that is continuous but nowhere differentiable, like the Weierstrass function.

Now, we will demonstrate two key examples of Brownian Motion: The Brownian Bridge and Geometric Brownian Motion.

Example 2.4 (The Brownian Bridge)**.** *Let W_t be a Brownian Motion. Then, we have a Brownian bridge by the process:*

$$Y_t = W_t - tW_1$$

This results in $Y_0 = Y_1 = 0$ as $Y_0 = W_0 - 0 \cdot W_1 = 0$ and $Y_1 = W_1 - W_1 = 0$. Visually, this can be seen as below:

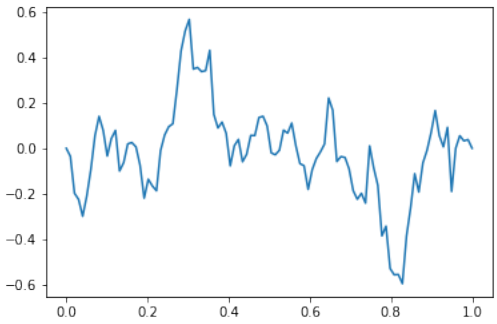


Figure 2.2: A sample path of a Brownian bridge

Example 2.5 (Geometric Brownian Motion)**.** *Using the same Brownian Motion W_t, we can define Geometric Brownian Motion as described by Black, Scholes, and Merton in the following way:*

$$S_t = e^{\mu t + \sigma W_t}$$

Where $t \geq 0, \mu, \sigma \in \mathbb{R}$. This is the exponentiation of Brownian Motion with Drift ($S'_t = \mu t + \sigma W_t$). Below is a sample path of Geometric Brownian Motion, with $\mu = .2, \sigma = .05$ on $t \in [0, 5]$ and $S_0 = 100$ with time steps $\delta t = .02$.

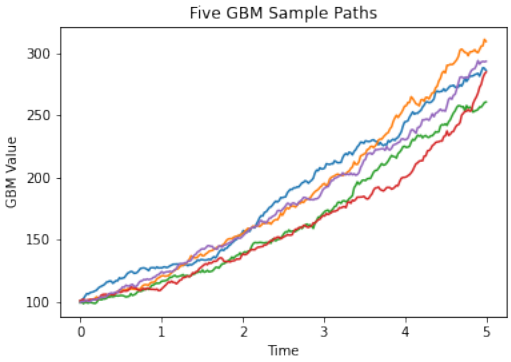


Figure 2.3: Geometric Brownian Motion (GBM) has an upward drift with high probability as $\mu > 0, \sigma > 0$, as we see in these samples.

Remark 2.2. *Geometric Brownian Motion was the key item utilized by Black and Scholes (1973) to create their Nobel Prize winning option pricing formulae.*

From this background and Brownian Motion, we can now move onto the stochastic calculus equivalent to the fundamental theorem of calculus: The Itô Integral and Itô's Lemma.

2.7 The Itô Integral and Itô's Lemma

Now, we will present two of the most important results of Stochastic Calculus: The Itô Integral and Itô's Lemma. Itô's Lemma serves as Stochastic Calculus' version of the fundamental theorem of calculus, which uses the Itô integral. Before continuing, we make a key distinction about Stochastic Calculus. Stochastic Calculus is heavily focused on the integration side, due to the lack of differentiability of sample paths of Brownian Motion. As such, Itô Calculus is sometimes referred to as an "integral" calculus. First, we will define the Itô integral as a limiting sum, similar to a Riemann Integral:

Definition 2.18 (Itô Integral)**.** *Let Π be a partition between $[s,t]$ of the form $\{s = t_0, ..., t_n = t\}$. Let W_t represent a Brownian Motion. Let $\Delta(t)$ represent a function that is constant on $[t_i, t_{i+1})$, but it is a predictable random variable. Consider the sum below:*

$$I(t) = \sum_{i=0}^{n-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i}) + \Delta(t_n)(W_t - W_{t_n}) \tag{2.4}$$

Then, as we allow $||\Pi|| \to 0$, we have that

$$I(t) = \int_s^t \Delta(u) dW_u \tag{2.5}$$

which is known as an Itô integral.

It is possible to define this integral more generally, but the version above is sufficient for our work.

Remark 2.3 (Properties of the Itô Integral)**.** *Many other properties can be derived for Itô integrals. Here, we list a few from (see Shreve, 2004, 4.3.1) The Itô integral $I(t)$ satisfies many of the following properties:*

(i) As a function of the upper limit of integration t, $I(t)$ has continuous sample paths

(ii) For each t, $I(t)$ is $\mathcal{F}_t$ measurable.

(iii) For $I(t), I'(t)$ as two Itô integrals with the same Brownian Motion, we have

$$I(t) + I'(t) = \int_0^t \Delta(u) + \Delta'(u) dW_u,$$

which means that Itô integrals are linear.

(iv) For some constant c, we have

$$cI(t) = \int_0^t c\Delta(u) dW_u$$

(v) We have Itô isometry, meaning

$$\mathbb{E}[I^2(t)] = \mathbb{E}[\int_0^t \Delta^2(u) du]$$

(vi) It has quadratic variation $[I, I](t) = \int_0^t \Delta^2(u) du$

Theorem 2.4. *The Itô integral has $\mathbb{E}[I(t)] = 0$ and it is a martingale with respect to the natural Brownian Motion filtration $\mathcal{F}_t$.*

Proof. We will first prove that $I(t)$ is a martingale, and use this result to show that it's expectation is 0.

In order to a martingale, we must have that $I(t)$ has finite expectation, it is adapted to the filtration $\mathcal{F}_t$, and that $\mathbb{E}[I(t)|\mathcal{F}_s] = I(s)$ for $s < t$. First, note that

$$\mathbb{E}[|I(t)|] < \infty$$

holds from the Isometry property, given in 2.3. Adaptedness comes from the fact that the integral is defined on Brownian Motion, and that we are using the filtration $\mathcal{F}_t$ adapted to the Brownian Motion. Since these variables are $\mathcal{F}_t-$ measurable, it holds that $I(t)$ must be $\mathcal{F}_t-$ measurable as well. Lastly, we will show that the martingale property holds.

First, consider the case where $s < t$ and $s \in [t_{k-1}, t_k]$ for some t_k's in the partition Π. In this case, we have that taking the conditional expectation

$$\mathbb{E}[I(t)|\mathcal{F}_s] = \mathbb{E}[\sum_{i=0}^{n-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i}) + \Delta(t_n)(W_t - W_{t_n})|\mathcal{F}_s]$$

Applying linearity of expectations and breaking the sum into two parts, we have

$$\begin{aligned}=&\mathbb{E}[\sum_{i=0}^{k-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i}) + \Delta(t_n)(W_t - W_{t_n})|\mathcal{F}_s] \\ &+ \mathbb{E}[\sum_{i=k}^{n-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i}) + \Delta(t_n)(W_t - W_{t_n})|\mathcal{F}_s]\end{aligned}$$

For the first part, we have that we can remove the conditional expectation as these are all the t_i's part of $\mathcal{F}_s$. For the second summation, we can remove $\Delta(t_i)$ as it is adapted, and obtain

$$\begin{aligned}=&\sum_{i=0}^{k-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i})\Delta(t_n)(W_t - W_{t_n}) + \\ &+ \Delta(t_i)\mathbb{E}[\sum_{i=k}^{n-1}(W_t - W_{t_n})|\mathcal{F}_s]\end{aligned}$$

Note as we are only in $\mathcal{F}_s$, the second part of the summation is 0 due to it being the conditional expectation of Brownian Motion, which has mean 0. Therefore, we have that

$$\mathbb{E}[I(t)|\mathcal{F}_s] = \sum_{i=0}^{k-1} \Delta(t_i)(W_{t_{i+1}} - W_{t_i}) + \Delta(t_n)(W_t - W_{t_n}) = I(s)$$

For the other cases, a similar argument applies when taking the conditional expectations. Thus $I(t)$ is a martingale.

From this, we have that the expectation function is constant with respect to the filtration $\mathcal{F}_t$, and we have that $I(0) = 0$, this means that

$$\mathbb{E}[I(t)] = \mathbb{E}[I(0)] = 0$$

□

With Itô integrals, we can then define Itô's Lemma.

Theorem 2.5 (Itô's Lemma (Simple))**.** *Let $f \in C^\infty(\mathbb{R})$, and W_t a Brownian Motion. Then, we have that*

$$f(W_t) - f(W_s) = \int_s^t f'(W_x)\ dW_x + \frac{1}{2}\int_s^t f''(W_x)\ dx,\ \ s < t \tag{2.6}$$

is a simple form of Itô's Lemma.

Proof. Let $\Pi = \{0 = t_0, ..., t_i, ..., t_n = t\}$ be a partition of $[s, t]$, indexed by x_i. Recalling the Taylor expansion for f, we have that

$$f(x_{i+1}) - f(x_i) = f'(x_i)(x_{i+1} - x_i) + \frac{1}{2}f''(x_i)(x_{i+1} - x_i)^2 + \text{stuff},$$

where the "stuff" are higher order terms of the Taylor expansion. Now, since we are concerned between the difference $f(W_t) - f(W_s)$, we have that this is equal to

$$f(W_t) - f(W_s) = \sum_{i=0}^{n-1}(f(W_{t_{i+1}}) - f(W_{t_i}))$$

$$\begin{aligned} f(W_t) - f(W_s) = &\sum_{i=0}^{n-1}(f'(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i})) \\ &+ \frac{1}{2}\sum_{i=0}^{n-1}(f''(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i}))^2 \\ &+ \text{higher order terms} \end{aligned}$$

As $||\Pi|| \to 0$, the difference $f(W_t) - f(W_s)$ does not change. However, right hand hand side does. Taking the limit, we obtain:

$$\begin{aligned} f(W_t) - f(W_s) = &\lim_{\Pi\to 0}\sum_{i=0}^{n-1}(f'(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i})) \\ &+ \lim_{\Pi\to 0}\frac{1}{2}\sum_{i=0}^{n-1}(f''(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i}))^2 \\ &+ \lim_{\Pi\to 0}\text{higher order terms} \end{aligned}$$

Note that due to quadratic variation, the second term becomes an ordinary integral:

$$\begin{aligned} f(W_t) - f(W_s) = &\lim_{\Pi\to 0}\sum_{i=0}^{n-1}(f'(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i})) \\ &+ \frac{1}{2}\int_s^t f''(W_x)\ dx \\ &+ \lim_{\Pi\to 0}\text{higher order terms} \end{aligned}$$

Thus, we must resolve what occurs with the first term, and with the higher order values. For the higher order values, as the quadratic variation is finite, further powers of the difference of Brownian Motion have limit 0 as $||\Pi|| \to 0$ (Shreve, 2004). Thus, we have

$$\begin{aligned} f(W_t) - f(W_s) = &\lim_{\Pi\to 0}\sum_{i=0}^{n-1}(f'(W_{t_{i+1}}))((W_{t_{i+1}}) - (W_{t_i})) \\ &+ \frac{1}{2}\int_s^t f''(W_x)\ dx \end{aligned}$$

This first term converges to an Itô Integral, which is an integral with respect to the Brownian Motion. Therefore, we have

$$\begin{aligned} f(W_t) - f(W_s) = &\int_s^t f'(W_x)dW_x \\ &+ \frac{1}{2}\int_s^t f''(W_x)\ dx, \end{aligned}$$

which is what we wanted to show, which concludes the proof. □

This form of the Itô's Lemma is considered to be the simple case. There are many other cases where the function f used has different assumptions on it's behavior. However, these versions of the Lemma are primarily for the cases of multiple variables and higher dimensions.

This Lemma now allows us to evaluate Integrals with Brownian Motion indirectly, as we have access to evaluating the function at a given Brownian Motion variable, as well as a deterministic integral that can be integrated using more standard analytical techniques.

Example 2.6. *With Itô's lemma, we compute the difference*

$$f(W_t) - f(W_s)$$

where $f(x) = x^2$, $s < t$, and W_t is a Brownian Motion. We know that $f'(x) = 2x$ and $f''(x) = 2$. Applying Itô's Lemma, we have

$$W_t^2 - W_s^2 = \int_s^t 2W_x dW_x + \int_s^t dx$$

From here, we have that

$$W_t^2 - W_s^2 = 2\int_s^t W_x dW_x + (t - s)$$

Therefore, we have

$$\frac{W_t^2 - W_s^2 - t + s}{2} = \int_s^t W_x dW_x$$

Additionally, if $s = 0$, we then have

$$\frac{1}{2}(W_t^2 - t) = \int_0^t W_x dW_x$$

Itô's Lemma and its generalizations have immense importance to stochastic calculus, and mathematical finance as a whole as well. Advanced techniques of stochastic calculus built off of the building blocks above have allowed for modeling of markets under fewer assumptions than the Black Scholes Model, thus more accurately modeling reality.

2.8 (Forward) Stochastic Differential Equations

Using Itô Integrals and Itô's Lemma, we can now construct the primary object of study for this report: stochastic differential equations (SDEs).

SDEs can be seen as the stochastic analogy to the ordinary differential equation (ODE). Instead, we have that the entire equation is driven by a stochastic process. Most popularly, we have the Itô SDE, defined below:

Definition 2.19 (Itô Process X_t)**.** *Using a Brownian Motion W_t, we can define the Itô stochastic differential equation as an integral equation (Mikosch, 1998):*

$$X_t = X_0 + \int_0^t a(s, X_s)ds + \int_0^t b(s, X_s)dW_s, 0 \leq t \leq T \tag{2.7}$$

Here, Brownian motion is the driving process of the Itô SDE (2.7). Additionally, it is important to note that as stochastic calculus is an integral calculus, using the term differential equation is a bit of a mathematical "slight of hand." Much like ODEs and PDEs, there are analogous versions of weak and strong solutions to be found in SDEs. Strong and weak solutions to SDEs are known as *diffusions*.

Definition 2.20 (Strong Solution)**.** *A strong solution to the Itô stochastic differential equation is a stochastic process $X = (X_t, t \in [0, T])$ such that*

(i) X is adapted to the filtration generated by Brownian Motion

(ii) The integrals defined in Equation (2.7) *are well defined in the Riemann, Lebesgue, or Itô sense respectively.*

(iii) X is a function of underlying Brownian sample paths prescribed via coefficient functions $a(t,x)$ and $b(t,x)$

Sometimes, the specific path behavior is not of interest, and the goal is to gain knowledge about the distribution of the process X. This requires finding a Brownian Motion that is sufficient to solve the SDE with the given initial condition X_0 as well as coefficient functions. This is known as a *weak* solution.

Remark 2.4. *Note that if $a = 0, b = 1$ we have that*

$$X_t = X_0 + \int_0^t dW_s$$

More generally, if a, b are continuous and Lipsschitz, Equation (2.7) *has a unique strong solution on $[0,T]$.*

2.9 The Feynman-Kac Formula

Below, we present the Feynman-Kac Formula as a mean to link between SDEs and nonlinear PDEs from (see Shreve, 2004, chapter 6.4).

Definition 2.21. *Consider the SDE of the following form:*

$$dX(u) = \beta(u, X(u))du + \gamma(u, X(u))dW(u) \tag{2.8}$$

Let $h(y)$ be a Borel measurable function, and fix $T > 0$. Define

$$g(t,x) = \mathbb{E}[h(X(T))|X(t) = x] \tag{2.9}$$

for a given $t \in [0,T]$. Assume further that $\mathbb{E}[h(X(T))|X(t) = x] < \infty, \forall t, x$. g satisfies the following PDE:

$$g_t(t,x) + \beta(t,x)g_x(t,x) + \frac{1}{2}\gamma^2(t,x)g_{xx}(t,x) = 0 \tag{2.10}$$

Note that this holds under certain assumptions for β, and γ. Typically, what is important is that β and γ are known, and that they are Lipschitz continuous. For a proof, we refer the reader to (see Shreve, 2004, lemma 6.4.2).

2.10 Important Classes of Random Variables

Below, we list three important classes of random variables that we will utilize throughout this work.

1. $\mathbb{L}^2(\mathbb{R})$:= the space of random variables $\xi : \omega \to \mathbb{R}$ such that $\mathbb{E}[|\xi|^2] < \infty$
2. $\mathbb{S}^2(\mathbb{R})$:= space of adapted, left continuous and right limited (cadlag) processes $Y : \Omega \times [0,T] \to \mathbb{R}$ such that $\mathbb{E}[\sup_{t\in[0,T]} |Y(t)|^2] < \infty$
   - A function f is Cadlag: $\forall t > 0, \forall \epsilon > 0, \exists \delta > 0$ such that $|f(t) - f(t-\delta)| < \epsilon$ (right continuous) and $\lim_{h\downarrow 0} f(t+h) = f(t)$ (left limited)
3. $\mathbb{H}^2(\mathbb{R})$:= square intergrable processes Z_t such that $\mathbb{E}[\int_0^T |Z(t)|^2 dt] < \infty$

These are spaces are vital for the results we provide in the following chapters.

2.11 Further Reading

For further reading on Stochastic Calculus, we refer the reader to (Mikosch, 1998), (see Øksendal, 2003, chapters 2,3,4, & 5), and (see Shreve, 2004, chapters 1, 2, 3, & 4).

Chapter 3

Backwards Stochastic Differential Equations (BSDE)

3.1 Introduction

Throughout this section, we introduce the general theory of backwards stochastic differential equations. In particular, we focus on the existence and uniqueness of BSDEs under certain assumptions. Additionally, we provide an example of a BSDE as well as it's solution. In particular, we focus on a BSDE of the following form:

Definition 3.1. *Let $T \in \mathbb{R}^+$, and let $\xi \in \mathbb{L}^2(\mathbb{R})$ (the terminal condition) and $f : [0,T] \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, we have a backwards stochastic differential equation (BSDE) is of the form*

$$Y(t) = \xi + \int_t^T f(s, Y, Z)ds - \int_t^T Z(s)dW(s), \tag{3.1}$$

Note that the solution $(Y, Z) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$ exists and is unique for Equation 3.1.

This is adapted from Delong (2013). Below, we provide proof of existence and uniqueness for certain generator functions f.

3.2 Theoretical Properties

Below, we provide theoretical properties to BSDEs. In particular, we focus on the existence and uniqueness of solutions to BSDEs with certain generator functions.

3.2.1 Existence and Uniqueness With 0 Generator

To begin, we will first assume that $f = 0$. Thus, we are concerned with a BSDE of the following form:

Definition 3.2. *Let $\xi \in \mathbb{L}^2(\mathbb{R})$. We then have a BSDE with 0 generator*

$$Y(t) = \xi - \int_t^T Z(s)dW(s), \tag{3.2}$$

where W is a Brownian Motion and Z is a stochastic processes adapted to W.

From this definition, we can state the following result:

Theorem 3.1. *The BSDE from Definition 3.2 has a unique solution $(Y, Z) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$. Furthermore, we can represent the solution as*

$$Y(t) = \mathbb{E}[\xi|\mathcal{F}_t], 0 \leq t \leq T \tag{3.3}$$

and Z can be derived from the representation:

$$\xi = \mathbb{E}[\xi] + \int_0^T Z(s)dW(s). \tag{3.4}$$

Proof. First, note that $\mathbb{E}[\xi|\mathcal{F}_t]$ is a martingale. To see this, we will show it satisfies the three requirements.

First, note that since $\xi \in \mathbb{L}^2(\mathbb{R})$, we have that $\mathbb{E}[|\xi|] < \infty$. Through the projection property (see Mikosch, 1998, 1.4.5), we have that $\mathbb{E}[\xi|\mathcal{F}_t]$ is also finite, which is the first property. The second portion of $Y(t)$ is a stochastic integral, which has expectation 0.

Next, we wish to show that $\mathbb{E}[\xi|\mathcal{F}_t]$ is adapted to $\mathcal{F}_t$. Note that by definition of conditional expectation (which exists and is unique for the BSDE above), we have that $\mathbb{E}[\xi|\mathcal{F}_t]$ is $\mathcal{F}_t$-measurable, which means that it is adapted to the filtration.

Lastly, we will show that the martingale property holds. To do this, we note that

$$\mathbb{E}[\mathbb{E}[\xi|\mathcal{F}_t]|\mathcal{F}_s] = \mathbb{E}[\xi|\mathcal{F}_s], 0 \leq s < t \leq T$$

from the *tower property.*

Thus, $\mathbb{E}[\xi|\mathcal{F}_t]$ is a martingale.

Since $\mathbb{E}[\xi|\mathcal{F}_t]$ is a martingale, we can apply the martingale representation theorem (see Shreve, 2004, 5.3.1) on the process Y to obtain:

$$Y(t) = Y(0) + \int_0^t Z(s)dW(s), 0 \leq t \leq T$$

for some process Z. Note then that by substituting $T = t$, we obtain

$$Y(T) = Y(0) + \int_0^T Z(s)dW(s).$$

Since ξ is the terminal condition, we have that $Y(T) = \xi$. Thus, we get

$$\xi = Y(0) + \int_0^T Z(s)dW(s).$$

Taking the expectation, we have

$$\mathbb{E}[\xi] = \mathbb{E}[Y(0) + \int_0^T Z(s)dW(s)].$$

Applying linearity of expectations and the fact that $\mathbb{E}[Y(0)] = Y(0)$, as well as the fact that the expectation of an Itô integral is 0 (as it is a martingale), we get that $Y(0) = \mathbb{E}[\xi]$. Substituting this, we obtain:

$$\xi = \mathbb{E}[\xi] + \int_0^T Z(s)dW(s).$$

From the martingale representation theorem and splitting the integral into 2 parts, we can express the process as below:

$$Y(T) = \xi = Y(0) + \int_0^t Z(s)dW(s) + \int_t^T Z(s)dW(s).$$

From here, we can take the conditional expectation with respect to $\mathcal{F}_t$ to obtain:

$$Y(t) = \mathbb{E}[\xi|\mathcal{F}_t] = \mathbb{E}[Y(0)|\mathcal{F}_t] + \mathbb{E}[\int_0^t Z(s)dW(s)|\mathcal{F}_t] + \mathbb{E}[\int_t^T Z(s)dW(s)|\mathcal{F}_t].$$

Since $Y(0)$ is known at t, we have

$$\mathbb{E}[\xi|\mathcal{F}_t] = Y(0) + \mathbb{E}[\int_0^t Z(s)dW(s)|\mathcal{F}_t] + \mathbb{E}[\int_t^T Z(s)dW(s)|\mathcal{F}_t].$$

The first integral is only up to time t, and as

$$\int_0^t Z(s)dW(s)$$

is adapted to the filtration $\mathcal{F}_t$, the conditional expectation is just the original integral. Therefore, we have that

$$\mathbb{E}[\xi|\mathcal{F}_t] = Y(0) + \int_0^t Z(s)dW(s) + \mathbb{E}[\int_t^T Z(s)dW(s)|\mathcal{F}_t].$$

Note for the second integral, it is no longer adapted as we are going from t to T. Note, note that the Itô integral is independent of the filtration at this time. This is is because we consider the integral as a sum, and consider the series

$$\lim_{||\Pi||\to 0} \sum_{i=0}^{n} Z_{t_i}(W_{t_{i+1}} - W_{t_i}).$$

Note that the Brownian increments are always independent of Z_{t_i} throughout the time period, and thus each term is independent. Since each term is independent from the filtration, the Itô Integral (the limit) is independent. Therefore, the conditional expectation of the integral here is 0. As we have that

$$Y(t) = Y(0) + \int_0^t Z(s)dW(s)$$

from the Martingale Representation Theorem. And at time T,

$$Y(T) = Y(0) + \int_0^t Z(s)dW(s) + \int_t^T Z(s)dW(s).$$

Substituting the first part for $Y(t)$, we have

$$\xi = Y(T) = Y(t) + \int_t^T Z(s)dW(s).$$

Therefore,

$$Y(t) = \mathbb{E}[\xi|\mathcal{F}_t] = \xi - \int_t^T Z(s)dW(s).$$

Thus, we have the existence of a solution (Y, Z), as in the form described in the theorem. Now, we will show that this solution is unique.

Assume by contradiction that there exists a solution $(Y', Z'$ where $Y' \neq Y$ and $Z' \neq Z)$ and it is a solution to

$$Y(t) = \xi - \int_t^T Z(s)dW(s).$$

From here, we can express the solution at time t in terms of each solution:

$$Y(t) = \xi - \int_t^T Z(s)dW(s)$$

$$Y'(t) = \xi - \int_t^T Z'(s)dW(s)$$

Then, we can take the expectation of both of them. This results in:

$$\mathbb{E}[Y(t)] = \mathbb{E}[\xi - \int_t^T Z(s)dW(s)]$$

$$\mathbb{E}[Y'(t)] = \mathbb{E}[\xi - \int_t^T Z'(s)dW(s)]$$

As $Y(t)$ is known at t (meaning that $Y(t)$ is $\mathcal{F}_t$ measurable with respect to the filtration $\mathcal{F}_t$), we can drop the expectation on the left. This results in

$$Y(t) = \mathbb{E}[\xi - \int_t^T Z(s)dW(s)]$$

$$Y'(t) = \mathbb{E}[\xi - \int_t^T Z'(s)dW(s)]$$

Now, we can apply linearity of expectations on each to obtain:

$$Y(t) = \mathbb{E}[\xi] - \mathbb{E}[\int_t^T Z(s)dW(s)]$$

$$Y'(t) = \mathbb{E}[\xi] - \mathbb{E}[\int_t^T Z'(s)dW(s)]$$

Note we are taking expectations of two Itô stochastic integrals, which are known to have expectation 0 as Z, Z' are square integrable, which was discussed in Theorem 2.4. Thus, we have

$$Y(t) = \mathbb{E}[\xi] = Y'(t), \forall t \in [0, T]$$

This holds almost (surely/everywhere) with respect to $d\mathbb{P} \bigotimes ds$.

From here, we wish to show that $Z = Z'$ in the same manner as above. Note that since $Y(t) = Y'(t)$, we have that

$$0 = Y(t) - Y'(t).$$

This also holds if we square both sides

$$0 = (Y(t) - Y'(t))^2.$$

And additionally, it holds if we take the expectation

$$0 = \mathbb{E}[(Y(t) - Y'(t))^2].$$

Now, we can apply the martingale representation theorem to Y and Y' to obtain

$$0 = \mathbb{E}[((Y(0) + \int_0^t Z(s)dW(s)) - (Y'(0) + \int_0^t Z'(s)dW(s)))^2].$$

Note that from above, we have that $Y(0) = Y'(0)$, so we can cancel those terms. Therefore, we have

$$0 = \mathbb{E}[(\int_0^t Z(s)dW(s) - \int_0^t Z'(s)dW(s))^2].$$

And now, by linearity of integration, we have

$$0 = \mathbb{E}[(\int_0^t Z(s) - Z'(s)dW(s))^2].$$

Here, we can apply Itô's lemma since Z and Z' are both square integrable to obtain:

$$0 = \mathbb{E}[\int_0^t (Z(s) - Z'(s))^2 ds].$$

Now, note that this integral is 0, and that the integrand is always greater than or equal to 0 as it is squared. Therefore, a classical result from Measure Theory implies that

$$Z(s) - Z'(s) = 0$$

In other words, we have that $Z = Z'$ almost everywhere with respect to $d\mathbb{P} \bigotimes ds$.

Therefore, we have a contradiction, which means that the assumption that the solutions were the same is incorrect, and the solution to the BSDE is infact unique. □

Remark 3.1. *Note that if f only depended on t, the results above do not change. This is due to the fact that any conditional expectations taken will simply allow $f(t)$ to be added where necessary. In particular, anywhere ξ is written, $a + \int_t^T f(s)ds$ is added. This is because $f(s)$ has no stochastic element with it, and is adapted where necessary. Thus, Y becomes:*

$$Y(t) = \mathbb{E}\left[\xi + \int_t^T f(s)|\mathcal{F}_t\right], 0 \leq t \leq T \tag{3.5}$$

Furthermore, the solution would not change if f depended on other random variables that were not related to Y or Z.

3.2.2 BSDEs with More Complex Generators

Now, we can assume that f is of the form $f(s, Y, Z)$, where Y and Z are defined as above in Definition 3.1. Now, we can provide existence and uniqueness for this class of BSDE.

Theorem 3.2. *Suppose that a BSDE of Equation 3.1 is given. Then, there exists a unique solution pair (Y, Z).*

Proof. We begin by construct a sequence $\{(Y^n, Z^n)\}_{n\in\mathbb{N}}$ such that each (Y^n, Z^n) is a unique solution to the BSDE below

$$Y^{n+1}(t) = \xi + \int_t^T f(s, Y^n(s), Z^n(s))\ ds - \int_t^T Z^{n+1}(s)dW(s).$$

Then, we will show that the sequence we construct will converge to (Y, Z), being the unique solution to the original BSDE.

Let $(Y^0(t), Z^0(t)) = (0, 0)$ for $t \in \mathbb{R},\ Z, Y \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$. Now, define the recursive sequence of BSDEs

$$Y^{n+1}(t) = \xi + \int_t^T f(s, Y^n(s), Z^n(s))ds - \int_t^T Z^{n+1}(s)dW(s),$$

where $(Y^n, Z^n) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$. therefore, we have

$$\mathbb{E}[\int_0^T |f(t, Y^n(t), Z^n(t))|^2 dt] \leq 2\mathbb{E}[\int_0^T |f(t, 0, 0)^2|dt] + 2K(T||Y^n||^2 + ||Z^n||^2_{\mathbb{H}^2}) < \infty.$$

For the first part of the inequality, recall that we have

$$\mathbb{E}[\int_0^T |f(t, Y^n(t), Z^n(t))|^2 dt] = \mathbb{E}[\int_0^T |f(t, Y^n(t), Z^n(t)) - f(t, 0, 0) + f(t, 0, 0)|^2 dt]$$

by adding in a form of 0. We can then break this into two terms and recall that $(x + y)^2 \leq 2x^2 + 2y^2$ for all $x, y \in \mathbb{R}$ as follows to obtain:

$$\leq 2\mathbb{E}[\int_0^T |f(t, 0, 0)^2|] + 2\mathbb{E}[\int_0^T |f(t, Y^n(s), Z^n(s)) - f(t, 0, 0)^2|dt]$$

Further, we have that the second part of the right hand side is Lipschitz with a constant K, and thus we obtain:

$$\leq K(||Y^n||^2 + ||Z^n||^2)$$

For the second half of the first inequality, we have the first part is finite by assumption, and the second part is finite due to the spaces that we defined for the Y^n, Z^n processes, as they are square intergrable.

From earlier, we know that there exists a unique solution $(Y^{n+1}, Z^{n+1}) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$ independent of $f(t, Y^n, Z^n)$ that solves the iterative BSDE.

We will show that this sequence (Y^n, Z^n) is Cauchy, and thus converges to a unique limit, and as $\mathbb{S}^2(\mathbb{R})$ and $\mathbb{H}^2(\mathbb{R})$ are complete spaces, we have that the sequences each converge in their respective space and have unique limits.

Consider Lemma 3.3.1 (Delong, 2013) and $Y^{n+1}, Z^{n+1}, Y^n, Z^n$. We are able to bound following differences which hold for all $\rho > 0, T > 0$.

$$||Y^{n+1} - Y^n||^2_{\mathbb{S}^2} + ||Z^{n+1} - Z^n||^2_{\mathbb{H}^2}$$
$$\leq K(\mathbb{E}[\int_0^T |f(t, Y^n, Z^n) - f(t, Y^{n-1}, Z^{n-1})|^2 dt])$$
$$+K'(\mathbb{E}[\int_0^T |f(t, Y^n, Z^n) - f(t, Y^{n-1}, Z^{n-1})|^2 dt])$$

Let $m, n \in \mathbb{N}$ such that $n \neq m$ be given. Without loss of generality, let $n < m$. Then, consider the equation

$$||Y^n - Y^m|| + ||Z^n - Z^m|| = ||Y^n + Y^{n+1} - Y^{n+1} + Y^{n+2} - Y^{n+2} + ... + Y^{m-1} + Y^m||$$
$$+||Z^n + Z^{n+1} - Z^{n+1} + Z^{n+2} - Z^{n+2} + ... + Z^{m-1} + Z^m||$$

Applying the triangle inequality yields:

$$||Y^n - Y^m|| + ||Z^n - Z^m|| \leq \sum_{i=n}^{m} ||Y^{i+1} - Y^i|| + ||Z^{i+1} - Z^i||.$$

Now, on each term, we can equation 3.11 from Delong to obtain:

$$\leq \frac{1}{\rho} \sum_{i=n}^{m} \mathbb{E}[\int_0^T e^{\rho t} |f(t, Y^i, Z^i) - f(t, Y^{i+1}, Z^{i+1})|^2 dt]$$

Which then creates a geometric series with ρ, and therefore this converges. Thus, we have that $(Y^n, Z^n) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$ is Cauchy in $\mathbb{H}^2$.

Recall that $(Y^n, Z^n) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$ converges to $(Y, Z) \in \mathbb{S}^2(\mathbb{R}) \times \mathbb{H}^2(\mathbb{R})$. Therefore, we want to show that

$$Y^n(t) = \xi_T + \int_t^T f(s, Y^n(s), Z^n(s)) ds - \int_t^T Z^n(s) dW(s)$$

converges in $\mathbb{H}^2 \times \mathbb{S}^2$ to

$$Y(t) = \xi_T + \int_t^T f(s, Y(s), Z(s)) ds - \int_t^T Z(s) dW(s)$$

Subtracting, we have that

$$|Y(t) - Y_n(t)| = |\int_t^T f(s, Y^n(s), Z^n(s)) ds - \int_t^T f(s, Y(s), Z(s)) ds$$
$$-(\int_t^T Z(s) dW(s) - \int_t^T Z^n(s) dW(s))|$$

Thus, we can apply the triangle inequality to obtain that

$$\leq |\int_t^T f(s, Y^n(s), Z^n(s)) ds - \int_t^T f(s, Y(s), Z(s)) ds| + |(\int_t^T Z(s) dW(s) - \int_t^T Z^n(s) dW(s))|$$

If these two terms both converge in $\mathbb{H}^2$, then we have that the solution exists and is unique to the general BSDE.

For the first term, from Theorem 2.2.1 of Delong and the BDG inequality (see Theorem Protter, 2005, IV.48), we have that Let $M := \int_t^T f(s, Y^n(s), Z^n(s)) ds - \int_t^T f(s, Y(s), Z(s)) ds$

$$\mathbb{E}[\sup_{t \in [0,T]} (|\int_t^T f(s, Y^n(s), Z^n(s)) ds - \int_t^T f(s, Y(s), Z(s)) ds|)^2]$$

$$\leq K_1 \mathbb{E}[[M, M](t)]$$

which is the quadratic variation of a deterministic integral, which has value 0. Therefore, we have that the first difference converges in $\mathbb{H}^2$.

For the second term. We have that

$$\mathbb{E}[\sup_{t\in[0,T]} \int_t^T Z(s)dW(s) - \int_t^T Z^n(s)dW(s)] \leq$$

$$\leq K_1 \mathbb{E}[[\int Z^n(s) - Z(s)dW(s), \int Z^n(s) - Z(s)dW(s)](T)$$

$$-[\int Z^n(s) - Z(s)dW(s), \int Z^n(s) - Z(s)dW(s)](t)]$$

$$= K_1 \mathbb{E}[\int_t^T (Z^n(s) - Z(s))^2 ds]$$

Again by the BDG inequality and quadratic variation. Then, the last item goes to 0 as $Z^n \to Z$ in $\mathbb{S}^2$. □

3.3 Dynamic Entropic Risk Measures

A common example of a BSDE is known as dynamic entropic risk measure. Risk measures can be developed axiomatically, and then we can apply these properties to solve certain BSDEs. Below, we present an example from Proposition 3.12 in Carmona (2009).

Example 3.1 (BSDE Example)**.** *Let $\gamma > 0, \gamma \in \mathbb{R}$ be given. Suppose we are given a BSDE of the form:*

$$dY_t = Z_t dW_t - \frac{1}{2\gamma}||Z_t||^2 dt, Y_T = \xi_T \tag{3.6}$$

Then, we have that $Y_t = \gamma \ln(M_t)$ is the solution, where

$$M_t(\xi_T) = \mathbb{E}[\exp(-\frac{1}{\gamma}\xi_T)|\mathcal{F}_t] \tag{3.7}$$

Lemma 3.1. *With M_t as defined above, we have that*

$$dM_t = \frac{1}{\gamma} M_t Z_t dW_t$$

Where dW_t is a Brownian Motion, and Z_t is a square intergrable stochastic process bounded away from 0.

Proof. From the Martingale Representation Theorem, we have that

$$M_t = M_0 + \int_0^t \Gamma_u dW_u.$$

In differential form, this is

$$dM_t = \Gamma_u dW_u.$$

We can rewrite this using a form of 1 as follows:

$$dM_t = \frac{1}{\gamma} M_u \gamma \frac{1}{M_u} \Gamma_u dW_u.$$

Define $Z_u := \frac{\gamma}{M_u}\Gamma_u$. We will show that Z_u is square intergrable and bounded away from 0. Note that ξ_T is bounded, and thus $-\frac{1}{\gamma}\xi_T$ is bounded. Therefore, we have that

$$e^{-\frac{1}{\gamma}\xi_T}$$

is bounded away from 0 and bounded above, and therefore we have that

$$\mathbb{E}[e^{-\frac{1}{\gamma}\xi_T}|\mathcal{F}_t]$$

preserves these properties. Due to the Martingale Representation Theorem, we have that multiplying this expectation by $\gamma\Gamma_u$ will preserve the square intergrability and bounds, proving the lemma. □

Now, we will prove that Y_t is a solution to (3.6).

Proof. Let $Y_t = \gamma \ln(M_t)$, with M_t defined as in (3.7).

Then, note that Itô's Lemma on $\gamma \ln(M_t)$ yields:

$$dY_t = \frac{\gamma}{M_t} dM_t + \frac{1}{2} \cdot -\frac{-\gamma}{M_t^2} d[M, M](t)$$

Where $d[M, M](t)$ is the differential of the quadratic variation of M. Applying the identity above, the first term becomes

$$\frac{\gamma}{M_t} \cdot (\frac{1}{\gamma} M_t Z_t dW_t) = Z_t dW_t,$$

and therefore we obtain

$$dY_t = Z_t dW_t - \frac{\gamma}{2M_t^2} d[M, M](t).$$

For the second term, we apply that

$$[M, M](t) = \int_0^t (\frac{1}{\gamma} M_t Z_t)^2 dt$$

and thus

$$d[M, M](t) = (\frac{1}{\gamma} M_t Z_t)^2 dt.$$

Applying this to the equation, we obtain

$$dY_t = Z_t dW_t - \frac{\gamma}{2M_t^2} (\frac{1}{\gamma} M_t Z_t)^2 dt$$

Simplying this becomes

$$dY_t = Z_t dW_t - \frac{1}{2\gamma} Z_t^2 dt$$

Which shows that Y_t is a solution to equation (3.6). From the results of existence and uniqueness, this is the only solution to this equation, completing the proof. □

3.4 The Nonlinear Feynman-Kac Formula

Similar to the Feynman Kac Formula, there has been work into developing *nonlinear* formulae analogous to the Feynman-Kacs formula for BSDEs. In particular, we will show a version from (E et al., 2017).

Theorem 3.3. *Let $\xi \in \mathbb{R}, Y : [0, T] \times \Omega \to \mathbb{R}$ and $Z : [0, T] \times \Omega \to \mathbb{R}^d$ be $\mathcal{F}$ adapted stochastic processes with continuous sample paths which satisfy the following BSDE:*

$$Y_t = g(\xi + W_T) + \int_t^T f(Y_s, Z_s) ds - \int_t^T \langle Z_s, dW_s \rangle_{\mathbb{R}^d}$$

Under assumptions of regularity on the behavior of f we have that the following PDE is related to the BSDE such that for all $t \in [0, T]$ it holds $\mathbb{P}$-a.s. (almost surely) that

$$Y_t = u(t, \xi + W_t) \in \mathbb{R}$$

and

$$Z_t = (\nabla_x u)(t, \xi + W_t) \in \mathbb{R}^d$$

One primary application that utilizes a nonlinear version of the Feynman-Kac formula is found in the seminal work of the Deep BSDE solver (E et al., 2017). Here, they utilize the formula to have neural networks that approximate solution to BSDEs also approximate the solution to a certain class of PDE. This has been expanded on immensely, such as in the work by (Huré et al., 2019) and (Gao et al., 2022).

Chapter 4

Reflected Backwards Stochastic Differential Equations

As a BSDE propagates backward from the terminal time T to time 0, the path is relatively unrestricted, outside of the requirements for the processes Y and Z. Sometimes it is necessary to restrict this path to be above a barrier. This occurs in certain option pricing problems in mathematical finance, which we will discuss in Chapter 6. Below, we provide a general overview for outlining the reflected BSDE (RBSDE).

4.1 BSDEs with Constraints

To apply BSDEs to other classes of problems (such as American option pricing in mathematical finance), we need to add constraints to the Y process such that it will remain above the boundary condition S_t. This is done through the following modification to a BSDE. We introduce a process S_t to serve as a constraint on the Y_t process.

Definition 4.1 (RBSDE)**.** *A reflected BSDE (RBSDE) is a BSDE of the following form:*

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s)ds + K_T - K_t - \int_t^T Z_s dW_s \tag{4.1a}$$

Where for all $0 \leq t \leq T$

$$Y_t \geq S_t, \quad (S_T \leq \xi \ \textit{almost surely}) \tag{4.1b}$$

and K_t *is a continuous, increasing process such that*

$$K_0 = 0, \quad \int_0^T (Y_t - S_t)dK_t = 0 \tag{4.1c}$$

As a RBSDE is a BSDE with a constraint, we know that RBSDEs will have a unique solution. Below, we state the key assumptions for existence and uniqueness of a RBSDE.

Theorem 4.1 (RBSDE Approximation)**.** *Under the follow assumptions (El Karoui et al., 1997)*

(i) $\xi \in \mathbb{L}^2(\mathbb{R})$

(ii) For all $(Y, Z) \in \mathbb{R} \times \mathbb{R}, t \in [0, T]$*, we have that* $f(t, Y, Z) \in \mathbb{H}^2(\mathbb{R})$

(iii) For some $K > 0$ *and for all* $Y, Y' \in \mathbb{R}, Z, Z' \in \mathbb{R}$*, we have almost surely that*

$$|f(t, Y, Z) - f(t, Y', Z')| \leq K(|Y - Y'| + |Z - Z'|),$$

which can be viewed as a Lipschitz condition.

(iv) $\mathbb{E}[\sup_{t\in[0,T]}(S_t^+)^2] < \infty$, *where* $\{S_t, t \in [0,T]\}$ *is a continuous progressively measurable real valued process (Here, progressively measurable is similar to being adapted, but slightly stronger as it allows the stopped process to be measurable). Furthermore, we assume that* $S_T \leq \xi$ *almost surely as defined above.*

(v) $Z \in \mathbb{H}^2(\mathbb{R})$

(vi) Y_t *is defined as in Definition 4.1.*

the RBSDE has a unique solution (Y_t, Z_t)

A similar iteration style approach can be taken for proving existence and uniqueness for the RBSDE. Additionally, there is another approach that could be taken, approximating the RBSDE object from a well constructed BSDE.

4.1.1 A BSDE Approximation of the RBSDE

Utilizing a BSDE with an extra term, we are able to approximate a BSDE. This is done through the following equation (see El Karoui et al., 1997, Section 6):

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s)ds + M\int_t^T (Y_s - S_t)^- ds - \int_t^T Z_s dW_s \tag{4.2}$$

Where M is a sufficiently large constant given the desired level of accuracy. Intuitively, we have that $(Y_s - S_t)^-$ acts as a "punishment" term, forcing the BSDE (4.2) to remain above the terminal condition g throughout the path $[0,T]$. For proof that this produces a unique solution of the RBSDE, we refer the reader to Section 6 of El Karoui et al. (1997). We provide an outline of the main ideas.

Definition 4.2 (Progressively Measurable)**.** *Let* $(\Omega, \mathcal{F}, \mathbb{P})$ *be a probability space, and* $\{\mathcal{F}_t | t \geq 0\}$ *be a filtration on* $\mathcal{F}$*. Let* $Y_t : [0,T] \times \Omega \to \mathbb{R}$ *be a stochastic process. We say that* Y_t *is* ***progressively measurable*** *if, for every* $t \in [0,T]$*, we have that the mapping* $(s,\omega) \mapsto Y_s(\omega)$ *is* $\mathcal{B}([0,t]) \bigotimes \mathcal{F}_t -$ *measurable, where* $0 \leq s \leq t$*.*

Here, $\mathcal{B}([0,t])$ denotes the Borel $\sigma -$ algebra. Progressively measurable processes are the largest process needed for stochastic integration. It can be thought of a similar concept to an adapt process on Ω, but with an added time element for the process.

Now, we will provide a proof sketch of Theorem 4.1 (see El Karoui et al., 1997, Section 6 for full proof).

Proof. For each $n \in \mathbb{N}$, let $\{(Y_t^n, Z_t^n) | t \in [0,T]\}$ denote the pair of progressively measurable processes valued in $\mathbb{R} \times \mathbb{R}$ such that

$$\mathbb{E}\left[\int_0^T |Z_t^n|^2 dt\right] < \infty$$

and satisfy the following BSDE:

$$Y_t^n = \xi + \int_t^T f(s, Y_s^n, Z_s^n)ds + n\int_t^T (Y_s^n - S_s)^- ds - \int_t^T Z_s^n dW_s$$

where $(\cdot)^-$ denotes the negative part of a function. As $n \to \infty$, we have that this term in the BSDE acts as a "punishment" term, forcing the solution (Y_t^n, Z_t^n) to remain above the boundary condition g. Thus, if we define

$$K_t^n = n\int_0^t (Y_s^n - S_s)^- ds,$$

then from unconstrained BSDEs, we know that for each n, we have that

$$\mathbb{E}[\sup_{t\in[0,T]} |Y_t^n|^2] < \infty$$

Now, El Karoui et al. (1997) generates a priori estimates for the solution *triple* $(Y^n, Z^m K^n)$ to show that each item is finite. Then, we can define

$$f_n(t, y, z) = f(t, y, z) + n(y - S_t)^-.$$

Therefore, we have that

$$f_n(t, y, z) \leq f_{n+1}(t, y, z).$$

Additionally, the comparison principles allow us to establish that for all $t \in [0, T]$, we have that, almost surely,

$$Y_t^n \leq Y_t^{n+1}.$$

Therefore, we can conclude that $Y_t^n \nearrow Y_t$ for all $t \in [0, T]$ almost surely. This, along with Fatou's Lemma, allows us to apply dominated convergence to conclude that

$$\mathbb{E}\left[\int_0^T (Y_t - Y_t^n)^2 dt\right] \to 0$$

as $n \to \infty$.

Other estimates then allow us to show that the sequence (Y_t^n, Z_t^n) is Cauchy, and thus convergent in this space. Additionally, we have that

$$\mathbb{E}\left[\sup_{0 \in [0,T]} |(Y_t^n - S_t)^-|^2\right] \to 0$$

as $n \to \infty$. These properties then imply that the sequence of BSDEs we created above satisfy the properties of being an RBSDE in the limit. Therefore, we have that the solution exists and is unique in the limit. □

Remark 4.1 (Connection to Optimal Stopping)**.** *RBSDEs are of particular use to a class of problems known as optimal stopping problems. These problems focus on the behavior of a given process, and finding the optimal point along the path of the RBSDE to take an action such that some value function is maximized. In Chapter 6, we will demonstrate this on the example of American Option pricing.*

Chapter 5

Numerical Implementation of (R)BSDEs

5.1 Introduction

In this section, we describe modern approaches that can numerically approximate the solutions to (R)BSDEs through Deep Learning. This is done through using feed forward neural networks as approximators for the Y and Z processes.

5.2 Neural Networks as Functional Approximators

In recent years, Neural Networks (NNs) have exploded in popularity. Feed forward neural networks have been shown to be very accurate approximators of wide classes of functions. Because of this, they are able to be utilized for many regression and classification problems. Throughout this work, we focus on a special type of neural network, known as a feed forward neural network.

A feed forward neural work structure is shown in Figure 5.1.

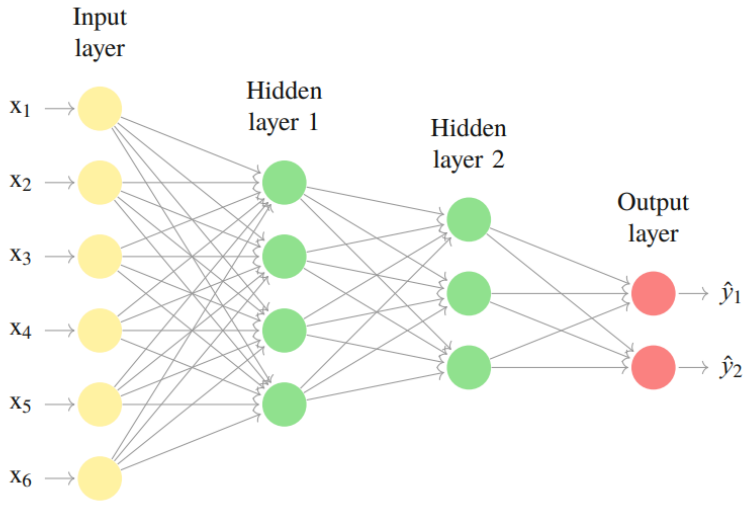


Figure 5.1: A feed forward NN with 6 inputs, 2 hidden layers, and 2 outputs from (see Dixon et al., 2020, Figure 4.1)

The first layer, known as the input layer, takes input data from the problem. Then, a number of hidden layers transform the datapoints, and feed into the output layer. Between each layer, there are parameters known as weights and biases that scale and transform the data. The process of determining these weights and biases is known as *training* the neural network.

This particular class of NN has been shown to be a very accurate approximator of many functions. This is due to a class of theorems known as universal approximation theorems. One of the first theorems in the field is from Cybenko (1989), who developed a universal approximation theorem for one layer feed forward NNs. To give a brief introduction to this universal approximation theorem, we first give some necessary definitions.

Definition 5.1 (Sigmoidal Functions)**.** *$\sigma : \mathbb{R} \to \mathbb{R}$ is sigmoidal if*

$$\sigma(t) = \begin{cases} 1 & t \to \infty \\ 0 & t \to -\infty \end{cases} \tag{5.1}$$

Definition 5.2 (Discriminatory Functions)**.** *Let I_n be the n-dimensional unit cube and $M(I_n)$ be finite, signed regular Borel measures on I_n. σ is discriminatory if for a measure $\mu \in \mathcal{M}(I_n)$, we have*

$$\int_{I_n} \sigma(y^T x + \theta) d\mu(x) = 0 \tag{5.2}$$

for all $y \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ implies that $\mu = 0$.

Theorem 5.1 (Universal Approximation Theorem (Cybenko, 1989))**.** *If σ is a continuous discriminatory function, then a sum of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \tag{5.3}$$

where $\alpha_j, \theta_j \in \mathbb{R}$ is dense in $C(I_n)$ (continuous functions on I_n). Given $\varepsilon > 0, f \in C(I_n)$, there exists values of α_j and θ_j, $j = 1, ..., N$ such that

$$|G(x) - f(x)| < \varepsilon, \ \forall \ x \in I_n \tag{5.4}$$

It is worth noting is that this is just an existence proof for the values α_j and θ_j. In deep learning, α_j and θ_j are called weights and biases. The universal approximation theorem does not provide insights on how to compute the "correct" weights and biases to approximate a given function of concern. Therefore, algorithms have been developed to approximate the values of the weights and biases. It is quite common is to minimize a loss function (such as mean squared error (MSE)) utilizing a stochastic gradient descent (SGD) type algorithm to find sufficient parameters.

5.2.1 Designing Feed Forward Neural Networks

For this work, we have created a multilayer feed forward neural network for approximating solutions to (R)BSDEs. Here, we provide a general overview for designing these neural networks for these problems. The following overview of the training process is adapted from (Negrini, 2022, section 2.1).

Algorithm 1 Designing A Feed Forward Neural Network

1. Choose a programming library for the neural network, such as TensorFlow or PyTorch.
2. Obtain a training dataset for the (R)BSDE
3. Determine the input and output dimensions, and decide on the number of neurons and hidden layers.
4. Choose the loss function, such as mean squared error.
5. Choose the Optimizer algorithm, such as SGD.
6. Train the network using the training data and the optimizer algorithm to find the desired weights and biases.
   (a) Iteratively update *hyperparameters* through observing the results on the training set to obtain the best possible results.

Below, we expand on the steps outlined above.

Choosing A Programming Architecture

A popular programming language for programming neural networks is python 3 (Van Rossum and Drake, 2009). Within python, there are two dominate libraries to choose from: TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019). For this work, we utilized PyTorch. Both have the ability to create feed forward neural networks, and it is primarily a matter of preference for the user.

Generating Training Data

In order to find optimal neural network parameters with the optimization algorithm, the network needs a training dataset. A training dataset contains datapoints that the neural network can be tested against for the loss function.

For our work in Chapter 6, we will be utilizing sample paths for a stock under a Black Scholes and stochastic volatility model. For other problems, this can be sales data, survey results, or biomedical data. For more on types of training data and thus application areas, we refer the reader to (see Alzubaidi et al., 2021, Applications of Deep Learning). For more on the financial applications of Deep Learning, we refer the reader to (Dixon et al., 2020).

Designing the Neural Network

From Figure 5.1, we have that the feedforward network takes on the following form:

$$\text{Linear Input Layer } L_1 \to \cdots \text{ Hidden Layers } \cdots \to \text{ Output Linear Layer } L_{m+1}$$

The input layer L_1 maps the input data of dimension n to an intermediary dimension p. p is considered to be the number of neurons of the NN. After m hidden layers is the output layer L_{m+1}. The output layer L_{m+1} maps data from p dimensions down to the number of dimensions for the output n'. Much like single layer NNs, multilayer NNs also have powerful approximation abilities, as well as other benefits.

Another popular initialization layer before the hidden layers is to have a *normalization* layer. This typically takes the form of:

$$\frac{X - \mathbb{E}[X]}{\sqrt{\text{Var}(X)}} \tag{5.5}$$

which is a transformation with respect to the mean and variance of the data vector X.

For the hidden layers, it is common to utilize sigmoidal functions, due to their use in the universal approximation theorem. Below are three common hidden layer functions:

1. tanh : Hyperbolic tangent

2. $\max\{x, 0\}$: Rectified Linear Unit (ReLU)

3. $\max\{x, 0\} - \alpha \min\{0, x\}, 0, \alpha, < 1$: "Leaky" ReLU. α is typically chosen to be .01.

For other hidden layer functions, see `https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity`.

Choosing the Loss Function

In order to find effective weights and biases, a *loss* function is employed for the model to be trained against before applying the optimizer. If there is known output data, a popular loss function is the mean squared error (MSE), which utilizes the L^2 norm, or mean absolute error (MAE), with the L^1 norm. The MSE takes the form of:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{5.6}$$

Where x_i is the known value, and $\hat{x}_i$ is the prediction from the neural network. For more loss functions, we refer the reader to `https://pytorch.org/docs/stable/nn.html#loss-functions`.

Choosing the Optimizer

Utilizing the training data, it is common for deep learning algorithms to break it up to some number of *batches*, and to train the network on one batch of data at a time. This then updates the weights and biases of the neural network using a value known as the learning rate ℓ. This is done through algorithms such as stochastic gradient descent (SGD), as well as the Adam Optimizer (see `https://pytorch.org/docs/stable/optim.html#algorithms`). Below is an outline of the algorithm adapted from Goodfellow et al. (2016):

Algorithm 2 Stochastic Gradient Descent (SGD) Algorithm

Require: Learning rates ℓ_k
Require: Initial Parameters θ
  $k \leftarrow 1$
  while Stopping criterion not met **do**
    Sample a batch of training data X and target data Y
    Compute Gradient Estimate: $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(X;\theta), Y)$
    Update: $\theta \leftarrow \theta - \ell_k \hat{g}$
    $k \leftarrow k + 1$
  end while

The L function is the loss function, and f represents the neural network. This is the gradient descent portion of the stochastic gradient descent. The randomness comes from how the data is sampled and the initial conditions. Additionally, the stopping criteria can be set to continue until the loss function is sufficiently close to 0, or until a certain number of iterations have been completed. The conditions necessary for convergence are:

$$\sum_{k=1}^{\infty} \ell_k = \infty \tag{5.7}$$

and

$$\sum_{k=1}^{\infty} \ell_k^2 < \infty \tag{5.8}$$

In practice, it is not uncommon for there to be a single learning rate, and only using a finite number of iterations for the stopping criteria. Alternatively, the learning rate can decay linearly up until a certain point, and then it is left constant. This will often lead to suboptimal weights and biases with respect to that loss function, but will be sufficient for practical purposes.

Lastly, one major difficulty in finding global minima for gradient descent algorithms is finding local minima. Local minima occur as loss surfaces for the neural network parameters are often not convex, and

thus there is no global minima for the gradient descent algorithm to discover. Therefore, the SGD algorithm can be generalized to try to prevent the algorithm terminating in these minima. One way to attempt to circumnavigate these problems is to test multiple learning rates, and see which produce better results. Other techniques include changing the size of the minibatch, changing the number of iterations, or refining the dataset further before training.

Another popular algorithm is known as the Adam optimizer (Kingma and Ba, 2014). The Adam optimizer works in a similar way to the SGD algorithm above, but also utilizes the second moment estimate in addition to the first moment estimate when using the gradient. This typically results in faster convergence to the minima of the loss surface as well.

For more on gradient descent algorithms, we refer the reader to Calin 2020, Section 14.3 and Goodfellow et al. 2016, Chapter 8.

Training the Network

In order to train the neural network, we utilize the training data, the loss function, and the network architecture to find effective weights and biases for the problem at hand. This is commonly done through the backpropagation algorithm (see Dixon et al., 2020, 5.1). A visualization of this algorithm is provided in Figure 5.2, from `https://pytorch.org/blog/how-computational-graphs-are-executed-in-pytorch/`.

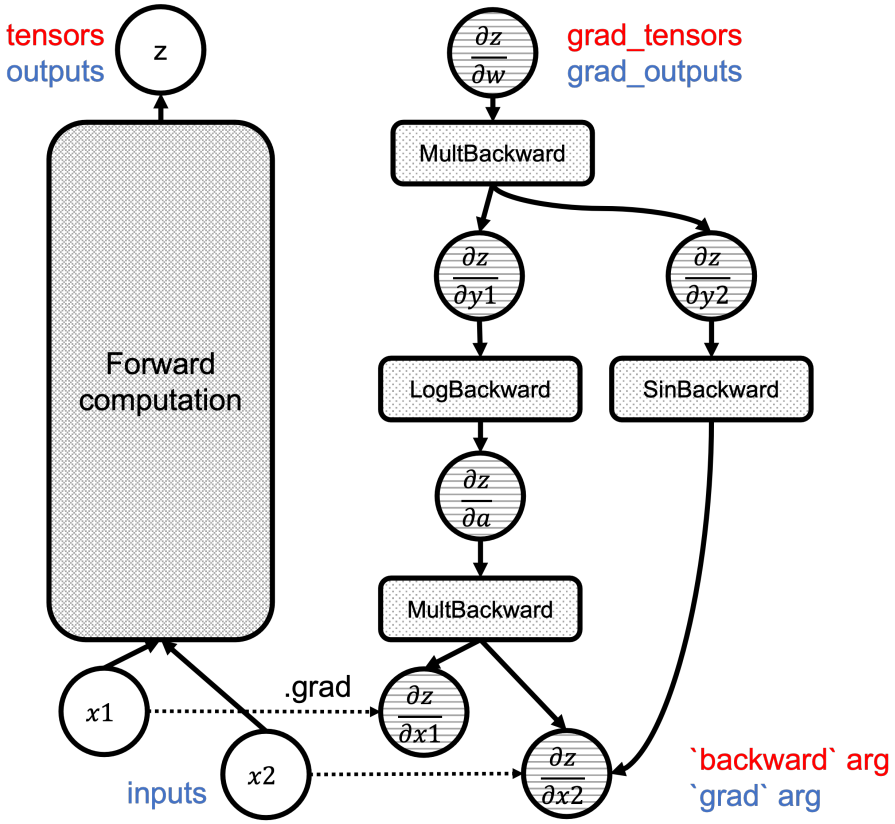


Figure 5.2: A visualization of the backpropagation algorithm

When designing the neural network, it is important to decide on the *hyper parameters*. Hyper parameters is a value used to control the learning process. The number of iterations to preform in the optimizer algorithm, learning rates, the size of data batches, and the number of neurons are all examples of hyper parameters. Other parameters include the design of the neural network itself, such as how many layers to utilize. These can be tuned as needed when testing to find the most accurate results for the problem at hand. One way to verify this accuracy is to use a subset of the data for testing the results, that is completely untouched aside

from this accuracy test.

5.3 Deep BSDE Solver

In this section, we discuss deep learning approaches to numerically solving (R)BSDEs.

The seminal work of (E et al., 2017) utilizes deep learning to find numeric solutions to BSDEs. At a high level, they discretize the BSDE and at each step and then utilize a neural network to predict the value of the Y process.

First, consider a BSDE as in Definition 3.1. This is of the form:

$$Y(t) = \xi + \int_t^T f(s, Y(s), Z(s))ds - \int_t^T Z(s)dW(s) \tag{5.9}$$

This can be discretized (E et al., 2017, Section 2.4) into Equation 5.10. Let Π be a partition of $[0, T], T > 0$ of size n. Then, we have that

$$Y(t_{n+1}) = Y(t_n) - f(t_n, Y(t_n), Z(t_n)))(t_{n+1} - t_n) + \langle Z(t_n), W(t_{n+1}) - W(t_n)\rangle \tag{5.10}$$

Where W is a Brownian Motion. Now, we can apply the Deep BSDE solver, as outlined below:

Algorithm 3 The Deep BSDE Solver

Sample X, dW from the stochastic process and distribution for Brownian Motion.
for $i = 0, ..., n$ **do**
  Let $\widehat{\mathcal{U}}_i$ be a NN approximation of Z_i
  Compute $Y(t_{i_{+1}})$ using Equation 5.10.
end for
Compute The Loss $L := \mathbb{E}[|Y(T) - g(X_T)|^2]$
Update the weights and biases of $\widehat{\mathcal{U}}_i$ for $i = 0, ..., n$ using L via SGD

This then provides a numerical approximation of $Y(0)$, which is the initial value of the BSDE. Intuitively, this value should be deterministic for the problem at hand, and thus this scheme will converge to the correct value under the loss function above. For proof, we refer the reader to E et al. 2017, Section 3.

5.4 Dynamic Programming Approach to (R)BSDE Approximation

In Huré et al. 2019, the authors utilize a dynamic programming approach to approximate both BSDEs and RBSDEs. Additionally, their approach utilizes multiple neural networks for both the Y and Z processes, instead of just using the Z process. The loss function is to minimize the difference between the terminal condition and each time step, as compared to just at the terminal time T as in the Deep BSDE solver. Also, the scheme has the ability to incorporate a reflection term by changing the value of $\widehat{U}_i$ after training that timestep, which allows this technique to approximate RBSDEs as well. This process also goes backwards instead of forwards, which is more in lined with the fact that BSDEs run backwards in time.

Consider a reflected BSDE of the form outlined in Definition 4.1. Then, we can apply Algorithm 4, outlined below.

Algorithm 4 The Hure et al. RBSDE Solver

Let $\Pi := 0 = t_0 < \cdots < t_N = T$ be a partition of the interval $[0, T]$
Sample X, dW from the stochastic process and distribution for Brownian Motion
Let $\widehat{\mathcal{U}}_N = g(X_T)$, the terminal condition be the approximation of the RBSDE at time T.
for $i = N-1, \cdots, 0$: **do**
  Given $\widehat{\mathcal{U}}_{i+1}$, use two neural networks $\mathcal{U}_i, \mathcal{Z}_i$ to approximate the Y and Z processes. Train the network to minimize:
  $\mathbb{E}|\widehat{\mathcal{U}}_{i+1}(X_{t_{i+1}}) - F(t_i, X_{t_i}, \mathcal{U}_i(X_{t_i}), \mathcal{Z}_i(X_{t_i}), \Delta t_i, \Delta W_{t_i})|^2$
  After training, set $\widehat{\mathcal{U}}_i = \max\{\mathcal{U}_{t_i}, g(X_{t_i})\}$
end for

In the dynamic programming step, F is the (R)BSDE approximation. When discretized, this of the form

$$F(t_{i+1}Y(t_{i+1}), Z(t_{i+1})) = Y(t_i) - f(t_i, Y(t_i), Z_i)h + Z(t_i)^\top \Delta W(t_i) \tag{5.11}$$

which is quite similar to equation 5.10.

This technique is able to directly handle an (R)BSDE due to the incorporation of the reflection step of taking the maximum between the approximation and the terminal condition. This step can be omitted to approximate a standard BSDE instead.

5.5 Gao et al. RBSDE Solver

Lastly, another similar technique has been developed to numerically approximate (R)BSDEs. Again, consider an RSBDE of the form 4.1. This algorithm seeks to minimize the variance of the $Y(0)$ value. Intuitively, this works as $Y(0)$ should have a deterministic value, as it is at the present time. Therefore, the variance around $Y(0)$ will converge to 0 when optimizing the weights and biases via an SGD type algorithm. For a more detailed proof, see Gao et al. 2022, Section 2.

This solver is also able to handle RBSDEs, as the maximum between $Y(t_i)$ and $g(X(t_i))$ is taken. This ensures that the approximation remains above the boundary condition g.

Algorithm 5 The Gao et al. RBSDE Solver

Let $\widehat{\mathcal{U}}_{N-1}, ..., \widehat{\mathcal{U}}_0$ be N neural networks to approximate Z_{t_i}
Sample X, dW from the stochastic process and distribution for Brownian Motion
Let $Y(T) = g(X_T)$
for $i = \{N-1, ..., 0\}$ **do**
  Let $\widehat{Z}(t_i)$ be the approximations from $\widehat{\mathcal{U}}_i$
  Let $Y(t_i) = Y(t_{i+1}) + f(t_i, X(t_i), Y(t_{i+1}), Z(t_i))\Delta t - Z(t_i)^\top \Delta W(t_i)$
  Let $Y_{t_i} = \max\{g(X(t_i)), Y(t_{i+1}) + f(t_i, X(t_i), Y(t_{i+1}), Z(t_i))\Delta t - Z(t_i)^\top \Delta W(t_i)\}$
end for
Update the weights and biases of $\widehat{\mathcal{U}}_i$ by setting the loss $L := \text{var}(Y_0)$ with SGD

In (Gao et al., 2022), the authors provide a proof of convergence for this algorithm to the original RBSDE.

Chapter 6

Modeling Stock Options with BSDEs and Neural Networks

One of the key areas of application of BSDEs and stochastic calculus is in the field of financial mathematics. Financial mathematics is using mathematical principles and styles of thinking about problems and applying to them to problems arising from financial markets.

6.1 Financial Background

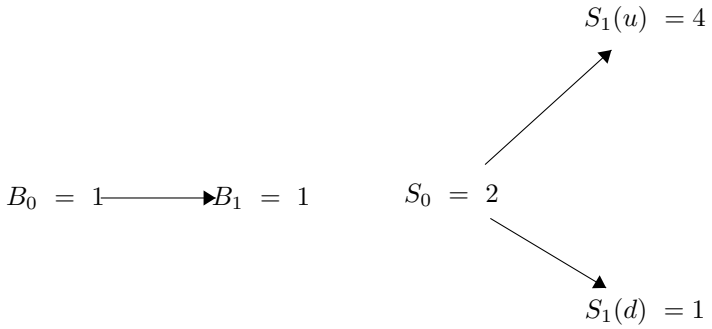
Generally speaking, a financial market is a place where people go to buy and sell financial assets. The example we focus on is stock options in a company. A stock is a share of ownership of a certain company, such as Apple (Hayes, 2023). An option is a contract which gives the buyer the right, but not requirement to buy or sell stocks at an agreed upon price K at (or up to) a time T. Stocks and options are traded at exchanges, where investors are able to buy and sell assets from other investors.

The mathematical theory of finance was developed by many different economists working on different areas, such as Harry Markowitz, William Sharpe, Merton Miller, Fischer Black, and Myron Scholes (wik, 2022).

Now, we provide a more technical explanation of financial markets to demonstrate a key application of deep learning to approximate the solution to models.

6.2 Complete Financial Markets

A financial market is where agents have the ability to buy and sell items from each other. As a guiding example through some of the technical definitions, we consider the following financial market that consists of one time period and two assets, a bond that carries no interest rate and a stock S_t. This can be represented in the figure below:



Here, u is the event that the stock will go up, and d is the event that the stock will go down.

Within this market, it is possible for financial agents to purchase a *option* on stock S_t. The two types of options are a *call* and *put* option. They both rely upon a strike price K. Mathematically, these options can be represented by the following functions, known as payoff functions. For a call:

$$C_t = (S_t - K)^+ = \max\{S_t - K, 0\} \tag{6.1}$$

And for a put:

$$P_t = (K - S_t)^+ = \max\{K - S_t, 0\} \tag{6.2}$$

Remark 6.1 (American and European Options)**.** *In more complex financial markets, such as those present in the real world, there are multiple flavors of option contracts that can be purchased. The primary two, offered referred as a vanilla are the American and European option. A European option gives the buyer the right, but not obligation to exercise the contract at the expiration date $0 < T < \infty$. For an American option, the buyer has the opportunity to exercise early. Early exercising cannot be seen in our one period model, but can be seen in a multiperiod or continuous time model.*

Example 6.1 (Computing the put payoff)**.** *For a strike price of $K = 3$, the put payoff function at $T = 1$ becomes*

$$(3 - S_t)^+ = \begin{cases} 0 & S_1(u) = 4 \\ 2 & S_1(d) = 1 \end{cases} \tag{6.3}$$

A key question is to determine what the *fair* price for the option at time 0. In this context, a fair price is one that does not allow for an arbitrage opportunity.

Definition 6.1 (Arbitrage, intuitively)**.** *Intuitively speaking, arbitrage is the ability to make money with no risk. This can be viewed through the following joke (adapted from Delbaen and Schachermayer (2006)):*

Professor Sturm and Forrest go on a walk and Forrest finds a \$100 *bill lying on the street. Forrest wants to pick it up, and Professor Sturm says "do not try to do that. It is impossible that there is a* \$100 *bill on the street. Assume by contradiction there is a* \$100 *bill on the street. Then, somebody else would have picked it up before you."*

The "somebody else" Professor Sturm is referring to here is a market agent eliminating arbitrage by exploiting the free money on the ground by picking up bills until it is gone.

For a more thorough treatment of arbitrage, we refer the reader to Delbaen and Schachermayer (2006). For our example, the fair price for the put option is one that allows *replication* the behavior of the option using just stock and bonds.

Example 6.2 (Computing the arbitrage free price)**.** *To find the arbitrage free price of the put option, we construct a portfolio V such that it will have the same value as the option at time T, no matter the result. This known as a replicating portfolio. This takes the form of the equation*

$$V_T = \alpha S_T + \beta B_T = (3 - S_T)^+.$$

Therefore, we have the equations

$$\alpha 4 + \beta = 0,$$
$$\alpha + \beta = 2.$$

This leads to the solution $\alpha = -\frac{2}{3}$ and $\beta = \frac{8}{3}$. This means that $V_T = P_T$, and as we do not want an arbitrage opportunity, we require that $V_0 = P_0$. Thus, using the same α and β, we obtain that

$$V_0 = \alpha S_0 + \beta B_0 = -\frac{2}{3}2 + \frac{8}{3}1 = \frac{4}{3}.$$

Therefore, the fair option price that is arbitrage free is $\frac{4}{3}$.

Note that this technique made no assumptions on the probability of u or d occurring, aside from the fact that they are both greater than 0. We can now apply an alternative method from Theorem 1.6.1 from Delbaen and Schachermayer (2006) and find a probability q such that S_t is a Martingale under q. We call this probability we have found to be the measure Q, known as the risk neutral measure.

Example 6.3 (Martingale calculation of the arbitrage free price)**.** *To find the appropriate probability q to use, we require that S_t becomes a martingale under q. This means that the expected value at time T is the same as the expected value at time 0. This leads to the equation*

$$4q + (1-q)2 = 2$$

which leads to the solution that $q = \frac{1}{3}$. Thus, under the risk neutral measure Q, we have that

$$\mathbb{E}^Q[(K - S_t)^+] = V_0 = 0\frac{1}{3} + 2\frac{2}{3} = \frac{4}{3},$$

This is the same arbitrage free price as before.

Remark 6.2 (Model independence)**.** *As seen in the example computation above, we see that arbitrage free prices are model independent in the sense that it does not depending on the probability of events occuring. This is useful for verifying prices of more complicated models by testing them against a more simple binomial model.*

A key result in the area of asset pricing are known as the fundamental theorems of asset pricing, stated below. First, we provide the definition of a complete market.

Definition 6.2 (Complete Market (Föllmer and Schied, 2016))**.** *A arbitrage free market model is complete if every contingent claim is attainable.*

Theorem 6.1 (The first and second fundamental theorems of asset pricing)**.** *For a complete market, we have that (adapted from Föllmer and Schied (2016)):*

1. *A market model is arbitrage free if and only if there exists a probability measure Q such that risk neutral measure for the assets.*
2. *An arbitrage free market model is complete if and only if Q is unique.*

Example 6.4 (The Black Scholes Complete Market)**.** *The Black Scholes market is the classical example of a complete market. In their seminal work (Black and Scholes, 1973), they assume:*

1. *There exists a risk free rate r*
2. *The stock price follows the following geometric Brownian motion:*

$$dS_t = \mu S_t dt + \sigma S_t dW$$

   where μ is the mean return, σ is the standard deviation of the returns of the asset (known as the volatility), and dW is an infinitesimal increment of Brownian motion. μ and σ are constant in this model.

3. *The stock does not pay a dividend*
4. *There is no arbitrage opportunity*
5. *It is possible to borrow and lend any amount of cash at the rate r*
6. *It is possible to buy and sell any fraction of the stock S*
7. *There are not transaction fees or costs (this is known as a frictionless market).*

Utilizing Algorithm 5, we can approximate the option price for calls and puts under these assumptions. We demonstrate this below with an American Put Option.

Code can be found in Appendix A.

For a European put option, we have that the driver and terminal condition functions are:

$$f(t, X_t, Y_t, Z_t) = -rY_t$$

and

$$g(t, X_t) = (K - X_t)^+.$$

Thus, the BSDE is

$$Y_t = (K - X_T)^+ + \int_t^T -rY_s ds - \int_t^T Z_s dW_s$$

The condition below then allows the BSDE to become an (R)BSDE to represent the American put option.

$$Y_t \geq g(t, X_t)$$

Here, X_t is the stock process, represented by

$$dX_t = \mu X_t dt + \sigma X_t dW$$

Discretized, this is

$$X_{t_{i+1}} = X_{t_i} + \mu X_{t_i} \Delta t + \sigma X_{t_i} \Delta W_{t_i}$$

For the network structure, we use a input layer of normalization, a linear input layer, two ReLU layers, followed by a linear output layer. For an optimizer, we utilize the Adam optimizer. We follow Algorithm 5. For numeric parameters, we refer the reader to Table 6.1.

Neural Network Parameters	Black Scholes Parameters
Time Steps = 10	$\mu = .08$
Learning Rate .008	$r = .02$
Batch Size = 1000	$\sigma = .20$
Number of epochs = 6000	$T = \frac{1}{12}$
Predicted Price = 3.949	Analytical Price $= 3.97
Absolute Relative Error: 0.05%	

Table 6.1: Black Scholes American Put Option Numerical Modeling

In Figure , a visual of a sample path is displayed:

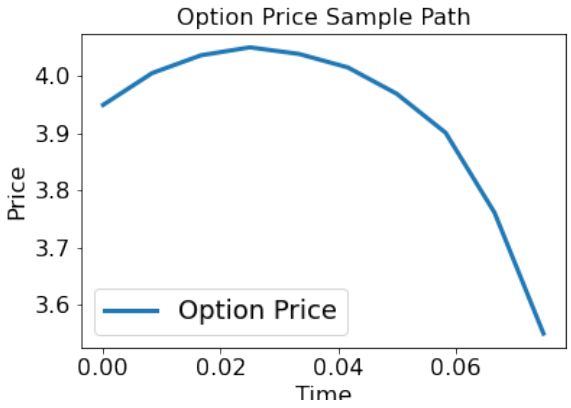


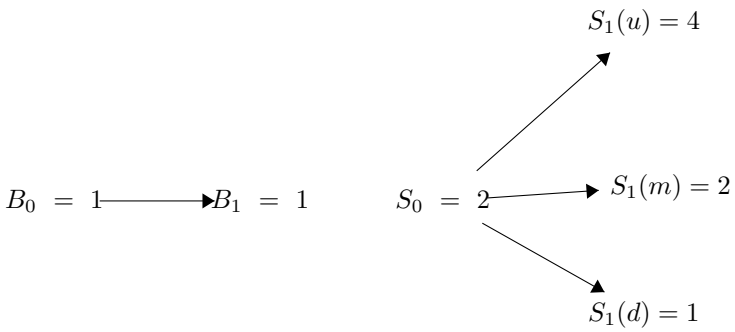
Figure 6.1: American Option Path Under Black Scholes

6.3 Incomplete Financial Markets

Above, we have assumed that we are working a complete market. Now, we introduce the notion of an incomplete market.

Definition 6.3 (Incomplete Market)**.** *An incomplete market is one where there are multiple risk neutral measures.*

In reality, most markets are incomplete. This is due to the fact that many of the assumptions, such as the ones used in the Black Scholes market model, are incorrect. Consider the following simplified trinomial model.



Note that this market is incomplete, but there can still be an arbitrage free price. This price now becomes an interval, as there are three variables (the probabilities q_1, q_2, q_3) and one equation for the time period.

Example 6.5 (Computing the arbitrage free interval)**.** *From the martingale condition, we have two equations:*

$$q_1 + q_2 + q_3 = 1$$

and

$$4q_1 + 2q_2 + q_3 = 2.$$

Note that probabilities are positive, so we have the extra constraint that $q_1, q_2, q_3 \geq 0$. Thus, we can solve the system to obtain the following solution

$$q_1 = \frac{q_3}{2},$$

$$q_2 = \frac{2 - 3q_3}{2},$$

$$q_3 \in (0, \frac{2}{3}).$$

Now, under the risk neutral measure, we can compute

$$\mathbb{E}^Q[(K - S_T)^+] = 0q_1 + q_2 + 2q_3$$

$$= \frac{2 - 3q_3}{2} + \frac{4q_3}{2} = \frac{2 + q_3}{2}$$

Therefore, the price interval is $(1, \frac{4}{3})$.

Even in these markets, the arbitrage free prices are model independent of probability of q_1, q_2, and q_3.

Remark 6.3. *For all of the examples above, the price of the option was symmetric, which means it was independent of the buyer or the seller. This is due to two primary reasons. The first relates to the replicating claim technique when pricing options. With this technique, there is no assumption on whether the agent is buying (being "long") or selling the option (going "short"). This implies that buying an option is the same as selling negative one of an option. The second reason is that we assume that the risk free rate r is the same for both buying and selling the option. This is incredibly idealistic, as one can see when considering the interest rate on their bank account compared to the interest rate with their credit card bill. When considering a different interest rate for buying and selling options, this breaks the linearity in the market, resulting in different option prices for buying and selling (see Bonner and Campanelli, 2016, Section 3).*

6.4 Indifference Pricing

Indifference pricing is a technique of valuing an option in a way that the value to a buyer is the same regardless if they purchase them for this price or do not purchase them. Indifference pricing removes some of the idealistic assumptions made by the Black Scholes and other models. In particular, this method allows for "inevitable intrinsic risk that cannot be completely hedged away but remains with the holder" (Encyclopedia of Quantitative Finance, 2010). This thus produces different results for the buyer and seller prospective as a nonlinear function is used when pricing the option on the buyer and seller side. Here, we will focus on the buyer. When a utility function is used, this is known as *utility indifference pricing.*

6.4.1 Utility Indifference Pricing

In order to use a utility function when pricing assets, the notion of a utility function must be formalized. A *utility* function measures the value of something (here, money) through the utility it gives the user. A utility function is an increasing, concave function U that represent the preferences of the investor. Some popular examples include:

1. $\frac{x^p}{p}, p \in (-\infty, 0)$ or $p \in (0, 1)$. This is known as a power utility function.
2. $\log x$. This is the log utility function.
3. $-e^{-\alpha x}, \alpha > 0$. This is the exponential utility function.

Below is an example of indifference pricing with utility functions using the complete market model from earlier.

Example 6.6 (Indifference Pricing with Utility Function)**.** *Using the complete market model from before, we can formulate the arbitrage free price as the following optimization problem*

$$\max_{\theta \in \mathbb{R}} \mathbb{E}[U(x + (K - S_t)^+ - P_0 + \theta(S_T - 2B_T))] = \max_{\theta \in \mathbb{R}} \mathbb{E}[U(x - P_0 + \theta(S_T - 2B_T))]$$

where U is the utility function. As the model is complete and arbitrage free prices are model independent, we have that the arbitrage free price remains the same as before. Note that this is from the buyer's perspective as we are adding the the value of the payoff function. If we were considering the seller, we would add P_0 and subtract $(K - S_t)^+$.

6.4.2 Risk Indifference Pricing

Another indifference pricing technique is known as *risk* indifference pricing. Instead of a utility function, a risk measure is used instead. A risk measure is a function that has quantifies the risk of a random variable. There are two dominant classes of risk measures, which we will outline axiomatically below:

Definition 6.4 (Coherent Risk Measure)**.** *A coherent risk measure $\rho : \mathcal{L}^\infty(\Omega, \mathcal{F}, \mathbb{P}) \to \mathbb{R}$ is a function that obeys the following properties:*

(i) Monotonicity: If $X \geq Y$ almost surely, then $\rho(X) \leq \rho(Y)$

(ii) Cash Invariance: If m is a deterministic process (like cash), then $\rho(X + m) = \rho(X) - m$

(iii) Subadditivity: For two variables X, Y, we have that $\rho(X + Y) \leq \rho(X) + \rho(Y)$

(iv) Positive Homogentiy: For $\lambda \geq 0$, we have that $\rho(\lambda X) = \lambda \rho(X)$

Intuitively, normalization means that we have that an holding 0 assets involves 0 risk. Monotonicity means that if one asset provides better returns than another, then the better performing asset results in less risk. Cash invariance means that holding cash in an account does not contribute to the risk. Subadditivity means that holding a diverse portfolio of multiple assets is less risky than a single asset. And positive homogenity means that risk scales linearly with the holdings of that asset.

Example 6.7 (Expected Shortfall (Tangpi, 2022)). *The expected shortfall of a random variable is a coherent risk measure. In finance, this known as the average value at risk. For $0 < \alpha < 1$, we have that the expected shortfall (ES) at α is*

$$ES_\alpha(X) = -\frac{1}{\alpha}\int_0^\alpha VaR_\gamma(X)d\gamma.$$

Where VaR is the value at risk.

Sometimes, using a coherent risk measure is too tight of a definition. Therefore, we relax these definitions and use *convex* risk measures instead. Essentially, subadditivity and positive homoegentiy is replaced with convexity, as done below.

Definition 6.5 (Convex Risk Measure). *A convex risk measure $\rho : \mathcal{L}^\infty(\Omega, \mathcal{F}, \mathbb{P}) \to \mathbb{R}$ is a function that obeys the following properties:*

(i) Monotonicity: If $X \geq Y$ almost surely, then $\rho(X) \leq \rho(Y)$

(ii) Cash Invariance: If m is a deterministic process (like cash), then $\rho(X + m) = \rho(X) - m$

(iii) Convexity: For $\lambda \in [0, 1]$, we have that $\rho(\lambda X + (1 - \lambda)Y) \leq \lambda\rho(X) + (1 - \lambda)\rho(Y)$

Remark 6.4 (Coherent Risk Measures are Convex). *Note that convexity is a weaker definition than subadditivity and positive homogenity. Therefore, all coherent risk measures are convex. However, not all convex risk measures are coherent. For example, the expected maximum drawdown of random variable is a convex risk measure.*

To tie everything together, we will now show that (R)BSDEs are (time consistent) convex risk measures. But first, we will define another property of time consistency:

Definition 6.6. *A risk measure ρ_t is said to be time consistent if for all $s \leq t \leq T$, we have that if $\rho_t(X) \geq \rho_t(Y)$ implies that $\rho_s(X) \geq \rho_s(Y)$. Another formulation is that*

$$\rho_s(\xi_T) = \rho_s(-\rho_t(\xi_T))$$

where ξ_T is the terminal condition of the (R)BSDE.

Theorem 6.2 ((R)BSDEs and risk measures). *As a function of the terminal condition, the solution to a (R)BSDE is a time consistent convex risk measure if f is convex, does not depend on Y, and grows at most quadratically.*

Proof. Let Y_t be a BSDE as below

$$Y_t(\xi) = \xi + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

describe a risk measure $\rho_t(\xi) = Y_t(\xi)$. First, note that the zero BSDE is 0, and thus our risk measure ρ_t is normalized.

By the comparison principles (see Delong, 2013, Proposition 13.1.2), we have that if $\xi \geq \xi'$ almost surely, we have that $Y_t(\xi) \geq Y_t(\xi')$, which means that our risk measure $\rho_t(\xi)$ is monotonic.

For cash invariance, consider the following two BSDEs

$$Y_t(\xi) = \xi + \int_t^T f(s, , Z_s)ds - \int_t^T Z_s dW_s$$

and

$$Y'_t(\xi') = \xi' + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

We will show that $Y' = Y - m$, where m is deteriminstic. Note that if we set $\xi' = \xi + m$, then we have that

$$Y'_t(\xi') = m + \xi + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

Subtracting results in

$$Y'_t(\xi') - Y_t(\xi) = -m$$

which thus means that

$$Y'_t(\xi') = Y_t(\xi) - m$$

Thus, we have that Y_t is cash invariant.

The convexity of Y_t follows from linearity of integration and the fact that f itself is convex.

To show time consistency, note that

$$Y_t(\xi) = \xi + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

Now, consider another BSDE with terminal condition $-Y_t$

$$Y'_t(-Y_t(\xi)) = -Y_t(\xi) + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

Let $s \leq t$. Note that $Y'_s = Y_s$, as

$$Y'_s(\xi) = Y_s(\xi) - (+\int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s) + \int_t^T f(s, Z_s)ds - \int_t^T Z_s dW_s$$

so we have that $Y'_s(\xi) = Y_s(\xi)$ for $s \leq t$, which yields time consistency. Therefore, we have that the BSDE is a convex risk measure. □

For a concrete example, we extend the work of Sircar and Sturm (2015) to price American Put Options under stochastic volatility from the buyer's and seller's perspective.

Buyer's Price First, we will discuss the buyer's perspective. Under the risk measure ρ that obeys the properties highlighted above, we have that the risk indifference price is:

$$\operatorname*{ess\,inf}_{h \in \mathcal{C}_{t,\tau^*}} \rho_{t,\tau^*}(h) - \operatorname*{ess\,inf}_{\tau \in \mathcal{T}_{t,T}} \operatorname*{ess\,inf}_{h \in \mathcal{C}_{t,\tau}} \rho_{t,\tau}(h + \xi_\tau) \tag{6.4a}$$

$$\tau^* = \operatorname*{arg\,min}_{\tau \in \mathcal{T}_{t,T}} \operatorname*{ess\,inf}_{h \in \mathcal{C}_{t,\tau}} \rho_{t,\tau}(h + \xi_\tau) \tag{6.4b}$$

To unpack this formula, note that the second component is represented with a (R)BSDE, while the first term is a BSDE. The second RBSDE term concerns taking two essential infima: one for the stopping time, and one for the trading strategy. The inner inf requires that the optimal trading strategy $h \in \mathcal{C}_{t,\tau}$ is taken. This strategy involves holding the option from t to τ. The second infimum involves finding the stopping time τ of the asset. This is when the option should be exercised. Therefore, this is the part of the indifference price that involves considers buying the option.

The first BSDE concerns never acting on the option, which means that $\xi = 0$. In the first BSDE term, $\tau^\star$ is used for the stopping times, as this represents the option being exercised on the second term.

Algorithm 6 Computing the Buyer's Price, coded in Appendix A

Determine a driver function for the risk measure and stock process dynamics
Train an RBSDE using Algorithm 5 with $\xi = (Ke^{-rt} - S_t)^+$ as the boundary and $-f(t, X_t, Y_t, -Z_1, -Z_2)$ as the driver
Using the **same** sample paths, determine $\tau^\star$, the first time the trained RBSDE goes below the boundary
Train a BSDE with $\xi = 0$ using the same sample paths and stopping at $\tau^\star$ on each sample path
Computing the indifference price with Equation (6.4)

Seller's Price The seller's indifference price is computed through similar means. The equation is given by

$$\operatorname*{ess\,inf}_{h\in\mathcal{C}_t^\tau} \operatorname*{ess\,sup}_{\tau\in\mathcal{T}_{t,T}} \rho_{t,T}(h-\xi_\tau) - \operatorname*{ess\,inf}_{h\in\mathcal{C}_t} \rho_{t,T}(h) \tag{6.5}$$

The first term is a RBSDE, while the second term is a BSDE. The set $\mathcal{C}_t^\tau$ represents permissible trading strategies that depend on the realization $\tau(\omega)$ of the stopping time τ as opposed to the stopping time τ itself. This differs from $\mathcal{C}_t$, which is the permissible trading strategies over all paths, independent of realization of τ. The RBSDE term is now reflected against a boundary $L_t = \xi_t + Y_0(t)$, where $Y_0(t)$ is the value of the BSDE term (with $\xi = 0$) at time 0. Below is an algorithmic outline

Algorithm 7 Computing the Seller's Indifference Price, code in A

Determine a driver function for the risk measure and stock process dynamics
Train a BSDE using Algorithm 5 with $\xi = 0$, calling the solution $Y_0(t)$
Compute $L_t = (Ke^{-rt} - S_t)^+ + Y_0(t)$
Train a RBSDE using Algorithm 5 with $\xi = (Ke^{-rt} - S_t)^+$ and the boundary L_t and the driver $f(t, X_t, Y_t, Z_1, Z_2)$
Computing the indifference price using Equation 6.5

Numerical Example For a particular example, we utilize the following stochastic volatility model:

$$dS_t = \mu(Y_t)S_t dt + \sigma(Y_t)S_t dW_t^1 \tag{6.6}$$

Where $\mu(Y_t) = \mu$, $\sigma(Y_t) = \frac{.7}{\pi}(\arctan(Y_t - 1) + \frac{\pi}{2}) + .03$. The volatility is

$$dY_t = b(Y_t)dt + a(Y_t)(\rho dW_t^1 + \sqrt{1-\rho^2}dW_t^2) \tag{6.7}$$

Where $b(Y_t) = \alpha(m - Y_t), a(Y_t) = \nu\sqrt{2\alpha}$. The terminal condition of this function is the put payoff $(K - S_t)^+$ and the driver is

$$f(t, X_t, Y_t, Z_1, Z_2) = -Z_1\left(\frac{\mu - r}{\sigma(Y_t)}\right) - \frac{1}{2\gamma}\left(\frac{\mu - r}{\sigma(Y_t)}\right)^2 + \frac{\gamma}{2}Z_2^2 + \eta\left(\frac{-(\mu - r)}{\sigma(Y_t)}\right)Z_2 \tag{6.8}$$

There are two Z_i processes as this is a two dimensional problem. The first dimension is the stock process and the second is the volatility process. Numerically, we utilize values found in Table 6.2. Values in parantheses were used for the seller's price due to memory constraints of this technique.

Parameter	Value	Role	Parameter	Value	Role
Batch Size	1000 (500)	Deep Learning	T	.25	
Epochs	6000 (4000)		μ	.08	
Learning Rate	.01		α	5	
Time Steps	25		m	0	Stock Model
Initial Price	100 \$	Stock Model	ν	1	
Initial Volatility	14.97%		ρ	-0.2	
Strikes	$[70, 115]$		r	0.02	

Table 6.2: Numeric Parameters for Indifference Price Computation

This produces the following smile curve in Figure 6.2.

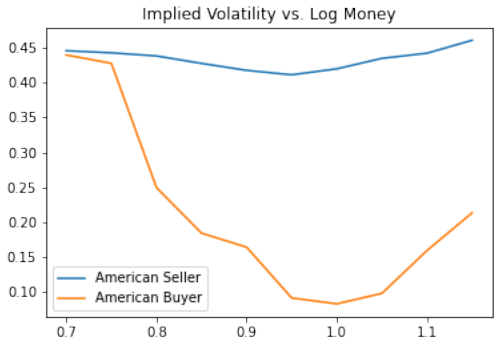


Figure 6.2: Smile curves for the American put option display a bid ask spread

Note that the strike price is given in log moneyness, which is $\ln(\frac{S_0}{K})$. We plot against implied volatility, which measures the volatility implied by the option at that log moneyness value. This is analogous to a log plot. In Figure 6.2, we have that the seller's price is always above the buyer's price. This creates a type of bid ask spread for the option, as the seller's implied volatility is above the buyer's.

Bibliography

Mathematical finance, Dec 2022. URL `https://en.wikipedia.org/wiki/Mathematical_finance`.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-dujaili, Ye Duan, Omran Al-Shamma, Jesus Santamaría, Mohammed Abdulraheem Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8, 2021.

Louis Bachelier. Théorie de la spéculation. *Annales scientifiques de l'École Normale Supérieure*, 3e série, 17:21–86, 1900. doi: 10.24033/asens.476.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *J. Polit. Econ.*, 81(3): 637–654, 1973. ISSN 0022-3808. doi: 10.1086/260062.

Catherine Jean Bonner and Jeremiah Campanelli. Arbitrage-free pricing of xva for options in discrete time, 2016.

Ovidiu Calin. *Deep learning architectures—a mathematical approach*. Springer Series in the Data Sciences. Springer, Cham, 2020. doi: 10.1007/978-3-030-36721-3.

René Carmona, editor. *Indifference pricing*. Princeton Series in Financial Engineering. Princeton University Press, Princeton, NJ, 2009. Theory and applications.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2 (4):303–314, 1989. ISSN 0932-4194. doi: 10.1007/BF02551274.

Freddy Delbaen and Walter Schachermayer. *The mathematics of arbitrage*. Springer Finance. Springer-Verlag, Berlin, 2006.

Łukasz Delong. *Backward stochastic differential equations with jumps and their actuarial and financial applications*. European Actuarial Academy (EAA) Series. Springer, London, 2013. doi: 10.1007/978-1-4471-5331-3. BSDEs with jumps.

Matthew F. Dixon, Igor Halperin, and Paul Bilokon. *Machine learning in finance—from theory to practice*. Springer, Cham, 2020. doi: 10.1007/978-3-030-41068-1.

Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, nov 2017. doi: 10.1007/s40304-017-0117-6.

Albert Einstein. *Investigations on the theory of the Brownian movement.* Dover Publications, New York, 1956 - 1926.

N. El Karoui, C. Kapoudjian, E. Pardoux, S. Peng, and M. C. Quenez. Reflected solutions of backward SDEs, and related obstacle problems for PDEs. *The Annals of Probability*, 25(2):702–737, 1997. ISSN 00911798.

Encyclopedia of Quantitative Finance. *Encyclopedia of Quantitative Finance.* John Wiley & Sons Ltd, 2010.

Hans Föllmer and Alexander Schied. *Stochastic finance.* De Gruyter Graduate. De Gruyter, Berlin, 2016. doi: 10.1515/9783110463453. An introduction in discrete time, Fourth revised and extended edition of [MR1925197].

Chengfan Gao, Siping Gao, Ruimeng Hu, and Zimu Zhu. Convergence of the backward deep bsde method with applications to optimal stopping problems, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

Adam Hayes. Stocks: What they are, main types, how they differ from bonds, Jan 2023. URL `https://www.investopedia.com/terms/s/stock.asp`.

Côme Huré, Huyên Pham, and Xavier Warin. Some machine learning schemes for high-dimensional nonlinear pdes. *arXiv preprint arXiv:1902.01599*, 33, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Thomas Mikosch. *Elementary stochastic calculus—with finance in view*, volume 6 of *Advanced Series on Statistical Science & Applied Probability.* World Scientific Publishing Co., Inc., River Edge, NJ, 1998. doi: 10.1142/9789812386335.

Elisa Negrini. *Robust Deep Learning Algorithms for System Identification.* PhD thesis, Worcester Polytechnic Institute, April 2022. URL `https://digital.wpi.edu/concern/etds/pv63g3374`.

Bernt Øksendal. *Stochastic differential equations.* Universitext. Springer-Verlag, Berlin, sixth edition, 2003. doi: 10.1007/978-3-642-14394-6. An introduction with applications.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Philip E. Protter. *Stochastic integration and differential equations*, volume 21 of *Stochastic Modelling and Applied Probability.* Springer-Verlag, Berlin, 2005. doi: 10.1007/978-3-662-10061-5. Second edition. Version 2.1, Corrected third printing.

Steven E. Shreve. *Stochastic calculus for finance. II.* Springer Finance. Springer-Verlag, New York, 2004. Continuous-time models.

Ronnie Sircar and Stephan Sturm. From smile asymptotics to market risk measures. *Mathematical Finance*, 25(2):400–425, 2015. doi: https://doi.org/10.1111/mafi.12015.

Ludovic Tangpi. Lecture notes on quantitative risk management, August 2022.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.

Norbert Wiener. *Norbert Wiener: Collected works.* MIT Press, 1976.

Appendix A

Python Code

A.1 American Option under Black Scholes

```
# necessary import statements for the solver
import torch
import torch.nn as nn
import numpy as np
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch.autograd import Variable
import numpy as np

# parameters

batchSize = 1000
numEpochs = 6001
learningRate = .008
timeSteps = 10 # number of discretized time steps of the forward process
T = 1/12 # total time period in years
dt = T / timeSteps # distance
print('dt = ', dt, 'sqrt dt = ', dt ** .5)
initialPrice = 95 # initial stock price
r = .02 # risk free rate
mu = .08 # mean of the stock process
sigma = .20 # volatility
K = 98 # strike price

# Functions
# forward path to get sample data
def getSample(seed):
    torch.manual_seed(seed)

    X = torch.ones(batchSize, timeSteps) * initialPrice
    dW = torch.normal(0, torch.sqrt(torch.tensor(dt)), size = (batchSize, timeSteps))

    for i in range(timeSteps -1):
        X[:, i+1] =  X[:,i] + mu * X[:,i]*dt + sigma * X[:,i]*dW[:,i]

    return X.clone().detach(), dW.clone().detach()
# payoff function / terminal / boundary condition
def fun_g(t, X, K):

    return torch.maximum(K - X, torch.tensor(0))

# driver function
def fun_f(y):
    return -1 * r * y
```

```
# model initialization
models = {}
params = list()
torch.manual_seed(0)
for i in range(timeSteps -1, -1, -1):
    models[i] = nn.Sequential(nn.LayerNorm(batchSize), nn.Linear(batchSize, 11) ,nn.ReLU(),
                                                    nn.ReLU(), nn.Linear(11, batchSize)) # ,
                                                   nn.LayerNorm(batchSize)
    # print(models[i])
    params += list(models[i].parameters())

optimizer = torch.optim.Adam(params,  lr=learningRate)

# network training
for epoch in range(0, numEpochs):
    # print('epoch = ', epoch, '\n--------')
    optimizer.zero_grad()
    X, dW = getSample(epoch) #
    previousNetworkResults = fun_g(timeSteps, X[:,timeSteps - 1], K)
    # print(previousNetworkResults)

    for timeStep in range(timeSteps -1, -1, -1):
        # print('timeStep =', timeStep)

        modelResults = models[timeStep](X[:,timeStep])

        currentNetworkResults = previousNetworkResults + fun_f(previousNetworkResults) *
                                                    torch.tensor(dt) - modelResults * dW[:
                                                    ,timeStep]
        currentNetworkResults = torch.maximum(fun_g(timeStep, X[:,timeStep], K),
                                                    currentNetworkResults)
        previousNetworkResults = currentNetworkResults
        if epoch % 500 == 0:
            print('timestep = ', timeStep, '| mean = ', round(torch.mean(
                                                        currentNetworkResults).item(),3))
                                                        # '| currentResults = ',
                                                        currentNetworkResults[:3],

    mean = torch.mean(currentNetworkResults) * torch.ones(batchSize)

    loss = torch.var(currentNetworkResults)
    loss.backward()
    clip_value = 5
    torch.nn.utils.clip_grad_norm_(params, clip_value)
    optimizer.step()
    if epoch % 500 == 0:
        print('epoch = ',epoch, '| loss: ', round(loss.item(),4))
        print('================================')

# network testing
testSize = 500
testResults = torch.empty(testSize, timeSteps)
for i in range(testSize):
    X, dW = getSample(-1 * i) #
    previousNetworkResults = fun_g(timeSteps, X[:,timeSteps - 1], K)
    # print(previousNetworkResults)
    pathMean = torch.empty(timeSteps)
    pathMean[timeSteps -1] = torch.mean(previousNetworkResults)
    for timeStep in range(timeSteps -1, -1, -1):
        # print('timeStep =', timeStep)

        modelResults = models[timeStep](X[:,timeStep])

        currentNetworkResults = previousNetworkResults + fun_f(previousNetworkResults) -
                                                    modelResults * dW[:,timeStep]
        currentNetworkResults = torch.maximum(fun_g(timeStep, X[:,timeStep], K),
                                                    currentNetworkResults)
        previousNetworkResults = currentNetworkResults
```

```
        pathMean[timeStep] = torch.mean(currentNetworkResults)
        # print('timestep = ', timeStep, '| mean = ', round(torch.mean(currentNetworkResults
                                                            ).item(),3))
    testResults[i, :] = pathMean
# test results
for timeStep in range(timeSteps):
    print('time = ', timeStep, '| price = ',torch.mean(testResults[:, timeStep]).item())
```

A.2 American Put from Buyer's Perspective under Stochastic Volatility

```
import torch
import torch.nn as nn
import numpy as np
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch.autograd import Variable
import numpy as np
from scipy.stats import norm
import inspect
print('American Put, Buyer, less detailed, trying SGD')

batchSize = 1000# 1000
numEpochs = 6001 # 4001
learningRate = .01 # .04
print('batchSize = ', batchSize)
print('NumEpochs = ', numEpochs)
print('Learning Rate = ', learningRate)

 # [0,1,2,3,4,5,6,7,8,9]
initialPrice = 100
print('initial price = ', initialPrice)
initialVol = .149793 # .3
print('initial Vol = ', initialVol)
mu =  0.08 #.4
print('mu = ', mu )
alpha = 5
print('alpha = ', alpha)
m = 0
print('m = ', m)
nu = 1
print('nu = ', nu)
a = nu * torch.sqrt(torch.tensor(2 * alpha))
print('nu * sqrt(2 alpha ) = ', a)
rho = -.2
print('rho = ', rho)
gamma = 1
print('gamma = ', gamma )
eta = .2
print('eta = ', eta)
timeSteps = 25 # 10
print('Total number of time steps = ', timeSteps )
r = .02 # 0.05
print('risk free rate r = ', r)
T = 1/4
print('T = ', T)
dt = T / timeSteps
print('dt = ', dt)
strikes = [70,75,80,85,90,95,100,105,110,115]
# strikes = [i for i in range(70,116,2)]
print('strikes = ', strikes)
print('logMoney = ', np.array(strikes) / initialPrice)
# for loop here
ivValues = []
```

```
print('-------------Beginning computation-----------------')

def getSample(seed):
    torch.manual_seed(seed)
    dW = torch.randn(batchSize,2,timeSteps) * torch.sqrt(torch.tensor(dt))
    X = torch.ones(batchSize,2, timeSteps)
    X[:,:,0] = X[:,:,0] * torch.tensor([initialPrice, initialVol])
    for i in range(timeSteps -1):
        stockTerm1 = mu * X[:,0,i] * dt
        stockTerm2 = (.7 / (np.pi) * (torch.atan(X[:,1,i] - 1) + np.pi/2) + .03) * X[:,0, i]
                                                    * dW[:,0,i] # sigma(Y_t)
        stockTerm3 = -r * X[:,0,i] * dt
        X[:,0, i+1] = X[:,0, i] + stockTerm1 + stockTerm2 + stockTerm3
        volTerm1 = alpha * (m - X[:,1,i]) * dt
        volTerm2 = a * (rho *dW[:,0,i] + np.sqrt(1 - rho ** 2)* dW[:,1,i])
        X[:,1, i+1] = X[:,1, i] + volTerm1 + volTerm2
    return X.clone().detach(), dW.clone().detach()
lines = inspect.getsource(getSample)
print('getSample Function')
print(lines)
X, dW = getSample(0)
print('sample path = ', X[0,:,:])
# print('sample brownian motion = ', dW[0,:,:])
print('sample BM mean: ', torch.mean(dW), '| variance: ', torch.var(dW))

def fun_g(t,X, return0, K):
    # print(X.shape)
    if return0 == True:
        return torch.tensor(0)
    elif return0 == False:
        return torch.maximum(K * torch.exp(torch.tensor(-r * (t/timeSteps) * T)) - X[:,0,t],
                                                    torch.tensor(0))

print('terminal condition: ')
lines = inspect.getsource(fun_g)
print(lines)
def fun_f(X, t, z1, z2):
    sigma = (.7 / (np.pi) * (torch.atan(X[:,1,t] - 1) + np.pi/2) + .03)
    term1 = -1 * z1 * ((mu - r) / sigma)
    term2 = -1 * (1/(2 * gamma) * ((mu - r) / sigma) **2)
    term3 =  (gamma / 2) * z2 ** 2
    term4 = -1 * eta * ((mu - r) / sigma) * z2

    regular = term1 + term2 + term3 + term4

    return regular
print('driver function g hat')
lines = inspect.getsource(fun_f)
print(lines)

def d1(sigma, t, T, S, K, r):
    frac = 1 / (sigma * np.sqrt(T - t))
    secondfactor = np.log(S / K) + (r + ((sigma ** 2) / 2)) * (T - t)
    return frac * secondfactor

def d2(sigma, t, T, S, K, r):
    return d1(sigma, t, T, S, K, r) - (sigma * np.sqrt(T - t))

def put_price(sigma, t, T, S, K, r):
    return (norm.cdf(-d2(sigma, t, T, S, K, r)) * K * np.exp(-r * (T - t))) - (norm.cdf(-d1(
                                                    sigma, t, T, S, K, r)) * S)
def call_price(sigma, t, T, S, K, r):
    return S*norm.cdf(d1(sigma, t, T, S, K,r)) - K*np.exp(-r*(T - t)*norm.cdf(d2(sigma, t, T
                                                    , S, K, r)))
def implied_vol(price, t, T, S, K, r):
    sigma1 = 0.1
    sigma2 = 0.7
    value1 = put_price(sigma1, t, T, S, K, r) - price
```

```
    value2 = put_price(sigma2, t, T, S, K, r) - price
    while value1 * value2 > 0:
        sigma1 = sigma1 - 0.01
        sigma2 = sigma2 + 0.01
        value1 = put_price(sigma1, t, T, S, K, r) - price
        value2 = put_price(sigma2, t, T, S, K, r) - price
    i = 0
    while True:
        i = i + 1
        if value1 * value2 == 0:
            if value1 == 0:
                imp_vol = sigma1 #print('implied volatility=', sigma1)
            elif value2 == 0:
                imp_vol = sigma2 #print('implied volatility=', sigma2)
            break
        elif value1 * value2 < 0:
            midpoint = (sigma1 + sigma2) / 2
            if (put_price(midpoint, t, T, S, K, r) - price) * value1 < 0:
                sigma1 = sigma1
                sigma2 = midpoint
                value1 = put_price(sigma1, t, T, S, K, r) - price
                value2 = put_price(sigma2, t, T, S, K, r) - price

            elif (put_price(midpoint, t, T, S, K, r) - price) * value2 < 0:
                sigma1 = midpoint
                sigma2 = sigma2
                value1 = put_price(sigma1, t, T, S, K, r) - price
                value2 = put_price(sigma2, t, T, S, K, r) - price

        if abs(sigma1 - sigma2) < 0.000001:
            imp_vol = sigma2
            break
    return imp_vol


def trainWithPayoff():
    models = {}
    params = list()
    torch.manual_seed(0)
    for i in range(timeSteps -1, -1, -1): # nn.LayerNorm(batchSize),
        models[i] = nn.Sequential(nn.LayerNorm((batchSize, 2)), nn.Linear(2, 12) ,nn.ReLU(),
                                                    nn.ReLU(), nn.Linear(12, 2))
        params += list(models[i].parameters())

    optimizer = torch.optim.SGD(params,  lr=learningRate)
    criterion = nn.MSELoss()
    return0Condition = False
    for epoch in range(0, numEpochs):
        optimizer.zero_grad()
        X, dW = getSample(epoch)
        previousNetworkResults = fun_g(timeSteps-1, X, return0Condition, K)



        if epoch % 500 == 0:
                print('timestep = ', timeSteps-1, '| mean = ', round(torch.mean(
                                                        previousNetworkResults).item()
                                                        ,3))
        for timeStep in range(timeSteps -2, -1, -1):
            # print('timeStep =', timeStep)

            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults - fun_f(X, timeStep, -1 *
                                                    modelResults[:,0], -1 *
                                                    modelResults[:,1])* torch.tensor(
                                                    dt) - torch.diag(torch.inner(
                                                    modelResults, dW[:,:,timeStep]))
```

```
            currentNetworkResults = torch.maximum(fun_g(timeStep, X, return0Condition, K),
                                                  currentNetworkResults)
            previousNetworkResults = currentNetworkResults
            if epoch % 500 == 0:
                print('timestep = ', timeStep, '| mean = ', round(torch.mean(
                                                      currentNetworkResults).item(),
                                                      3)) # '| currentResults = ',
                                                      currentNetworkResults[:3],


        loss = torch.var(currentNetworkResults)
        loss.backward()
        clip_value = 5
        torch.nn.utils.clip_grad_norm_(params, clip_value)
        optimizer.step()



        if epoch % 500 == 0:
            print('epoch = ',epoch, '| loss: ', round(loss.item(),4))
            print('================================')
    return currentNetworkResults, models


lines = inspect.getsource(trainWithPayoff)
print('train with payoff')
print(lines)

def computeBoundary(models):
    boundaries = torch.empty((numEpochs, batchSize, timeSteps)) # boundary is only concerned
                                                  with the stock process
    paths = torch.empty((numEpochs, batchSize, timeSteps))
    tauStars = torch.ones((numEpochs, batchSize)) # just need the tau value
    return0Condition = False
    for epoch in range(0, numEpochs):
        X, dW = getSample(epoch)
        stockPath = X[:,0,:]
        boundary = torch.maximum(K* torch.exp(torch.tensor((-r  / timeSteps)*T) * torch.
                                                Tensor([i for i in range(timeSteps)]))
                                                  - stockPath, torch.tensor(0))
        boundaries[epoch,:,:] = boundary.clone().detach() # store the boundary

        path = torch.empty((batchSize, timeSteps))
        path[:,timeSteps - 1] = boundary[:,timeSteps - 1]
        previousNetworkResults = boundary[:,timeSteps - 1]
        for timeStep in range(timeSteps -2, -1, -1):
            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults - fun_f(X, timeStep, -1 *
                                                  modelResults[:,0], -1 *
                                                  modelResults[:,1])* torch.tensor(
                                                  dt) - torch.diag(torch.inner(
                                                  modelResults, dW[:,:,timeStep]))
            currentNetworkResults = torch.maximum(fun_g(timeStep, X, return0Condition, K),
                                                  currentNetworkResults)
            previousNetworkResults = currentNetworkResults
            path[:, timeStep] = currentNetworkResults
        paths[epoch,:,:] = path.clone().detach()
        belowBoundary = (path <= boundary).clone().detach()
        belowBoundary = belowBoundary.long().clone().detach()
        tauStar = torch.where(belowBoundary.any(axis=1), belowBoundary.argmax(axis=1),
                                                  timeSteps-1).long() # was argmax
        tauStars[epoch,:] = tauStar.long().clone().detach()
        if epoch % 500 == 0:
            print('epoch = ', epoch)
    return tauStars


lines = inspect.getsource(computeBoundary)
```

```
print('compute boundary function')
print(lines)

def trainWithZero(tauStars):
    models = {}
    params = list()
    torch.manual_seed(0)
    for i in range(timeSteps -1, -1, -1): # nn.LayerNorm(batchSize),
        # models[i] = nn.Sequential(nn.Flatten(0),nn.LayerNorm(batchSize * 2) ,  nn.Linear(
                                                    batchSize * 2, 11) ,nn.ReLU(),  nn.
                                                    ReLU(), nn.Linear(11, batchSize * 2),
                                                    nn.Unflatten(0, (batchSize, 2))) # ,
                                                    nn.LayerNorm(batchSize)
        models[i] = nn.Sequential(nn.LayerNorm((batchSize, 2)), nn.Linear(2, 12) ,nn.ReLU(),
                                                      nn.ReLU(),nn.ReLU(), nn.Linear(12, 2
                                                    ))
        params += list(models[i].parameters())

    optimizer = torch.optim.SGD(params,  lr=learningRate)
    criterion = nn.MSELoss()
    return0Condition = True
    for epoch in range(0, numEpochs):
        optimizer.zero_grad()
        X, dW = getSample(epoch)
        previousNetworkResults = fun_g(timeSteps-1, X, return0Condition, K)
        # print(previousNetworkResults)


        if epoch % 500 == 0:
                print('timestep = ', timeSteps-1, '| mean = ', 0)
        path = torch.empty((batchSize, timeSteps))
        path[:,timeSteps - 1] = torch.tensor(0)
        for timeStep in range(timeSteps -2, -1, -1):
            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults + fun_f(X, timeStep, modelResults
                                                    [:,0], modelResults[:,1])* torch.
                                                    tensor(dt) - torch.diag(torch.
                                                    inner(modelResults, dW[:,:,
                                                    timeStep]))
            previousNetworkResults = currentNetworkResults
            path[:, timeStep] = currentNetworkResults
            if epoch % 500 == 0:
                print('timestep = ', timeStep, '| mean = ', round(torch.mean(
                                                        currentNetworkResults).item(),
                                                        8)) # '| currentResults = ',
                                                        currentNetworkResults[:3],
        currentNetworkResults = path[:,tauStars[epoch,:].long()]
        # from # https://discuss.pytorch.org/t/get-the-value-at-a-specific-index-in-pytorch/
                                                    134097

        loss = torch.var(currentNetworkResults)
        loss.backward()
        clip_value = 5
        torch.nn.utils.clip_grad_norm_(params, clip_value)
        optimizer.step()



        if epoch % 500 == 0:
            print('epoch = ',epoch, '| loss: ', round(loss.item(),4))
            print('================================')
    return currentNetworkResults

lines = inspect.getsource(trainWithZero)
print('train with zero function')
print(lines)

print('==============COMPUTATION==============')
```

```
for K in strikes:
    print('strike = ', K)
    payoffPrice, models = trainWithPayoff()
    payoffPrice = torch.mean(payoffPrice).item()
    print('payoffPrice = ', payoffPrice)
    print('Computing TauStar')
    tauStars = computeBoundary(models)
    zeroConditionPrice = trainWithZero(tauStars)
    zeroConditionPrice = torch.mean(zeroConditionPrice).item()
    print('return 0 bsde value = ', zeroConditionPrice)
    ivVal = implied_vol(payoffPrice + zeroConditionPrice, 0, T, 100,K, r)
    print('indifference price: ', payoffPrice + zeroConditionPrice)
    print('ivVal = ', ivVal)
    ivValues.append(ivVal)
    print('------------------------------------------------ \n \n \n')

print('all the Ivs: ', ivValues)
```

A.3 American Put from the Seller's Perspective

```
   import torch
import torch.nn as nn
import numpy as np
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch.autograd import Variable
import numpy as np
from scipy.stats import norm
import inspect
print('American Put: Sellers, SGD optimizer')
batchSize = 500# 1000
numEpochs = 4001 # 4001
learningRate = .01 # .01
print('batchSize = ', batchSize)
print('NumEpochs = ', numEpochs)
print('Learning Rate = ', learningRate)
initialPrice = 100
print('initial price = ', initialPrice)
initialVol = .149793 # .3
print('initial Vol = ', initialVol)
mu = .08 # .04
print('mu = ', mu )
alpha = 5
print('alpha = ', alpha)
m = 0
print('m = ', m)
nu = 1
print('nu = ', nu)
a = nu * torch.sqrt(torch.tensor(2 * alpha))
print('nu * sqrt(2 alpha ) = ', a)
rho = -.2
print('rho = ', rho)
gamma = 1
print('gamma = ', gamma )
eta = .2
print('eta = ', eta)
timeSteps = 25 # 10
print('Total number of time steps = ', timeSteps )
r = 0.02 # 0.05
print('risk free rate r = ', r)
T = 1/4
print('T = ', T)
dt = T / timeSteps
print('dt = ', dt)
strikes = [70,75,80,85,90,95,100,105,110,115]
```

```
# strikes = [i for i in range(70,116,2)]
print('strikes = ', strikes)
print('logMoney = ', np.array(strikes) / initialPrice)
# for loop here
ivValues = []
print('-------------Beginning computation-----------------')
def getSample(seed):
    torch.manual_seed(seed)
    dW = torch.randn(batchSize,2,timeSteps) * torch.sqrt(torch.tensor(dt))
    X = torch.ones(batchSize,2, timeSteps)
    X[:,:,0] = X[:,:,0] * torch.tensor([initialPrice, initialVol])
    for i in range(timeSteps -1):
        stockTerm1 = mu * X[:,0,i] * dt
        stockTerm2 = (.7 / (np.pi) * (torch.atan(X[:,1,i] - 1) + np.pi/2) + .03) * X[:,0, i]
                                                    * dW[:,0,i] # sigma(Y_t)
        stockTerm3 = -r * X[:,0,i] * dt
        X[:,0, i+1] = X[:,0, i] + stockTerm1 + stockTerm2 + stockTerm3
        volTerm1 = alpha * (m - X[:,1,i]) * dt
        volTerm2 = a * (rho *dW[:,0,i] + np.sqrt(1 - rho ** 2)* dW[:,1,i])
        X[:,1, i+1] = X[:,1, i] + volTerm1 + volTerm2
    return X.clone().detach(), dW.clone().detach()
lines = inspect.getsource(getSample)
print('getSample Function')
print(lines)
X, dW = getSample(0)
print('sample path = ', X[0,:,:])
# print('sample brownian motion = ', dW[0,:,:])
print('sample BM mean: ', torch.mean(dW), '| variance: ', torch.var(dW))

def fun_g(t,X, return0, K):
    # print(X.shape)
    if return0 == True:
        return torch.tensor(0,dtype=torch.float64)
    elif return0 == False:
        return torch.maximum(K * torch.exp(torch.tensor(-r * (t/timeSteps) * T)) - X[:,0,t],
                                                    torch.tensor(0))
print('terminal condition: ')
lines = inspect.getsource(fun_g)
print(lines)
def fun_f(X, t, z1, z2):
    sigma = (.7 / (np.pi) * (torch.atan(X[:,1,t] - 1) + np.pi/2) + .03)
    term1 = -1 * z1 * ((mu - r) / sigma)
    term2 = -1 * (1/(2 * gamma) * ((mu - r) / sigma) **2)
    term3 =  (gamma / 2) * z2 ** 2
    term4 = -1 * eta * ((mu - r) / sigma) * z2

    regular = term1 + term2 + term3 + term4

    return regular
print('driver function g hat')
lines = inspect.getsource(fun_f)
print(lines)


def d1(sigma, t, T, S, K, r):
    frac = 1 / (sigma * np.sqrt(T - t))
    secondfactor = np.log(S / K) + (r + ((sigma ** 2) / 2)) * (T - t)
    return frac * secondfactor

def d2(sigma, t, T, S, K, r):
    return d1(sigma, t, T, S, K, r) - (sigma * np.sqrt(T - t))

def put_price(sigma, t, T, S, K, r):
    return (norm.cdf(-d2(sigma, t, T, S, K, r)) * K * np.exp(-r * (T - t))) - (norm.cdf(-d1(
                                                    sigma, t, T, S, K, r)) * S)
def call_price(sigma, t, T, S, K, r):
    return S*norm.cdf(d1(sigma, t, T, S, K,r)) - K*np.exp(-r*(T - t)*norm.cdf(d2(sigma, t, T
                                                    , S, K, r)))
```

```
def implied_vol(price, t, T, S, K, r):
    sigma1 = 0.1
    sigma2 = 0.7
    value1 = put_price(sigma1, t, T, S, K, r) - price
    value2 = put_price(sigma2, t, T, S, K, r) - price
    while value1 * value2 > 0:
        sigma1 = sigma1 - 0.01
        sigma2 = sigma2 + 0.01
        value1 = put_price(sigma1, t, T, S, K, r) - price
        value2 = put_price(sigma2, t, T, S, K, r) - price
    i = 0
    while True:
        i = i + 1
        if value1 * value2 == 0:
            if value1 == 0:
                imp_vol = sigma1 #print('implied volatility=', sigma1)
            elif value2 == 0:
                imp_vol = sigma2 #print('implied volatility=', sigma2)
            break
        elif value1 * value2 < 0:
            midpoint = (sigma1 + sigma2) / 2
            if (put_price(midpoint, t, T, S, K, r) - price) * value1 < 0:
                sigma1 = sigma1
                sigma2 = midpoint
                value1 = put_price(sigma1, t, T, S, K, r) - price
                value2 = put_price(sigma2, t, T, S, K, r) - price

            elif (put_price(midpoint, t, T, S, K, r) - price) * value2 < 0:
                sigma1 = midpoint
                sigma2 = sigma2
                value1 = put_price(sigma1, t, T, S, K, r) - price
                value2 = put_price(sigma2, t, T, S, K, r) - price

        if abs(sigma1 - sigma2) < 0.000001:
            imp_vol = sigma2
            break
    return imp_vol

def trainWithZero():
    models = {}
    params = list()
    torch.manual_seed(0)
    for i in range(timeSteps -1, -1, -1): # nn.LayerNorm(batchSize),
        models[i] = nn.Sequential(nn.LayerNorm((batchSize, 2)), nn.Linear(2, 12) ,nn.ReLU(),
                                                                nn.ReLU(), nn.Linear(12, 2))
        params += list(models[i].parameters())

    optimizer = torch.optim.SGD(params,  lr=learningRate)
    return0Condition = True
    for epoch in range(0, numEpochs):
        optimizer.zero_grad()
        X, dW = getSample(epoch)
        previousNetworkResults = fun_g(timeSteps-1, X, return0Condition, K)



        if epoch % 500 == 0:
                print('timestep = ', timeSteps-1, '| mean = ', round(torch.mean(
                                                                  previousNetworkResults).item()
                                                                  ,3))
        for timeStep in range(timeSteps -2, -1, -1):
            # print('timeStep =', timeStep)

            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults + fun_f(X, timeStep, modelResults
                                                                  [:,0], modelResults[:,1])* torch.
                                                                  tensor(dt) - torch.diag(torch.
                                                                  inner(modelResults, dW[:,:,
```

```
                                                    timeStep]))
            previousNetworkResults = currentNetworkResults
            if epoch % 500 == 0:
                print('timestep = ', timeStep, '| mean = ', round(torch.mean(
                                                        currentNetworkResults).item(),
                                                        3))
        loss = torch.var(currentNetworkResults)
        loss.backward()
        clip_value = 5
        torch.nn.utils.clip_grad_norm_(params, clip_value)
        optimizer.step()



        if epoch % 500 == 0:
            print('epoch = ',epoch, '| loss: ', round(loss.item(),4))
            print('================================')
    return currentNetworkResults, models
lines = inspect.getsource(trainWithZero)
print('train with 0 terminal condition')
print(lines)

def computeBoundary(models):
    L = torch.empty([numEpochs, batchSize, timeSteps])
    for epoch in range(numEpochs):
        X, dW = getSample(epoch)
        previousNetworkResults = fun_g(timeSteps-1, X, True, K)
        for timeStep in range(timeSteps -2, -1, -1):
            # print('timeStep =', timeStep)
            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults + fun_f(X, timeStep, modelResults
                                                    [:,0], modelResults[:,1])* torch.
                                                    tensor(dt) - torch.diag(torch.
                                                    inner(modelResults, dW[:,:,
                                                    timeStep]))
            previousNetworkResults = currentNetworkResults
            L[epoch,:,timeStep] = currentNetworkResults + fun_g(timeStep, X, False, K)
        if epoch % 500 == 0:
            print('epoch = ', epoch)
    L = L.clone().detach()
    return L
lines = inspect.getsource(computeBoundary)
print('compute boundary function')
print(lines)


def trainWithPayoff(L):
    models = {}
    params = list()
    torch.manual_seed(0)
    for i in range(timeSteps -1, -1, -1): # nn.LayerNorm(batchSize),
        models[i] = nn.Sequential(nn.LayerNorm((batchSize, 2)), nn.Linear(2, 12) ,nn.ReLU(),
                                                    nn.ReLU(), nn.Linear(12, 2))
        params += list(models[i].parameters())

    optimizer = torch.optim.SGD(params,  lr=learningRate)
    return0Condition = False
    for epoch in range(0, numEpochs):
        optimizer.zero_grad()
        X, dW = getSample(epoch)
        previousNetworkResults = fun_g(timeSteps-1, X, return0Condition, K)



        if epoch % 500 == 0:
                print('timestep = ', timeSteps-1, '| mean = ', round(torch.mean(
                                                        previousNetworkResults).item()
                                                        ,3))
```

```
        for timeStep in range(timeSteps -2, -1, -1):
            # print('timeStep =', timeStep)

            modelResults = models[timeStep](X[:,:,timeStep])
            currentNetworkResults = previousNetworkResults + fun_f(X, timeStep, modelResults
                                                    [:,0], modelResults[:,1])* torch.
                                                    tensor(dt) - torch.diag(torch.
                                                    inner(modelResults, dW[:,:,
                                                    timeStep]))
            currentNetworkResults = torch.maximum(L[epoch,:,timeStep], currentNetworkResults
                                                    )
            previousNetworkResults = currentNetworkResults
            if epoch % 500 == 0:
                print('timestep = ', timeStep, '| mean = ', round(torch.mean(
                                                        currentNetworkResults).item(),
                                                        3)) # '| currentResults = ',
                                                        currentNetworkResults[:3],


        loss = torch.var(currentNetworkResults)
        loss.backward()
        clip_value = 5
        torch.nn.utils.clip_grad_norm_(params, clip_value)
        optimizer.step()



        if epoch % 500 == 0:
            print('epoch = ',epoch, '| loss: ', round(loss.item(),4))
            print('================================')
    return currentNetworkResults
lines = inspect.getsource(trainWithPayoff)
print('train with payoff')
print(lines)



print('==============COMPUTATION==============')
for K in strikes:
    print('Strike = ', K)
    print('Computing the Return 0 BSDE')
    zeroConditionPrice, models = trainWithZero()
    zeroConditionPrice = torch.mean(zeroConditionPrice).item()
    print('zeroConditionPrice = ', zeroConditionPrice)
    L = computeBoundary(models)
    print('Computing the Boundary')
    print('Boundary Computed')
    print('============Computing the RBSDE============')
    payoffPrice = trainWithPayoff(L)
    payoffPrice = torch.mean(payoffPrice).item()
    print('rbsde done')
    print('payoffPrice = ', payoffPrice)
    print('Indifference Price, Seller: ', payoffPrice - zeroConditionPrice)
    print('implied vol: ', implied_vol(payoffPrice - zeroConditionPrice, 0, T, 100, K, r))
    ivValues.append(implied_vol(payoffPrice - zeroConditionPrice, 0, T, 100, K, r))
print('iv Values = ', ivValues)
```