Using tree based regression to solve BSDE

Prach Siriviriyakul
Wolfson College
University of Oxford

A thesis submitted in partial fulfillment of the MSc in
Mathematical and Computational Finance

June 26, 2014

Acknowledgement

I would like to thank my supervisor Dr. Samuel Cohen who suggested me the possibility of using a tree-based method to solve BSDES and supervised me for this thesis. I am really appreciate his help and advise, which are all valuable for me throughout this project.

Abstract

Backward stochastic differential equation (BSDE) is a very important tool that appears in various areas of financial mathematics. However, there has not been any well-established schemes for solving BSDEs numerically, particularly when they are of high dimension. In this dissertation, we present a new approach, based on tree-based regression for solving BSDEs. Our numerical experiments show that this method works well even when the number of dimension is as large as 25. Further more, when applying this method to solve reflected BSDEs, we found that the instability of the scheme is very high. We present herein a way to overcome this issue. Our results suggest that tree-based method is very promising for solving BSDEs.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem formulation

This dissertation focuses on a numerical method for solving a decoupled Forward Backward Stochastic Differential Equation (FBSDE) specifically in high dimensions problem, i.e, the forward process is more than 10 dimensional. To define FBSDE, we follow the formulation in [11]. Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_t, \mathbb{P})$ be a probabilistic space with the filtration that satisfies usual conditions. Assume that the filtration is generated by a d-dimensional Brownian motion (W_t), A forward process $X : \Omega \times [0,T] \mapsto \mathbb{R}^d$ can be defined by

$$X_t = X_0 + \int_0^t \mu(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s$$

when $\mu : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$, $\sigma : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}^d \times \mathbb{R}^d$, and the integration is done component-wise.

Equivalently, we can express X_t in a derivative form as

$$\mathrm{d}X_t = \mu(t, X_t)\mathrm{d}t + \sigma(t, X_t)\mathrm{d}W_t$$

The existence and uniqueness of X_t is guaranteed when μ and σ are Lipschitz functions and have a linear growth [17].

A backward process $Y_t : \Omega \times [0,T] \mapsto \mathbb{R}$ can be defined by

$$Y_t = \Phi(X_T) - \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s + \int_t^T Z_s \mathrm{d}W_s$$

when $f : [0,T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \mapsto \mathbb{R}$ and $Z_t : \Omega \times [0,T] \mapsto \mathbb{R}^d$ is a d dimensional process where the solution to this equation exists.

Equivalently, we can express Y_t in a derivative form as a Backward Stochastic Differential Equation (BSDE)

$$\mathrm{d}Y_t = -f(t, X_t, Y_t, Z_t)\mathrm{d}t + Z_t \mathrm{d}W_t$$

The existence and uniqueness of (Y_t, Z_t) is ,again, guaranteed when Φ and f satisfy Lipschitz and linear growth conditions [16] , [9]. Under these conditions, Peng and Pardoux[16] showed that Y_t and Z_t also satisfy

$$\mathbb{E}\left[\sup_{0\leq t\leq T} |Y_t|^2 + \int_0^T |Z_t|^2 \mathrm{d}t\right] < \infty$$

1.2 Previous literature

It is generally impractical to solve FBSDE analytically, especially when the number of dimension is greater than one. Consequently, FBSDE problems are usually solved numerically. The numerical method for solving the forward equation has been well developed. Possible methods include Euler and Milstein schemes, which can solve the forward process quickly with weak error of order $\mathrm{O}(\frac{1}{N})$, when N is the number of time steps [10] . However, the numerical method for solving backward process has not been established, particularly for the problem with high dimensional process. These issues will be discussed in further later in this dissertation.

Bouchard and Touzi [3] proposed a discretization method for BSDE and showed that the error is in order $\mathrm{O}(\frac{1}{N})$, when N is the number of time steps, given that the conditional expectation required in each step of the discretization scheme can be calculated exactly. This discretization method will be explained in more detail in Chapter 2. However, in practice, these conditional expectations required at each step can be difficult to estimated. Moreover, since the error from each time step can propagate and accumulate to later steps, using a high number of time steps does not always increase the accuracy of this numerical scheme.

The common method to estimate the conditional expectation at each time step is to use regression. By simulating a large number of forward paths and using them as training data for regression, we can calculate the required conditional expectation. This idea was developed by Longstaff and Schwarz [15] in the context of American option valuation. Gobet, Lemor, and Warin [12] derived the error bounds when applying this regression method to solve FBSDE. Even though their empirical results showed that such the bound is not optimal, the authors can derive the minimum number of simulations required so that the error propagated throughout time steps

will not explode. This result is a very important benchmark and will be discussed in more detail in Chapter 2.

There have been many papers studying variation of the regression methods. Bouchard and Touzi [3] used a kernel function and Malliavin calculus approach. These method are complicated to implement but allow error bounds and the convergence of schemes to be derived. Gobet and Lemor [11] used Voronoi partitions and low order polynomials. Their empirical results showed that both models worked well when the dimension of the forward process is low, requiring a much lower number of simulation paths than theoretically expected. Bender and Steiner [2] used a basis that has a martingale property. Their numerical experiment showed that this method greatly reduces the number of simulations required for each step. However, this martingale basis is still ad hoc and depends on the structure of the BSDE. Therefore, this method is often not applicable to general problems.

Similarly to other regression problems, the curse of dimensionality is generally a very severe issue when using these numerical schemes. Voronoi partitions and polynomial regressions are impossible to use in high dimension since the size of the basis required grows exponentially as the dimension increases. This problem becomes worse when applying a numerical method to solve BSDE, since the number of simulation paths must be increased to avoid overfitting and give good estimates for the regression parameters. Therefore, it is crucial to have other methods that can avoid the curse of dimensionality.

1.3 Application of BSDE

1.3.1 Connection with PDE

There is a natural connection between FBSDE and partial differential equation (PDE), which can be proved by using the Feynman − Kac theorem and Ito formula. By using these formula, we can express the solution of a semi-linear PDE in terms of the expectation of a particular function of forward process, under different measure. By Girsamov's theorem, this is equivalent to a BSDE. Therefore, if we can solve a semi-linear PDE, we will also get a solution of the corresponding FBSDE. This is the result of the following theorem (Proposition 4.3 of El Kairoui et al [9]). Here, $X^{x,t}$ denotes a forward process started at x at time t and $Y^{x,t}$ denotes a corresponding back ward process.

Theorem 1 *Let v be a function that is smooth enough that we can apply Ito's formula to $v(t, X_t)$ ($C^{1,2}$, for example) and if (i) there exists C such that*

$$|v(t,x)| + |\sigma(t,x)^T \partial_x v(t,x)| \leq C(1+|x|)$$

for any (t,x) and (ii) v satisfies semi-linear PDE

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{d} a_{ij}(t,x)\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{d} \mu_i(t,x)\frac{\partial u}{\partial x_i} + f(t,x,u,\sigma(t,x)u) = 0 \quad \text{in}(0,T)\times\mathbb{R}^d$$

when $a_{ij} = [\frac{1}{2}\sigma\sigma^T]_{ij}$, then $v(t,x) = Y_t^{x,t}$ (a process Y_t^x started at x) is the unique solution of BSDE.

We can also get the converse of this result. For a semi-linear PDE, there is a corresponding FBSDE problem where the solution of this FBSDE also solves the original semi-linear BSDE. This property is the result of the following theorem (Pardoux and Peng 1992) [16].

Theorem 2 *If f and Phi are uniformly continuous with respect to x, then $v(t,x) = Y_t^{x,t}$ is a viscosity solution of PDE.*

This BSDE-PDE equivalence implies that a method for solving BSDE will also be a useful tool for solving PDE. This is a very important application since solving a high dimensional PDE is difficult, due to the curse of dimensionality of the finite difference scheme. Even though finite difference methods can work well in low dimension, the number of discretized grid points increases exponentially as the number of dimension increases linearly. In this case, solving the corresponding FBSDE problem would be more efficient. For example, when $f(t,x,y,z) = 0$, the BSDE reduces to a problem that can be solved by using simple Monte Carlo method. When the number of dimension is high, Monte Carlo is much more efficient than finite difference methods and hence Monte Carlo is a preferred method for solving the original PDE.

1.3.2 Financial application

FBSDE occurs in virtually all areas of financial mathematics. The general review for the application of FBSDE is in El Karoui et al [9]. The simplest example of BSDE comes from the use of a self-financing portfolio to replicate the price of financial products such as options. Let S_t be the forward process of tradable risky assets, modelled by geometric Brownian motion, with dynamics

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$$

Let B_t be risk free asset with risk free assets r and dynamics

$$\mathrm{d}B_t = rB_t \mathrm{d}t$$

Let Π_t be the value of self-financing portfolio consisting of risky asset worth ϕ_t and risk free asset worth ψ_t, i.e,

$$\Pi_t = \phi_t + \psi_t$$

Under the self-financing condition,

$$\begin{aligned}\mathrm{d}\Pi_t &= \phi_t \frac{\mathrm{d}S_t}{d_t} + \psi_t \frac{\mathrm{d}B_t}{B_t} \\ &= \phi_t \mu \mathrm{d}t + \phi_t \sigma \mathrm{d}W_t + \psi_t r \mathrm{d}t \\ &= \phi_t(\mu - r)\mathrm{d}t + r\Pi_t \mathrm{d}t + \phi_t \sigma \mathrm{d}W_t\end{aligned}$$

If we let $Z_t = \phi_t \sigma$, $Y_t = \Pi_t$, and $\theta = \frac{\mu - r}{\sigma}$, we get a BSDE

$$\mathrm{d}Y_t = (\theta Z_t + rY_t)\mathrm{d}t + Z_t \mathrm{d}W_t \tag{1.1}$$

For replication, we set its terminal condition equal to the value of some financial product at time T. The asset price is then given by Y_t. This problem can be solved by using discounted value of Y_t to find Y_0 under risk neutral measure. The problem can be reduced to a simple Monte Carlo simulation in this case.

However, this method only works in some special cases. In most cases where the driver f is non-linear, we need to solve BSDE directly. For example, when the interest rates for longing (r) and shorting (R) a risk free asset are not the same, the BSDE becomes

$$\mathrm{d}Y_t = [\theta Z_t + rY_t - (Y_t - \frac{Z_t}{\sigma})^-(R - r)]\mathrm{d}t + Z_t \mathrm{d}W_t \tag{1.2}$$

where $(\ )^-$ is the negative part. This BSDE is certainly non-linear in both Y_t and Z_t. and we need to solve it numerically.

We can also use similar idea to analyse non-tradable assets. In this incomplete market, some assets cannot be hedged. This includes the situations when the assets or indexes are impossible to be traded, such as temperature level, market indexes, and minimum wages; or when it is more preferable not to trade these assets due

to transaction cost, liquidity issues, or physical distance between the markets, for example.

There are various ways to price these non-tradable assets. One way is to use tradable assets to partially hedge these non-tradable assets and evaluate their prices subject to risk preference, or equivalently, using the utility function of an investor. Another method is to use super-hedging to get the upper prices of these assets. In either case, BSDE occurs naturally when we form portfolio and try to replicate some claims.

BSDE is also a very useful tool in stochastic optimization and control theory. For example, BSDE is one of the key parts for stochastic maximization principle or Pontyragin maximization principle. We formulate the problem following [8]. In this stochastic optimization problem, define the controlled diffusion (forward process) as

$$\mathrm{d}X_t = \mu(X_t, \alpha_t)\mathrm{d}t + \sigma(X_t, \alpha_t)\mathrm{d}W_t$$

when α_t is the control process under a particular admissible control set. The aim of stochastic optimization is to maximize

$$J(\alpha) = \mathbb{E}\left[\int_0^T h(t, X_t, \alpha_t)\mathrm{d}t + g(X_t)\right]$$

when h is continuous in (t, x) for any admissible strategy. g is concave C^1 function and both of them satisfy quadratic growth in x. Define the Hamiltonian

$$\mathcal{H}(t, x, a, y, z) = \mu(x, a)y + tr((\sigma(x, a)z)) + h(t, x, a)$$

Assuming that $\mathcal{H}$ is differentiable with respect to x, define the processes Y_t, Z_t by

$$\mathrm{dY_t} = -D_x\mathcal{H}(t, X_t, \alpha_t, Y_t, Z_t)\mathrm{d}t + Z_t\mathrm{d}W_t$$

when D_x is derivative with respective to x and with the terminal condition $Y_T = D_x g(X_T)$

The stochastic maximization principle states that if

1. the solution (Y_t^*, Z_t^*) of BSDE exists
2. $(x, a) \mapsto \mathcal{H}(t, x, \alpha_t, Y_t^*, Z_t^*)$ is concave
3. there is a control α_t^* and its corresponding diffusion process X_t^* such that $\mathcal{H}(t, x, \alpha_t, Y_t^*, Z_t^*)$ gives maximum value of $\mathcal{H}$ overall possible control

then α_t^* is also an optimal control for original problem. This stochastic maximum principle is also an example of how a BSDE can be used to solve stochastic control problem.

1.3.3 American option

Another important stochastic control problem where BSDE can be employed to is in pricing American options. Since we will consider this application in some detail and we also did numerical experiment on American options, we denote one section for it, separated from other applications. In an American style option, the holder can exercise option at any time before maturity. In order to price this option, we need to find the optimal stopping policy that results in the largest expected wealth. Since this option can be exercised at any time, its price must always be greater than or equal to the intrinsic exercise price at the exercising time. When the option price is greater than the exercise price, the early exercise feature at that time is useless and the option price follows the same dynamics as the European one.

One way to solve this problem is to use a reflected BSDE (RBSBE). In the following specification of RBSDE, we follow Gobet et al [12] . For terminal condition, $\Phi(t, X_t)$ (note that unlike previous case, Φ now also depend on time), the solution (Y, Z, A) of the RBSDE satisfies

$$
\begin{aligned}
&Y_t = \Phi(T, X_T) - \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s + \int_t^T Z_s \mathrm{d}W_s + A_T - A_t \\
&Y_t \geq \Phi(t, x_t) \\
&A_0 = 0, \int_0^T (Y_s - \Phi(s, X_s))\mathrm{d}A_s = 0
\end{aligned}
$$

when A_t is a continuous, and increasing process. In order to guarantee the existence and uniqueness of the solution to this RBSDE, we need the following conditions in addition to those for simple FBSDE, stated before.

- μ and σ in the forward process must be continuously differentiable with uniformly bounded derivatives with respective to x. $\sigma\sigma^T \geq \epsilon I, \epsilon > 0$ when I is identity matrix.

- For any $(t_1, x_1, y_1, z_1), (t_2, x_2, y_2, z_2) \in [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$, there exist C such that $|f(t_1, x_1, y_1, z_1) - f(t_2, x_2, y_2, z_2)| \leq C(|t_1 - t_2|^{\frac{1}{2}} + |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|)$ and $|\Phi(t_1, x_1) - \Phi(t_2, x_2)| \leq C(|t_1 - t_2|^{\frac{1}{2}} + |x_1 - x_2|)$

As shown in previous section, when forming a portfolio to replicate the value of some option, in the case of simple FBSDE, we get driver $f(t, x, y, z) = (\theta z + ry)$. When using this same driver with RBSDE, we get the price of American option with exercise payoff $\Phi(t, x)$. Unfortunately, this reflected BSDE cannot be solve analytically, except for several special case such as American call option. Therefore, we must use numerical method to solve this problem. The discretization method for RBSDE will be discussed in Chapter 3.

Chapter 2

Theoretical Background

2.1 Discretization of BSDE

The discretization of a forward process is very straight forward and well established. The simplest way to discretize this forward process is to use Euler-Maruyama scheme. For the process X_t defined in Chapter 1, the Euler-Maruyama discretization of this process is

$$\hat{X}_{t_{n+1}} = \hat{X}_{t_n} + \mu(t_n, \hat{X}_{t_n})\Delta t_n + \sigma(t_n, \hat{X}_{t_n})\Delta W_{t_n}$$

when $\hat{X}_{t_n}$ is the approximate of X_{t_n}, $\Delta t_n = t_{n+1} - t_n$, $\Delta W_{t_n} = W_{t_{n+1}} - W_{t_n}$

Bally and Talay [1] showed that for a bounded measurable European payoff f, this scheme converges and the weak error $\mathbb{E}[f(X_T)] - \mathbb{E}[f(\hat{X}_{t_N})]$ is of order $O(\sup_n(\Delta t_n))$ or $O(\frac{1}{N})$ when using equal step size with N time steps. This is an efficient scheme for discretizing the forward process and hence the forward part is not a problem for solving FBSDE, numerically.

Unlike a forward process, the discretization for BSDE is not as simple. One of the main reasons is that the estimation at each time step must be adapted to the filtration at the same time step. If we try to imitate Euler-Maruyama scheme by simply estimating the differential terms by the difference between values of process at consecutive time steps, the estimate one step back in time will not be measurable with respective to the filtration at the same time since it depends on future value. In order to overcome this problem, we need to discretize the backward scheme as

$$
\begin{aligned}
\hat{Z}_{t_n} &= \mathbb{E}\left[\hat{Y}_{t_{n+1}} \frac{\Delta W_{t_n}}{\Delta t_n} | \mathcal{F}_{t_n}\right] \\
\hat{Y}_{t_n} &= \mathbb{E}\left[\hat{Y}_{t_{n+1}} + f(t_n, \hat{X}_{t_n}, \hat{Y}_{t_n}, \hat{Z}_{t_n})\Delta t_n | \mathcal{F}_{t_n}\right] \\
\hat{Y}_{t_N} &= \Phi(X_{t_N})
\end{aligned} \tag{2.1}
$$

The idea is to discretize a backward process in the same fashion as the Euler-Maruyama scheme and take conditional expectation of all non-measurable terms so that all of them become adapted to the filtration at the one step back in time. With this method, one can get the discretization of Y_t as in equation 2.1. In order to discretize Z_t, we have to multiply the whole BSDE by $\mathrm{d}W$ and think of $\mathrm{d}W$ as approximately in the same order of magnitude as $\mathrm{d}t$, then ignore $\mathrm{d}t\mathrm{d}W_t$ term and approximate $\mathrm{d}W\mathrm{d}W$ by $\mathrm{d}t$.

To study the accuracy and convergence of this scheme, we define the error of a numerical method for BSDE as

$$
\sup_{0\le t\le T} \mathbb{E}|\hat{Y}_t - Y_t|^2 + \mathbb{E}\left[\int_0^T |\hat{Z}_t - Z_t|^2 \mathrm{d}t\right]
$$

Let $\pi = \sup_n \Delta_{t_n}$ be the the maximum time stepsize. Bouchard and Touzi [3] shows that

$$
\limsup_{|\pi|\to 0} |\pi|^{-1} \left(\sup_{0\le t\le T} \mathbb{E}|\hat{Y}_t - Y_t|^2 + \mathbb{E}\left[\int_0^T |\hat{Z}_t - Z_t|^2 \mathrm{d}t\right]\right) < \infty
$$

Note that in this convergence result, the estimation of conditional expectation at each time step is assumed to be exact. This is certainly not the case in practice. Even though this result suggests that the error will decrease as we reduce the step size, in practice, since the error from the estimation of conditional expectation can propagate through time steps, increasing the number of time steps do not necessarily improve the performance of algorithm. If we want to increase the number of time steps to reduce global error, we also have to increase the accuracy of the estimation of conditional expectation at each time step.

Next, we consider the error bound including the error from estimation of conditional expectation. It is easy to show that (see [12])

$$
|\hat{Z_{t_n}}| \le \frac{1}{\pi}\sqrt{(\mathbb{E}\left[\hat{Y}^2_{t_{n+1}} | \mathcal{F}_{t_n}\right])}
$$

We assume that the terminal value $\hat{Y}_{t_N} = \Phi(X_{t_N})$ is bounded, so, by backward induction, we can see that $\hat{Y}_{t_n}$ and $\hat{Z}_{t_n}$ are all L^2 functions. By property of conditional expectation, the discretization [12] is equivalent to maximization over L^2 function (Y_{t_n}, Z_{t_n}) of

$$\mathbb{E}\left[(\hat{Y}_{t_{n+1}} - Y_{t_n} + f(t_n, \hat{X}_{t_n}, Y_{t_n}, Z_{t_n})\Delta t_n - Z_{t_n}\frac{\Delta W_{t_n}}{\Delta t_n})^2\right]$$

This formulation is very useful for regression since it is equivalent to the minimization of L^2 error. In order to get the bound for global error that includes this regression error, let R be discretization bound for X_t and R_0 be discretization bound for $\frac{\Delta W_t}{\pi}$. Let P_0 be a vector value function of X_t that consists of basis of the regression that estimates $\mathbb{E}\left[\hat{Y}_{t_{n+1}}|\mathcal{F}_{t_n}\right]$, let α_0 be its corresponding coefficient, and let K_0 be the number of basis, i.e, the dimension P_0. For example, if we use fist order polynomial and constant term as basis,

$$P_0(X_{t_n}) = (1, X_{t_n}^{(1)}, X_{t_n}^{(2)}, X_{t_n}^{(3)}, ..., X_{t_n}^{(d)})$$

when $X_{t_n}^{(i)}$ is the $i-th$ component of X_{t_n}

Then, let $\hat{\mathbb{E}}$ be the approximation of $\mathbb{E}$, we get

$$\hat{\mathbb{E}}\left[\hat{Y}_{t_{n+1}}|\mathcal{F}_{t_n}\right] = \alpha_0 \bullet P_0(X_{t_n})$$

Similarly, for $i = 1, 2, 3, .., d$, let P_i be a vector value function of X_t that consists of basis for the regression that estimate $\mathbb{E}\left[(\hat{Z}_{t_{n+1}}\frac{\Delta W_{t_n}}{\Delta t_n})^{(i)}|\mathcal{F}_{t_n}\right]$, let α_i be its corresponding coefficient, and let K_i be the number of basis, i.e, the dimension P_i. we get

$$\hat{\mathbb{E}}\left[(\hat{Z}_{t_{n+1}}\frac{\Delta W_{t_n}}{\Delta t_n})^{(i)}|\mathcal{F}_{t_n}\right] = \alpha_i \bullet P_i(X_{t_n})$$

With this setting, Lemor et al [12] showed that there exist constant C such that for any $\beta \in (0, 1]$,

$$
\begin{aligned}
&\max_{0\le k\le N}\ \mathbb{E}\left[(Y_{t_k}-\hat{Y}_{t_k})^2\right]+\pi\,\mathbb{E}\left[\sum_{k=0}^{N-1}|Z_{t_k}-\hat{Z}_{t_k}|^2\right]\\
&\le C\frac{R^2}{M}N\sum_{i=0}^{d}K_i+C\pi^{\beta}\\
&+C\sum_{k=0}^{N-1}\left(\inf_{\alpha}\mathbb{E}\left[(Y_{t_k}-\alpha\bullet P_0(X_{t_k}))^2\right]+\sum_{i=1}^{d}\inf_{\alpha}\mathbb{E}\left[(\sqrt{\pi}Z_{t_k}-\alpha\bullet P_i(X_{t_k}))^2\right]\right)\\
&+C\frac{R^2}{\pi}\sum_{k=0}^{N-1}\{\mathbb{E}[K_0\exp(-\frac{M\pi^{\beta+2}}{72R^2K_0}+CK_0\log\frac{CRK_0^{\frac{1}{2}}}{\pi^{\frac{\beta+2}{2}}})]\\
&+\pi\mathbb{E}[\sum_{i=1}^{d}K_i\exp(-\frac{M\pi^{\beta+2}}{72R^2R_0^2K_0}+CK_0\log\frac{CRR_0K_i^{\frac{1}{2}}}{\pi^{\frac{\beta+1}{2}}})]\\
&+exp(CK_0\log(\frac{CR}{\pi^{\frac{\beta+1}{2}}})-\frac{M\pi^{\beta+2}}{72R^2})\}
\end{aligned}
\tag{2.2}
$$

In a simple case when equal size hypercubes are used as a basis, Lemor et al [14] can analytically estimate the error from regression and hence significantly simplify the error bound. This type of basis is clearly ineffective when the number of dimension is high since the number of hypercube required will grow exponentially as the number of dimension increases. However, the analysis of this simple case is very useful since it can be a good benchmark that the more complicated basis can compare to. In particular, it gives an explicit relationship between the number of simulations required for a particular time step size to ensure that the error will not explode. This relationship is very hard to determine analytically for other type of basis.

Let M be the number of simulation at each time step. Lemor et al [12] showed that the error from regression by using hypercube as basis is in the order π^2. To minimize the error bound, β must equal one. If we ignore the effect of discretization boundary for Y_t, to get square error of order π (which is the same order of magnitude as the error from discretization described before), we need to use hypercubes of size $\delta=\frac{\pi}{C}$, which is equivalent to $O(\frac{1}{\pi^d})$ number of basis , and we need to use a number of simulations $M=C\pi^{-3-2d}|log(\pi)|$ for some constant C large enough. If we use uniform time step of size $\pi=\frac{1}{N}$, the number of simulation required becomes $M=CN^{3+2d}|log(N)|$.

The numerical result for the case of one dimension problem in [12] showed that the number of simulation required is much less than in theoretical result. Specifically, the number of simulations M is required to be in order of N^3, instead of N^5 in theoretical calculation. However, if the number of simulations required grows exponentially as

dimension increases, the scheme is certainly not suitable for high dimension problem. For example, in 10 dimensions, the number of simulations for 10 steps may need to be as large as 10^{10}, which is clearly impossible in practice.

Consider the error term in (2.2), the exponential of dimension term in M comes from the fact that the error from using hypercube as basis for regression grows exponentially as dimension increases. Since polynomial regression also share this same property, we may expect it to face the same curse of dimensionality when applying to solve BSDE. Consequently, polynomial regression may not be an efficient method for this problem. We need other regression method that can deal with high dimension. The best method would be the one that error increases in order of small degree polynomial of dimension. The method that error increases in order of exponential of dimension but with small coefficient in exponent may also work for fairly high dimension problem.

2.2 Discretization of RBSDE

The simplest way to discretize RBSDE is to compare use the usual discretization for BSDE but also compare the estimated value of Y_{t_n} from conditional expectation ,called $\tilde{Y_{t_n}}$,with the intrinsic value at each step and choose the larger one as $\hat{Y}_{t_n}$ (the approximation of Y_{t_n}). This method of comparing and choosing the larger value is the same as the usual dynamic programming method to price American option by finite difference scheme. The discretization scheme can be expressed as

$$\begin{aligned}
\hat{Z}_{t_n} &= \mathbb{E}\left[\hat{Y}_{t_{n+1}}\frac{\Delta W_{t_n}}{\Delta t_n}|\mathcal{F}_{t_n}\right] \\
\tilde{Y}_{t_n} &= \mathbb{E}\left[\hat{Y}_{t_{n+1}} + f(t_n, \hat{X}_{t_n}, \hat{Y}_{t_n}, \hat{Z}_{t_n})\Delta t_n|\mathcal{F}_{t_n}\right] \\
\hat{Y}_{t_n} &= \max(\tilde{Y}_{t_n}, \Phi(t_n, X_{t_n})) \\
\hat{Y}_{t_N} &= \Phi(t_N, X_{t_N})
\end{aligned} \tag{2.3}$$

Gobet et al [12] showed that when Φ is C^1 in time, continuously Lipschitz in both variable, and $\Phi(t, X_t), \Phi(t, \hat{X}_t)$ satisfy a specific Ito's expansion (see [12] for detail), we get error bound

$$\max_{0\le k\le N} \mathbb{E}\left[(Y_{t_k} - \hat{Y}_{t_k})^2\right] + \pi\left[\sum_{k=0}^{N-1}\int_{t_k}^{t_{k+1}} \mathbb{E}|Z_t - \hat{Z}_{t_k}|^2 \mathrm{d}t\right] \le C(1 + |X_0^4|)N^{-\frac{1}{2}} \tag{2.4}$$

This error bound suggests that the mean square error for this scheme has the rate of convergence $N^{-\frac{1}{4}}$, which is slower than $N^{-\frac{1}{2}}$ in the case of simple BSDE. Therefore, when implementing this scheme we may need to use more time steps to get accurate results. Since we discretize RBSDE in a similar fashion as BSDE, this scheme also inherits the error back propagation problem. Since the convergence rate for RBSDE is slower and we need more time steps, the error back propagation becomes more severe and we will need more simulations to prevent this problems. We expect the scheme for RBSDE to be less accurate than simple BSDE and require more computation time.

There are also several other ways to discretize RBSDE. These methods are mostly inspired by the variations of finite difference scheme for solving American option. For example, we can use penalty method [11] to discretize RBSDE by introducing penalty term to the usual discretization Y_t. This term equals to the negative part of the difference between $\hat{Y}_t$ and its corresponding intrinsic value and captures the violation of immediate exercising bound. When $\hat{Y}_t$ is lower than the immediate exercising value, this penalty term increases value of $\hat{Y}_t$ in the next time step. [7] showed that this scheme converges to the corresponding solution of RBSDE as the coefficient of penalty term go to infinity. Another way is to use regularization method [11] that is based on the semi-martingale representation of Φ. Due to time constraint, we leave these methods for future research.

2.3 Tree based regression

Tree regression partitions the whole input space into a family of hyperrectangle according to the structure of data. This method is different from simple equal size hypercube regression where the construction of these basis is independent of data. Tree regression try to partition space so that the input lying in the same hyperrectangle correspond to similar output. When building regression tree, we need to find the best splitting point for hyperrectangle and the best common predictor for point in each sub-hypercube in order to minimize a particular loss function. The following presentation of tree method largely inspired by [13]. Let $R_1, R_2, R_3, ..., R_H$ be the hyperrectangles that form the partition of whole space for a tree regression and let $c_1, c_2, c_3, ..., c_H$ be the corresponding coefficient for each hyperrectangle. This regression tree predicts the output $\hat{Y}_{t_n}$ of the unseen input X_{t_n} by

$$\hat{Y_{t_n}} = \sum_{i=1}^{H} c_i I\{X_{t_n} \in R_i\}$$

when I is indicator function. In short, this method finds the hyperrectangle where X_{t_n} belongs to and use the coefficient of that hypperrectangle as the predictor for the value of Y_{t_n}.

The most common way to build this tree is to use binary tree, which divides hyperrectangle into half at each time step by using single split point at each time. Let L be the loss function that penalize the difference between $\hat{y}$ and y. The simplest way to build this binary tree is to use greedy algorithm at each time step by calculating loss of each hyperrectangle and splitting the one with highest loss. Let R_J be the the hyperrectangle with highest loss, $R_J^{(1,j)}$ and $R_J^{(2,j)}$ be the two sub-rectangle of R_J when using j as the split point, and (x_i, y_i) be pairs of input and output training data. At each step, we need to solve

$$\min_j \left[\min_{c_1}\{ \sum_{x_i \in R_J^{(1,j)}} L(y_i, c_1)\} + \min_{c_2}\{ \sum_{x_i \in R_J^{(2,j)}} L(y_i, c_2)\} \right] \tag{2.5}$$

This greedy algorithm does not always lead to the partition that minimizes the loss of all training data. However, using an exhaust search over all possible partitions of space would be computationally impossible and this greedy algorithm usually lead to an acceptable approximation in practice [13]. Note that even though we use only one split point at each step, the final partition can be very complicated and we hope that this hyperrectangle basis will capture the feature space well.

Trees that minimize sum square error are relatively easy to build. This is because for each split point, the optimal values that minimize sum square error in each sub-regions are just mean value of all points in that that sub-regions, i.e.,

$$c_1 = \text{average}_{x_i \in R_J^{(1,j)}}(y_i)$$
$$c_2 = \text{average}_{x_i \in R_J^{(2,j)}}(y_i)$$

These formula largely reduce the number of computation required to solve (2.5). For each split point j, we can easily determine the coefficient of the two corresponding sub-hyperrectangles. Therefore, the only task left is to find this optimal splitting point j. The obvious method is to use exhaustive search for all possible splitting points. This method guarantee to give the most optimal split point. However, this method

requires lots of computation. The number of possible split points is the number of points in that region time dimension. To reduce the computation time, other heuristic search can be employed. One of the fastest way is to compare only the median points on each dimension. However, this method certainly sacrifices some accuracy since it does not give the minimal sum square error. If the tree level is large enough, we may expect this approximation to be closed to the ideal split point. We can explore the trade-off between these two extremes by increasing the number of points we consider in each dimension. For example, we may consider quartiles of each dimension instead of simply using just median.

Tree regression can be used as a building block for other complicated methods. For example, boosting trees, like Adaboost, use the sum of several or many trees as a predictor. The idea of constructing these trees is the same as in a single case where we have to find partition and corresponding coefficient to minimize some loss function. However, in this case, the computation complexity may make it possible to train these trees at all since we need to find the best partition not only across all hyperrectangles, but also across all tree, simultaneously. One way to deal with this problem is to use the sub-optimal partition by finding the best tree that can be added to the model at each step without changing the learned trees. For square error lost function, the optimal tree to be added is relatively much more easy to find since this tree is simply the one that best estimates the residual from previous step. Therefore, the complexity in finding this additive tree is the same as in training a single tree. With this simplification, boosting tree with square error loss belongs to the family of the forward stagewise additive model, which iteratively adds new basis function to estimate the residual from previous step, without changing the learned parameters.

Other examples of tree based method include random forest and Bayesian additive regression tree (BART) [6]. Random forest uses collection of many independent identically distributed random trees to learn data and let them "vote" for the output. Breiman [4] used mode as the vote for these trees. He found that the performance of random tree is comparable to Adaboost but it is more robust against noise. Bayesian additive regression tree (BART) [6] also uses the sum of trees to make prediction. However, trees are grown using a common prior to prevent over-fitting. The main difference from gradient boosting tree is that BART tries to make the effect of each tree equal to each other. This is in contrast with gradient boosting tree where later tree estimates smaller residual. Due to the time limit of the project, we can only explore basic tree approach in high detail. These tree based methods are certainly interesting for future research.

Tree-based methods are wildly used across different areas because of its power and flexibility, especially when dealing with high dimension data. For example, Hastie et al [13] suggested that trees are good at handling large quantities of data (number of simulations in our case). Caruana et al [5] used empirical experiment to show that tree based methods outperform other statistical learning technique in high dimension. These results suggest that tree method may work with high dimension problem and hence may be applicable to the numerical method to solve BSDE.

As tree-based methods asymptotically give accurate estimate when doing regression, using the same method as Gobet el al [12], we can see that using tree regression to approximate conditional expectation in 2.3 will theoretically converge to the value of the BSDE. In the next chapter, we will therefore focus on weather these methods perform well in practice.

Chapter 3

Empirical result

In this chapter, we present our empirical result using tree regression for various problems. Our aim is to get the regression method that can deal with high dimension problem using normal laptop or PC. Therefore, we limit ourself to the capacity of our laptop and perform experiment under this constraint. It turned out that the limiting factor in our experiment was the number of simulations. The maximum array size that our Matlab program can deal with is around 150 million. When using 15 time steps and in 20 dimensions, the maximum number of simulations that we can do is around half a million.

3.1 Numerical method for BSDE

3.1.1 one dimensional BSDE

We first begin by considering one dimension case. We use the same problem as in Gobet et al [12], which gives a good benchmark for our experiment. We consider the problem of different interest rates defined in Chapter 2. Let the forward process be geometric Brownian motion with drift $\mu = 0.05$, volatility $\sigma = 0.2$, and initial value $X_0 = 100$. We use $[40, 180]$ as discretization bound for these Brownian motions. We consider a call spread option as the terminal payoff $(X_T - 95)^+ - 2(X_T - 105)^+$. Let short interest rate $r = 0.01$, long interest rate $R = 0.06$ and maturity $T = 0.25$. The driver of this BSDE derived in Chapter 2 is

$$f(t, x, y, z) = -\theta z - ry + (y - \frac{z}{\sigma})^-(R - r)$$

The empirical result by Gobet et al[12] showed that the solution of this problem is 2.95. We will use this value as reference to calculate bias and variance of our method.

To compare tree building methods, we tested (i) using full exhaustive search to find optimal splitting points and (ii) using only median as the candidate for splitting point. From now on, we will refer to them as full and median tree regression, respectively.

We first fixed the number of time steps and vary the number of simulations. Gobet et al [12] showed that N less than 20 is enough to low discretization bias, given that the number of simulations is large enough. Here, we fixed $N = 15$ and use the same number of simulations as in [12], which is up to around 25000. The regression tree that we used for this problem consists of 20 levels. We ran whole algorithm 10 times to estimate bias and variance.

Fig 3.1 and Fig 3.2 showed the result of full tree regression and median tree regression, respectively. In each figure, the left graph is the plot between the number of simulations and predicted solution and the right graph is the corresponding log-log plot. The star symbols represent plus and minus two standard deviations. The error in both cases are comparable to the error from the uniform hypercubes method used in Gobet et al [12]. However, the time required is much higher. This drawback of tree-based method is much more severe for the full exhausted search. The computational difference becomes more apparent as the number of simulations increases. For example, when using 65536 simulations, the full exhaustiv search used 1278 seconds to finish the whole algorithm while median tree regression used only 34 seconds. Despite this time difference, the error from both methods are comparable. This result suggests that it may be impossible and not useful to perform exhausted search when we solve the problem with higher number of dimension. We also found that boosting neither improve the accuracy nor reduce the number of simulation required. Therefore, we drop both full tree regression and boosting in further experiments.

The error in this case can be further reduced by increasing the number of simulations. Fig 3.3 shows the same problem using median tree regression but with more simulation paths. This figure shows that the scheme really converge as the number of simulations increase. The error when using the largest number of simulations is less than 0.01. This result also confirms that the number of time steps $N = 15$ is enough to get small bias in this problem.

3.1.2 multi-dimensional noisy BSDE

In order to test the tree-based method in higher dimensions cases, we next consider the same problem but with high dimension noise. In this experiment, only the first dimension of the forward process will be used to evaluate the terminal payoff. However, we also include other geometric Brownian motions as additional dimensions of

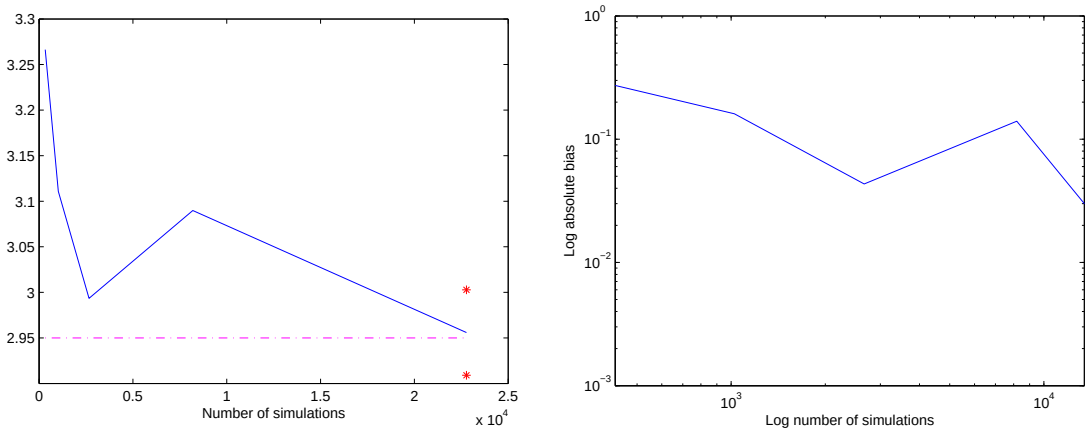


Figure 3.1: one dimension problem using full regression tree

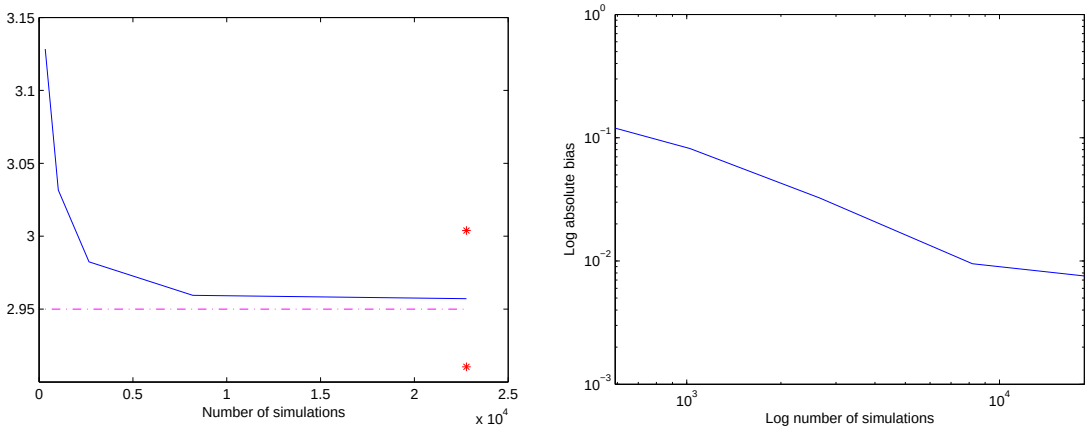


Figure 3.2: one dimension problem using median regression tree

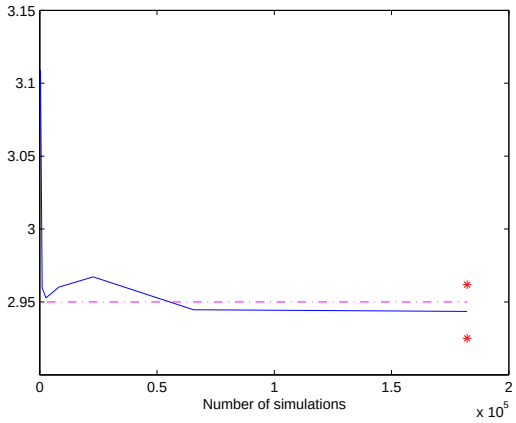


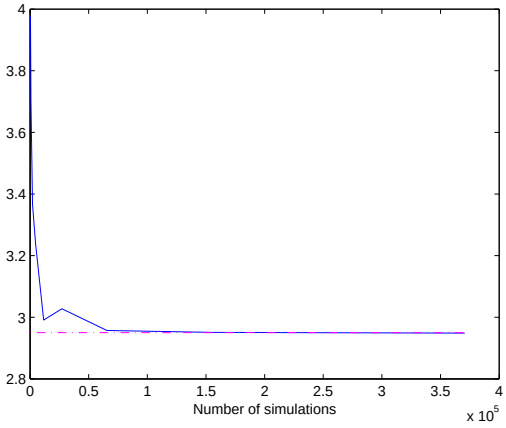
Figure 3.3: one dimension problem using median regression tree with more simulations

the forward process to create noisy inputs. We use same drift, volatility, and initial value for all geometric Brownian motions used in this problem. All Brownian motions are independent of the Brownian motion in other dimensions. Since the problem is basically the same as the one dimension case, we expected the number of time steps required by this scheme to be the same as in previous case but we may need more simulations to capture this noise.

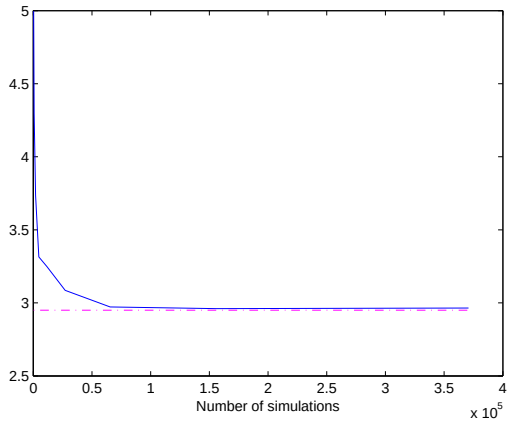
Fig 3.4 shows the results of median tree regression for 10, 20 and 25 dimensional problems. In all cases, we used tree with 20 levels, which is a fairly low number but is enough for our algorithm to converge to the true value in these case. The reason why we do not need high level tree may be because the real underlying forward process is only the first dimension of the input. Therefore, low level of tree is enough since ideally all the split points must be in the first dimension. This result also shows that median regression tree is robust to noise and can capture the dominant dimension well. Due to the computer memory constrain, when the number of dimensions of the problem is more than 25, we cannot use the same number of simulations as we use in previous cases. We leave this issue for future research.

3.1.3 Weighted sum of Brownian motion BSDE

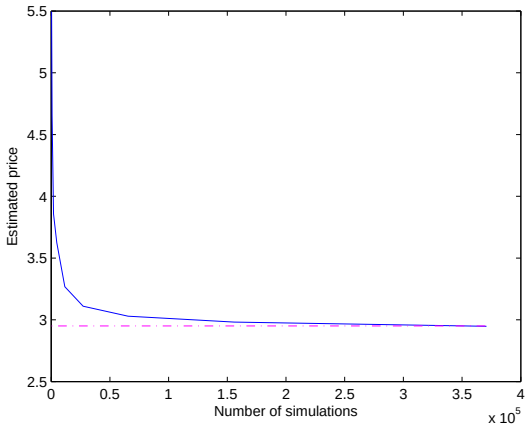
To test the performance of our algorithm with the problem where payoff depends on more than one dimension of the forward process, we work backward by firstly generating one geometric Brownian motion to be used in the payoff function. This is the real underlying forward process of this problem. Instead of feeding this geometric Brownian motion directly to the scheme, we generate another $d-1$ independent geometric Brownian motion and use d random weighted sum of these geometric Brownian motion



(a) 10 dimensional noisy input



(b) 20 dimensional noisy input



(c) 25 dimensional noisy input

Figure 3.4: regression tree for noisy input

as the forward process. For example, when $d = 2$, we first generate geometric Brownian motion X^1 and use it to as the real underlying forward process of this problem. We then generate another independent geometric Brownian motion X_2 and 4 random numbers a_1, a_2, a_3, a_4. Finally, we use $\tilde{X}^1 = a_1X^1 + a_2X^2$ and $\tilde{X}^2 = a_3X^1 + a_4X^2$ as forward input processes for our scheme. With this method, the underlying geometric Brownian motion will depends on all dimensions of the inputs. Note that the derived forward process in this case is no longer a geometric Brownian motion. Since the real forward process of this problem is still the same as in one dimension case, the solution of this problem should be the same as in the previous cases.

We first begin by using the same number of time steps and maximum tree levels as in previous cases. Fig 3.5 shows the result for 10 dimensional problem. As we can see from this figure, the scheme does not converge to the true solution. We can also see that the variance is very low in this case. This is a common bias-variance trade off problem where bias can be very high when variance is too low. Since the real underlying process in this problem is the same as in previous case, we expect that the number of time step $N = 15$ should still be enough and does not largely contribute to this bias. Instead, we believe that this bias comes from using too few basis. Therefore, we change the maximum tree level to 100 in the next experiment.

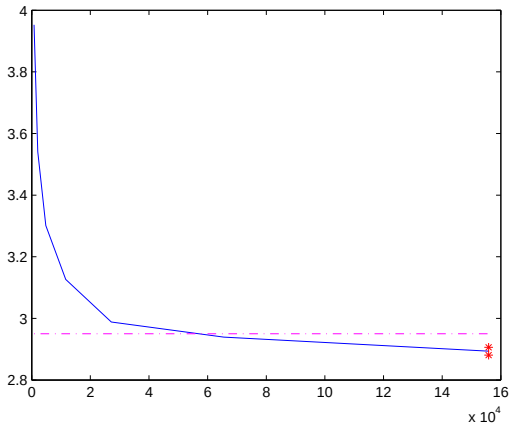
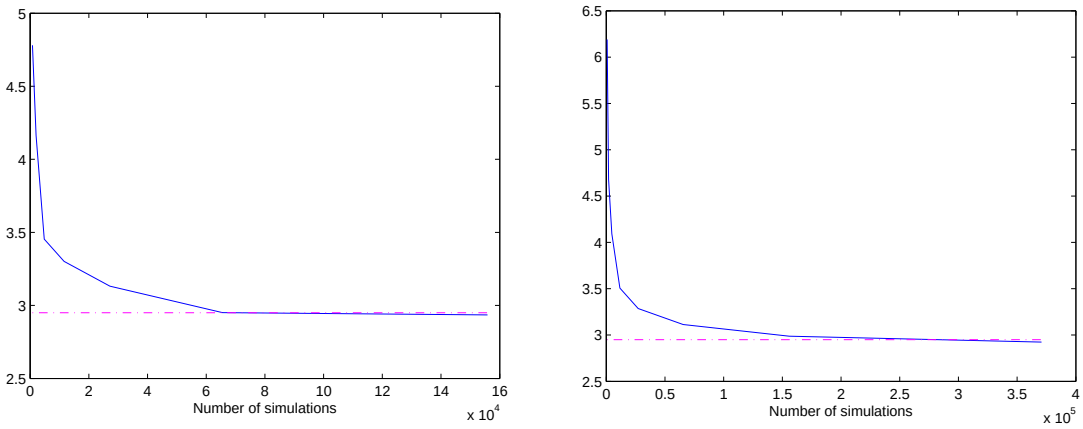


Figure 3.5: 10 dimensions weighted sum of BW problem with 20 tree levels

Fig 3.6 shows the results for the cases of 10 and 20 dimensional problem when using 100 tree levels. The scheme for both cases is now converge to the true solution with error at the largest number of simulations less than 0.01. These results suggest that our scheme still works even in the case where the payoff depends on many dimensions.

For higher dimensions, the same scheme seems to converge but the convergence rate is too low that we cannot run enough simulations until it close to the true



(a) 10 dimensions (b) 20 dimensions

Figure 3.6: weighted sum of BW problem with 100 tree levels

solution. Fig 3.7 shows the the result for 25 dimensions problem. This figure suggests that we may need twice the number of simulations more than in the previous lower dimension cases.

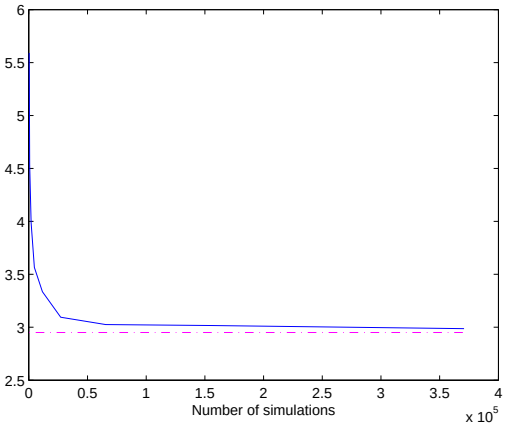


Figure 3.7: 25 dimensional weighted sum of BW problem with 20 tree levels

3.2 Numerical method for RBSDE

In this section, we present the results of our numerical experiment with RBSDE. The forward process X_t is still Brownian motion. Let $X_t^{(i)}$ be the ith dimension of X_t. Each $X_t^{(i)}$ has the same drift μ that equals to interest rate $r = 0.05$ under risk neutral measure, and has the same initial value $X_0^{(i)} = 100$. All dimensions of Brownian motions in this problem have same volatility but each case has its own

value of volatility. We consider the American basket put option with payoff $\Phi(t, X_t) = (K - (\prod_{i=1}^{d} X_t^{(i)})^{\frac{1}{d}})^+$ when $(\,)^+$ represents positive part. The maturity $T = 1$.This is the same problem as in Bouchard and Touzi [3].

Since the product of geometric Brownian motion is still a log-normal process, this problem can be reduced to pricing one dimension American put option. The geometric Brownian motion $X_t^{(i)}$ can be written as

$$X_t^{(i)} = X_0 \exp((r - \frac{1}{2}\sigma^2)t + \sigma W_t^{(i)})$$

when $W_t^{(i)}$ are independent Brownian motions at time t. Then, their geometric average becomes

$$(\prod_{i=1}^{d} X_t^{(i)})^{\frac{1}{d}} = X_0 \exp((r - \frac{1}{2}\sigma^2)t + \frac{\sigma}{d}\sum_{i=1}^{d} W_t^{(i)})$$

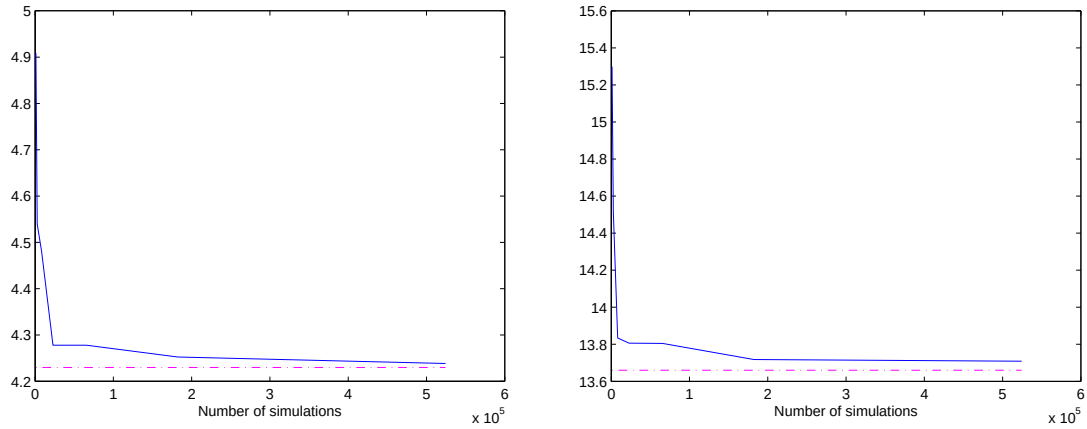
It is easy to show by Levy's characterization that $\frac{1}{\sqrt{d}}\sum_{i=1}^{d} W_t^{(i)}$ is also a Brownian motion. Let $B = \frac{1}{\sqrt{d}}\sum_{i=1}^{d} W_t^{(i)}$, then,

$$(\prod_{i=1}^{d} X_t^{(i)})^{\frac{1}{d}} = X_0 \exp((r - \frac{1}{2}\sigma^2)t + \frac{\sigma}{\sqrt{d}}B)$$

This is not a geometric Brownian motion under the risk neutral measure. However, if $f(t) = \exp(-\frac{1}{2}\sigma^2\frac{1-d}{d})$, we can express the geometric average as

$$\begin{aligned}(\prod_{i=1}^{d} X_t^{(i)})^{\frac{1}{d}} &= X_0 \exp((r - \frac{1}{2}\sigma^2)t + \frac{\sigma}{\sqrt{d}}B) \\ &= X_0 \exp(-\frac{1}{2}\sigma^2\frac{1-d}{d}) \exp((r - \frac{1}{2}(\frac{\sigma}{\sqrt{d}})^2)t + \frac{\sigma}{\sqrt{d}}B) \\ &= f(t)X_0 \exp((r - \frac{1}{2}(\frac{\sigma}{\sqrt{d}})^2)t + \frac{\sigma}{\sqrt{d}}B)\end{aligned}$$

Let $S_t = X_0 \exp((r - \frac{1}{2}(\frac{\sigma}{\sqrt{d}})^2)t + \frac{\sigma}{\sqrt{d}}B)$, then, S_t is a geometric Brownian motion under the risk neutral measure. Moreover, since B is a linear combination of $W_t^{(i)}$, S_t can be hedged using $X_t^{(i)}$. Therefore, the price of this American basket option can be reduced to American put option with payoff $\Phi'(t, S_t) = (K - f(t)S_t)^+$. This one dimension problem can be solved by usual finite difference method such as Crank-Nicolson scheme. We compare our experimental result with the corresponding the value estimated by using Crank-Nicolson scheme. This one dimension scheme is quite accurate and provides good references for our experiment.



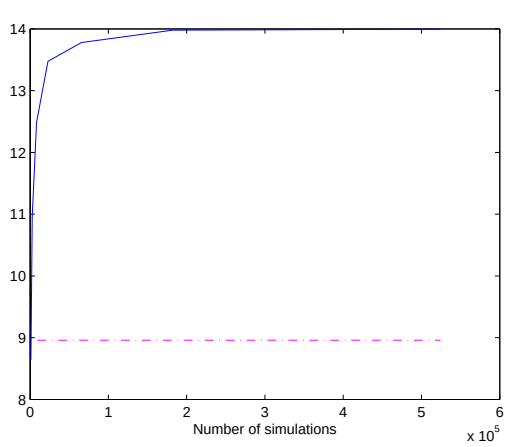
(a) Volatility 0.15 (b) Volatility 0.4

Figure 3.8: One dimension American basket option

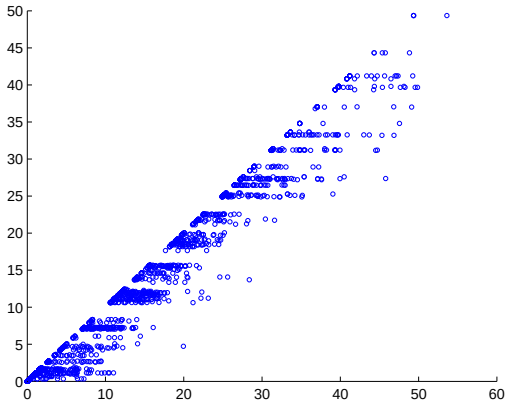
We first begin with one dimension case. We use number of time steps $N = 50$,which is the same as in [3] and the maximum number of tree level 200. We still use median method to build tree. Fig 3.8 shows the result for problems when *sigma* equal 0.15 and 0.4. Our scheme converges closed to the value calculated by finite difference method, which are 4.23 and 13.66,respectively. The error when using the maximal number of simulations is less than 0.1 in both case.

Even though this American basket payoff with geometric average provides good reference value that we can compare with, the multiplication structure of the payoff can be problematic for numerical scheme. If the geometric Brownian motion for some dimension is accidentally sampled from the tail of distribution and becomes abnormally large, the overall product of all dimensions can be also quite large due to the multiplication. This result can lead to the instability of numerical scheme. This is not a big problem for simple BSDE since we need to evaluate this multiplication only once for the terminal payoff. The irregularity may be moderated throughout the time backward marching. However, in the case of RBSDE, this problem becomes much more severe since we need to evaluate this multiplication at each time step to compare the value from regression with the immediate exercising value. The error from instability can occur at any time steps and can propagate through time step and hence becomes much more severe than simple BSDE.

This instability becomes worse for our tree regression method since tree is well known for its variance and instability [13]. One main reason is that the error from splitting at the earlier levels can propagate throughout later levels of splitting. The instability severely effect the performance of our tree regression method.



(a) estimated price when volatility is 0.4



(b) plot between the estimate before and after the maximization step

Figure 3.9: 3 dimensional American basket option

Fig 3.9a shows the result for the three dimension case when σ equal 0.15 and 0.4. Finite difference scheme suggests that the solution should be 2.12 and 8.96,respectively. As we can see from figure 3.9a, our numerical methods do not work in both case. Our estimate largely overestimate the option prices. To confirm that this is the result of instability, we plot the estimate $\hat{Y}_{t_{N(i)-1}}$, which is the estimation of Y_{t_n} after comparing with intrinsic value, against $\tilde{Y}_{t_{N(i)-1}}$, which is the estimate of conditional expectation using tree regression. The time $t_{N(i)-1}$ is the first step in our backward time marching. Fig 3.9b shows this plot. If the scheme is stable, we would expect that $\hat{Y}_{t_{N(i)-1}}$ and $\tilde{Y}_{t_{N(i)-1}}$ will be closed to each other. The graph in fig 3.9b should be a slope one linear line and some points deviate not much from the main linear line. However, as we can see from fig 3.9b, this is not the case. There are lots of large deviation from the main linear line. The first step of numerical scheme alone shifts the price up a lot. This figure confirms the instability of our scheme.

In order to fix this problem, we limit the maximum increment that can be added to $\tilde{Y}_{t_n}$. Instead of

$$\hat{Y}_{t_n} = \max(\tilde{Y}_{t_n}, \Phi(t_n, X_{t_n}))$$

, we introduce the following steps to our algorithm:

$$I_{t_n} = \max(\tilde{Y}_{t_n}, \Phi(t_n, X_{t_n})) - \tilde{Y}_{t_n}$$
$$\hat{Y}_{t_n} = \tilde{Y}_{t_n} + \frac{I_{t_n}}{1 + pI_{t_n}}$$

when p is non-negative value that control the size of increment. $p = 0$ corresponds to no limit on increment. This corresponds to the taming schemes suggested by Szpruch [18].

By limiting the maximum increment at each step, our scheme becomes more stable and converges the value estimated by using finite difference method. Fig 3.11 shows the results for three dimensions problems when σ equal 0.15 and 0.4. We use $p = 20$ and $p = 75$

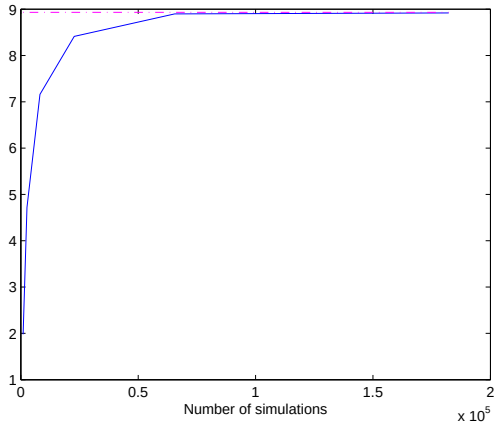
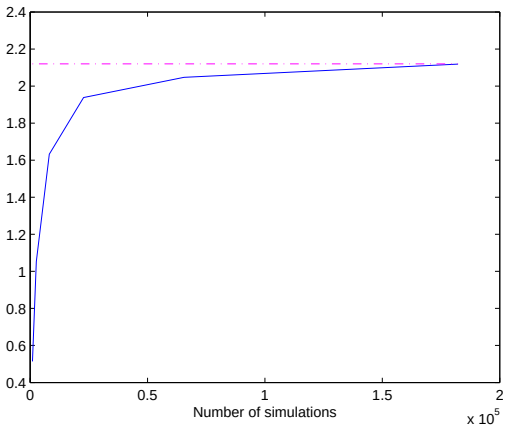


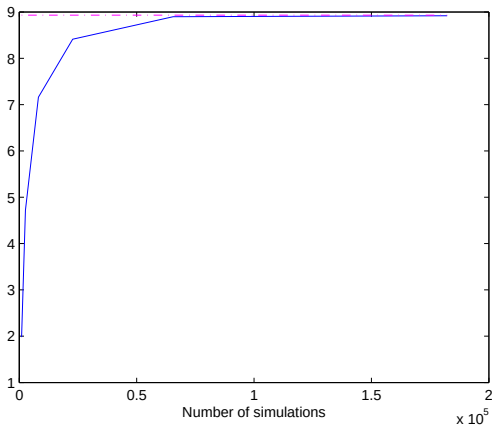
Figure 3.10: 3 dimensional American put problem when volatility is 0.4

The main drawback of our algorithm now is that specifying appropriate p for each problem is still ad hoc. This is a big issue that we cannot fully explore in this dissertation and have to leave it for future research. However, our numerical experiment suggests that the optimal p depends on the volatility σ and the dimension of problem. When $\sigma = 0.4$, we found that $p = 25*$dimension works well for dimension less than 20. Fig 3.12 shows the result of our numerical experiments in this case when dimension is $3, 5, 10$, and 20. For $\sigma = 0.8$, we found that $p = 2 * 25 *$ dimension also work well for dimension less than 20. Fig3.13 shows these results when dimension is $3, 5, 10$, and 20. These result suggests that p may have some linear dependence on dimension and volatility.

However, when we tried to apply similar relationship with the case when $\sigma = 0.15$, we do not get the same simple linear relationship to determine p. When dimension are 3 and 5, $p = 7 \times 3$ and 5×5, respectively, gives good performance. Fig 3.14 shows the result in this case. The result when $\sigma = 0.15$ is not in consistent with the previous two cases. This result might also be because in this case the solutions to these problems are relatively small, unlike the cases of $\sigma = 0.4$ and $\sigma = 0.8$, and

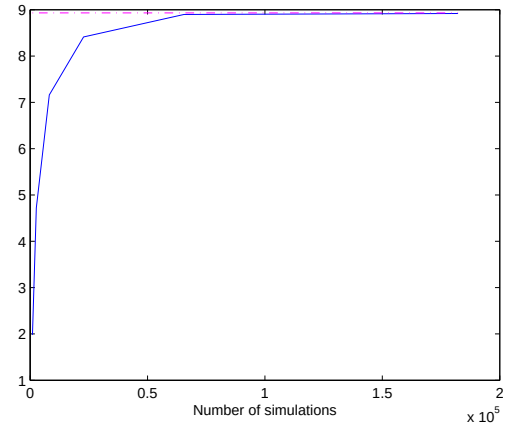


(a) volatility is 0.15 and p is 20

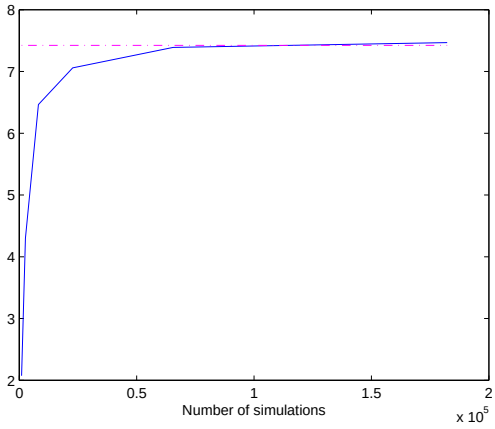


(b) volatility is 0.4 and p is 75

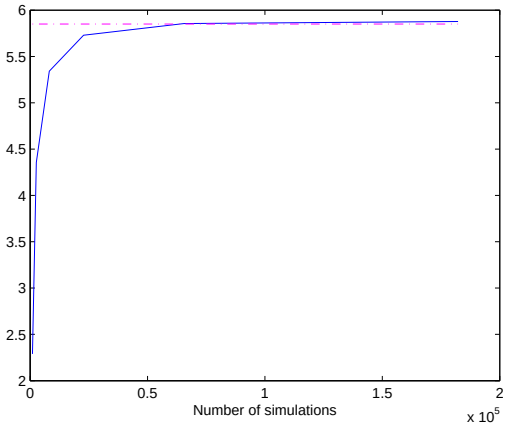
Figure 3.11: 3 dimensional American put problem



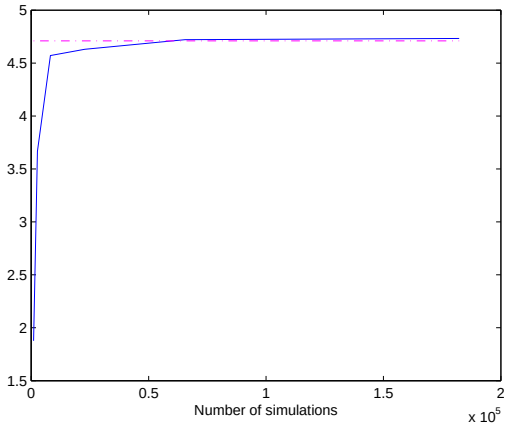
(a) 3 dimensions



(b) 5 dimensions

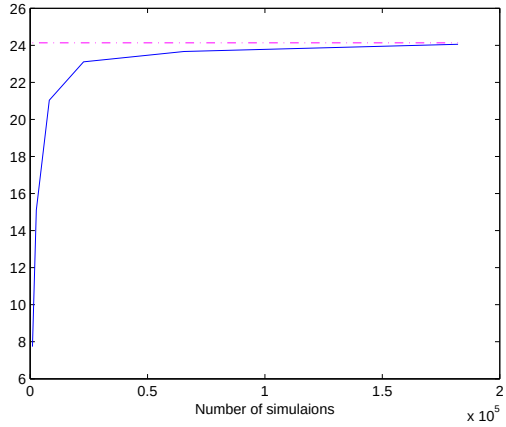


(c) 10 dimensions

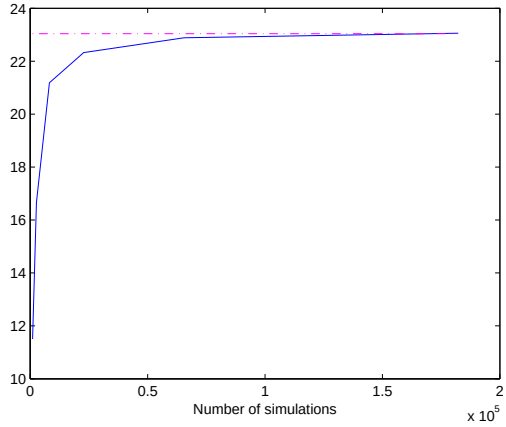


(d) 20 dimensions

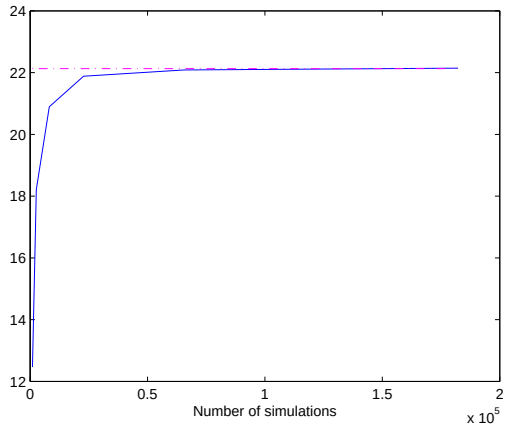
Figure 3.12: American basket option with ,volatlity 0.4, $p = 25 * \dim$



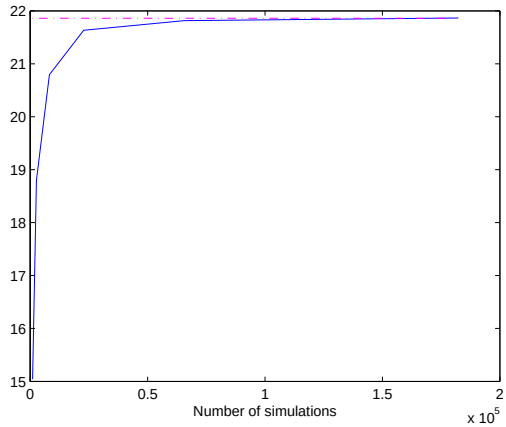
(a) 3 dimensions



(b) 5 dimensions

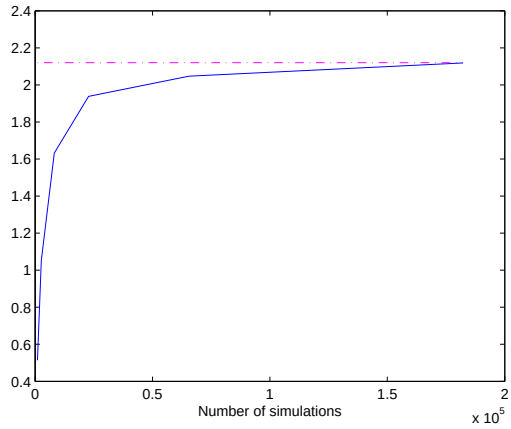


(c) 10 dimensions

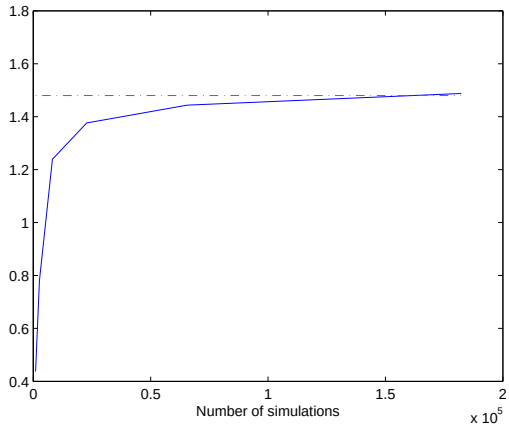


(d) 20 dimensions

Figure 3.13: American basket option with, volatlity 0.8, $p = 25 * \dim$



(a) 3 dimensions when $p = 7 \times 3$



(b) 5 dimensions when $p = 5 \times 5$

Figure 3.14: American basket option with, volatlity 0.15

hence the relative error become larger. This relationship between p, volatility, and dimension is a very interesting topic for future research.

Chapter 4

Conclusions and future works

In this dissertation, we review the numerical method for solving a decoupled FBSDE. In order to deal with a high dimension problem, we propose a new scheme based on tree-based regression. Our numerical results showed that this tree-based scheme can work well for the problem where dimension is less than 30. This may be because of the inherit ability of tree-method to deal with a high dimension problem.

We found that comparing the median points of each dimension is enough to build a fairly accurate tree, given that the number of simulation paths is large enough. Using exhuasted search to build tree required much more time and neither improve the accuracy nor reduce the number of simulation required significantly. One interesting extension to this result is to analyse the case when we use more points, such as using quantile or percentile, as the candidate for tree splitting point. Since the performance of median regression tree that we used is already good, we do not found significant difference when using more points as candidate for the splitting points. We do not found significant improvement when using L^2 boosting trees, either. However, the situation may be changed for different problems and it is interesting to compare the performance of these methods with other drivers and terminal conditions.

The constrain that prevent us from solving a decoupled FBSDE with dimension higher than 30 is the memory required to store the simulation paths of forward process. For general problem, we may not be able to improve this limit much since more simulation paths are naturally required to represent a feature space, which will become bigger as the dimension increases. However, in a special case when the forward process follows geometric Brownian motion (or some smooth function of Brownian motion), we may be able to avoid this problem by using Brownian bridge to simulate the forward process. Brownian bridge is based on the idea that a Brownian motion at anytime in a particular time interval is conditionally independent of all Brownian motions outside this interval, given the Brownian motions at both end of this interval.

Consequently, we can need to sample only the Brownian motion required at each time steps and do not have to store the whole Brownian path the beginning. This method may help us to overcome the memory constrain and solve a decoupled FBSDE with dimension higher than 30. This modification of the scheme is very interesting for future research.

We also tested our scheme with American basket option, which is a special case of RBSDE. We found that the main obstacle in this case is that the our scheme is no longer stable since it is not robust the erratic change of Brownian motion. We can solve this issue by limiting the maximum increment size at each step of numerical scheme. This modified scheme works well for problems with various volatility and dimension. However, the method that we used to determine parameter that limits the increment size is still ad hoc. We also found that this parameter depends on dimension and volatility, which is reasonable since as dimension and volatility increase, there are more chance that at least one dimension of Brownian motion will be abnormally large and hence effect the stability of whole scheme. We also found that in when the option price is not too small, this limiting parameter is linearly proportional to dimension and we can use the appropriate parameter for low dimension case to predict the one for high dimension case with the same volatility. This is a very preliminary result and need to be investigated much more in future research.

The main contribution of this dissertation is not to find the best method to numerically solve BSDE. Instead, our result suggests that tree-based method can be a good alternative for Voronoi partition or polynomial regression in the discretization scheme for BSDE, especially when the number of dimension is high. Using the more complicated tree-based method may further improve the performance of our scheme. We believe that a tree-based regression is a very promising approach to solve high dimension BSDE.

Appendix: Matlab code

The following Matlab codes were used in this dissertation.

Building regression tree

```
function [regt term sq] = findtreeregressionsevptv3( x,yo,nlev,pt)
% % Find tree regression
% x : matirx of input, each row corresponds to each dim
% yo : row vector of estimated output
    % y(k+1,:)*(w(k,:)-w(k,:))/h or
    % y(k+1,:)+f(...)*h
% nlev : number of level of tree

%return regt matrix each row corresponds to node
%col 1 position (row #) in terminal list or 0 if not a terminal node
%col 2 dim of split point (0 for terminal)
%col 3 split point as x value (0 for terminal)
%col 4 row # of parent in regt
%col 5 row # of left daughter (0 for terminal)
%col 6 row # of right daughter (0 for terminal)
%col 7 value

%

m_yo = sum(yo)/length(yo);

sq = sum((yo-m_yo).^2);

dim = length(x(:,1)); %number of dimension

[s_x, i_x]  = sort(x,2); %sort x and index of sort along each dim
[~, o_x] = sort(i_x,2); %order of each column in original matrix

regt = zeros(1,7); % 1.Position in terminal list (0 if not terminal) 2.dim 3.split a
regt(1) = 1;

term = zeros(1,2+2*dim); %list of terimal node:position in regt,error, index of boun
termpt = ones(size(yo)); %the terminal node that each point belong to

term(1) =1; term(2) =sq;
term(3:end) = repmat([1,length(x(1,:))],1,dim);

for l = 1:nlev
    %find splitting point
    te_min = intmax;

    c1temp = 0;
    c2temp = 0;

    m_min = 0; %index for split point
    d_min =1;
```

```
%find region with max sq error
[max_er, inv] = max(term(:,2));

%find index of y in this region
setty = find(termpt==inv);



for d = 1:dim
    l_setty = length(x(d,setty));
    if l_setty<pt
        search = 1:l_setty;
    else
        search = floor(l_setty*linspace(0,1,pt+2));
        search = search(2:end-1);
    end
    filter = zeros(size(i_x(d,:)));
    filter(o_x(d,setty)) = ones(size(o_x(d,setty)));
    colsx = i_x(d,filter==1);
    cysum = cumsum(yo(colsx));
    cysumsq = cumsum(yo(colsx).^2);


    for m = search;
        if m == 0
            continue
        end
        c1 = cysum(m)/m;
        temp1 = -m*(c1^2);
        c2 = (cysum(end)-cysum(m))/(l_setty-m);
        temp2 = -(l_setty-m)*(c2^2);

        temp = temp1+temp2;
        if(te_min>temp)
            m_min = o_x(d,colsx(m));
            te_min = temp;
            tempcolsx = colsx;
            tempm = m;
            c1temp = c1;
            c2temp = c2;
            sq_er1 = cysumsq(m)+temp1;
            sq_er2 = (cysumsq(end)-cysumsq(m))+temp2;
            d_min = d;
        end

    end
end

lset = tempcolsx(1:tempm);
rset = tempcolsx(tempm+1:end);

if m_min ~= 0
    % inv is the position of terminal node that will be splitted
```

```
        %add to term
        %change error of parent node in term to 0
        term(inv,2) = 0;

        %add to term
        term = [term; term(inv,:)];
        term(end,1) = length(regt(:,1))+1;
        term(end,2) = sq_er1;
        term(end,2+2*d_min) = m_min;
        termpt(lset) = size(term,1)*ones(size(lset));

        term = [term; term(inv,:)];
        term(end,1) = length(regt(:,1))+2;
        term(end,2) = sq_er2;
        term(end,2+2*d_min-1) = m_min;
        termpt(rset) = size(term,1)*ones(size(rset));

        %add to regt
        %change parent
        regt(term(inv,1),1) = 0;
        regt(term(inv,1),2) = d_min;
        regt(term(inv,1),3) = s_x(d_min,m_min);
        regt(term(inv,1),5) = length(regt(:,1))+1;
        regt(term(inv,1),6) = length(regt(:,1))+2;

        regt = [regt; length(term(:,1))-1, 0, 0, term(inv,1),0,0,c1temp]; %left term
        regt = [regt; length(term(:,1)), 0, 0, term(inv,1),0,0,c2temp]; %right termi
    end

    sq = sum(term(:,2));

end

end
```

Projection from regression tree

```
function [ y ] = projecttree( regtree, x0)
%Projection from created tree
%  x0 is input matrix each low corresponds to each dimension

y = zeros(1,length(x0(1,:)));

for i = 1:length(x0(1,:))
   nn = 1;
   d = regtree(nn,2);
   count = 0;
   while (d ~=0 ) && (count <10000)
       if x0(d,i)<regtree(nn,3)
           nn = regtree(nn,5); %next node
       else
           nn = regtree(nn,6);
```

```
        end
        d = regtree(nn,2);
        count = count+1;
    end
    y(i) = regtree(nn,7);

end



end
```

Tree-based regression for solving different interest rate problems with noise

```
clear;
clc;

dim = 25; %dimension
str = datestr(now,'yyyymmdd_HHMMSS');
fname = sprintf('spreadwithnoiseserpt_%d_%s.mat',dim,str);

x0 = 100;
mu = 0.05;
sig = 0.2;

T = 0.25;
r = 0.01;
R = 0.06;
K1 = 95;
K2 = 105;

gamma = 2.5; %beta = 1;

B = 1;


nu = 7:1:15;

numtest = length(nu);

N = floor(2*(sqrt(2).^(nu-1)));
M = floor(2*((N/2).^gamma));
N = 15*ones(1,numtest);

tre_lev = 20*ones(1,numtest);

pt = 1; %number of points searched for tree building


lb = 40; %lower bound for BM
ub = 180; %lower bound for BM
```

```
bound = [lb; ub];

bound = repmat(bound,1,dim);

h = T./N;

num_run = 1;
Y0 = zeros(num_run,length(N));

etime = [];

for round = 1:num_run
    round
    for i = 1:numtest
        if round ==1
            tic
        end

        T_ind = linspace(0,T,N(i)+1);

        x = zeros(dim,N(i),M(i));
        y = zeros(N(i),M(i));
        z = zeros(dim,N(i),M(i));

        w_mat = zeros(dim,N(i),M(i));

        %intitialize x

        w_mat(:,1,:) = sqrt(h(i))*randn(dim,1,M(i));

        for n = 2:N(i)
            w = sqrt(h(i))*randn(dim,1,M(i));
            w_mat(:,n,:) = w_mat(:,n-1,:)+w;
            x(:,n,:) = x0*exp((mu-sig*sig/2)*n*h(i)+sig*w_mat(:,n,:));
        end



        %intialize terminal y

        y(N(i),:) = max((squeeze(x(1,N(i),:))-K1),0)- 2*max((squeeze(x(1,N(i),:))-K2

        incy = y(N(i),:);

        %backward iteration
        for k = N(i)-1:-1:1
            k

            %find z
            for d = 1:dim
                if i == numtest
                    d
                end
                %size(incy)
```

```
            %size(squeeze(w_mat(d,k+1,:)-w_mat(d,k,:))'/h(i))
            residual = incy.*(squeeze(w_mat(d,k+1,:)-w_mat(d,k,:))')/h(i);

            for t = 1:B
                %find tree
                regt =  findtreeregressionsevpt(reshape(squeeze(x(:,k,:)),dim,M(

                %projection with tree in this step, final z is the sum from all
                %projection
                zproj =  projecttree( regt, reshape(squeeze(x(:,k,:)),dim,M(i))
                z(d,k,:) = z(d,k,:)+reshape(zproj,size(z(d,k,:)));

                %find residual for next step
                residual = residual - zproj;

            end
        end


        %find y coeff
        if dim==1
            f = -((mu-r)/sig)*(reshape(squeeze(z(:,k,:)),size(y(k+1,:))))-r*y(k+
        else
            f = -((mu-r)/sig)*sum(squeeze(z(:,k,:)))-r*y(k+1,:)-min(y(k+1,:)-sum
        end

        incy = incy + h(i)*f;

        residual = incy;

        for t = 1:B

            %find tree
            regt =  findtreeregressionsevpt(reshape(squeeze(x(:,k,:)),dim,M(i)),

            %projection with tree in this step, final z is the sum from all
            %projection
            yproj = projecttree(regt, reshape(squeeze(x(:,k,:)),dim,M(i)) );
            y(k,:) = y(k,:)+yproj;

            %find residual for next step
            residual = residual-yproj;
        end


    end

    %final step (k = 0) just find average
    last_z =zeros(1,dim);

    %find z
    for d = 1:dim
        inc = incy.*(reshape(squeeze(w_mat(d,1,:)),1,M(i)))/h(i);
        last_z(d) = last_z(d)+sum(inc);
```

```
        end
        last_z = last_z/M(i);

        %find y coeff
        f = -((mu-r)/sig)*sum(last_z)-r*y(1,:)-min(y(1,:)-sum(last_z)/sig,0)*(R-r);
        last_y = incy +h(i)*f;

        last_y = last_y/M(i);

        Y0(round,i) = sum(last_y);

        if round ==1
            etime(end+1) = toc
        end
    end

    if round == 1
        estimate = Y0(1,:);
        pic = figure;
        plot(M,estimate)
        hold on
        plot(M,2.95*ones(1,numtest),'-.m')
        hold off
    else
        estimate = sum(Y0(1:round,:))/round;
        var = sum((Y0(1:round,:)-repmat(estimate,round,1)).^2)/round;
        sd = sqrt(var);

        bias  = estimate-2.95*ones(1,numtest);

        pic = figure;

        plot(M,estimate)
        hold on
        plot(M(end),estimate(end)+2*sd(end),'r*')
        hold on
        plot(M(end),estimate(end)-2*sd(end),'r*')
        hold on
        plot(M,2.95*ones(1,numtest),'-.m')
        hold off
    end

    saveas(pic,str,'fig')
    save(fname)
end
```

Tree-based regression for solving different interest rate problems with weighted sum of Brownian motions

```
clear;
clc;
```

```
dim = 20; %dimension

str = datestr(now,'yyyymmdd_HHMMSS');

fname = sprintf('spreadwithnoisesevpthiddennonbm_%d_%s.mat',dim,str);

x0 = 100;
mu0 = 0.05;
sig0 = 0.2;

T = 0.25;
r = 0.01;
R = 0.06;
K1 = 95;
K2 = 105;

%mu = 0.1*rand(1,dim);
%sig = 0.5*rand(1,dim);

mu = (mu0)*ones(1,dim);
sig = (sig0)*ones(1,dim);

gamma = 2.5; %beta = 1;


B = 1;

pt = 1;

nu = 8:1:15;

numtest = length(nu);

N = floor(2*(sqrt(2).^(nu-1)));
M = floor(2*((N/2).^gamma));
N = 15*ones(1,numtest);

tre_lev = 100*ones(1,numtest);

lb = 40; %lower bound for BM
ub = 180; %lower bound for BM

bound = [lb; ub];

bound = repmat(bound,1,dim);

h = T./N;

num_run = 1;
Y0 = zeros(num_run,length(N));

tranx = rand(dim,dim);

etime = [];
```

```
for round = 1:num_run
    round
    for i = 1:numtest
        if round ==1
            tic
        end

        T_ind = linspace(0,T,N(i)+1);

        x = x0*ones(dim,N(i),M(i));
        y = zeros(N(i),M(i));
        z = zeros(dim,N(i),M(i));

        xhidden = zeros(dim,M(i));

        w_mat = zeros(dim,N(i),M(i));

        %intitialize x

        w_mat(:,1,:) = sqrt(h(i))*randn(dim,1,M(i));

        for n = 2:N(i)
            w = sqrt(h(i))*randn(dim,1,M(i));
            w_mat(:,n,:) = w_mat(:,n-1,:)+w;
            x(:,n,:) = x0*exp(repmat((mu'-sig'.*sig'/2)*n*h(i),1,1,M(i))+repmat(sig'
        end

        xhidden = x(1,N(i),:);

        %hide x
        for n = 2:N(i)
            x(:,n,:) = reshape(tranx*reshape(x(:,n,:),dim,M(i)),dim,1,M(i));
        end

        %intialize terminal y

        y(N(i),:) = max((xhidden-K1),0)- 2*max((xhidden-K2),0);

        incy = y(N(i),:);

        %backward iteration
        for k = N(i)-1:-1:1
            k

            %find z
            for d = 1:dim
                %size(incy)
                %size(squeeze(w_mat(d,k+1,:)-w_mat(d,k,:))'/h(i))
                residual = incy.*(squeeze(w_mat(d,k+1,:)-w_mat(d,k,:))')/h(i);

                for t = 1:B
                    %find tree
                    regt =  findtreeregressionsevpt(reshape(squeeze(x(:,k,:)),dim,M(
```

```
                %projection with tree in this step, final z is the sum from all
                %projection
                zproj =  projecttree( regt, reshape(squeeze(x(:,k,:)),dim,M(i))
                z(d,k,:) = z(d,k,:)+reshape(zproj,size(z(d,k,:)));

                %find residual for next step
                residual = residual - zproj;

            end
        end


        %find y coeff
        if dim==1
            f = -((mu(1)-r)/sig(1))*(reshape(squeeze(z(:,k,:)),size(y(k+1,:))))-
        else
            f = -((mu(1)-r)/sig(1))*sum(squeeze(z(:,k,:)))-r*y(k+1,:)-min(y(k+1,
        end

        incy = incy + h(i)*f;

        residual = incy;

        for t = 1:B

            %find tree
            regt =  findtreeregressionsevpt(reshape(squeeze(x(:,k,:)),dim,M(i)),

            %projection with tree in this step, final z is the sum from all
            %projection
            yproj = projecttree(regt, reshape(squeeze(x(:,k,:)),dim,M(i)) );
            y(k,:) = y(k,:)+yproj;

            %find residual for next step
            residual = residual-yproj;
        end


    end

    %final step (k = 0) just find average
    last_z =zeros(1,dim);

    %find z
    for d = 1:dim
        inc = incy.*(reshape(squeeze(w_mat(d,1,:)),1,M(i)))/h(i);
        last_z(d) = last_z(d)+sum(inc);
    end
    last_z = last_z/M(i);

    %find y coeff
    f = -((mu(1)-r)/sig(1))*sum(last_z)-r*y(1,:)-min(y(1,:)-sum(last_z)/sig(1),0
    last_y = incy +h(i)*f;
```

```
        last_y = last_y/M(i);

        Y0(round,i) = sum(last_y);

        if round ==1
            etime(end+1) = toc
        end
    end

    if round == 1
        estimate = Y0(1,:);
        pic = figure;
        plot(M,estimate)
        hold on
        plot(M,2.95*ones(1,numtest),'-.m')
        hold off
    else
        estimate = sum(Y0(1:round,:))/round;
        var = sum((Y0(1:round,:)-repmat(estimate,round,1)).^2)/round;
        sd = sqrt(var);

        bias  = estimate-2.95*ones(1,numtest);

        pic = figure;

        plot(M,estimate)
        hold on
        plot(M(end),estimate(end)+2*sd(end),'r*')
        hold on
        plot(M(end),estimate(end)-2*sd(end),'r*')
        hold on
        plot(M,2.95*ones(1,numtest),'-.m')
        hold off
    end

    saveas(pic,str,'fig')
    save(fname)

end
```

Tree-based regression for solving American basket option

```
clear;
clc;

dim = 3; %dimension
str = datestr(now,'yyyymmdd_HHMMSS');
fname = sprintf('spreadwithnoisesevptAmerican_%d_%s.mat',dim,str);

x0 = 100;
mu = 0.05;
```

```
sig = 0.8;

T = 1;
r = mu;
K = 100;

gamma = 3; %beta = 1;

findiffsol = 8.96;
B = 1;

nu = 7:1:12;

numtest = length(nu);

N = floor(2*(sqrt(2).^(nu-1)));
M = floor(2*((N/2).^gamma));
N = 50*ones(1,numtest);

tre_lev =200*ones(1,numtest);

pt = 1; %number of points searched for tree bd

lb = 40; %lower bound for BM
ub = 180; %lower bound for BM

bound = [lb; ub];

bound = repmat(bound,1,dim);

h = T./N;

num_run = 1;
Y0 = zeros(num_run,length(N));

etime = [];

for round = 1:num_run
    round
    for i = 1:numtest
        if round ==1
            tic
        end

        T_ind = linspace(0,T,N(i)+1);

        x = zeros(dim,N(i),M(i));
        y = zeros(N(i),M(i));
        ytemp = zeros(N(i),M(i));

        w_mat = zeros(dim,N(i),M(i));
        cap = log(N(i))*sqrt(h(i));
        %cap = 10;
        %intitialize x
```

```
w_mat(:,1,:) = sqrt(h(i))*randn(dim,1,M(i));
w_mat(:,1,:) = max(min(w_mat(:,1,:), cap),-cap);
x(:,1,:) = x0*exp((mu-sig*sig/2)*h(i)+sig*w_mat(:,1,:));

for n = 2:N(i)
    w = sqrt(h(i))*randn(dim,1,M(i));
    w_mat(:,n,:) = w_mat(:,n-1,:)+w;
    x(:,n,:) = x0*exp((mu-sig*sig/2)*n*h(i)+sig*w_mat(:,n,:));
end

%intialize terminal y

if dim ==1
    y(N(i),:) = max((K - reshape((x(1,N(i),:)),1,M(i))),0);
else
    y(N(i),:) = max((K - nthroot(prod(squeeze(x(:,N(i),:))),dim)),0);
end

ytemp(N(i),:) = y(N(i),:);

incy = y(N(i),:);

%backward iteration
for k = N(i)-1:-1:1
    k

    %find y coeff
    if dim==1
        f = -r*y(k+1,:);
    else
        f = -r*y(k+1,:);
    end

    incy = y(k+1,:) + h(i)*f;

    residual = incy;

    for t = 1:B

        %find tree
        regt =  findtreeregressionsevptv3(reshape(squeeze(x(:,k,:)),dim,M(i)

        %projection with tree in this step, final z is the sum from all
        %projection
        yproj = projecttree(regt, reshape(squeeze(x(:,k,:)),dim,M(i)) );
        y(k,:) = y(k,:)+yproj;

        %find residual for next step
        residual = residual-yproj;
    end

    if dim ==1
        ytemp = y(k,:);
```

```
                y(k,:) = max(max((K - reshape(x(1,k,:),1,M(i))),0),ytemp);
            else
                ytemp(k,:) = y(k,:);
                stepper = max(K - nthroot(prod(squeeze(x(:,k,:))),dim), ytemp(k,:))-
                y(k,:) = ytemp(k,:) + stepper./(1+50*dim*abs(stepper));
            end


        end

        %final step (k = 0) just find average

        %find y coeff
        f = -r*y(1,:);
        last_y = y(1,:) +h(i)*f;

        %Y0(round,i) = max(max(K-x0,0),sum(last_y)/M(i));
        Y0(round,i) = sum(last_y)/M(i);

        if round ==1
            etime(end+1) = toc
        end
    end

    if round == 1
        estimate = Y0(1,:);
        pic = figure;
        plot(M,estimate)
        hold on
        plot(M,findiffsol*ones(1,numtest),'-.m')
        hold off
    else
        estimate = sum(Y0(1:round,:))/round;
        var = sum((Y0(1:round,:)-repmat(estimate,round,1)).^2)/round;
        sd = sqrt(var);

        bias  = estimate-findiffsol*ones(1,numtest);

        pic = figure;

        plot(M,estimate)
        hold on
        plot(M(end),estimate(end)+2*sd(end),'r*')
        hold on
        plot(M(end),estimate(end)-2*sd(end),'r*')
        hold on
        plot(findiffsol*ones(1,numtest),'-.m')
        hold off
    end

    saveas(pic,str,'fig')
    save(fname)
end
```

Bibliography

[1] Vlad Bally, Denis Talay, et al. The law of the euler scheme for stochastic differential equations: Ii. convergence rate of the density. 1995.

[2] Christian Bender and Jessica Steiner. Least-squares monte carlo for backward sdes. In *Numerical methods in finance*, pages 257–289. Springer, 2012.

[3] Bruno Bouchard and Nizar Touzi. Discrete-time approximation and monte-carlo simulation of backward stochastic differential equations. *Stochastic Processes and their applications*, 111(2):175–206, 2004.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[6] Hugh A Chipman, Edward I George, Robert E McCulloch, et al. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

[7] Nicole El Karoui, Christophe Kapoudjian, Etienne Pardoux, Shige Peng, and Marie-Claire Quenez. Reflected solutions of backward sde's, and related obstacle problems for pde's. *the Annals of Probability*, pages 702–737, 1997.

[8] Nicole El Karoui and Laurent Mazliak. *Backward stochastic differential equations*, volume 364. CRC Press, 1997.

[9] Nicole El Karoui, Shige Peng, and Marie Claire Quenez. Backward stochastic differential equations in finance. *Mathematical finance*, 7(1):1–71, 1997.

[10] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer, 2004.

[11] Emmanuel Gobet and Jean-Philippe Lemor. Numerical simulation of bsdes using empirical regression methods: theory and practice. *arXiv preprint arXiv:0806.4447*, 2008.

[12] Emmanuel Gobet, Jean-Philippe Lemor, Xavier Warin, et al. A regression-based monte carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15(3):2172–2202, 2005.

[13] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[14] Jean-Philippe Lemor, Emmanuel Gobet, Xavier Warin, et al. Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. *Bernoulli*, 12(5):889–916, 2006.

[15] Francis A Longstaff and Eduardo S Schwartz. Valuing american options by simulation: A simple least-squares approach. *Review of Financial studies*, 14(1):113–147, 2001.

[16] Etienne Pardoux and Shige Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications*, pages 200–217. Springer, 1992.

[17] Steven E Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer, 2004.

[18] Lukasz Szpruch. V-stable tamed euler schemes. *arXiv preprint arXiv:1310.0785*, 2013.