REINFORCEMENT LEARNING FOR DERIVATIVE PRICING UNDER THE UNCERTAIN VOLATILITY MODEL

by Thomas Leygonie

REINFORCEMENT LEARNING FOR DERIVATIVE PRICING UNDER THE UNCERTAIN VOLATILITY MODEL

by

Thomas Leygonie (CID: 01601115)

Department of Mathematics
Imperial College London
London SW7 2AZ
United Kingdom

Thesis submitted as part of the requirements for the award of the MSc in Mathematics and Finance, Imperial College London, 2018-2019

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signature and date:

[illegible]

10th September 2019

Acknowledgements

First of all, I would like to thank my external supervisor Dr Daniel Bloch for giving me the opportunity to deal with this very interesting topic. I also would like to thank my internal supervisor Dr Antoine Jacquier for his support, his positive attitude in spite of the difficulties I encountered and his insightful suggestions throughout this project.

In addition, I would like to thank Dr Alex Tse and Dr Eyal Neuman for their time and their support.

Special thanks to Dr Marco Avellaneda, for taking time to explain me carefully his model. I also would like to thank Mrs Carol Hutchins for providing me Antonio Paras' thesis.

I am grateful to my friend Hossein. Throughout this project he provided me an enormous number of relevant references and ideas. He was always available to help me at every step of my project.

Last but not least, I wish to thank a lot my friend Michel and my mother. Their friendship/love allows me to overcome all the obstacles I encountered throughout this year.

To my grandmother, Anne-Marie.

List of accronyms

BOE Bellman optimality equation

BS Black-Scholes

BSB Black-Scholes-Barenblatt

DQL Deep Q-learning

MDP Markov decision process

RL Reinforcement Learning

SARSA state action reward state action

UVM Uncertain volatility model

List of mathematical symbols

$[\![a, b]\!]$	Set of all integers between the reals $a \leq b$ (equal to $\mathbb{Z} \cap [a, b]$)
$\|f\|_\infty$	Uniform norm of the function $f : X \to \mathbb{R}$ where X is a set (equal to $\sup\limits_{x \in X} \lvert f(x) \rvert$)
$\lvert X \rvert$	Cardinal of the (finite) set X
$\mathbb{E}^{\mathbb{Q}}$	Expectation under risk-neutral/pricing measure
$\mathbb{E}_\pi$	Expectation depending on the policy
$\mathbb{N}^*$	Set of strictly positive integers
$\mathbb{Q}$	Risk-neutral/pricing measure
$\mathbb{R}, \mathbb{N}, \mathbb{Z}$	Respectively sets of real numbers, non-negative integers and integers
$\mathbb{R}_+$	Set of non-negatives real numbers (equal to $[0, +\infty[$)
$\mathbb{R}_+^*$	Set of strictly positive real numbers (equal to $]0, +\infty[$)

$\mathcal{A}$	Action space of the Markov decision process
$\mathcal{B}(X)$	σ-field generated by the borel set $X \subset \mathbb{R}^d$
$\mathcal{C}^{i,j}(I \times J, \mathbb{R})$	Set of functions from the product of intervals $I \times J$ to $\mathbb{R}$ that are i times continuously differentiable for the first variable and j times continuously differentiable for the second variable
$\mathcal{D}$	Memory set used for experience replay
$\mathcal{F} \bigotimes \mathcal{G}$	Product σ-field generated by the σ-fields $\mathcal{F}$ and $\mathcal{G}$
$\mathcal{L}$	Loss function (mean squared error)
$\mathcal{S}$	State space of the Markov decision process
$\nabla_w f(x, w)$	Gradient of the function f at point (x, w) with respect to the vector w
ϕ	Probability density function of a normal distribution of mean 0 and standard deviation 1
π	Policy of the Markov decision process
$\sigma(X_i, i \in I)$	σ-field generated by the family of random variables $(X_i)_{i \in I}$
$L^\infty(X)$	Set of bounded and measurable functions from X to $\mathbb{R}$ where $X \subset \mathbb{R}^d$ is a borel set
$L^{Pol}(X)$	Set of measurable functions from X to $\mathbb{R}$ with polynomial growth ($i.e$ for a given function f, there exists $K > 0$ and $m \in \mathbb{N}$ such that $\forall x \in X, \lvert f(x)\rvert \leq K(1 + \lvert x\rvert^m)$) where $X \subset \mathbb{R}$ is a borel set

Contents

1 Introduction

Derivative pricing is a central issue in financial markets. To fix this issue, numerous mathematical models have been introduced. One of the most used is the Black-Scholes model (see [BS73]). This model considers the stock price as a geometric brownian motion depending on a drift (an average increasing factor) and a volatility parameter (an noise factor). It is extremely tractable and thus able to give an analytical price for most of derivatives introduced in financial markets. However, the Black-Scholes model has one relevant drawback: the volatility parameter is generally both unknown and difficult to estimate accurately.

A way to overcome that problem is to consider all the possible volatility processes/scenarios and to assume that all these processes lie between two bounds σ_{min} and σ_{max}. This yields a new model known as the uncertain volatility model (see [ALP95]). Even if this model is less tractable than Black-Scholes, there exists methods to give a price for derivative securities under this model in the best/worst scenario.

In this thesis, we estimate the highest/lowest possible price under the uncertain volatility model using a Reinforcement Learning approach. Usually, Reinforcement Learning consists in helping an agent (*e.g* a machine, a computer, a robot) interact correctly to a given environment (*e.g* a financial market, a video game...). To train the agent, we teach it how to select the best action by rewarding it. During training, the goal of the agent is to maximise the expected sum of rewards.

In our case, we use Reinforcement Learning to only estimate this maximal expectation of sum of rewards. Reinforcement Learning for derivative pricing has already been used (see for instance [Gra] or [Hal17]) but has never been used to find a price under the uncertain volatility model.

In this thesis, we present independently a state-of-the-art on Reinforcement Learning (section 2) and Mathematical Finance (section 3) where we explain the Black-Scholes and uncertain volatility models. Afterwards, we construct our own Reinforcement Learning model in order to estimate the extreme UVM prices (section 4). We also show in this section why the model approximates correctly those prices in theory. Finally we detail the implementation of that model and the results (section 5).

2 Reinforcement Learning

In this section, we explain how a Reinforcement Learning (RL) problem is formulated by giving some definitions. Afterwards, we explain how to solve this problem. Most of this part is based on [SB18] and [MRT18, Chapter 17].

2.1 Definition of the problem

We firstly define a Markov decision process by introducing the mathematical formalism.

Definition 2.1. Let $d, m \in \mathbb{N}^*$ fixed. A **state space** $\mathcal{S}$ (resp. an **action space** $\mathcal{A}$) is a Borel subset of $\mathbb{R}^d$ (resp. $\mathbb{R}^m$). We call $\mathcal{B}(\mathcal{S})$ (resp. $\mathcal{B}(\mathcal{A})$) its Borel σ-field.

Let $X := (X_n)_{n\in\mathbb{N}}$ be a sequence of $\mathcal{S}$ -valued random variables. Let $(\mathcal{F}_n)_{n\in\mathbb{N}}$ be the canonical filtration generated by X ($i.e$ $\forall n \in \mathbb{N}$, $\mathcal{F}_n := \sigma(X_0, ..., X_n)$), and let $A := (A_n)_{n\in\mathbb{N}}$ be an $(\mathcal{F}_n)_{n\in\mathbb{N}}$-adapted sequence of $\mathcal{A}$ -valued random variables

Definition 2.2. The process (X, A) is said to be a **Markov decision process (MDP)** if and only if there exists two functions

$$p : \begin{array}{rcl} \mathcal{B}(\mathcal{S}) \times \mathcal{S} \times \mathcal{A} & \to & [0,1] \\ (B, x, a) & \to & p(B|x,a) \end{array} \quad \text{and } \pi : \begin{array}{rcl} \mathcal{B}(\mathcal{A}) \times \mathcal{S} & \to & [0,1] \\ (B', x) & \to & \pi(B'|x) \end{array}$$

such that for every $(n, B, B') \in \mathbb{N} \times \mathcal{B}(\mathcal{S}) \times \mathcal{B}(\mathcal{A})$:

$$\begin{cases} \mathbb{P}\left[A_n \in B' | \mathcal{F}_n\right] & = \pi(B'|X_n) \\ \mathbb{P}\left[X_{n+1} \in B | \mathcal{F}_n\right] & = p(B|X_n, A_n) \end{cases} \tag{2.1}$$

In other words, given a state X_n, the action A_n depends (potentially randomly) only on the current state X_n and the next state X_{n+1} depends (potentially randomly) only on the current state X_n and the current action A_n. Note the following sequence of events we have in a MDP:

$$X_n \to A_n \to (X_{n+1}, R(X_n, A_n))$$

Where R is a reward function (see definition 2.5 below). In Reinforcement learning, given a state X_n, we have to find the best action A_n to optimise a sum of reward functions.

If $\mathcal{S} \times \mathcal{A}$ is finite or countable, every probability distribution on this space is characterised by the probabilities of their singletons. This is why for the sake of simplicity, we denote $p(x'|x,a) := p(\{x'\}|x,a)$ and $\pi(a|x) := \pi(\{a\}|x)$ for every $(x,a,x') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$.

Remark 2.1. We can notice that the functions p and π in definition 2.2 do not depend on time $n \in \mathbb{N}$. If a process (S,A) verifies equation (2.1) where π or p depends on n, the process (X,A) defined by $X_n := (S_n, n)$ for every $n \in \mathbb{N}$ is then an MDP according to definition 2.2 (the state space $\mathcal{S}$ is also replaced by $\mathcal{S}' = \mathcal{S} \times \mathbb{N}$)

The choice of A_n with respect to X_n is mathematically given by the policy. This notion is introduced in the two following definitions.

Definition 2.3. Let (X,A) be a MDP. A **stochastic policy** is the function

$$\pi : \begin{array}{ccc} \mathcal{B}(\mathcal{A}) \times \mathcal{S} & \rightarrow & [0,1] \\ (B',x) & \rightarrow & \pi(B'|x) \end{array}$$

in definition 2.2.

Definition 2.4. The MDP (X,A) is said to have a **deterministic action** if and only if its stochastic policy $\pi^{(sto)}$ is deterministic $i.e$ there exists a measurable function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ such that:

$$\forall (B',x) \in \mathcal{B}(\mathcal{A}) \times \mathcal{S},\ \pi^{(sto)}(B'|x) = \begin{cases} 1 & \text{if } \pi(x) \in B' \\ 0 & \text{otherwise} \end{cases}$$

π is the **deterministic policy**.

The intuition behind the last definition is as follows: given the current state X_n, one chooses the action according to a deterministic function π (and not randomly).

The action A_n is then simply given by the relation $A_n = \pi(X_n)$.

A policy is either stochastic ($\pi(B'|x)$ is function of two variables and represents a probability) or deterministic ($\pi(x)$ is function of one variable and represents the action chosen). Throughout the thesis, a policy is *a priori* assumed to be stochastic (because it is a more general definition) except if the contrary is explicitly mentioned.

Definition 2.5. Let (X, A) be a MDP. Let $\gamma \in [0, 1]$. A **reward function** R is a measurable function:

$$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

Such that for every policy π, the sum of the discounted back rewards is integrable, that is:

$$\forall x \in \mathcal{S}, \mathbb{E}_\pi \left[\sum_{k=0}^{+\infty} \left| \gamma^k R(X_k, A_k) \right| \middle| X_0 = x \right] < +\infty$$

Where $\mathbb{E}_\pi$ denotes the expectation under the policy π

The goal of Reinforcement Learning is to maximise the expectation:

$$\mathbb{E}_\pi \left[\sum_{k=0}^{+\infty} \gamma^k R(X_k, A_k) \middle| X_0 \right]$$

with respect to the policy π, the only variable we can modify.

2.2 The value function and the Q-value function

In this sub-section, we define from an MDP (X, A) the two main functions used in Reinforcement Learning, namely the value function and Q-value function. We also derive several theoretical results that will be useful to understand the RL approach. These results are based on proofs of [MRT18, Chapter 17].

2.2.1 Definition of V_π and V_*

The value function is the expected sum of rewards we wish to optimise. More precisely:

Definition 2.6. Let (X, A) be an MDP with a policy π, let $\gamma \in [0,1]$ be fixed and let R be a reward function. The **value function** $V_\pi : \mathcal{S} \to \mathbb{R}$ is defined by the expression:

$$V_\pi(x) := \mathbb{E}_\pi \left[\sum_{k=0}^{+\infty} \gamma^k R(X_k, A_k) \middle| X_0 = x \right]$$

Given an initial state $X_0 = x$, we wish to find:

$$V_*(x) := \sup_{\pi \text{ policy}} V_\pi(x)$$

An optimal policy (if it exists) is denoted by π^*. We finally give a useful definition for the rest of the thesis.

Definition 2.7. An MDP (X, A) with discount factor γ and reward function R is said to have a **finite horizon** if and only if there exists $N \in \mathbb{N}$ such that:

$$\forall \pi \text{ policy}, \forall x \in \mathcal{S}, V_\pi(x) = \mathbb{E}_\pi \left[\sum_{k=0}^{N} R(X_k, A_k) \middle| X_0 = x \right]$$

2.2.2 First Properties

We immediately state a first relevant result (see appendix A for the proof).

Proposition 2.1. Let π be a policy. Then for every $x \in \mathcal{S}$, the value function has the following recursive relation:

$$V_\pi(x) = \mathbb{E}_\pi \left[R(x, A_0) + \gamma V_\pi(X_1) | X_0 = x \right] \tag{2.2}$$

If $\mathcal{S} \times \mathcal{A}$ is countable, this relation becomes:

$$V_\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left(R(x,a) + \gamma \sum_{x' \in \mathcal{S}} p(x'|x,a) V_\pi(x') \right) \tag{2.3}$$

These two relations are both known as the **Bellman equation**.

If additional conditions hold, the Bellman equation characterises V_π as stated below (see appendix A for the proof).

Proposition 2.2. Suppose that $\gamma < 1$ and V_π is bounded. Then V_π is the unique solution of Bellman equation (2.2) in $L^\infty(\mathcal{S})$ the space of bounded and measurable functions from $\mathcal{S}$ to $\mathbb{R}$.

2.2.3 Definition of Q_π and Q_*

We now define the Q-value function. This function is close to the value function but the action at time 0 A_0 is assumed to be known in addition to X_0. More precisely:

Definition 2.8. Let (X, A) be a MDP with policy π, let $\gamma \in [0, 1]$ fixed and let R be a reward function. A **Q-value function** $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined by the expression:

$$Q_\pi(x, a) := R(x, a) + \mathbb{E}_\pi \left[\sum_{k=1}^{+\infty} \gamma^k R(X_k, A_k) \middle| X_0 = x, A_0 = a \right]$$

Similarly as value function, one can define the optimal Q-value function. Given an initial state $X_0 = x$ and an initial action $A_0 = a$ the optimal Q-value function is defined by the expression:

$$Q_*(x, a) := \sup_{\pi \text{ policy}} Q_\pi(x, a)$$

2.2.4 Further properties

We give here some results on the value function and the Q-value function. Most of these results are not proved in this thesis but the proofs can be found in [MRT18, Chapter 17]. The interpretation of these results is presented in the next sub-section. Let us start with a first remark linking the value function to the Q-value function:

Remark 2.2. Given a fixed policy π, knowing V_π is equivalent to knowing Q_π thanks to the relations:

$$\forall (x, a) \in \mathcal{S} \times \mathcal{A}, \begin{cases} V_\pi(x) &= \mathbb{E}_\pi \left[Q_\pi(x, A_0) | X_0 = x \right] \\ Q_\pi(x, a) &= R(x, a) + \gamma \mathbb{E} \left[V_\pi(X_1) | X_0 = x, A_0 = a \right] \end{cases}$$

Where $\mathbb{E}$ is an expectation independent of the policy π. If $\mathcal{S} \times \mathcal{A}$ is countable, these relations become:

$$\forall (x, a) \in \mathcal{S} \times \mathcal{A}, \begin{cases} V_\pi(x) &= \sum_{a' \in \mathcal{A}} \pi(a'|x) Q_\pi(x, a') \\ Q_\pi(x, a) &= R(x, a) + \gamma \sum_{x' \in \mathcal{S}} p(x'|x, a) V_\pi(x') \end{cases} \tag{2.4}$$

The proof of those results is similar to the proof of proposition 2.1.

We have introduced the Q-value function because of the following result (see [MRT18, Theorem 17.6] for a proof):

Proposition 2.3. Let (X, A) be a MDP with policy π. Suppose that either $\gamma < 1$ and V_π is bounded on $\mathcal{S}$, or (X, A) has a finite horizon. Let π' be another policy. We then have:

$$(\forall x \in \mathcal{S}, \mathbb{E}_{\pi'}\left[Q_\pi(x, A_0)|X_0 = x\right] \geq V_\pi(x)) \implies (\forall x \in \mathcal{S}, V_{\pi'}(x) \geq V_\pi(x))$$

And if the inequality is strict for at least one state on the left-hand-side, then the inequality on the right-hand-side is also strict for at least one state. π' is said to be **better than** π if and only if the inequality on the right-hand side is satisfied.

A particular policy is always better than the previous one. This policy is easy to set if Q_π is known and $\mathcal{A}$ is small.

Definition 2.9. Under conditions of proposition 2.3 and assuming further that $\mathcal{A}$ is finite, the deterministic policy π' defined by the expression:

$$\pi'(x) := \operatorname*{argmax}_{a \in \mathcal{A}} Q_\pi(x, a)$$

is called a **greedy policy** (if several values are possible, take only one of them). This policy is better than π according to the above proposition.

Thanks to this proposition and this definition, we henceforth have a method to optimise a policy. Note that the boundedness condition is satisfied if $\mathcal{S} \times \mathcal{A}$ is finite. We now give a condition on policy optimality (see [MRT18, Theorem 17.7] for a proof):

Proposition 2.4. Let (X, A) be an MDP with policy π. Suppose that either $\gamma < 1$ and V_π is bounded on $\mathcal{S}$, or (X, A) has a finite horizon. Then π is optimal if and only if:

$$\forall (x, a) \in \mathcal{S} \times \mathcal{A}, \left(\pi(a|x) > 0 \implies a \in \operatorname*{argmax}_{a' \in \mathcal{A}} Q_\pi(x, a')\right)$$

This result shows that if the policy cannot be improved any more (using the greedy policy method), then this policy is nothing but an optimal policy. We now state a result when $\mathcal{S} \times \mathcal{A}$ is finite (see [MRT18, Theorem 17.8] for a proof).

Proposition 2.5. Let (X, A) be a MDP. Suppose that $\gamma < 1$ and $\mathcal{S} \times \mathcal{A}$ is finite. Then there exists a deterministic optimal policy π^*.

We finally state two last results regarding the best Q-value function Q_* (The proofs can be found in appendix A)

Proposition 2.6. Let $\gamma \in [0.1[$. Suppose that $\mathcal{A}$ is finite. Then the function $\psi : L^\infty(\mathcal{S} \times \mathcal{A}) \to L^\infty(\mathcal{S} \times \mathcal{A})$ ($L^\infty(\mathcal{S} \times \mathcal{A})$ denotes the space of bounded and measurable functions from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$) defined by the expression

$$\forall (Q, x, a) \in L^\infty(\mathcal{S}\times\mathcal{A})\times\mathcal{S}\times\mathcal{A}, \psi(Q)(x, a) := R(x, a) + \gamma \int_{x' \in \mathcal{S}} \max_{a' \in \mathcal{A}} Q(x', a') p(dx'|x, a)$$

Has unique fixed point Q_∞. Q_∞ is said to satisfy the **Bellman optimality equation (BOE)**

Proposition 2.7. Suppose firstly that $\gamma < 1$, $\mathcal{A}$ is finite. Suppose secondly that there exists an optimal policy π^* such that $Q_{\pi^*} = Q_*$ is bounded. Then Q_* is the unique element in $L^\infty(\mathcal{S}\times\mathcal{A})$ verifying BOE in proposition 2.6 which can be written as ($\mathbb{E}$ does not depend on any policy):

$$Q_*(x, a) = R(x, a) + \gamma \mathbb{E}\left[\max_{a' \in \mathcal{A}} Q_*(X_1, a') \middle| X_0 = x, A_0 = a\right] \tag{2.5}$$

2.3 Generalised policy iteration

In the rest of the section, we assume now that $\gamma < 1$, for any policy π, V_π is bounded on $\mathcal{S}$ (or the MDP has a finite horizon) and $\mathcal{A}$ is finite. These assumptions are made to satisfy the above theoretical results. Most of RL algorithms are based on a general principle called *Generalised policy iteration*. It consists of two tasks:

- *Policy evaluation*: starting from the last policy π updated in policy improvement, the Q-value function Q_π is estimated for every state and action.
- *Policy improvement*: We improve π by setting its greedy policy: for all $x \in \mathcal{S}$, $\pi(x) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} Q_\pi(x, a)$

These two steps are repeated until convergence (*i.e* an optimal policy has been found). In the first policy evaluation, the policy is arbitrary. Thanks to proposition 2.3, this general algorithm constantly improves the policy. Proposition 2.4 shows that if the policy cannot be improved any more, we have found an optimal policy. This result is remarkable because in this case, we have not only reached a local maximum but a general maximum (contrary to some usual optimisation algorithms such as stochastic gradient descent). Finally, according to proposition 2.5, if $\mathcal{S}$ is finite, then the generalised policy iteration algorithm terminates because there is a finite number of deterministic policies and at least one of them is optimal. All the algorithms developed below use (sometimes implicitly) this principle to improve the policy. When $\mathcal{S}$ and $\mathcal{A}$ are finite, the general algorithm 1 is detailed below.

Algorithm 1 Generalised policy iteration

```
Require: $\mathcal{S} \times \mathcal{A}$ is finite
  Input: a policy $\pi$
  Output: a deterministic policy
  repeat
    for all $(x, a) \in \mathcal{S} \times \mathcal{A}$ do
      evaluate $Q_\pi(x, a)$
    end for
    for all $x \in \mathcal{S}$ do
      $\pi(x) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_\pi(x, a)$
    end for
  until $Q_\pi$ is no longer improved
```

2.4 Main algorithms if $\mathcal{S}$ and $\mathcal{A}$ are finite

In this sub-section, we assume that the state and action spaces are finite and not too large to limit computation complexity. We explain here some algorithms for the policy evaluation step, that is given a policy π, the algorithm should return the Q-value function Q_π for each state and action. The policy improvement step is always the same for these algorithms (apart from Q-learning) and can be found in algorithm 1.

Several algorithms are quickly developed below without going into details. The goal here is to have a general idea of the possible methods to find the best policy. For further details, see [SB18] where all these algorithms (and some other ones) are mentioned.

2.4.1 Dynamic programming

Further information on that algorithm can be found in [SB18, Chapter 4] and [MRT18, parts 17.3.4-17.4.2]. This first main algorithm requires the knowledge of the transition probabilities $p(x'|x,a)$ for every $(x',x,a) \in \mathcal{S} \times \mathcal{S} \times \mathcal{A}$. The algorithm is based on the Bellman equation (2.3) and on the proof of proposition 2.2 that can be found in appendix A. We remind below the Bellman equation in a vector form:

$$V_\pi = \hat{R} + \gamma P V_\pi$$

Where $\hat{R}$ and P are defined in equation (A.2). According to remark A.1, these two elements are respectively finite-dimensional vector and matrix. Note that the sum of P's columns is equal to 1 and all its coefficients are non-negative (P is said to be stochastic). According to proposition 2.2, this equation has a unique solution which is nothing but V_π. The problem here consists in finding V_π to deduce Q_π thanks to relation (2.4). Two methods then come up and both work:

- One can solve the linear system by inverting the matrix $I - \gamma P$ (I denotes the identity matrix)
- According to the proof of proposition 2.2 and fixed-point theorem A.1. One can build the following sequence $(V_n)_{n\in\mathbb{N}}$ by choosing V_0 arbitrarily and by setting the recursive relation $V_{n+1} := \Psi(V_n) = \hat{R} + \gamma P V_n$ (see equation (A.1) for a definition of Ψ). This sequence converges to V_π at speed γ^n.

In practice, the second method known as *policy iteration* is more efficient.

2.4.2 Monte-Carlo method

This method is detailed in [SB18, Chapter 5]. Contrary to the previous one, the Monte-Carlo method neither needs to know explicitly the transition probabilities $p(x'|x,a)$ nor $\gamma < 1$. This method is based on independent simulations of processes (X, A). Each process generated is called an *episode*. Even if the MDP model does not need to be explicitly known, it must be able to generate easily and quickly episodes to be efficient.

We present here an *off-line* method. This means that the Q-value function is updated as soon as the episode is finished. It is possible to build an *on-line* method for Monte-Carlo. In this case, the Q-value function is updated as soon as a new state appear.

The Monte-Carlo method is as follows:

- Choose a time horizon $N \in \mathbb{N}$ (such that $\sum_{n=N+1}^{+\infty} \gamma^n R(X_n, A_n)$ is small enough)
- For each $(x,a) \in \mathcal{S} \times \mathcal{A}$, generate independently M episodes $(X^{(j),x,a}, A^{(j),x,a})$ until time N under policy π and starting with $X_0 = x$ and $A_0 = a$.
- Set for each $(x,a) \in \mathcal{S} \times \mathcal{A}$,

$$Q_M(x,a) = \frac{1}{M} \sum_{j=1}^{M} \Big(R(x,a) + \gamma R(X_1^{(j),x,a}, A_1^{(j),x,a}) + ... + \gamma^N R(X_N^{(j),x,a}, A_N^{(j),x,a}) \Big)$$

According to law of large numbers, Q_M approximates correctly Q_π if M and N are large enough.

2.4.3 Temporal-difference learning: SARSA algorithm

The last two algorithms are detailed in [SB18, Chapter 6] (particularly in [SB18, part 6.4-6.5]). This algorithm is also mentioned in [MRT18, part 17.5.4]) Monte-Carlo method can only update a Q-value function for only one state and action per episode. This can be seen as a drawback of the method because the frequency of updates is not high enough. To fix that problem, the temporal-difference learning was introduced. As well as Monte-Carlo, this method needs to generate easily and quickly episodes, but episodes do not need to be generated from all states if all states are sufficiently explored by the episodes.

One of the possible temporal difference learning algorithms is called SARSA (standing for state-action-reward-state-action). This on-line algorithm is organised as follows:

- Choose a time horizon $N \in \mathbb{N}$ (such that $\sum_{n=N+1}^{+\infty} \gamma^n R(X_n, A_n)$ is small enough) and initialise Q arbitrarily.
- Generate independently M episodes (X, A) until time N under policy π (the episodes should explore sufficiently all the states).
- For $\alpha \in [0,1]$ (the learning rate) and given an episode (X, A), set at each episode step n,

$$Q(X_n, A_n) \leftarrow Q(X_n, A_n) + \alpha \left(R(X_n, A_n) + \gamma Q(X_{n+1}, A_{n+1}) - Q(X_n, A_n) \right) \tag{2.6}$$

According to [SB18, part 6.4] and [MRT18, part 17.5.4], this algorithm converges almost surely to the optimal policy if we use the generalised policy iteration algorithm and if all states and actions are sufficiently explored (*i.e* infinitely explored if $M \to \infty$).

2.4.4 Q-learning

The algorithm is detailed in [MRT18, part 17.5.3]. This last method is another version of temporal difference learning but does not evaluate Q_π. This algorithm is generally *off-policy*, that is it evaluates and improves a policy (here the greedy policy) different from that used to generate the data. On the contrary, the two previous algorithms are *on-policy*: they evaluate and improve the policy that generates the data. Q-learning converges directly to Q_* and then does not use explicitly (but only implicitly) the generalised policy iteration. This algorithm can also be viewed as a method to estimate the solution of Bellman optimality equation (2.5). Similarly as SARSA, the algorithm is exactly the same apart from the update (2.6) which becomes:

$$Q(X_n, A_n) \leftarrow Q(X_n, A_n) + \alpha \left(R(X_n, A_n) + \gamma \max_{a \in \mathcal{A}} Q(X_{n+1}, a) - Q(X_n, A_n) \right) \quad (2.7)$$

According to [SB18, part 6.5] and especially [MRT18, part 17.5.3] (there is a complete proof), if all states and actions are sufficiently explored ($i.e$ infinitely explored if $M \to \infty$) and if some usual stochastic conditions hold, this Q converges almost surely to Q_*. Even if this algorithm does not use explicitly generalised policy iteration, it constantly improves implicitly the policy by modifying directly the Q-value function.

2.5 Deep Q-learning

All the sub-section is based on [Mni+13] and [Sim18]. We explain here the Deep Q-learning (DQL) algorithm. This algorithm is similar to the Q-learning algorithm but has one advantage over all the previous algorithms developed: it can manage infinite (or very large) state sets. Before explaining how the algorithm works, we give some notions on neural networks, a central mathematical object in deep learning that is used in the DQL algorithm. Afterwards, we detail the algorithm.

2.5.1 Neural networks

A neural network is a function Q mapping an input X and an output Y and depending on weights parameters $w := (w_1, ..., w_d) \in \mathbb{R}^d$. We just deal with the simple version of the neural network called *feed-forward neural network*. Its general structure is presented in figure 1. A neural network consists of layers of neurons. The first layer is the *input layer*: the number of neurons must match with the dimension of the input X. Each neuron of that layer returns a component of X. The next layers are called *hidden layers*. for each neuron of that layer, its inputs are the outputs of each neuron of the previous layer (see figure 1.a). Each neuron returns an output

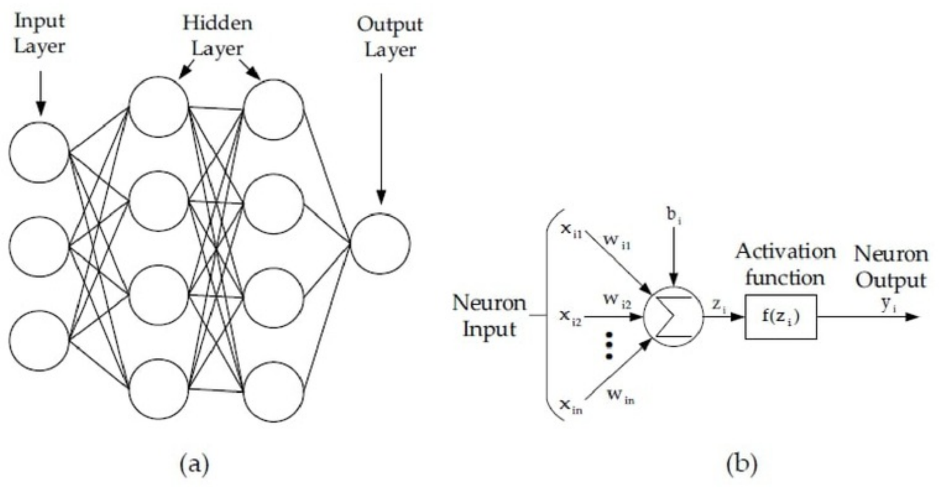


Figure 1: (a) Neural network scheme and (b) Structure of a neuron coming from a hidden layer or the output layer. Using the notations in this figure, the neuron i has n real inputs $x_{i,1}, ..., x_{i,n} \in \mathbb{R}$ and n weights $w_{i,1}, ..., w_{i,n} \in \mathbb{R}$ for each input plus a bias $b_i \in \mathbb{R}$. The output of the neuron is given by $y_i = f(z_i)$ where $f : \mathbb{R} \to \mathbb{R}$ is an activation function and $z_i = b_i + \sum_{j=1}^{n} w_{i,j} x_{i,j}$ is the weighted average of the inputs plus the bias. The figure comes from [Pha+19]

depending on weights and an activation function (see figure 1.b for further details). The last layer is called the *output layer*. This last layer works exactly like the hidden layers but its output is the exactly the output of the neural network. The number of neurons in the output layer must match with the dimension of the output Y

After choosing the activation functions, the goal is to find the best weights w (the weights include the biases) so that Q has the best estimation of the target Y given X. To do so, we have to define a *loss function* $\mathcal{L}$ to evaluate the quality of the estimation in this sense: the lower the loss function, the better the estimation. Assuming that $Y \in \mathbb{R}^m$ lies in an infinite (or very large) set, we do a regression (and not a classification) of Y with respect to X. In this case, a common loss function is the *mean squared error* defined by:

$$\mathcal{L}(X, Y; w) := \frac{1}{2m} \sum_{i=1}^{m} (Y_i - Q_i(X; w))^2$$

Where Q_i and Y_i are respectively the i^{th} component of Q and Y.

To make this loss function decrease with respect to the weights w, we use the stochastic gradient descent method. This method consists in selecting randomly (under a uniform distribution) in the data k instances of X and Y denoted respectively by $(x^{(1)}, y^{(1)}), ..., (x^{(k)}, y^{(k)})$. This set of data is called a *(mini) batch* of size k. Given a learning rate α we then update the weights w by the following gradient descent rule:

$$w \leftarrow w - \alpha \sum_{j=1}^{k} \nabla_w \mathcal{L}(x^{(j)}, y^{(j)}; w) = w + \alpha \sum_{j=1}^{k} \left(\frac{1}{m} \sum_{i=1}^{m} \left(y_i^{(j)} - Q_i(x^{(j)}; w) \right) \nabla_w Q_i(x^{(j)}; w) \right)$$

To do this step, we need to know $\nabla_w Q(x^{(j)}; w)$, which is equivalent to knowing the derivatives of the activation functions. This step is done using *back-propagation.* We will not explain this method but a theoretical explanation can be found in [Wer82], one of the first articles linking back-propagation to neural networks. In general, if the instances of the data are independent and identically distributed, the neural network is better trained.

As we said before, we need to choose the activation functions such that their derivatives are known. In addition, a neural network consisting of those functions should be able to approximate any function. A common activation function satisfying these two constrains (the second condition is satisfied according to [Lu+17]) is the rectified linear unit (ReLU) function. Its expression and its derivative expression are respectively given by:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad \text{and } \text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Hence, we will create a neural network where each neuron of the hidden layers will have the ReLU function as an activation function. The output layer will have the identity (or linear) function as activation function.

2.5.2 Principle of the algorithm

We now go back to the DQL algorithm. It is similar to Q-learning algorithm but instead of using a table to store the Q-values of each state-action couple, the Q-value is modelled as a neural network.

In our case and according to [Mni+13], the neural network Q is modelled as the Q-value function. Its input is the state $x \in \mathcal{S}$ (dimension $\dim(\mathcal{S})$) and its outputs are for each action $a \in \mathcal{A}$ the Q-value function $Q(x, a; w)$ (dimension $|\mathcal{A}|$). This neural network has to estimate the optimal Q-value function Q_*. In addition to generalised policy iteration point of view, this problem can also be seen as solving Bellman optimality equation (2.5) according to propositions 2.6 and 2.7, . This is why the right-hand side of this equation ($R(X_n, A_n) + \gamma \max_{a \in \mathcal{A}} Q(X_{n+1}, a; w)$) can be viewed as the target Y of the neural network.

The principle of the DQL algorithm is close to Q-learning but has some differences for the following reasons:

- We generate different episodes of (X, A) (with an off-policy method detailed below)
- At each time n of each episode, the Q-value is updated (like Q-learning) by performing a stochastic gradient descent step (with learning rate α) where the error is between the estimation $Q(X_n, A_n; w)$ and the target $R(X_n, A_n) + \gamma \max_{a \in \mathcal{A}} Q(X_{n+1}, a; w)$ (different from equation (2.7) used by Q-learning)

However, the target *a priori* depends on the neural network Q itself, which is not really convenient for learning. To allow the neural network Q to better learn the target, we use the *double Q-learning* method developed in [HGS15]. It consists in "freezing" the target in the following sense: we create a second neural network

$Q^{(target)}$. This neural network has exactly the same architecture as Q but has different weights. These weights are fixed for τ_{max} steps and are updated at every τ_{max} step by setting $Q^{(target)} \leftarrow Q$. To define the target value for the neural network properly, $\max_{a \in \mathcal{A}} Q(X_{n+1}, a, w)$ is replaced by $Q^{(target)}\left(X_{n+1}, \operatorname{argmax}_{a \in \mathcal{A}} Q(X_{n+1}, a, w); w'\right)$ (w' are the $Q^{(target)}$'s weights). In this case, according to [HGS15], the action selection is decoupled from action evaluation, which prevents from overestimating the Q-value function.

In addition, another technique called *experience replay* is used to improve the learning. It consists in storing several previous steps ($i.e$ a tuple $(X_n, A_n, R(A_n, X_n), X_{n+1})$) in a (finite) memory set $\mathcal{D}$ and selecting a mini-batch in $\mathcal{D}$ at each step to improve the target estimation by the neural network Q.

Like Q-learning, DQL can be an off-policy algorithm (the underlying policy that is improved is the greedy policy). But how could we choose the policy to generate the data? [Mni+13] answers this question by taking an *ε-greedy policy*, that is a policy that chooses the best action with respect to the neural network Q with probability $1 - \varepsilon$ or choose randomly another action with probability ε. If ε is close to 1, the algorithm explores as many couples (state,action) as possible. In contrast, if ε is close to 0, the policy is almost a greedy policy and then exploits the data to improve as much as possible the Q-value function. To manage this balance exploration-exploitation, one generally chooses ε close to 1 at the beginning of training to generate sufficiently diverse data to make the learning easier, and ε close to 0 at the end of training to find the optimal Q-value function more quickly. The value of ε can for example decrease exponentially with respect to training step, yielding a sequence $(\varepsilon_n)_{n \in \mathbb{N}}$ defined by the relation:

$$\varepsilon_n := (\varepsilon_{max} - \varepsilon_{min}) e^{-n\Delta\varepsilon} + \varepsilon_{min}$$

Where ε_{max} denotes the maximum exploration probability, ε_{min} denotes the minimum exploration probability, and $\Delta\varepsilon$ represents the exploration decay coefficient.

The behaviour of $(\varepsilon_n)_{n\in\mathbb{N}}$ is showed in algorithm 2.

The DQL algorithm detailed in algorithm 2 has several advantages:

- The use of a neural network instead of a table allows to apply a Reinforcement Learning algorithm with a large or even infinite state space $\mathcal{S}$
- Since the algorithm is off-policy and an ε-greedy policy is used, all possible states and actions can be explored more easily. Moreover, the distribution of steps in $\mathcal{D}$ is not too "biased": not every element in the data points to the same direction. In particular, if this direction was wrong, it would prevent from a convergence to the optimal Q-value function.
- Consecutive steps are correlated and this slows down the learning. The introduction of $\mathcal{D}$ breaks this correlation and then reduces the variance, improving the learning.
- Each step can be used several times and then can improve more weights.

Remark 2.3. According to [Sch+16] and [Bri+19], the Q-learning algorithm can be improved using prioritised (sequence) experience replay. The idea consists in selecting more often relevant data stored in experience replay memory (prioritised experience replay) and updating more data priority coefficients (prioritised sequence experience replay). Unfortunately, [Sch+16] explains that this method introduced a bias, which make our method developed in part 4 inefficient.

Algorithm 2 Deep Q-learning

```
Initialise $Q$ with random weights, set $\tau \leftarrow 0$, $\varepsilon \leftarrow \varepsilon_{max}$ and $Q^{(target)}(.,.,w') \leftarrow Q(.,.,w)$
for $i = 1$ to $M$ do
  Initialise independently $X_0^{(i)}$
  for $n = 1$ to $N$ do
    With probability $1-\varepsilon$, select $A_n^{(i)} \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(X_n^{(i)}, a, \theta)$
    Otherwise select another action $A_n^{(i)}$ randomly
    Evaluate $X_{n+1}^{(i)}$ and $R_n^{(i)} \leftarrow R(X_n^{(i)}, A_n^{(i)})$
    Store step $(X_n^{(i)}, A_n^{(i)}, R_n^{(i)}, X_{n+1}^{(i)})$ in $\mathcal{D}$ (delete the oldest step if $\mathcal{D}$ is full )
    Sample a $k$-sized random minibatch of steps $(x_1, a_1, r_1, x_1'), ..., (x_k, a_k, r_k, x_k')$ from $\mathcal{D}$
    $\Delta \leftarrow 0$ and $\tau \leftarrow \tau + 1$, $\varepsilon \leftarrow e^{-\Delta\varepsilon}(\varepsilon - \varepsilon_{min}) + \varepsilon_{min}$
    for $j = 1$ to $k$ do
      Set $y_j \leftarrow r_j + \gamma \max_{a' \in \mathcal{A}} Q^{(target)}(x_j', a', w')$
      $\Delta \leftarrow \Delta + (y_j - Q(x_j, a_j, w))\nabla_w Q(x_j, a_j, w)$
    end for
    $w \leftarrow w + \alpha \cdot \Delta$ (gradient descent step)
    if $\tau > \tau_{max}$ then
      Update the target neural network $Q^{(target)}(.,.,w') \leftarrow Q(.,.,w)$
      $\tau \leftarrow 0$
    end if
  end for
end for
```

3 Mathematical Finance

In this section, we introduce two continuous financial models. The first one is the famous Black-Scholes (BS) model. We will use this model for our RL problem because it is both tractable and (as we will see) closely linked to the second model: the uncertain volatility model (UVM). This model is like the BS model except that the volatility process is unknown. We just know that this process is bounded and the bounds are assumed to be known. The goal of this thesis is to use Reinforcement Learning to approximate the UVM highest and lowest prices (with respect to volatility processes).

3.1 The Black-Scholes model

In this sub-section, we remind the BS model and two useful results under this model: an explicit expression of the stock price S^{σ_c} and its partial differential equation for pricing. This very tractable model will allow us to build our RL problem. Let $\mathbb{Q}$ be the risk-neutral/pricing measure.

3.1.1 Definition and explicit expression of S^{σ_c}

Definition 3.1. Let $\sigma_c > 0$ be a volatility constant and $T, r > 0$. Let W be a $\mathbb{Q}$-brownian motion. The couple $(S^{\sigma_c}, B) := (S_t^{\sigma_c}, B_t)_{0 \leq t \leq T}$ follows the **Black-Scholes model** if and only if B (the risk-free bond) satisfies the differential equation:

$$dB_t = rB_t dt$$

and S^{σ_c} satisfies the stochastic differential equation

$$dS_t^{\sigma_c} = S_t^{\sigma_c}(rdt + \sigma_c dW_t) \tag{3.1}$$

On the interval $[0, T]$ where $S_0 > 0$ is deterministic.

The BS model is very tractable because of the following result (the existence can be proved using Itô's formula).

Theorem 3.1. With notations in the above definition 3.1 and given the constants $S_0, \sigma_c, r > 0$, there exists a unique solution (S^{σ_c}, B) to the BS model. The solutions are given by:

$$\forall t \in [0,T], \begin{cases} B_t &= B_0 e^{rt} \\ S_t^{\sigma_c} &= S_0 e^{\sigma_c W_t + \left(r - \frac{\sigma_c^2}{2}\right)t} \end{cases}$$

Remark 3.1. Let $0 \leq t' < t \leq T$. $S_t^{\sigma_c}$ can be expressed in terms of $S_{t'}^{\sigma_c}$ and an independent quantity. More precisely:

$$S_t^{\sigma_c} = S_{t'}^{\sigma_c} e^{\sigma_c (W_t - W_{t'}) + \left(r - \frac{\sigma_c^2}{2}\right)(t-t')}$$

3.1.2 The Black-Scholes partial differential equation

We now state the following theorem giving a link between the price of an option and a partial differential equation.

Theorem 3.2. Let $C_{BS}(t, S_t^{\sigma_c}) = e^{-r(T-t)} \mathbb{E}^{\mathbb{Q}}\left[F(S_T^{\sigma_c}) | S_t^{\sigma_c}\right]$ be the BS price at time $t \in [0,T]$ of an European derivative whose payoff is given by $F(S_T^{\sigma_c})$ (this payoff is assumed to be integrable under the risk-neutral/pricing measure $\mathbb{Q}$). If $C_{BS} \in \mathcal{C}^{1,2}([0,T[\times \mathbb{R}_+^*, \mathbb{R})$ then C_{BS} satisfies the BS partial differential equation on $[0,T[\times \mathbb{R}_+^*$:

$$\frac{\partial C_{BS}}{\partial t} + r\left(S \frac{\partial C_{BS}}{\partial S} - C_{BS}\right) + \frac{\sigma_c^2}{2} S^2 \frac{\partial^2 C_{BS}}{\partial S^2} = 0 \tag{3.2}$$

With terminal condition:

$$C_{BS}(T,S) = F(S)$$

3.2 The Uncertain volatility model

3.2.1 Definition of the model

We consider here a much more general model than the Black-Scholes model. This uncertain volatility model is here viewed as a stochastic control problem where the control is on volatility. For further details on stochastic control theory, see appendix B. Let $\mathbb{Q}$ be the pricing/risk-neutral measure. We now define what the UVM is and we comment on this definition afterwards.

Definition 3.2. Let $T > 0$, $0 < \sigma_{min} \leq \sigma_{max}$ and let W be a $\mathbb{Q}$-brownian motion. Define the following sets:

- The filtration $(\mathcal{F}_t)_{0\leq t\leq T}$ is given by $\mathcal{F}_t := \sigma(W_s, s \leq t)$ for every $t \in [0,T]$
- The set of possible volatility processes is $Vol := \mathcal{U}_0$ according to equation (B.1) in definition B.2 for $U = [\sigma_{min}, \sigma_{max}]$. More precisely, $Vol = \{\sigma := (\sigma_t)_{0\leq t\leq T}$ progressively measurable : $\forall t \in [0,T], \sigma_{min} \leq \sigma_t \leq \sigma_{max}\}$ where the progressive measurability is between the filtration $(\mathcal{B}([0,t]) \bigotimes \mathcal{F}_t)_{0\leq t\leq T}$ and the borel σ-field $\mathcal{B}([\sigma_{min}, \sigma_{max}])$

Given $\sigma \in Vol$ a bounded stochastic process, the couple $(S^\sigma, B) = (S^\sigma_t, B_t)_{0\leq t\leq T}$ follows the **uncertain volatility model (UVM)** if and only if B (the risk-free bond) satisfies the differential equation:

$$dB_t = rB_t dt$$

Where $r > 0$ and satisfies the following stochastic differential equation:

$$dS^\sigma_t = S^\sigma_t (rdt + \sigma_t dW_t) \tag{3.3}$$

On the interval $[0,T]$ where $S_0 > 0$ is deterministic.

Regarding B, exactly like the BS model for bonds, the differential equation can be solved immediately and yields:

$$\forall t \in [0,T], B_t = B_0 e^{rt}$$

Regarding S, this model is a particular case of the stochastic control equation (B.2) we develop in appendix B:

$$\begin{cases} U & := & [\sigma_{min}, \sigma_{max}] & \\ b(t,S,u) & := & rS & \text{where } r > 0 \\ \varsigma(t,S,u) & := & uS & \end{cases}$$

The boundedness condition for σ ensure the existence and the uniqueness of S^σ given a deterministic initial condition $S_0 > 0$ according to Theorem B.1. In addition, the

BS model is a particular case of UVM where $\sigma_c = \sigma_{min} = \sigma_{max}$ is constant and deterministic. Hence theorem 3.1 is also proved.

The interval $[\sigma_{min}, \sigma_{max}]$ can be interpreted as a confidence interval on volatility (95% or 99% for example). Furthermore, all the BS implied volatilities quoted in the market must be within this interval to avoid arbitrage opportunities, this gives another interpretation to the couple $(\sigma_{min}, \sigma_{max})$.

Contrary to the BS model, the UVM is not very tractable because there is no explicit solution for the stochastic differential equation (3.3) satisfied by S^σ. However, if we know how to price under this model, we get a much better price than the BS price. This is due to the fact that the BS volatility is necessary to be known in order to find a price of an option but this volatility parameter is unknown in practice. On the other hand, the UVM price requires only bounds on volatility.

The goal of this thesis is to approximate the price of a derivative whose payoff is given by $F(S_T)$ in the best (or worst) case by using Reinforcement Learning that is, to approximate:

$$\begin{aligned} & C^+(t,S) := \sup_{\sigma \in Vol} \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}F(S_T^\sigma)|S_t^\sigma = S\right] \\ \text{or} \quad & C^-(t,S) := \inf_{\sigma \in Vol} \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}F(S_T^\sigma)|S_t^\sigma = S\right] \end{aligned} \tag{3.4}$$

Where $\mathbb{E}^{\mathbb{Q}}$ is the expectation under the pricing/risk-neutral measure $\mathbb{Q}$

3.2.2 A result on extreme prices

We now give a useful result that can be found in [ALP95, equations (7),(8) and (9)] on the two extreme prices C^+ and C^- defined above in equation (3.4). A proof of this result can be found in appendix B

Theorem 3.3. If the payoff function F verifies the quadratic growth condition:

$$\exists K > 0, \forall S \in \mathbb{R}_+^*, |F(S)| < K(1 + S^2)$$

and if the functions C^+ and C^- defined in equation (3.4) belong to $\mathcal{C}^{1,2}([0,T[\times\mathbb{R}_+^*,\mathbb{R})$, then they satisfy the Black-Scholes-Barenbatt (BSB) equation:

$$\frac{\partial C^{\pm}}{\partial t} + r\left(S\frac{\partial C^{\pm}}{\partial S} - C^{\pm}\right) + \frac{1}{2}\left(\Sigma^{\pm}\left(\frac{\partial^2 C^{\pm}}{\partial S^2}\right)\right)^2 S^2\frac{\partial^2 C^{\pm}}{\partial S^2} = 0 \tag{3.5}$$

On $[0,T[\times\mathbb{R}_+^*$, where

$$\forall x\in\mathbb{R}, \Sigma^+(x) := \begin{cases} \sigma_{max} & \text{if } x\geq 0 \\ \sigma_{min} & \text{if } x<0 \end{cases} \text{ and } \Sigma^-(x) := \begin{cases} \sigma_{min} & \text{if } x\geq 0 \\ \sigma_{max} & \text{if } x<0 \end{cases} \tag{3.6}$$

and with terminal condition:

$$C^{\pm}(T,S) = F(S)$$

Note that $C^{\pm}$ denotes either C^+ or C^-, but if we deal with C^+, we necessarily refer to Σ^+ in equation (3.5) and conversely, if we deal with C^-, we necessarily refer to Σ^- in this same equation.

This result is extremely close to the famous BS partial differential equation. The only difference lies in the function Σ which varies with respect to the sign of the gamma (*i.e.* the second derivative of $C^{\pm}$ with respect to S) in the BSB equation. Note that the BS partial differential equation (theorem 3.2) is proved because it is a particular case of the above result. Thanks to these remarks, we can interpret the problem as follows (a more precise scheme is given part 3.2.4):

- To find the highest (or the lowest) price of the derivative, we firstly compute the gamma (or at least its sign).
- Then, knowing the gamma sign on a small interval, we choose either σ_{min} or σ_{max} according to the equation (3.6).
- In this case, the stock price follows the BS model on this small interval where the constant volatility is the volatility we chose in the previous step.

The most useful part of that result is the binary behaviour of the volatility parameter σ to determine the extreme prices. We will use that corollary to construct our RL problem in section 4. We will also use this result to explain why our RL problem finds asymptotically the extreme prices.

3.2.3 Properties of BSB equation

We now state two interesting properties on BSB equation (3.5) with boundary condition $C^{\pm}(S,T) = F(S)$. These two results come from [LLX19].

The first result shows that the BSB equation has a unique "weak" solution called a viscosity solution. We give here a definition of that concept that can also be found in [LLX19, Definition 2.1].

Definition 3.3. Let X be a topological space. A function $u : X \to \mathbb{R}$ is said to be **upper (resp. lower) semi-continuous** if and only if for every $a \in \mathbb{R}$, $\{x \in X : u(x) \geq a\}$ (resp. $\{x \in X : u(x) \leq a\}$) is a closed set.

Definition 3.4. A **viscosity sub-solution (resp. sup-solution)** of (3.5) with boundary condition $C^{\pm}(S,T) = F(S)$ on $]0,T[\times\mathbb{R}$ is an upper (resp. lower) semi-continuous function $u :]0,T[\times\mathbb{R} \to \mathbb{R}$ such that, for all $(t,x) \in]0,T[\times\mathbb{R}$, $\phi \in \mathcal{C}^{1,2}(]0,T[\times\mathbb{R})$ such that $u(t,x) = \phi(t,x)$ and $u < \phi$ (resp. $u > \phi$) on $]0,T[\times\mathbb{R}\backslash(t,x)$, we have:

$$\frac{\partial \phi}{\partial t} + r\left(S\frac{\partial \phi}{\partial S} - \phi\right) + \frac{1}{2}\left(\Sigma^{\pm}\left(\frac{\partial^2 \phi}{\partial S^2}\right)\right)^2 S^2 \frac{\partial^2 \phi}{\partial S^2} \geq 0 \text{ (resp. } \leq 0)$$

A **viscosity solution** of this equation is both a sup-solution and a sub-solution

We can then state the first result (see [LLX19, Theorem 2.2]):

Theorem 3.4. If there exists $K > 0$ and $m \in \mathbb{N}$ (both depending on F) such that:

$$\forall S_1, S_2 \in \mathbb{R}, |F(S_1) - F(S_2)| \leq K(1 + |S_1|^m + |S_2|^m)|S_1 - S_2|$$

Then BSB equation (3.5) with boundary condition $C^{\pm}(S,T) = F(S)$ has a unique viscosity solution on $]0,T[\times\mathbb{R}_+^*$.

The second result states that if the payoff function F is either convex or concave (this is the case if for example the option is a Call option or a Put option), then the BSB solution is also a BS solution. More precisely (see [LLX19, Proposition 2.4, equation (2.2)] for a proof)

Proposition 3.1. Let $C^{\pm}$ be the (viscosity) solution of BSB equation (3.5) with boundary condition $C^{\pm}(S,T) = F(S)$ (F satisfies the condition of Theorem 3.4). We then have:

$$\begin{cases} F \text{ convex} & \iff \quad C^+ = C_{BS}^{\sigma_{max}} \quad \text{and} \quad C^- = C_{BS}^{\sigma_{min}} \\ F \text{ concave} & \iff \quad C^+ = C_{BS}^{\sigma_{min}} \quad \text{and} \quad C^- = C_{BS}^{\sigma_{max}} \end{cases}$$

Where $C_{BS}^{\sigma_c}$ is the solution of the BS partial differential equation (3.2) and $\sigma_c > 0$ is the volatility parameter.

Even if the result has not been proved for C^- in [LLX19] it suffices to consider $D := -C^-$ and to see that for any $x \in \mathbb{R}$, $\Sigma^+(-x) = \Sigma^-(x)$. This quantity verifies the same BSB equation as C^+ with terminal condition $-F$. This yields immediately the expected result. This useful result will help us test if our UVM price programming returns the expected price because BS prices of common options with convex or concave payoffs (*e.g* a Call option, a Put option) are well-known analytically.

3.2.4 Discretisation of BSB equation

According to the article [ALP95], the BSB equation (3.5) can be solved using a (recombining) trinomial tree. We present here this method.

Let $N \in \mathbb{N}^*$, $\Delta t := \frac{T}{N}$ (it can be viewed as a trading period) and let $p := (p_n)_{0 \le n \le N-1}$ be a sequence of probability parameters lying in $\left[\frac{\sigma_{min}^2}{2\sigma_{max}^2}, \frac{1}{2}\right]$. We also set:

$$\begin{cases} U := & e^{\sigma_{max}\sqrt{\Delta t} + r\Delta t} & \text{and probability} & P_U(q) := & q\left(1 - \frac{\sigma_{max}\sqrt{\Delta t}}{2}\right) \\ M := & e^{r\Delta t} & \text{and probability} & P_M(q) := & 1 - 2q \\ D := & e^{-\sigma_{max}\sqrt{\Delta t} + r\Delta t} & \text{and probability} & P_D(q) := & q\left(1 + \frac{\sigma_{max}\sqrt{\Delta t}}{2}\right) \end{cases}$$

Where $q \in \left[\frac{\sigma_{min}^2}{2\sigma_{max}^2}, \frac{1}{2}\right]$ is a probability parameter. We can notice that $UD = M^2$ to ensure that the we have a recombining tree. The stock price $S := (S_n)_{0 \le n \le N}$ is then a Markov process driven by the following discrete dynamics:

$$\forall n \in [\![0, N-1]\!], S_{n+1} := \begin{cases} US_n & \text{with probability} \quad P_U(p_n) \\ MS_n & \text{with probability} \quad P_M(p_n) \\ DS_n & \text{with probability} \quad P_D(p_n) \end{cases}$$

Given $S_0 > 0$ fixed, we can immediately show that the possible values of S at time $n \in [\![0, N]\!]$ are:

$$S_n^{(j)} := S_0 e^{j\sigma_{max}\sqrt{\Delta t} + nr\Delta t}$$

Where $-n \leq j \leq n$ (note the difference in notations between the random variable S_n and the possible value $S_n^{(j)}$). We can then set the (discrete) extreme prices (with respect to the probability parameter sequence p):

$$\begin{aligned} C_n^{(j),+} &:= e^{-(N-n)r\Delta t} \sup_p \mathbb{E}\left[F(S_N)|S_n = S_n^{(j)}\right] \\ C_n^{(j),-} &:= e^{-(N-n)r\Delta t} \inf_p \mathbb{E}\left[F(S_N)|S_n = S_n^{(j)}\right] \end{aligned}$$

Where $0 \leq n \leq N$ and $-n \leq j \leq n$. They verify the terminal condition:

$$\forall j \in [\![-N, N]\!], C_N^{(j),\pm} = F(S_N^{(j)})$$

And the recursive relations ($n \leq N-1$ here):

$$\begin{aligned} C_n^{(j),+} &= e^{-r\Delta t} \sup_{p_n} \left(P_U(p_n)C_{n+1}^{(j+1),+} + P_M(p_n)C_{n+1}^{(j),+} + P_D(p_n)C_{n+1}^{(j-1),+}\right) \\ C_n^{(j),-} &= e^{-r\Delta t} \inf_{p_n} \left(P_U(p_n)C_{n+1}^{(j+1),-} + P_M(p_n)C_{n+1}^{(j),-} + P_D(p_n)C_{n+1}^{(j-1),-}\right) \end{aligned}$$

A quick calculus yields for $C_n^{(j),+}$:

$$C_n^{(j),+} = e^{-r\Delta t} \begin{cases} C_{n+1}^{(j),+} + \frac{1}{2}L_{n+1}^{(j),+} & \text{if } L_{n+1}^{(j),+} \geq 0 \\ C_{n+1}^{(j),+} + \frac{\sigma_{min}^2}{2\sigma_{max}^2}L_{n+1}^{(j),+} & \text{if } L_{n+1}^{(j),+} < 0 \end{cases}$$

Similarly we have for $C_n^{(j),-}$:

$$C_n^{(j),-} = e^{-r\Delta t} \begin{cases} C_{n+1}^{(j),-} + \frac{1}{2}L_{n+1}^{(j),-} & \text{if } L_{n+1}^{(j),-} < 0 \\ C_{n+1}^{(j),-} + \frac{\sigma_{min}^2}{2\sigma_{max}^2}L_{n+1}^{(j),-} & \text{if } L_{n+1}^{(j),-} \geq 0 \end{cases}$$

where $L_{n+1}^{(j),\pm} := C_{n+1}^{(j+1),\pm} - 2C_{n+1}^{(j),\pm} + C_{n+1}^{(j-1),\pm} + \frac{\sigma_{max}\sqrt{\Delta t}}{2}(C_{n+1}^{(j-1),\pm} - C_{n+1}^{(j+1),\pm})$. According to [ALP95], this last quantity can be interpreted as a discretisation of the second derivative operator times the stock price times the volatility parameter σ_{max}. In [Par95] it is showed that the numerical scheme we built converges to the solution of the BSB equation (3.5).

4 The RL problem for derivative pricing

In this section we transform a derivative pricing problem into a RL problem. Our goal consists in approximating the extreme UVM prices using Reinforcement Learning. To do so, we consider a piece-wise BS process as an MDP.

4.1 Construction of the RL problem

4.1.1 The Markov Decision Process

First of all, we introduce the problem with some notations. Let $N \in \mathbb{N}^*$ be the discrete time horizon. Throughout this part, we consider an European derivative with maturity $T > 0$ and payoff $F(S_T)$ where S_T is the the stock price at time T and $F : \mathbb{R}_+^* \to \mathbb{R}$ is a measurable function such that $F(S_T)$ is integrable. Given a initial time $t_0 \geq 0$, we also introduce the following notations:

$$\tau := T - t_0, \ ,\forall n \in \mathbb{N}, t_n := t_0 + \frac{n\tau}{N}, \ \Delta t := \frac{\tau}{N} = t_{n+1} - t_n$$

In our problem, we have to find the extreme prices under the UVM with respect to volatility (these prices are defined in equation (3.4)). This is why at first sight, the MDP should be the stock price process with a control on volatility. However, according to theorem 3.3, volatility may depend on time. This is why as remark 2.1 suggests, the MDP is the couple (stock price, time). To define this MDP properly, we have to define the action and the state spaces $\mathcal{S}$ and $\mathcal{A}$. Since this process is the couple (stock price, time), we define naturally according to remark 2.1

$$\mathcal{S} := \mathbb{R}_+^* \times \mathbb{N}$$

Regarding $\mathcal{A}$, we know from theorem 3.3 that the UVM extreme prices are only determined by the volatility bounds σ_{min} and σ_{max}. Hence, the action space admits exactly two elements *i.e*

$$\mathcal{A} := \{\sigma_{min}, \sigma_{max}\}$$

Let us now define the MDP. Let W be a brownian motion under the pricing/risk-neutral measure $\mathbb{Q}$. We introduce the notation (for $n \in \mathbb{N}$):

$$\Delta W_n := W_{t_{n+1}} - W_{t_n}$$

Knowing that a brownian motion has independant and starionnary increments, the sequence $(\Delta W_n)_{n\in\mathbb{N}}$ is a sequence of independent and identically distributed random variables. We build the stock-price process $(S, \sigma) := (S_n, \sigma_n)_{n\in\mathbb{N}}$ by the recursive relation:

$$\forall n \in \mathbb{N}, S_{n+1} := S_n e^{\sigma_n \Delta W_n + \left(r - \frac{\sigma_n^2}{2}\right)\Delta t} \tag{4.1}$$

Assuming that S_0 is known and for any $n \in \mathbb{N}$, σ_n depends (potentially randomly) on time n and current state S_n. As we said before, according to BSB equation (3.5) volatility may depend on time. This last assumption for σ is then necessary to assure the convergence of the optimal discounted sum of rewards to the UVM price. Equation (4.1) shows that S_{n+1} depends on S_n σ_n and a quantity independent of $S_0, ..., S_{n-1}$. Hence, (S, σ) is almost a MDP in the sense of remark 2.1 (it depends on time). According to this remark, the process (X, σ) defined by the expression:

$$\forall n \in \mathbb{N}, X_n := (S_n, n)$$

is a MDP. Note here that we have implicitly assumed here that X starts at time t_0, which is not compulsory. We can start at another time if we wish.

Thanks to remark 3.1 and equation (4.1), it can be seen that for $n \in \mathbb{N}$, S is constructed similarly as a BS model between times t_n and t_{n+1}. This means that S follows a piece-wise BS model. That is, for each $n \in \mathbb{N}$, S is modelled with a constant volatility $\sigma_n \in \mathcal{A}$ on each time interval $[t_n, t_{n+1}[$.

4.1.2 Reward function

The definition of reward function is based on [Bue+19] and [Gra]. To define a reward function, we need to define γ. In Mathematical Finance, this last quantity can be seen as a discount factor. Then we can set $\gamma := e^{-r\Delta t}$. We assume that the interest rate r is strictly positive (which is true most of the time in Finance). Thus $\gamma < 1$. Furthermore, the discounted sum of rewards has to be equal to the expectation of the discounted payoff under pricing/risk-neutral measure:

$$\mathbb{E}_{\pi}^{\mathbb{Q}}\left[\sum_{k=0}^{+\infty} \gamma^k R(X_k, \sigma_k) \middle| X_0, n = 0\right] = \mathbb{E}_{\pi}^{\mathbb{Q}}\left[e^{-r\tau} F(S_N) | S_0\right]$$

Where π is a policy. We then define the reward function by the expression:

$$\forall x = (s, n) \in \mathcal{S}, \forall \sigma_c \in \mathcal{A}, R(x, \sigma_c) = \begin{cases} 0 & \text{if } n \neq N \\ F(s) & \text{if } n = N \end{cases}$$

Note that if $n > N$, the problem is not interesting any more since every reward is null (the derivative vanished). Thus, the MDP (X, σ) has a finite time horizon in the sense of definition 2.7. This setting is useful to simulate (X, σ): we will start systematically at time 0 with an arbitrary S_0 and we will end each simulation at time N.

Regarding the theoretical results (in part 2.2) that can be applied to this RL model, it can be seen that $\gamma < 1$ and (X, σ) has a finite horizon. Thus, principles of generalised policy iteration (propositions 2.3 and 2.4) hold. Unfortunately there is no proof that there exists an optimal policy ($\mathcal{S}$ is infinite), and unless F is bounded (this is the case for a Put option or a Digital/Binary option), the value function is not necessarily bounded. The Bellman equations are then not necessarily satisfied. Nevertheless, we assume that an optimal policy exists. Moreover, even if the value function is not bounded, it is "almost bounded" if F is continuous (this is the case for a Call option) because in practice, the S_0 considered lies in a line segment and the (discounted back) expectation of such a payoff is generally continuous with respect to S_0, and thus bounded. Hence we can admit that the Bellman equations are satisfied.

We can also notice that we use here a learning algorithm to estimate the price of the derivative. However, the main goal of Reinforcement Learning is to allow an agent (*e.g* a robot, a machine, a computer) to interact with a well-defined environment (*e.g* a financial market or a video game, see [Sim18] for examples of such video games), which is not what we do here. Our goal here is to estimate the highest price or in other words the highest expectation of a sum of rewards. This is exactly what a learning algorithm is supposed to do and this is why we use this method here.

4.2 From Reinforcement Learning to UVM Pricing

The goal of the RL model we described above is to approximate the UVM highest price C^+. In this sub-section we explain why if Δt is sufficiently small, this RL model approximates the BSB equation. This explanation is not perfectly rigorous (the issues will be mentioned) but aims at linking our model to the BSB equation, and thus to the UVM highest price. In particular we explain why the volatility choice is closely linked to the gamma, that is the second derivative of the price with respect to the stock price. Note that a similar result can be obtained for the UVM lowest price C^- using a symmetric argument like in theorem 3.3 and proposition 3.1.

4.2.1 Underlying numerical scheme

First of all, we built a mathematical operator and we show that the best possible value function defined above in part 4.1 verifies a numerical scheme linked to this operator. The different results are proved in appendix C. To do so, set the space of functions:

$$L^{Pol}(\mathbb{R}_+^*) := \left\{V : \mathbb{R}_+^* \to \mathbb{R} \text{ measurable} : \exists (K,m) \in \mathbb{R}_+^* \times \mathbb{N}, \forall s > 0, |V(s)| \leq K(1+s^m)\right\}$$

Where Pol stands for "Polynomial growth" here. Note that $L^\infty(\mathbb{R}_+^*) \subset L^{Pol}(\mathbb{R}_+^*)$. Define the operator $\Phi : L^{Pol}(\mathbb{R}_+^*) \to L^{Pol}(\mathbb{R}_+^*)$ by the expression:

$$\forall (V,s) \in L^{Pol}(\mathbb{R}_+^*) \times \mathbb{R}_+^*, \Phi(V)(s) := \max_{\sigma \in \mathcal{A}} \int_{-\infty}^{+\infty} e^{-r\Delta t} \phi(x) V\left(s e^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\Delta t\right)}\right) dx$$

Where ϕ denotes the probability density function of the normal distribution with mean 0 and standard deviation 1.

Proposition 4.1. The operator Φ is well-defined.

We now state two results that will be useful for the remaining of the sub-section. In the first one, we show that we can apply the operator to the highest UVM price:

Proposition 4.2. If $F \in L^{Pol}(\mathbb{R}_+^*)$ and the UVM price C^+ satisfies BSB equation (3.5), then for any $t \in [0, T]$, the highest UVM price at time t $C^+(\cdot, t)$ also belongs to $L^{Pol}(\mathbb{R}_+^*)$.

The second result is nothing but the numerical scheme satisfied by the best possible value function introduced in part 4.1:

Proposition 4.3. Assume that $F \in L^{Pol}(\mathbb{R}_+^*)$ and there exists an optimal policy π^* for the RL model built in part 4.1. Then the associated value functions $(V_n)_{0 \leq n \leq N}$ defined by the expressions:

$$\forall x = (s, n) \in \mathbb{R}_+^* \times [\![0, N]\!], V_n(s) := e^{-r(N-n)\Delta t} \mathbb{E}_{\pi^*}^{\mathbb{Q}} \left[F(S_N) | X_n = x \right] \tag{4.2}$$

is a sequence of $L^{Pol}(\mathbb{R}_+^*)$ that verifies the following numerical scheme:

$$\begin{cases} V_N & := F \\ V_n & := \Phi(V_{n+1}) \end{cases}$$

In practice, building explicitly such a numerical scheme can be very expensive in computation time (it can be exponential in N). The only way to reduce dramatically the computation time is to use a recombining tree that approximates the integrals, exactly like the numerical method developed in part 3.2.4.

4.2.2 Local error satisfied by BSB equation

After finding the numerical scheme satisfied by the best value function of our RL model, we show here that the highest UVM price C^+ (also known as the unique solution of the BSB equation (3.5) with volatility function Σ^+) satisfies the numerical scheme locally with an error of $o(\Delta t)$. Let $s \in \mathbb{R}_+^*$ and $x \in \mathbb{R}$ be fixed. We introduce the following notations:

$$\forall n \in [\![0, N]\!], \begin{cases} C_n^+ & := C^+(\cdot, t_n) \\ \frac{\partial C_n^+}{\partial t} & := \frac{\partial C^+}{\partial t}(\cdot, t_n) \\ \frac{\partial C_n^+}{\partial s} & := \frac{\partial C^+}{\partial s}(\cdot, t_n) \\ \frac{\partial^2 C_n^+}{\partial s^2} & := \frac{\partial^2 C^+}{\partial s^2}(\cdot, t_n) \end{cases}$$

We first assume that C^+ is smooth enough to satisfy BSB equation and to have the right to perform a Taylor expansion. Hence, it follows that if Δt is small enough,

$$\begin{aligned} e^{-r\Delta t} C_{n+1}^+ \left(s e^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)\Delta t} \right) & \underset{\Delta t \to 0}{=} C_{n+1}^+ \left(s + (s\sigma x)\sqrt{\Delta t} + \left(rs - \frac{\sigma^2}{2}s + \frac{\sigma^2}{2}sx^2 \right)\Delta t + o(\Delta t) \right) \\ & - r\Delta t C_{n+1}^+ \left(s + (s\sigma x)\sqrt{\Delta t} + \left(rs - \frac{\sigma^2}{2}s + \frac{\sigma^2}{2}s^2x^2 \right)\Delta t + o(\Delta t) \right) + o(\Delta t) \\ & \underset{\Delta t \to 0}{=} C_{n+1}^+(s) + \left(sx\sigma \frac{\partial C_{n+1}^+}{\partial s}(s) \right)\sqrt{\Delta t} \\ & + \left(r\left(s\frac{\partial C_{n+1}^+}{\partial s}(s) - C_{n+1}^+(s) \right) + \frac{\sigma^2}{2}s\left(\frac{\partial C_{n+1}^+}{\partial s}(s)(x^2-1) \right) + s^2x^2\frac{\partial^2 C_{n+1}^+}{\partial s^2}(s) \right)\Delta t + o(\Delta t) \end{aligned}$$

We then multiply the expression by $\phi(x)$ and we integrate against the variable x (x is then no longer fixed). We *assume here* that this integration does not change the $o(\Delta t)$. We use the linearity of multiplication and integration and the following results:

$$\int_{-\infty}^{+\infty} \phi(x)dx = 1, \quad \int_{-\infty}^{+\infty} x\phi(x)dx = 0 \text{ and } \int_{-\infty}^{+\infty} x^2\phi(x)dx = 1$$

The Taylor expansion thus becomes:

$$\begin{aligned} \int_{-\infty}^{+\infty} \phi(x) e^{-r\Delta t} C_{n+1}^+ \left(s e^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)\Delta t} \right) dx \underset{\Delta t \to 0}{=} & \; C_{n+1}^+(s) + \\ & \left(r\left(s\frac{\partial C_{n+1}^+}{\partial s}(s) - C_{n+1}^+(s) \right) + \frac{\sigma^2}{2}s^2\frac{\partial^2 C_{n+1}^+}{\partial s^2}(s) \right)\Delta t + o(\Delta t) \end{aligned}$$

Afterwards, we apply the maximum with respect to volatility and we *assume here* that the $o(\Delta t)$ is small enough so that this maximum is determined only by the terms of order 0 and 1. It turns out that:

$$\begin{aligned} \Phi(C_{n+1}^+)(s) & \underset{\Delta t \to 0}{=} C_{n+1}^+(s) + \left(r\left(s\frac{\partial C_{n+1}^+}{\partial s}(s) - C_{n+1}^+(s) \right) + \max_{\sigma \in \mathcal{A}} \left(\frac{\sigma^2}{2}s^2\frac{\partial^2 C_{n+1}^+}{\partial s^2}(s) \right) \right)\Delta t + o(\Delta t) \\ & \underset{\Delta t \to 0}{=} C_{n+1}^+(s) + \left(r\left(s\frac{\partial C_{n+1}^+}{\partial s}(s) - C_{n+1}^+(s) \right) + \frac{1}{2}\left(\Sigma^+\left(\frac{\partial^2 C_{n+1}^+}{\partial s^2}(s) \right) \right)^2 s^2\frac{\partial^2 C_{n+1}^+}{\partial s^2}(s) \right)\Delta t \\ & + o(\Delta t) \end{aligned}$$

We also perform a second Taylor expansion with respect of time:

$$C^+_{n+1}(s) - C^+_n(s) = \underset{\Delta t \to 0}{=} \frac{\partial C^+_{n+1}}{\partial t}(s)\Delta t + o(\Delta t)$$

Plugging this relation into the first expansion, we finally get:

$$\begin{aligned}\Phi(C^+_{n+1})(s) - C^+_n(s) &\underset{\Delta t \to 0}{=} \left(\frac{\partial C^+_{n+1}}{\partial t}(s) + r\left(s\frac{\partial C^+_{n+1}}{\partial s}(s) - C^+_{n+1}(s)\right)\right)\Delta t \\ &+ \left(\frac{1}{2}\left(\Sigma^+\left(\frac{\partial^2 C^+_{n+1}}{\partial s^2}(s)\right)\right)^2 s^2 \frac{\partial^2 C^+_{n+1}}{\partial s^2}(s)\right)\Delta t + o(\Delta t) \quad (4.3) \\ &\underset{\Delta t \to 0}{=} o(\Delta t) \text{ according to BSB equation (3.5)}\end{aligned}$$

Hence, we have showed that the BSB solution satisfies the numerical scheme locally with an error of $o(\Delta t)$.

4.2.3 Properties of the operator Φ

Before looking at the case where the above approximation is uniform, we state two properties of the operator Φ. these properties are proved in appendix C.

Proposition 4.4. For any $n \in \mathbb{N}^*$, $\Phi^n = \Phi \circ ... \circ \Phi$ (n times) is sub-additive that is:

$$\forall U, V \in L^{Pol}(\mathbb{R}^*_+), \Phi^n(U+V) \leq \Phi^n(U) + \Phi^n(V)$$

Proposition 4.5. For any $n \in \mathbb{N}^*$, $\Phi^n = \Phi \circ ... \circ \Phi$ (n times) verifies the inequality:

$$\forall V \in L^\infty(\mathbb{R}^*_+), ||\Phi^n(V)||_\infty \leq e^{-rn\Delta t}||V||_\infty$$

4.2.4 Case of uniform approximation of the numerical scheme

We show here that if the local approximation derived in equation (4.3) is uniform, The RL model approximates correctly the UVM highest price. More precisely:

Theorem 4.1. Let C^+ be the upper solution of the BSB equation (3.5) on the interval $[0, T]$ with terminal condition $C^+(., T) = F$. We first assume that C^+ is the UVM highest price and $F \in L^{Pol}(\mathbb{R}^*_+)$. Furthermore, assume that the local error proved in equation (4.3) is henceforth uniform $i.e$ there exists a function $\varepsilon : \mathbb{R}_+^* \to \mathbb{R}_+^*$ such that

$$\lim_{\Delta t \to 0} \varepsilon(\Delta t) = 0$$

And if Δt is small enough:

$$\forall n \in [\![0, N-1]\!], ||\Phi(C_{n+1}) - C_n||_\infty \leq \varepsilon(\Delta t)\Delta t$$

Then there exists $K > 0$ such that $||V_0 - C^+(\cdot, t_0)||_\infty \leq K\varepsilon(\Delta t)$

In this case, the RL price approximates uniformly the highest UVM price, which shows in theory why this our RL model is relevant. The proof below is close to the proof in [Par95, pages 49-50].

Proof. Firstly, denote

$$\forall n \in [\![0, N-1]\!], c_n^+ := \Phi(C_{n+1}^+) - C_n^+ \in L^\infty(\mathbb{R}_+^*)$$

By induction, using proposition 4.4, we have the following inequality:

$$\begin{aligned}
V_0 = \Phi^N(V_N) = \Phi^N(C_N^+) &= \Phi^{N-1}(C_{N-1}^+ + c_{N-1}^+) \\
&\leq \Phi^{N-1}(C_{N-1}^+) + \Phi^{N-1}(c_{N-1}^+) \\
&= \Phi^{N-2}(C_{N-2}^+ + c_{N-2}^+) + \Phi(c_{N-1}^+) \\
&\leq \Phi^{N-2}(C_{N-2}^+) + \Phi(c_{N-2}^+) + \Phi(c_{N-1}^+) \\
&\leq C_0^+ + \sum_{n=0}^{N-1} \Phi^n(c_n^+) = C^+(t_0, .) + \sum_{n=0}^{N-1} \Phi^n(c_n^+)
\end{aligned}$$

For the opposite bound of the inequality, we use a similar reasoning:

$$\begin{aligned}
C^+(t_0, .) = C_0^+ = \Phi(C_1^+) - c_0^+ &= \Phi(\Phi(C_2^+) - c_1^+) - c_0^+ \\
&\leq \Phi^2(C_2^+) + \Phi(-c_1^+) - c_0^+ \\
&\leq \Phi^N(C_N^+) + \sum_{n=0}^{N-1} \Phi^n(-c_n^+) = V_0 + \sum_{n=0}^{N-1} \Phi^n(-c_n^+)
\end{aligned}$$

Using now proposition 4.5, we then get:

$$\begin{aligned} ||V_0 - C^+(t_0, .)||_\infty &\leq \left|\left|\sum_{n=0}^{N-1} \Phi^n(\pm c_n^+)\right|\right|_\infty \\ &\leq \sum_{n=0}^{N-1} ||\Phi^n(\pm c_n^+)||_\infty \\ &\leq \sum_{n=0}^{N-1} e^{-rn\Delta t}||c_n^+||_\infty \\ &\leq \varepsilon(\Delta t)\Delta t \sum_{n=0}^{N-1} e^{-rn\Delta t} \\ &= \frac{\varepsilon(\Delta t)\Delta t(1-e^{-r\tau})}{1-e^{-r\Delta t}} \end{aligned}$$

Since the function $\Delta t \to \frac{\Delta t}{1-e^{-r\Delta t}}$ is bounded on $]0, \tau]$ and Δt lies in this set, the proof is complete. □

5 Numerical Experiments

We implemented the model developed in the previous section. In this section, we present the method and the results of this implementation

5.1 Method of implementation

First of all, all the RL code was implemented in Python. Most of the code is based on [Sim18]. We only implemented the deep Q-learning (DQL) algorithm 2 because this is the only algorithm mentioned in this thesis capable of managing infinite state sets. We also implemented in C++ the BSB scheme defined in part 3.2.4 to compare our results with this method.

We firstly coded the MDP (and the reward function) defined in part 4.1. The associated hyper-parameters are mentioned in table 1. We used the Keras library to build the neural network defined in parts 2.5.1 and 2.5.2. The hyper-parameters of the neural network as well as the memory size and the frequency of target neural network updates are presented in table 2.

One of the biggest problem we encountered is time complexity. This is due to the fact that the neural network weights w are initialised randomly and give at the beginning a very poor and inconsistent estimation of the price. For example the estimated price of an option can be negative whereas it is always strictly positive, or the estimation has not the same order of magnitude (10^4 instead of 10 for instance) as the price. Thus, the neural network needs to be trained for hours in algorithm 2 or in other words has a very long transitional regime before estimating a price that has the same order of magnitude as the true price.

To limit time complexity, we **pre-trained** the neural network on key states. When the portfolio consists of only one derivative with final payoff $F(S_T)$, the idea is to estimate the (deterministic) function F with the neural network by simulating m

Notation	**Hyper-parameter type**	**Value**	**Defined in part**
M	Total number of episodes	400	2.4.2
N	Total number of steps per episode/Time horizon	100	2.4.2
σ_{max}	Volatility upper bound of UVM	0.02	3.2.1
σ_{min}	Volatility lower bound of UVM	0.08	3.2.1
r	Interest rate	0.03	3.2.1
t_0	Initial time	0	4.1.1
S_0	Stock price at time t_0	100	4.1.1
ε_{max}	Maximum exploration probability	1	2.5.2
ε_{min}	Minimum exploration probability	0.01	2.5.2
$\Delta\varepsilon$	Exploration decay coefficient	$\frac{0.02}{N} = 2 \cdot 10^{-4}$	2.5.2

Table 1: Hyper-parameters used to simulate the MDP X

instances of $S_T = S_N$ (say $S_N^{(1)}, ..., S_N^{(m)}$) and setting $(S_N^{(j)}, N)$ as an input of the network and $(F(S_N^{(j)}), F(S_N^{(j)}))$ as an output of the network. In our case, the functions are smooth enough to use a small number of instances distributed regularly over a 99% confidence interval for S_N. In this case, the network should have a very good or even a perfect estimation of the Q-value function at final time N (because the Q-value function at times $n > N$ is null), which makes the convergence of the DQL algorithm much faster.

If we have a portfolio of derivatives with two different maturities $T_0 < T$ whose payoffs are respectively given by $G(S_{T_0})$ and $F(S_T)$, we use a similar method so that the neural network takes into account both F and G. Let $N_0 \in [\![0, N-1]\!]$ be the first integer such that $N_0\Delta t + t_0 \geq T_0$ (this can be interpreted as the discrete maturity of the derivative with payoff $G(S_{T_0})$). To learn G in addition to F (at time N), the neural network learns the following deterministic function at time N_0: $G(S_{N_0}) + F(S_{N_0})$. It is assumed here that $F(S_{N_0})$ has the same order of magnitude

Notation	Hyper-parameter type	Value	Defined in part
α	Learning rate	$1 \cdot 10^{-4}$	2.5.1
k	Batch size	64	2.5.1
	Number of neurons in the hidden layer	50	2.5.1
$\lvert\mathcal{D}\rvert$	Maximum memory size	$2^{20} = 1,048,576$	2.5.2
τ_{max}	Number of steps between two updates of $Q^{(target)}$	$2N = 200$	2.5.2
	Pre-training epoch	$20,000$	5.1
m	Number of instances simulated for pre-training	50	5.1

Table 2: Hyper-parameters for the neural network, experience replay and double Q-learning

as the price of the associated derivative at time N_0. Indeed, contrary to a portfolio consisting of one derivative, the last estimation of the Q-value is not the true Q-value but should have the same order of magnitude. It suffices to run the DQL algorithm 2 to have more accurate estimation of the Q-value at this time. A similar method can be used for several derivatives.

Note that pre-training can be applied in a lot of RL problems where the reward is often null and the key states ($i.e$ states where the agent get a crucial reward to estimate the Q-value function) are easy to simulate. This method can be applied even if the reward function depends on the current state and action (S_n, A_n) plus a random quantity independent of the previous states and actions.

5.2 Results and interpretation

In this part, we compute the UVM highest price of three different portfolios using the BS prices (if all components of the portfolio are either convex or concave), the DQL algorithm 2 and the descretisation of BSB equation defined in part 3.2.4. For the last two methods, the total number of steps is $N = 100$. The DQL estimation is the highest output of the neural network representing the Q-value function. If the BS prices cannot be used to compute the portfolio price, we will use the discretisation of BSB equation to estimate this price with $N = 5000$. This price will be assumed to be the true price.

5.2.1 Price of a Call option

The first portfolio we price consists of only a Call option. Its payoff is given by $F(S) := \max(S - K, 0)$ where $K > 0$ is the *strike*. We chose one unit of Call option with maturity $T = 1$ and strike $K = 95$ Here are the results we obtained after running the DQL algorithm 2:

True price	**DQL estimation**	**DQL time**	**BSB tree estimation**	**BSB tree time**
8.428	9.138	25min 25s	8.424	0.041 s

Table 3: Estimations of a Call option price with maturity $T = 1$ and strike 95. The DQL estimation represents the estimation of the DQL algorithm while the BSB tree estimation represents the estimation of the price using the method in part 3.2.4 (the times represent the running time of each algorithm)

Note that according to proposition 3.1, the true price can be calculated using the BS prices (with volatility parameter σ_{max}) here because a Call option has a convex payoff function. We also present in figures 2 and 3 the prediction of the price by DQL algorithm with respect to the number of episodes and the learning error defined as the absolute value of difference between the true price and the estimated price. Finally, we show in figure 4 the payoff/price predictions of the DQL algorithm just after pre-training (figure 4a) and after running the DQL algorithm (figures 4b and 4c)

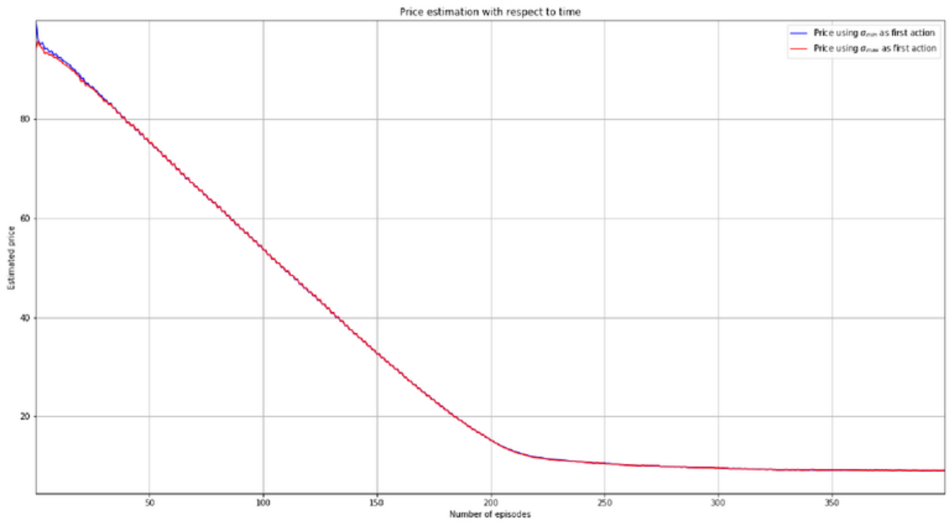


Figure 2: Price estimation using either σ_{min} (blue curve) or σ_{max} (red curve) as a first volatility action. It can be seen that the price starts from approximately 100 to collapse to roughly 15 from first episode to episode 210 (transitional regime). The estimation then gets more steady but still decreases gradually (permanent regime)

Before comparing the results presented in table 3, we focus only our interpretation on the DQL result and the graphs. First of all, the DQL algorithm starts from a value (100) that has not the same order of magnitude as the true price (10). Nevertheless, the neural network predicts almost perfectly the payoff in 30 seconds just after pre-training (see figure 4a) and this speeds up the transitional regime that takes around 200 episodes instead of 2500 without pre-training. Thus, the DQL prediction gets close to the true price in a reasonable time. Besides, as figure 3 shows, the prediction is relatively close to the true price (the learning error is around 0.7) without converging to the true price. This over-estimation can be due to the predictions in figures 4b and 4c. In particular, the estimation of the payoff at maturity N changes and becomes a straight line with a small slope after performing the DQL algorithm (whereas the payoff is still the target at final time N).

From a RL point of view, this result remains good because firstly, the estimation

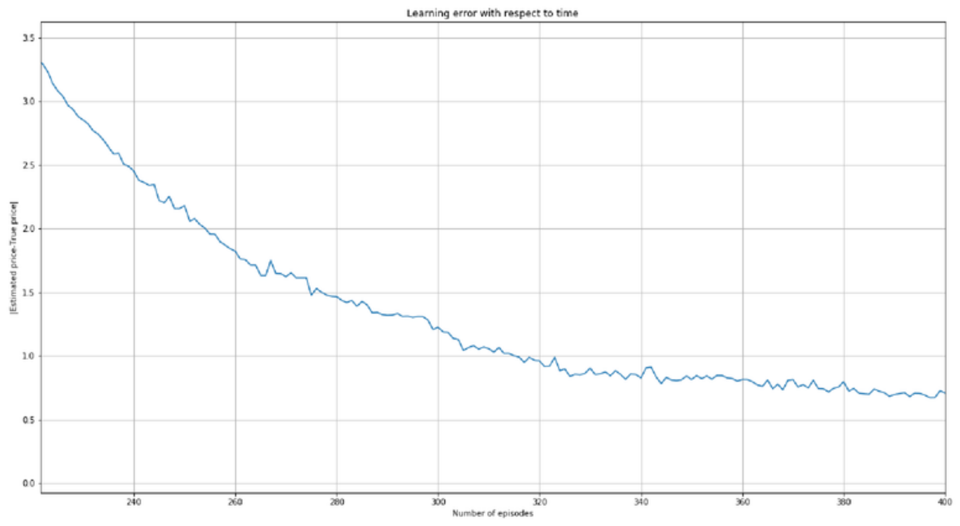
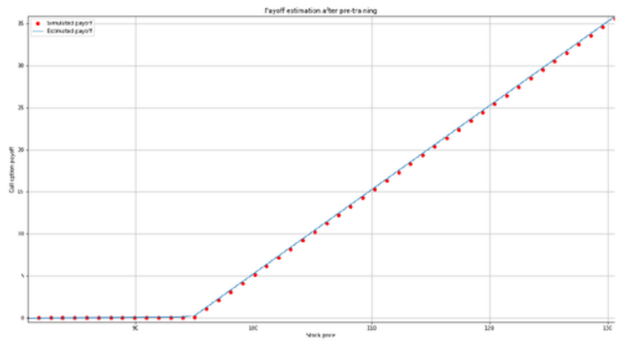


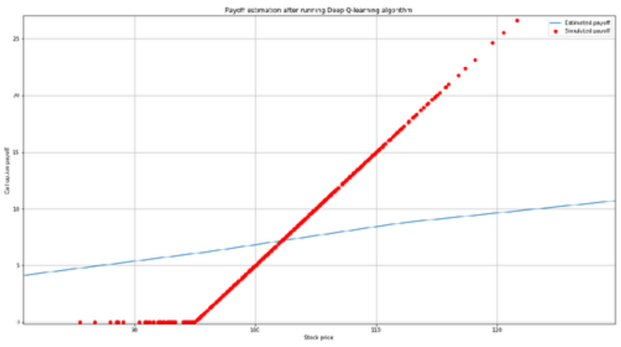
Figure 3: Learning error with respect to the number of episodes during permanent regime. From episode 280 to 400 this error slightly decreases (with a small noise) to reach roughly 0.75

is close to the true price whereas there is no mathematical proof that the DQL algorithm converges to the best expected sum of rewards. Secondly,the execution time is relatively short knowing that this algorithm is very recent and the execution time is generally much longer.

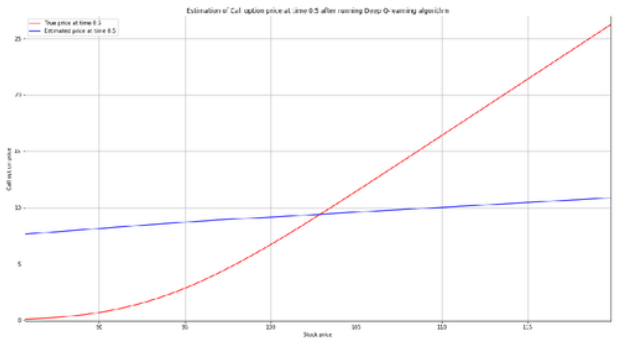
The algorithm also estimates correctly the policy because it predicts most of the time the right action σ_{max}. In figure 2, the estimation using σ_{max} as a first action between times $n = 0$ and $n = 1$ (red curve) is just above the estimation using σ_{min} as a first action (blue curve). Moreover, the two curves are extremely close because the difference between the two values should be linked in theory by the two values $\sigma_{max}\Delta t = 8 \cdot 10^{-4}$ and $\sigma_{min}\Delta t = 2 \cdot 10^{-4}$. Since these values are very close, the two estimations are also very close. Note that to better estimate the price, we could perform a Monte-Carlo method knowing the policy to derive the expectation. However, this method takes an additional amount of time and has not been performed.



(a) Payoff estimation after pre-training. The blue curve represents the estimated payoff and the red dots represent the simulated payoff during pre-training



(b) Payoff estimation after DQL Algorithm. Contrary to above, the red dots represent the simulated payoff by the algorithm



(c) Price estimation at time 0.5 after DQL algorithm. The red (resp. blue) curve represents the true (resp. estimated) price

Figure 4: DQL estimations of payoffs and price after performing a pre-training or the DQL algorithm

Unfortunately, comparing these results to the BSB numerical scheme, it can be easily seen that this scheme is much better than the DQL algorithm in terms of accuracy and speed: this method developed in part 3.2.4 is approximately 30,000 times as fast as the DQL algorithm to get the same (or even a better) accuracy. Note further that this accuracy can be significantly improved by increasing N (say $N = 5000$) in a reasonable execution time (1s to 5s).

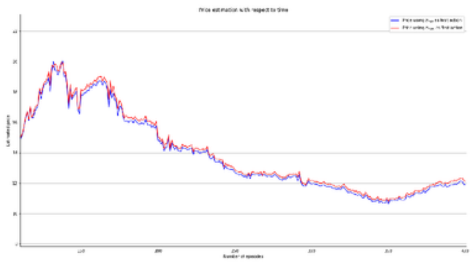
5.2.2 Price of a simple convex portfolio

In the remaining of this section, we give results on two other portfolios but we only add very few comments. The second portfolio we price consists of the same Call option (maturity $T = 1$, strike $K = 95$) as the previous portfolio plus a Put option (whose payoff is given by $F(S) := \max(0, K - S)$) with maturity $T = 0.5$ and strike $K = 105$. Since the two functions are convex (this is why the portfolio is then said to be "convex"), the UVM highest price is also the BS price with volatility σ_{max}. The results are presented below and in figure 5:

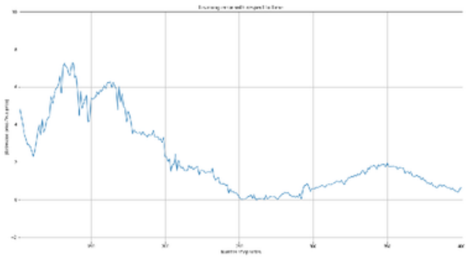
True price	**DQL estimation**	**DQL time**	**BSB tree estimation**	**BSB tree time**
12.839	12.105	27min 14s	11.775	0.054s

Table 4: Estimations of a price of a convex portfolio consisting of one Put option price with maturity $T = 0.5$ and strike 105 and one Call option price with maturity $T = 1$ and strike 95.

In addition to the comments we made previously, the DQL algorithm reaches the permanent regime after 20 minutes and has relatively large oscillations around the price (see figure 5), which here shows a frequent drawback of the algorithm. Nevertheless, our DQL algorithm works correctly because it returns a good approximation of the price. This shows that intermediate rewards are taken into account by the algorithm.



(a) Price



(b) Learning error

Figure 5: Estimation of the convex portfolio price in permanent regime (with respect to the number of episodes). From episode 200 onwards, the error (blue curve in (b)) is less than 2 (bold horizontal line in (b)). However, the estimation may fluctuate without converging to the true price. Note that the best action chosen at time 0 is very often σ_{max} (red curve in (a))

5.2.3 Price of a complex portfolio

We now price a more complex portfolio consisting of a combination of Put and Call options at different maturities that can be either sold (non-negative payoff/quantity=+1)

Option type	Strike K	Maturity T	Quantity	New option type
Call	95	3	1	Bull Call spread
Call	115	3	-1	
Put	93	2	1	Bear Put-Call spread
Call	110	2	-1	
Put	95	1	1	Strangle
Call	107	1	1	
Call	105	0.5	-1	Call
Put	103	0.25	1	Put

Table 5: Composition of the complex portfolio. The new option type is the name of the option if the combination of Call and Put options are joined together

or bought (negative payoff/quantity=-1). This allows us to test if a non-convex (and non-concave) portfolio can be priced by the DQL algorithm. The composition of the portfolio is detailed in table 5 and the results are showed in table 6 below:

True price (BSB tree accurate estimation)	**Time to find the true price**	**DQL estimation**	**DQL time**	**BSB tree estimation**	**BSB tree** time
20.874	5.063 s	23.517	22min 12s	20.818	0.062s

Table 6: Price estimations of the complex portfolio. The true price was computed using the a BSB tree scheme with $N = 5000$. For the two other estimations, N is still equal to 100.

These results show exactly the same behaviour as explained above. The only thing we can add here is the ability of the DQL algorithm to adapt its Q-value estimations despite the potential change of optimal volatility parameter $\sigma \in \mathcal{A}$ even if this adaptation raises a little larger error.

6 Conclusion

In this thesis we used a Reinforcement Learning approach to find the highest price of a portfolio of (European) derivatives under the uncertain volatility model. To do so, we modelled the Markov decision process as a piece-wise Black-Scholes model where the action is the volatility parameter (between σ_{min} and σ_{max}). We explained why the best possible volatility choice (to maximise the discounted expectation of the payoff under pricing/risk-neutral measure) yields a price close to the Black-Scholes-Barenblatt equation, a partial differential equation satisfied by the UVM highest price.

In order to manage an infinite state space, we used the deep Q-learning algorithm. This learning algorithm approximated correctly different portfolios of derivatives with a non-negligible but relatively small error within 20 to 25 minutes. Knowing that this algorithm is very recent, does not necessarily converge to the optimal value, and takes generally hours for training, it is very efficient from a Reinforcement learning point of view and gives promising results.

However, we also tested a numerical scheme/recombining tree introduced in [ALP95], that remains much better than the Reinforcement learning approach. Indeed, this method is roughly 30,000 times as fast as the deep Q-learning algorithm (less than 0.1s) to return a similar or even a better result. Moreover, using this method, the highest price under the uncertain volatility model is also very well approximated within 1 to 5 seconds. It then turns out that this method is much more appropriate for uncertain volatility pricing.

As a potential future work, one can think about hedging an option with a set of vanilla options (*i.e* reducing the extreme prices or the risk of an option using vanilla options) under either the Reinforcement Learning model we introduced in this thesis or the recombining tree (this scheme also gives implicitly the volatility choice). To do so, exactly like [Bue+19], one can implement a second Reinforcement Learning model where the MDP has exactly the same state process as above but the action is here the quantity of vanilla options. Similarly as above the reward is the payoff of the option (times the quantity) if maturity is reached.

The only difference between approach in [Bue+19] and our approach lies in volatility. In [Bue+19] volatility is simulated from a well-known model (*e.g* Black-Scholes, Heston), whereas in our approach, volatility is determined by either our Reinforcement Learning model above or the recombining tree given the quantities for each vanilla option. To generalise the optimal volatility policy for every possible quantity, one can use for example a neural network where the inputs are the quantities and the state and the output is the optimal volatility or a probability of being the optimal volatility for each volatility parameter σ_{min} and σ_{max}. Once this task has been done, hedging can be implemented like [Bue+19].

References

[ALP95] M Avellaneda, A. Levy, and A Parás. "Pricing and hedging derivative securities in markets with uncertain volatilities". In: *Appl Math Finance* 2 (1995), pp. 73–88. URL: https://www.math.nyu.edu/faculty/avellane/UVMfirst.pdf.

[BS73] Fischer Black and Myron Scholes. "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3 (1973), pp. 637–54. URL: https://www.journals.uchicago.edu/doi/pdfplus/10.1086/260062.

[Bri+19] Marc Brittain et al. "Prioritized Sequence Experience Replay". In: *CoRR* abs/1905.12726 (2019). arXiv: 1905.12726. URL: http://arxiv.org/abs/1905.12726.

[Bue+19] Hans Buehler et al. "Deep Hedging: Hedging Derivatives Under Generic Market Frictions Using Reinforcement Learning". In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3355706.

[Gra] Thomas Grassl. *A reinforcement learning approach for pricing derivatives.* URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=AD10411FF5C85C570D8D04445102AC88?doi=10.1.1.374.9702&rep=rep1&type=pdf.

[Hal17] Igor Halperin. "QLBS: Q-Learner in the Black-Scholes(-Merton) Worlds". In: *SSRN Electronic Journal* (Dec. 2017). DOI: 10.2139/ssrn.3087076.

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[LLX19] Xinpeng Li, Yiqing Lin, and Weicheng Xu. "On properties of solutions to Black–Scholes–Barenblatt equations". In: *Advances in Difference Equations* 2019 (2019). DOI: 10.1186/s13662-019-2135-z.

[Lu+17] Zhou Lu et al. "The Expressive Power of Neural Networks: A View from the Width". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 6231–6239. URL: `http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf`.

[Mni+13] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013). URL: `https://arxiv.org/pdf/1312.5602.pdf`.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. 2nd ed. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2018. 504 pp. ISBN: 978-0-262-03940-6. URL: `https://cs.nyu.edu/~mohri/mlbook/`.

[Par95] Antonio Parás. "Non-Linear Diffusion Equations in Mathematical Finance: A Study of Transaction Costs and Uncertain Volatility". PhD thesis. New York University, 1995.

[Pha+19] Tung Lam Pham et al. "A Novel Neural Network-Based Method for Decoding and Detecting of the DS8-PSK Scheme in an OCC System". In: *Applied Sciences* 9.11 (2019). ISSN: 2076-3417. DOI: `10.3390/app9112242`. URL: `https://www.mdpi.com/2076-3417/9/11/2242`.

[Sch+16] Tom Schaul et al. "Prioritized Experience Replay". In: *International Conference on Learning Representations*. 2016. URL: `https://arxiv.org/pdf/1511.05952.pdf`.

[Sim18] Thomas Simonini. *Deep Reinforcement Learning course*. 2018. URL: `https://simoninithomas.github.io/Deep_reinforcement_learning_Course/`.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/bookdraft2017nov5.pdf`.

[Tou10] Nizar Touzi. *Optimal stocahastic control, Stochastic target problems, and Backward SDEs*. Ecole Polytechnique, Département des mathématiques appliquées. May 2010, pp. 22–42. URL: `http://www.cmap.polytechnique.fr/~touzi/Fields-LN.pdf`.

[Wer82] Paul Werbos. “Applications of Advances in Nonlinear Sensitivity Analysis”. In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*. 1982, pp. 762–770. URL: `http://werbos.com/Neural/SensitivityIFIPSeptember1981.pdf`.

A Proofs of results on Bellman equations

In this appendix, we prove propositions 2.1, 2.2, 2.6 and 2.7. To do so, we remind two useful theorems of functional analysis below:

Theorem A.1. (*Banach's fixed point theorem*) Let (X, d) be a complete metric space and let $\gamma \in [0, 1[$. If $\phi : X \to X$ is a γ-contracting mapping ($i.e$ for any $x, y \in X, d(\phi(x), \phi(y)) \leq \gamma d(x, y)$) then there exists a unique x_∞ such that $\phi(x_\infty) = x_\infty$. Furthermore, the sequence $(x_n)_{n\in\mathbb{N}}$ defined by the recursive relation $x_{n+1} := \phi(x_n)$ with an arbitrary $x_0 \in X$ converges to x_∞ and also verifies $d(x_\infty, x_n) \underset{n\to+\infty}{=} O(\gamma^n)$.

Theorem A.2. Let $\mathcal{S}$ be a set. The space $L^\infty(\mathcal{S}) := \{V : \mathcal{S} \to \mathbb{R} \text{ measurable and bounded}\}$ with uniform norm $||\cdot||_\infty$ is a complete metric space.

Proof of proposition 2.1. This proof does not use the above results. A simple calculus yields the above result (2.2). Let $x \in \mathcal{S}$, we have:

$$\begin{aligned} V_\pi(x) &= \mathbb{E}_\pi\left[\sum_{k=0}^{+\infty}\gamma^k R(X_k, A_k)\middle| X_0 = x\right] \\ &= \mathbb{E}_\pi\left[R(X_0, A_0) + \gamma\sum_{k=1}^{+\infty}\gamma^{k-1}R(X_k, A_k)\middle| X_0 = x\right] \\ &= \mathbb{E}_\pi\left[R(x, A_0) + \gamma\mathbb{E}_\pi\left[\sum_{k=0}^{+\infty}\gamma^k R(X_{k+1}, A_{k+1})\middle| X_1\right]\middle| X_0 = x\right] \text{ by linearity and tower property} \\ &= \mathbb{E}_\pi\left[R(x, A_0) + \gamma V_\pi(X_1)|X_0 = x\right] \end{aligned}$$

Since the distribution of (X, A) does not depend on time n, the expression $\mathbb{E}_\pi\left[\sum_{k=0}^{+\infty}\gamma^k R(X_{k+1}, A_{k+1})|X_1\right]$ is another definition of the value function, which shows the last equality. □

Proof of proposition 2.2. Set for every $V \in L^\infty(\mathcal{S})$ the functional map:

$$\Psi(V) := \hat{R} + \gamma P(V) \tag{A.1}$$

Where for every $x \in \mathcal{S}$:

$$\begin{cases} \hat{R}(x) & := \int_{a\in\mathcal{A}} R(x, a)\pi(da|x) \\ P(V)(x) & := \int_{(a,x')\in\mathcal{A}\times\mathcal{S}} V(x')p(dx'|x, a)\pi(da|x) \end{cases} \tag{A.2}$$

Any point in $L^\infty(\mathcal{S})$ satisfies Bellman equation (2.2) if and only if it is a fixed point of I. This the case for V_π according to proposition 2.1 and because V_π is bounded and measurable (it is defined from a conditional expectation). We show here that I is γ-contracting to apply fixed-point theorem A.1 and thus show the expected result. We firstly notice that P is a linear mapping verifying:

$$\forall (V,x) \in L^\infty(\mathcal{S}) \times \mathcal{S}, |P(V)(x)| \leq \int_{(a,x') \in \mathcal{A} \times \mathcal{S}} ||V||_\infty p(dx'|x,a)\pi(da|x) \leq ||V||_\infty$$

Hence, it turns out that for any $U, V \in L^\infty(\mathcal{S})$,

$$||\Psi(U) - \Psi(V)||_\infty \leq \gamma ||P(U-V)||_\infty \leq \gamma ||U - V||_\infty$$

Which concludes the proof. □

Remark A.1. If $\mathcal{S} \times \mathcal{A}$ is finite, then the boundedness condition is satisfied and functions $\hat{R}$ and P are respectively finite-dimensional vectors and finite-dimensional matrix. This proof and fixed point theorem will help us define a method to find V_π later.

Proof of proposition 2.6. This result is a consequence of fixed point theorem A.1 because ψ is γ-contracting and $L^\infty(\mathcal{S} \times \mathcal{A})$ is complete according to theorem A.2 (replace $\mathcal{S}$ by $\mathcal{S} \times \mathcal{A}$). Let $(Q_1, Q_2, x, a) \in L^\infty(\mathcal{S} \times \mathcal{A})^2 \times \mathcal{S} \times \mathcal{A}$. Let $a^*(x)$ denote the action maximising Q_1 knowing x. We then have:

$$\begin{aligned}\psi(Q_1)(x,a) - \psi(Q_2)(x,a) &\leq \gamma \int_{x' \in \mathcal{S}} (Q_1(x', a^*(x')) - Q_2(x', a^*(x')))p(dx'|x,a), \\ &\leq \gamma \int_{x' \in \mathcal{S}} ||Q_1 - Q_2||_\infty p(dx'|x,a) = \gamma ||Q_1 - Q_2||_\infty\end{aligned}$$

This inequality shows that ψ is γ-contracting, and then proves the result. □

Proof of proposition 2.7. Using equations in remark 2.2, we have:

$$\forall (x,a) \in \mathcal{S} \times \mathcal{A}, Q_*(x,a) = R(x,a) + \gamma \mathbb{E}_{\pi^*}\left[Q_*(X_1, A_1) | X_0 = x, A_0 = a\right]$$

But according to proposition 2.4 and since π^* is optimal, $A_1 \in \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q_*(X_1, a')$ almost surely. This yields the expected result. □

B Some theorems on Stochastic control and proof of BSB equation

We state below some theoretical results on stochastic control in the dimension 1 case. A more general version of these results can be found in [Tou10, Chapter 2, pages 22-42] with all the proofs. No proof of those results will be developed in this appendix. Afterwards, we use this theory to show that the UVM extreme prices satisfy the BSB equation (theorem 3.3).

Definition B.1. A **control set** is a borel set $U \subset \mathbb{R}$

Let $T \in [0, +\infty[$ and let W be a brownian motion. Set for every $t \in [0, T]$, $\mathcal{F}_t = \sigma(W_u, u \leq t)$ to get the natural filtration $(\mathcal{F}_t)_{0 \leq t \leq T}$

Definition B.2. The **set of possible control processes** is defined by:

$$\mathcal{U}_0 := \left\{ (\nu_t)_{0 \leq t \leq T} : [0, T] \times \Omega \to U : \left\| \begin{array}{c} \text{progressively measurable} \\ \mathbb{E}\left[\int_0^T \nu_t^2 dt\right] < +\infty \end{array} \right. \right\} \tag{B.1}$$

Where the progressive measurability is with respect the filtration $(\mathcal{B}([0, t]) \bigotimes \mathcal{F}_t)_{0 \leq t \leq T}$ and the borel σ-field $\mathcal{B}(U)$

Given a control process $\nu \in \mathcal{U}_0$, we consider the controlled stochastic differential equation:

$$dS_t^\nu = b(t, S_t^\nu, \nu_t)dt + \varsigma(t, S_t^\nu, \nu_t)dW_t \tag{B.2}$$

Where $b : [0, T] \times \mathbb{R} \times U \to \mathbb{R}$ and $\varsigma : [0, T] \times \mathbb{R} \times U \to \mathbb{R}$ are two functions.

Theorem B.1. If there exists $K > 0$ such that for every $(t, (x, y), u) \in [0, T] \times \mathbb{R}^2 \times U$,

$$\begin{cases} |b(t, x, u) - b(t, y, u)| + |\varsigma(t, x, u) - \varsigma(t, y, u)| & \leq K|x - y| \\ |b(t, x, u)| + |\varsigma(t, x, u)| & \leq K(1 + |x| + |u|) \end{cases}$$

Then given a constant initial condition $S_{t_0}^\nu = x \in \mathbb{R}$ at time $t_0 \in [0, T[$, the stochastic differential equation (B.2) has a unique solution denoted by $S^{t_0, x, \nu}$ on the interval $[t_0, T]$

This theorem allows us to define the value function in the stochastic control context.

Definition B.3. Let $g, r : [0,T[\times\mathbb{R}\times U \to \mathbb{R}$ be two continuous functions such that r is bounded and let $f : \mathbb{R} \to \mathbb{R}$ be a function verifying the quadratic growth condition:

$$\exists K' > 0, \forall (t,x,u) \in [0,T[\times\mathbb{R}\times U, |g(t,x,u)| + |f(x)| \leq K'(1 + |u| + x^2)$$

A **value function** $C : [0,T]\times\mathbb{R} \to \mathbb{R}$ is the function defined by the expression (for every $(t,x) \in [0,T]\times\mathbb{R}$):

$$\begin{aligned} C(t,x) &= \sup_{\nu\in\mathcal{U}_0} \mathbb{E}\left[\int_t^T \beta^\nu(t,s) g(s, S_s^\nu, \nu_s) ds + \beta^\nu(t,T) f(S_T^\nu) \Big| S_t^\nu = x\right] \\ &= \sup_{\nu\in\mathcal{U}_0} \mathbb{E}\left[\int_t^T \beta^\nu(t,s) g(s, S_s^{t,x,\nu}, \nu_s) ds + \beta^\nu(t,T) f(S_T^{t,x,\nu})\right] \end{aligned}$$

where $\beta^\nu(t,s) = e^{-\int_t^s r(\theta, S_\theta^\nu, \nu_\theta) d\theta}$ in the first equality and $\beta^\nu(t,s) = e^{-\int_t^s r(\theta, S_\theta^{t,x,\nu}, \nu_\theta) d\theta}$ in the second equality

Definition B.4. Define the map H:

$$H : \begin{array}{ccc} [0,T]\times\mathbb{R}^4 & \to & \mathbb{R} \\ (t,x,\alpha,\delta,\gamma) & \to & \sup\limits_{u\in U} \begin{pmatrix} -r(t,x,u)\alpha + b(t,x,u)\delta \\ +\frac{1}{2}(\varsigma(t,x,u))^2\gamma + g(t,x,u) \end{pmatrix} \end{array}$$

We now state the main result of this appendix to show that the UVM extreme prices satisfy the BSB equation.

Theorem B.2. Suppose firstly that the functions b and ς satisfy the conditions in theorem B.1 and secondly the functions r, f and g satisfy the conditions in definition B.3 and $C \in \mathcal{C}^{1,2}([0,T[\times\mathbb{R},\mathbb{R})$. Suppose further that H is continuous and:

$$\forall (t,x) \in [0,T[\times\mathbb{R}, H\left(t, x, C(t,x), \frac{\partial C}{\partial x}(t,x), \frac{\partial^2 C}{\partial x^2}(t,x)\right) > -\infty$$

Then C satisfies the Hamilton-Jacobi-Bellman equation on $[0,T[\times\mathbb{R}$:

$$\frac{\partial C}{\partial t}(t,x) + H\left(t, x, C(t,x), \frac{\partial C}{\partial x}(t,x), \frac{\partial^2 C}{\partial x^2}(t,x)\right) = 0 \tag{B.3}$$

A more general version of theorem B.1 with a proof can be found in [Tou10, Theorem 2.1, page 28]. Regarding theorem B.1, its more general version (with a proof) can be found in [Tou10, Theorem 2.6, page 38]

Proof of theorem 3.3. First of all, the terminal condition immediately holds if we look at time T. Regarding the BSB equation (3.5) for C^+, it is a particular case of the HJB equation (B.3) where $r > 0$ is a constant and:

$$\begin{cases} U & := & [\sigma_{min}, \sigma_{max}] \\ b(t,S,u) & := & rS \\ \varsigma(t,S,u) & := & Su \\ g(t,S,u) & := & 0 \\ f(S) & := & F(S) \end{cases}$$

We then get:

$$\begin{aligned} H(t,S,\theta,\alpha,\gamma) &= -r\theta + rS\alpha + \frac{1}{2}S^2 \sup_{\sigma_{min} \leq u \leq \sigma_{max}} (u^2\gamma) \\ &= -r\theta + rS\alpha + \frac{1}{2}\left(\Sigma^+(\gamma)\right)^2 S^2\gamma \end{aligned}$$

All the constraints in Theorem B.2 are checked (H is continuous) apart from the smoothness of C^+. This last hypothesis is assumed in the theorem statement. We can then use the Theorem B.2 to show that:

$$\frac{\partial C^+}{\partial t} - rV^+ + rS\frac{\partial C^+}{\partial S} + \frac{1}{2}\left(\Sigma^+\left(\frac{\partial^2 C^+}{\partial S^2}\right)\right)^2 S^2 \frac{\partial^2 C^+}{\partial S^2} = 0$$

We also obtain the expected result for C^- by applying the above result with $D := -C^-$ instead of C^+. The only change is $f(S) = -F(S)$. Once the BSB equation is derived for D, it suffices to replace D by $= -C^-$ and to notice that for every $x \in \mathbb{R}, \Sigma^+(-x) = \Sigma^-(x)$ to derive the BSB equation for C^-. □

C Proofs of properties satitisfied by the map Φ

In this last appendix, we show propositions 4.1,4.2, 4.3, 4.4 and 4.5 that are helpful to understand the operator Φ as well as the link between our RL problem and the BSB equation.

Proof of proposition 4.1. The operator Φ is well defined if and only if for any $V \in L^{Pol}(\mathbb{R}_+^*)$, $\Phi(V)$ exists and belongs to $L^{Pol}(\mathbb{R}_+^*)$. Let V be such a function and let $\sigma \in \mathcal{A}$. By definition, there exists $K > 0$ and $m \in \mathbb{N}$ such that:

$$\forall s \in \mathbb{R}_+^*, |V(s)| \leq K(1 + s^m)$$

and for $s \in \mathbb{R}_+^*$,

$$\begin{aligned}\left|\int_{-\infty}^{+\infty} \phi(x) V\left(se^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)\Delta t}\right) dx\right| &\leq \int_{-\infty}^{+\infty} \phi(x) \left|V\left(se^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)\Delta t}\right)\right| dx \\ &\leq K \int_{-\infty}^{+\infty} \phi(x)\left(1 + \left(se^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)\Delta t}\right)^m\right) dx \\ &\leq K\left(1 + s^m \int_{-\infty}^{+\infty} \phi(x) e^{m\sigma\sqrt{\Delta t}x + m\left(r - \frac{\sigma^2}{2}\right)\Delta t} dx\right)\end{aligned}$$

The last integral is finite because ϕ is a $O(e^{-\frac{x^2}{2}})$ as $x \to \pm\infty$. Knowing that $\mathcal{A}$ is finite, the proof is complete. □

Proof of proposition 4.2. As reminder, C^+ is defined by the expression

$$\forall t, s \in [0, T] \times \mathbb{R}_+^*, C^+(s, t) = \sup_{\sigma \in Vol} \mathbb{E}^{\mathbb{Q}}\left[F(S_T) | S_t = s\right]$$

Since $F \in L^{Pol}(\mathbb{R}_+^*)$, it follows that for any UVM process S with initial condition s at time t:

$$\left|\mathbb{E}^{\mathbb{Q}}\left[F(S_T)|S_t = s\right]\right| \leq K\left(1 + \mathbb{E}^{\mathbb{Q}}\left[S_T^m | S_t = s\right]\right)$$

Since the new payoff function on the right-hand-side of the inequality $s \to s^m$ is convex, the supremum on Vol is a BS model with volatility σ_{max} according to proposition 3.1. In this case, S_T^m follows a log-normal distribution exatly like the proof of proposition 4.1. The proof is thus complete. □

Proof of proposition 4.3. We prove this result by a backward induction. By definition, $V_N = F$. Let $n \in [\![0, N-1]\!]$. According to proposition 2.4, the optimal policy π^* verifies for any state $x = (s, n) \in \mathcal{S}$ the condition $\forall \sigma \in \mathcal{A}, \pi^*(\sigma|x) > 0 \implies \sigma \in \underset{\sigma \in \mathcal{A}}{\operatorname{argmax}}\, Q_{\pi^*}(x, \sigma)$. Furthermore, remark 2.2 shows that for $\sigma_c \in \mathcal{A}$:

$$Q_{\pi^*}(x, \sigma_c) = \gamma \mathbb{E}^{\mathbb{Q}}\left[V_{\pi^*}(X_1)|X_0 = (s, n), \sigma_0 = \sigma_c\right] = \gamma \mathbb{E}^{\mathbb{Q}}\left[V_{n+1}(X_1)|X_0 = (s, n), \sigma_0 = \sigma_c\right]$$

Because the reward is null. Applying again remark 2.2 yields:

$$V_n(s) = \max_{\sigma_c \in \mathcal{A}} Q_{\pi^*}(x, \sigma_c) = \gamma \mathbb{E}^{\mathbb{Q}}\left[V_{n+1}(X_1)|X_0 = (s, n), \sigma_0 = \sigma_c\right] = \Phi(V_{n+1})$$

Because the above expectation considers a log-normal random variable driven by a BS process. It follows immediately from proposition 4.1 that $V_n \in L^{Pol}(\mathbb{R}_+^*)$ $\square$

Proof of proposition 4.4. We prove this result by induction. For the case $n = 1$, Let $\sigma \in \mathcal{A}$. The operator

$$\Phi_\sigma : V \to \int_{\mathbb{R}} e^{-r\Delta t} \phi(x) V\left(\cdot \times e^{\sigma\sqrt{\Delta t}x + \left(r - \frac{\sigma^2}{2}\right)}\right) dx$$

is linear, non-decreasing ($i.e$ if $U \leq V$ for $U, V \in L^{Pol}(\mathbb{R}_+^*)$, then $\Phi_\sigma(U) \leq \Phi_\sigma(V)$) and $\Phi = \max_{\sigma \in \mathcal{A}} \Phi_\sigma$. Hence, for any $U, V \in L^{Pol}(\mathbb{R}_+^*)$,

$$\Phi_\sigma(U + V) = \Phi_\sigma(U) + \Phi_\sigma(V) \leq \Phi(U) + \Phi(V)$$

Taking the maximum with respect to σ on the left-hand-side of the inequality shows the sub-additivity of Φ. If in addition $U \leq V$, then

$$\Phi_\sigma(U) \leq \Phi_\sigma(V) \leq \Phi(V)$$

Likewise, the maximum with respect to σ on the left-hand-side of the inequality shows that Φ is non-decreasing.

Let $n \in \mathbb{N}^*$ such that Φ^n is both sub-additive and non-decreasing. First, if $U, V \in L^{Pol}(\mathbb{R}_+^*)$ and $U \leq V$,then:

$$\Phi^{n+1}(U) = \Phi^n(\Phi(U)) \leq \Phi^n(\Phi(V)) = \Phi^{n+1}(U)$$

Which shows that Φ^{n+1} is non-decreasing as well. If now $U, V \in L^{Pol}(\mathbb{R}_+^*)$, then $\Phi(U+V) \leq \Phi(U) + \Phi(V)$ and since Φ^n is non-decreasing then:

$$\Phi^{n+1}(U+V) \leq \Phi^n(\Phi(U) + \Phi(V)) \leq \Phi^{n+1}(U) + \Phi^{n+1}(V)$$

Which completes the proof. $\square$

Proof of proposition 4.5. Let $\sigma \in \mathcal{A}$ and $V \in L^\infty(\mathbb{R}_+^*)$, using the same notations as the last proof, we have:

$$||\Phi_\sigma(V)||_\infty \leq e^{-r\Delta t} \int_{\mathbb{R}} \phi(x) ||V||_\infty dx = e^{-r\Delta t} ||V||_\infty$$

Taking the maximum in $\mathcal{A}$ (a finite set), we obtain the expected result for $n = 1$ since the right-hand-side of the inequality does not depend on σ. The end of the proof is an induction. Let n such that the expected result is satisfied. It turns out that:

$$||\Phi^{n+1}(V)||_\infty = ||\Phi(\Phi^n(V))||_\infty \leq e^{-r\Delta t} ||\Phi^n(V)||_\infty \leq e^{-r(n+1)\Delta t} ||V||_\infty$$

$\square$

REINFORCEMENT LEARNING FOR DERIVATIVE PRICING UNDER THE UNCERTAIN VOLATILITY MODEL

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64

PAGE 65

PAGE 66

PAGE 67