Some applications of machine learning in finance

Zineb El Filali Ech-Chafiq

Université Grenoble Alpes

Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information Informatique

Laboratoire Jean Kuntzmann

Thèse présentée pour obtenir le grade universitaire de Docteur

Spécialité: Mathématiques Appliquées

Zineb EL FILALI ECH-CHAFIQ

Quelques applications de l'apprentissage automatique en finance

Some applications of machine learning in finance

Sous la direction de Jérôme Lelong

Soutenance devant le jury composé de:

Jean François COEURJOLLY	Président du Jury
Agnès SULEM	Rapportrice
Stéphane CREPEY	Rapporteur
Emmananuel GOBET	Examinateur
Ludovic GOUDENEGE	Examinateur
Jérôme LELONG	Directeur de thèse
Adil REGHAI	Encadrant en entreprise

Abstract

In this thesis, we study some applications of machine learning in mathematical finance. We first review some interesting applications from the literature before we present our main contribution which includes two algorithms for option pricing using machine learning. The first application consists in building automatic control variates using neural networks for option pricing with Monte Carlo. The second application consists of an algorithm for pricing Bermudan options using regression trees and random forests.

In the first part, we look at the problem of European option pricing using Monte Carlo. First, we recall some well-known variance reduction techniques that allow to optimize the performance of the Monte Carlo pricer. Then, we present some fast pricing methods using machine learning. Finally, we discuss our own contribution to the subject which consists of two methods for building control variates using neural networks which can be conceived as a conservative method for fast pricing using neural networks. The first algorithm relies on the fact that many high-dimensional financial problems are of low effective dimensions. We expose a method to reduce the dimension of such problems in order to keep only the necessary variables. The pricing can then be done using fast numerical integration techniques such as Gaussian quadrature. The second approach consists of building an automatic control variate using an adapted neural network. We learn the function to be integrated (which incorporates the diffusion model plus the payoff function) in order to build a network that is highly correlated to it. As the network that we use can be integrated exactly, we can use it as a control variate.

In the second part, we focus on the problem of pricing Bermudan options. The challenge with these options comes from the determination of an optimal stopping time to exercise the option. This problem can be formulated as a dynamic programming equation which we expose and propose various methods for solving it: quantization, some least-squares methods, random trees, and stochastic mesh algorithms. We then review two new alternative methods for pricing Bermudan options without using the dynamic programming equation. The first method consists in estimating the optimal stopping time through a randomized exercise strategy. The second method approximates the optimal stopping time using neural networks. As for our contribution, we suggest solving the dynamic programming equation using regression trees and random forests. We prove that using regression trees to estimate the conditional expectations in the dynamic programming equation allows us to converge to the correct price. We show through numerical examples how random forests can solve the curse of dimensionality for Bermudan options.

Key words: Monte Carlo, variance reduction, neural networks, American options, regression trees, random forests, fast pricing

Résumé

Dans cette thèse, nous étudions certaines applications de l'apprentissage automatique en mathématiques financières. Nous faisons d'abord un compte rendu de certaines applications intéressantes de la littérature avant de présenter notre propre contribution qui comprend deux algorithmes de valorisation d'options utilisant l'apprentissage automatique. La première application consiste à construire des variables de contrôle automatiques à l'aide de réseaux de neurones pour la valorisation d'options européennes avec Monte Carlo. La deuxième application consiste en un algorithme de valorisation d'options bermudéennes à l'aide d'arbres de régression et de forêts aléatoires.

Dans la première partie, nous examinons le problème de la valorisation d'options européennes avec Monte Carlo. Tout d'abord, nous rappelons quelques techniques de réduction de variance qui permettent d'optimiser les performances de l'estimateur Monte Carlo. Ensuite, nous présentons quelques méthodes de fast pricing utilisant l'apprentissage automatique. Enfin, nous exposons notre propre contribution qui consiste en deux méthodes pour construire des variables de contrôle à l'aide de réseaux de neurones. Nos algorithmes peuvent être conçus comme des méthodes conservatrices de fast pricing à l'aide de réseaux de neurones. Le premier algorithme que nous proposons repose sur le fait que plusieurs problèmes de grande dimension en finance ont de petites dimensions effectives. Nous exposons une méthode pour réduire la dimension de tels problèmes afin de ne garder que les variables nécessaires. La valorisation peut alors être effectuée à l'aide de techniques d'intégration numérique rapide telles que des quadratures gaussiennes. La deuxième approche consiste à construire une variable de contrôle automatique à l'aide d'un réseau de neurones adapté. Nous apprenons la fonction à intégrer (qui contient le modèle de diffusion plus le payoff) afin de construire un réseau qui lui est fortement corrélé. Comme le réseau que nous utilisons peut être intégré exactement, nous pouvons l'utiliser comme variable de contrôle.

Dans la deuxième partie, nous nous concentrons sur le problème de la valorisation d'options bermudéennes. Le challenge avec ces options vient de la détermination d'un temps d'arrêt optimal pour exercer l'option. Ce problème peut être formulé comme une équation de programmation dynamique que nous exposons et proposons différentes méthodes pour la résoudre : la quantification, certaines méthodes de moindres carrés, des arbres aléatoires, et des algorithmes de maillage stochastique. Nous passons ensuite en revue deux nouvelles méthodes alternatives de valorisation d'options bermudéennes qui ne reposent pas sur l'équation de programmation dynamique. La première méthode consiste à estimer le temps d'arrêt optimal grâce à une stratégie d'exercice randomisée. La deuxième méthode estime le temps d'arrêt optimal à l'aide de réseaux de neurones. En ce qui concerne Notre contribution, nous proposons de résoudre l'équation de programmation dynamique à l'aide d'arbres de régression et de forêts aléatoires. Nous prouvons que l'utilisation d'arbres de régression pour

estimer les espérances conditionnelles dans l'équation de programmation dynamique nous permet de converger vers le bon prix de l'option. Nous montrons à travers des exemples numériques comment la méthode des forêts aléatoires peut résoudre le problème des grandes dimensions pour les options bermudéennes.

Remerciements

Mes remerciements s'adressent tout d'abord à mon Directeur de thèse, Jérôme LELONG, qui a su m'apporter tout au long de cette thèse un soutien sans faille, une disponibilité, une écoute et des conseils précieux et avisés.

Ils s'adressent aussi à Adil REGHAI, qui a également encadré ce travail de recherche et m'a accueilli au sein de son équipe. Je le remercie infiniment pour ses conseils constructifs qui ont guidé ce travail jusqu'au bout.

Je remércie également Pierre HENRY LABORDERE qui a aussi encadré une partie de cette thèse. Je suis très heureuse d'avoir croisé le chemin d'une personne aussi compétente.

J'exprime également mes remerciements à Agnès SULEM et Stéphane CREPEY qui m'ont fait l'honneur d'être les rapporteurs de cette thèse.

Je tiens à remercier tous les membres du jury de ma thèse qui ont accepté de se déplacer à Grenoble pour assister à ma soutenance: Emmanuel GOBET, Ludovic GOUDENEGE et Jean-François COEURJOLLY.

Je remércie également Ludovic GOUDENEGE et Clémentine PRIEUR d'avoir assuré le suivi de ma thèse pendant ces trois années. Leurs remarques constructives m'ont beaucoup servi pour améliorer mon travail.

Je tiens à remercier tout particulièrement mes collègues de l'équipe de recherche quantitatives dérivées actions et matières premières de Natixis pour leur disponibilité, leur partage d'information et pour l'environnement de travail agréable qu'ils m'ont offerts: Marouane MESSAOUD, Claude MULLER, Luc MATHIEU, Rida MAHI, Olivier CROISSANT, Joachim ADRIEN, Florian MONCIAUD, Rachid LOUZIR et Othmane KETTANI.

Enfin, je remercie ma mère, Aicha JAFARI, de m'avoir transféré son amour pour les mathématiques et d'avoir tant sacrifié pour voir le résultat de ce projet de thèse concrétisé.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 History of machine learning and the industrial challenge in the financial field

In 1950, in his paper "Computing Machinery and Intelligence," Alan Turing discussed the idea of creating intelligent machines and introduced the Turing test, originally called the imitation game. He proposed an experience that tests a machine's ability to carry out an intelligent human-like conversation. Judged by a human evaluator, the machine would discuss with another human anonymously. The three participants are all separated, and the conversation is restricted only to text. If the evaluator could not tell the machine from the human, then the machine would pass the Turing test. Ever since, it has become a challenge for researchers to create such a machine. However, the idea of building intelligent objects was not at all new. Ancient Greeks had myths about mechanical men mimicking human behavior, Egyptian engineers built automatons in the Bronze Age, and Chinese artisans built an entire mechanical orchestra for their emperor 300 years BC. Decidedly, humankind has always dreamt of making inanimate objects come alive, copying human intelligence to the fullest extent.
In 1956, at the Dartmouth Summer Research Project, researchers formally founded Artificial Intelligence, a field that studies the concept of machines carrying out tasks that we would consider intelligent. Researchers put much work into this, but the conditions were not of help. Computers could not store enough commands, and computing was very costly. Consequently, AI went through some dry periods called AI winters where research was not moving forward. In 1997, IBM revived the field by presenting Deep Blue, the first machine ever to beat a chess champion. In the same year, Dragon Systems implemented speech recognition software. These discoveries constituted a big step forward for AI. The field was becoming more promising, and governments had enough faith in it to provide sufficient funding for research. Today, no machine has yet indeed passed the Turing test. However, the AI field has evolved so much, and we came to know self-driving cars, automatic translators, and medical diagnoses with machines, which is, in a way, better than having a machine hold a real human-like conversation.
In the financial industry, AI made some impactful changes. Nevertheless, there

was a struggle to have a real breakthrough in this field. First, the notion of risk management is fundamental in finance. In banks, there are different departments to manage each type of risk : operational, credit default, and most importantly market risk. Bringing AI to the equation generates additional risk, requiring much work to maintain and supervise the new computational systems. Next, there is the challenge of integration. On the one hand, most banks use programming languages such as C++, C# or Java, and implement their libraries from scratch to control their code without depending on external libraries. On the other hand, the most developed and performing AI algorithms are implemented in Python libraries. This issue might seem irrelevant, but actually, the Python libraries used by data scientists are very sophisticated, and reproducing them from scratch in any other language is an enormous burden. The third issue is transparency. Financial institutions are subject to many regulation projects aiming at maintaining the stability and integrity of the financial system. Thus, financial institutions should explain all their models, the data used to calibrate them, and the final decisions resulting from these models. Everything should be thoroughly tested and documented. However, AI algorithms were believed to be non-transparent because we could not track how their outputs were derived. Thus, some AI algorithms have long been assimilated to black boxes, which was enough to scare researchers in finance even to try using them.
But the financial industry is forever changing. With each new traded product and each new regulation, new computations need to be done. For instance, the FRTB regulation requires each desk to compute its capital requirements daily, which results in nearly 600 000 pricings per day. The SR-11-07 regulation requires back-testing every model and every payoff used in the financial institution, computing various prices and sensitivities in different scenarios for each SR-11-07 documentation. The amount of computations is tremendous, and conventional methods could not keep up with the industry's requirements. So how can we solve the issues that come with AI to benefit from its computational power in finance ?
First, we should recognize that AI systems cannot be trained on all possible scenarios and data. In the presence of AI, human intelligence would still be needed to assess the algorithms, comprehend the AI's decisions, and eventually adapt them to the context. In regards to the issue of integration, there are two possible solutions. The first solution is to accept that building AI algorithms from scratch is a huge project that will take some time to mature. Financial engineers can then add their implementation of AI algorithms to the existing financial libraries, which will offer better maintenance and a better understanding of these algorithms. However, the process will take a long time to reach a performance equivalent to the existing Python libraries. The second solution is to create new pipelines between the financial libraries and the Python AI libraries. For instance, data can be extracted into text files, trained using Python, and the best models registered again in text files that the financial libraries can finally put to use. The Python libraries offer some features that simplify this task such as the possibility to extract Tensorflow models into Json files. This process is the simplest and most efficient on the short term. In terms of transparency and

explainability, AI does not have to be perceived as a black box. It all depends on how and where we use each algorithm. Imagine that a bank uses some neural network to price its products. Two traders sell the same product to client A and client B, respectively. Both traders use the same network to price the product, but for some reason, the network yields different prices for each of them, which is a very dangerous situation. The problem here might have resulted from different scenarios. The traders might have been using the same network architecture but not the same network because each of them relaunched the training process in his machine. The data used to train the network might have been corrupted, leading to inconsistent results. Or the model was simply being used for a non adapted problem. As the pricing formula is too complicated and usually estimated with fine-tuned algorithms, expecting a huge interpolator such as a neural network to find a magical approximation for it might be a little unrealistic. The transparency of AI algorithms depends on the way we use them and whether there is a centralised system to control their outputs or not. In conclusion, AI integration in the financial system is a real challenge, but not impossible. The integration process should take some time and effort, which is normal. After all, Rome was not built in a day.

1.2 Neural network based algorithms and application in financial mathematics

1.2.1 Feed forward neural networks

The use of neural networks is based on the following theorem.

Theorem 1.2.1 (Arnold–Kolmogorov representation theorem)**.** *There exist* $d(2d+1)$ *universal continuous functions* $\phi_{ij} : [0,1] \to [0,1]$ *such that for all continuous function* $f : [0,1]^d \to [0,1]$*, there exist* $(2d+1)$ *continuous functions* $g_i : [0,1] \to [0,1]$ *verifying*

$$f(x_1, \dots, x_d) = \sum_{i=1}^{2d+1} g_i \left(\sum_{j=1}^{d} \phi_{ij}(x_j) \right) \quad \forall (x_1, \dots, x_d) \in [0,1]^d$$

This theorem tells us that any non-linear continuous function f can be represented using $d(2d+1)$ one dimensional universal continuous functions (independent from f) and $(2d+1)$ one dimensional specific functions. A neural network allows us to find such an approximation. The building blocks for any neural network are perceptrons which also constitute the most basic type of neural networks. A perceptron is an application a_i defined as follows

$$\begin{aligned} a_i : \mathbb{R}^d &\to \mathbb{R} \\ (x_1, \dots, x_d) &\mapsto \phi_i(W_i x + b_i) \end{aligned}$$

with $W_i \in \mathcal{M}_{q,d}$ and $b_i \in \mathcal{M}_{q,1}$ and ϕ_i a continuous non linear function called the activation function applied element wise on the vector $W_i x + b_i$. The most commonly used activation functions are:

- The identity function: $\phi_i(x) = x$. (Degenerates a perceptron into a linear regression model).
- The sigmoid function: $\phi_i(x) = \frac{1}{1+\exp(-x)}$.
- The hyperbolic tangent function (tanh): $\phi_i(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$.
- The softplus function: $\phi_i(x) = \ln(1 + e^x)$.
- The Rectified Linear Unit (ReLU): $\phi_i(x) = \max(0, x)$.
- The leaky Relu: $\phi_i(x) = \max(\epsilon x, x), \epsilon \approx 0.1$

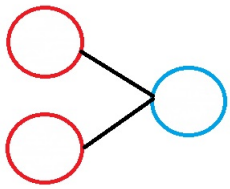


Figure 1.1: Schematic representation of a perceptron with $d = 2$ and $q = 1$

Figure 1.1, shows a schematic representation of a perceptron. It simply connects input cells to an output cell. The next type of networks is the most widely used in financial mathematics applications and is called the Feed Forward neural network or the multi layer perceptron. For $x \in \mathbb{R}^d$, a Feed Forward neural network with L layers that approximates a function with values in $\mathbb{R}^q$ writes

$$\begin{aligned} y = a_L(x) &\in \mathbb{R}^q \\ a_L(x) &= W_L a_{L-1}(x) + b_L \\ a_k(x) &= \sigma_k(W_k a_{k-1}(x) + b_k) \text{ for } k = 1, \dots, L-1 \\ a_0(x) &= x \in \mathbb{R}^d \end{aligned}$$

The network is the combination of linear transformations followed by an element wise application of a non linear activation function. The mapping a_0 is called the input layer and is simply given by the input of the function. The mapping a_L is the output layer which represents the final output of the network. The mappings $a_1, \dots, a_{L-1}$ are called the hidden layers and the dimension of the output in each layer is called the number of neurons or units in that layer. Figure 1.2 shows a schematic representation of a Feed Forward neural network.

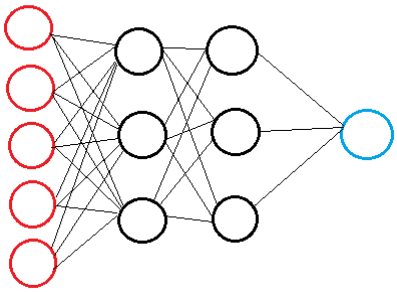


Figure 1.2: Schematic representation of a Feed Forward neural network with an input layer containing $d = 5$ neurons and an output layer with $q = 1$ neuron. The network has two hidden layers containing 3 neurons each.

1.2.2 Network optimization

1.2.2.1 Gradient descent algorithms

Let a_L be a Feed Forward neural network with parameters $\theta := (W_k, b_k)_{k=1,\dots,L}$. We wish to approximate a function f from which we have a sample data $(x_i, y_i = f(x_i))_{i=1,\dots,M}$. We define an objective function J that we wish to minimise with respect to θ. As examples of loss (objective) functions, one may consider

- The mean squared error (MSE): $J(\theta) = \frac{1}{M}\sum_{i=1}^{M}|a_L(\theta, x_i) - y_i|^2$.
- The mean absolute error (MAE): $J(\theta) = \frac{1}{M}\sum_{i=1}^{M}|a_L(\theta, x_i) - y_i|$.

In any case, we can always write the objective function as

$$J(\theta) = \frac{1}{M}\sum_{i=1}^{M} j(a_L(\theta, x_i), y_i).$$

In order to minimise the objective function, we use a gradient descent algorithm such as the Stochastic Gradient Descent (SGD). In this algorithm, we draw randomly and without replacement an index $m \in \{1, \dots, M\}$ and apply

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta j(a_L(\theta, x_m), y_m)$$

where $\eta_t > 0$ is the learning rate at time step t verifying $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$. Once the whole training set has been used, we say that we have performed one epoch of training. Practically, one may perform as many epochs of training as needed until convergence. This is particularly helpful when we use historical data or when simulating new data is too costly. Otherwise, we can alternatively generate a larger set of training data and perform only one epoch of training.
Another type of gradient descent algorithm is the batch gradient descent where the gradient is calculated on the whole data at each time step

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta J(\theta)$$

This variant can be very slow and does not allow us to update the model online (with new data on the fly). The algorithm is very efficient for convex or relatively smooth functions and allows us to move directly towards an optimum solution. For non convex functions, we are guaranteed to at least converge to a local minimum. For problems that require large data sets for training, the usage of SGD is preferable as it computes the gradient on only one sample for each time step and thus optimises the computational cost. However, unlike the batch gradient descent we will not move directly towards the optimum but rather perform small fluctuations which will allow us to jump to new and potentially better local minima. Studies show that if the learning rate is slowly decreased over time, SGD will end up converging to a local minimum for non convex problems or to the global minimum for convex problems.
The third variant, is the mini-batch gradient descent where the gradient is calculated on a small batch of data at each time step. It reduces the variance of the updates in comparison to SGD and also reduces the computational cost in comparison to batch gradient descent. It is the equilibrium of the two previous methods and thus the most preferred method when training a neural network. For a mini-batch with size M_b, the optimisation equation for the k'th batch writes

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \frac{1}{M_b} \sum_{m=kM_b+1}^{(k+1)M_b} j(a_L(\theta, x_m), y_m)$$

1.2.2.2 Adaptive gradient descent algorithms

The previous algorithms may be accelerated using an adaptive learning rate. Several algorithms allow us to perform this task such as Adagrad (Duchi et al., 2011), RMSprop (Dauphin et al., 2015) or ADAM (Kingma and Ba, 2015). These methods adapt the learning rate to the data's moments. The most recent of these algorithms is ADAM and is considered as an improvement of the two other methods. Its algorithm is given by:
For $m \in [|1, M|]$ chosen randomly and without replacement, set

$$\begin{aligned} G_t &= \nabla_\theta j\left(a_L(\theta, x_m), y_m\right) \\ m_{t+1} &= \beta_1 m_t + (1-\beta_1) G_t \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2) G_t^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^t}, \quad \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^t} \end{aligned}$$

with $m_0 = 0, \quad v_0 = 0$. The parameters $\beta_1, \beta_2, \epsilon$, and α are to be determined. The authors suggest that setting $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-8}$ and $\alpha = 10^{-3}$ yields good results. $\hat{m}_t$ represents an unbiased estimate for the first order moment of the gradient and $\hat{v}_t$ an unbiased estimate for the second order moment of the gradient. Finally the parameters of the network are updated as follows (in the context of SGD, but we can also use mini-batches)

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1} + \epsilon}}.$$

1.2.3 Back Propagation

All the previous gradient optimisation algorithms require the computation of $\nabla_\theta j(a_L(\theta, x_m), y_m)$. But, the dimension of θ depends on the network's architecture (the number of layers and the number of neurons in each layer) and can easily become huge. The computation of the gradients might then seem costly. However, the main advantage of using neural networks is that the gradients can be computed in a smart way using automatic differentiation also called in the terminology of machine learning back propagation. Once the gradients with respect to the output are computed, the rest of the gradients can be deduced as follows:

$$\tilde{y} = a_L(x), \quad \delta_i^L := \left[\nabla_{\tilde{y}} j(\tilde{y}, y)\right]_i, \quad i \in \{1, \dots, q\}$$

$$\left[\frac{\partial j}{\partial W_L}\right]_{ij} = \delta_i^L a_{L-1}^j, \quad \left[\frac{\partial j}{\partial b_L}\right]_i = \delta_i^L, \quad \left[\frac{\partial j}{\partial a_{L-1}}\right]_i = \sum_k \left[W_L^T\right]_{ik} \delta_k^L$$

$$\delta_i^{L-1} := \left[\frac{\partial j}{\partial a_{L-1}}\right]_i \left[\frac{\partial \sigma}{\partial a_{L-1}}\right]_i$$

$$\left[\frac{\partial j}{\partial W_{L-1}}\right]_{ij} = \delta_i^{L-1} a_{L-2}^j, \quad \left[\frac{\partial j}{\partial b_{L-1}}\right]_i = \delta_i^{L-1}, \quad \left[\frac{\partial j}{\partial a_{L-2}}\right]_i = \sum_k \left[W_{L-1}^T\right]_{ik} \delta_k^{L-1}, \dots$$

To summarise, the training of a neural network (using SGD for example) is done in two principal phases at each time step. A forward phase where we compute the values $\tilde{y}_i = a_L(x_i), \quad i = 1, \dots, M$. A backward phase where we compute $\nabla_\theta j(a_L(x_m), y_m)$ for some $m \in [|1, M|]$. Thanks to automatic differentiation, these two computations can be achieved in only $\mathcal{O}(\theta)$ operation.

1.2.4 Recurrent Neural Networks, GRU and LSTM

Recurrent Neural Networks (RNN) are networks that deal with time series. At each time step, in addition to the input at time t, RNNs use the output of the previous time step also as an input data. Consider an input sequence $x = (x_1, \dots, x_T)$, and a corresponding output sequence $y = (y_1, \dots, y_T)$. A RNN approaches the value of y with a sequence $z = (z_1, \dots, z_T)$ defined as follows:

$$h_t = \sigma_1(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$
$$z_t = \sigma_2(W_{hz} h_t + b_z).$$

Note that W_{hh}, W_{xh}, b_h and W_{hz} are the same for all time steps. Figure 1.3 shows a schematic representation of a recurrent neural network.

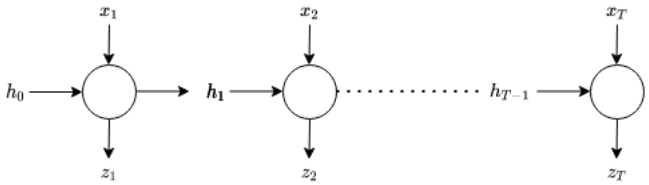


Figure 1.3: Schematic representation of a recurrent neural network

We may derive a back propagation scheme for this type of neural networks like we did for multi layer perceptrons, but during this propagation we may face a problem of vanishing gradients. This problem occurs because the multiplicative gradients cause the old time dependencies to disappear from the equation, leading to a short term memory. Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) neural networks, see (Hochreiter and Schmidhuber, 1997), improve the basic RNNs in order to deal with this vanishing gradient problem. They are characterised with gates of the form

$$\Gamma_t^k = \sigma(W_k x_t + U_k h_{t-1} + b_k)$$

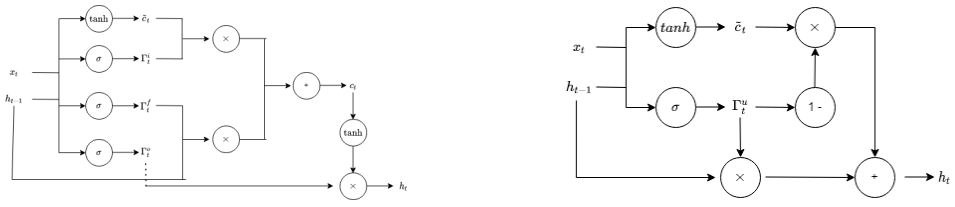
where σ here designates the sigmoid function. The different gates act as filters that allow us to choose which information is relevant and which information should be withdrawn. LSTM has three gates: the input gate Γ_t^i which decides how much of the recent information is important, the forget gate Γ_t^f which decides how much of the old information is important and the output gate Γ_t^o which decides what information is relevant in the output. The LSTM equations are given by:

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c h_{t-1} + U_c x_t + b_c) \\
c_t &= \Gamma_t^f c_{t-1} + \Gamma_t^i \tilde{c}_t \\
h_t &= \Gamma_t^o \tanh(c_t).
\end{aligned}$$

As for GRU networks, we have only one gate called the update gate Γ_t^u that decides how to update the output given the old and recent information. The GRU equations are given by:

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c h_{t-1} + U_c x_t + b_c) \\
c_t &= \Gamma_t^u c_{t-1} + (1 - \Gamma_t^u)\tilde{c}_t. \\
h_t &= c_t.
\end{aligned}$$

Figure 1.4 shows a schematic representation of GRU and LSTM neural networks.



(a) Schematic representation of a LSTM network

(b) Schematic representation of a GRU network

Figure 1.4: LSTM versus GRU

The LSTM network can be seen as a generalization of the GRU network. LSTM deals better with long term dependencies, nevertheless it is computationally very costly in comparison to GRU.

1.2.5 Generative Adversarial Networks (GAN)

Consider an unknown density μ^R in $\mathbb{R}^d$, from which we have some samples $(x_m)_{1\leq m\leq M}$. We wish to have a non parametric generative model following the law of μ^R. To do so, we choose μ^0 an initial density on $\mathbb{R}^l$ with $l \ll d$ easy to simulate (for example, one may consider uncorrelated uniform or Gaussian random variables) and we consider a mapping function $G : \mathbb{R}^l \to \mathbb{R}^d$. The push forward measure resulting from the transfer of μ^0 via G defines a probability measure on $\mathbb{R}^d$ that we denote by μ^G. As we wish to approximate the density μ^R, we will look for a mapping function G that solves

$$\inf_{G:\mathbb{R}^l\to\mathbb{R}^d} d(\mu^R, \mu^G)$$

where d is a well-chosen distance. Practically, we consider a f-divergence D_f with $f : (0, \infty) \to \mathbb{R}$ convex and verifying $f(1) = 0$. For two probability measure μ, ν on $\mathbb{R}^d$, the f-divergence is given by

$$D_f(\mu, \nu) = \mathbb{E}^\nu \left[f\left(\frac{d\mu}{d\nu}\right)\right]$$

This divergence admits a dual representation

$$D_f(\mu, \nu) = \sup_T \mathbb{E}^\mu [T(X)] - \mathbb{E}^\nu [f^*(T(Y))]$$

with $f^*(x) := \sup_u\{xu - f(u)\}$ the Legendre transform of f. In fact

$$\begin{aligned}
\sup_T \mathbb{E}^\mu [T(X)] - \mathbb{E}^\nu [f^*(T(Y))] &= \sup_T \int \nu(dx) \left(\frac{\mu(dx)}{\nu(dx)} T(x) - f^*(T(x))\right) \\
&= \int \nu(dx) \sup_{T(x)} \left(\frac{\mu(dx)}{\nu(dx)} T(x) - f^*(T(x))\right) \\
&= \int \nu(dx) f^{**}(\frac{\mu(dx)}{\nu(dx)}) \\
&= D_f(\mu, \nu)
\end{aligned}$$

since $f^{**} = f$. The problem of approaching the density μ^R can then be rewritten as

$$\inf_G \sup_T \mathbb{E}^{\mu^R} [T(X)] - \mathbb{E}^{\mu^0} [f^*(T \circ G)(Y)] .$$

We will now focus on the solution proposed in (Goodfellow et al., 2014) that relies on neural networks and called Generative Adversarial Networks (GAN). In this algorithm we choose a convex function f such that $f^*(x) = -\ln(1 - e^x)$ and we write $T(x) := \ln(D(x))$ with $D : \mathbb{R}^d \to [0, 1]$. The Generative Adversarial network problem then writes

$$\inf_G \sup_D \mathbb{E}^{\mu^R} [\ln(D(X))] + \mathbb{E}^{\mu^0} [\ln(1 - D(G(Y)))] .$$

The functions G and D are then modelled with feed forward neural networks called respectively generative model and discriminative model. We sample i.i.d

$(y_m)_{1\leq m\leq M}$ from the law of μ^0. We denote by θ_G respectively θ_D the parameters of the generative respectively discriminative model. The discrete optimization problem then writes

$$\min_{\theta_G}\max_{\theta_D}\frac{1}{M}\sum_{m=1}^{M}\ln(D_{\theta_D}(x_m)+\ln(1-D_{\theta_D}(G_{\theta_G}(y_m)))$$

The networks G and D are trained respectively using a gradient descent and a gradient ascent. (Goodfellow et al., 2014) suggest to perform one update on the generative model for every n updates in the discriminative model. (Henry-Labordere, 2019) suggest other choices for the f-divergence and also using a real distance (Wasserstein). He then draws several versions of the GAN method and tests them on some financial problems against the GAN method exposed in (Goodfellow et al., 2014). We will talk about some of these applications later.

1.2.6 Stochastic optimal control and reinforcement learning

Let $(\Omega,\mathcal{F},\mathbb{P},(\mathcal{F}_t)_{t\leq T})$ be a filtered probability space and $t_0,\dots,t_n=T$ be a discrete time grid. We consider a controlled state process given by the dynamic

$$X^\alpha_{t_{i+1}}=F(X^\alpha_{t_i},\alpha_{t_i},\epsilon_{t_{i+1}}),X^\alpha_0=x_0\in\mathbb{R}^d,$$

where $(\epsilon_{t_i})_{0\leq i\leq n}$ are i.i.d random variables with values in $\mathbb{R}$ representing the noise data such that for every i, $\epsilon_{t_{i+1}}$ is independent of $\mathcal{F}_{t_i}$, $\alpha=(\alpha_{t_i})_{0\leq i\leq n}$ is an $(\mathcal{F}_{t_i})_i-$adapted process representing the control with values in $\mathbb{R}^q$ and F a measurable function from $\mathbb{R}^d\times\mathbb{R}^q\times\mathbb{R}$ to $\mathbb{R}^d$. We define the cost function $C(\alpha)$ as follows

$$C(\alpha)=\mathbb{E}\left[\sum_{i=0}^{n}f(X^\alpha_{t_i})+g(X^\alpha_{t_n})\right]$$

where f is a running cost function and g a terminal cost function. We define the Stochastic control problem (SCP)

$$V_0(x_0):=\inf_{\alpha\in\mathbb{R}^q}C(\alpha)$$

We call V the optimal value function. We denote the transition probability from state $x\in\mathbb{R}^d$ to state $x'\in\mathbb{R}^d$ under the control $a\in\mathbb{R}^q$ by $\mathbb{P}^a(x,x')$. For any measurable function $\phi\in\mathbb{R}^d$, $\mathbb{P}^a\phi(x):=\int\phi(u)\mathbb{P}(x,du)=\mathbb{E}\left[\phi(F(x,a,\epsilon_0))\right]$. We may then rewrite the SCP as a dynamic programming equation with terminal condition $V_{t_n}(x)=g(x)\quad\forall x\in\mathbb{R}^d$ given by:

$$\begin{cases}Q_{t_i}(x,a)=f(x)+\mathbb{P}^aV_{t_{i+1}}(x),x\in\mathbb{R}^d,a\in\mathbb{R}^q,\\ V_{t_i}(x)=\inf\limits_{a\in\mathbb{R}^q}Q_{t_i}(x,a),\end{cases}$$

for $i=0,\dots,n-1$. We call Q the optimal state-action value function. This equation can be solved with various algorithms that we will see in detail in Chapter 5 for the special case of pricing Bermudan options. Here, we will focus on deep reinforcement learning solutions. (Huré et al., 2021) suggest two algorithms:

- Control learning by performance iteration: Consider at time t_k, the approximated optimal policies $\hat{a}_i, i = k+1, \dots, n$ known. We approximate the optimal policy a_k at time t_k with a neural network $\hat{a}_k = \mathcal{NN}(,;\hat{\theta}_k)$ where $\hat{\theta}_k$ is chosen to minimize the total cost from t_k to t_n.

$$\text{argmin}_\theta \, \mathbb{E}\left[f\left(X_{t_k}^{\mathcal{NN}(.;\theta)} \right) + \sum_{i=k+1}^{n} f(X_{t_i}^{\hat{a}_i}) + g(X_{t_n}^{\hat{a}_n}) \right].$$

  This algorithm is a variation of the algorithm presented in (Han and E, 2019) where the authors optimize over the policies at all time steps all at once. In case the neural networks are too big, this approach becomes very costly and may cause a vanishing or exploding gradients problem. solving step by step as suggested here comes as an improvement for this algorithm.

- Control learning by hybrid iteration This approach relies on the dynamic programming equation we exposed earlier. It is called hybrid iteration because in the first step we approximate the policy at time step t_k with a neural network $\hat{a}_k = \mathcal{NN}_1(,;\hat{\theta}_k)$ such that we solve

$$\min_\theta \mathbb{E}\left[f(X_{t_k}, \mathcal{NN}_1(,;\theta)) + V_{t_{k+1}}(X_{t_{k+1}}) \right].$$

  In the second step, we approach the value function with a neural network $\mathcal{NN}_2(.;\zeta)$ such that we minimize

$$\mathbb{E}\left[\left| f(X_{t_k}, \hat{a}_k) + V_{t_{k+1}}(X_{t_{k+1}}) - \mathcal{NN}_2(X_{t_k}, \zeta) \right|^2 \right]$$

  over ζ.

The analysis of the convergence of these two algorithms and more details can be found in (Huré et al., 2021).

1.2.7 Applications in financial mathematics

Feed Forward neural networks have several direct applications in mathematical finance. For example (McGhee, 2018), use several neural networks to calibrate the SABR stochastic volatility model. For different ranges of maturities, they consider different networks trained to estimate the implied volatility given a set of model parameters and strikes. The input parameters are generated randomly in a reasonable range and the corresponding volatilities are generated by a finite difference scheme. The network computes 10 000 times faster than the finite difference scheme, with a high level of accuracy. (Horvath et al., 2021) use deep neural networks to calibrate some rough volatility models. the calibration process is divided into two steps; first a neural network is used to approach the pricing formula, then the calibration is performed to match the neural network prices to the market prices. By doing so, the calibration process becomes very fast as the computation of the calibration products prices becomes instantaneous. Several papers use feed forward neural networks to approach the pricing formula

for some securities. For instance, (Garcia and Gençay, 2000) take inspiration from the Black and Scholes formula and divides the pricing into two parts, one controlled by S_t/K and one controlled by a function of time to maturity. They then learn the pricing map through two feed forward neural networks accordingly. (Ferguson and Green, 2018) use deep neural networks to approach the pricing formula for a call on a basket of underlyings. The method remains valid for other options. They develop a methodology for generating appropriate training data randomly and explore the impact of several parameters including layer width and depth, training data quality and quantity on model speed and accuracy. (Chataigner et al., 2020) employ neural networks to provide a pricing proxy for vanilla options and build the loss function so as to respect the no-arbitrage condition. (Becker et al., 2019) use feed forward neural networks to approach optimal stopping times with applications to Bermudan option pricing.

As times series are very important in finance, the LSTM and GRU models have been explored in multiple works to forecast the next values of a financial times series. For instance, (Siami-Namini and Namin, 2018), compare the accuracy of an LSTM model to the classical ARIMA method in forecasting financial time series. (Kwak and Lim, 2021), combine the AdaBoost algorithm and GRU model for financial time series forecasting. they compare their model to ARIMA model, single LSTM model, single GRU model, and Adaboost-LSTM ensemble for various time series. (Althelaya et al., 2018) forecast the S&P500 index historical data using deep recurrent neural networks and compare them to a simple feed forward neural network.

The generative adversarial networks also have many applications in financial mathematics. For instance, (Chen et al., 2019), use GAN models to estimate the price of individual stock returns. The criterion function is formulated by the no-arbitrage condition. (Eckerli, 2021), use GAN models to generate new financial times series. They show several applications in finance such as generation of synthetic data by JP Morgan AI Research and fraud detection by American Express AI Labs. (Pun et al., 2020), simulate various scenarios for stock variation using GAN models in order to build optimal portfolios. (Cuchiero et al., 2020), use generative adversarial networks to calibrate local stochastic volatility. They generate volatility surfaces using feed forward neural networks, and assess the quality of these surfaces by quantifying the distances to market prices through an adversarial neural network.

Several applications of reinforcement learning have been explored as well. For instance, (Buehler et al., 2019), and (Bachouch et al., 2022) formulate the hedging problem as a reinforcement problem and solve using an adapted neural network. (Henry-Labordere, 2017), solves BSDEs using deep neural networks with applications to CVA and initial margin modelling applications.

1.3 Regression trees and random forests with applications in financial mathematics

Let X be a r.v with values in $[0,1]^d$ and Y a squared integrable real-valued r.v. We want to approximate the conditional expectation $\mathbb{E}[Y|X]$. Throughout this thesis, we will consider for computational convenience that X has a density f_X in $[0,1]^d$ w.r.t the Lebesgue measure. We assume given a training sample $D_M = \{(X_1, Y_1), \dots, (X_M, Y_M) \in [0,1]^d \times \mathbb{R}\}$ where the (X_i, Y_i)'s are i.i.d random variables following the law of (X, Y). An approximation using a regression tree consists in writing the conditional expectation as a piecewise constant function of X. Each domain where the function is constant can be seen as a terminal leaf of a tree. Let us start with the one-dimensional case ($d = 1$). Consider the function

$$m_M : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \frac{1}{M} \sum_{i=1}^{M} (Y_i - y_l \mathbf{1}_{\{X_i \leq x\}} - y_r \mathbf{1}_{\{X_i > x\}})^2. \quad (1.1)$$

With probability $q > 0$, choose x as the midpoint $x = \frac{1}{2}$ and solve the minimization problem $\inf_{y_l, y_r} m_M(y_l, y_r, \frac{1}{2})$. With probability $0 < 1 - q < 1$, solve the minimization problem $\inf_{y_l, y_r, x} m_M(y_l, y_r, x)$. For a fixed x^*, the optimal values of y_l and y_r are given by

$$y_r^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i > x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i > x^*\}}}; \quad y_l^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i \leq x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i \leq x^*\}}}. \quad (1.2)$$

The problem $\inf_{y_l, y_r, x} m_M(y_l, y_r, x)$ may admit more than one minimizer. To make the tree unique, we choose the minimizer (y_l^*, y_r^*, x^*) with the minimal third component, ie any solution $(y_l^\dagger, y_r^\dagger, x^\dagger)$ to $\inf_{y_l, y_r, x} m_M(y_l, y_r, x)$ satisfies $x^\dagger \geq x^*$. Once the threshold x^* is determined, we split the samples into two groups following the sign of $X_i - x^*$ and repeat the process for each group. We stop the process if introducing a new leaf does not improve the MSE (ie when x^* is on the boundary of the interval on which we solve the minimization problem) or when enough iterations have been made. In the end, we have a tree that approximates the conditional expectation with a piecewise constant function. The regression trees are an algorithmic tool to find an adapted partition and the corresponding weights of this piecewise constant function. Note that we can also solve a global optimization problem to find such a piece wise constant function, but the computation will be costly. The regression tree does not yield a solution to such a global optimisation problem. In fact, if we are, for instance, looking for a piece wise constant function with 4 values $[0,1]$, The global optimisation problem would write

$$\min_{x_1, x_2, x_3, v_1, v_2, v_3, v_4} \frac{1}{M} \sum_{i=1}^{M} (Y_i - v_1 \mathbb{1}_{0 < X_i \leq x_1} - v_2 \mathbb{1}_{x_1 < X_i \leq x_2} - v_3 \mathbb{1}_{x_2 < X_i \leq x_3} - v_4 \mathbb{1}_{x_3 < X_i \leq 1})^2,$$

whereas the regression tree algorithm will first find an approximation with one cut which will not necessarily coincide with x_1^*, x_2^* or x_3^*. Then, it will search

for a solution with one cut on each of the new intervals iteratively. In fact, a regression tree will always perform only one cut on each iteration.
In the multi-dimensional case, we choose the direction (the index along which the optimization is performed) uniformly for each new split. Then, the process is iterated as in the one-dimensional case.

We denote the resulting tree by $\hat{\mathcal{T}}_p^M(X)$ where p represents the depth of the tree, i.e., the number of iterations done in the process described above. A tree of depth p has 2^p leaves.
When the size of the training data is infinite, we use the same procedure as above but with expectations rather than empirical means. Consider the function

$$m : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \mathbb{E}\left[(Y - y_l \mathbf{1}_{\{X \leq x\}} - y_r \mathbf{1}_{\{X > x\}})^2\right]. \tag{1.3}$$

In this case,

$$y_r^* = \mathbb{E}\left[Y | X > x^*\right]; \quad y_l^* = \mathbb{E}\left[Y | X \leq x^*\right].$$

Again, when we optimize the value of x^*, we choose the minimizer (y_l^*, y_r^*, x^*) with the minimal third component. We denote the tree of depth p obtained with an infinite data set by $\mathcal{T}_p(X)$.

A Random Forest is a collection of regression trees $\{\mathcal{T}_{p,\Theta_k}, k = 1, \ldots\}$ where the $\{\Theta_k\}$ are i.i.d random vectors. We denote the resulting forest by $\mathcal{H}_{B,p}(X) = \sum_{k=1}^{B} \frac{1}{B} \mathcal{T}_{p,\Theta_k}(X)$ where B is the number of trees in the forest and p the depth of the trees, and $\mathcal{H}_p = \mathbb{E}_\Theta\left[\mathcal{T}_{p,\Theta}\right] = \lim_{B\to\infty} \mathcal{H}_{B,p}(X)$.
The vector $(\Theta_k)_k$ will allow us to differentiate the trees in the random forest and can be chosen in different ways. For example, one can draw for each tree a sub-sample of training from the global training data without replacement (this method is called bagging and is thoroughly studied in (Breiman, 1999)). A second method is random split selection, where at each node, the split is selected at random among the K best splits, see (Dietterich, 2000). Other methods for aggregating regression trees into random forests can be found in the literature, see for example (Breiman, 2001) or (Ho, 1998).

Regression trees and random forests have some applications in financial mathematics, but they are not as widely used as neural networks in the field. (Sorensen et al., 2000) for example, use decision trees to evaluate some stocks performances based on various variables, resulting in a ranking method for stocks. (Cappelli et al., 2021) use decision trees to classify risky financial institutions into risk groups. (An and Suh, 2020), work on statement fraud detection with decision rules obtained from random forests. (Luong and Dokuchaev, 2018) use random forests to forecast the realised volatility of some financial series. (Kumar and M., 2011) use various methods to predict the direction of S&P CNX NIFTY Market Index of the National Stock Exchange movement. They show that random forests outperform neural networks and other machine learning algorithms.

1.4 Other machine learning algorithms and applications

Besides neural networks and random forests, several machine learning algorithms have been explored in the mathematical finance field. For example, the Gaussian process regression that we recall in Section 3.4 of Chapter 3 has been explored in many problems. (Crépey and Dixon, 2019) use this method to evaluate the portfolios involved in CVA computations. As modelling counterparty risk can be computationally challenging, this method makes the pricing much faster and thus improves the computational time. (Spiegeleer et al., 2018) use GPR models to construct a fast pricer for various securities. (Yu et al., 2014) employ support vector machine to construct a stock selection model. They also use principal component analysis to extract the low-dimensional and efficient feature information in order to increase the accuracy of their model. SVM is also used in (Cao and Tay, 2001) to forecast financial time series and compare it to feed forward neural networks. (Alfaro et al., 2008) predict corporate failure using the AdaBoost model. (Deng and Mei, 2009) employ K-means clustering to detect fraudulent financial statements. (Wang, 2006) study the impact of dimension reduction using principal component analysis on some high dimensional financial problems. The list of machine learning applications in finance is much bigger than what we have stated here but we have tried to give a global view on how vast the field has become and how we have overcome the fear of using machine learning in finance throughout the years.

1.5 Thesis results

In this thesis, we investigate two new applications of machine learning in mathematical finance, which led to two papers: (Ech-Chafiq et al., 2021b), published in *Monte Carlo Methods and Applications*, and (Ech-Chafiq et al., 2021a), which is currently submitted. Thus, the thesis is divided in two parts.

In the first part, we deal with the problem of pricing European options using Monte Carlo. In Chapter 3, we introduce option pricing with Monte Carlo and present some variance reduction techniques to improve its convergence. Namely, we present the technique of antithetic variates, the importance sampling algorithm, the stratified sampling algorithm and the technique of control variates. We also, discuss some state of the art algorithms that use machine learning to construct fast pricing methods. Specifically, we introduce the method of GPR models presented in (Spiegeleer et al., 2018), where the authors train Gaussian process regression models to price European options. We also, review (Horvath et al., 2021), where the authors suggest a two step approach to calibrating rough volatility models using neural networks. And finally, we talk about the deep hedging method (Buehler et al., 2019) where the authors expose a new algorithm for building hedging portfolios using neural networks. Chapter 4 corresponds to our published article (Ech-Chafiq et al., 2021b). In this chapter,

we expose two methods for creating efficient control variates for Monte Carlo methods using neural networks. The first method deals with high dimensional problems. Several financial high dimensional problems have actually low effective dimensions according to the studies in (Wang and Fang, 2003) and (Wang and Sloan, 2005). We build a network that reduces the dimension of the random variables needed to evaluate the payoff. This part can be seen as a new method of principal components analysis using neural networks. Then, we reconstruct the initial space using only these new variables and we evaluate the expectation of the control variate using a simple numerical quadrature integration. The second method consists in creating an automatic control variate using an adapted neural network. Given the random samples used to sample the model, we learn through a neural network the pricing formula (which incorporates the model, the dicretization and the payoff formula). The resulting network is naturally correlated to the payoff. We use a simple architecture for the network so that it remains easy to integrate analytically. Our numerous numerical examples, for various payoffs and models, exhibit impressive speed-ups compared to the crude Monte Carlo method. The first method has proved to be robust to parameter variations and thus can be used to efficiently compute Greeks as well. Note that, in addition to the original version of the paper, we have added Section 4.6. The second method is a practical adaptation of the Carr Madan formula stating that for any twice differentiable payoff function f, we may write

$$f(S) = f(K) + f'(K)(S-K) + \int_K^\infty f''(S-K)^+ dK + \int_{-\infty}^K f''(K-S)^+ dK.$$

This means that we can decompose the payoff in question into an infinite set of vanilla options. We tried to use a bespoke neural network to find an approximation for this decomposition. The resulting portfolio is then used as a control variate for the original payoff.

In the second part, we deal with the problem of American/Bermudan options pricing. In Chapter 5, we review the dynamic programming equation and present a survey of some related numerical methods. Namely, we review the quantization method and show how one can derive an optimal quantizer. Next, we discuss some least squares based algorithms such as the Tsitsiklis Van Roy algorithm, the Longstaff Schawrz algorithm with polynomial regression, local functions regression and neural networks regression. Then, we discuss Tree methods and Stochastic mesh methods. We also discuss some solutions to the pricing problem that do not rely on the dynamic programming equation. That is, the randomized exercise strategies method as described in (Bayer et al., 2020), and the algorithm given in (Becker et al., 2019), which approximates the optimal stopping time using neural networks. Chapter 6 corresponds to our article (Ech-Chafiq et al., 2021a), in which we studied the resolution of the dynamic programming equation using regression trees and random forests. We prove the convergence of the conditional expectations approximated using regression trees to the true value of the expectations representing the continuation values in the dynamic programming equation. We also prove that for a fixed depth of the regression trees, the

Monte Carlo approximation converges to the true price of the option. This problem was particularly hard to solve given that the construction of the regression trees cannot be formulated as a global optimisation problem as does the functional regression used in the original Longstaff Schwarz setting. This algorithm provides a practical solution to approximate any given expectation on a a set of multi dimensional piecewise constant functions. Note that in high dimension, the partition supporting the piecewise approximation cannot be regular because of the curse of dimensionality. Numerical experiments on our algorithm show that regression trees price the Bermudan options accurately, and that aggregating them into random forests gives even better results. We came to the conclusion that for small dimensional problems, a simple algorithm such as Longstaff Schwarz suffices to solve the dynamic programming equation. However, for high dimensional problems, the usage of polynomial regression becomes virtually impossible due to the curse of dimensionality, in which case it becomes interesting to use a more powerful approximating tool such as random forests.

Chapter 2

Introduction en français

2.1 Histoire de l'apprentissage automatique et le défi industriel dans le domaine de la finance

En 1950, Alan Turing aborde l'idée de créer des machines intelligentes dans son article intitulé "Computing Machinery and Intelligence" et y expose le "test de Turing", originellement dénommé "jeu de l'imitation". Il y propose une expérience permettant de tester la capacité d'une machine à mener une conversation inspirée des facultés conversationnelles humaines. évaluée par un évaluateur humain, la machine discuterait anonymement avec un autre humain . Les trois participants seraient séparés, et la conversation limitée uniquement à un format "texte". La machine réussit le test de Turing si l'évaluateur ne parvient pas à distinguer la machine de l'humain. Depuis, nombreux sont les chercheurs qui se sont attelés au défi de créer une telle machine. Il est cependant nécessaire de souligner que l'idée de créer des objets intelligents était déjà présente avant Turing. En effet, on retrouve dans la mythologie grecque des créatures mécaniques imitant le comportement humain. En Égypte, des ingénieurs ont construit des automates datant de l'âge de bronze, sans oublier les artisans chinois qui ont construit un orchestre entièrement mécanique pour leur empereur 300 ans avant Jésus-Christ. Décidément, l'humanité a toujours rêvé de donner vie aux objets inanimés, en essayant de reproduire les manifestations de l'intelligence humaine.

En 1956, les chercheurs présents au Dartmouth Summer Research Project y ont officiellement posé le socle du domaine de l'intelligence artificielle. Celui-ci ayant pour objet l'étude de machines capables d'exécuter des tâches que nous considérerions comme intelligentes. Dès lors, beaucoup de travail y a été fourni par les chercheurs dans des conditions souvent non optimales: Les ordinateurs étaient incapables de stocker suffisamment de commandes et les calculs informatiques étaient très coûteux. Par conséquent, l'IA a traversé des périodes marquées par un manque de productivité, appelées "hivers de l'IA", au cours desquelles les recherches ne progressaient pas. En 1997, IBM y a redonné un nouveau souffle par le biais de son projet "Deep Blue", la première machine à avoir battu un champion d'échecs. La même année, Dragon Systems a mis en place un logiciel de reconnaissance vocale. Ces découvertes constituent un grand pas en avant pour l'IA. Le domaine de l'IA commença à devenir de plus en plus prometteur,

ceci étant accompagné par un regain notable de la confiance des gouvernements envers cette technologie, les incitant à financer la recherche. Aujourd'hui, aucune machine n'a encore réussi le test de Turing. Certes, nous n'avons toujours pas créé la machine permettant de mener une conversation humaine en bonne et due forme, toutefois l'IA a porté ses fruits sur plusieurs plans; pour n'en citer que les voitures à conduite autonome, les traducteurs automatiques et les diagnostics médicaux effectués par des machines.

Le monde financier a su également bénéficier des bienfaits de l'IA malgré les quelques difficultés entravant la percée de celle-ci dans ce domaine. Cette difficulté prend forme sous plusieurs aspects: De prime abord, la notion de gestion des risques occupe un rôle fondamental dans la finance. Ainsi, dans les banques, il existe différents départements pour gérer chaque type de risque : opérationnel, de défaut de crédit, et surtout de marché. L'introduction de l'IA dans l'équation génère un risque supplémentaire, impliquant une charge de travail supplémentaire permettant de superviser les nouveaux systèmes informatiques. De plus, l'intégration de l'IA dans l'environnement informatique banquier constitue un réel défi. D'une part, la plupart des banques utilisent des langages de programmation tels que C++, C# ou Java, et implémentent leurs bibliothèques à partir de zéro pour contrôler leur code sans dépendre de bibliothèques externes. D'autre part, les algorithmes d'IA les plus développés et les plus performants sont implémentés dans des bibliothèques Python. Cette question peut sembler sans intérêt, mais en réalité, les bibliothèques Python utilisées par les data scientists sont très complexes, et les reproduire à partir de zéro dans d'autres langages n'est pas chose aisée. Le troisième aspect de cette difficulté porte sur la transparence. Les institutions financières sont soumises à de nombreux projets de réglementation visant à maintenir la stabilité et l'intégrité du système financier. Elles se doivent donc d'expliquer tous leurs modèles, les données utilisées pour les calibrer, ainsi que les décisions finales qui en résultent. Tout doit être minutieusement testé et documenté. Toutefois, Les algorithmes de l'intelligence artificielle ont longtemps été considérés non transparents parce qu'on ne pouvait pas expliquer comment ils construisaient leurs estimations. Ainsi, certains algorithmes de l'IA ont été assimilés à des boîtes noires, ce qui entrainait les chercheurs en finance à les éviter.

Mais l'industrie financière est en perpétuelle évolution. Dès l'intégration d'un nouveau produit ou l'apparition d'une nouvelle réglementation, de nouveaux calculs doivent être effectués. Par exemple, la réglementation FRTB oblige chaque desk à calculer quotidiennement ses besoins en capitaux, ce qui entraîne près de 600 000 pricings par jour. La réglementation SR-11-07 exige des backtests sur tout modèle et payoff utilisés dans l'institution financière, en calculant divers prix et sensibilités dans différents scénarios pour chaque documentation SR-11-07. La quantité de calculs est conséquente, et les méthodes conventionnelles ne pourraient pas répondre aux exigences du secteur. Alors, comment résoudre les problèmes liés à l'IA pour tirer parti de ses capacités de calcul dans la finance ?

Tout d'abord, on ne peut que reconnaître que les systèmes d'IA ne peuvent pas traiter tous les scénarios et données possibles. En présence d'IA, l'intelligence humaine sera toujours nécessaire pour évaluer la pertinence des

algorithmes, appréhender les décisions de l'IA et éventuellement les adapter au contexte d'utilisation. En ce qui concerne la question de l'intégration, les deux solutions suivantes pourraient être envisageables. La première consiste à accepter le temps de maturation nécessaire à la création d'algorithmes d'IA à partir de zéro. Les ingénieurs financiers pourraient donc implémenter leur propre version des algorithmes de l'IA dans les librairies internes de l'institution financière, ce qui offrira une bonne maitrise de ceux-là et facilitera leur entretien. Cependant, ce processus prendra beaucoup de temps pour atteindre la performance des algorithmes des libraires de Python. La deuxième solution consiste à créer des pipelines entre les libraires internes de la banque et les libraires de Python. On pourrait par exemple extraire les données d'entrainement dans des fichiers textes que les libraires Python utiliseraient pour entrainer leurs modèles. Ensuite, les modèles seraient de nouveau enregistrés dans des fichiers textes que la librairie financière pourrait finalement utiliser en interne. Certaines libraires python offrent déjà des fonctionalités qui facilitent ce processus tel que la librairie Tensorflow qui permet d'enregistrer les modèles de réseaux de neurones dans des fichiers Json. Sur le court terme, ce choix est le plus simple et le plus efficace. En ce qui concerne la transparence et l'expliquabilité, l'IA ne doit certainement pas être perçue comme une boîte noire. Tout dépend de comment on utilise les algorithmes de l'IA et à quels problèmes on les applique. Prenant un exemple d'une banque qui utilise un réseau de neurones pour évaluer un certain produit. Deux tradeurs vendent le même produit à deux clients A et B. Les deux tradeurs utilisent le même réseau, cependant chacun d'eux trouve un prix différent. Le problème ici aurait pu résulter de plusieurs scénarios. Les tradeurs auraient pu utiliser le même réseau en architecture, mais chacun d'eux aurait relancé le processus d'entrainement sur sa propre machine résultant en deux réseaux différents. Les données utilisées pour entraîner le réseau ont peut-être été corrompues, entraînant des résultats incohérents. Soit le modèle était simplement utilisé pour un problème non adapté. Comme la formule de pricing est trop compliquée et généralement estimée avec des algorithmes affinés, s'attendre à ce qu'un énorme interpolateur tel qu'un réseau de neurones trouve une approximation magique pourrait être un peu irréaliste. La transparence des algorithmes d'IA dépend de la façon dont nous les utilisons et de l'existence ou non d'un système centralisé pour contrôler leurs sorties. En conclusion, l'intégration de l'IA dans le système financier est un véritable défi, mais pas impossible. Le processus d'intégration devrait prendre du temps et des efforts, ce qui est normal. Après tout, Rome ne s'est pas construite en un jour.

2.2 Applications basées sur les réseaux de neurones

2.2.1 Les réseau Feed Forward

L'utilisation des réseaux de neurones est basée sur le théorème 1.2.1. Ce théorème nous dit que toute fonction continue sur un compact, non linéaire f peut être

représentée en utilisant $d(2d+1)$ fonctions continues universelles unidimensionnelles (indépendentes de f) et $(2d+1)$ fonctions spécifiques unidimensionnelles. Un réseau de neurones permet de trouver une telle approximation. Les blocs de construction de tout réseau de neurones sont des perceptrons qui constituent également le type le plus fondamental de réseaux. Un perceptron est une application a_i définie comme suit

$$\begin{aligned} a_i : \mathbb{R}^d &\to \mathbb{R} \\ (x_1, \ldots, x_d) &\mapsto \phi_i(W_i x + b_i) \end{aligned}$$

avec $W_i \in \mathcal{M}_{q,d}$ et $b_i \in \mathcal{M}_{q,1}$ et ϕ_i une fonction continue non linéaire qu'on appelle fonction d'activation appliquéé élément par élément sur le vecteur $W_i x + b_i$. Les fonctions d'activation les plus couramment utilisées sont :

- La fonction identité: $\phi_i(x) = x$. (Dégénere un perceptron en un modèle de régression linéaire).
- La fonction sigmoid: $\phi_i(x) = \frac{1}{1+\exp(-x)}$.
- La fonction tangente hyperbolique (tanh): $\phi_i(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$.
- La fonction softplus: $\phi_i(x) = \ln(1 + e^x)$.
- La fonction ReLU: $\phi_i(x) = \max(0, x)$.
- La fonction leaky Relu: $\phi_i(x) = \max(\epsilon x, x), \epsilon \approx 0.1$

La figure 1.1, montre une représentation schématique d'un perceptron. Il connecte simplement des cellules d'entrée à une cellule de sortie. Le type de réseau suivant est le plus largement utilisé dans les applications de mathématiques financières et s'appelle le réseau de neurones Feed Forward ou le perceptron multicouche. Pour $x \in \mathbb{R}^d$, un réseau de neurones Feed Forward contenant L couches approchant une fonction avec des valeurs dans $\mathbb{R}^q$ s'écrit

$$\begin{aligned} y = a_L(x) &\in \mathbb{R}^q \\ a_L(x) &= W_L a_{L-1}(x) + b_L \\ a_k(x) &= \sigma_k(W_k a_{k-1}(x) + b_k) \text{ for } k = 1, \ldots, L-1 \\ a_0(x) &= x \in \mathbb{R}^d \end{aligned}$$

Le réseau est la combinaison de transformations linéaires suivies d'une application élément par élément d'une fonction d'activation non linéaire. La fonction a_0 est appelé la couche d'entrée et est simplement donné par l'entrée de la fonction. La fonction a_L est la couche de sortie qui représente la sortie finale du réseau. Les fonctions $a_1, \ldots, a_{L-1}$ sont appelés les couches cachées et la dimension de la sortie dans chaque couche est appelée le nombre de neurones ou d'unités dans cette couche. La figure 1.2 montre une représentation schématique d'un réseau de neurones Feed Forward.

2.2.2 Optimisation du réseau

2.2.2.1 Les algorithmes de déscente de gradient

Soit a_L un réseau de neurones Feed Forward avec des paramètres $\theta := (W_k, b_k)_{k=1,\dots,L}$. Nous souhaitons estimer une fonction f à partir de laquelle nous avons un échantillon de données $(x_i, y_i = f(x_i))_{i=1,\dots,M}$. On définit une fonction objectif J que l'on souhaite minimiser par rapport à θ. Comme exemples de fonctions de perte (objectif), on peut considérer

- L'erreur moyenne quadratique (MSE) : $J(\theta) = \frac{1}{M}\sum_{i=1}^{M}|a_L(\theta, x_i) - y_i|^2$.
- L'erreur moyenne absolue(MAE) : $J(\theta) = \frac{1}{M}\sum_{i=1}^{M}|a_L(\theta, x_i) - y_i|$.

Dans tous les cas, on peut toujours écrire la fonction objectif sous la forme

$$J(\theta) = \frac{1}{M}\sum_{i=1}^{M} j(a_L(\theta, x_i), y_i).$$

Afin de minimiser la fonction objectif, nous utilisons un algorithme de descente de gradient tel que l'algorithme de descente de gradient stochastique (SGD). Dans cet algorithme, on tire aléatoirement et sans remise un indice $m \in \{1, \dots, M\}$ et on applique

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta j(a_L(\theta, x_m), y_m)$$

où $\eta_t > 0$ est le taux d'apprentissage à l'instant t vérifiant $\sum_t \eta_t = \infty$ et $\sum_t \eta_t^2 < \infty$. Une fois que l'ensemble de l'entraînement a été utilisé, nous disons que nous avons effectué une époque d'entraînement. Pratiquement, on peut effectuer autant d'époques d'entraînement que nécessaire jusqu'à converger. Ceci est particulièrement utile lorsque nous utilisons des données historiques ou lorsque la simulation de nouvelles données est trop coûteuse. Sinon, nous pouvons alternativement générer un plus grand ensemble de données d'entraînement et effectuer une seule époque d'entraînement.
Un autre type d'algorithme de descente de gradient est la descente de gradient par lot où le gradient est calculé sur l'ensemble des données à chaque pas de temps

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta J(\theta)$$

Ce variant peut être très lent et ne nous permet pas de mettre à jour le modèle en ligne (avec de nouvelles données à la volée). L'algorithme est très efficace pour les fonctions convexes ou relativement lisses et permet de se diriger directement vers une solution optimale. Pour les fonctions non convexes, on a la garantie de converger au moins vers un minimum local. Pour les problèmes qui nécessitent de grands ensembles de données, l'utilisation de SGD est préférable car il calcule le gradient sur un seul échantillon pour chaque pas de temps et optimise ainsi le coût de calcul. Cependant, contrairement à la descente de gradient par lots, nous ne dirigerons pas directement vers l'optimum mais effectuerons plutôt de petites fluctuations qui nous permettront de sauter vers de nouveaux minima locaux (potentiellement meilleurs). Des études montrent que si le taux d'apprentissage diminue lentement au fil du temps, SGD finira par converger vers un minimum

local pour les problèmes non convexes ou vers le minimum global pour les problèmes convexes.
Le troisième variant, est la descente de gradient avec mini-batchs où le gradient est calculé sur un petit lot de données à chaque pas de temps. Il réduit la variance par rapport à SGD et réduit également le coût de calcul par rapport à la descente de gradient par lots. Cet algorithmes constitue le juste milieu par rapport aux deux méthodes précédentes et il est donc le plus préférable pour entraîner les réseaux de neurones. Pour un mini-batch de taille M_b, l'équation d'optimisation pour le k'ième lot s'écrit

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \frac{1}{M_b} \sum_{m=kM_b+1}^{(k+1)M_b} j(a_L(\theta, x_m), y_m)$$

2.2.2.2 Les algorithmes adaptatifs

Les algorithmes précédents peuvent être accélérés à l'aide d'un taux d'apprentissage adaptatif. Plusieurs algorithmes nous permettent de réaliser cette tâche comme Adagrad (Duchi et al., 2011), RMSprop (Dauphin et al., 2015) ou ADAM (Kingma and Ba, 2015). Ces méthodes adaptent le taux d'apprentissage aux moments des données. Le plus récent de ces algorithmes est ADAM et est considéré comme une amélioration des deux autres méthodes. Son algorithme est donné par :
Pour $m \in [|1, M|]$ choisi aléatoirement et sans remise, posez

$$\begin{aligned} G_t &= \nabla_\theta j\left(a_L(\theta, x_m), y_m\right) \\ m_{t+1} &= \beta_1 m_t + (1-\beta_1) G_t \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2) G_t^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^t}, \quad \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^t} \end{aligned}$$

avec $m_0 = 0, \quad v_0 = 0$. Les paramètres $\beta_1, \beta_2, \epsilon$, et α sont à déterminer. Les auteurs suggèrent de choisir $\beta_1 = 0,9, \beta_2 = 0,99, \epsilon = 10^{-8}$ et $\alpha = 10^{-3}$. $\hat{m}_t$ représente une estimation non biaisée pour le moment de premier ordre du gradient et $\hat{v}_t$ une estimation non biaisée pour le moment de second ordre du gradient. Enfin les paramètres du réseau sont mis à jour comme suit (dans le cadre de SGD, mais on peut aussi utiliser des mini-batchs)

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}.$$

2.2.3 La propagation rétrograde

Tous les algorithmes d'optimisation de gradient précédents nécessitent le calcul de $\nabla_\theta j(a_L(\theta, x_m), y_m)$. Mais, la dimension de θ dépend de l'architecture du réseau (le nombre de couches et le nombre de neurones dans chaque couche) et peut facilement devenir énorme. Le calcul des gradients peut alors sembler coûteux.

Cependant, le principal avantage de l'utilisation des réseaux de neurones est que les gradients peuvent être calculés de manière intelligente en utilisant la différenciation automatique également appelée dans la terminologie du machine learning propagation rétrograde. Une fois les gradients par rapport à la sortie calculés, le reste des gradients peut être déduit comme suit :

$$\tilde{y} = a_L(x), \quad \delta_i^L := \left[\nabla_{\tilde{y}} j(\tilde{y}, y)\right]_i, \quad i \in \{1, \dots, q\}$$

$$\left[\frac{\partial j}{\partial W_L}\right]_{ij} = \delta_i^L a_{L-1}^j, \quad \left[\frac{\partial j}{\partial b_L}\right]_i = \delta_i^L, \quad \left[\frac{\partial j}{\partial a_{L-1}}\right]_i = \sum_k \left[W_L^T\right]_{ik} \delta_k^L$$

$$\delta_i^{L-1} := \left[\frac{\partial j}{\partial a_{L-1}}\right]_i \left[\frac{\partial \sigma}{\partial a_{L-1}}\right]_i$$

$$\left[\frac{\partial j}{\partial W_{L-1}}\right]_{ij} = \delta_i^{L-1} a_{L-2}^j, \quad \left[\frac{\partial j}{\partial b_{L-1}}\right]_i = \delta_i^{L-1}, \quad \left[\frac{\partial j}{\partial a_{L-2}}\right]_i = \sum_k \left[W_{L-1}^T\right]_{ik} \delta_k^{L-1}, \dots$$

Pour résumer, l'apprentissage d'un réseau de neurones (à l'aide de SGD par exemple) se fait en deux phases principales à chaque pas de temps. Une phase forward où l'on calcule les valeurs $\tilde{y}_i = a_L(x_i), \quad i = 1, \dots, M$. Une phase backward où l'on calcule $\nabla_\theta j(a_L(x_m), y_m)$ pour un certain $m \in [|1, M|]$. Grâce à la différenciation automatique, ces deux calculs peuvent être réalisés en $\mathcal{O}(\theta)$ opérations.

2.2.4 Les réseaux récurrents, GRU et LSTM

Les réseaux de neurones récurrents (RNN) sont des réseaux qui traitent des séries temporelles. À chaque pas de temps, en plus de l'entrée à l'instant t, les RNN utilisent également la sortie à l'instant précédent comme donnée d'entrée. Considérons une séquence d'entrée $x = (x_1, \dots, x_T)$ et une séquence de sortie correspondante $y = (y_1, \dots, y_T)$. Un RNN approche la valeur de y avec une séquence $z = (z_1, \dots, z_T)$ définie comme suit:

$$h_t = \sigma_1(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$
$$z_t = \sigma_2(W_{hz} h_t + b_z).$$

Notez que W_{hh}, W_{xh}, b_h et W_{hz} sont les mêmes à tous les pas de temps. La figure 1.3 montre une représentation schématique d'un réseau neuronal récurrent.
Nous pouvons dériver un schéma de propagation rétrograde pour ce type de réseaux de neurones comme nous l'avons fait pour les Feed Forward, mais au cours de cette propagation, nous pouvons être confrontés à un problème de disparition de gradients. Ce problème se produit parce que les gradients multiplicatifs font disparaître les anciennes dépendances temporelles de l'équation, conduisant à une mémoire à court terme. Les réseaux de neurones Gated Recurrent Unit (GRU) et Long Short Term Memory (LSTM), voir (Hochreiter and Schmidhuber, 1997), améliorent les RNN afin de faire face à ce problème de disparition des gradients. Ils sont caractérisés par des portes de la forme

$$\Gamma_t^k = \sigma(W_k x_t + U_k h_{t-1} + b_k)$$

où σ désigne ici la fonction sigmoïde. Les différents portails agissent comme des filtres qui nous permettent de choisir quelles informations sont pertinentes et quelles informations doivent être retirées. LSTM a trois portes: la porte d'entrée Γ_t^i qui décide de l'importance des informations récentes, la porte d'oubli Γ_t^f qui décide de l'importance des anciennes informations et la porte de sortie Γ_t^o qui décide quelles informations sont pertinentes dans la sortie. Les équations LSTM sont données par :

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c h_{t-1} + U_c x_t + b_c) \\
c_t &= \Gamma_t^f c_{t-1} + \Gamma_t^i \tilde{c}_t \\
h_t &= \Gamma_t^o \tanh(c_t).
\end{aligned}$$

Pour les réseaux GRU, nous n'avons qu'une seule porte appelée la porte de mise à jour Γ_t^u qui décide comment mettre à jour la sortie en fonction des informations anciennes et récentes. Les équations GRU sont données par :

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c h_{t-1} + U_c x_t + b_c) \\
c_t &= \Gamma_t^u c_{t-1} + (1 - \Gamma_t^u)\tilde{c}_t. \\
h_t &= c_t.
\end{aligned}$$

La figure 1.4 montre une représentation schématique des réseaux de neurones GRU et LSTM. Le réseau LSTM peut être vu comme une généralisation du réseau GRU. LSTM traite mieux les dépendances à long terme, néanmoins il est très coûteux en calcul par rapport à GRU.

2.2.5 Generative Adversarial Networks (GAN)

Considérons une densité inconnue μ^R dans $\mathbb{R}^d$, dont nous avons des échantillons $(x_m)_{1\leq m\leq M}$. Nous souhaitons avoir un modèle génératif non paramétrique suivant la loi de μ^R. Pour ce faire, on choisit μ^0 une densité initiale sur $\mathbb{R}^l$ avec $l \ll d$ facile à simuler (par exemple, on peut considérer des variables aléatoires uniformes ou gaussiennes non corrélées) et on considère une fonction de mapping $G : \mathbb{R}^l \to \mathbb{R}^d$. La mesure push forward résultant du transfert de μ^0 via G définit une mesure de probabilité sur $\mathbb{R}^d$ que nous notons μ^G. Comme on souhaite approximer la densité μ^R, on cherchera une fonction de mapping G qui résout

$$\inf_{G:\mathbb{R}^l\to\mathbb{R}^d} d(\mu^R, \mu^G)$$

où d est une distance bien choisie. Concrètement, on considère une f-divergence D_f avec $f : (0, \infty) \to \mathbb{R}$ convexe et vérifiant $f(1) = 0$. Pour deux mesures de probabilité μ, ν sur $\mathbb{R}^d$, la f-divergence est donnée par

$$D_f(\mu, \nu) = \mathbb{E}^\nu \left[f\left(\frac{d\mu}{d\nu}\right)\right]$$

Cette divergence admet une dualisation donnée par:

$$D_f(\mu, \nu) = \sup_T \mathbb{E}^\mu [T(X)] - \mathbb{E}^\nu [f^*(T(Y))]$$

avec $f^*(x) := \sup_u\{xu - f(u)\}$ La transformée de Legendre de la fonction f. En effet

$$\begin{aligned}\sup_T \mathbb{E}^\mu [T(X)] - \mathbb{E}^\nu [f^*(T(Y))] &= \sup_T \int \nu(dx) \left(\frac{\mu(dx)}{\nu(dx)} T(x) - f^*(T(x))\right) \\ &= \int \nu(dx) \sup_{T(x)} \left(\frac{\mu(dx)}{\nu(dx)} T(x) - f^*(T(x))\right) \\ &= \int \nu(dx) f^{**}(\frac{\mu(dx)}{\nu(dx)}) \\ &= D_f(\mu, \nu)\end{aligned}$$

puisque $f^{**} = f$. Le problème d'approximation de la densité μ^R peut alors être réécrit comme

$$\inf_G \sup_T \mathbb{E}^{\mu^R} [T(X)] - \mathbb{E}^{\mu^0} [f^*(T \circ G)(Y)] .$$

Nous allons maintenant nous concentrer sur la solution proposée dans (Goodfellow et al., 2014) qui s'appuie sur les réseaux de neurones et appelée Generative Adversarial Networks (GAN). Dans cet algorithme on choisit une fonction convexe f telle que $f^*(x) = -\ln(1 - e^x)$ et on écrit $T(x) := \ln(D(x))$ avec $D : \mathbb{R}^d \to [0, 1]$. Le problème de GAN s'écrit alors

$$\inf_G \sup_D \mathbb{E}^{\mu^R} [\ln(D(X))] + \mathbb{E}^{\mu^0} [\ln(1 - D(G(Y)))] .$$

Les fonctions G et D sont alors modélisées avec des réseaux de neurones feed forward appelés respectivement modèle génératif et modèle discriminatif. Nous échantillonnons i.i.d $(y_m)_{1\leq m\leq M}$ à partir de la loi de μ^0. On note θ_G respectivement θ_D les paramètres du modèle génératif respectivement discriminatif. Le problème d'optimisation discret s'écrit alors

$$\min_{\theta_G} \max_{\theta_D} \frac{1}{M} \sum_{m=1}^{M} \ln(D_{\theta_D}(x_m) + \ln(1 - D_{\theta_D}(G_{\theta_G}(y_m)))$$

Les réseaux G et D sont entraînés respectivement à l'aide d'une descente de gradient et d'une montée de gradient. (Goodfellow et al., 2014) suggère d'effectuer une mise à jour sur le modèle génératif pour chaque n mises à jour dans le modèle discriminatif. (Henry-Labordere, 2019) propose d'autres choix pour la f-divergence et également en utilisant une distance réelle (Wasserstein). Il génère ainsi plusieurs versions de la méthode GAN et les teste sur quelques problèmes financiers par rapport à la méthode GAN exposée dans (Goodfellow et al., 2014). Nous parlerons de certaines de ces applications plus loin dans ce chapitre.

2.2.6 Contrôle optimal stochastique et apprentissage par renforcement

Soit $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\leq T})$ un espace de probabilité filtré et $t_0, \ldots, t_n = T$ une grille en temps discret. On considère un processus d'état contrôlé donné par la dynamique

$$X^\alpha_{t_{i+1}} = F(X^\alpha_{t_i}, \alpha_{t_i}, \epsilon_{t_{i+1}}), X^\alpha_0 = x_0 \in \mathbb{R}^d,$$

où $(\epsilon_{t_i})_{0\leq i\leq n}$ sont des variables aléatoires i.i.d avec des valeurs dans $\mathbb{R}$ représentant les données de bruit telles que pour chaque i, $\epsilon_{t_{i+1}}$ est indépendant de $\mathcal{F}_{t_i}$, $\alpha = (\alpha_{t_i})_{0\leq i\leq n}$ est un $(\mathcal{F}_{t_i})_i-$ processus adapté représentant le contrôle avec des valeurs dans $\mathbb{R}^q$ et F une fonction mesurable de $\mathbb{R}^d \times \mathbb{R}^q \times \mathbb{R}$ à $\mathbb{R}^d$. Nous définissons la fonction de coût $C(\alpha)$ comme suit

$$C(\alpha) = \mathbb{E}\left[\sum_{i=0}^{n} f(X_{t_i}^{\alpha}) + g(X_{t_n}^{\alpha})\right]$$

où f est la fonction de coût courante et g une fonction de coût terminal. Nous définissons le problème de contrôle stochastique (SCP)

$$V_0(x_0) := \inf_{\alpha\in\mathbb{R}^q} C(\alpha)$$

Nous appelons V la fonction de valeur optimale. On note la probabilité de transition de l'état $x \in \mathbb{R}^d$ à l'état $x' \in \mathbb{R}^d$ sous le contrôle $a \in \mathbb{R}^q$ par $\mathbb{P}^a(x, x')$. Pour toute fonction mesurable $\phi \in \mathbb{R}^d$, $\mathbb{P}^a\phi(x) := \int \phi(u)\mathbb{P}(x, du) = \mathbb{E}\left[\phi(F(x, a, \epsilon_0))\right]$. On peut alors réécrire le SCP comme une équation de programmation dynamique avec la condition terminale $V_{t_n}(x) = g(x) \quad \forall x \in \mathbb{R}^d$ donnée par :

$$\begin{cases} Q_{t_i}(x, a) = f(x) + \mathbb{P}^a V_{t_{i+1}}(x), x \in \mathbb{R}^d, a \in \mathbb{R}^q, \\ V_{t_i}(x) = \inf_{a\in\mathbb{R}^q} Q_{t_i}(x, a), \end{cases}$$

pour $i = 0, \ldots, n-1$. Nous appelons Q la fonction valeur optimale état-action. Cette équation peut être résolue avec divers algorithmes que nous verrons en détail dans le chapitre 5 pour le cas particulier de pricing des options bermudiennes. Ici, nous nous concentrerons sur les solutions d'apprentissage par renforcement profond. (Huré et al., 2021) propose deux algorithmes:

- Control learning by performance iteration: Considérons à l'instant t_k, les stratégies optimales approchées $\hat{a}_i, i = k+1, \ldots, n$ connues. Nous approchons la stratégie optimale a_k à l'instant t_k avec un réseau de neurones $\hat{a}_k = \mathcal{NN}(, ; \hat{\theta}_k)$ où $\hat{\theta}_k$ est choisi pour minimiser le coût total de t_k à t_n.

  $$\text{argmin}_\theta\, \mathbb{E}\left[f\left(X_{t_k}^{\mathcal{NN}(.;\theta)}\right) + \sum_{i=k+1}^{n} f(X_{t_i}^{\hat{a}_i}) + g(X_{t_n}^{\hat{a}_n})\right].$$

  Cet algorithme est une variante de l'algorithme présenté dans (Han and E, 2019) où les auteurs optimisent toutes les stratégies à tous les instants en une seule fois. Dans le cas où les réseaux de neurones sont trop grands, cette approche devient très coûteuse et peut entraîner un problème de disparition ou d'explosion de gradients. Résoudre le problème étape par étape comme suggéré ici améliore cet algorithme.

- Control learning by hybrid iteration: Cette approche repose sur l'équation de programmation dynamique que nous avons exposée précédemment. l'appellation "itération hybride" vient du fait que dans une première étape,

nous approchons la stratégie à l'instant t_k avec un réseau de neurones $\hat{a}_k = \mathcal{NN}_1(,;\hat{\theta}_k)$ tel que nous résolvons

$$\min_{\theta} \mathbb{E}\left[f(X_{t_k}, \mathcal{NN}_1(,;\theta)) + V_{t_{k+1}}(X_{t_{k+1}})\right].$$

Dans une seconde étape, nous approchons la fonction de valeur avec un réseau de neurones $\mathcal{NN}_2(.;\zeta)$ tel que nous minimisons

$$\mathbb{E}\left[\left|f(X_{t_k}, \hat{a}_k) + V_{t_{k+1}}(X_{t_{k+1}}) - \mathcal{NN}_2(X_{t_k}, \zeta)\right|^2\right]$$

par rapport à ζ.

L'analyse de la convergence de ces deux algorithmes et plus de détails se trouvent dans (Huré et al., 2021).

2.2.7 Applications en mathématiques financières

Les réseaux de neurones Feed Forward ont plusieurs applications directes en finance mathématique. Par exemple (McGhee, 2018), utilisent plusieurs réseaux de neurones pour calibrer le modèle de volatilité stochastique SABR. Pour différents intervalles d'échéances, ils considèrent différents réseaux formés pour estimer la volatilité implicite compte tenu d'un ensemble de paramètres de modèle et de prix d'exercice. Les paramètres d'entrée sont générés aléatoirement dans une plage raisonnable et les volatilités correspondantes sont générées par un schéma de différences finies. Le réseau est 10 000 fois plus rapide que le schéma de différences finies, avec un haut niveau de précision. (Horvath et al., 2021) utilisent des réseaux de neurones profonds pour calibrer certains modèles de volatilité. le processus de calibration est divisé en deux étapes; un réseau neuronal est d'abord utilisé pour approcher la formule de pricing, puis la calibration est effectuée pour mapper les prix du réseau neuronal aux prix du marché. Ce faisant, le processus de calibration devient très rapide car le calcul des prix des produits de calibration devient instantané. Plusieurs articles utilisent des réseaux de neurones Feed Forward pour approcher la formule de pricing de certains titres. Par exemple, (Garcia and Gençay, 2000) s'inspirent de la formule de Black Scholes et divisent le pricing en deux parties, l'une contrôlée par S_t/K et l'autre contrôlée par une fonction du temps restant jusqu'à échéance. Ils apprennent ensuite la pricing map via deux réseaux de neurones. (Ferguson and Green, 2018) utilisent des réseaux de neurones profonds pour approcher la formule de pricing d'un call sur un panier de sous-jacents. La méthode reste valable pour d'autres options. Ils développent une méthodologie pour générer des données d'apprentissage appropriées de manière aléatoire et explorent l'impact de la variation de l'architecture du réseau, et de la qualité et la quantité des données d'apprentissage sur la vitesse et la précision du modèle. (Chataigner et al., 2020) utilisent des réseaux de neurones pour fournir un proxy de prix pour les options vanilles. Ils construisent une fonction de perte de manière à respecter la condition de non-arbitrage. (Becker et al., 2019) utilisent des réseaux de neurones pour approcher les temps d'arrêt optimaux avec des applications au pricing des options Bermudéenes.

Comme les séries temporelles sont très importantes en finance, les modèles LSTM et GRU ont été explorés dans plusieurs travaux pour prévoir les prochaines valeurs d'une série temporelle financière. Par exemple, (Siami-Namini and Namin, 2018), comparent la précision d'un modèle LSTM à la méthode ARIMA classique dans la prévision de séries chronologiques financières. (Kwak and Lim, 2021), combinent l'algorithme AdaBoost et le modèle GRU pour la prévision de séries chronologiques financières. ils comparent leur modèle au modèle ARIMA, au modèle LSTM, au modèle GRU et à l'ensemble Adaboost-LSTM pour diverses séries temporelles. (Althelaya et al., 2018) prévoient les données historiques de l'indice S&P500 à l'aide de réseaux de neurones récurrents profonds et les comparent à un simple réseau de neurones feed forward.
Les modèles GAN ont également de nombreuses applications en mathématiques financières. Par exemple, (Chen et al., 2019), utilisent les modèles GAN pour estimer le prix des rendements d'actions. La fonction critère est formulée par la condition de non-arbitrage. (Eckerli, 2021), utilisent des modèles GAN pour générer de nouvelles séries chronologiques financières. Ils montrent plusieurs applications en finance telles que la génération de données synthétiques par JP Morgan AI Research et la détection de fraude par American Express AI Labs. (Pun et al., 2020), simulent différents scénarios de variation de stock à l'aide de modèles GAN afin de construire des portefeuilles optimaux. (Cuchiero et al., 2020), utilisent des modèles GAN pour calibrer la volatilité stochastique locale. Ils génèrent des surfaces de volatilité à l'aide de réseaux de neurones Feed Forward et évaluent la qualité de ces surfaces en quantifiant les distances aux prix du marché par le biais d'un deuxième réseau neuronal.
Plusieurs applications de l'apprentissage par renforcement ont également été explorées. Par exemple, (Buehler et al., 2019) et (Bachouch et al., 2022) formulent le problème de couverture comme un problème de renforcement et le résolvent à l'aide d'un réseau de neurones adapté. (Henry-Labordere, 2017), résolvent des BSDE à l'aide de réseaux de neurones profonds avec des applications à la CVA et des applications de modélisation de la marge initiale.

2.3 Applications basées sur les arbres de régression et les forêts aléatoires

Soit X une variable aléatoire à valeurs dans $[0,1]^d$ et Y une variable aléatoire réélle de carré intégrable. Nous voulons estimer l'espérance conditionnelle $\mathbb{E}[Y|X]$. Nous supposons dans la suite, pour faciliter les calculs que X a une densité f_X dans $[0,1]^d$ par rapport à la mesure de Lebesgue. Nous supposons avoir un ensemble d'entraînement $D_M = \{(X_1, Y_1), \ldots, (X_M, Y_M) \in [0,1]^d \times \mathbb{R}\}$ où les (X_i, Y_i)'s sont des variables aléatoires i.i.d suivant la loi de (X, Y). Une approximation qui utilise un arbre de régression consite à écrire l'espérance conditionnnelle sous forme d'une fonction constante par morceaux de X. Chaque intervalle où la fonction est constante peut être vu comme une feuille terminale d'un arbre. Commençons par le cas mono-dimensionnel. Considérons la fonction

$$m_M : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \frac{1}{M}\sum_{i=1}^{M}(Y_i - y_l \mathbf{1}_{\{X_i \leq x\}} - y_r \mathbf{1}_{\{X_i > x\}})^2. \quad (2.1)$$

Avec probabilité $q > 0$, on choisit x comme le point au milieu de l'intervalle $x = \frac{1}{2}$ et on résout le problème de minimisation $\inf_{y_l,y_r} m_M(y_l, y_r, \frac{1}{2})$. Avec probabilité $0 < 1-q < 1$, on résout le problème de minimisation $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$. Pour un x^* fixé, les valeurs optimales de y_l et y_r sont données par

$$y_r^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i > x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i > x^*\}}}; \quad y_l^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i \leq x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i \leq x^*\}}}. \quad (2.2)$$

Le problèlme $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$ peut admettre plus qu'une solution. Pour rendre l'arbre unique, nous choisissons la solution (y_l^*, y_r^*, x^*) qui a le plus petit x^*, ie toute solution $(y_l^\dagger, y_r^\dagger, x^\dagger)$ de $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$ satisfera $x^\dagger \geq x^*$. Une fois x^* déterminé, nous séparons les données en deux groupes suivant le signe de $X_i - x^*$ et nous répétons la même procédure pour les deux sous groupes. Nous arrêtons le processus dès que l'introduction d'une nouvelle feuille n'optimise plus la MSE (ie quand x^* se trouve au bord de l'intervalle sur lequel on optimise) où quand on a déroulé suffisemment d'itérations. A la fin, nous avons un arbre qui approche l'espérance conditionnelle avec une fonction constante par morceaux. Le arbres de régression constituent un outil algorithmique pour trouver une partition adaptée et les poids correspondants de cette fonction constante par morceaux. Notez que la recherche d'une telle fonction en résolvant un problème d'optimisation globale n'est pas pareil que la recherche par arbre. En effet, si nous cherchons une fonction constante par morceaux à 4 valeurs dans [0,1], le problème d'optimisation globale s'écrit

$$\min_{x_1,x_2,x_3,v_1,v_2,v_3,v_4} \frac{1}{M}\sum_{i=1}^{M}(Y_i - v_1\mathbb{1}_{0<X_i\leq x_1} - v_2\mathbb{1}_{x_1<X_i\leq x_2} - v_3\mathbb{1}_{x_2<X_i\leq x_3} - v_4\mathbb{1}_{x_3<X_i\leq 1})^2,$$

alors que l'algorithme d'un arbre de régression trouvera d'abord une approximation à une seule coupe (qui ne coîncidera pas nécessairement avec x_1^*, x_2^* ou x_3^*). Ensuite il cherechera la solution au problème d'optimisation à une seule coupe sur les deux nouveaux intervalles de façon itérative. En fait, un arbre de régression ne résout que des problèmes à une seule coupe à chaque itération.
Dans le cas multidimensionnel, nous choisissons la direction (l'indice suivant lequel on optimise) uniformément pour chaque nouveau split. Ensuite, la procédure est répété comme dans le cas mono dimensionnel.

Nous notons l'arbre résultant par $\hat{\mathcal{T}}_p^M(X)$ où p représente la profondeur de l'arbre, i.e., le nombre d'itérations dans la procédure décrite ci-dessus. Un arbre de profondeur p a 2^p feuilles. Quand la taille des données d'entrainement est infinie nous utilisons la même procédure en remplaçant les moyennes empiriques par des espérances. Considérons la fonction

$$m : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \mathbb{E}\left[(Y - y_l \mathbf{1}_{\{X\leq x\}} - y_r \mathbf{1}_{\{X>x\}})^2\right]. \quad (2.3)$$

dans ce cas,

$$y_r^* = \mathbb{E}\left[Y|X > x^*\right]; \quad y_l^* = \mathbb{E}\left[Y|X \leq x^*\right].$$

Cette fois aussi, quand on optimise la valeur de x^*, on choisit la solution (y_l^*, y_r^*, x^*) qui a le plus petit x^*. Nous notons l'arbe de profondeur p obtenu avec un nombre infini de données par $\mathcal{T}_p(X)$.

Une forêt aléatoire est une collection d'arbres de régression $\{\mathcal{T}_{p,\Theta_k}, k = 1, \ldots\}$ où les $\{\Theta_k\}$ sont des vecteurs aléatoires i.i.d. On note la forêt résultante par $\mathcal{H}_{B,p}(X) = \sum_{k=1}^{B} \frac{1}{B} \mathcal{T}_{p,\Theta_k}(X)$ où B est le nombre d'arbres dans la forêt et p la profondeur des arbres, et $\mathcal{H}_p = \mathbb{E}_\Theta\left[\mathcal{T}_{p,\Theta}\right] = \lim_{B\to\infty} \mathcal{H}_{B,p}(X)$.
Le vecteur $(\Theta_k)_k$ nous permettra de différencier les arbres de la forêt aléatoire et peut être choisi de différentes manières. Par exemple, on peut tirer pour chaque arbre un sous-échantillon d'entraînement à partir des données globales d'entraînement sans remise (cette méthode est appelée bagging et est étudiée en détail dans (Breiman, 1999)). Une deuxième méthode est la sélection par découpage aléatoire, où à chaque nœud, le découpage est sélectionné au hasard parmi les K meilleurs découpages, voir (Dietterich, 2000). D'autres méthodes d'agrégation d'arbres de régression en forêts aléatoires peuvent être trouvées dans la littérature, voir par exemple (Breiman, 2001) ou (Ho, 1998).
Les arbres de régression et les forêts aléatoires ont certaines applications en mathématiques financières, mais ils ne sont pas aussi largement exploités que les réseaux de neurones. (Sorensen et al., 2000) par exemple, utilisent des arbres de décision pour évaluer les performances de certaines actions en fonction de diverses variables, résultant en une méthode de classement des actions. (Cappelli et al., 2021) utilisent des arbres de décision pour classifier les institutions financières en groupes de risque. (An and Suh, 2020), travaillent sur la détection de fraude avec des règles de décision obtenues à partir de forêts aléatoires. (Luong and Dokuchaev, 2018) utilisent des forêts aléatoires pour prévoir la volatilité réalisée de certaines séries financières. (Kumar and M., 2011) utilisent diverses méthodes pour prédire la direction de l'indice S&P CNX NIFTY . Ils montrent que les forêts aléatoires surpassent les réseaux de neurones et autres algorithmes d'apprentissage automatique.

2.4 Autres algorithmes d'apprentissage automatiques appliqués à la finance

Outre les réseaux de neurones et les forêts aléatoires, plusieurs algorithmes d'apprentissage automatique ont été explorés dans le domaine de la finance. Par exemple, le modèle de régression sur un processus gaussien que nous rappelons dans la section 3.4 du chapitre 3 a été exploré dans de nombreux problèmes. (Crépey and Dixon, 2019) utilisent cette méthode pour évaluer les portefeuilles impliqués dans les calculs de CVA. Comme la modélisation du risque de contrepartie peut être difficile à calculer d'un point de vue numérique, cette méthode vient améliorer les temps de calculs. (Spiegeleer et al., 2018) utilisent des modèles GPR pour construire un pricer rapide pour diverers produits. (Yu et al.,

2014) utilisent le modèle SVM pour construire un modèle de sélection de stocks. Ils utilisent également l'analyse en composantes principales pour extraire les informations les plus importantes améliorant ainsi la précision de leur modèle. SVM est également utilisé dans (Cao and Tay, 2001) pour prévoir des séries chronologiques financières et les comparer aux réseaux de neurones Feed Forward. (Alfaro et al., 2008) prédisent les faillites d'entreprises à l'aide du modèle AdaBoost. (Deng and Mei, 2009) utilisent le clustering K-means pour détecter les états financiers frauduleux. (Wang, 2006) étudient l'impact de la réduction de dimension à l'aide de l'analyse en composantes principales sur certains problèmes financiers de grande dimension. La liste des applications d'apprentissage automatique en finance est beaucoup plus longue que ce que nous avons présenté, mais nous avons essayé de donner une vision globale de l'étendue du domaine et de la manière dont nous avons surmonté la peur de l'utilisation du l'apprentissage automatique en finance au fil des ans.

2.5 Résultats de la thèse

Dans cette thèse, nous étudions deux nouvelles applications de l'apprentissage automatique en mathématiques financières, qui ont conduit à deux articles: (Ech-Chafiq et al., 2021b), publié dans *Monte Carlo Methods and Applications*, et (Ech-Chafiq et al., 2021a), qui est actuellement soumis. Ainsi, la thèse est divisée en deux parties.

Dans la première partie, nous traitons le problème de pricing des options européennes avec Monte Carlo. Au Chapitre 3, nous introduisons l'évaluation des options avec Monte Carlo et présentons quelques techniques de réduction de variance pour améliorer sa convergence. A savoir, nous présentons la technique des variables antithétiques, l'algorithme d'échantillonnage par importance, l'algorithme d'échantillonnage stratifié et la technique des variables de contrôle. Nous discutons également de certains algorithmes de pointe qui utilisent l'apprentissage automatique pour construire des méthodes de valorisation rapides. Plus précisément, nous introduisons la méthode des modèles GPR présentée dans (Spiegeleer et al., 2018), où les auteurs entraînent des modèles de régression de processus gaussien pour évaluer les options européennes. Nous examinons également (Horvath et al., 2021), où les auteurs suggèrent une approche en deux étapes pour calibrer les modèles de volatilité approximative à l'aide de réseaux de neurones. Et enfin, nous parlons de la méthode de couverture profonde (Buehler et al., 2019) où les auteurs exposent un nouvel algorithme pour construire des portefeuilles de couverture à l'aide de réseaux de neurones. Le chapitre 4 correspond à notre article publié (Ech-Chafiq et al., 2021b). Dans ce chapitre, nous exposons deux méthodes pour créer des variables de contrôle efficaces pour les méthodes de Monte Carlo à l'aide de réseaux de neurones. La première méthode traite des problèmes de grande dimension. Plusieurs problèmes financiers de grande dimension ont en fait des dimensions effectives faibles selon les études de (Wang and Fang, 2003) et (Wang and Sloan, 2005). Nous construisons un réseau qui réduit la dimension des variables aléatoires nécessaires pour évaluer

le payoff. Cette partie peut être vue comme une nouvelle méthode d'analyse en composantes principales utilisant des réseaux de neurones. Ensuite, nous reconstruisons l'espace initial en utilisant uniquement ces nouvelles variables et nous évaluons l'espérance de la variable de contrôle à l'aide d'une simple intégration en quadrature numérique. La deuxième méthode consiste à créer une variable de contrôle automatique à l'aide d'un réseau de neurones adapté. En utilisant les variables aléatoires utilisées pour générer le modèle, nous apprenons via un réseau de neurones la formule de pricing (qui intègre le modèle, la dicrétisation et la formule du payoff). Le réseau résultant est naturellement corrélé au payoff. Nous utilisons une architecture simple pour le réseau afin qu'il reste facile à intégrer analytiquement. Nos nombreux exemples numériques, pour divers payoffs et modèles, présentent des accélérations impressionnantes par rapport à la méthode brute de Monte Carlo. La première méthode s'est avérée robuste aux variations de paramètres et peut donc également être utilisée pour calculer efficacement les greques. Notez qu'en plus de la version originale de l'article, nous avons ajouté la section 4.6. La deuxième méthode est une adaptation pratique de la formule de Carr Madan indiquant que pour toute fonction payoff deux fois différentiable f, nous pouvons écrire

$$f(S) = f(K) + f'(K)(S-K) + \int_K^\infty f''(S-K)^+ dK + \int_{-\infty}^K f''(K-S)^+ dK.$$

Cela signifie que nous pouvons décomposer le payoff en question en un ensemble infini d'options vanilles. Nous avons essayé d'utiliser un réseau de neurones adapté pour trouver une approximation de cette décomposition. Le portefeuille résultant est ensuite utilisé comme variable de contrôle pour le payoff initial.

Dans la deuxième partie, nous abordons le problème de la valorisation des options américaines/bermudiennes. Dans le chapitre 5, nous passons en revue l'équation de programmation dynamique et présentons un aperçu de certaines méthodes numériques associées. A savoir, nous passons en revue la méthode de quantification et montrons comment on peut dériver un quantificateur optimal. Ensuite, nous discutons de certains algorithmes basés sur les moindres carrés tels que l'algorithme de Tsitsiklis Van Roy, l'algorithme de Longstaff Schawrz avec régression polynomiale, la régression sur des fonctions locales et la régression par réseaux de neurones. Ensuite, nous discutons des méthodes d'arbre et des méthodes de maillage stochastique. Nous discutons également de certaines solutions du problème de pricing qui ne reposent pas sur l'équation de programmation dynamique. C'est-à-dire la méthode des stratégies d'exercices aléatoires décrite dans (Bayer et al., 2020) et l'algorithme donné dans (Becker et al., 2019), qui approche le temps d'arrêt optimal à l'aide de réseaux de neurones. Le chapitre 6 correspond à notre article (Ech-Chafiq et al., 2021a), dans lequel nous avons étudié la résolution de l'équation de programmation dynamique à l'aide d'arbres de régression et de forêts aléatoires. Nous prouvons la convergence des espérances conditionnelles approchées à l'aide d'arbres de régression vers la vraie valeur des espérances représentant les valeurs de continuation dans l'équation de programmation dynamique. Nous avons également prouvé que pour une profondeur fixe

des arbres de régression, l'approximation de Monte Carlo converge vers le vrai prix de l'option. Ce problème était particulièrement difficile à résoudre étant donné que la construction des arbres de régression ne peut pas être formulée comme un problème d'optimisation globale comme le fait la régression fonctionnelle utilisée dans le cadre original de Longstaff Schwarz (voir Section 6.2.1). Cet algorithme fournit une solution pratique pour approcher toute espérance conditionnelle par un ensemble de fonctions constantes par morceaux multidimensionnelles. Notez qu'en grande dimension, la partition supportant l'approximation par morceaux ne peut pas être régulière à cause de la malédiction de la dimensionnalité. Des expériences numériques sur notre algorithme montrent que les arbres de régression évaluent avec précision les options bermudiennes, et que les agréger en forêts aléatoires donne des résultats encore meilleurs. Nous sommes arrivés à la conclusion que pour les problèmes de petite dimension, un algorithme simple tel que Longstaff Schwarz suffit pour résoudre l'équation de programmation dynamique. Cependant, pour les problèmes de grande dimension, l'utilisation de la régression polynomiale devient pratiquement impossible en raison de la malédiction de la dimensionnalité, auquel cas il devient intéressant d'utiliser un outil d'approximation plus puissant comme les forêts aléatoires.

Part I

Monte Carlo pricing for European options and variance reduction with automatic control variates

Chapter 3

Option Pricing with Monte Carlo

3.1 Introduction

In financial markets, the computation of the true price of a derivative is decisive. It represents the initial amount to invest in a self-financing portfolio to replicate the product's payoff at maturity. In a complete market, any payoff function is attainable, meaning that it can be replicated with a self-financing portfolio, called the hedging portfolio. Under the condition of completeness, there exists a unique probability measure (that we call risk-neutral) equivalent to the historical probability measure, under which any discounted price process is a local martingale. Therefore, the computation of a derivative's price boils down to the computation of some mathematical expectation. More details on option pricing and hedging can be found for example in (Guyon and Henry-Labordère, 2013) or (Kou, 1998). In small dimensions, an expectation can be approximated with simple methods such as Gaussian quadratures or a trapezoid rule. For slightly larger dimensions, one can consider the methods of quasi Monte Carlo which have small errors for relatively small dimensions (see for example (Glasserman, 2004)). However, due to time discretization and multi asset products, financial problems are often high dimensional. As a result, the most commonly used method is the Monte Carlo method. In a d-dimensional space and given M data points, a trapezoid rule error is in the order of $\mathcal{O}(M^{-\frac{2}{d}})$, whereas the Monte Carlo estimator with M samples has an error in the order of $\mathcal{O}(M^{-\frac{1}{2}})$. For a one-dimensional problem, it is then clear that the trapezoid rule is much more efficient, but as d grows the Monte Carlo estimator becomes more interesting as its convergence error remains the same independently from the problem's dimension. Using the central limit theorem, one can derive an asymptotic confidence interval for the Monte Carlo estimation which shows that adding one decimal place of precision requires multiplying the number of the Monte Carlo samples by 100. However, The convergence of the Monte Carlo estimator can be improved with variance reduction techniques without having to increase the computational resources by considering a too large number of simulations. In this chapter, we review the method of Monte Carlo and analyse its convergence. Then, we present some techniques of variance reduction: antithetic variates, importance sampling, stratification and control variates. The latter method is the core subject of Chapter 4, where we

present some automatic methods for constructing control variates using neural networks. Next, we talk about some algorithms for fast pricing using machine learning. We discuss why our usage of neural networks in Chapter 4 to construct control variates is safer than pricing directly with them.

3.2 The Monte Carlo method

Let X be a r.v with values in $\mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$. We aim at computing $\mathbb{E}[f(X)]$. First we recall the Law of large numbers and the central limit theorem, which are at the very core of Monte Carlo methods.

Theorem 3.2.1 (The law of large numbers)**.** *Let $(X_m)_{m>0}$ be a sequence of i.i.d r.v following the law of X. We suppose that $\mathbb{E}[|f(X)|] < \infty$. Then,*

$$\frac{1}{M}\sum_{i=1}^{M} f(X_i) \xrightarrow[a.s]{M\to\infty} \mathbb{E}[f(X)].$$

For the proof, refer to (Kolmogoroff, 1928).

Theorem 3.2.2 (The central limit theorem)**.** *Let $(X_m)_{m>0}$ be a sequence of i.i.d r.v following the law of X. We suppose that $\mathbb{E}[|f(X)|^2] < \infty$. Let $\sigma^2 = \mathrm{Var}(f(X))$, then*

$$\frac{\sqrt{M}}{\sigma}\left(\frac{1}{M}\sum_{i=1}^{M} f(X_i) - \mathbb{E}[f(X)]\right) \xrightarrow[\mathcal{L}]{M\to\infty} \mathcal{N}(0,1).$$

Using the law of large numbers, we can compute the expectation $\mathbb{E}[f(X)]$ using the Monte Carlo estimator

$$S_M = \frac{1}{M}\sum_{i=1}^{M} f(X_i),$$

where the X_i's are i.i.d $\sim X$ and M a sufficiently large number. As $M \to \infty$, $S_M \to \mathbb{E}[f(X)]$ a.s.
From the central limit theorem, we have

$$\frac{\sqrt{M}}{\sigma}\left(S_M - \mathbb{E}[f(X)]\right) \xrightarrow[\mathcal{L}]{M\to\infty} \mathcal{N}(0,1).$$

Note that the parameter σ is not observable. Consider an estimator of σ^2

$$\sigma_M^2 = \frac{1}{M}\sum_{i=1}^{M} f(X_i)^2 - S_M^2.$$

Since

$$\sigma_M^2 \xrightarrow[a.s]{M\to\infty} \sigma^2,$$

using Slutsky's theorem, we can prove that

$$\frac{\sqrt{M}}{\sigma_M}\left(S_M - \mathbb{E}[f(X)]\right) \xrightarrow[\mathcal{L}]{M\to\infty} \mathcal{N}(0,1).$$

Corollary 3.2.3. *Let z_δ be the $1-\delta$ quantile of the standard normal distribution, The function $x \mapsto \mathbb{1}_{|x|\leq z_\delta}$ is a.s continuous w.r.t. the normal distribution and bounded by 1, so*

$$\mathbb{E}\left[\mathbb{1}_{\frac{\sqrt{M}}{\sigma_M}|S_M - \mathbb{E}[f(X)]|\leq z_\delta}\right] \to \mathbb{E}\left[\mathbb{1}_{|Z|\leq z_\delta}\right] \quad \text{when } M \to \infty.$$

where $Z \sim \mathcal{N}(0,1)$. Then,

$$\mathbb{P}\left(\mathbb{E}\left[f(X)\right] \in \left(S_M \pm \frac{z_\delta \sigma_M}{\sqrt{M}}\right)\right) \to \mathbb{P}\left(|Z| \leq z_\delta\right) \quad \text{when } M \to \infty.$$

For example, for $\delta = 0.05, z_\delta = 1.96$, which gives a 95% confidence interval for the Monte Carlo estimator: $\left(S_M \pm \frac{1.96\sigma_M}{\sqrt{M}}\right)$.

3.3 Variance Reduction

Given the asymptotic confidence interval derived in the previous section, we notice that to increase the accuracy of the Monte Carlo estimator by a factor 10, we should multiply the number of samples M, and thus the computational time, by a factor 100. Note also that

$$\mathrm{Var}(S_M) = \mathrm{Var}\left(\frac{1}{M}\sum_{i=1}^{M} f(X_i)\right) = \frac{\sigma^2}{M}. \tag{3.1}$$

Where $\sigma^2 = \mathrm{Var}(f(X))$. Therefore, it is possible to reduce the variance and thus increase the accuracy of the Monte Carlo estimator by searching for a variable Y such that

$$\begin{cases} \mathbb{E}(Y) = \mathbb{E}(f(X)), \\ \mathrm{Var}(Y) < \mathrm{Var}(f(X)). \end{cases}$$

This can be achieved through different methods of variance reduction that we will explore in the following sections.

3.3.1 Antithetic variates

Suppose that the simulation of the Monte Carlo paths relies on some uniform i.i.d variables $(U_1, \ldots, U_M)$. We notice that if U is a uniform r.v, then so is $1 - U$. We say that U and $1 - U$ form antithetic variates. This is the case also for $Z \sim \mathcal{N}(0,1)$ and $-Z$, W a standard Brownian motion, and $-W$. The antithetic variates method consists in using the paths $(U_1, \ldots, U_M)$ as well as $(1 - U_1, \ldots, 1 - U_M)$. In practice, we consider

$$S_M = \frac{1}{M}\sum_{i=1}^{M} f(U_i).$$

$$S'_M = \frac{1}{M}\sum_{i=1}^{M} f(1 - U_i).$$

and we construct the estimator

$$\begin{aligned}\bar{S}_M &= \frac{1}{2}\left(S_{\frac{M}{2}} + S'_{\frac{M}{2}}\right) \\ &= \frac{1}{M}\sum_{i=1}^{\frac{M}{2}} f(U_i) + f(1-U_i)\end{aligned}$$

On one hand, using the law of large numbers,

$$\bar{S}_M \xrightarrow[a.s]{M\to\infty} \mathbb{E}\left[f(U)\right].$$

On the other hand,

$$\begin{aligned}\mathrm{Var}(\bar{S}_M) &= \frac{1}{2M}\mathrm{Var}(f(U) + f(1-U)) \\ &= \frac{1}{M}\mathrm{Var}(f(U)) + \frac{1}{M}\mathrm{Cov}(f(U), f(1-U)) \\ &= \mathrm{Var}(S_M) + \frac{1}{M}\mathrm{Cov}(f(U), f(1-U))\end{aligned}$$

Thus, $\mathrm{Var}(\bar{S}_M) < \mathrm{Var}(S_M)$ if and only if $\mathrm{Cov}(f(U), f(1-U)) < 0$. A sufficient condition to achieve this negative correlation between $f(U)$ and $f(1-U)$ is to have a monotonous function f. For more details on antithetic variates, refer to (Hammersley and Morton, 1956), (Fishman and Huang, 1983) or (Rubinstein et al., 1985).

3.3.2 Importance sampling

We consider that X has a density h_X in $\mathbb{R}^d$. Then

$$\mathbb{E}\left[f(X)\right] = \int_{\mathbb{R}^d} f(x)h_X(x)dx.$$

The idea of importance sampling is to change the probability from which the Monte Carlo samples are generated to put emphasis on the most important outcomes. To this end, we consider a probability density g on $\mathbb{R}^d$ different from h_X satisfying

$$h_X(x) > 0 \implies g(x) > 0,\ \forall x \in \mathbb{R}^d.$$

We can write

$$\mathbb{E}\left[f(X)\right] = \int_{\mathbb{R}^d} f(x)\frac{h_X(x)}{g(x)}g(x)dx = \mathbb{E}\left[f(Y)\frac{h_X(Y)}{g(Y)}\right]$$

with the convention $\frac{0}{0} = 1$ and where Y has density g. We draw $Y_1, \ldots, Y_M$ from the density function g. The importance sampling estimator for the quantity $\mathbb{E}\left[f(X)\right]$ is given by

$$\tilde{S}_M = \frac{1}{M}\sum_{i=1}^{M} f(Y_i)\frac{h_X(Y_i)}{g(Y_i)}.$$

Clearly, $\mathbb{E}\left[\tilde{S}_M\right] = \mathbb{E}\left[f(X)\right]$. To compare the variance of the classic Monte Carlo estimator and the importance sampling estimator, it suffices then to compare the second moments.

$$\begin{aligned}\mathbb{E}\left[\tilde{S}_M^2\right] &= \mathbb{E}\left[f^2(Y)\frac{h_X^2(Y)}{g^2(Y)}\right] \\ &= \mathbb{E}\left[f^2(X)\frac{h_X(X)}{g(X)}\right].\end{aligned}$$

If f is positive, then so is the product $h_X f$. We can normalise this product to derive a probability density function g. We write

$$g(x) = \alpha f(x) h_X(x) \forall x$$

In order for g to be a true probability density we need to ensure that

$$\int_{-\infty}^{\infty} g(u)du = 1$$

which means that

$$\alpha = \frac{1}{\int_{-\infty}^{\infty} f(u)h_X(u)du} = \frac{1}{\mathbb{E}\left[f(X)\right]}.$$

In this case, the estimator $\tilde{S}_M$ has a zero variance. However, this result is useless in practice since the quantity we wish to estimate originally is $\mathbb{E}\left[f(X)\right]$ itself. Nevertheless, it leads to a heuristic approach: choose g as close as possible to $|h_X f|$ such that the quantity $g/\int g$ is easy to compute.

Example 1. *Let $Z \sim \mathcal{N}(0,1)$, we wish to compute $\mathbb{E}\left[\mathbb{1}_{Z>25}\right]$. If we use the standard Monte Carlo method, it would take in average $M \approx 3.26 \times 10^{137}$ to get just one non null simulation. However*

$$\begin{aligned}\mathbb{E}\left[\mathbb{1}_{Z>25}\right] &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \mathbb{1}_{x>25} \exp\left(-\frac{x^2}{2}\right)dx \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \mathbb{1}_{x>25} \frac{\exp\left(-\frac{x^2}{2}\right)}{\exp\left(-\frac{(x-\mu)^2}{2}\right)} \exp\left(-\frac{(x-\mu)^2}{2}\right) \\ &= \mathbb{E}\left[\mathbb{1}_{Z^\mu>25} \exp\left(\frac{\mu^2}{2} - \mu Z^\mu\right)\right]\end{aligned}$$

where $Z^\mu \sim \mathcal{N}(\mu, 1)$. If we take $\mu = 25$ and $M = 10^7$ for example, we find an asymptotic 95% confidence interval for $\mathbb{E}\left[\mathbb{1}_{Z>25}\right]$ given by $[3.053, 3.074] \times 10^{-138}$.

3.3.3 Stratified sampling

Let Y be a r.v in $\mathbb{R}^p$ and consider $A_1, \ldots, A_K$ disjoint subsets of $\mathbb{R}^p$ s.t $\mathbb{P}\left(Y \in \bigcup_i A_i\right) = 1$. The A_i's are called strata and Y the stratification variable. Using the law of total probability,

$$\mathbb{E}\left[f(X)\right] = \sum_{i=1}^{K} \mathbb{P}\left(Y \in A_i\right) \mathbb{E}\left[f(X)/Y \in A_i\right].$$

We could choose a r.v Y that is highly correlated to X and such that the variability of Y within each stratum A_i is small. In a classic Monte Carlo simulation, the fraction of samples falling in A_i does not necessarily correspond to $p_i = \mathbb{P}(Y \in A_i)$ (Although it should converge to it as the number of Monte Carlo simulations $M \to \infty$). When using stratification, we decide in advance what fraction of the samples should be drawn from each stratum A_i. We set M the total number of Monte Carlo simulations and let $m_i = Mp_i$, that we suppose integer for simplification, be the number of simulations falling in the stratum A_i. We denote by (X_{ij}, Y_{ij}) for $i = 1, \dots, K$ and $j = 1, \dots, m_i$ the sample number j such that Y_{ij} is in the stratum A_i. The stratified Monte Carlo estimator is given by

$$\begin{aligned}\tilde{S}_M &= \sum_{i=1}^{K} p_i \frac{1}{m_i} \sum_{j=1}^{m_i} f(X_{ij}) \\ &= \frac{1}{M} \sum_{i=1}^{K} \sum_{j=1}^{m_i} f(X_{ij})\end{aligned}$$

We denote $\mu_i = \mathbb{E}[f(X_{ij})] = \mathbb{E}[f(X)/Y \in A_i] \ \forall j = 1, \dots, m_i$ and $\sigma_i^2 = \mathrm{Var}(f(X_{ij})) = \mathrm{Var}(f(X)/Y \in A_i) \ \forall j = 1, \dots, m_i$. The estimator is unbiased since

$$\begin{aligned}\mathbb{E}\left[\tilde{S}_M\right] &= \sum_{i=1}^{K} p_i \frac{1}{m_i} \sum_{j=1}^{m_i} \mathbb{E}[f(X_{ij})] \\ &= \sum_{i=1}^{K} p_i \mu_i = \mathbb{E}[f(X)].\end{aligned}$$

Its variance is given by

$$\begin{aligned}\mathrm{Var}(\tilde{S}_M) &= \sum_{i=1}^{K} p_i^2 \,\mathrm{Var}\Big(\frac{1}{m_i} \sum_{j=1}^{m_i} f(X_{ij})\Big) \\ &= \sum_{i=1}^{K} p_i^2 \frac{\sigma_i^2}{m_i} \\ &= \frac{1}{M} \sum_{i=1}^{K} p_i \sigma_i^2.\end{aligned}$$

Since the variance of the classic Monte Carlo is given by $\frac{1}{M}\sigma^2$. It suffices to prove that $\sum_{i=1}^{K} p_i \sigma_i^2 < \sigma^2$ to show that the stratification allows us to have a variance reduction. Using the conditional variance formula we have

$$\begin{aligned}\sigma^2 = \mathrm{Var}(f(X)) &= \mathbb{E}[\mathrm{Var}(f(X)/Y)] + \mathrm{Var}(\mathbb{E}[f(X)/Y]) \\ &\geq \mathbb{E}[\mathrm{Var}(f(X)/Y)] \\ &= \sum_{i=1}^{K} p_i \sigma_i^2,\end{aligned}$$

which proves that the stratification method can only reduce the variance of the Monte Carlo Estimator. We now allow the number of samples m_i in each stratum to be random and we write $q_i = \frac{m_i}{M}$. The stratified Monte Carlo estimator is then given by

$$\begin{aligned}\tilde{S}_M &= \sum_{i=1}^{K} p_i \frac{1}{m_i} \sum_{j=1}^{m_i} f(X_{ij}) \\ &= \frac{1}{M} \sum_{i=1}^{K} \frac{p_i}{q_i} \sum_{j=1}^{m_i} f(X_{ij}).\end{aligned}$$

The variance of the estimator in this case is given by

$$\begin{aligned}\mathrm{Var}(\tilde{S}_M) &= \sum_{i=1}^{K} \frac{p_i^2}{m_i} \sigma_i^2 \\ &= \frac{1}{M} \sum_{i=1}^{K} \frac{p_i^2}{q_i} \sigma_i^2 \\ &= \frac{\sigma^2(q)}{M}.\end{aligned}$$

We can solve for an optimal probability vector q such that we minimise $\sigma^2(q)$, which yields

$$q_i^* = \frac{p_i \sigma_i}{\sum_{l=1}^{K} p_l \sigma_l}.$$

In this case, $\sigma^2(q^*)$ is given by

$$\sigma^2(q^*) = \left(\sum_{i=1}^{K} p_i \sigma_i \right)^2.$$

3.3.4 Control variates

We write

$$Y = \mathbb{E}[f(X) - h(X)] + \mathbb{E}[h(X)],$$

where $\mathbb{E}[h(X)]$ can be computed explicitly. The control variate estimator is given by

$$\tilde{S}_M = \frac{1}{M} \sum_{i=1}^{M} (f(X_i) - h(X_i)) + \mathbb{E}[h(X)].$$

The estimator is clearly unbiased. The control variate variance is given by

$$\mathrm{Var}(Y) = \mathrm{Var}(f(X)) + \mathrm{Var}(h(X)) - 2\,\mathrm{Cov}(f(X), h(X)),$$

which means that the control variate is efficient if and only if

$$\mathrm{Cov}(f(X), h(X)) > \frac{1}{2} \mathrm{Var}(h(X)).$$

Thus, we need to choose h so as $h(X)$ and $f(X)$ are highly correlated and such that $\mathbb{E}[h(X)]$ can be computed exactly.

Example 2. *Consider an Asian option. We wish to compute the integral* $I := \int_0^1 e^x dx$ *which can be rewritten as* $\mathbb{E}[e^U]$ *where* $U \sim U(0,1)$. *Using Taylor's expansion of the exponential function near 0, we can affirm that* $e^x \approx 1+x$. *We suggest then to use as control variate the variable* $Y = 1+U$ *to compute the expectation of* $X = e^U$. *The expectation of* Y *is given by* $\mathbb{E}[Y] = \mathbb{E}[1+U] = \frac{3}{2}$. *We compute then the variances*

$$\begin{cases} \text{Var}(X) = \frac{1}{2}(e^2-1) - (e-1)^2 = 0.242 \\ \text{Var}(X+Y) = \frac{1}{2}e^2 - 2e + \frac{11}{6} - (e-\frac{5}{2})^2 = 0.0436. \end{cases}$$

3.4 Fast pricing with machine learning

In this section, we introduce some algorithms of fast pricing using machine learning. This will consist an introduction to the next chapter where we present our own algorithm for fast pricing using neural networks. In (Spiegeleer et al., 2018), the authors use Gaussian process regression (GPR) models to construct a fast pricer. First, we briefly review the GPR model. Let $(x_i, y_i)_{1\leq i\leq M}$ be i.i.d samples of a couple of random variables $(X,Y) \in \mathbb{R}^d \times \mathbb{R}$. We write

$$y_i = f(x_i) + \epsilon_i, \quad 1 \leq i \leq M,$$

where $f(x)$ is a Gaussian process and $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$ are i.i.d variables representing the noise in the data. The process $f(x)$ is totally defined by its covariance matrix (we can always assume that it is a centred process). We then write $f(x) \sim \mathcal{N}(0, K(X,X))$ with $K(X,X)$ the covariance matrix defined as follows

$$K(X,X) = \begin{bmatrix} k(x_1,x_1) & \dots & k(x_1,x_M) \\ k(x_2,x_1) & \dots & k(x_2,x_M) \\ \vdots & \ddots & \vdots \\ k(x_M,x_1) & \dots & k(x_M,x_M) \end{bmatrix},$$

and k is chosen to be a Gaussian kernel

$$k(x,x') = \sigma_f^2 \exp\left(\frac{-|x-x'|^2}{2l^2}\right) = \sigma_f^2 \exp\left(-\frac{\sum_{k=1}^d (x_k - x'_k)^2}{2l^2}\right)$$

where σ_f and l are the hyper-parameters of the model and need to be estimated from the training data, for instance by maximising the likelihood.
This model is then used to estimate the price of some securities as follows. The first step is to generate a training set $(x_i, y_i)_{1\leq i\leq M}$. Each x_i contains product, model and market parameters. For example, strike, barrier levels, volatility, interest rate, model specific parameters (for example, if one considers a Heston model, the parameters would be $\kappa, \rho, \theta, \eta$ and ν_0). A sample of these parameters is then generated randomly within specified ranges. Each y_i is given by the price of the option in the setting x_i computed with a given algorithm that we consider slow (Fast Fourier Transform or Monte Carlo for example). Once the GPR model

trained, it starts computing the price of the security given new product, model, or market parameters. Provided that the GPR model is well calibrated, the pricing phase should be much faster than the original pricing algorithm.
A similar approach is studied in (Horvath et al., 2021), where the authors suggest a two step algorithm for learning the pricing formula under some rough volatility models using neural networks. Formally we consider a model parametrized by a set of parameters $\theta \in \Theta$, and an option with parameters $\xi \in Z$. We denote by $P(\theta, \xi)$ the price of this option in the given model and $P(\xi)$ its market price. We introduce $\tilde{P}(\theta, \xi)$ a neural network approximation of the price $P(\theta, \xi)$. The first step consists in finding this approximation for any given parameters θ, ξ within acceptable ranges. The second step consists in calibrating the resulting network to market data. This means that we look for the parameters $\hat{\theta}$ that best approximate the market price in the network formula.

$$\hat{\theta} = \text{argmin}_{\hat{\theta} \in \Theta} \, \delta \left(\left(\tilde{P}(\theta, \xi) \right)_{\xi \in Z}, (P(\xi))_{\xi \in Z} \right)$$

Calibrating a neural network to market models is so much faster than direct calibration using classical algorithms. Once the model is calibrated, the pricing formula bounds to a forward evaluation of a neural network, which can be done instantaneously.
The last fast pricing algorithm that we present here is the one proposed in (Buehler et al., 2019). The goal in this article is to construct a hedging portfolio in the presence of market frictions such as transaction costs using a deep neural network. Consider the terminal value of a portfolio

$$PL_T(Z, p_0, \delta) := -Z + p_0 + (\delta.S)_T - C_T(\delta),$$

where Z is the liability to be hedged, p_0 the initial cash, δ is the vector of quantities to be bought of each hedging instrument and S is the vector of the hedging instruments. $C_T(\delta)$ is the total cost of trading the hedging strategy δ and is defined as follows

$$C_T(\delta) := \sum_{i=0}^{n} c_i(\delta_i - \delta_{i-1}),$$

where n is the number of rebalancing dates of the portfolio and c_i is the cost of rebalancing the portfolio at time t_i. We choose a convex risk measure ρ and define

$$\pi(X) := \inf_{\delta} \rho(X + (\delta.S)_T - C_T(\delta))$$

The idea is then to parameterise each δ_i with an adapted feed forward neural network and solve the optimization problem above with gradient descent.

Chapter 4

Automatic control variates for option pricing using neural networks

This chapter is based on a published article, (Ech-Chafiq et al., 2021b). Section 4.6 was not part of the published paper.

4.1 Introduction

Classic methods for pricing and hedging can be very slow at times. With the new regulatory context and as we seek high level performances, the demand for efficient pricing and hedging methods becomes very pressing. The question is how efficient are the methods commonly used. Depending on a number of factors such as the diffusion model or the number of underlying assets, different pricers can be considered. However, we notice that Monte Carlo is the most commonly used one. In fact, it offers a big flexibility since it can be used under any diffusion model and with as many underlying assets as needed. Moreover, its variance is immune to the dimension of the problem which makes it very suitable for high dimensional integration problems which are very common in finance. So how does the Monte Carlo method work and how do we use it in financial problems?

We consider the following SDE governing the diffusion of the underlying assets S^i

$$dS_t^i = \mu_i(t, S_t^i)dt + \sigma_i(t, S_t^i)dB_t^i. \tag{4.1}$$

where the B^i's are correlated Brownian motions and let g be a payoff function depending on the processes $(S_t^i)_{0\leq t\leq T}$. The computation of g's price often bounds to the computation of an expectation of the form $\mathbb{E}(f(X))$ which using the strong law of large numbers may be approached with the estimator $S_M = \frac{1}{M}\sum_{i=1}^{M} f(X_i)$, where the X_i's are random samples of X and M a sufficiently large number. S_M is the Monte Carlo estimator. We pose $\sigma^2 = \mathrm{Var}(f(X))$, the variance of S_M is given by

$$\mathrm{Var}(S_M) = \mathrm{Var}\left(\frac{1}{M}\sum_{i=1}^{M} f(X_i)\right) = \frac{\sigma^2}{M}. \tag{4.2}$$

Note that the variance of the estimator is proportional to the variance of the integrand. Therefore, it is always interesting to find a variable Y such that

$$\begin{cases} \mathbb{E}(Y) = \mathbb{E}(f(X)), \\ \mathrm{Var}(Y) < \mathrm{Var}(f(X)). \end{cases}$$

There are several techniques to find such a variable Y. Here, we will focus on control variates.

Let $Y = f(X) - h(X) + \mathbb{E}(h(X))$ where we assume that we can compute $\mathbb{E}(h(X))$ exactly. If

$$\begin{cases} \mathrm{Var}(Y) = \mathrm{Var}(f(X)) + \mathrm{Var}(h(X)) - 2\,\mathrm{Cov}(f(X), h(X)), \\ \mathrm{Cov}(f(X), h(X)) > \dfrac{1}{2}\,\mathrm{Var}(h(X)) \end{cases} \tag{4.3}$$

then $\mathrm{Var}(Y) < \mathrm{Var}(f(X))$.

In (Kim and Henderson, 2004) and (Kim and Henderson, 2007), the authors study control variates that depend on some parameter θ, the control variate has then the form $h(X, \theta) - \mathbb{E}\left(h(X, \theta)\right)$ and suggest two adaptive algorithms to find an adequate parameter θ^* optimising the control variate. The first one uses a stochastic approximation scheme where the parameter θ^* is chosen as to minimise the variance of the estimator and the minimisation algorithm is based on stochastic gradient descent. The second algorithm is based on a sample average algorithm where θ^* is estimated using a first random sample of X so as to minimise the variance of the estimator. Then a second sample of X is drawn independently from the first one to estimate the actual expectation. A similar work has been conducted in (Lapeyre and Lelong, 2011), the authors work on adaptive Monte Carlo which is a generalisation of this parametric control variate problem. Actually, one can write $\mathbb{E}(h(X))$ as $\mathbb{E}(H(\theta, X))$. The problem is then to find such a parametric representation of $h(X)$ and then solve for θ^*. Another issue that needs to be addressed is how to find the actual $h(X)$ that approximates $f(X)$ appropriately. In (Maire, 2003), the author suggests to use a projection of $f(X)$ on the first p terms of an orthonormal basis $(e_k)_{1 \leq k \leq p}$ in $L^2(D)$ where D is the domain of integration of f. So $h(X)$ writes $\sum_{k=1}^{p} a_k e_k(X)$. The paper works on finding an optimal estimation for the coefficients a_k. In (Glynn and Szechtman, 2002), Glynn and Szechtman show how to construct a control variate in the case where the simulation of X involves simulating $V_1, V_2, \ldots, V_\tau$ iid with $\mathbb{E}(V_j)$ known for all j and τ a stochastic variable. In (Pellizzari, 2001), the authors suppose that for a multivariate scope, if we replace all the variables but one with their means, the resulting function's expectation becomes easy to compute and it has a lot of information about the multivariate function. they then combine all these resulting functions to construct the control variate. In, (Portier and Segers, 2019), the authors study the combination of multiple control variates and the convergence limit of Monte Carlo when the number of control variates grows to infinity.

In this paper, we want to create sophisticated control variates which automatically adapt to the function f. To do so, one needs to find a function h, highly

correlated to f, and such that we have an exact method for computing $\mathbb{E}(h(X))$ or at least a numerical method that is more precise than Monte Carlo. We suggest two main methods for creating automatic control variates. The first one, deals with high dimensional problems with low effective dimensions (see (Wang and Sloan, 2005) and (Wang and Fang, 2003)). For these cases, the function f only relies on some components of X that we find using an adequate neural network and thus f can be integrated using a simple numerical integration method such as Gaussian quadratures. The idea of reducing the dimension of X before searching for the control variate has been exposed in (De Luigi et al., 2016) where the authors use Principal Component Analysis to reduce the dimension of the problem before using an adaptive integration algorithm based on quasi Monte Carlo. In the second method, we suggest to use a neural network H as the control variate. The network learns $f(X)$ so that it is naturally correlated to it (we actually learn the diffusion model, the discretization and the payoff function all at once). The architecture of the network is chosen wisely, so as $\mathbb{E}(H(X))$ can be computed easily.

4.2 On feed forward neural networks

A feed forward network is composed of a stack of layers each of which is a composition of a linear transformation with an activation function (a non linear function). The idea behind these networks comes from the Universal Approximation Theorem which states that a neural network with one hidden layer can approximate any continuous function for inputs within a specific range (see (Kůrková, 1992)). More formally, let $\psi : \mathbb{R}^p \longrightarrow \mathbb{R}^q$ (for our case $p = N$ and $q = 1$) be an unknown function that we wish to approximate. We will construct a feed forward network containing a stack of L layers as follows:

$$\begin{aligned} y = a_L(x) &\in \mathbb{R}^q \\ a_L(x) &= W_L a_{L-1}(x) + b_L \\ a_k(x) &= \sigma_a(W_k a_{k-1}(x) + b_k) \; \forall k \in [|1, L-1|] \\ a_0(x) &= x \in \mathbb{R}^p \end{aligned}$$

a_0 is the input layer, it simply contains the input of the network. $a_1, \ldots, a_{L-1}$ are hidden layers that apply (non linear) transformations to their inputs. Finally, a_L is the output layer which contains the output of the network. Each layer k is parametrized by some weights W_k and bias b_k. Training the network consists in finding the best parameters to approximate ψ.

We define a loss function $l_\theta(\psi(x), a_L(x))$ measuring the distance between the real function and the estimated values where θ represents the whole set of weights and bias. We run a minimisation algorithm on l (for example stochastic gradient descent, Adam or RMSprop) in order to find the optimal weights and biases. The optimisation problem writes

$$\min_\theta \; l_\theta(\psi(x), a_L(x)) \tag{4.4}$$

All the optimisers that we might use are first order methods, which means that we need to compute the gradient of the objective function with respect to each parameter W_k or b_k. Computing all these gradients can become very consuming. However, we use the back propagation algorithm which allows to compute only the gradients with respect to the last layer's parameters and propagate them backward to obtain all the rest of the gradients. More details on back propagation can be found in (Cilimkovic, 2010). In the following, we use the euclidean distance as a loss function, i.e $l(z, y) = ||y - z||_2$ and Adam for optimisation (more details on this algorithm can be found in (Kingma and Ba, 2015).

4.3 First Approach: Network dimension reduction

Let f be a function depending on N independent identically distributed standard normal variables $(Z_1, \dots, Z_N)$. We will search for n normal directions summarising the variance of f of the form $\tilde{Z}_i = \sum_{k=1}^{N} u_{ik} Z_k \ \forall i = 1, \dots, n$ with the condition $\sum_{k=1}^{N} u_{ik}^2 = 1 \ \forall i = 1, \dots, n$. This condition allows the directions to be standardised (i.e $\text{Var}(\tilde{Z}_i) = 1 \ \forall i = 1, \dots, n$). For this purpose, we will build a network that predicts the payoff function given as inputs the random variables $(Z_1, \dots, Z_N)$. It will be constituted of a number of hidden layers which we divide into two sections. The first section is the dimension reduction cell. It contains one hidden non activated layer (this means we only apply a linear transformation to its input) with exactly n units. We also omit the bias from this layer so as it takes the form WZ. The output of this layer should contain the $\tilde{Z}_i$'s. The second section may contain as many hidden layers as needed. Its main role is to reconstitute the payoff function given only the directions obtained in the output of the first part of the network. As the network will try to converge, the dimension reduction cell will find the variables that summarise best the variance, this way allowing the second part of the network to recover the payoff function easily. This idea is summarised in Figure 4.1

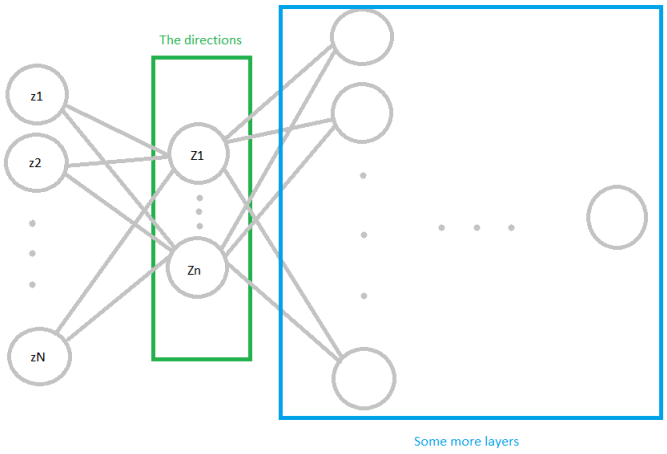


Figure 4.1: Architecture of the network used for dimension reduction

Once the network is trained, we have $\tilde{Z}_i = \sum_{k=1}^{N} u_{ik} Z_k \ \forall i = 1, \ldots, n$ with $U = (u_{ik})_{1 \leq i \leq n, 1 \leq k \leq N}$ being the normalised weight matrix of the first layer. We look for a matrix V with size $(N-n) \times N$ such that the matrix $\begin{pmatrix} U \\ V \end{pmatrix}$ has full rank N and UZ and VZ are independent. We write $\tilde{Z} = UZ$ and $\tilde{Z}^{\perp} = VZ$.The independence of $\tilde{Z}$ and $\tilde{Z}^{\perp}$ is equivalent to

$$0 = \mathrm{Cov}(\tilde{Z}, \tilde{Z}^{\perp}) = \mathrm{Cov}(UZ, VZ) = U \ \mathrm{Cov}(Z, Z) \ V^T = UV^T$$

which means that

$$\mathrm{Im}(V^T) \subseteq \ker(U). \tag{4.5}$$

In addition we have the following results

1. $\dim(\mathrm{Im}(U)) = n$ because the network will output independent directions, otherwise the effective dimension is smaller than the number of directions we impose on the network. This implies that $\dim(\ker(U)) = N - n$

2. On one hand, by adding the $\tilde{Z}^{\perp}$ variable, we wish to complete the original space, meaning that $\dim(\mathrm{Im}(V)) = N - n$ and $\dim(\ker(V)) = n$. On the other hand $\mathrm{Im}(V^T) = \ker(V)^T$ which means that $\dim(\mathrm{Im}(V^T)) = \dim(\ker(V)^T) = N - n$

$$\dim(\mathrm{Im}(V^T)) = \dim(\ker(U)) = N - n. \tag{4.6}$$

Finally, we conclude from Equations 4.5 and 4.6 that $\mathrm{Im}(V^T) = \ker(U)$. To determine V, it suffices to find a basis of $\ker(U)$. The determination of such a basis can be done using Gaussian elimination as follows
We consider the matrix U of size $n \times N$. We construct the row augmented matrix $\begin{pmatrix} U \\ I_N \end{pmatrix}$ where I_N is the $N \times N$ identity matrix. We perform Gaussian

elimination on this new matrix (column wise) and we obtain a matrix $\begin{pmatrix} A \\ B \end{pmatrix}$. A basis of $\ker(U)$ consists in the non-zero columns of B for which the corresponding column in A is a zero column. If the method is correct, this matrix should be of rank $N-n$. If not, this means that the columns of U were not independent and the number n of directions is bigger than the effective dimension of the problem. Once this is done, we simulate $\tilde{Z} \sim \mathcal{N}_n(0, UU^T)$ and $\tilde{Z}^\perp \sim \mathcal{N}_{N-n}(0, VV^T)$ since $\mathrm{Cov}(\tilde{Z}) = \mathrm{Cov}(UZ) = UU^T$ and $\mathrm{Cov}(\tilde{Z}^\perp) = \mathrm{Cov}(VZ) = VV^T$. The price of the option is given by $\mathbb{E}(f(Z))$ and since

$$\begin{aligned}
\mathbb{E}\left(f(Z)\right) &= \mathbb{E}\left(\mathbb{E}\left(f(Z)/\tilde{Z}\right)\right) \\
&= \mathbb{E}\left(f\left(\mathbb{E}\left(Z/\tilde{Z}\right)\right)\right) + \mathbb{E}\left(\mathbb{E}\left(f(Z)/\tilde{Z}\right) - f\left(\mathbb{E}\left(Z/\tilde{Z}\right)\right)\right) \\
&= \mathbb{E}\left(f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\0\end{pmatrix}\right)\right) + \mathbb{E}\left(f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\\tilde{Z}^\perp\end{pmatrix}\right) - f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\0\end{pmatrix}\right)\right)
\end{aligned} \tag{4.7}$$

We will use the variable $f(\mathbb{E}(Z/\tilde{Z}))$ as a control variate. Since it depends only on the new variables $\tilde{Z}$ of small dimension, we can use a numerical integrator much faster and more precise than Monte Carlo. Also we can use Sobol Sequences for $\tilde{Z}$ simulation to have better convergence results (for more details on Sobol Sequences please refer to (Sobol, 2001)). $\tilde{Z}^\perp$ is simulated using standard normal random variables.

4.4 Second Approach: Automatic control variate

Let $H : \mathbb{R}^N \longrightarrow \mathbb{R}$ be a neural network that approximates the payoff function f given the normal variable $Z = (Z_1, \ldots, Z_N)$. We write

$$Y = f(Z) - H(Z) + \mathbb{E}(H(Z))$$

If the network is well trained, $H(Z)$ is highly correlated to $f(Z)$ and we just need to know how to compute $\mathbb{E}(H(Z))$ analytically in order to use it as a control variate. We suggest two ideas for doing so.

4.4.1 Numerical integration

For this part, we stick to the network architecture given in Figure 4.1. We may write

$$H(Z) = (\tilde{H}\ o\ a_1)(Z)$$

where $a_1(Z) = WZ$ is the first hidden layer of the network (the green block in Figure 4.1) and $\tilde{H}$ represents the rest of the layers (the blue block in Figure 4.1).

$$\begin{aligned}
\mathbb{E}(H(Z)) &= \mathbb{E}((\tilde{H}\ o\ a_1)(Z)) \\
&= \mathbb{E}(\tilde{H}(\tilde{Z}))
\end{aligned}$$

where $\tilde{Z} = a_1(Z) = WZ \sim \mathcal{N}(0, WW^T)$. Since $\tilde{Z}$ has a small dimension, the integral can be estimated numerically using Gauss Hermite polynomials for example. Of course, this should only be used when the effective dimension is small enough.

4.4.2 Analytic integration

Here, we suggest a network having a simpler form, which will allow us to integrate it with an exact formula. This will make the computation much faster since the integration of the whole network can be numerically costly. Moreover, the network we suggest is capable of dealing with any payoff function with no restriction on the number of neurons of the first layer. The architecture of this network is described in Figure 4.2

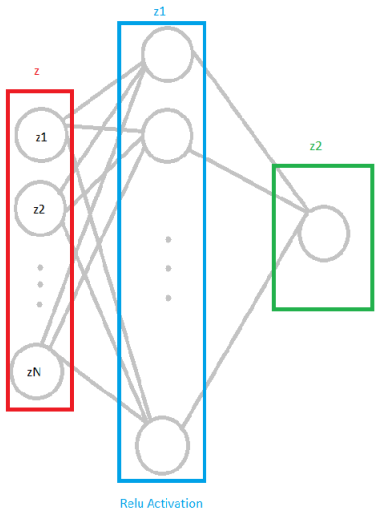


Figure 4.2: Architecture for automatic control variates

We consider the Relu activation function

$$Relu(x) = \max(0, x) = x_+$$

Then, the network H writes

$$\begin{aligned} H(Z) &= Z_2 \\ &= W_2 Z_1 + b_2 \\ &= W_2 (W_1 Z + b_1)^+ + b_2 \end{aligned}$$

Thus,

$$\mathbb{E}(H(Z)) = W_2 \mathbb{E}((W_1 Z + b_1)^+) + b_2$$

But

$$W_1 Z + b_1 = \begin{pmatrix} \sum_{j=1}^N (W_1)_{1j} Z_j + (b_1)_1 \\ \vdots \\ \sum_{j=1}^N (W_1)_{nj} Z_j + (b_1)_n \end{pmatrix} = \begin{pmatrix} \sigma_1 Y_1 + \mu_1 \\ \vdots \\ \sigma_n Y_n + \mu_n \end{pmatrix}$$

with $\mu_i = (b_1)_i$ and $\sigma_i^2 = \sum_{j=1}^N (W_1)_{ij}^2$ and $Y_i \sim \mathcal{N}(0,1)$. Then, we just have to compute $\mathbb{E}((\sigma Y + \mu)^+)$. In fact

$$\begin{aligned}
\mathbb{E}((\sigma Y + \mu)^+) &= \sigma \int_{-\frac{\mu}{\sigma}}^{+\infty} y f_{\mathcal{N}(0,1)}(y) dy + \mu \mathbb{P}\left(Y \geq -\frac{\mu}{\sigma}\right) \\
&= \sigma \int_{-\frac{\mu}{\sigma}}^{+\infty} \frac{1}{\sqrt{2\pi}} y \exp\left(-\frac{1}{2}y^2\right) dy + \mu\left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right) \\
&= \frac{\sigma}{\sqrt{2\pi}} \left[-\exp(-\frac{1}{2}y^2)\right]_{-\frac{\mu}{\sigma}}^{+\infty} + \mu\left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right) \\
&= \frac{\sigma}{2\pi} \exp\left(-\frac{1}{2}\frac{\mu^2}{\sigma^2}\right) + \mu\left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right),
\end{aligned}$$

where $\mathcal{N}$ is the cumulative distribution function of the standard normal distribution. Finally,

$$\mathbb{E}(H(Z)) = W_2 \begin{pmatrix} \cdots \\ \frac{\sigma_i}{2\pi} \exp\left(-\frac{1}{2}\frac{\mu_i^2}{\sigma_i^2}\right) + \mu_i\left(1 - \mathcal{N}\left(-\frac{\mu_i}{\sigma_i}\right)\right) \\ \cdots \end{pmatrix} + b_2. \tag{4.8}$$

with $\mu_i = (b_1)_i$ and $\sigma_i^2 = \sum_{j=1}^N (W_1)_{ij}^2$

4.5 Main numerical results

4.5.1 Black & Scholes

Let us consider the Black and Scholes model.

$$\begin{aligned}
dS_t^i &= rS_t^i dt + \sigma_i S_t^i dB_t^i \\
d(B_t^i B_t^j) &= \rho_{ij} dt
\end{aligned}$$

This is a relatively simple model, where each underlying volatility is constant. We will first test our control variates on this model for different payoffs.

- a call on a basket $\max(0.0, \sum_i \omega_i S_T^i - K)$
- a put on a worst of $\max(0.0, K - \min_i(S_T^i))$
- an arithmetic Asian option $\max(0.0, \frac{1}{d}\sum_i \sum_j \omega_j S_{t_i}^j - K)$
- a binary option (a digit) on the basket $G\mathbb{1}_{\sum_i \omega_i S_T^i \geq K}$

We use the following parameters
$S_0^i = 100.0\ \forall i, K = 100.0, \omega_i = \frac{1}{nbsj}\ \forall i, T = 1$ year, $d = 10, G = 100.0, \sigma_i = 0.4\ \forall i, \rho_{ij} = 0.75\ \forall i, j, nbsj = 10$.

4.5.1.1 Network dimension reduction

In this first experiment we will compare the two methods that use dimension reduction presented in Section 4.3 (method 1) and Section 4.4.1 (method 2). Table 4.1 shows the price given by Monte Carlo as well as the ones given by the networks for the different payoffs. We define Cost as the sum of the training time and the price computation time using the network divided by the Monte Carlo computation time. The speed up represents the ratio between the variances of the standard Monte Carlo and the one with the control variate divided by Cost

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	15.95	9.75	46.00	26.48
Method 1 price	16.30	9.86	49.51	26.62
Method 1 Var ratio	336.12	166.79	15.28	4.08
Method 1 Cost	24.23	10.22	19.34	23.76
Method 1 Speed up	13.87	16.31	0.79	0.17
Method 2 price	16.19	9.86	47.50	26.55
Method 2 Var ratio	266.93	379.32	35.98	11.44
Method 2 Cost	14.32	3.34	14.97	14.23
Method 2 Speed up	18.64	113.46	2.43	0.80

Table 4.1: Dimension reduction control variates performances under Black and Scholes

We can see that the control variate is effective for most of the payoffs. Nevertheless, the first method is much more efficient when the function to be integrated is close to being linear (the first term in 4.7 converges to the actual expectation and we do not even need the bias correction represented by the second term of 4.7). This is the case for the Basket or the Asian option. In the other cases, the methods still give good control variates and we can see that we have a significant variance reduction in most cases. Note that the Worst Of and the digit are better approximated with the second method since the first one will only capture the linear part of the function whilst the second method will learn the whole payoff function.

4.5.1.2 Automatic control variates

In this section we study the same payoffs described earlier, we test the automatic control variates described in Section 4.4.2.

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	15.86	9.83	46.57	26.68
Control variate	16.22	9.81	46.95	26.64
Var ratio	673.90	119.54	16.07	21.74
Cost	13.82	4.82	12.98	9.75
Speed up	48.76	24.80	1.23	2.23

Table 4.2: Automatic control variates performances under Black and Scholes

Again, the control variate allows significant speed ups for the different payoffs. We also notice that for the Digit and Worst Of payoffs the speeds are better than when we use the method of dimension reduction and we learn only the linear part of the functions.

4.5.2 Local Volatility model

We consider a local volatility model where we consider the following form for the volatility function

$$\sigma(t,x) = 0.6(1.2 - e^{-0.1t}e^{-0.001(xe^{rt}-s)^2})e^{-0.05\sqrt{t}}$$

This parametric volatility function is taken from (Kebaier and Lelong, 2018), where the authors advise to take s equal to the spot price of the underlying asset in order for the formula to make sense. This will allow the bottom of the smile to be located at the forward money. There is no exact simulation method for this model as in the Black Scholes model, thus we need a discretization scheme. We use an Euler discretization with 100 steps per year. The results are displayed in Table 4.3.

4.5.2.1 Network dimension reduction

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	7.91	4.33	60.41	8.56
Method 1 Price	7.92	4.37	61.75	8.56
Method 1 Var ratio	36.58	76.48	2.81	2.68
Method 1 Cost	8.83	3.02	4.89	4.71
Method 1 Speed up	4.14	25.32	0.57	0.56
Method 2 price	7.84	4.36	61.77	8.50
Method 2 Var ratio	48.59	78.95	3.33	8.18
Method 2 Cost	1.21	1.24	1.50	1.21
Method 2 Speed up	40.15	63.67	2.22	6.76

Table 4.3: Dimension reduction control variates performances under a local volatility model

Here, we have introduced two levels of complexity. First, the model is more complicated than the Black and Scholes model. Second, the introduction of the time grid increases the inputs for the networks. The first method's speed ups decrease a little but are still acceptable. The second method deals better with this complexity and we even have some improvements in the speed ups.

4.5.2.2 Automatic control variates

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	7.90	4.36	60.49	8.54
Control variate	7.85	4.36	60.38	8.49
Var ratio	8.29	9.77	2.06	5.59
Cost	1.85	1.75	2.62	1.85
Speed up	4.48	5.58	0.78	3.02

Table 4.4: Automatic control variates performances under a local volatility model

The automatic control variate struggles a little bit more with this model than the control variate with numerical integration. This might be due to the fact that we only use one hidden layer in the network and since this model is more complicated we think that one hidden layer is not really sufficient to capture its complexity. However, we still have some interesting speed ups which proves that no matter how complicated the payoff, the model or the time grid can get, the network will still make the best of what you feed to it to give interesting results.

4.5.3 Stochastic volatility model

We consider the Heston model for the diffusion of the assets:

$$\begin{aligned}
dS_t^i &= rS_t^i dt + \sqrt{\sigma_t^i} S_t^i dB_t^i \\
d\sigma_t^i &= \kappa_i(a_i - \sigma_t^i)dt + \nu_i\sqrt{\sigma_t^i}(\gamma_i dB_t^i + \sqrt{1-\gamma_i^2}d\tilde{B}_t^i) \\
d\langle B\rangle_t &= \Gamma dt,\ d\langle \tilde{B}\rangle_t = I_N dt
\end{aligned}$$

Γ is the correlation matrix between the different assets, I_N is the $N \times N$ identity matrix, κ is the reversion rate of the volatility process, a is the mean level of the volatility process, ν is the volatility of the volatility process and γ_i is the correlation between an asset and its volatility process which we consider constant for the sake of simplicity. the model can be equivalently written

$$\begin{aligned}
dS_t^i &= rS_t^i dt + \sqrt{\sigma_t^i} S_t^i dB_t^i \\
d\sigma_t^i &= \kappa_i(a_i - \sigma_t^i)dt + \nu_i\sqrt{\sigma_t^i}d\hat{B}_t^i
\end{aligned}$$

Where B and $\hat{B}$ are Wiener processes satisfying

$$d\langle B\rangle_t = \Gamma dt,\ d\langle B, \hat{B}\rangle_t = \gamma\Gamma dt,\ d\langle \hat{B}\rangle_t = \left(\gamma^2\Gamma + (1-\gamma^2)I_N\right)dt$$

The process $(B, \hat{B})$ with values in $\mathbb{R}^{2N}$ is a Wiener process with covariance

$$\tilde{\Gamma} = \begin{pmatrix} \Gamma & \gamma\Gamma \\ \gamma\Gamma & \gamma^2\Gamma + (1-\gamma^2)I_N \end{pmatrix}$$

Hence, the pair of processes $(B, \hat{B})$ can be easily simulated by applying the Cholesky factorization of $\tilde{\Gamma}$ to a standard Brownian motion with values in $\mathbb{R}^{2N}$.
We use the following model parameters
$\forall i\ \kappa_i = 2, \sigma_i(0) = 0.04, a_i = 0.04, \nu_i = 0.01, \gamma_i = -0.2$

4.5.3.1 Network dimension reduction

We test the Heston model with the payoffs described earlier. The results are shown in Table 4.5

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	9.69	5.76	56.92	12.40
Method 1	9.49	5.58	56.37	12.43
Method 1 Var ratio	20.91	32.83	1.00	2.54
Method 1 Cost	13.85	13.54	11.70	13.82
Method 1 Speed up	1.51	2.42	0.08	0.18
Method 2 price	9.57	5.92	55.19	12.58
Method 2 Var ratio	31.03	502.49	1.63	5.44
Method 2 Cost	1.28	1.27	1.36	1.35
Method 2 Speed up	24.24	395.66	1.19	4.03

Table 4.5: Dimension reduction control variates performances under a stochastic volatility model

We added again more complexity to the model and also we doubled the size of the networks inputs. However, the results haven't been degraded in comparison to the local volatility model. This means that the performance of the networks isn't really a decreasing function of the model's complexity. Surprisingly enough we notice that for the Asian payoff the performance of the second method is even much better with this more complicated model.

4.5.3.2 Automatic control variates

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	9.48	5.96	55.78	12.48
Control variate with analytic integration	9.57	5.71	56.24	12.53
Var ratio	4.02	4.40	1.73	3.98
Cost	2.37	2.27	4.20	2.73
Speed up	1.69	1.93	0.41	1.46

Table 4.6: Automatic control variates performances under a stochastic volatility model

This is an even more complicated model and the dimensions are again bigger than before. However, we still have some important speed ups and the computation times are still very acceptable.

4.6 Payoff decomposition to vanilla options

In this section, we suggest to build a neural network that allows us to write any payoff function as a decomposition of vanilla options. The network H is given

by

$$H(Z) = W_2(\sigma e^{W_1 Z} + b_1)^+ + b_2$$

Where σ is a vector where the first half elements are equal to 1 and the second half is equal to -1, to replicate calls and puts respectively. The expectation of the network can be computed analytically element by element similarly to what we have done in Section 4.4.2 using the following formulas

$$\mathbb{E}\left[(\exp(\sigma Z) + \mu)^+\right] = \begin{cases} \text{if } \mu \geq 0, \quad \mu + \exp\left(\dfrac{\sigma^2}{2}\right) \\ \text{else}, \quad \mu\left(1 - \mathcal{N}\left(\dfrac{\log(-\mu)}{\sigma}\right)\right) + \exp\left(\dfrac{\sigma^2}{2}\right)\left(1 - \mathcal{N}\left(\dfrac{\log(-\mu) - \sigma^2}{\sigma}\right)\right) \end{cases}$$

$$\mathbb{E}\left[(-\exp(\sigma Z) + \mu)^+\right] = \begin{cases} \text{if } \mu \leq 0, \quad 0 \\ \text{else}, \quad \mu\mathcal{N}\left(\dfrac{\log(\mu)}{\sigma}\right) - \exp\left(\dfrac{\sigma^2}{2}\right)\mathcal{N}\left(\dfrac{\log(\mu) - \sigma^2}{\sigma}\right) \end{cases}$$

We use this decomposition to form a control variate as in Section 4.4.2. In Annex B, we include a code snippet, to show how one can derive such a network using Tensorflow2.0.

Example 3. *We consider a Selecto payoff on an index of $n = 20$ underlyings. The payoff is given by*

$$\frac{1}{n_b}\sum_{i=1}^{n_b} \tilde{S}_T^i - \frac{1}{n}\sum_{i=1}^{n} S_T^i$$

where $\tilde{S}_T$ represents the vector S_T of performances rearranged such that $\tilde{S}_T^1 \geq \tilde{S}_T^2 \geq \ldots \geq \tilde{S}_T^n$ and $n_b < n$ is the number of selected underlyings. Table 4.7 represents the prices of the Selecto payoff with Monte Carlo and its variance with and without the vanilla decomposition control variate. We use The Black Scholes model and consider the following set of parameters $S_0 = (120, 40, 95, 60, 90, 40, 80, 100, 90100, 90, 250, 420, 100, 170, 160, 36, 85, 120, 160)$, $\sigma = (23, 35, 22, 29, 22, 40, 23, 27, 25, 19, 24, 40, 50, 21, 33, 32, 25, 22, 20, 33)$ in percentage, $\rho = 0.62$, $T = 1$ year. We decompose the payoff on 1000 call plus 1000 put which simply means that the network contains 2000 units in its first hidden layer.

n_b	Monte Carlo price	Control Variate	Var ratio	C	Speed up
1	308.26	305.90	81.36	19.35	4.20
2	216.73	218.48	220.65	12.81	17.22
3	167.85	167.85	310.49	12.46	24.92
4	135.15	136.22	407.72	13.58	30.02
5	113.24	113.50	545.30	13.32	40.94
6	95.62	95.52	672.86	13.17	51.09
7	81.57	81.33	750.83	13.08	57.40
8	69.71	69.78	728.68	12.61	57.79
9	59.88	60.18	758.02	13.24	57.25
10	52.03	52.04	718.17	13.04	55.07
11	44.94	45.02	665.68	13.00	51.21
12	38.57	38.84	711.87	13.88	51.29
13	33.32	33.32	581.74	12.51	46.50
14	28.38	28.30	591.35	13.22	44.73
15	23.63	23.64	402.98	12.63	31.91
16	19.25	19.18	592.73	13.21	44.86
17	14.47	14.48	515.61	13.59	37.94
18	9.50	9.50	321.36	12.88	24.95
19	4.67	4.66	143.55	12.36	11.61

Table 4.7: Decomposition of a Selecto payoff into Vanilla options

4.7 Robustness of the control variates

In this part, we test whether the networks presented in Section 4.3 are capable of resisting changes in the variables of the payoff and the model. Since this method only learns the most important directions for the payoff and the model and not the actual combination of the two, we hope it will be resisting to some parameters changes.

4.7.1 Asian price sensitivity

To study this interesting aspect, we first consider an arithmetic Asian option $(\frac{1}{d}\sum_{i=1}^{d} S_{t_i} - K)^+$. We train the network with the following set of parameters $K = 100.0, \sigma = 0.2, S_0 = 100.0, T = 1$ year and $d = 20$.
Once the network is trained, we test it for a range of parameters larger than the ones used for the training. The prices given by the network in the following figures are the raw prices without using them as control variates. Since this example is relatively simple, those prices are quite accurate without the need to correct the bias through the Monte Carlo.

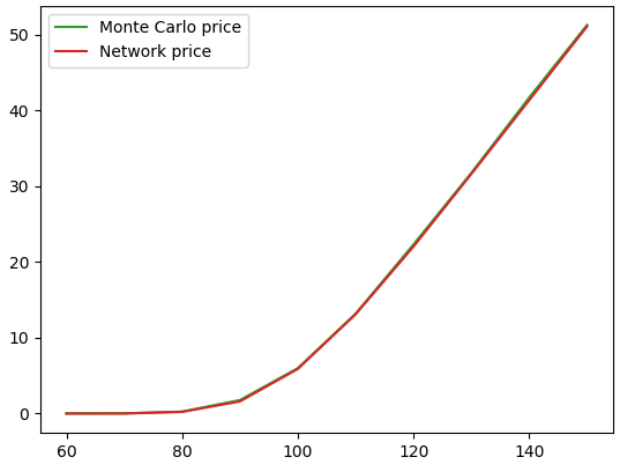


Figure 4.3: Asian price as a function of the spot

Trained with a spot value 100, the network is capable of approaching the model for a range of spots going from 60 to 150. This is very interesting, because it means we will not have to retrain the networks for every change in the market spot.

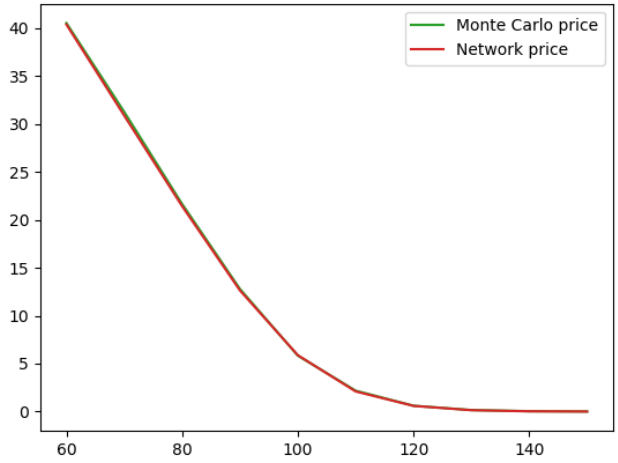


Figure 4.4: Asian price as a function of the strike

The network is also robust to the change of the strike parameter. Meaning that with one network we can cover a huge range of products having different strikes.

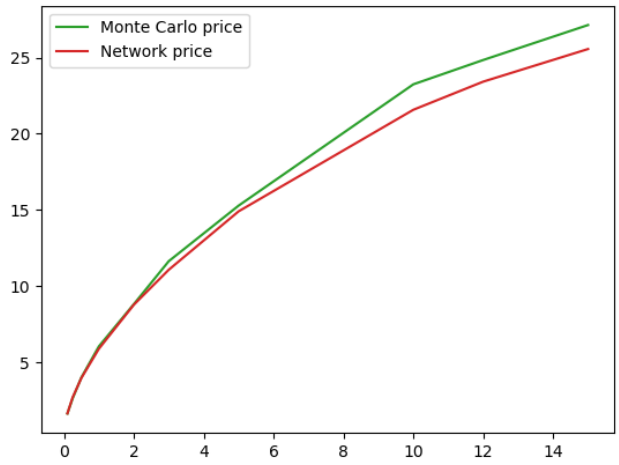


Figure 4.5: Asian price as a function of the maturity (Monte Carlo vs Network)

The price as a function of the maturity time has some bias, using the network as a control variate corrects this error as shown in the following figure.

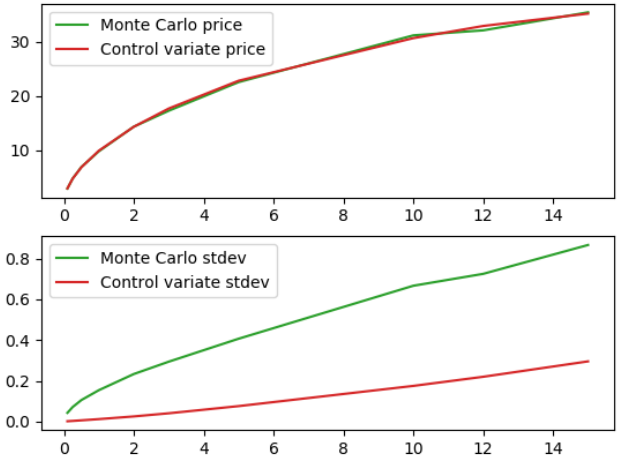


Figure 4.6: Asian price and stdev as a function of the maturity (Monte Carlo vs Control Variate)

Clearly, the control variate which is trained with a maturity 1 year performs some interesting variance reduction for a much larger and smaller maturities.

4.7.2 AutoCall price sensitivity

We now consider a more industrial payoff : The AutoCall. This payoff pays the following:

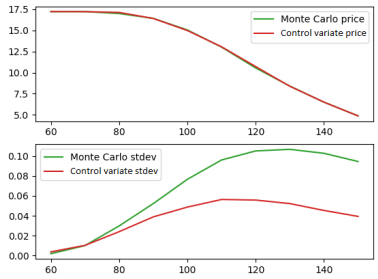
At each recall date t_i the option pays the following conditional coupon

$$\Pi_i = C_i \mathbb{1}_{P_{t_i} \geq AB_i} \prod_{j=0}^{i-1} \mathbb{1}_{P_{t_j} < AB_j}.$$

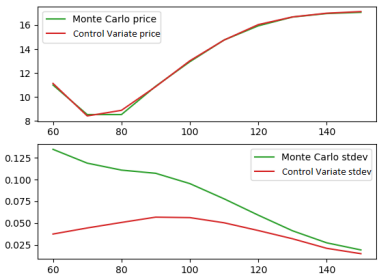
At maturity, if no AutoCall event has occurred, the option pays the following

$$\Pi_T = -(K - P_T)^+ \mathbb{1}_{P_T < DB} \prod_{i=0}^{d} \mathbb{1}_{P_{t_i} < AB_i}.$$

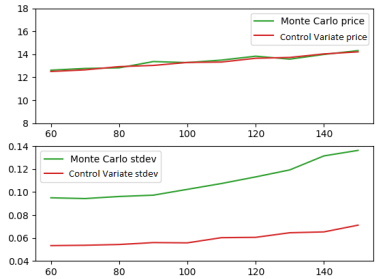
Where C_i is the value of the cash coupon at the date t_i, P_{t_i} is the performance of the underlyings at date t_i (in this context, we consider a basket performance type), AB_i is the AutoCall barrier at date t_i. If at some recall date, the performance goes above the AutoCall barrier, the client receives the corresponding coupon, and the product life ends immediately. At the maturity if the performance goes very low (lower than the Down And In Barrier DB), the client pays a put on the basket with strike K.
We train the network with the following set of parameters: $T = 3$ years, recall dates every year. $C_1 = 20.0, C_2 = 25.0, C_3 = 30.0$, $AB_i = 100.0\ \forall i$, $K = 80.0$, and $DB = 40.0$. However this network will be tested on parameters that it has never seen. The results are shown in Figure 4.7



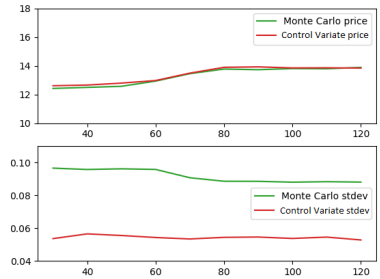
(a) AutoCall price and stdev in function of the AutoCall Barrier



(b) AutoCall price and stdev in function of the spot



(c) AutoCall price and stdev in function of the PDI strike



(d) AutoCall price and stdev in function of the PDI barrier

Figure 4.7: Control Variate for pricing Autocall

The interest of the dimension reduction method lies in its robustness to parameters change. In fact, this is due to the fact that we learn the directions where

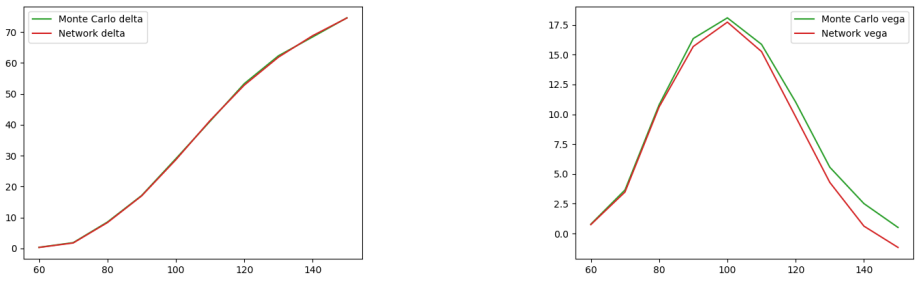
the variance of the payoff is concentrated rather than the actual payoff/model. The previous examples show that by learning a certain payoff given fixed parameters, the obtained network can be used to price the same payoff with different parameters.

4.7.3 Some Greeks profiles

Since the dimension reduction control variate is resistent to some parameters changes, we will use it to compute some greeks which are quantities representing the sensitivity of the payoff to some parameters. We consider for this section the basket option described earlier. We compute by finite difference some of the option's Greeks using the network. Notice that these Greeks are computed with the raw network without using it as a control variate. We compare them with the Greeks given by Monte Carlo.
We define the following Greeks:

- Delta $\delta_i = \frac{\partial \Pi}{\partial S^i}$ where Π is the option price and S^i the i'th underlying.
- Vega $\nu_i = \frac{\partial \Pi}{\partial \sigma^i}$ where σ^i the i'th underlying volatility.

These Greeks profiles are shown in Figure 4.8



(a) Delta as a function of the spot (b) Vega as a function of the spot

Figure 4.8: Basket Greeks using dimension reduction control variate

The Greeks profiles are very smooth and fit nicely to the curves given by Monte Carlo. This shows that the method can not only be used to reduce the variance of Monte Carlo price, but also in Greeks computation.

4.8 Conclusion

We have presented three new techniques for creating control variates using neural networks. We have tested our methods under different models and showed through numerical examples that the control variates allow significant speed ups without being time consuming. The first method we suggested represents a major quality of resistance to parameters change. So even if in some tests the method had us believing that it works better with payoffs that are close to being linear,

we still highly recommend using it for this aspect of robustness to parameters change that it has. We have proven through numerical examples that even for highly complicated payoffs (such as the Autocall example) the method is still efficient. The other two methods are much more intuitive since we use a given network to approximate a payoff function. they work better with non linear functions and give very important speed ups. The last method which uses only one hidden layer is less efficient with complicated models but it has the advantage of being very fast because the expectation of the control can be computed analytically.

Part II

Bermudan options pricing

Chapter 5

American options

5.1 Optimal strategies and dynamic programming equations

Unlike European options which can only be exercised at maturity time T, an American option can be exercised at any time prior to the option expiry date T. If exercised at time t, the option holder receives a cash flow $\tilde{Z}_t$. When the number of exercising dates is finite, we speak about Bermudan options. In practice, we will always deal with discretized problems, thus from now on we will only consider Bermudan options.

Let $(\Omega, \mathcal{A}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$ be a filtered probability space and consider a Bermudan option with exercising dates $0 = t_0 < t_1 < \ldots, t_N = T$. We model the underlying asset by an adapted Markov process $(X_t)_{t\geq 0}$ taking values in $\mathbb{R}^d$. We denote the discounted payoff of the Bermudan option at time t_j by Z_{t_j} and we consider that it is completely described by a non-negative measurable function h_j of the variable X_{t_j}, $Z_{t_j} = h_j(X_{t_j})$. We assume that for each $j = 0, \ldots, N$, $\mathbb{E}[Z_{t_j}^2] < \infty$. We write U_{t_j} the value of the option at time t_j for all $j = 0, \ldots, N$. At maturity time t_N, the discounted option price is given by $U_{t_N} = Z_{t_N}$. At time t_{N-1}, the option holder can either exercise his option immediately and receive the discounted payoff $Z_{t_{N-1}}$, or wait until the next step in which case he actually holds a European option with maturity time t_N and paying the discounted payoff U_{t_N}. The price of the European option is $\mathbb{E}[U_{t_N}/\mathcal{F}_{t_{N-1}}]$. So, the discounted price of the Bermudan option at time t_{N-1} is given by

$$U_{t_{N-1}} = \max(Z_{t_{N-1}}, \mathbb{E}\left[Z_{t_N}|\mathcal{F}_{t_{N-1}}\right])$$

We can reproduce this reasoning at all times: at time t_j, the holder of the option can either exercise or wait until t_{j+1} and it is as if he holds a European option expiring at t_{j+1} with discounted payoff $U_{t_{j+1}}$. Then, we deduce that the discounted price is given by the dynamic programming principle

$$\begin{cases} U_{t_N} &= Z_{t_N} \\ U_{t_j} &= \max\left(Z_{t_j}, \mathbb{E}\left[U_{t_{j+1}}|\mathcal{F}_{t_j}\right]\right) \text{ for } 1 \leq j \leq N-1. \end{cases} \tag{5.1}$$

The interpretation of this equation is that at each exercising date t_j, one should compare the immediate exercising value Z_{t_j} to the continuation value

$\mathbb{E}\left[U_{t_{j+1}}|\mathcal{F}_{t_j}\right]$, and decide to exercise only if the exercising value is greater than the continuation value. For $j = 0, \ldots, N$, we define

$$\mathcal{T}_{t_j} = \{\mathcal{F} - \text{stopping times with values in } \{t_j, \ldots, t_N\}\}.$$

An optimal stopping time for the sequence $(Z_{t_j})_{j=0,\ldots,N}$ is a stopping ν time maximising the value of the option, i.e such that

$$\mathbb{E}[Z_\nu] = \sup_{\tau \in \mathcal{T}_{t_0}} \mathbb{E}\left[Z_\tau\right].$$

Using the Snell envelope theory, we can state that

$$\tau_0 = \inf\{t_j, j = 0, \ldots, N \,:\, U_{t_j} = Z_{t_j}\}$$

is an optimal strategy for the the stopping problem and we have

$$U_0 = \mathbb{E}\left[Z_{\tau_0}\right] = \sup_{\tau \in \mathcal{T}_{t_0}} \mathbb{E}\left[Z_\tau\right]. \tag{5.2}$$

In the next sections, we will explore different methods for solving the dynamic programming equation. The difficulty in solving this equation comes from the computation of the conditional expectation $\mathbb{E}\left[U_{t_{j+1}}|\mathcal{F}_{t_j}\right]$ representing the continuation value. The first step of every algorithm is to approximate these expectations by a deterministic function of the underlying asset at the current time. We introduce a family of real valued functions $(\phi(\,\cdot\,, \theta))_{\theta \in \Theta}$ to be used as a proxy for the conditional expectations. We will make the functions and the parameter space Θ precise in the description of the algorithms. We introduce the approximate dynamic programming principle in which the parameter θ will carefully chosen at every step j.

$$\begin{cases} U_{t_N} & = Z_{t_N} \\ U_{t_j} & = \max\left(Z_{t_j}, \phi(X_{t_j}, \theta)\right) \text{ for } 1 \leq j \leq N-1. \end{cases} \tag{5.3}$$

First, we will focus on quantization. Then, we will explore some regression based algorithms such as the least-square approach proposed by Longstaff Schwartz and several further developments of their original algorithm. We will then, present an alternative to the dynamic programming equation as in (Becker et al., 2019), where the optimal stopping policy is directly estimated using neural networks. The following chapter, which represents our main contribution to this part, represents an approach using regression trees and random forests to solve the dynamic programming equation.

5.2 Numerical resolution of the dynamic programming equation

5.2.1 Quantization

5.2.1.1 General definitions

Quantization consists in approximating a random variable taking infinitely many values by a discrete variable taking at most n values called the quantizer. Let X be a random variable taking values in $\mathbb{R}^d$ that we wish to quantize.

Definition 5.2.1. *Let $y = \{y_1, \dots, y_n\} \in (\mathbb{R}^d)^n$ and $(C_i(y))_{1\leq i\leq n}$ be a Borelian partition of $\mathbb{R}^d$ such that $\forall i = 1, \dots, n$ $y_i \in C_i(y)$. A n-quantizer is an application $h_n : \mathbb{R}^d \to \mathbb{R}^d$ of the form*

$$h_n(x) = \sum_{i=1}^{n} y_i \mathbb{1}_{\{C_i(y)\}}(x), \quad \forall x \in \mathbb{R}^d$$

5.2.1.2 Optimal quantization

In this section, we assume that $\mathbb{E}[|X|^2] < \infty$. Then, we can define an optimal quantizer for this random variable in the $\mathbb{L}^2$-sense as follows

$$y^* = \inf_{y\in(\mathbb{R}^d)^n} \left\{ \mathbb{E}\left[\min_{y_i, i=1,\dots,n} |X - y_i|^2 \right] \right\}. \tag{5.4}$$

The corresponding optimal partition $(C_i^*(y^*))_{1\leq i\leq n}$ is the Voronoi tessellation associated to the nodes y^*. We recall that the Voronoi cell $C_i^*(y^*)$ is the set of points of $\mathbb{R}^d$ that are closer to y_i^* than to any other y_j^* for $j \neq i$

$$C_i^*(y^*) = \left\{ x \in \mathbb{R}^d, \forall j \neq i, |x - y_i^*| \leq |x - y_j^*| \right\}.$$

The sets $C_i(y^*)$ are compact and verify that their interiors are disjoint sets. However, the boundaries of the sets are not disjoint but this is not an issue as soon as the random variable X has a density with respect to the Lebesgue measure.

Gaussian case Assume X is a normal vector in $\mathbb{R}^d$ with covariance matrix $\lambda^2 I_d$, with $\lambda > 0$ then X writes λG where G is a standard normal vector in $\mathbb{R}^d$. Let y^* be optimal in (5.4) for G

$$y^* = \inf_{y\in(\mathbb{R}^d)^n} \left\{ \mathbb{E}\left[\min_{y_i, i=1,\dots,n} |G - y_i|^2 \right] \right\}.$$

Note that for $y_1, \dots, y_n \in \mathbb{R}^d$,

$$\begin{aligned} \mathbb{E}\left[\min_{y_i, i=1,\dots,n} |X - y_i|^2 \right] &= \mathbb{E}\left[\min_{y_i, i=1,\dots,n} |\lambda G - y_i|^2 \right] \\ &= \lambda^2 \mathbb{E}\left[\min_{y_i, i=1,\dots,n} \left|G - \lambda^{-1} y_i\right|^2 \right] \end{aligned}$$

Hence, we deduce that $\{\lambda y_1^*, \dots, \lambda y_n^*\}$ is optimal for X. This dilation property of the normal distribution plays a central role in the quantization of the Brownian motion as it boils down to quantizing the multi-dimensional standard distribution.

Computing an optimal quantizer In the following, we briefly present three commonly used approaches to compute an optimal quantizer. We refer to (Pagès and Printems, 2003) for a detailed presentations of these algorithms in a Gaussian framework. We want to solve (5.4).

- Fixed point method We suppose that X has a density. We can prove that for an optimal quantizer

$$\mathbb{E}\left[(X - y_i)\mathbb{1}_{C_i(y)}(X)\right] = 0 \; \forall i = 1, \dots, n$$

which can be rewritten as

$$y_i = \frac{\mathbb{E}\left[X\mathbb{1}_{C_i(y)}(X)\right]}{\mathbb{E}\left[\mathbb{1}_{C_i(y)}(X)\right]}$$

Lloyd's algorithm consists in defining a recursive sequence $(y^p)_{p\geq 0}$ in $(\mathbb{R}^d)^n$ starting at some point $y \in (\mathbb{R}^d)^n$

$$\begin{cases} y^0 & = y \\ y_i^{p+1} & = F_i(y^p) = \frac{1}{\mathbb{P}(X \in C_i(y^p))}\mathbb{E}[X\mathbb{1}_{C_i(y^p)}(X)] \; \forall i = 1, \dots, n \end{cases}$$

In dimension $d = 1$, one can show that if X has a strictly log-concave density function, then $x \mapsto (F_i(x))_{i=1,\dots,n}$ admits a unique fixed point which represents the optimal grid for the quantization. However, in higher dimensions the algorithm is hard to implement and there is no proof for its convergence. So this method is generally reserved only for the case $d = 1$.

- Deterministic gradient method We denote by $\hat{X}^y$ the quantized variable given by

$$\hat{X}^y = \sum_{i=1}^{n} y_i \mathbb{1}_{C_i(y)}(X)$$

and we define

$$D_n^X(y) = \mathbb{E}\left[\left|X - \hat{X}^y\right|^2\right]$$

An optimal n-quantizer y^* solves the optimization problem

$$\inf_{y \in (\mathbb{R}^d)^n} D_n^X(y)$$

The idea of this algorithm is to solve this minimization problem using a gradient descent as follows. Fix $y^0 \in (\mathbb{R}^d)^n$ and recursively define the sequence $(y^p)_p$ by

$$y_i^{p+1} = y_i^p - \eta_p \nabla D_n^X(y_i^p), \quad \forall i = 1, \dots, n$$

where $(\eta_p)_{p\geq 0}$ is a positive sequence verifying $\sum \eta_p = \infty$ and $\sum \eta_p^2 < \infty$.

- Stochastic gradient descent Note that

$$\begin{aligned} D_n^X(y) &= \mathbb{E}\left[|X - \hat{X}^y|^2\right] \\ &= \sum_{i=1}^{n} \mathbb{E}\left[|X - y_i|^2 \mathbb{1}_{C_i(y)}(X)\right] \\ &= \mathbb{E}\left[\min_{i=1,\dots,n} |X - y_i|^2\right] \end{aligned}$$

Let $d_n(x,u) = \min_{i=1,\dots,n}|x-u_i|^2$. From this follows that

$$\nabla D_n^X(y) = \mathbb{E}\left[\nabla_y d_n(y,X)\right]$$

where $\nabla_y d_n(y,u) = \left(2(y_i-u)\mathbb{1}_{C_i(y)}\right)_{1\le i\le n}$. The stochastic gradient algorithm consists in writing

$$y_i^{p+1} = y_i^p - \frac{\eta_p}{2}\nabla_y d_n(y_i^p, X_{p+1}) \quad \forall i = 1,\dots,n$$

where $(\eta_p)_{p\ge 0}$ is a positive sequence verifying $\sum \eta_p = \infty$ and $\sum \eta_p^2 < \infty$ and $(X_p)_p$ is a sequence of i.i.d random variables following the law of X. We can rewrite this equation as follows

$$y_i^{p+1} = y_i^p - \eta_p(y_i^p - X_{p+1})\mathbb{1}_{C_i(y^p)}(X_{p+1}) \quad \forall i = 1,\dots,n$$

Each iteration consists two phases. A first phase where we search for the cell where the variable X_{p+1} belongs called the competition phase. A second phase where we update the grid called the learning phase.

All these methods are computationally demanding and computing the optimal quantization grid is often considered as the major drawback of quantization. In practice, we hardly ever compute the true optimal quantizer when the underlying asset writes as a function f of the Brownian motion (or of its increments). Instead, we rely on the dilation property of the normal distribution to infer an optimal quantizer of the Brownian motion based on the optimal quantization grids for the standard normal distribution available at http://www.quantize.maths-fi.com. Then, we apply f to the nodes of the optimal quantization grid for the Brownian motion and use the new grid as if it were optimal for our model, which is not and it can actually be far from optimal.

5.2.1.3 Application to the dynamic programming equation

We wish to solve the dynamic programming equation (5.1). We recall that in our formulation, the discounted payoff process is entirely described by the values of the underlying assets $Z_{t_j} = h_j(X_{t_j})$. At each time t_j, we quantize the random variable X_{t_j} and define its n_j-quantizer by

$$\hat{X}_{t_j} := \sum_{i=1}^{n_j} y_i^j \mathbb{1}_{C_i^j(y^j)}(X_{t_j})$$

where the tessellation C and the nodes y are obtained by one of the strategies presented before.

We approximate the dynamic programming equation by

$$\begin{cases} U_{t_N} &= Z_{t_N} \\ U_{t_j} &= \max\left(Z_{t_j}, \mathbb{E}\left[U_{t_{j+1}}|\hat{X}_{t_j}\right]\right) \text{ for } 1\le j\le N-1. \end{cases}$$

The quantized process $(\hat{X}_{t_j})_{1\leq j\leq N}$ is not a Markov process with respect to its natural filtration. However, we can compute its transition probabilities for $j \in \{0,\dots,N\}$

$$\begin{aligned} p^j_{mn} &= \mathbb{P}\left(\hat{X}_{t_{j+1}} = y^{j+1}_n / \hat{X}_{t_j} = y^j_m\right) \\ &= \frac{\mathbb{P}\left(X_{t_{j+1}} \in C^{j+1}_n(y^{j+1}), X_{t_j} \in C^j_m(y^j)\right)}{\mathbb{P}\left(X_{t_j} \in C^j_m(y^j)\right)}. \end{aligned}$$

These quantities are usually estimated using Monte Carlo simulations. With the help of these transition probabilities, the conditional expectations write as discrete and finite sums. We obtain

$$\begin{cases} U_{t_N}\left(y^N_n\right) & = h_N\left(y^N_n\right) \ \forall n = 1,\dots,n_N \\ U_{t_j}\left(y^j_n\right) & = \max\left(h_j\left(y^j_n\right), \sum_{m=1}^{n_{j+1}} p^j_{mn} U_{t_{j+1}}\left(y^{j+1}_m\right)\right) \ \forall j \in \{0,\dots,N-1\}, n \in \{1,\dots,n_j\} \end{cases}$$

5.2.2 The least squares based algorithms

5.2.2.1 The Tsitsiklis Van Roy algorithm

The algorithm aims at solving the dynamic programming equation (5.1) using a least square method to determine the optimal parameter θ to use in (5.3). The original algorithm of Tsitsiklis and Van Roy approaches these conditional expectations with a linear regression on some basis functions $e = (e_1,\dots,e_p)$ for some $p \in \mathbb{N}$. In this setting, the function ϕ writes $\phi(x,\theta) = \sum_{i=1}^p \theta_i e_i(x)$ for $\theta \in \mathbb{R}^p$. We define

$$U^p_{t_j} = \sum_{i=1}^p \theta^p_{j,i} e_i(X_{t_j}) = \theta^p_j \cdot e(X_{t_j}).$$

We simulate M paths, and look for $\theta^p_j \in \mathbb{R}^p$ solving least squares problem

$$\inf_{a\in\mathbb{R}^p} \sum_{m=1}^M \left(U^{p,m}_{t_j} - a \cdot e(X^m_{t_j})\right)^2.$$

The algorithm is given below

1: Simulate M paths $(X^m_{t_i})_{i=0,\dots,N}$ for $m = 1,\dots,M$
2: On each path, compute the discounted payoff Z^m
3: Set $U^{p,m}_{t_N} = Z^m_{t_N}$ for $m = 1,\dots,M$
4: **for** $j = N-1,\dots,1$ **do**
5: Set $U^{p,m}_{t_j} = \max(Z^m_{t_j}, U^{p,m}_{t_{j+1}})$
6: Compute θ^p_j minimising

$$\sum_{m=1}^M \left(U^{p,m}_{t_j} - (\theta^p_j) \cdot e(X^m_{t_j})\right)^2$$

7: Set $U^{p,m}_{t_j} = \theta^p_j \cdot e(X^m_{t_j})$ for $m = 1,\dots,M$

8: **end for**
9: Return

$$\max\left(Z_0, \frac{1}{M}\sum_{m=1}^{M} U_{t_1}^{p,m}\right).$$

Note that the time t_0 is handled separately in the algorithm although it follows the same equation in the dynamic programming principle. This comes from the fact that the sigma field $\mathcal{F}_0$ is trivial and therefore conditional expectations w.r.t $\mathcal{F}_0$ are actually standard expectations, which must be computed as such and not using the least squares approach.

This algorithm gives a first intuition on how to use least squares to approximate the value function. We will see in the next method that approaching the policy gives better results, as the errors made in the estimation of the continuation values are less abusive than when we directly approach the value function.

5.2.2.2 The Longstaff Schwartz algorithm

Longstaff and Schwartz suggested to rewrite (5.1) in terms of optimal stopping. Their approach is often referred to as policy iteration. Let $(\tau_j)_{j=0,\dots,N}$ be a sequence of stopping times such that for a fixed $0 \leq j \leq N$, τ_j is the optimal stopping time after time t_j

$$\tau_j = \min\{t \in \{t_j, \dots, t_N\} \text{ s.t. } Z_t \geq U_t\}.$$

The sequence $(\tau_j)_{j=0,\dots,N}$ can then be rewritten recursively as dynamic programming equation

$$\begin{cases} \tau_N &= t_N = T \\ \tau_j &= t_j \mathbb{1}_{\left\{Z_{t_j} \geq \mathbb{E}\left[U_{t_{j+1}} | \mathcal{F}_{t_j}\right]\right\}} + \tau_{j+1}\mathbb{1}_{\left\{Z_{t_j} < \mathbb{E}\left[U_{t_{j+1}} | \mathcal{F}_{t_j}\right]\right\}}, \text{ for } 1 \leq j \leq N-1 \end{cases}$$

Note that $U_{t_{j+1}} = \mathbb{E}\left[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}\right]$ for every j. Then, we obtain for $j = 1, \dots, N$

$$\tau_j = t_j \mathbb{1}_{\left\{Z_{t_j} \geq \mathbb{E}\left[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}\right]\right\}} + \tau_{j+1}\mathbb{1}_{\left\{Z_{t_j} < \mathbb{E}\left[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}\right]\right\}}. \tag{5.5}$$

The difficulty in this equation still comes from the computation of the conditional expectations representing the continuation values. However, since the strategy is built on the policy, the errors in the estimation of these conditional expectations do not necessarily induce an error in the estimation of the optimal strategy as long as the estimation errors do not modify the exercising decision. In fact, all that matters is the sign of $Z_{t_j} - \mathbb{E}\left[Z_{\tau_{j+1}} / \mathcal{F}_{t_j}\right]$ and not the exact value of $\mathbb{E}\left[Z_{\tau_{j+1}} / \mathcal{F}_{t_j}\right]$. Longstaff and Schwartz suggest to use a linear regression on some basis functions $e = (e_1, \dots, e_p)$ to approach the conditional expectations, typically polynomial functions. The approximation is done by minimising the least square error as in the Tsitsiklis Van Roy algorithm. As in the previous sections, we denote this approximation by $\phi(X_{t_j}, \theta_j^p)$ with $\phi(x, \theta) = \theta \cdot x$ and we define

$$\begin{cases} \tau_N^p &= t_N = T \\ \tau_j^p &= t_j \mathbb{1}_{\left\{Z_{t_j} \geq \phi(X_{t_j}, \theta_j^p)\right\}} + \tau_{j+1}^p \mathbb{1}_{\left\{Z_{t_j} < \phi(X_{t_j}, \theta_j^p)\right\}} \text{ for } 1 \leq j \leq N-1 \end{cases}$$

where θ_j^p solves the least squares problem

$$\inf_\theta \mathbb{E}\left[|\phi(X_{t_j},\theta) - Z_{\tau_{j+1}^p}|^2\right].$$

Then, the price is approximated by

$$U_0^p = \max\left(Z_{t_0}, \mathbb{E}\left[Z_{\tau_1^p}\right]\right).$$

Now, we present the algorithm obtained when replacing the expectations by Monte-Carlo approximations.

1: Simulate M paths $(X_{t_i}^m)_{i=0,\dots,N}$ for $m = 1,\dots,M$
2: On each path, compute the discounted payoff Z^m for all time steps
3: Set $\tau_N^{p,m} = t_N$ for $m = 1,\dots,M$
4: **for** $j = N-1,\dots,1$ **do**
5: Compute θ_j^p minimising

$$\sum_{m=1}^{M}\left|\phi(X_{t_j}^m,\theta) - Z_{\tau_{j+1}^{p,m}}^m\right|^2 \tag{5.6}$$

6: For $m = 1,\dots,M$

$$\tau_j^{p,m} = t_j \mathbb{1}_{\left\{Z_{t_j}^m \geq \phi(X_{t_j}^m,\theta_j^p)\right\}} + \tau_{j+1}^{p,m}\mathbb{1}_{\left\{Z_{t_j}^m < \phi(X_{t_j}^m,\theta_j^p)\right\}}$$

7: **end for**
8: Return

$$U_0^{p,M} = \max\left(Z_0, \frac{1}{M}\sum_{m=1}^{M} Z_{\tau_1^{p,m}}^m\right).$$

In practice we only keep the in the money paths at the current time.

Whereas U_0^p approximates the true price U_0 from below, we do not know the bias of $U_0^{p,M}$ because the computation of θ_p^j mixes all the samples. To actually ensure to have a lower approximation of the true price (modulo the Monte Carlo errors), we draw a new set of paths $\tilde{X}^m$ for $m = 1,\dots,M$ independently of $(X^m)_{m=1,\dots,M}$ and use them to estimate $\mathbb{E}\left[Z_{\tau_1}^p\right]$ using Monte-Carlo. For every new path $\tilde{X}^m$, $m = 1,\dots,M$ we may compute

$$\begin{cases} \tilde{\tau}_N^{p,m} & = t_N = T \\ \tilde{\tau}_j^{p,m} & = t_j \mathbb{1}_{\left\{\tilde{Z}_{t_j}^m \geq \phi(\tilde{X}_{t_j}^m,\theta_j^p)\right\}} + \tilde{\tau}_{j+1}^{p,m}\mathbb{1}_{\left\{\tilde{Z}_{t_j}^m < \phi(\tilde{X}_{t_j}^m,\theta_j^p)\right\}} \text{ for } 1 \leq j \leq N-1 \end{cases}$$

where the sequence $(\theta_j^p)_{j=1,\dots,N-1}$ was computed during the first stage using the samples X^m for $m = 1,\dots,M$.

We refer to (Clément et al., 2002) for a detailed theoretical analysis of the convergence of the algorithm. For a fixed p, they prove a strong law of large numbers for the a.s. convergence of $U_0^{p,M}$ to U_0^p when $M \to \infty$. They also prove a central limit theorem but unfortunately the asymptotic variance is not explicit and therefore it cannot be used to derive a confidence interval. This is actually a major drawback of American Monte-Carlo methods for which no confidence interval can be built online as for linear Monte-Carlo methods.

5.2.3 Solving the least square problem

5.2.3.1 Regression on a vector space

To solve the least square problem (5.6), we differentiate the least square error with respect to the variable a

$$\sum_{m=1}^{M} \left(Z_{\tau_{j+1}^{p,m}} - a^T e(X_{t_j}^m) \right) e(X_{t_j}^m) = 0$$
$$\left(\sum_{m=1}^{M} e(X_{t_j}^m) e(X_{t_j}^m)^T \right) a = \sum_{m=1}^{M} Z_{\tau_{j+1}^{p,m}} e(X_{t_j}^m)$$
$$\frac{1}{M} \sum_{m=1}^{M} Z_{\tau_{j+1}^{p,m}} e(X_{t_j}^m) = D_M a$$

with

$$D_M = \frac{1}{M} \sum_{m=1}^{M} e(X_{t_j}^m) e(X_{t_j}^m)^T.$$

The matrix D_M is symmetric and writes as the sum of matrices with rank 1. Note that D_M converges to $D = \mathbb{E}[e(X_{t_j}) e(X_{t_j})^T]$, which is definite positive as soon as the family e is a free family and X_{t_j} takes *sufficiently many different* values. The definite positiveness of D is equivalent to

$$\forall x \in \mathbb{R}^p, \quad x^T e(X_{t_j}) \neq 0.$$

For instance, if e denotes the basis of polynomials with degree less or equal than p in dimension 1, this condition is met as soon as X_{t_j} takes at least $p+1$ different values. For large enough M, D_M is a also definite positive and can be written $D_M = L^T L$ where L is the Cholesky factorization of D_M. Then, the original problem boils down to solving two triangular linear systems. However, in practice the matrix D_M is generally ill conditioned and the Cholesky algorithm yields null or slightly negative eigen-values due to the relatively numerical stability of the Cholesky factorization. To solve this issue, we advise to use a QR decomposition with pivoting instead, which is much more stable from a numerical point of view. The matrix R is triangular and the matrix Q is orthonormal, meaning $Q^T Q = I$. Hence, the original problem is transformed into a triangular linear system and a matrix multiplication.

5.2.3.2 Regression on a family of local functions

We consider K functions $(\phi_k)_{k=1,\dots,K}$ defined on $\mathbb{R}^d$ with disjoint supports. We assume that all the functions ϕ_k vanish outside the hyper-cube $[-H, H]^d$ for some $H > 0$ and that the support of each ϕ_k is itself a sub hyper-cube of $[-H, H]^d$. Let $n \in \mathbb{N}$ be the number of cuts per dimension. We denote by $\Delta = \frac{2H}{n}$ the width of each support along a given dimension. We define the support cubes as $\mathcal{C}_{i_1,\dots,i_d} = [-H + i_1\Delta, -H + (i_1+1)\Delta] \times \dots, \times [-H + i_d\Delta, -H + (i_d+1)\Delta]$ with

$i_j = 0, \dots, n-1$ and $j = 1, \dots, d$. Figure 5.1 illustrates these cubes for the case $H = 1, d = 2$ and $n = 4$.

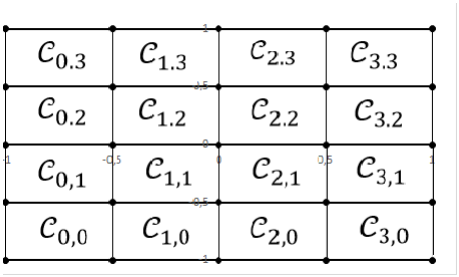


Figure 5.1: Illustration of a cubic cut in $[-1, 1]^2$ with 4 cuts per dimension

Using local functions to solve the least squares problem reduces to a cube-by-cube resolution since the cubes have disjoint supports, which considerably reduces the complexity of the problem. (Gobet, 2016) suggests to use local polynomials for example. In this case, he shows that if the conditional expectation to approach is a bounded function of class $\mathcal{C}^{p+\beta}(\mathbb{R}^d, \mathbb{R})$ with $p \in \mathbb{N}, \beta \in (0, 1]$ having bounded derivatives and β-Hölder continuous p-th order derivatives, then

$$\mathbb{E}\left[\left|\mathbb{E}\left[Y/X\right] - \sum_{k=1}^{K} \alpha_i \phi_k(X)\right|^2\right] \leq C\frac{(\log(\frac{1}{\Delta})\Delta^{-1})^d}{M} + C\Delta^{2(p+\beta)}$$

where the α_i's are the coefficients of the regression. We will consider the case where in each cube we approach the conditional expectation by a constant value (which can be considered as a special case of local polynomials with degree 0). The final approximation yields a piece-wise constant function of the form $\sum_{i_1=0}^{n-1} \dots \sum_{i_d=0}^{n-1} \alpha_{i_1,\dots,i_d} \mathbb{1}_{\mathcal{C}_{i_1,\dots,i_d}}$.

Example 4. *Let* $(Z_1, Z_2) \sim \mathcal{N}\left(0, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right)$. *We may write* $Z_2 = \rho Z_1 + \sqrt{1-\rho^2}Y$ *with* $Y \sim \mathcal{N}(0,1) \perp Z_1$. *Thus*

$$\begin{aligned}
\mathbb{E}\left[Z_2^2 | Z_1\right] &= \mathbb{E}\left[\rho^2 Z_1^2 + 2\rho\sqrt{1-\rho^2}Z_1 Y + (1-\rho^2)Y^2 | Z_1\right] \\
&= \rho^2 Z_1^2 + 2\rho\sqrt{1-\rho^2}Z_1 \mathbb{E}\left[Y\right] + (1-\rho^2)\mathbb{E}\left[Y^2\right] \\
&= 1 + \rho^2(Z_1^2 - 1)
\end{aligned}$$

We simulate $M = 10000$ *samples of* (Z_1, Z_2) *and look for the best approximation by local constant functions on* $[-3, 3]$ *using the cubic splits defined above with 16 cuts. We also, use regression trees to perform a different splitting into a piece-wise constant function for comparison. Figure 5.2 shows the results of these approximations compared to the real expectation.*

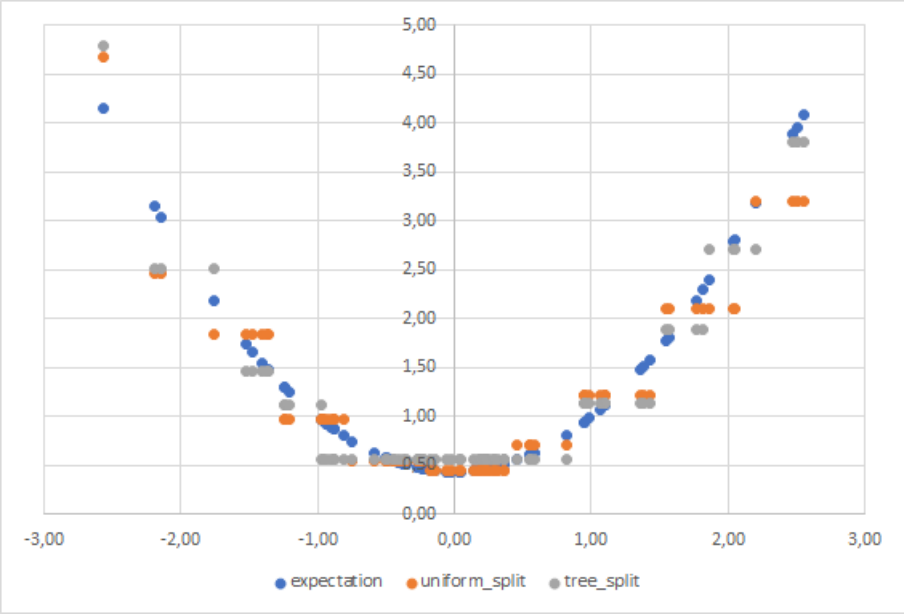


Figure 5.2: Approximation of $\mathbb{E}[Z_2^2|Z_1]$ using local constant functions and regression trees with 16 split

The uniform split using local functions fits well the expectation function. However, it does not optimize the cuts to fit the curve in the best way. Regression trees can be seen as an algorithmic alternative to uniform splitting which adapts the cuts to the function at hand. In Figure 4.2, we can see that the regression tree has one constant value in $[-1, 1]$ as the expectation function has a null tangent in this interval whereas the uniform splitting method is obliged to make several unnecessary cuts in this interval as the cuts are predetermined. Tested on 1000 new samples, the uniform split method yields a mean squared error of 0.07, whereas the regression tree methods yields a mean squared error of only 0.05. In the next chapter, we will see how to use regression trees to solve the Bermudan option pricing problem and study the convergence of this method. As for local function regression, (Gobet, 2016) studies the usage of local polynomials for solving the dynamic programming equation.

5.2.3.3 Neural networks regression

So far, we have focused on linear approximations: the set of functions $\phi(\cdot, \theta)_{\theta \in \mathbb{R}^p}$ was actually the vector space spanned by the functions $e_1, \ldots, e_p$ and $\phi(\cdot, \theta) = \theta^T e$. In this section, we present an alternative approximation method using neural networks proposed by (Lapeyre and Lelong, 2021) . They take for the family of functions $(\phi(\cdot, \theta))_{\theta \in \mathbb{R}^p}$ a feed forward neural network with a fixed number of layers and neurons per layer. Note that the set of all neural networks with fixed width and depth is not a vector space. This actually completely changes the nature of the approximation and the mathematical analysis relies on completely different techniques. In the case of neural networks, (Lapeyre and Lelong, 2021) managed to prove theoretical results very similar to the one obtained by (Clément et al., 2002) in the case of linear approximations using techniques coming

from stochastic optimization and more precisely stochastic programming.

Their algorithm follows exactly the same lines as the original Longstaff Schwartz approach but the least squares problem is replaced by the tuning of a neural network. Consider M samples of the underlying assets X^m for $m = 1, \dots, M$, they consider the dynamic programming equation on stopping times

$$\begin{cases} \tau_N^{p,m} & = t_N = T \\ \tau_j^{p,m} & = t_j \mathbb{1}_{\left\{Z_{t_j}^m \geq \phi_p(X_{t_j}^m, \theta_j^M)\right\}} + \tau_{j+1}^{p,m} \mathbb{1}_{\left\{Z_{t_j}^m < \phi_p(X_{t_j}^m, \theta_j^M)\right\}} \text{ for } 1 \leq j \leq N-1 \end{cases}$$

where θ_j^M solves the following optimization problems

$$\inf_\theta \frac{1}{M} \sum_{m=1}^{M} \left| \phi_p(X_{t_j}^m, \theta) - Z_{\tau_{j+1}^{p,m}}^m \right|^2.$$

The algorithm shows great numerical results especially in high dimensions where classic linear regressors such as linear least squares approaches become less efficient as they suffer from the curse of dimensionality.

5.2.4 Other methods

In this section, we present two alternative methods based on Monte Carlo simulation.

5.2.4.1 Tree method

(Broadie and Glasserman, 1997) suggest to price Bermudan options using random trees. The method consists in simulating different scenarios. Starting from the initial point X_0, we choose a branching parameter $b \geq 2$ and we simulate independently $X_1^1, \dots, X_1^b$ following the law of X_1, then starting from each X_1^i, we simulate b independent successors $X_2^{i_1}, \dots, X_2^{i_b}$ from the law of $X_2/X_1 = X_1^i$ and so on. At the $j-$th time step, there are b^j nodes. Let $X_j^{i_1,\dots,i_j}$ be the path obtained at time step t_j by choosing the nodes following the indices $i_1, \dots, i_j$ respectively, we may approach the dynamic programming equation (5.1) with the following equation

$$\begin{cases} U_{t_N,b}^{i_1,\dots,i_N} & = h(X_{t_N}^{i_1,\dots,i_N}) \\ U_{t_j,b}^{i_1,\dots,i_j} & = \max\left(h(X_{t_j}^{i_1,\dots,i_j}), \frac{1}{b} \sum_{i=1}^{b} U_{t_{j+1},b}^{i_1,\dots,i_j,i} \right). \end{cases}$$

Working backward, we deduce the price of the option at time 0. To illustrate this method, we show an example from (Glasserman, 2004), dealing with a call option in the money with strike 100. Figure 5.3 shows a random tree with a branching $b = 3$ and 3 exercising dates.

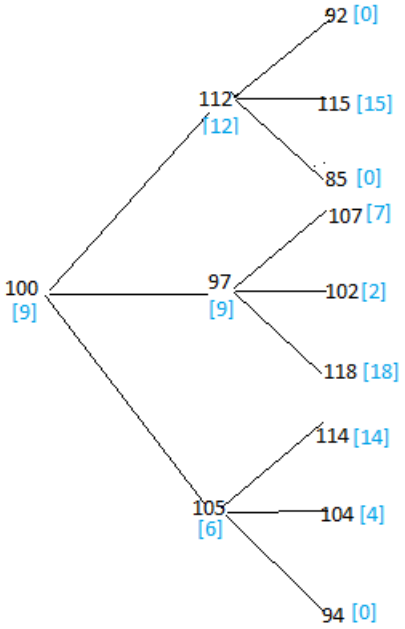


Figure 5.3: Random tree for pricing a call option in the money with strike 100. In black the different paths for the underlying, and in blue the value function of the option at each node.

This method is very simple to understand and easy to implement. However, its complexity grows exponentially with each new level as it behaves like a non recombining tree.

5.2.4.2 Stochastic mesh methods

This method consists in generating different branches starting from a point X_0, resulting in a certain tree similarly to the random trees method. However, the only difference is that when valuing the option at a certain node at time step t_j, one would not use only the paths that are generated from the current node but rather all the nodes at step t_{j+1}. We denote by $U^i_{t_j,b}$ the value of the option computed at the node i at time t_j and we define

$$\begin{cases} U^i_{t_N,b} &= h(X^i_{t_N}) \\ U^i_{t_j,b} &= \max\left(h(X^i_{t_j}), \frac{1}{b}\sum_{k=1}^{b} W^j_{ik} U^k_{t_{j+1},b}\right) \end{cases}$$

where W^j_{ik} is the weight attached to the arc joining $X^i_{t_j}$ to $X^k_{t_{j+1}}$. The price of the option at time 0 is finally given by

$$U_0 = \frac{1}{b}\sum_{k=1}^{b} U^k_{t_1,b}.$$

We write $X^{(b)}_{t_j} = (X^1_{t_j}, \ldots, X^b_{t_j})$ for all $j = 1, \ldots, N$. We enforce the following conditions on the mesh

- The mesh construction is Markovian, meaning that for all $j = 1, \dots, N$, $(X_{t_0}^{(b)}, \dots, X_{t_{j-1}}^{(b)})$ and $(X_{t_{j+1}}^{(b)}, \dots, X_{t_N}^{(b)})$ are independent given $X_{t_j}^{(b)}$.
- Each W_{ik}^j is a deterministic function of $X_{t_j}^{(b)}$ and $X_{t_{j+1}}^{(b)}$.
- For all $j = 1, \dots, N-1, i = 1, \dots, b$

$$\frac{1}{b}\sum_{k=1}^{b} \mathbb{E}\left[W_{ik}^j U_{t_{j+1},b}^k | X_{t_j}^{(b)}\right] = \mathbb{E}\left[U_{t_{j+1}} | X_{t_j} = X_{t_j}^i\right].$$

We suppose in addition that the Markov chain $(X_{t_0}, \dots, X_{t_N})$ admits transition densities $f_1, \dots, f_N$. Meaning that for $A \subseteq \mathbb{R}^d$ and $x \in \mathbb{R}^d$

$$\mathbb{P}\left(X_{t_j} \in A / X_{t_{j-1}} = x\right) = \int_A f_j(x, y)dy$$

Then, the marginal density of X_{t_1} is given by $g_1(.) = f_1(X_0, .)$ and for $j = 2, \dots, N$, the marginals are given by

$$g_j(y) = \int g_{j-1}(x) f_j(x, y)dx$$

We wish to estimate $\mathbb{E}\left[U_{t_{j+1}} | X_{t_j} = x\right] = \int U_{t_{j+1}}(y) f_{j+1}(x, y)dy$. We choose

$$W_{ik}^j = \frac{f_{j+1}(X_{t_j}^i, X_{t_{j+1}}^k)}{g_{j+1}(X_{t_{j+1}}^k)}$$

which represents the likelihood ratio between the transition density and the mesh density. Given these weights, we have

$$\begin{aligned}
\frac{1}{b}\sum_{k=1}^{b} W_{ik}^j U_{t_{j+1}}(X_{t_{j+1}}^k) \to \mathbb{E}_g\left[W_i^j U_{t_{j+1}}(X_{t_{j+1}}) | X_{t_j}^i\right] \\
&= \int \frac{f_{j+1}(X_{t_j}^i, y)}{g(y)} U_{t_{j+1}}(y) g(y) dy \\
&= \int f_{j+1}(X_{t_j}^i, y) U_{t_{j+1}}(y) dy \\
&= \mathbb{E}\left[U_{t_{j+1}} | X_{t_j} = X_{t_j}^i\right]
\end{aligned}$$

which ensures the second condition. (Broadie and Glasserman, 2004) show that with these conditions, the estimator gives the correct price at each node of the tree.

5.3 Alternative to the dynamic programming equation: direct approximation of the optimal stopping time

5.3.1 Randomized exercise strategies

In this section, we present a randomized exercise strategy algorithm by (Bayer et al., 2020). This algorithm is completely different from the ones we have seen

so far as it does not rely on the dynamic programming equation. The price of the option is estimated directly by solving Equation 5.2. To do so, for every function $f : [0,T] \times \mathbb{R}^d_+ \to \mathbb{R}_+$, we consider the following randomized exercise strategy

$$\tau_f := \inf \left\{ t \in [0,T] \ : \ \int_0^t \lambda_u du \geq Y \right\}$$

with $\lambda_t = f(t, X_t)$, for all $t \in [0,T]$ and Y a standard exponential random variable independent from the process $(X_t)_{t\in[0,T]}$. The family of random times will be used to approximate the optimal stopping time for the American option. Note that τ_f is the first jump time of a Poisson process with rate $(\lambda_t)_{t\in[0,T]}$ and is therefore not a stopping time. Yet, when conditioning on Y, τ_f becomes a stopping time. We denote the expected payoff under a randomized exercise strategy as follows:

$$\psi(f) := \mathbb{E}\left[U_{\tau_f \wedge T}\right]$$

and define the process $L_t = \exp\left(-\int_0^t \lambda_u du\right)$ for $t \in [0,T]$. The expected payoff under a randomized strategy is then given by

$$\psi(f) = \mathbb{E}\left[\int_0^T U_t \lambda_t L_t dt + U_T L_T\right].$$

The authors show that the price of the option is given by

$$U_0 = \sup_{f:[0,T]\times\mathbb{R}^d_+ \to \mathbb{R}_+} \psi(f).$$

Using this result, we may compute the option price by choosing a set of parametric functions and optimising over these parameters. For example, the authors consider the set of functions defined by

$$F_p := \{f_p(t,x) := \mathbb{1}_{h_t(x)>0} \exp(p(t, \log(x)))/p \in \mathcal{P}\}$$

where $\mathcal{P}$ is a finite linear space of functions on $\mathcal{T} \times \mathbb{R}^d$. For example, the set of polynomial functions with degrees less than or equal to k. Numerical experiments of the paper show that the randomized exercise policy based on polynomials yield remarkably accurate price estimates within few iterations of optimization.

5.3.2 Approximating the optimal stopping time using neural networks

As in the previous section, the goal here is to solve Equation (5.2) directly without using the dynamic programming equation. To do so, for $j = 0, \ldots, N$, we consider a set of measurable functions $f_j, f_{j+1}, \ldots, f_N : \mathbb{R}^d \to \{0,1\}$ with $f_N \equiv 1$ and we write

$$\tau_j = \sum_{k=j}^{N} t_k f_k(X_{t_k}) \prod_{p=j}^{k-1} \left(1 - f_p(X_{t_p})\right). \tag{5.7}$$

A stopping time given by Equation (5.7) is optimal among $\mathcal{T}_{t_j}$ (See (Becker et al., 2019, Theorem 1)). In particular,

$$\tau = \sum_{j=1}^{N} t_j f_j(X_{t_j}) \prod_{p=j}^{k-1} \left(1 - f_p(X_{t_p})\right)$$

is an optimal stopping time for the problem (5.2). Given this result, (Becker et al., 2019) implement an algorithm using neural networks to approximate the optimal stopping time for problem (5.2). For each $j = 1, \ldots, N-1$, they approximate the function f_j with a neural network f^{θ_j}, with $\theta_j \in \mathbb{R}^q$ representing the weights of the network.
At time t_N, we know that $f_N \equiv 1$. We proceed then by backward induction. at time t_j, we assume that $\theta_{j+1}, \ldots, \theta_{N-1}$ have been estimated so as $\tau_{j+1} = \sum_{k=j+1}^{N} t_k f^{\theta_k}(X_{t_k}) \prod_{p=j+1}^{k-1} \left(1 - f^{\theta_p}(X_{t_p})\right)$ produces an expected value $\mathbb{E}_{t_{j+1}}\left[Z_{\tau_{j+1}}\right]$ close to $U_{t_{j+1}}$. We then build a neural network F^{θ_j} with values in $(0,1)$ which produces a stopping probability in $(0,1)$. Such a network is obtained by using in the output layer the logistic function $\psi(x) = \frac{1}{1+e^{-x}}$ as an activation function for example. This network estimates the probability of exercise at time t_j that maximises the value of the option at time t_j. Thus θ_j solves the following equation

$$\sup_{\theta \in \mathbb{R}^q} \mathbb{E}\left[F^{\theta}(X_{t_j}) Z_{\tau_j} + \left(1 - F^{\theta}(X_{t_j})\right) Z_{\tau_{j+1}}\right] \tag{5.8}$$

In practice, the expectation is approximated with a Monte Carlo estimator and the optimum is obtained with a gradient ascent using mini-batch. Once we have estimated θ_j, we define the stopping decision function f^{θ_j} by

$$f^{\theta_j}(x) = \begin{cases} 0 & \text{if } F^{\theta_j}(x) < \frac{1}{2} \\ 1 & \text{if } F^{\theta_j}(x) \geq \frac{1}{2} \end{cases} \qquad \forall x \in \mathbb{R}^d.$$

The authors show that provided a flexible network with enough parameters is used, this method yields almost optimal stopping decisions for all $t_j, j = 0, \ldots, N$.

Chapter 6

Pricing Bermudan Options using regression trees/random forests

This chapter is based on (Ech-Chafiq et al., 2021a).

6.1 Introduction

Bermudan options are very widespread in financial markets. Their valuation adds a challenge of optimal stopping determination in comparison to European options. Bermudan options offer the investor the possibility to exercise his option at any date of his choice among a certain number of dates prior to the option expiry, called exercise dates. Naturally, the option holder will have to find the most optimal date to exercise. To do so, at each exercise date, he will compare the payoff of the immediate exercise to the expected value of continuation of the option and decide to exercise only if the immediate exercise value is the highest. We can formulate this problem as a dynamic programming equation, where the main difficulty comes from the computation of the conditional expectation representing the expected continuation value of the option. Many papers have discussed this issue, starting with regression-based algorithms; see for example (Tsitsiklis and Roy, 1999) and (Carriere, 1996). Also, in this category falls the most commonly used method for pricing Bermudan options which is the Least Squares Method (LSM) presented by Longstaff and Schwartz in (Longstaff and Schwartz, 2001) where the conditional expectation is estimated by a least squares regression of the post realized payoffs from continuation on some basis functions of the state variables (usually polynomial functions). Another class of algorithms focuses on quantization approaches, see for example (Bally et al., 2005). The algorithm consists in computing the conditional expectations by projecting the diffusion on some optimal grid. We also have a class of duality based methods that give an upper bound on the option value for a given exercise policy by adding a quantity that penalizes the incorrect exercise decisions made by the sub-optimal policy, see for example (Rogers, 2002), (Andersen and Broadie, 2004) and (Lelong, 2018). The last class of algorithms is based on machine learning techniques. For example, using Neural networks to estimate the continuation values in (Kohler et al., 2010) or more recently in (Lapeyre and Lelong, 2021), or

using Gaussian process regression as in (Ludkovski, 2018). Our solution falls in this last category of algorithms. We examine Bermudan options' prices when the continuation values' estimation is done using regression trees or random forests.

Let X, Y be two random variables with values in $[0,1]^d$ and $\mathbb{R}$ respectively. A regression tree approximates the conditional expectation $\mathbb{E}[Y|X]$ with a piecewise constant function. The tree is built recursively, generating a sequence of partitions of $[0,1]^d$ that are finer and finer. The approximation value on each set in the partition can be seen as a terminal leaf of the tree. This algorithm is very simple and efficient. However, it can easily over-fit the data, which results in high generalization errors. To solve this issue, we use ensemble methods to aggregate multiple trees, which means that we create multiple trees and then combine them to produce improved results. We suggest using random forests (see (Breiman, 2001)). This method consists in averaging a combination of trees where each tree depends on a random vector sampled independently and identically for each tree in the forest. This vector will allow to differentiate the trees in the random forest and can be chosen in different ways. For example, one can draw for each tree a sub-sample of training from the global training data without replacement (this method is called bagging and is thoroughly studied in (Breiman, 1999)). A second method is random split selection, where at each node, the split is selected at random from among the K best splits, see (Dietterich, 2000). Other methods for aggregating regression trees into random forests can be found in the literature, see for example (Breiman, 2001) or (Ho, 1998).
The structure of the paper will be as follows. First, we present the regression trees algorithm and the algorithm of least squares using regression trees. Then, we proceed to present some convergence results for regression trees and study the convergence of the LS algorithm when regression trees are used to estimate the continuation values. Then, we briefly talk about Random Forests before we finally study some numerical examples.

6.2 Regression trees

6.2.1 Definition of a regression tree

Let X be a r.v with values in $[0,1]^d$ and Y a squared integrable real-valued r.v. We want to approximate the conditional expectation $\mathbb{E}[Y|X]$. Throughout this paper, we will consider for computational convenience that X has a density f_X in $[0,1]^d$ w.r.t the Lebesgue measure. We assume given a training sample $D_M = \{(X_1, Y_1), \dots, (X_M, Y_M) \in [0,1]^d \times \mathbb{R}\}$ where the (X_i, Y_i)'s are i.i.d random variables following the law of (X, Y). An approximation using a regression tree consists in writing the conditional expectation as a piecewise constant function of X. Each domain where the function is constant can be seen as a terminal leaf of a tree. Let us start with the one-dimensional case ($d = 1$). Consider the

function

$$m_M : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \frac{1}{M}\sum_{i=1}^{M}(Y_i - y_l \mathbf{1}_{\{X_i \leq x\}} - y_r \mathbf{1}_{\{X_i > x\}})^2. \quad (6.1)$$

With probability $q > 0$ [1], choose x as the midpoint $x = \frac{1}{2}$ and solve the minimisation problem $\inf_{y_l,y_r} m_M(y_l, y_r, \frac{1}{2})$. With probability $0 < 1 - q < 1$, solve the minimisation problem $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$. For a fixed x^*, the optimal values of y_l and y_r are given by

$$y_r^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i > x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i > x^*\}}}; \quad y_l^* = \frac{\sum_{i=1}^{M} Y_i \mathbb{1}_{\{X_i \leq x^*\}}}{\sum_{i=1}^{M} \mathbb{1}_{\{X_i \leq x^*\}}}. \quad (6.2)$$

The problem $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$ may admit more than one minimizer. To make the tree unique, we choose the minimizer (y_l^*, y_r^*, x^*) with the minimal third component, ie any solution $(y_l^\dagger, y_r^\dagger, x^\dagger)$ to $\inf_{y_l,y_r,x} m_M(y_l, y_r, x)$ satisfies $x^\dagger \geq x^*$. Once the threshold x^* is determined, we split the samples into two groups following the sign of $X_i - x^*$ and repeat the process for each group. We stop the process if introducing a new leaf does not improve the MSE (ie when x^* is on the boundary of the interval on which we solve the minimisation problem) or when enough iterations have been made. In the end, we have a tree that approximates the conditional expectation with a piecewise constant function. The regression trees are an algorithmic tool to find an adapted partition and the corresponding weights of this piecewise constant function. Note that we can also solve a global optimization problem to find such a piece wise constant function, but the computation will be costly. The regression tree does not yield a solution to such a global optimisation problem. In fact, if we are, for instance, looking for a piece wise constant function with 4 values $[0, 1]$, The global optimisation problem would write

$$\min_{x_1,x_2,x_3,v_1,v_2,v_3,v_4} \frac{1}{M}\sum_{i=1}^{M}(Y_i - v_1\mathbb{1}_{0<X_i\leq x_1} - v_2\mathbb{1}_{x_1<X_i\leq x_2} - v_3\mathbb{1}_{x_2<X_i\leq x_3} - v_4\mathbb{1}_{x_3<X_i\leq 1})^2,$$

whereas the regression tree algorithm will first find an approximation with one cut which will not necessarily coincide with x_1^*, x_2^* or x_3^*. Then, it will search for a solution with one cut on each of the new intervals iteratively. In fact, a regression tree will always perform only one cut on each iteration.
In the multi-dimensional case, we choose the direction (the index along which the optimization is performed) uniformly for each new split. Then, the process is iterated as in the one-dimensional case.

We denote the resulting tree by $\hat{\mathcal{T}}_p^M(X)$ where p represents the depth of the tree, i.e., the number of iterations done in the process described above. A tree of depth p has 2^p leaves.

[1] The introduction of q simplifies the calculations. In the numerical experiments, we will set $q = 0$

When the size of the training data is infinite, we use the same procedure as above but with expectations rather than empirical means. Consider the function

$$m : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [0,1] \longmapsto \mathbb{E}\left[(Y - y_l \mathbf{1}_{\{X \leq x\}} - y_r \mathbf{1}_{\{X > x\}})^2\right]. \tag{6.3}$$

Equation (6.2) becomes

$$y_r^* = \mathbb{E}\left[Y | X > x^*\right]; \quad y_l^* = \mathbb{E}\left[Y | X \leq x^*\right].$$

Again, when we optimize the value of x^*, we choose the minimizer (y_l^*, y_r^*, x^*) with the minimal third component. We denote the tree of depth p obtained with an infinite data set by $\mathcal{T}_p(X)$.

6.2.2 Some key properties of regression trees

For $p \in \mathbb{N}$, let $\mathcal{A} = \left(\prod_{j=1}^d \left[a_p^{i-1}(j), a_p^i(j)\right)\right)_{1 \leq i \leq 2^p}$ be a partition of $[0,1]^d$ with 2^p elements. We write

$$\left[a_p^{i-1}, a_p^i\right) := \prod_{j=1}^d \left[a_p^{i-1}(j), a_p^i(j)\right)$$

For $(\alpha_p^i)_{1 \leq i \leq 2^p} \in \mathbb{R}^{2^p}$, we define $\mathcal{P}_p$ as the piecewise constant function on the partition $\mathcal{A}$ with values α_p^i. For $x \in [0,1]^d$,

$$\mathcal{P}_p(x, (a_p^i)_{0 \leq i \leq 2^p}, (\alpha_p^i)_{1 \leq i \leq 2^p}) = \sum_{i=1}^{2^p} \alpha_p^i \mathbb{1}_{\{x \in [a_p^{i-1}, a_p^i)\}}$$

If $\mathcal{A}$ denotes the partition obtained in the regression tree $\mathcal{T}_p$, and we choose

$$\alpha_p^i = \mathbb{E}\left[Y | X \in \left[a_p^{i-1}, a_p^i\right)\right],$$

then the regression tree $\mathcal{T}_p(X)$ can be written as follows

$$\mathcal{T}_p(X) = \mathcal{P}_p(X, (a_p^i)_{0 \leq i \leq 2^p}, (\alpha_p^i)_{1 \leq i \leq 2^p}) \tag{6.4}$$

Similarly, if $\mathcal{A}^M = \left(\left[a_p^{i-1,M}, a_p^{i,M}\right)\right)_{1 \leq i \leq 2^p}$ denotes the partition obtained in the regression tree $\hat{\mathcal{T}}_p^M$ and

$$\alpha_p^{i,M} = \frac{\sum_{m=1}^M Y_m \mathbf{1}_{\left\{X_m \in \left[a_p^{i-1}, a_p^i\right)\right\}}}{\sum_{m=1}^M \mathbf{1}_{\left\{X_m \in \left[a_p^{i-1,M}, a_p^{i,M}\right)\right\}}},$$

then the regression tree $\hat{\mathcal{T}}_p^M(X)$ can be written as follows

$$\hat{\mathcal{T}}_p^M(X) = \mathcal{P}_p(X, (a_p^{i,M})_{0 \leq i \leq 2^p}, (\alpha_p^{i,M})_{1 \leq i \leq 2^p}) \tag{6.5}$$

Once the partition of the tree is computed, the values associated to each element of the partition are uniquely determined by a global optimization problem

Proposition 6.2.1. *Let* $\mathcal{A} = \left(\prod_{j=1}^{d} \left[a_p^{i-1}(j), a_p^i(j)\right)\right)_{1\leq i\leq 2^p}$ *denote the partition associated to* $\mathcal{T}_p$*. Then,*

$$\mathbb{E}\left[(\mathcal{T}_p(X) - \mathbb{E}[Y|X])^2\right] = \inf_{(\alpha_p^i)_{1\leq i\leq 2^p}} \mathbb{E}\left[\left(\mathcal{P}_p(X, (a_p^i)_{0\leq i\leq 2^p}, (\alpha_p^i)_{1\leq i\leq 2^p}) - \mathbb{E}[Y|X]\right)^2\right]$$

and moreover the right hand side admits a unique minimizer given by

$$\alpha_p^i = \mathbb{E}\left[Y|X \in \left[a_p^{i-1}, a_p^i\right)\right].$$

Proof. Since the function $\mathcal{P}_p$ is linear w.r.t the parameter α, the function

$$\alpha \in \mathbb{R}^{2^p+1} \longmapsto \mathbb{E}\left[\left(\mathcal{P}_p(X, (a_p^i)_{0\leq i\leq 2^p}, (\alpha_p^i)_{1\leq i\leq 2^p}) - \mathbb{E}[Y|X]\right)^2\right]$$

is strongly convex. Hence, it admits a unique minimizer defined by the first order optimality condition stating that for all $0 \leq i \leq 2^p + 1$,

$$\begin{aligned}
&\mathbb{E}\left[\left(\mathcal{P}_p(X, (a_p^i)_{0\leq i\leq 2^p}, (\alpha_p^i)_{1\leq i\leq 2^p}) - \mathbb{E}[Y|X]\right)\mathbf{1}_{\left\{X\in\left[a_p^{i-1}, a_p^i\right)\right\}}\right] = 0\\
&\mathbb{E}\left[\left(\alpha_p^i - \mathbb{E}[Y|X]\right)\mathbf{1}_{\left\{X\in\left[a_p^{i-1}, a_p^i\right)\right\}}\right] = 0\\
&\alpha_p^i = \mathbb{E}\left[Y|X \in \left[a_p^{i-1}, a_p^i\right)\right].
\end{aligned}$$

□

A similar result holds for $\hat{\mathcal{T}}_p^M$ by replacing the conditional expectations by empirical conditional expectations.

We define

$$J_p\left(a_p^0, \ldots, a_p^p\right) := \mathbb{E}\left[\left|\sum_{i=1}^{p} \alpha_p^i \mathbb{1}_{\{X\in\left[a_p^{i-1} - a_p^i\right)\}} - \mathbb{E}\left[Y|X\right]\right|^2\right].$$

with $\alpha_p^i = \mathbb{E}[Y|X \in [a_p^{i-1} - a_p^i)]$.

Lemma 6.2.2. *Let* X *be a r.v with a density* f_X *w.r.t to the Lebesgue measure on* $[0,1]^d$ *and* Y *be a real valued square integrable random variable. Let* $\left(\left(\left[a_p^{i-1}, a_p^i\right)\right)_{1\leq i\leq p}\right)_{p\in\mathbb{N}}$ *be a sequence of partitions of* $[0,1]^d$ *such that* $\lim_{p\to\infty} \max_{1\leq i\leq p} \max_{1\leq j\leq d} \left|a_p^i(j) - a_p^{i-1}(j)\right| = 0$*. Then,*

$$\lim_{p\to\infty} J_p\left(a_p^0, \ldots, a_p^p\right) = 0.$$

Proof. Consider the function

$$\begin{aligned}
f : [0,1]^d &\to \mathbb{R}\\
x &\mapsto \mathbb{E}\left[Y|X = x\right]
\end{aligned}$$

and define a piecewise constant approximation of f

$$x \mapsto f^{(p)}(x) = \sum_{i=1}^{p} \left(\frac{1}{\mu\left(\left[a_p^{i-1}, a_p^i\right]\right)} \int_{\left[a_p^{i-1}, a_p^i\right]} f(s)ds \right) \mathbb{1}_{\{x \in \left[a_p^{i-1}, a_p^i\right)\}}$$
$$= \sum_{i=1}^{p} \alpha_p^i \mathbb{1}_{\{x \in \left[a_p^{i-1}, a_p^i\right)\}}$$

with μ the Lebesgue measure. First, we consider that f is continuous on $[0,1]^d$. Then, it is uniformly continuous on the compact set $[0,1]^d$. So,

$$\forall \epsilon > 0, \exists \eta > 0 \text{ s.t } \forall x, y \; |x - y| < \eta, |f(x) - f(y)| < \epsilon$$

Let $\epsilon > 0$ and $\eta > 0$ satisfying the above condition. For a large enough p

$$\max_{1 \leq i \leq p} \max_{1 \leq j \leq d} \left| a_p^i(j) - a_p^{i-1}(j) \right| < \sqrt{\frac{\eta}{d}}.$$

So,

$$\forall 1 \leq i \leq p, \; \left| a_p^{i-1} - a_p^i \right| = \sum_{j=1}^{d} \left| a_p^i(j) - a_p^{i-1}(j) \right|^2 < \eta.$$

Then,

$$\mathbb{E}\left[\left| f(X) - f^{(p)}(X) \right|^2 \right] = \int_{[0,1]^d} \left| f(x) - f^{(p)}(x) \right|^2 f_X(x)dx$$
$$= \sum_{i=1}^{p} \int_{[a_p^{i-1}, a_p^i]} \left| f(x) - \frac{1}{\mu([a_p^{i-1}, a_p^i])} \int_{[a_p^{i-1}, a_p^i]} f(s)ds \right|^2 f_X(x)dx$$
$$\leq \sum_{i=1}^{p} \int_{[a_p^{i-1}, a_p^i]} \frac{1}{\mu([a_p^{i-1}, a_p^i])} \int_{[a_p^{i-1}, a_p^i]} |f(x) - f(s)|^2 ds f_X(x)dx$$
$$\leq \sum_{i=1}^{p} \int_{[a_p^{i-1}, a_p^i]} \epsilon^2 f_X(x)dx \leq \epsilon^2.$$

Finally,

$$\lim_{p \to \infty} \int_{[0,1]^d} \left| f(x) - f^{(p)}(x) \right|^2 f_X(x)dx = 0.$$

The set of continuous functions on $[0,1]^d$ being dense in $\mathbb{L}^2\left([0,1]^d\right)$, the result still holds without the continuity assumption, which ends the proof. □

Theorem 6.2.3. *The tree $\mathcal{T}_p$ defined in Section 6.2.1 satisfies*

$$\lim_{p \to \infty} \mathbb{E}\left[\left| \mathcal{T}_p(X) - \mathbb{E}[Y|X] \right|^2 \right] = 0.$$

Proof. For all $1 \leq j \leq d$

$$
\begin{aligned}
&\mathbb{E}\left[\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right|\right]\\
&\leq \frac{1}{d}\left[q\frac{1}{2}\mathbb{E}\left[\max_{1\leq i\leq 2^{p-1}}\left|a_{p-1}^i(j)-a_{p-1}^{i-1}(j)\right|\right]+(1-q)\mathbb{E}\left[\max_{1\leq i\leq 2^{p-1}}\left|a_{p-1}^i(j)-a_{p-1}^{i-1}(j)\right|\right]\right]\\
&+\frac{d-1}{d}\mathbb{E}\left[\max_{1\leq i\leq 2^{p-1}}\left|a_{p-1}^i(j)-a_{p-1}^{i-1}(j)\right|\right]\\
&\leq (1-\frac{q}{2d})\mathbb{E}\left[\max_{1\leq i\leq 2^{p-1}}\left|a_{p-1}^i(j)-a_{p-1}^{i-1}(j)\right|\right]\\
&\leq (1-\frac{q}{2d})^{2^p}
\end{aligned}
$$

In fact, with a probability $\frac{1}{d}$, the index j is chosen for optimisation. In the other $d-1$ cases, we do not even cut along that direction, in which case the interval length is at most equal to the length of the largest interval at step $p-1$. When the index j is chosen: with probability q, the length of the interval is cut in two, and with probability $1-q$, it is cut to optimize the MSE. In that case, the interval length is at most equal to the length of the largest interval at step $p-1$. Since $\sum_{p=0}^{\infty}(1-\frac{q}{2d})^{2^p} < \infty$, so is $\sum_{p=0}^{\infty}\mathbb{E}\left[\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right|\right]$. As $\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right|$ is non negative for all p, using Tonelli's theorem we conclude that $\mathbb{E}\left[\sum_{p=0}^{\infty}\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right|\right] < \infty$. As a result, the series $\sum_{p=0}^{\infty}\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right|$ converges a.s. Then, $\lim_{p\to\infty}\max_{1\leq i\leq 2^p}\left|a_p^i(j)-a_p^{i-1}(j)\right| = 0$ a.s for all j, and $\lim_{p\to\infty}\max_{1\leq i\leq 2^p}\max_{1\leq j\leq d}\left|a_p^i(j)-a_p^{i-1}(j)\right| = 0$.

Let $\mathcal{G}$ be the σ-field generated by the splitting strategy (direction choice and threshold strategy). Conditioning by $\mathcal{G}$ allows us to consider the partition $\left([a_p^{i-1}-a_p^i)\right)_{1\leq i\leq 2^p}$ deterministic and we can apply Lemma 6.2.2 to prove

$$\lim_{p\to\infty}\mathbb{E}\left[|\mathcal{T}_p(X)-\mathbb{E}\left[Y|X\right]|^2|\mathcal{G}\right]=0 \text{ a.s.}$$

Note that

$$
\begin{aligned}
\mathbb{E}\left[|\mathcal{T}_p(X)|^2|\mathcal{G}\right] &\leq \mathbb{E}\left[\sum_{i=1}^{2^p}\mathbb{E}\left[Y^2|X\in[a_p^{i-1},a_p^i)\right]\mathbb{1}_{\{X\in[a_p^{i-1},a_p^i)\}}|\mathcal{G}\right]\\
&\leq \sum_{i=1}^{2^p}\mathbb{E}\left[\mathbb{E}\left[Y^2\mathbb{1}_{\{X\in[a_p^{i-1},a_p^i)\}}|X\in[a_p^{i-1},a_p^i)\right]|\mathcal{G}\right]\\
&\leq \sum_{i=1}^{2^p}\mathbb{E}\left[Y^2\mathbb{1}_{\{X\in[a_p^{i-1},a_p^i)\}}|\mathcal{G}\right]\\
&\leq \mathbb{E}\left[Y^2|\mathcal{G}\right]
\end{aligned}
$$

Then,

$$\begin{aligned}\mathbb{E}\left[|\mathcal{T}_p(X) - \mathbb{E}[Y|X]|^2|\mathcal{G}\right] &\leq 2\mathbb{E}\left[|\mathcal{T}_p(X)|^2|\mathcal{G}\right] + 2\mathbb{E}\left[\mathbb{E}[Y|X]^2\right] \\ &\leq 2\left(\mathbb{E}\left[Y^2|\mathcal{G}\right] + \mathbb{E}\left[\mathbb{E}[Y|X]^2\right]\right)\end{aligned}$$

Using Lebesgue's bounded convergence theorem,

$$\lim_{p\to\infty} \mathbb{E}\left[|\mathcal{T}_p(X) - \mathbb{E}[Y|X]|^2\right] = \lim_{p\to\infty} \mathbb{E}\left[\mathbb{E}\left[|\mathcal{T}_p(X) - \mathbb{E}[Y|X]|^2|\mathcal{G}\right]\right] = 0. \qquad \square$$

6.3 LS algorithm with regression trees

Let T be a fixed maturity, and consider the filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{0\leq t\leq T}, \mathbb{P})$ where $\mathbb{P}$ is the risk neutral measure. Consider a Bermudan option that can be exercised at dates $0 = t_0 < t_1 < t_2 < \ldots < t_N = T$. When exercised at time t_j, the option's discounted payoff is given by $Z_{t_j} = h_j(X_{t_j})$ with $(X_{t_j})_j$ being an adapted Markov process taking values in $[0,1]^d$ such that for every j, X_{t_j} has a density f_X^j on $[0,1]^d$ uniformly bounded from below on any compact set of $(0,1)^d$ by a strictly positive number. The discounted value $(U_j)_{0\leq j\leq N}$ of this option is given by

$$U_{t_j} = \sup_{\tau\in\mathcal{T}_{t_j},T} \mathbb{E}\left[Z_\tau|\mathcal{F}_{t_j}\right]. \tag{6.6}$$

Using the Snell envelope theory, we know that U solves the dynamic programming equation

$$\begin{cases} U_{t_N} &= Z_{t_N} \\ U_{t_j} &= \max\left(Z_{t_j}, \mathbb{E}\left[U_{t_{j+1}}|\mathcal{F}_{t_j}\right]\right) \text{ for } 1 \leq j \leq N-1. \end{cases} \tag{6.7}$$

This equation can be rewritten in terms of optimal policy as follows

$$\begin{cases} \tau_N &= t_N = T \\ \tau_j &= t_j \mathbb{1}_{\left\{Z_{t_j} \geq \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right\}} + \tau_{j+1}\mathbb{1}_{\left\{Z_{t_j} < \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right\}} \text{ for } 1 \leq j \leq N-1 \end{cases} \tag{6.8}$$

where τ_j is the smallest optimal stopping time after t_j. The true price of the option at time 0 is then given by

$$U_0 = \mathbb{E}\left[Z_{\tau_0}\right] = \sup_{\tau\in\mathcal{T}_{t_0}} \mathbb{E}\left[Z_\tau\right].$$

As we are in a Markovian setting, we can write $\mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right] = \mathbb{E}\left[Z_{\tau_{j+1}}|X_{t_j}\right]$. The main difficulty in solving this equation comes from the computation of the continuation value $\mathbb{E}\left[Z_{\tau_{j+1}}|X_{t_j}\right]$. In the Least Squares approach presented by (Longstaff and Schwartz, 2001), this conditional expectation is estimated by a linear regression on a countable set of basis functions of X_{t_j}. In our approach, we suggest to estimate it using a regression tree of depth p. Let $\mathcal{T}_p^j(X_{t_j})$ be

the regression tree approximating $\mathbb{E}\left[Z_{\tau_{j+1}}|X_{t_j}\right]$ with an infinite data set. The algorithm solves the following policy

$$\begin{cases} \tau_N^p &= t_N = T \\ \tau_j^p &= t_j \mathbb{1}_{\left\{Z_{t_j} \geq \mathcal{T}_p^j(X_{t_j})\right\}} + \tau_{j+1}\mathbb{1}_{\left\{Z_{t_j} < \mathcal{T}_p^j(X_{t_j})\right\}} \text{ for } 1 \leq j \leq N-1. \end{cases} \tag{6.9}$$

We sample M paths of the model $X_{t_0}^{(m)}, \ldots, X_{t_N}^{(m)}$ along with the corresponding payoff paths $Z_{t_0}^{(m)}, \ldots, Z_{t_N}^{(m)}$, $m = 1, \ldots, M$. For each date $j = 1, \ldots, N-1$ we approximate the conditional expectations $\mathbb{E}[Z_{\tau_{j+1}}|X_{t_j}]$ on the path m using the regression tree $\hat{\mathcal{T}}_p^{j,M}(X_{t_j}^{(m)})$ built with the samples $(X_{t_j}^{(m)}, Z^m_{\hat{\tau}_{j+1}^{p,(m)}})_{1\leq m\leq M}$ where

$$\begin{cases} \hat{\tau}_N^{p,(m)} &= t_N = T \\ \hat{\tau}_j^{p,(m)} &= t_j \mathbb{1}_{\left\{Z_{t_j}^{(m)} \geq \hat{\mathcal{T}}_p^{j,M}(X_{t_j}^{(m)})\right\}} + \hat{\tau}_{j+1}^{(m)}\mathbb{1}_{\left\{Z_{t_j}^{(m)} < \hat{\mathcal{T}}_p^{j,M}(X_{t_j}^{(m)})\right\}} \text{ for } 1 \leq j \leq N-1. \end{cases} \tag{6.10}$$

Finally, the time-0 price of the option is approximated by

$$U_0^{p,M} = \max\left(Z_0, \frac{1}{M}\sum_{m=1}^{M} Z^{(m)}_{\hat{\tau}_1^{p,(m)}}\right). \tag{6.11}$$

6.4 Convergence of the algorithm

6.4.1 Notation

For each time step $1 \leq j \leq N-1$, let $\theta_j^p = ((a_{i,j}^p)_{0\leq i\leq 2^p}, (\alpha_{i,j}^p)_{1\leq i\leq 2^p})$ be the parameters of $\mathcal{T}_p^j$ and $\hat{\theta}_j^{p,M} = ((a_{i,j}^{p,M})_{0\leq i\leq 2^p}, (\alpha_{i,j}^{p,M})_{1\leq i\leq 2^p})$ be the parameters of $\mathcal{T}_{p,M}^j$. Note that with this notation and Equations (6.4) and (6.5), we have

$$\mathcal{T}_p^j(X_{t_j}) = \mathcal{P}_p(X_{t_j}, \theta_j^p) \tag{6.12}$$

$$\hat{\mathcal{T}}_p^{j,M}(X_{t_j}^{(m)}) = \mathcal{P}_p(X_{t_j}^{(m)}, \hat{\theta}_j^{p,M}) \tag{6.13}$$

where for the sake of conciseness we have shrunk the partition and weight parameters of the tree into a single multi-dimensional parameter.

We introduce the vector ϑ of the coefficients of the successive expansions $\vartheta^p = (\theta_1^p, \ldots, \theta_{N-1}^p)$ and $\hat{\vartheta}^{p,M} = (\hat{\theta}_1^{p,M}, \ldots, \hat{\theta}_{N-1}^{p,M})$.
Let $t^p = (t_1^p, \ldots, t_{N-1}^p) \in \left(([0,1]^d)^{2^p+1}\right)^{N-1}$ be a deterministic parameter, $z = (z_1, \ldots, z_N) \in \mathbb{R}^N$ and $x = (x_1, \ldots, x_N) \in ([0,1]^d)^N$ be deterministic vectors. Following the notation of (Clément et al., 2002), we define the vector field $F = F_1, \ldots, F_N$ by

$$\begin{cases} F_N(t^p, z, x) &= z_N \\ F_j(t^p, z, x) &= z_j \mathbb{1}_{\{z_j \geq \mathcal{P}_p(x_j, t_j^p)\}} + F_{j+1}(t^p, z, x)\mathbb{1}_{\{z_j < \mathcal{P}_p(x_j, t_j^p)\}}, \text{ for } 1 \leq j \leq N-1. \end{cases}$$

$F_j(t^p, z, x)$ only depends on $t^p_j, \dots, t^p_{N-1}$ and not on the first $j-1$ components of t^p. Using (6.12), we have

$$F_j(\vartheta^p, Z, X) = Z_{\tau^p_j},$$
$$F_j(\hat{\vartheta}^{p,M}, Z^{(m)}, X^{(m)}) = Z^{(m)}_{\hat{\tau}^{p,(m)}_j}.$$

Moreover, we clearly have that for all $t^p \in (([0,1]^d)^{2^p})^{N-1}$:

$$|F_j(t^p, Z, X)| \leq \max_{k \geq j} |Z_{t_k}|. \tag{6.14}$$

6.4.2 Convergence of the conditional expectations

6.4.2.1 Approximation of the conditional expectations with regression trees

Proposition 6.4.1.

$$\lim_{p \to \infty} \mathbb{E}\left[Z_{\tau^p_j} | \mathcal{F}_{t_j}\right] = \mathbb{E}\left[Z_{\tau_j} | \mathcal{F}_{t_j}\right] \ in\ \mathbb{L}^2(\Omega) \text{ for } 1 \leq j \leq N \tag{6.15}$$

Proof. We proceed by induction.
For $j = N$, the proposition is true since $\tau^p_N = \tau_N = T$. Assume that the result holds for $j+1$, Let us prove that it still holds for j:

$$\begin{aligned}
\mathbb{E}\left[Z_{\tau^p_j} - Z_{\tau_j} | \mathcal{F}_{T_j}\right] &= Z_{t_j}\left(\mathbb{1}_{\{Z_{t_j} \geq \mathcal{T}^j_p(X_{t_j})\}} - \mathbb{1}_{\{Z_{t_j} \geq \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}]\}}\right) \\
&+ \mathbb{E}\left[Z_{\tau^p_{j+1}} \mathbb{1}_{\{Z_{t_j} < \mathcal{T}^j_p(X_{t_j})\}} - Z_{\tau_{j+1}} \mathbb{1}_{\{Z_{t_j} < \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}]\}}\right] \\
&= (Z_{t_j} - \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}])\left(\mathbb{1}_{\{Z_{t_j} \geq \mathcal{T}^j_p(X_{t_j})\}} - \mathbb{1}_{\{Z_{t_j} \geq \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}]\}}\right) \\
&+ \mathbb{E}[Z^p_{\tau_{j+1}} - Z_{\tau_{j+1}} | \mathcal{F}_{t_j}] \mathbb{1}_{\{Z_{t_j} < \mathcal{T}^j_p(X_{t_j})\}} \\
&= A^p_j + \mathbb{E}[Z^p_{\tau_{j+1}} - Z_{\tau_{j+1}} | \mathcal{F}_{t_j}] \mathbb{1}_{\{Z_{t_j} < \mathcal{T}^j_p(X_{t_j})\}}
\end{aligned}$$

Where A^p_j is defined by

$$A^p_j = (Z_{t_j} - \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}])\left(\mathbb{1}_{\{Z_{j_k} \geq \mathcal{T}^j_p(X_{t_j})\}} - \mathbb{1}_{\{Z_{t_j} \geq \mathbb{E}[Z_{\tau_{j+1}} | \mathcal{F}_{t_j}]\}}\right)$$

On one hand, since the conditional expectation is an orthogonal projection, we have

$$\mathbb{E}\left[\left|\mathbb{E}\left[Z_{\tau^p_{j+1}} - Z_{\tau_{j+1}} | \mathcal{F}_{t_j}\right]\right|^2\right] \leq \mathbb{E}\left[\left|\mathbb{E}\left[Z_{\tau^p_{j+1}} - Z_{\tau_{j+1}} | \mathcal{F}_{t_{j+1}}\right]\right|^2\right]$$

and using the induction assumption $\mathbb{E}\left[Z_{\tau^p_{j+1}} - Z_{\tau_{j+1}} | \mathcal{F}_{t_j}\right] \to 0$ in $\mathbb{L}^2(\Omega)$ when $p \to \infty$. On the other hand,

$$
\begin{aligned}
\left|A_j^p\right| &= \left|Z_{t_j} - \mathbb{E}[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}]\right| \left|\mathbb{1}_{\left\{Z_{t_j} \geq \mathcal{T}_p^j(X_{t_j})\right\}} - \mathbb{1}_{\left\{Z_{t_j} \geq \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right\}}\right| \\
&\leq \left|Z_{t_j} - \mathbb{E}[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}]\right| \left|\mathbb{1}_{\left\{\mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right] > Z_{t_j} \geq \mathcal{T}_p^j(X_{t_j})\right\}} - \mathbb{1}_{\left\{\mathcal{T}_p^j(X_{t_j}) > Z_{t_j} \geq \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right\}}\right| \\
&\leq \left|Z_{t_j} - \mathbb{E}[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}]\right| \left|\mathbb{1}_{\left\{\left|Z_{t_j} - \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right| \leq \left|\mathcal{T}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right|\right\}}\right| \\
&\leq \left|\mathcal{T}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right| \\
&\leq \left|\mathcal{T}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}^p}|\mathcal{F}_{t_j}\right]\right| + \left|\mathbb{E}\left[Z_{\tau_{j+1}^p}|\mathcal{F}_{t_j}\right] - \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right|.
\end{aligned}
$$

Using the induction assumption, the second term goes to zero in $\mathbb{L}^2(\Omega)$ when $p \to \infty$. Let $([a_{i-1}(p), a_i(p)))_{1 \leq i \leq 2^p}$ be the partition generated by $\mathcal{T}_p^j$. We define

$$
\bar{\mathcal{T}}_p^j(X_{t_j}) = \sum_{i=1}^{2^p} \mathbb{E}\left[Z_{\tau_{j+1}}|X_{t_j} \in [a_{i-1}(p), a_i(p))\right] \mathbb{1}_{\{X_{t_j} \in [a_{i-1}(p), a_i(p))\}}.
$$

Note that $\bar{\mathcal{T}}_p^j$ uses the partition given by $\mathcal{T}_p^j(X_{t_j})$ but the coefficients $\alpha_i(p)$ are given by the conditional expectations of $Z_{\tau_{j+1}}$ w.r.t X_{t_j} and not those of $Z_{\tau_{j+1}^p}$. Using Proposition 6.2.1, we have the following inequality stating that $\bar{\mathcal{T}}_p^j(X_{t_j})$ is sub-optimal compared to $\mathcal{T}_p^j(X_{t_j})$

$$
\begin{aligned}
&\mathbb{E}\left[\left|\mathcal{T}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}^p}|\mathcal{F}_{t_j}\right]\right|^2\right] \\
&\leq \mathbb{E}\left[\left|\bar{\mathcal{T}}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}^p}|\mathcal{F}_{t_j}\right]\right|^2\right] \\
&\leq 2\mathbb{E}\left[\left|\bar{\mathcal{T}}_p^j(X_{t_j}) - \mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right]\right|^2\right] + 2\mathbb{E}\left[\left|\mathbb{E}\left[Z_{\tau_{j+1}}|\mathcal{F}_{t_j}\right] - \mathbb{E}\left[Z_{\tau_{j+1}^p}|\mathcal{F}_{t_j}\right]\right|^2\right].
\end{aligned}
$$

The second term goes to 0 using the induction assumption. As for the first term, note that the partition obtained with $\mathcal{T}_p^j$ verifies the conditions of Lemma 6.2.2. Then, using the same arguments as in the proof of Theorem 6.2.3, we can show that the first term also goes to 0. □

6.4.3 Convergence of the Monte Carlo approximation

In this section, the depth p of the trees is fixed and we study the convergence with respect to the number of samples M. We assume that, for all dates j, the trees $\mathcal{T}_p^j(X_{t_j})$ and $\hat{\mathcal{T}}_p^{j,M}(X_{t_j})$ are built using the same splitting strategy (ie the same splitting directions and threshold strategies).

6.4.3.1 Convergence of optimisation problems

We present three major results to study the convergence of stochastic optimization problems related to regression trees.

The first result is a uniform strong law of large numbers, see (Leake et al., 1994, Chap. 2, Lemma. A1), which can be seen as a particular case of the strong law of large numbers in Banach spaces (Ledoux and Talagrand, 1991, Corollary 7.10, page 189).

Lemma 6.4.2. *Let $(\xi_i)_{i\geq 1}$ be a sequence of i.i.d $\mathbb{R}^n$-valued random vectors and $h : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}$ be a measurable function. Assume that*

(i) For all $\bar{\theta} \in \mathbb{R}^d$, the function $\theta \in \mathbb{R}^d \mapsto h(\theta, \xi_1)$ is continuous at $\bar{\theta}$ a.s.,

(ii) $\forall C > 0, \mathbb{E}\left[\sup_{|\theta|\leq C} |h(\theta, \xi_1)|\right] < \infty$.

Then, a.s $\theta \in \mathbb{R}^d \mapsto \frac{1}{n}\sum_{i=1}^n h(\theta, \xi_i)$ converges locally uniformly to the continuous function $\theta \in \mathbb{R}^d \mapsto \mathbb{E}\left[h(\theta, \xi_1)\right]$, i.e

$$\lim_{n\to\infty} \sup_{|\theta|\leq C} \left| \frac{1}{n}\sum_{i=1}^n h(\theta, \xi_i) - \mathbb{E}\left[h(\theta, \xi_1)\right] \right| = 0 \text{ a.s.}$$

This lemma is a slight improvement of (Leake et al., 1994, Chap. 2, Lemma. A1), which was formulated under the assumption that the function $\theta \in \mathbb{R}^d \mapsto h(\theta, \xi_1)$ is almost surely continuous. However, looking closely at their proof, it turns out that it suffices to assume (i) for the conclusion to hold. Condition (i) allows the $\mathbb{P}-$null sets on which the continuity at $\bar{\theta}$ does not hold to depend on $\bar{\theta}$.

Consider a sequence of real valued functions $(f_n)_n$ defined on a compact set $K \subset \mathbb{R}^d$ such that there exists a sequence of $(x_n)_n$ satisfying

$$f_n(x_n) = \inf_{x\in K} f_n(x).$$

From (Leake et al., 1994, Chap. 2, Theorem A1), we can derive the following lemma

Lemma 6.4.3. *Assume that the sequence $(f_n)_n$ converges uniformly on K to a continuous function f. Let $v^* = \inf_{x\in K} f(x)$ and $\mathcal{S}^* = \{x \in K \ : \ f(x) = v^*\}$. Then, $f_n(x_n) \to \inf_{x\in K} f(x)$ and $d(x_n, \mathcal{S}^*) \to 0$.*

Now, we focus on a canonical minimization problem appearing at each node of the regression tree. We stick to the notation of Section 6.2.1. Let $[\underline{a}, \bar{a}]$ be a rectangle in $[0,1]^d$. For some index $1 \leq \delta \leq d$, and a real number a in $]\underline{a}^\delta, \bar{a}^\delta[$, we define the two new rectangles

$$\mathcal{R}^{\delta,l}_{\underline{a},\bar{a}}(a) = \{x \in [\underline{a}, \bar{a}] \ : \ x^\delta \leq a\}; \quad \mathcal{R}^{\delta,r}_{\underline{a},\bar{a}}(a) = \{x^\delta \in [\underline{a}, \bar{a}] \ : \ x > a\}. \tag{6.16}$$

We consider the cost functions

$$m_M : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [\underline{a}^\delta, \bar{a}^\delta] \longmapsto \frac{1}{M}\sum_{i=1}^M \left(Y_i - y_l \mathbf{1}_{\left\{X_i \in \mathcal{R}^{\delta,l}_{\underline{a},\bar{a}}(x)\right\}} - y_r \mathbf{1}_{\left\{X_i \in \mathcal{R}^{\delta,r}_{\underline{a},\bar{a}}(x)\right\}}\right)^2 \tag{6.17}$$

$$m : y_l \in \mathbb{R}, y_r \in \mathbb{R}, x \in [\underline{a}^\delta, \bar{a}^\delta] \longmapsto \mathbb{E}\left[\left(Y - y_l \mathbf{1}_{\left\{X \in \mathcal{R}^{\delta,l}_{\underline{a},\bar{a}}(x)\right\}} - y_r \mathbf{1}_{\left\{X \in \mathcal{R}^{\delta,r}_{\underline{a},\bar{a}}(x)\right\}}\right)^2\right]. \tag{6.18}$$

Let $(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ be the solution to

$$\inf_{y_l, y_r, x \in [\underline{a}^\delta + \varepsilon_0, \bar{a}^\delta - \varepsilon_0]} m_M(y_l, y_r, x) \tag{6.19}$$

with the smallest third component for some arbitrary small $\varepsilon_0 > 0$. It is clear that (6.19) may not have a unique solution. Choosing the solution with the minimal third component is a standard way to get a unique minimizer (see for instance Seijo and Sen (2011)).

Lemma 6.4.4. *Assume that the density f_X of X satisfies $f_X(x) \geq \underline{f} > 0$ for all x in any compact set of $]0,1[^d$ and that the minimization problem*

$$\inf_{y_l, y_r, x \in [\underline{a}^\delta + \varepsilon_0, \bar{a}^\delta - \varepsilon_0]} m(y_l, y_r, x) \tag{6.20}$$

has a unique minimizer (y_l^*, y_r^*, x^*).

Then, $(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ converges almost surely to (y_l^*, y_r^*, x^*) and $m_M(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ converges almost surely to $m(y_l^*, y_r^*, x^*)$ when M goes to infinity.

Proof. Consider the function

$$h(y_l, y_r, x, y, \xi) = \left(y - y_l \mathbf{1}_{\left\{\xi \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,l}(x)\right\}} - y_r \mathbf{1}_{\left\{\xi \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,r}(x)\right\}} \right)^2$$

Since X has a density on $[0,1]^d$, the function $(y_l, y_r, x) \longmapsto h(y_l, y_r, x, Y, X)$ satisfies Condition (i) of Lemma 6.4.2. Moreover, for $C > 0$

$$\mathbb{E}\left[\sup_{|y_l| \leq C, |y_r| \leq C, x \in [\underline{a}^\delta + \varepsilon_0, \bar{a}^\delta - \varepsilon_0]} |h(y_l, y_r, x, Y, X)| \right] \leq 2\mathbb{E}[Y^2] + 2C^2$$

Hence, we deduce from Lemma 6.4.2 that m_M converges a.s. locally uniformly to m. Note that for a fixed $x \in [\underline{a}^\delta + \varepsilon_0, \bar{a}^\delta - \varepsilon_0]$, the optimal values of y_l and y_r are given by

$$y_l = \mathbb{E}\left[Y | X \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,l}(x) \right]$$
$$y_r = \mathbb{E}\left[Y | X \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,r}(x) \right]$$

Observe that for $x \in [\underline{a}^\delta + \varepsilon_0, \bar{a}^\delta - \varepsilon_0]$

$$\left| \mathbb{E}\left[Y | X \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,l}(x) \right] \right| \leq \frac{\sqrt{\mathbb{E}[Y^2]}}{\mathbb{P}(X \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,l}(x))} \leq \frac{\sqrt{\mathbb{E}[Y^2]}}{\mathbb{P}(X \in \mathcal{R}_{\underline{a},\bar{a}}^{\delta,l}(a^\delta - \varepsilon_0))}$$

The uniform upper bound is finite thanks to the assumption on the density f_X of f. Hence, as a function of x, y_l is uniformly bounded. A similar results holds for y_r. Then, we deduce from Lemma 6.4.3, that $m_M(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ converges almost surely to $m(y_l^*, y_r^*, x^*)$ and $(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ converges almost surely to (y_l^*, y_r^*, x^*). □

To the best of our knowledge, without the uniqueness assumption on (y_l^*, y_r^*, x^*), we can only prove that $(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ converges in probability to (y_l^*, y_r^*, x^*). See for instance (Ferger, 2004, Corollary 1), which yields that for any $\eta > 0$, $\mathbb{P}(\hat{x}^M < x^* + \eta) \to 1$. Combining this result with $d((\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M), S^*) \to 0$ a.s. yields the convergence in probability of $(\hat{y}_l^M, \hat{y}_r^M, \hat{x}^M)$ to (y_l^*, y_r^*, x^*).

6.4.3.2 Strong law of large numbers

In order to prove the almost sure convergence of $\hat{\mathcal{T}}_{j,p}^M(X)$, we slightly modify the design of our regression trees. At every node, when computing the optimal splitting point x^*, we do not perform the optimization on the entire cell but we actually leave a margin to make sure that y_l^* and y_r^* are uniformly bounded w.r.t x^*. Consider a node at depth p, the optimal splitting is obtained by minimizing m_M defined in (6.17) where $\underline{a}$ and $\bar{a}$ were computed at depth $p-1$. As in (6.19), we search for the optimal value of x^* in $[\underline{a}^\delta - \varepsilon_p, \bar{a}^\delta - \varepsilon_p]$ where δ is the coordinate chosen for the splitting and $\varepsilon_p > 0$ is a technical security padding decreasing to 0. When there is no room left for this padding ($|\underline{a}^\delta - \bar{a}^\delta| < 2\varepsilon_p$), we stop the splitting procedure.

From (Clément et al., 2002), we have the following result

Lemma 6.4.5. *For every $j = 1, \dots, N-1$,*

$$|F_j(a, Z, X) - F_j(b, Z, X)| \leq \left(\sum_{i=j}^{N} |Z_{t_i}|\right) \left(\sum_{i=j}^{N-1} \mathbb{1}_{\{|Z_{t_i} - \mathcal{P}_p^i(X_{t_i}, b_i)| \leq |\mathcal{P}_p^i(X_{t_i}, a_i) - \mathcal{P}_p^i(X_{t_i}, b_i)|\}}\right).$$

Proposition 6.4.6. *Assume that for all $p \in \mathbb{N}^*$, and all $1 \leq j \leq N-1$, $\mathbb{P}(Z_{t_j} = \mathcal{P}_p^j(X_{t_j}, \theta_j^p)) = 0$. For all $1 \leq j \leq N-1$ and all $1 \leq i \leq 2^p$, the optimization problems*

$$\inf_{y_l, y_r, x} \mathbb{E}\left[\left(Z_{\tau_j^p} - y_l \mathbf{1}_{\left\{X_{t_j} \in \mathcal{R}^{\delta,l}_{a_p^{i-1}, a_p^i}(x)\right\}} - y_r \mathbf{1}_{\left\{X_{t_j} \in \mathcal{R}^{\delta,r}_{a_p^{i-1}, a_p^i}(x)\right\}}\right)^2\right]$$

with $x \in [a_p^{i-1,\delta} + \varepsilon_p, a_p^{i,\delta} - \varepsilon_p]$ admit a unique solution where δ is the coordinate selected by the splitting strategy.

Then, for all $j = 1, \dots, N-1$, $\hat{\theta}_j^{p,M}$ converges a.s. to θ_j^p and $\mathcal{P}_p^j(X_{t_j}, \hat{\theta}_j^{p,M})$ converges a.s. to $\mathcal{P}_p^j(X_{t_j}, \theta_j^p)$ as $M \to \infty$.

Proof. We proceed by backward induction on j with a nested forward induction on p.

Step 1: $j = N-1$

- For $p = 1$, conditionally on splitting at the best point (and not at the midpoint) the new nodes are obtained by solving

$$\mathcal{P}_1^{N-1}(X_{t_{N-1}}, \hat{\theta}_{N-1}^{1,M}) = \inf_{\alpha,\beta,a} \frac{1}{M} \sum_{m=1}^{M} \left| Z_{t_N}^{(m)} - \alpha \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in \mathcal{R}_{0,1}^{\delta,l}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in [a,1]\}} \right|^2$$

$$\mathcal{P}_1^{N-1}(X_{t_{N-1}}, \theta_{N-1}^{1}) = \inf_{\alpha,\beta,a} \mathbb{E}\left[\left| Z_{t_N} - \alpha \mathbb{1}_{\{X_{t_{N-1}} \in \mathcal{R}_{0,1}^{\delta,l}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}} \in [a,1]\}} \right|^2 \right]$$

  where $a \in [\varepsilon_1, 1 - \varepsilon_1]$. We can apply Lemma 6.4.4 to obtain $\mathcal{P}_1^{N-1}(X_{t_{N-1}}, \hat{\theta}_{N-1}^{1,M})$ converges to $\mathcal{P}_1^{N-1}(X_{t_{N-1}}, \theta_{N-1}^{1})$ a.s as $M \to \infty$ and so does $\hat{\theta}_{N-1}^{1,M}$ converge a.s. to θ_{N-1}^1. If the split is at the midpoint, the conclusion is even easier to obtain as the infimums are only computed w.r.t α and β but not a. The same situation will occur repeatedly and for the sake of clearness, we will only treat the case of splitting at the best point which is harder to handle.

- Assume that $\mathcal{P}_p^{N-1}(X_{t_{N-1}}, \hat{\theta}_{N-1}^{p,M})$ converges to $\mathcal{P}_p^{N-1}(X_{t_{N-1}}, \theta_{N-1}^{p})$ a.s and that $\hat{\theta}_{N-1}^{p,M}$ converges a.s. to θ_{N-1}^{p} as $M \to \infty$ for $p \geq 1$, we will prove it for $p+1$. For $i \in \{1, \ldots, 2^p\}$, we consider the computation of the $i-th$ node in $\hat{\mathcal{T}}_{p+1}^{j,M}$ at depth $p+1$.

$$\hat{\nu}_{p,N-1}^{M}(\alpha,\beta,a) = \frac{1}{M}\sum_{m=1}^{M} |Z_{t_N}^{(m)} - \alpha \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in \mathcal{R}^{\delta,l}_{a_{i-1,N-1}^{p,M}, a_{i,N-1}^{p,M}}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in \mathcal{R}^{\delta,r}_{a_{i-1,N-1}^{p,M}, a_{i,N-1}^{p,M}}(a)\}}|^2.$$

The parameters $\hat{\theta}_{N-1}^{p+1,M}$ are obtained by computing all the nodes. We also introduce a modified version of $\hat{\nu}_{p,N-1}^{M}$ in which we use the splits computed in $\mathcal{T}_p^j$.

$$\nu_{p,N-1}^{M}(\alpha,\beta,a) = \frac{1}{M}\sum_{m=1}^{M} |Z_{t_N}^{(m)} - \alpha \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in \mathcal{R}^{\delta,l}_{a_{i-1,N-1}^{p}, a_{i-1,N-1}^{p}}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}}^{(m)} \in \mathcal{R}^{\delta,r}_{a_{i-1,N-1}^{p}, a_{i-1,N-1}^{p}}(a)\}}|^2.$$

The random functions $\nu_{p,N-1}^{M}$ write as standard empirical means and we will prove they are close to $\hat{\nu}_{p,N-1}^{M}$ for M large.

Using Lemma 6.4.2 along with the arguments of the proof of Lemma 6.4.4, it is easy to see that the random function $\alpha, \beta, a \mapsto \nu_{p,N-1}^{M}(\alpha,\beta,a)$ converges a.s locally uniformly to the function $\alpha, \beta, a \mapsto \mathbb{E}\left[\left|Z_{t_N} - \alpha \mathbb{1}_{\{X_{t_{N-1}} \in \mathcal{R}^{\delta,l}_{a_{i-1,N-1}^{p}, a_{i,N-1}^{p}}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}} \in \mathcal{R}^{\delta,r}_{a_{i-1,N-1}^{p}, a_{i,N-1}^{p}}(a)\}}\right|^2\right]$.

$$\sup_{a\in[0,1]^d, |\alpha|\leq C, |\beta|\leq C} \left|\hat{\nu}_{p,N-1}^{M}(\alpha,\beta,a) - \nu_{p,N-1}^{M}(\alpha,\beta,a)\right|$$
$$\leq \frac{1}{M}\sum_{m=1}^{M}\left[\left|2Z_{t_N}^{(m)}\right| + 4C\right] C\left(\mathbb{1}_{\{X_{t_{N-1}}^{(m),\delta} \in (a_{i-1}^{p,M,\delta}, a_{i-1}^{p,\delta})\}} + \mathbb{1}_{\{X_{t_{N-1}}^{(m),\delta} \in (a_{i}^{p,M,\delta}, a_{i}^{p,\delta})\}}\right).$$

Using the induction assumption on p, $a_i^{p,M}$ (resp. $a_{i-1}^{p,M}$) converges a.s. to a_i^p (resp. a_{i-1}^p). Let $\epsilon > 0$,

$$\limsup_M \left|\hat{\nu}_{p,N-1}^{M}(\alpha,\beta,a) - \nu_{p,N-1}^{M}(\alpha,\beta,a)\right|$$
$$\leq \limsup_M \frac{1}{M}\sum_{m=1}^{M}\left[\left|2Z_{t_N}^{(m)}\right| + 4C\right] C\left(\left|\mathbb{1}_{\{\left|X_{t_{N-1}}^{(m),\delta} - a_i^{p,\delta}\right| \leq \epsilon\}}\right| + \left|\mathbb{1}_{\{\left|X_{t_{N-1}}^{(m),\delta} - a_{i+1}^{p,\delta}\right| \leq \epsilon\}}\right|\right)$$
$$\leq C(4C + 2\mathbb{E}\left[|2Z_{t_N}|\right])\left(\mathbb{P}(\left|X_{t_{N-1}}^{\delta} - a_{ia}^{p,\delta}\right| \leq \epsilon) + \mathbb{P}(\left|X_{t_{N-1}}^{\delta} - a_{i+1}^{p,\delta}\right|) \leq \epsilon\right).$$

Since $X_{t_{N-1}}$ has a density, $\lim_{\epsilon\to 0}\mathbb{P}(\left|X_{t_{N-1}}^{\delta} - a_i^{p,\delta}\right| \leq \epsilon) = \mathbb{P}(X_{t_{N-1}}^{\delta} = a_p^{i,\delta}) = 0$ and $\lim_{\epsilon\to 0}\mathbb{P}(\left|X_{t_{N-1}}^{\delta} - a_{i+1}^{p,\delta}\right| \leq \epsilon) = \mathbb{P}(X_{t_{N-1}}^{\delta} = a_{i+1}^{p,\delta}) = 0$. As a result,

$\left|\hat{\nu}^M_{p,N-1} - \nu^M_{p,N-1}\right| \to 0$ a.s. locally uniformly when $M \to \infty$. Thus, the random function $\hat{\nu}^M_{p,N-1}$ converges a.s. locally uniformly to the function

$$\alpha, \beta, a \mapsto \mathbb{E}\left[\left|Z_{t_N} - \alpha \mathbb{1}_{\{X_{t_{N-1}} \in \mathcal{R}^{\delta,l}_{a^p_{i-1}, a^p_i}(a)\}} - \beta \mathbb{1}_{\{X_{t_{N-1}} \in \mathcal{R}^{\delta,r}_{a^p_{i-1}, a^p_i}(a)\}}\right|^2\right].$$

Now, we apply Lemma 6.4.3 along with the same arguments of Lemma 6.4.4 to conclude that $\mathcal{P}^{N-1}_{p+1}(X_{t_{N-1}}, \hat{\theta}^{p+1,M}_{N-1})$ converges to $\mathcal{P}^{N-1}_{p+1}(X_{t_{N-1}}, \theta^{p+1}_{N-1})$ a.s as $M \to \infty$ and that $\hat{\theta}^{p+1,M}_{N-1}$ converges a.s. to θ^{p+1}_{N-1}.

Step 2: $j < N-1$.
So far, we have proved that for all p, $\mathcal{P}^{N-1}_p(X_{t_{N-1}}, \hat{\theta}^{p,M}_{N-1})$ converges to $\mathcal{P}^{N-1}_p(X_{t_{N-1}}, \theta^p_j)$ a.s and that $\hat{\theta}^{p,M}_{N-1}$ converges a.s. to θ^p_{N-1} as $M \to \infty$. Now, suppose that $\mathcal{P}^k_p(X_{t_k}, \hat{\theta}^{p,M}_k)$ (resp. $\hat{\theta}^{p,M}_k$) converges to $\mathcal{P}^k_p(X_{t_k}, \theta^p_k)$ (resp. θ^p_k) a.s as $M \to \infty$ for all p and for $k = N-1, \dots, j+1$.We should prove that these convergence results hold for j and this will be done by induction on p. Now that we have understood that considering multidimensional random variables X_{t_j} does not play any role in the proof, we will make the rest of the proof as if X were having values in $\mathbb{R}$ in order to use a little lighter notation

- For $p = 1$, consider

$$\hat{\nu}^M_{1,j}(\alpha, \beta, a) = \frac{1}{M}\sum_{m=1}^M \left|F_{j+1}\left(\hat{\vartheta}^{1,M}, Z^{(m)}, X^{(m)}\right) - \alpha \mathbb{1}_{\{X^{(m)}_{t_j} \in [0,a)\}} - \beta \mathbb{1}_{\{X^{(m)}_{t_j} \in [a,1]\}}\right|^2$$

$$\nu^M_{1,j}(\alpha, \beta, a) = \frac{1}{M}\sum_{m=1}^M \left|F_{j+1}\left(\vartheta^1, Z^{(m)}, X^{(m)}\right) - \alpha \mathbb{1}_{\{X^{(m)}_{t_j} \in [0,a)\}} - \beta \mathbb{1}_{\{X^{(m)}_{t_j} \in [a,1]\}}\right|^2.$$

The function $\nu^M_{1,j}$ writes as the sum of i.i.d random variables. Let $C \geq 0$, using Equation (6.14)

$$\begin{aligned}
&\mathbb{E}\left[\sup_{a \in [0,1]^d, |\alpha| \leq C, |\beta| \leq C} \left|F_{j+1}\left(\vartheta^1, Z, X\right) - \alpha \mathbb{1}_{\{X_{t_j} \in [0,a)\}} - \beta \mathbb{1}_{\{X_{t_j} \in [a,1]\}}\right|^2\right] \\
&\leq 2\mathbb{E}\left[\left|F_{j+1}\left(\vartheta^1, Z, X\right)\right|^2\right] + 2\mathbb{E}\left[\sup_{a \in [0,1]^d, |\alpha| \leq C, |\beta| \leq C} \left|\alpha \mathbb{1}_{\{X_{t_j} \in [0,a)\}} + \beta \mathbb{1}_{\{X_{t_j} \in [a,1]\}}\right|^2\right] \\
&\leq 2\mathbb{E}\left[\max_{l \geq j+1}(Z_{t_l})^2\right] + 2C^2 < \infty.
\end{aligned}$$

Using Lemma 6.4.2, $\alpha, \beta, a \mapsto \nu^M_{1,j}(\alpha, \beta, a)$ converges a.s locally uniformly to the function $\alpha, \beta, a \mapsto \mathbb{E}\left[\left|F_{j+1}(\vartheta^1, Z, X) - \alpha \mathbb{1}_{\{X_{t_j} \in [0,a)\}} - \beta \mathbb{1}_{\{X_{t_j} \in [a,1]\}}\right|^2\right]$. It remains to prove that $\forall C > 0 \sup_{a \in [0,1]^d, |\alpha| \leq C, |\beta| \leq C} \left|\hat{\nu}^M_{1,j}(a, \alpha, \beta) - \nu^M_{1,j}(a, \alpha, \beta)\right| \to 0$ a.s when

$M \to \infty$.
Then, using Equation (6.14) and Lemma 6.4.5

$$\sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C} \left|\hat{\nu}^M_{1,j}(a,\alpha,\beta) - \nu^M_{1,j}(a,\alpha,\beta)\right|$$
$$\leq \sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C} \frac{1}{M}\sum_{m=1}^{M}\left|F_{j+1}\left(\hat{\vartheta}^{1,M}, Z^{(m)}, X^{(m)}\right) - F_{j+1}\left(\vartheta^1, Z^{(m)}, X^{(m)}\right)\right|$$
$$\left|F_{j+1}\left(\hat{\vartheta}^{1,M}, Z^{(m)}, X^{(m)}\right) + F_{j+1}\left(\vartheta^1, Z^{(m)}, X^{(m)}\right) - 2\alpha\mathbb{1}_{\{X^{(m)}_{t_j}\in[0,a)\}} - 2\beta\mathbb{1}_{\{X^{(m)}_{t_j}\in[a,1]\}}\right|$$
$$\leq \sum_{m=1}^{M} 2\left(\max_{l\geq j+1}\left|Z^{(m)}_{t_l}\right| + 2C\right)\left|F_{j+1}\left(\hat{\vartheta}^{1,M}, Z^{(m)}, X^{(m)}\right) - F_{j+1}\left(\vartheta^1, Z^{(m)}, X^{(m)}\right)\right|$$
$$\leq \frac{1}{M}\sum_{m=1}^{M} 2\left(\max_{l\geq j+1}\left|Z^{(m)}_{t_l}\right| + 2C\right)\left(\sum_{i=j+1}^{N}\left|Z^{(m)}_{t_i}\right|\sum_{i=j+1}^{N-1}\mathbb{1}_{\{\left|Z^{(m)}_{t_i}-\mathcal{T}^i_1(X^{(m)}_{t_i})\right|\leq\left|\hat{\mathcal{T}}^{i,M}_1(X^{(m)}_{t_i})-\mathcal{T}^i_1(X^{(m)}_{t_i})\right|\}}\right).$$

Using the induction assumption on j, $\hat{\mathcal{T}}^{i,M}_1(X^{(m)}_{t_i})$ converges a.s. to $\mathcal{T}^i_1(X^{(m)}_{t_i})$ for all $N-1 \geq i \geq j+1$. Let $\epsilon < 0$.

$$\limsup_M \sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C} \left|\hat{\nu}^M_{1,j}(a,\alpha,\beta) - \nu^M_{1,j}(a,\alpha,\beta)\right|$$
$$\leq \frac{1}{M}\sum_{m=1}^{M} 2\left(\max_{l\geq j+1}\left|Z^{(m)}_{t_l}\right| + 2C\right)\left(\sum_{i=j+1}^{N}\left|Z^{(m)}_{t_i}\right|\sum_{i=j+1}^{N-1}\mathbb{1}_{\{\left|Z^{(m)}_{t_i}-\mathcal{T}^i_1(X^{(m)}_{t_i})\right|\leq\epsilon\}}\right).$$

Since $\mathbb{P}(Z^{(m)}_{t_j} = \mathcal{T}^j_p(X^{(m)}_{t_j})) = 0$, then $\lim_{\epsilon\to 0}\mathbb{1}_{\{\left|Z^m_{t_i}-\mathcal{T}^i_1(X^{(m)}_{t_i})\right|\leq\epsilon\}} = 0$ a.s and we conclude that a.s. $\left|\hat{\nu}^M_{1,j}(a,\alpha,\beta) - \nu^M_{1,j}(a,\alpha,\beta)\right|$ converges to zero uniformly. Thus, $\hat{\nu}^M_{1,j}$ converges a.s uniformly to the function $a,\alpha,\beta \mapsto \mathbb{E}\left[\left|F_{j+1}(\vartheta^1, Z, X) - \alpha\mathbb{1}_{\{X_{t_j}\in[0,a)\}} - \beta\mathbb{1}_{\{X_{t_j}\in[a,1]\}}\right|^2\right]$. Then, we apply Lemma 6.4.2 and the arguments of the proof of Lemma 6.4.4 to deduce that $\mathcal{P}^j_1(X_{t_j}, \hat{\theta}^{1,M}_j)$ converges to $\mathcal{P}^j_j(X_{t_j}, \theta^j_j)$ a.s and that $\hat{\theta}^{j,M}_j$ converges a.s. to θ^p_j as $M \to \infty$.

- We assume that $\mathcal{P}^k_p(X_{t_k}, \hat{\theta}^{p,M}_k)$ (resp. $\hat{\theta}^{p,M}_k$) converges to $\mathcal{P}^k_p(X_{t_k}, \theta^p_k)$ (resp. θ^p_k) a.s as $M \to \infty$ for some p and for $k = N-1, \ldots, j$. We will prove that it holds for $p+1$.

  Let $i \in \{1, \ldots, 2^p\}$ and consider

$$\hat{\nu}^M_{p,j}(\alpha,\beta,a) = \frac{1}{M}\sum_{m=1}^{M}\left|F_{j+1}(\hat{\vartheta}^{p,M}, Z^{(m)}, X^{(m)}) - \alpha\mathbb{1}_{\{X^{(m)}_{t_j}\in[a^{p,M}_{i-1,j},a)\}} - \beta\mathbb{1}_{\{X^{(m)}_{t_j}\in[a,a^{p,M}_{i,j})\}}\right|^2.$$
$$\nu^M_{p,j}(\alpha,\beta,a) = \frac{1}{M}\sum_{m=1}^{M}\left|F_{j+1}(\vartheta^p, Z^{(m)}, X^{(m)}) - \alpha\mathbb{1}_{\{X^{(m)}_{t_j}\in[a^{p}_{i-1,j},a)\}} - \beta\mathbb{1}_{\{X^{(m)}_{t_j}\in[a,a^{p}_{i,j})\}}\right|^2$$

The function $\nu_{p,j}^M$ writes as the sum of i.i.d random variables. Let $C \geq 0$,

$$\begin{aligned}
&\mathbb{E}\left[\sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C}\left|F_{j+1}\left(\vartheta^p, Z^{(m)}, X^{(m)}\right) - \alpha\mathbb{1}_{\{X_{t_j}^{(m)}\in[a_{i-1,j}^p,a)\}} - \beta\mathbb{1}_{\{X_{t_j}^{(m)}\in[a,a_{i,j}^p)\}}\right|^2\right] \\
&\leq 2\mathbb{E}\left[\left|F_{j+1}\left(\vartheta^p, Z^{(m)}, X^{(m)}\right)\right|^2\right] + 2C^2 \\
&\leq 2\mathbb{E}\left[\max_{l\geq j+1}(Z_{t_l})^2\right] + 2C^2 < \infty.
\end{aligned}$$

We conclude using Lemma 6.4.2 that a.s $\nu_{p,j}^M$ converges locally uniformly to the function

$$\alpha,\beta,a \mapsto \mathbb{E}\left[\left|F_{j+1}(\vartheta^p, Z, X) - \alpha\mathbb{1}_{\{X_{t_j}\in[a_{i-1,j}^p,a)\}} - \beta\mathbb{1}_{\{X_{t_j}\in(a,a_{i,j}^p)\}}\right|^2\right].$$

Let $C > 0$,

$$\begin{aligned}
&\sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C}\left|\hat{\nu}_{p,j}^M(\alpha,\beta,a) - \nu_{p,j}^M(\alpha,\beta,a)\right| \\
&\leq \sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C}\frac{1}{M}\sum_{m=1}^M\left(2\max_{l\geq j+1}\left|Z_{t_l}^{(m)}\right| + 4C\right) \\
&\left[\left(\left(\sum_{i=j+1}^N\left|Z_{t_i}^{(m)}\right|\right)\sum_{i=j+1}^{N-1}\mathbb{1}_{\{\left|Z_{t_i}-\mathcal{T}_p^i(X_{t_i}^{(m)})\right|\leq\left|\hat{\mathcal{T}}_p^{i,M}(X_{t_i}^{(m)})-\mathcal{T}_p^i(X_{t_i}^{(m)})\right|\}}\right)\right. \\
&\left.+\left|\alpha\mathbb{1}_{\{X_{t_j}^{(m)}\in[a_{i-1,j}^p,a_{i-1,j}^{p,M})\}} + \beta\mathbb{1}_{\{X_{t_j}^{(m)}\in[a_{i,j}^{p,M},a_{i,j}^p]\}}\right|\right].
\end{aligned}$$

Using the induction assumption on p, $\lim_{M\to\infty} a_{i-1,j}^{p,M} = a_{i-1,j}^p$ and $\lim_{M\to\infty} a_p^{i,M} = a_p^i$ a.s and using the induction assumption on j, $\lim_{M\to\infty}\hat{\mathcal{T}}_p^{i,M}(X_{t_i}^{(m)}) = \mathcal{T}_p^i(X_{t_i}^{(m)})\ \forall i \geq j+1$. Let $\epsilon > 0$,

$$\begin{aligned}
&\limsup_M \sup_{a\in[0,1]^d,|\alpha|\leq C,|\beta|\leq C}\left|\hat{\nu}_{p,j}^M(\alpha,\beta,a) - \nu_{p,j}^M(\alpha,\beta,a)\right| \\
&\leq \limsup_M \frac{1}{M}\sum_{m=1}^M\left(2\max_{l\geq j+1}\left|Z_{t_l}^{(m)}\right| + 4C\right) \\
&\left[C\left(\mathbb{1}_{\{\left|X_{t_j}^{(m)}-a_{i-1,j}^p\right|\leq\epsilon\}} + \beta\mathbb{1}_{\{\left|X_{t_j}^{(m)}-a_{i,j}^p\right|\leq\epsilon\}}\right) + \left(\left(\sum_{k=j+1}^N\left|Z_{t_k}^{(m)}\right|\right)\sum_{k=j+1}^{N-1}\mathbb{1}_{\{\left|Z_{t_k}^{(m)}-\mathcal{T}_p^k(X_{t_k}^{(m)})\right|\leq\epsilon\}}\right)\right].
\end{aligned}$$

Since $\lim_{\epsilon\to 0}\mathbb{1}_{\{\left|Z_{t_i}-\mathcal{T}_p^i(X_{t_i}^{(m)})\right|\leq\epsilon\}} = \lim_{\epsilon\to 0}\left|\alpha\mathbb{1}_{\{\left|X_{t_j}^{(m)}-a_{i-1,j}^p\right|\leq\epsilon\}} + \beta\mathbb{1}_{\{\left|X_{t_j}^{(m)}-a_{i,j}^p\right|\leq\epsilon\}}\right| = 0$, we conclude that a.s $\left|\hat{\nu}_{p,j}^M(\alpha,\beta,a) - \nu_{p,j}^M(\alpha,\beta,a)\right| \to 0$ locally uniformly when $M \to \infty$, and thus the random function $\hat{\nu}_{p,j}^M$ converges a.s locally uniformly to the function $\alpha,\beta,a \mapsto \mathbb{E}\left[\left|F_{j+1}(\vartheta^p, Z, X) - \alpha\mathbb{1}_{\{X_{t_j}\in[a_p^{i-1},a)\}} - \beta\mathbb{1}_{\{X_{t_j}\in(a,a_p^i)\}}\right|^2\right]$. Then we apply Lemma 6.4.3 along with the arguments of the proof of Lemma 6.4.4 to

conclude that $\mathcal{P}^k_{p+1}(X_{t_k}, \hat{\theta}^{p+1,M}_k)$ (resp. $\hat{\theta}^{p+1,M}_k$) converges to $\mathcal{P}^k_{p+1}(X_{t_k}, \theta^p_k)$ (resp. θ^{p+1}_k) a.s as $M \to \infty$ for $k = N-1, \dots, j$. This completes the induction.

□

Theorem 6.4.7. *Assume that for all $p \in \mathbb{N}^*$, and all $1 \le j \le N-1$, $\mathbb{P}(Z_{t_j} = \mathcal{T}^j_p(X_{t_j})) = 0$. Then, for $\alpha = 1, 2$ and for every $j = 1, \dots, N$,*

$$\lim_{M\to\infty} \frac{1}{M} \sum_{i=1}^{M} \left(Z^{(m)}_{\tau^{p,(m)}_j} \right)^{\alpha} = \mathbb{E}\left[(Z_{\tau^p_j})^{\alpha} \right] \text{ a.s.}$$

Proof. Note that $\mathbb{E}\left[(Z_{\tau^p_j})^{\alpha} \right] = \mathbb{E}\left[F_j(\vartheta^p, Z, X)^{\alpha} \right]$ and by the strong law of large numbers

$$\lim_{M\to\infty} \frac{1}{M} \sum_{m=1}^{M} F_j\left((\vartheta^p, Z^{(m)}, X^{(m)})^{\alpha} \right) = \mathbb{E}\left[F_j(\vartheta^p, Z, X)^{\alpha} \right] \text{ a.s.}$$

It remains to prove that

$$\Delta F_M = \frac{1}{M} \sum_{m=1}^{M} F_j(\hat{\vartheta}^{p,M}, Z^{(m)}, X^{(m)})^{\alpha} - F_j(\vartheta^p, Z^{(m)}, X^{(m)})^{\alpha} \xrightarrow[M\to\infty]{a.s} 0.$$

For any $x, y \in \mathbb{R}$, and $\alpha = 1, 2$, $|x^{\alpha} - y^{\alpha}| \le |x-y||x^{\alpha-1} + y^{\alpha-1}|$. Using Lemma 6.4.5 and Equation (6.14), we have

$$\begin{aligned}
|\Delta F_M| &\le \frac{1}{M} \sum_{m=1}^{M} \left| F_j(\hat{\vartheta}^{p,M}, Z^{(m)}, X^{(m)})^{\alpha} - F_j(\vartheta^p, Z^{(m)}, X^{(m)})^{\alpha} \right| \\
&\le 2 \frac{1}{M} \sum_{m=1}^{M} \sum_{i=j}^{N} \max_{k\ge j} \left| Z^{(m)}_{t_k} \right|^{\alpha-1} \left| Z^{(m)}_{t_i} \right| \sum_{i=j}^{N-1} \mathbb{1}_{\left\{ \left| Z^{(m)}_{t_i} - \mathcal{T}^i_p(X^{(m)}_{t_i}) \right| \le \left| \hat{\mathcal{T}}^{i,M}_p(X^{(m)}_{t_i}) - \mathcal{T}^i_p(X^{(m)}_{t_i}) \right| \right\}}.
\end{aligned}$$

Using Proposition 6.4.6, for all $i = j, \dots, N-1$, $\left| \hat{\mathcal{T}}^{i,M}_p(X_{t_i}) - \mathcal{T}^i_p(X_{t_i}) \right| \to 0$ a.s when $M \to \infty$. Then for any $\epsilon > 0$,

$$\begin{aligned}
&\limsup_M |\Delta F_M| \\
&\le 2 \limsup_M \frac{1}{M} \sum_{m=1}^{M} \sum_{i=j}^{N} \max_{k\ge j} \left| Z^{(m)}_{t_k} \right|^{\alpha-1} \left| Z^{(m)}_{t_i} \right| \sum_{i=j}^{N-1} \mathbb{1}_{\left\{ \left| Z^{(m)}_{t_i} - \mathcal{T}^i_p(X^{(m)}_{t_i}) \right| \le \epsilon \right\}} \\
&\le 2\mathbb{E}\left[\sum_{i=j}^{N} \max_{k\ge j} |Z_{t_k}|^{\alpha-1} |Z_{t_i}| \sum_{i=j}^{N-1} \mathbb{1}_{\left\{ \left| Z_{t_i} - \mathcal{T}^i_p(X_{t_i}) \right| \le \epsilon \right\}} \right].
\end{aligned}$$

We conclude that $\limsup_M |\Delta F_M| = 0$ by letting ϵ go to 0 which ends the proof.

□

6.5 Random forests

Definition 6.5.1. *A Random Forest is a collection of regression trees* $\{\mathcal{T}_{p,\Theta_k}, k = 1, \ldots\}$ *where the* $\{\Theta_k\}$ *are i.i.d random vectors. We denote the resulting forest by* $\mathcal{H}_{B,p}(X) = \sum_{k=1}^{B} \frac{1}{B} \mathcal{T}_{p,\Theta_k}(X)$ *where* B *is the number of trees in the forest and* p *the depth of the trees, and* $\mathcal{H}_p = \mathbb{E}_\Theta [\mathcal{T}_{p,\Theta}] = \lim_{B\to\infty} \mathcal{H}_{B,p}(X)$

Theorem 6.5.2.

$$\lim_{B\to\infty} \mathbb{E}\left[|Y - \mathcal{H}_{B,p}(X)|^2\right] = \mathbb{E}\left[|Y - \mathcal{H}_p(X)|^2\right]$$

See Theorem 11.1 in (Breiman, 2001).

Theorem 6.5.3.

$$\mathbb{E}\left[|Y - \mathcal{H}_p(X)|^2\right] \leq \bar{\rho}\mathbb{E}_\Theta\left[\mathbb{E}\left[|Y - \mathcal{T}_{p,\Theta}(X)|^2\right]\right]$$

where $\bar{\rho}$ *is the weighted correlation between the residuals* $Y - \mathcal{T}_{p,\Theta}(X)$ *and* $Y - \mathcal{T}_{p,\Theta'}(X)$ *and* Θ *and* Θ' *are independent. See Theorem 11.2 in (Breiman, 2001)*

Theorem 6.5.3 says that to have a good generalization error in the random forest, one should have small generalization errors in the basis trees, and the basis trees should not be highly correlated.

6.6 Numerical results

6.6.1 Description

This section studies the price of some Bermudan options using regression trees or random forests to approximate the conditional expectations. We compare the results to some reference prices and those given by the standard Longstaff Schwarz method with regression on polynomial functions. We use the Scikit-Learn library in Python, (Pedregosa et al., 2011). For regression trees, this library offers two methods of splitting: "best" to choose the best split, meaning that the split threshold is the one that minimizes the MSE and the direction for splitting is the one that gives the lowest MSE among all directions. "random" to choose the best random split, meaning that the split threshold is the one that minimizes the MSE and the direction for splitting is chosen randomly. For the following tests, we will use the latter method, which is just slightly different from what we presented in Section 6.2.1 in the way that no mid-point cuts will be considered. We also use the feature `min_samples_leaf` which allows us to set a minimum number of samples in each node. This will allow us to avoid over-fitting. For random forests, we will use the bootstrapping method (`Bootstrap=True`), meaning that for each tree in the forest, we will use a sub-sample drawn randomly and with replacement from the training data. We will also use the feature `max_samples` which allows having a specific number of data points or a percentage of the training data attributed to each tree. Having the trees trained on different data as much as possible allows us to have a low correlation between the trees which,

using Theorem 6.5.3, should make the random forest more robust.
Following the work of (Longstaff and Schwartz, 2001), we only use the in-the-money paths to learn the continuations values, which significantly improves the numerical computations. All the prices that we show are obtained after resimulation, meaning that the paths used in the estimation of the conditional expectations are not the same ones used by the Monte Carlo which means that the prices we show are unbiased.

6.6.2 Black and Scholes

Consider the Black and Scholes model

$$\begin{cases} dS_t^i & = rS_t^i dt + \sigma_i S_t^i dB_t^i, \\ d < B^i, B^j >_t & = \rho_{ij} dt. \end{cases}$$

where σ_i is the volatility of the underlying S^i, assumed to be deterministic, r is the interest rate, assumed constant, and ρ_{ij}, represents the correlation between the underlyings S^i and S^j, assumed constant.

6.6.2.1 One-dimensional put option

We consider the Bermudan put option with payoff $(K - S_\tau)^+$ with maturity $T = 1$ year, $K = 110$, $S_0 = 100$, $\sigma = 0.25$, exercisable at $N = 10$ different dates. We consider $r = 0.1$. We have a reference price for this option of 11.987 computed by a convolution method in (Lord et al., 2007). The LSM algorithm converges to the correct price with only a polynomial of degree 3. Figure 6.1, shows the price of the option when we use regression trees with a random split strategy (continuous line) or a best split strategy (dotted line) to estimate the conditional expectations. With the random strategy, the best price we get is 11.89. The case `min_samples_leaf`=1 and `max_depth`=20 gives a price of 10.5, which is far from the reference price. This result is due to over-fitting. In fact, for this case, the number of degrees of freedom is too big. The tree fits the training data too well, but it cannot generalize when confronted with new data. For the best split strategy, we obtain a slightly better price of 11.94. However, depending on the tree parameters, the price fluctuates, and we can see that the best split strategy is not necessarily better than the random split strategy. Thus, for the following, we will keep using the random split strategy. Random forests with basis trees of maximum depth 5 and minimum 100 samples in each leaf converge to the correct price with only ten trees.

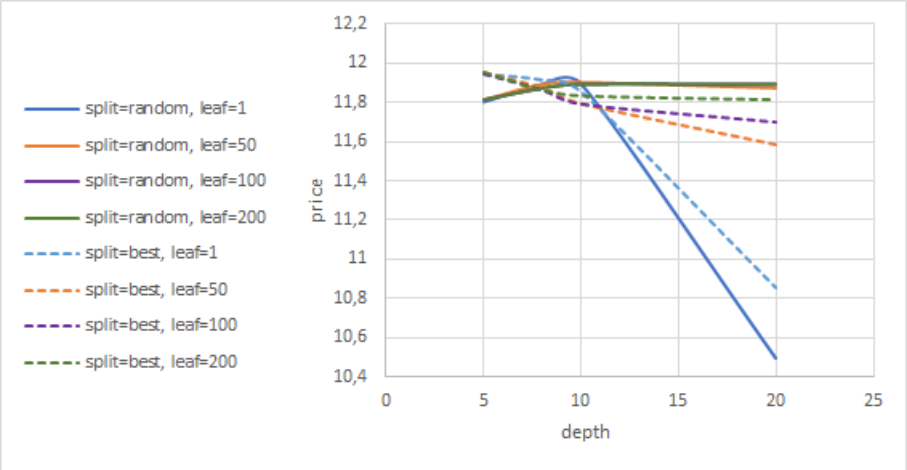


Figure 6.1: one dimensional put with regression trees, true price=11.987

6.6.2.2 Call option on the maximum of two assets

We consider a call option on the maximum of 2 assets with payoff $(\max(S_\tau^1, S_\tau^2) - K)^+$, we use the same set of parameters as in (Glasserman, 2004), for which we have reference prices of 13.90, 8.08 and 21.34 for $S_0^i = 100, 90$ and 110 respectively. The LSM algorithm using a polynomial of degree 5 converges to a price of 13.90, 8.06, 21.34 for the cases $K = 100, 90, 110$ respectively. This is a small dimensional problem, so the convergence of the LSM is expected. With regressions trees we have slightly less satisfying results as shown in Figure 6.2. We can still see the case of over-fitting when giving the regression trees too many degrees of freedom. Aggregating the regression trees into random forests immediately improves the results as shown in Figure 6.3. Note that the lower the percentage of data in each basis tree, the better the results. This confirms the results of Theorem 6.5.3 .

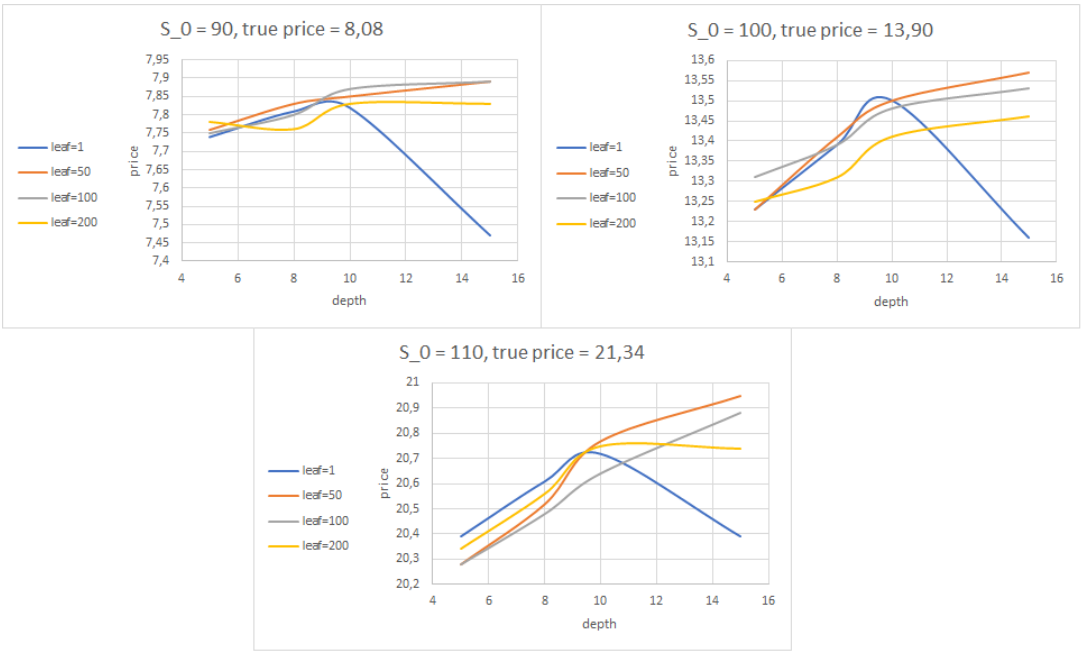


Figure 6.2: Call on the maximum of two assets with regression trees, $K = 100, T = 3$ years, $\sigma^i = 0.2, r = 0.05, \rho_{ij} = 0, \delta_i = 0.1, N = 9, M = 100,000$

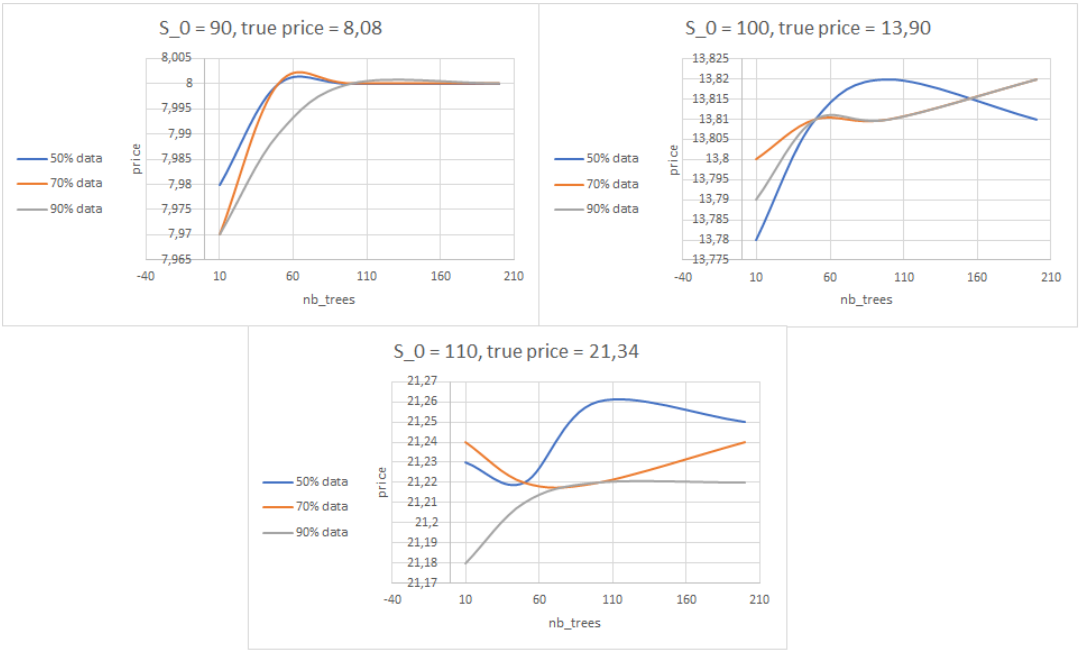


Figure 6.3: Call on the maximum of two assets with random forests, $K = 100, T = 3$ years, $\sigma^i = 0.2, r = 0.05, \rho_{ij} = 0, \delta_i = 0.1, N = 9, M = 100,000$

6.6.2.3 Geometric basket option

We consider a Bermudan Put option on a geometric basket of d underlying with payoff $\left(K - (\prod_{i=1}^{d} S_\tau^i)^{\frac{1}{d}}\right)^+$. We test the following option for $d = 2, 10, 40$ for which we have reference prices from (Cox et al., 1979) using the CRR tree method. With the LSM algorithm, we converge to the correct price 4.57 for the case $d = 2$, using only a polynomial of degree 3. For the case $d = 10$, we can at most use a polynomial of degree 3 due to the curse of dimensionality. With this parametrization, we obtain a price of 2.90 for a true price of 2.92. For the case $d = 40$, we cannot go further than a polynomial of degree 1, which yields a price of 2.48 for a reference price of 2.52. Figure 6.4 shows the results obtained with regression trees. For the case $d = 2$, the best price we get is 4.47 and, as expected, the LSM algorithm has a better performance. This is also the case for the cases $d = 10$ and $d = 40$ where the best prices we obtain are 2.84 and 2.46 respectively. Notice that even though these are high dimensional cases, the trees converge with only a depth of 5 or 8. We also notice the importance of the parameter `min_samples_leaf`. In fact, letting the trees grow without managing this parameter (case leaf1) leads to a problem of over-fitting. The results get better when we use random forests as shown in Figure 6.5. For these random forests we used basis trees of `max_depth=8` and `min_samples_leaf=100`. Notice for the case $d = 2$, the curve where only 50% of the data is used gives much better results as in this case the basis trees are the less correlated. For the cases $d = 10$ and $d = 40$, the best choice is not necessarily to use 50% of the data in each tree. As these are larger dimensions, having the trees trained on a small percentage of the training data maybe not enough. One may consider extending the size of the training data itself. Furthermore, we notice that once the percentage of data to use in each tree is chosen, the price of the option converges as the number of trees in the forest grows.

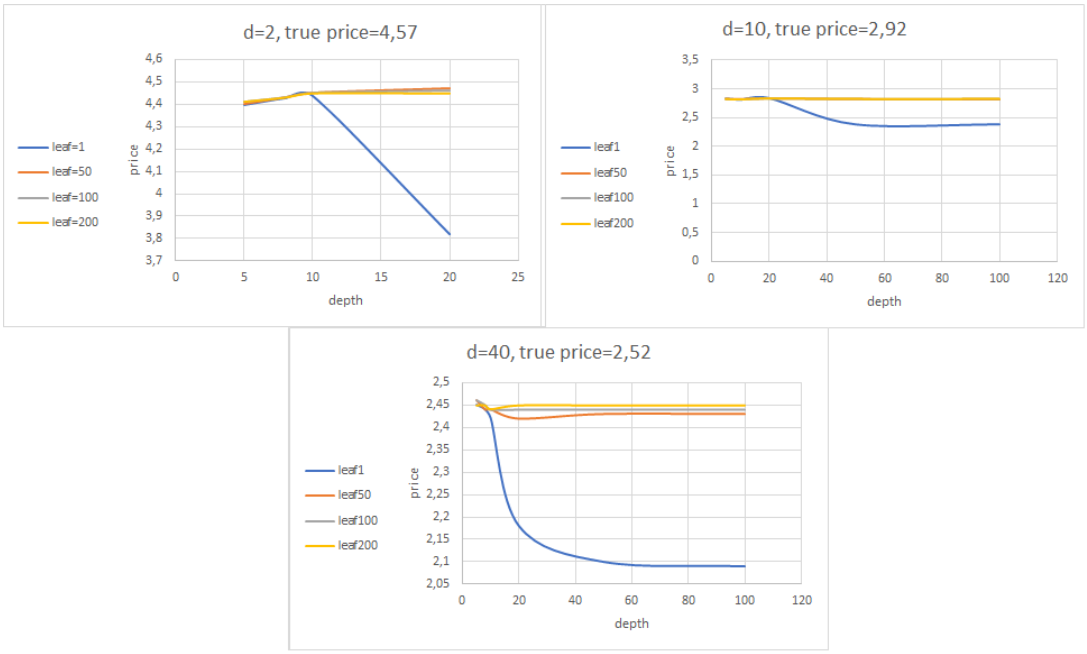


Figure 6.4: Geometric put option with regression trees

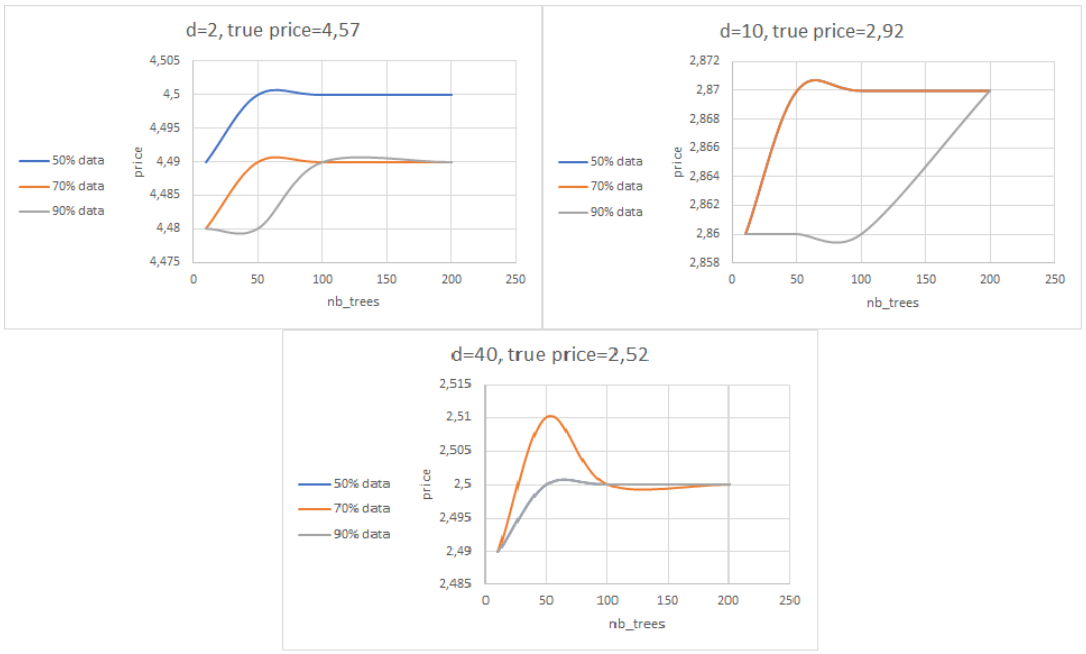


Figure 6.5: Geometric put option with random forests

6.6.2.4 A put basket option

We consider a put option on the basket of $d = 40$ asset with payoff $\left(K - \sum_{i=1}^{d} \omega_i S_T^i\right)^+$. We test this payoff for $d = 40$ for which we have a reference price from (Goudenège et al., 2019) between 2.15 and 2.22 using the following set of parameters: $T = 1, S_i = 100, K = 100, r = 0.05, \sigma_i = 0.2, \rho_{ij} = 0.2, \omega_i = \frac{1}{d}$

and $N = 10$. With a polynomial of degree 1, we obtain a price of 2.15 using the LSM algorithm. The results obtained with regression trees are shown in Figure 6.6

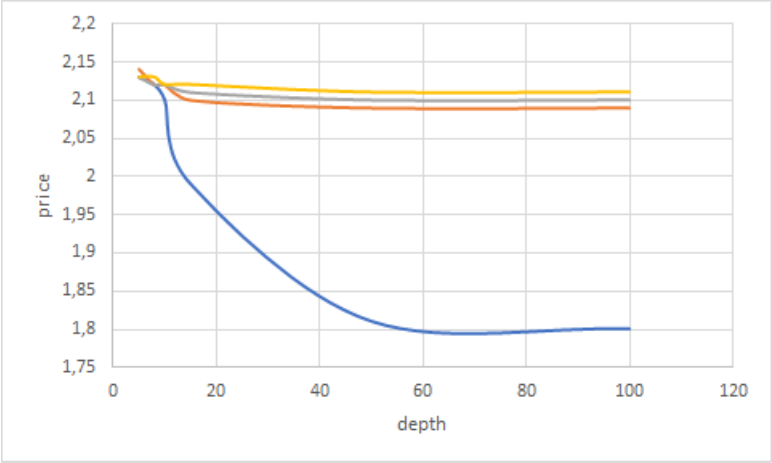


Figure 6.6: Put on a basket of 40 asset with regression trees

Even though this example is high dimensional, we do not need a lot of parameters to estimate the conditional expectations (the trees converge for very small depths). This will not be the case for the next example which is very non linear. The aggregation into random forests leads to a price of 2.16 using only 50 trees.

6.6.2.5 A call on the max of 50 asset

We consider a call option on the maximum of $d = 50$ asset with payoff $\left(\max_{1\leq i\leq d}(S_T^i) - K\right)^+$ with the following characteristics: $K = 100, T = 3$ years, $S_0^i = 100, \sigma_i = 0.2, \delta_i = 0.1, \rho_{ij} = 0\ \forall i, j, r = 0.05, N = 9, M = 100000$. (Becker et al., 2019) report $[69.56, 69.95]$ as the 95% confidence interval for the option price. With the LSM algorithm we find a price of 67.88 with a polynomial of degree 1. This a difficult example and we need to use bigger trees to approach the conditional expectations. At maturity, the payoff depends only on one direction (corresponding to the best performance), if the cuts in the tree never consider that direction, the estimation will not be correct. As a result, we consider a number of cuts big enough to ensure that each direction is taken into consideration. We allow the depth to grow while monitoring the `min_samples_leaf` in order to have a significant number of samples in each leaf. Table 6.1 shows the results obtained with regression trees. As the best price we obtain is given by depth=100 and `min_samples_leaf` = 100, we use this set of parameters for the random forest part. Table 6.2 shows the results that we obtain with this method.

depth	`min_samples_leaf`	price
50	50	66,89
50	100	66.88
100	50	67.13
100	100	67.31
200	50	67.16
200	100	67.28

Table 6.1: A call option on the maximum of 50 asset with regression trees

`nb_trees`	`max_samples`	price
10	50%	68,32
10	70%	68,32
10	90%	68,29

Table 6.2: A call option on the maximum of 50 asset with random forests

Using only regression trees is not enough to have acceptable results. However, as soon as we aggregate the regressor into random forests, we obtain very satisfying results and with just 10 trees we converge to a good price. We can also notice in this example that using uncorrelated trees leads to better results (see the case `max_samples`$= 50\%$ or 70% against the case `max_samples` $= 90\%$).

6.6.3 A put in the Heston model

We consider the Heston model defined by

$$\begin{aligned} dS_t &= S_t(r_t dt + \sqrt{\sigma_t}(\rho dW_t^1 + \sqrt{1-\rho^2} dW_t^2)) \\ d\sigma_t &= \kappa(\theta - \sigma_t)dt + \xi\sqrt{\sigma_t} dW_t^1 \end{aligned}$$

and we consider a put option with payoff $(K - S_T)^+$. we have no reference price for this option, so we will just compare the results of regression trees and random forests to the LSM method. We use the following set of parameters: $K = 100, S_0 = 100, T = 1, \sigma_0 = 0.01, \xi = 0.2, \kappa = 2, \rho = -0.3, r = 0.1, N = 10$ and $M = 100,000$. The LSM method yields a price of 1.70. Figures 6.7 and 6.8 show the results obtained with regression trees and random forests. Both methods converge to the same price of LSM. We notice for this example the occurrence of the over-fitting phenomenon for regression trees with `max_depth=15` and `min_sample_leaf=1`. We also have the same behavior for random forests in function of the percentage of data given to each basis tree.

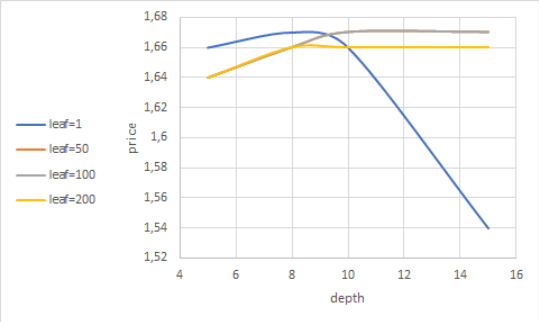


Figure 6.7: A put option in the Heston model with regression trees

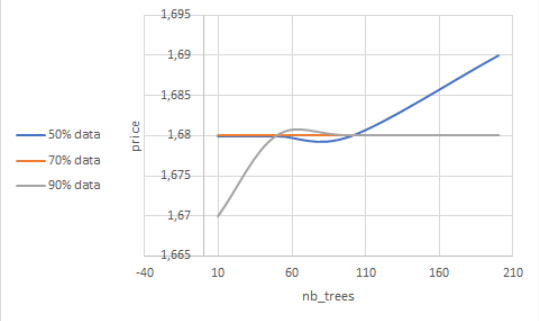


Figure 6.8: A put option in the Heston model with random forests

6.7 Conclusion

Pricing Bermudan options comes down to solving a dynamic programming equation where the main trouble comes from the computation of the conditional expectations representing the conditional expectations. We have explored the usage of regression trees and random forests for the computations of these quantities. We have proved in two steps the convergence of the algorithm when regression trees are used: first, the convergence of the conditional expectations; Then, the convergence of the Monte Carlo approximation. This problem was particularly hard to solve given that the regression trees do not solve a global optimization problem as does the functional regression used in the LSM algorithm. We have shown through numerical experiments that we obtain good prices for some classical examples using regression trees. The aggregation of regression trees into random forests yields even better results. We came to the conclusion that for small dimensional problems, a simpler algorithm like the LSM is efficient enough. However, for high dimensional problems, the usage of polynomial regressions becomes impossible as this technique suffers from the curse of dimensionality. In this case, it is interesting to consider using random forests. Instead of using all the features of the problem, the basis trees in the forest only use a subset of the features which can help combat the problem of the curse of dimensionality.

Chapter 7

Conclusion

In this thesis, we have studied the usage of machine learning in application to financial mathematics. Although, the usage of machine learning in this field has long been feared, financial researchers came up with various interesting applications that could be used safely in the financial services. As for our own contribution, we have come up with two new applications of machine learning for option pricing. The first application consists in building control variates with neural networks to reduce the variance of the Monte Carlo pricer. we exposed two methods for creating efficient control variates. The first method deals with high dimensional problems. We built a network that reduces the dimension of the random variables needed to evaluate the payoff. Then, we reconstructed the initial space using only these new variables and we evaluated the expectation of the control variate using a simple numerical quadrature integration. The second method consists in creating an automatic control variate using an adapted neural network. Given the random samples used to sample the model, we learnt through a neural network the pricing formula (which incorporates the model, the dicretization and the payoff formula). The resulting network is naturally correlated to the payoff. We used a simple architecture for the network so that it remains easy to integrate analytically. Our numerous numerical examples, for various payoffs and models, exhibited impressive speed-ups compared to the crude Monte Carlo method. The first method has proved to be robust to parameter variations and thus we used it to efficiently compute Greeks as well. This work was made concrete by the publication of the article (Ech-Chafiq et al., 2021b). In the second application, we studied the resolution of the dynamic programming equation using regression trees and random forests. We proved of the convergence of the conditional expectations approximated using regression trees to the true value of the expectations representing the continuation values in the dynamic programming equation. We also proved that for a fixed depth of the regression trees, the Monte Carlo approximation converges to the true price of the option. This problem was particularly hard to solve given that the construction of the regression trees cannot be formulated as a global optimisation problem as does the functional regression used in the original Longstaff Schwarz setting. This algorithm provides a practical solution to approximate any given expectation on a a set of multi dimensional piecewise constant functions.

Numerical experiments on our algorithm showed that regression trees price the Bermudan options accurately, and that aggregating them into random forests gives even better results. We came to the conclusion that for small dimensional problems, a simple algorithm such as Longstaff Schwarz suffices to solve the dynamic programming equation. However, for high dimensional problems, the usage of polynomial regression becomes virtually impossible due to the curse of dimensionality, in which case it becomes interesting to use a more powerful approximating tool such as random forests.

Bibliography

Esteban Alfaro, Noelia García, Matías Gámez, and David Elizondo. Bankruptcy forecasting: An empirical comparison of adaboost and neural networks. *Decision Support Systems*, 45, 2008. ISSN 01679236. doi: 10.1016/j.dss.2007.12.002.

Khaled A. Althelaya, El Sayed M. El-Alfy, and Salahadin Mohammed. Stock market forecast using multivariate analysis with bidirectional and stacked (lstm, gru). *21st Saudi Computer Society National Computer Conference, NCC 2018*, 2018. doi: 10.1109/NCG.2018.8593076.

Byungdae An and Yongmoo Suh. Identifying financial statement fraud with decision rules obtained from modified random forest. *Data Technologies and Applications*, 54, 2020. ISSN 25149288. doi: 10.1108/DTA-11-2019-0208.

Leif Andersen and Mark Broadie. Primal-dual simulation algorithm for pricing multidimensional american options. *Management Science*, 50, 2004. ISSN 00251909. doi: 10.1287/mnsc.1040.0258.

Achref Bachouch, Côme Huré, Nicolas Langrené, and Huyên Pham. Deep neural networks algorithms for stochastic control problems on finite horizon: Numerical applications. *Methodology and Computing in Applied Probability*, 24, 2022. ISSN 15737713. doi: 10.1007/s11009-019-09767-9.

Vlad Bally, Gilles Pagès, and Jacques Printems. A quantization tree method for pricing and hedging multidimensional american options. *Mathematical Finance*, 15, 2005. ISSN 09601627. doi: 10.1111/j.0960-1627.2005.00213.x.

Christian Bayer, Raúl Tempone, and Sören Wolfers. Pricing american options by exercise rate optimization. *Quantitative Finance*, 2020. ISSN 14697696. doi: 10.1080/14697688.2020.1750678.

Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20, 2019. ISSN 15337928.

Leo Breiman. Using adaptive bagging to debias regressions. *Technical Report 547*, 1, 1999. ISSN 1098-6596.

Leo Breiman. Random forests. *Machine Learning*, 45(1), 2001. ISSN 08856125. doi: 10.1023/A:1010933404324.

Mark Broadie and Paul Glasserman. Pricing american-style securities using simulation. *Journal of Economic Dynamics and Control*, 21, 1997. ISSN 01651889. doi: 10.1016/s0165-1889(97)00029-8.

Mark Broadie and Paul Glasserman. A stochastic mesh method for pricing high-dimensional american options. *The Journal of Computational Finance*, 7, 2004. ISSN 14601559. doi: 10.21314/jcf.2004.117.

H. Buehler, L. Gonon, J. Teichmann, and B. Wood. Deep hedging. *Quantitative Finance*, 19, 2019. ISSN 14697696. doi: 10.1080/14697688.2019.1571683.

Lijuan Cao and Francis E.H. Tay. Financial forecasting using support vector machines. *Neural Computing and Applications*, 10, 2001. ISSN 09410643. doi: 10.1007/s005210170010.

Carmela Cappelli, Francesca Di Iorio, Angela Maddaloni, and Pierpaolo D'Urso. Atheoretical regression trees for classifying risky financial institutions. *Annals of Operations Research*, 299, 2021. ISSN 15729338. doi: 10.1007/s10479-019-03406-9.

Jacques F. Carriere. Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics*, 19, 1996. ISSN 01676687. doi: 10.1016/S0167-6687(96)00004-2.

Marc Chataigner, Stéphane Crépey, and Matthew Dixon. Deep local volatility. *Risks*, 8, 2020. ISSN 22279091. doi: 10.3390/risks8030082.

Luyang Chen, Markus Pelger, and Jason Zhu. Deep learning in asset pricing. *SSRN Electronic Journal*, 2019. doi: 10.2139/ssrn.3350138.

Mirza Cilimkovic. Neural Networks and Back Propagation Algorithm. *Fett.Tu-Sofia.Bg*, pages 3–7, 2010.

Emmanuelle Clément, Damien Lamberton, and Philip Protter. An analysis of a least squares regression method for american option pricing. *Finance and Stochastics*, 6, 2002. ISSN 09492984. doi: 10.1007/s007800200071.

John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7, 1979. ISSN 0304405X. doi: 10.1016/0304-405X(79)90015-1.

Stéphane Crépey and Matthew Francis Dixon. Gaussian process regression for derivative portfolio modeling and application to cva computations. *SSRN Electronic Journal*, 2019. doi: 10.2139/ssrn.3325991.

Christa Cuchiero, Wahid Khosrawi, and Josef Teichmann. A generative adversarial network approach to calibration of local stochastic volatility models. *Risks*, 8, 2020. ISSN 22279091. doi: 10.3390/risks8040101.

Yann Dauphin, Harm Vries, Junyoung Chung, and Y. Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv*, 35, 02 2015.

Christophe De Luigi, Jérôme Lelong, and Sylvain Maire. Robust adaptive numerical integration of irregular functions with applications to basket and other multi-dimensional exotic options. *Applied Numerical Mathematics*, 100, 2016.

Qingshan Deng and Guoping Mei. Combining self-organizing map and k-means clustering for detecting fraudulent financial statements. *2009 IEEE International Conference on Granular Computing, GRC 2009*, 2009. doi: 10.1109/GRC.2009.5255148.

Thomas G. Dietterich. Experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40, 2000. ISSN 08856125. doi: 10.1023/A:1007607513941.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2011. ISSN 15324435.

Zineb El Filali Ech-Chafiq, Pierre Henry-Labordere, and Jérôme Lelong. Pricing bermudan options using regression trees/random forests. *ssrn*, 2021a.

Zineb El Filali Ech-Chafiq, Jérôme Lelong, and Adil Reghai. Automatic control variates for option pricing using neural networks. *Monte Carlo Methods and Applications*, 27, 2021b. ISSN 15693961. doi: 10.1515/mcma-2020-2081.

Florian Eckerli. Generative adversarial networks in finance: an overview. *SSRN Electronic Journal*, 2021. doi: 10.2139/ssrn.3864965.

Dietmar Ferger. A continuous mapping theorem for the argmax-functional in the non-unique case. *Statistica Neerlandica*, 58(1):83–96, February 2004. doi: 10.1046/j.0039-0402.2003.

Ryan Ferguson and Andrew Green. Deeply Learning Derivatives. Papers 1809.02233, arXiv.org, September 2018. URL https://ideas.repec.org/p/arx/papers/1809.02233.html.

George S. Fishman and Baosheng D. Huang. Antithetic variates revisited. *Communications of the ACM*, 26, 1983. ISSN 15577317. doi: 10.1145/182.358462.

René Garcia and Ramazan Gençay. Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics*, 94, 2000. ISSN 03044076. doi: 10.1016/S0304-4076(99)00018-4.

Paul Glasserman. *Monte Carlo method in financial engineering*, volume 4. springer, New York, 2004. doi: 10.1080/14697680400008601.

Peter W. Glynn and Roberto Szechtman. *Some New Perspectives on the Method of Control Variates*. Springer Berlin Heidelberg, 2002. doi: 10.1007/978-3-642-56046-0_3.

Emmanuel Gobet. *Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear*. Chapman & Hall/CRC, 2016. ISBN 1498746225.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3, 2014. ISSN 10495258. doi: 10.3156/jsoft.29.5_177_2.

Ludovic Goudenège, Andrea Molent, and Antonino Zanette. Variance reduction applied to machine learning for pricing bermudan/american options in high dimension, 2019.

Julien Guyon and Pierre Henry-Labordère. *Nonlinear option pricing*. Chapman and Hall/CRC, 2013. doi: 10.1201/b16332.

J. M. Hammersley and K. W. Morton. A new monte carlo technique: antithetic variates. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52 (3):449–475, 1956. doi: 10.1017/S0305004100031455.

J. Han and W. E. Deep learning approximation for stochastic control problems. *NIPS, Deep Reinforcement Learning Workshop*, 2019.

Pierre Henry-Labordere. Deep primal-dual algorithm for bsdes: Applications of machine learning to cva and im. *SSRN Electronic Journal*, 2017. doi: 10.2139/ssrn.3071506.

Pierre Henry-Labordere. Generative models for financial data. *SSRN Electronic Journal*, 2019. doi: 10.2139/ssrn.3408007.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 1998. ISSN 01628828. doi: 10.1109/34.709601.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*, 21, 2021. ISSN 14697696. doi: 10.1080/14697688.2020.1817974.

Côme Huré, Huyén Pham, Achref Bachouch, and Nicolas Langrené. Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis. *SIAM Journal on Numerical Analysis*, 59, 2021. ISSN 00361429. doi: 10.1137/20M1316640.

Ahmed Kebaier and Jérôme Lelong. Coupling importance sampling and multilevel monte carlo using sample average approximation. *Methodology and Computing in Applied Probability*, 20, 2018. ISSN 15737713. doi: 10.1007/s11009-017-9579-y.

Sujin Kim and Shane G. Henderson. Adaptive control variates. In *Proceedings - Winter Simulation Conference*, 2004.

Sujin Kim and Shane G. Henderson. Adaptive control variates for finite-horizon simulation. *Mathematics of Operations Research*, 32, 2007. ISSN 0364765X. doi: 10.1287/moor.1070.0251.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

Michael Kohler, Adam Krzyzak, and Nebojsa Todorovic. Pricing of high-dimensional american options by neural networks. *Mathematical Finance*, 20, 2010. ISSN 09601627. doi: 10.1111/j.1467-9965.2010.00404.x.

A. Kolmogoroff. Über die summen durch den zufall bestimmter unabhängiger größen. *Mathematische Annalen*, 99, 1928. ISSN 00255831. doi: 10.1007/BF01459098.

S. G. Kou. Introduction to stochastic calculus applied to finance, by damien lamberton and bernard lapeyre. *Journal of Applied Mathematics and Stochastic Analysis*, 11, 1998. ISSN 1048-9533. doi: 10.1155/s1048953398000094.

Manish Kumar and Thenmozhi M. Forecasting stock index movement: A comparison of support vector machines and random forest. *SSRN Electronic Journal*, 2011. doi: 10.2139/ssrn.876544.

Nae Won Kwak and Dong Hoon Lim. Financial time series forecasting using adaboost-gru ensemble model. *Journal of the Korean Data And Information Science Society*, 32, 2021. ISSN 1598-9402. doi: 10.7465/jkdi.2021.32.2.267.

Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3):501–506, 1992.

Bernard Lapeyre and Jérôme Lelong. A framework for adaptive monte carlo procedures. *Monte Carlo Methods and Applications*, 17, 2011. ISSN 09299629. doi: 10.1515/MCMA.2011.002.

Bernard Lapeyre and Jérôme Lelong. Neural network regression for bermudan option pricing. *Monte Carlo Methods and Applications*, 27, 2021. ISSN 15693961. doi: 10.1515/mcma-2021-2091.

Charles Leake, Reuven Y. Rubinstein, and Alexander Shapiro. Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method. *The Journal of the Operational Research Society*, 45(8), 1994. ISSN 01605682. doi: 10.2307/2584023.

Michel Ledoux and Michel Talagrand. *Probability in Banach Spaces*. Springer, Berlin, Heidelberg, 1991. doi: 10.1007/978-3-642-20212-4.

Jérôme Lelong. Dual pricing of american options by wiener chaos expansion. *SIAM Journal on Financial Mathematics*, 9, 2018. ISSN 1945497X. doi: 10.1137/16M1102161.

Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *Review of Financial Studies*, 14, 2001. ISSN 08939454. doi: 10.1093/rfs/14.1.113.

R. Lord, F. Fang, F. Bervoets, and C. W. Oosterlee. A fast and accurate fft-based method for pricing early-exercise options under levy processes. *SIAM Journal on Scientific Computing*, 30, 2007. ISSN 10648275. doi: 10.1137/070683878.

Mike Ludkovski. Kriging metamodels and experimental design for bermudan option pricing. *Journal of Computational Finance*, 22(1), 2018. ISSN 17552850. doi: 10.21314/JCF.2018.347.

Chuong Luong and Nikolai Dokuchaev. Forecasting of realised volatility with the random forests algorithm. *Journal of Risk and Financial Management*, 11, 2018. doi: 10.3390/jrfm11040061.

Sylvain Maire. Reducing variance using iterated control variates. *Journal of Statistical Computation and Simulation*, 73, 2003. ISSN 00949655. doi: 10.1080/00949650215726.

William A McGhee. An artificial neural network representation of the sabr stochastic volatility model. *SSRN Electronic Journal*, 2018. doi: 10.2139/ssrn.3288882.

Gilles Pagès and Jacques Printems. Optimal quadratic quantization for numerics: The gaussian case. *Monte Carlo Methods and Applications*, 9, 2003. ISSN 09299629. doi: 10.1163/156939603322663321.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. Pellizzari. Efficient monte carlo pricing of european options using mean value control variates. *Decisions in Economics and Finance*, 24, 2001. ISSN 11296569. doi: 10.1007/s102030170002.

FranÇois Portier and Johan Segers. Monte carlo integration with a growing number of control variates. *Journal of Applied Probability*, 56, 2019. ISSN 00219002. doi: 10.1017/jpr.2019.78.

Chi Seng Pun, Lei Wang, and Hoi Ying Wong. Financial thought experiment: A gan-based approach to vast robust portfolio selection. *IJCAI International Joint Conference on Artificial Intelligence*, 2021-January, 2020. ISSN 10450823. doi: 10.24963/ijcai.2020/637.

L. C.G. Rogers. Monte carlo valuation of american options. *Mathematical Finance*, 12, 2002. ISSN 09601627. doi: 10.1111/1467-9965.02010.

Reuven Y. Rubinstein, Gennady Samorodnitsky, and Moshe Shaked. Antithetic variates, multivariate dependence and simulation of stochastic systems. *Management Science*, 31, 1985. ISSN 00251909. doi: 10.1287/mnsc.31.1.66.

Emilio Seijo and Bodhisattva Sen. A continuous mapping theorem for the smallest argmax functional. *Electronic Journal of Statistics*, 5(none):421 – 439, 2011.

Sima Siami-Namini and Akbar Siami Namin. Forecasting economics and financial time series: Arima vs. lstm. *ArXiv*, abs/1803.06386, 2018.

I. M. Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55, 2001. ISSN 03784754. doi: 10.1016/S0378-4754(00)00270-6.

Eric H. Sorensen, Keith L. Miller, and Chee K. Ooi. The decision tree approach to stock selection. *The Journal of Portfolio Management*, 27, 2000. ISSN 0095-4918. doi: 10.3905/jpm.2000.319781.

Jan De Spiegeleer, Dilip B. Madan, Sofie Reyners, and Wim Schoutens. Machine learning for quantitative finance: fast derivative pricing, hedging and fitting. *Quantitative Finance*, 18, 2018. ISSN 14697696. doi: 10.1080/14697688.2018.1495335.

John N. Tsitsiklis and Benjamin Van Roy. Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44, 1999. ISSN 00189286. doi: 10.1109/9.793723.

Xiaoqun Wang. On the effects of dimension reduction techniques on some high-dimensional problems in finance. *Operations Research*, 54, 2006. ISSN 0030364X. doi: 10.1287/opre.1060.0334.

Xiaoqun Wang and Kai Tai Fang. The effective dimension and quasi-monte carlo integration. *Journal of Complexity*, 19, 2003. ISSN 0885064X. doi: 10.1016/S0885-064X(03)00003-7.

Xiaoqun Wang and Ian H. Sloan. Why are high-dimensional finance problems often of low effective dimension? *SIAM Journal on Scientific Computing*, 27, 2005. ISSN 10648275. doi: 10.1137/S1064827503429429.

Huanhuan Yu, Rongda Chen, and Guoping Zhang. A svm stock selection model within pca. *Procedia Computer Science*, 31, 2014. ISSN 18770509. doi: 10.1016/j.procs.2014.05.284.

Annexes

A On effective dimension

Definition A.1. *Let $\mathcal{A} = \mathcal{A}_d = \{1, 2, \dots, d\}$. for any subset $u \subseteq \mathcal{A}$, we denote by $|u|$ its cardinality and let $\mathcal{A} - u$ be its complementary in $\mathcal{A}$. Let f be a square integrable function defined on $[0,1]^d$. We say that $f(x) = \sum_{u \subseteq \mathcal{A}} f_u(x)$ is an* ***Anova decomposiotion*** *of f if for every f_u with $\emptyset \neq u \subseteq \mathcal{A}$*

$$\int_0^1 f_u(x) dx_j = 0, \text{ if } j \in u \tag{1}$$

where f_u is defined recursively by

$$\begin{cases} f_\emptyset & = I_d(f) = \int_{[0,1]^d} f(x)dx \\ f_u(x) & = \int_{[0,1]^{d-|u|}} f(x)dx_{\mathcal{A}-u} - \sum_{v \subset u} f_v(x), \forall \emptyset \neq u \subseteq \mathcal{A} \end{cases}$$

From 1, we conclude that every Anova decomposition is orthogonal, *ie* $\int_{[0,1]^d} f_u(x) f_v(x) dx = 0$ for all $u \neq v$. Thus

$$\sigma^2(f) = \sum_{\emptyset \neq u \subseteq \mathcal{A}} \sigma_u^2(f)$$

where $\sigma^2(f) = I_d(f^2) - I_d(f)^2$ is the variance of f and $\sigma_u^2(f) = \int_{[0,1]^d} f_u(x)^2 dx$ is the variance of f_u.

Let u be a non empty subset of $\mathcal{A}$. The variance corresponding to u and all its subsets is defined by

$$T_u(f) := \sum_{\emptyset \neq v \subseteq u} \sigma_v^2(f) \tag{2}$$

Definition A.2. *We say that f has* ***an effective dimension dimension*** *d_t* ***in the truncation sens (TD)*** *if*

$$T_(1, \dots, d_t) \geq p\sigma^2(f)$$

where p is a parameter close to 1.

Definition A.3. *We say that f has an* ***dimension effective*** *d_s* ***in the superposition sens (SD)*** *if*

$$\sum_{0 \leq |u| \leq d_s} \sigma_u^2(f) \geq p\sigma^2(f)$$

where p is a parameter close to 1.

B Payoff decomposition into vanilla options using Tensorflow

```
1 import numpy as np
2 from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras import Model

# Network definition
class Network(object):
    def __init__(self, payoff, model):
        self.payoff = payoff
        self.diffusion_model = model
        self.model = Sequential()
        self.input_dim = self.diffusion_model.dimension
        self.calls_layer = None
        self.puts_layer = None
        self.relu_layer = None
        self.outputs_layer = None
        self.H = None
        self.W1 = None
        self.b1 = None
        self.W2 = None
        self.b2 = None

    def build(self, n_units, activations):
        inputs = tf.keras.Input(shape=self.input_dim)
        self.calls_layer = Dense(n_units / 2, activation=tf.exp,
        use_bias=False)
        self.puts_layer = Dense(n_units / 2, activation=minus_exp,
        use_bias=False)
        x1 = self.calls_layer(inputs)
        x2 = self.puts_layer(inputs)
        x = tf.keras.layers.Concatenate()([x1, x2])
        self.relu_layer = ReluLayer(n_units)
        x = self.relu_layer(x)
        self.outputs_layer = Dense(1)
        outputs = self.outputs_layer(x)
        self.model = Model(inputs=inputs, outputs=outputs)

    def train(self, x_train, y_train, batch_size, epochs):
        self.model.compile(loss='mean_squared_error', optimizer='adam',
        metrics=['mse'])
        check_pointer = ModelCheckpoint(filepath="weights.hdf5", verbose=1,
        save_best_only=True, monitor="mse")
        self.model.fit(x_train, y_train, epochs=10, batch_size=16,
        callbacks=[check_pointer])
        self.model.load_weights("weights.hdf5")
```

```
49
50      def compute_pricing_params(self):
51          self.H = self.model.predict
52          W_calls = self.calls_layer.get_weights()[0]
53          W_puts = self.puts_layer.get_weights()[0]
54          self.W1 = np.concatenate((W_calls, W_puts), axis=1)
55          self.b1 = self.relu_layer.get_weights()[0]
56          self.W2 = self.outputs_layer.get_weights()[0]
57          self.b2 = self.outputs_layer.get_weights()[1]
58
59
60  # Customized Dense layer with Relu activation with no weight matrix.
61  class ReluLayer(tf.keras.layers.Layer):
62      def __init__(self, *args, **kwargs):
63          super(ReluLayer, self).__init__(*args, **kwargs)
64          self.bias = None
65
66      def build(self, input_shape):
67          self.bias = self.add_weight('bias',
68                      shape=input_shape[1:],
69                      initializer='zeros',
70                      trainable=True)
71          super(ReluLayer, self).build(input_shape)
72
73      def call(self, x):
74          return tf.nn.relu(x + self.bias)
75
76  # Customized activation function, needed for the put part of the decomposition
77  def minus_exp(x):
78      return - K.exp(x)
79
80  # The fast pricing method inside the Monte Carlo class
81  def fast_price(self):
82      W1 = self.network.W1
83      mu = self.network.b1
84      W2 = self.network.W2
85      b2 = self.network.b2
86      sigma = np.array([np.linalg.norm(W1[:, i])
87      for i in range(W1.shape[1])])
88      custom_layer_expect = custom_layer_expectation(mu, sigma)
89      return (custom_layer_expect @ W2 + b2)[0] *
90      np.exp(-self.model.free_risk_rate * self.payoff.maturity)
91
92  # Computes the expectation of the function (lambda * exp(sigma z) + mu)^+
93  # lambda=1 for i in [1, len(mu) / 2] and lambda=-1
94  # for i in [len(mu)/2, len(mu)]
```

```
# where z is a standard scaled normal variable
def custom_layer_expectation(mu, sigma):
    output = np.zeros(len(mu))
    for i in range(int(len(mu) / 2)):
        if mu[i] >= 0:
            output[i] = (mu[i] + exp(sigma[i] * sigma[i] * 0.5))
        else:
            output[i] = mu[i] * (1 - norm.cdf(log(-mu[i]) / sigma[i]))
            + exp(sigma[i] * sigma[i] / 2) *
            (1 - norm.cdf((log(-mu[i]) - sigma[i] * sigma[i]) / sigma[i]))
    for i in range(int(len(mu) / 2), len(mu)):
        if mu[i] <= 0:
            output[i] = 0.0
        else:
            output[i] = mu[i] * norm.cdf(log(mu[i]) / sigma[i])
            - exp(sigma[i] * sigma[i] / 2) *
            norm.cdf((log(mu[i]) - sigma[i] * sigma[i]) / sigma[i])
    return output
```