OPTION AND CVA GREEKS WITH ADJOINT ALGORITHMIC DIFFERENTIATION

by

Antoine Collas (CID: 01425804)

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

Thesis submitted as part of the requirements for the award of the
MSc in Mathematics and Finance, Imperial College London, 2017-2018

Declaration

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

Signature and date:

Antoine COLLAS

11th September 2018

Acknowledgements

I would like to thank my supervisor Mr. Damiano Brigo for his guidance throughout the writing of this thesis. His knowledge and constant support contributed greatly to the results I obtained during these four months. I fully measure the luck I had to benefit from the advice of such a reference in mathematical finance.

Secondly, I would like to thank Mr. Luca Capriotti for taking the time to discuss one of his paper with me. A lot of the work in this thesis relies on Mr. Capriotti's papers and his insightful explanations have fed into the final writing.

Finally, many thanks to the Mazars' quant team for their trust and support during this internship. Without their help and guidance, it would have been impossible to perform such dissertation and obtain these results.

Abstract

This thesis focuses on the financial application of the *Algorithmic Differentiation* method (AD) to compute sensitivities of option prices and CVAs. The emphasis was put on Basket and Bermudan options as there are no other fast methods to compute their Greeks and CVA sensitivities. All the numerical results were obtained using an application built in C++ for the occasion.

Contents

1 Introduction

Since the financial crisis, institutions can no longer neglect credit risk and as such, market practices have considerably changed. Estimate first order derivatives of financial products' prices has become one of the biggest computational challenge. Therefore, discovering new ways to quantify efficiently sensitivities of financial instruments has become the cornerstone of risk management practices.

Additionally, institutions need to measure counterparty credit risk, leading to the notion of Credit Valuation Adjustment (CVA). Pricing a derivative and quantifying a CVA are closely linked. Indeed, CVA is obtained by pricing the counterparty risk of a deal using the same type of methods as those used to price financial products [1]. As such, the notion of CVA's sensitivities has emerged and is also crucial for hedging purposes.

The most widespread pricing techniques on financial markets are undoubtedly Monte-Carlo based methods as they are often the only way to price complex financial products. In parallel, sensitivities are mostly estimated using finite-differences methods. Even though these methods can be used where no other numerical methods would work, their main drawback is that they are very computationally expensive. In addition to this, the more sensitivities one wants to compute the longer the computation time will be.

In this context, the Algorithmic Differentiation (AD) method has become a hot topic in Finance as it enables to compute price and greeks of complex options saving computational time and without any loss in precision. Indeed, with AD, results are computed with no theoretical approximation: the only barrier is the machine precision level. The speed of this method is also very impressive as it enables to compute the function value and **all** its first order derivatives using only the time needed to compute 3 or 4 function values [19]. This result is all the more interesting that it is true no matter the number of derivatives computed (only one or ten thousands) !

However, the AD method is not free of drawbacks. Indeed, even if it enables to decrease impressively the time of computation, the method is memory consuming. Indeed, the technique is very expensive in terms of memory used as one needs to keep track of a lot of variables to compute efficient adjoint differentiations. Therefore the coding part of this method is quite challenging.

We make the choice to perform all the coding part in C++. While we could have used any object oriented programming language, C++ has features very powerful when computing the AD method. Indeed, C++ is known to be extremely fast and enables to manage memory allocation manually (which is not the case of Java for example). As memory management is the main challenge while coding the AD method, we have chosen to use C++ for all the simulations. Moreover the use of operator overloading often makes the AD coding part easier.

Again, as the price of financial products and their CVA are closely linked, the AD method applies very similarly to options and CVAs. Nevertheless, credit valuation adjustments are more complex instruments and thus require more simplifications and assumptions. This is why it is essential to have a good understanding of the Algorithmic Differentiation framework on option's prices before being able to apply it on CVAs.

Finally, we organize this thesis along four axis. First, we will describe the mathematical framework underlying the AD method. This step is crucial as the AD method is not only used in Finance and can be applied to compute the derivatives of any function. Secondly, we will explain how the AD method operates on financial products especially on European vanillas, Basket and Bermudan options, and their CVAs. The numerical results obtained with this method will be provided as well as relevant benchmarks. Then, we will highlight axis of research to apply the AD method under less assumptions to cope with the reality observed on financial markets (to take into account Wrong Way Risk on CVA for example). Finally, the last section will focus on the architecture of the C++ code. The design of this code was thought to be easily customizable so that new payoffs and dynamics of underlyings can be added as time goes.

2 Literature review

This thesis relies on many papers and books which give very important insights to perform the AD method on both derivatives and CVAs. It is important to note that what we call *AD method* in this thesis is also referred as AAD method in the literature. However, the AAD denomination is unclear and can both stand for *Automatic Algorithmic Differentiation* or *Adjoint Algorithmic Differentiation.*

In principle, the two denominations are not equivalent and what is treated in this thesis (and referenced as AD method) is closer to the *Adjoint Algorithmic Differentiation* definition. Indeed, *Automatic Algorithmic Differentiation* should refer to algorithms where all intermediate derivatives are also computed using AD while *Adjoint Algorithmic Differentiation* is more flexible and authorizes to compute intermediary quantities using finite differences for example. Authors often do not specify what they call AAD method as the difference is very thin. Nevertheless, it is worth noting that a proper *Automatic Algorithmic Differentiation* algorithm is extremely binding and needs special "manual" treatments. Therefore, although faster, proper *Automatic Algorithmic Differentiation* is rare in the literature.

Mark Henrard's book entitled *Algorithmic Differentiation in Finance explained* [19] gives precious explanations regarding the AD method on simple financial instruments such as European call options. Therefore this book has been of great help to understand the basic concepts underlying the AD method. Moreover, this publication details good practices while coding the AD method with an object oriented programming language. Hence, it was used as inspiration to write the Mathematical Preliminaries (Section 3) and many coding concepts developed in it were used in the coding process of this thesis.

The work performed on Basket and Bermudan options in this thesis relies on Mr. Capriotti's publications (see [11] and [12]). In one of his paper [11], Mr. Capriotti gives a framework to compute price and greeks of Bermudan options with the AD method using Monte-Carlo based algorithms. Moreover, he explains in detail how to perform the AD method on CVA for Bermudan options. This paper has been a great source of inspiration for this thesis and has been used to write the sections 4.2 and 5.2.

Finally, this thesis relies a lot on Mr Brigo's guidance and lecture notes (see [1]). The vast majority of concepts and notations used, especially regarding CVA, stem from Mr Brigo's work. These lecture notes give a recipe for the calculation of CVA and thus many concepts discussed in it were used in this dissertation.

Even if this thesis focuses mainly on the application of the AD method on options' prices and XVAs under strong assumptions (especially under independence between the time of default and options' payoff), we have undertaken further research. Indeed, the framework to quantify CVA's sensitivities is often too simplistic and one should take into account collateralization and Wrong Way Risk (WWR).

The reference [4] should be cited as it explains what happens to CVA under full collateralization but with instantaneous contagion at default, leading to sizeable gap risk CVA even in absence of explicit jumps in the dynamics.

Moreover, previous works on WWR should be cited as they were of great help to understand this more realistic XVA framework. We give a short list here:

- WWR on CDS without collateral: [8];
- WWR on Credit products (CDS) with collateral and gap risk: [4];
- WWR on rates: [6];
- WWR on rates with collateral: [5];
- WWR on commodities (Oil): [7];
- WWR on equity: [2].

3 Mathematical preliminaries

At first, the *Algorithmic Differentiation (AD)* method was not created directly for financial applications. Indeed, this is a very general method that can be applied on any function in order to compute its partial derivatives. Therefore, it is crucial to understand how the method can be applied on an arbitrary function in order be able to use it in more complex contexts such as Finance.

3.1 Mathematical framework

The AD method is meant to provide the partial derivatives of any function and is based on a very simple mathematical concept: the chain rule for the composition of functions.

Definition 3.1 (Chain rule)**.** Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be $\mathcal{C}^1$ functions. Then,

$$(fog)^{'} = (f^{'}og)g^{'}.$$

There are two ways to apply the AD method, which only differ according to the manner the chain rule is performed. We call these two versions of the *Algorithmic Differentiation*: forward and backward/adjoint accumulations (or modes).

Definition 3.2 (Forward mode)**.** Let n be an integer and $f_n : \mathbb{R} \to \mathbb{R}$ be $\mathcal{C}^1$ functions.

$$\forall n,\ f_n(x_n) = x_{n+1}.$$

The *Algorithmic Differentiation*'s forward mode describes the following recursive relation:

$$\frac{\delta x_i}{\delta x_0} = \frac{\delta x_i}{\delta x_{i-1}}\frac{\delta x_{i-1}}{\delta x_0} = \frac{\delta x_i}{\delta x_{i-1}}\frac{\delta x_{i-1}}{\delta x_{i-2}}\frac{\delta x_{i-2}}{\delta x_{i-3}}...\frac{\delta x_2}{\delta x_1}\frac{\delta x_1}{\delta x_0}. \tag{3.1}$$

Definition 3.3 (Adjoint mode)**.** Let n be an integer and $f_n : \mathbb{R} \to \mathbb{R}$ be $\mathcal{C}^1$ functions.

$$\forall n,\ f_n(x_n) = x_{n+1}.$$

The *Algorithmic Differentiation*'s adjoint mode describes the following recursive relation:

$$\frac{\delta x_i}{\delta x_0} = \frac{\delta x_i}{\delta x_1}\frac{\delta x_1}{\delta x_0} = \frac{\delta x_i}{\delta x_2}\frac{\delta x_2}{\delta x_1}\frac{\delta x_1}{\delta x_0} = \frac{\delta x_i}{\delta x_3}\frac{\delta x_3}{\delta x_2}\frac{\delta x_2}{\delta x_1}\frac{\delta x_1}{\delta x_0} = ... \tag{3.2}$$

Remark 3.4 (Notations)**.** Let $f : \mathbb{R} \to \mathbb{R}$ be a $\mathcal{C}^1$ function and x in $\mathbb{R}$.

In the following sections we will use the notations below:

$$\dot{f} = \frac{\delta f}{\delta x} \qquad \bar{x} = \frac{\delta f}{\delta x}.$$

3.2 Application of the forward and backward modes

The AD method is highly visual and therefore can be easily understood on a simple example. Hence, this section highlights how the two modes would apply to a simple function. As they stem from the same underlying concept (a.k.a chaine rule), the two versions are of course equivalent.

3.2.1 Application of the forward accumulation

Consider the following function:

$$z = f(x, y) = \log(\cos(y) + \exp(x + 1)).$$

We are interested in the partial derivative of f with respect to x. Hence, we apply the AD method in forward mode to obtain it. First, we need to decompose the function in order to write $z = f(x, y)$ as the composition of several functions (step 1). Then, we can apply the Forward Algorithmic Differentiation according to the recursive relation 3.1 as explained in step 2:

1) Function decomposition	2) Forward accumulation
$a = x + 1$	$\dot{a} = \frac{\delta a}{\delta x} = 1.0$
$b = y$	$\dot{b} = \frac{\delta b}{\delta x} = 0.0$
$c = \exp(a)$	$\dot{c} = \frac{\delta c}{\delta x} = \frac{\delta c}{\delta a}\frac{\delta a}{\delta x} = \dot{a}c = c$
$d = \cos(b)$	$\dot{d} = \frac{\delta d}{\delta x} = \frac{\delta d}{\delta b}\dot{b} = 0.0$
$e = c + d$	$\dot{e} = \frac{\delta e}{\delta c}\frac{\delta c}{\delta x} + \frac{\delta e}{\delta d}\frac{\delta d}{\delta x} = \dot{c} + \dot{d} = c$
$f = \log(e)$	$\dot{f} = \frac{\delta f}{\delta e}\frac{\delta e}{\delta x} = \frac{\dot{e}}{e} = \frac{c}{e}$

Table 1: Forward Algorithmic Differentiation on a multivariate function

3.2.2 Application of the adjoint accumulation

Consider again the following function:

$$z = f(x, y) = \log(\cos(y) + \exp(x + 1)).$$

The same way as before, we apply the AD method in adjoint mode on f to obtain $\frac{\delta f}{\delta x}$. Again, we need to decompose the function and apply the Adjoint Algorithmic Differentiation on this segmentation according to the recursive relation 3.2:

1) Function decomposition	2) Adjoint accumulation
$a = x + 1$	$\bar{f} = 1.0$
$b = y$	$\bar{e} = \frac{\bar{f}}{e} = \frac{1}{e}$
$c = \exp(a)$	$\bar{d} = \bar{e} = \frac{1}{e}$
$d = \cos(b)$	$\bar{c} = \bar{e} = \frac{1}{e}$
$e = c + d$	$\bar{b} = -\bar{d}\sin(b)$
$f = \log(e)$	$\bar{a} = c\bar{c} = \frac{c}{e}$

Table 2: Adjoint Algorithmic Differentiation on a multivariate function

3.2.3 Equivalence between the two modes

The two previous computations give the same results, giving a good intuition of the equivalence between the two modes. In both cases, the derivative of z with respect to x is given by:

$$\frac{\delta z}{\delta x} = \frac{\exp(x+1)}{exp(x+1) + \cos(y)}.$$

Remark 3.5 (Equivalence)**.** Even if the two modes are mathematically equivalent, the adjoint mode is often preferred in financial applications. Therefore, in what follows we only use the AD method in its adjoint version. It explains why, on financial markets, this technique is often referred as *Adjoint Algorithmic Differentiation.*

3.3 A simple financial application

Now that the general AD framework has been introduced for any multivariate function, we present on a simple option how this method applies in finance. The main idea is simple: regard payoffs (or directly prices if a closed-form exists) as multivariate functions of their parameters (e.g their volatility, strike, time to maturity, etc ...)

The use of the AD method on a basic European Call option in Black-Scholes settings is likely to be the simplest financial application of this technique. This example closely follows the work of Marc Henrard in his book *Algorithmic Differentiation in Finance explained* [19]. European Call option, as often, is the best way to familiarize with the AD method in a financial framework and understand the mathematical concepts underlying Algorithmic Differentiation.

3.3.1 Mathematical development

Definition 3.6 (Black-Scholes Model)**.** Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and let $(W_t)_{t\geq 0}$ be a Brownian motion. In the Black-Scholes model, the stock price process $(S_t)_{t\geq 0}$ is the unique strong solution to the following stochastic differential equation:

$$\frac{\mathrm{d}S_t}{S_t} = r\mathrm{d}t + \sigma \mathrm{d}W_t, \qquad S_0 > 0, \tag{3.3}$$

where $r \geq 0$ denotes the instantaneous risk-free interest rate and $\sigma > 0$ the instantaneous volatility. A European call price $C_t(S_0, K, \sigma)$ with maturity $T > 0$ and strike $K > 0$ pays at maturity $(S_T - K)_+ = \max(S_T - K, 0)$. When the stock price follows the Black-Scholes SDE (3.3), there is a closed form for the price of such option, given by:

$$C_0(S_0, K, \sigma) = S_0\mathcal{N}(d_+) - K\mathrm{e}^{-rT}\mathcal{N}(d_-),$$

where

$$d_+ := \frac{\log(S_0/K) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}, \qquad d_- := d_+ - \sigma\sqrt{T}.$$

and where $\mathcal{N}$ denotes the cumulative distribution function of the Gaussian random variable.

The AD method on a European call option under Black and Scholes dynamic consists in two steps. First, we perform a forward sweep and decompose the closed form of the price of a call option. Then, we work backward using the adjoint mode on this decomposition.

This example gives a very fast way to compute greeks and price of European options as we do not work on the payoff but directly on the closed form of the price. Thus, no Monte Carlo simulations are needed and the price and greeks are directly obtained after the backward sweep.

We denote by ϕ the probability density function of a Gaussian random variable and get a pseudo-code close to the one given in Mark Henrard's book [19, page 32-33]:

Price closed-form decomposition	Adjoint accumulation
$periodVolatility = \sigma\sqrt{T}$	$\overline{price} = 1.0$
$d_+ = \frac{\log(S_0/K)+rT}{periodVolatility} + \frac{1}{2}periodVolatility$	$\overline{N_-} = -K\mathrm{e}^{-rT}\overline{price}$
$d_- = d_+ - periodVolatility$	$\overline{N_+} = S_0$
$N_+ = \mathcal{N}(d_+)$	$\overline{d_-} = \phi(d_-)\overline{N_-}$
$N_- = \mathcal{N}(d_-)$	$\overline{d_+} = \phi(d_+)\overline{N_+} + \overline{d_-}$
$price = S_0N_+ - K\mathrm{e}^{-rT}N_-$	$\overline{periodVolatility} = (\frac{-\log(S_0/K)+rT}{periodVolatility^2} + 0.5)\overline{d_+} - \overline{d_-}$

Table 3: Adjoint Algorithmic Differentiation on European call

We can then access to the Greeks by computing the adjoint of model parameters using a scheme similar to the one stated by Mr. Henrard [19, page 34]:

Sensitivities		AAD method result
$\frac{\delta C_0}{\delta \sigma}$	(vega)	$\bar{\sigma} = \sqrt{T}\overline{periodVolatility}$
$\frac{\delta C_0}{\delta T}$	(-theta)	$\overline{T} = r\mathrm{e}^{-rT}KN_{-}\overline{price} + \frac{\sigma}{2\sqrt{T}}\overline{periodVolatility} + \frac{r}{periodVolatility}\overline{d_+}$
$\frac{\delta C_0}{\delta K}$		$\overline{K} = -\mathrm{e}^{-rT}N_{-}\overline{price} - \frac{1}{KperiodVolatility}\overline{d_+}$
$\frac{\delta C_0}{\delta r}$	(rho)	$\bar{r} = KT\mathrm{e}^{-rT}N_{-}\overline{price} + \frac{T}{periodVolatility}\overline{d_+}$
$\frac{\delta C_0}{\delta S_0}$	(delta)	$\overline{S_0} = N_{+}\overline{price} + \frac{1}{S_0 periodVolatility}\overline{d_+}$

Table 4: Greeks using AAD on European call

Remark 3.7 (optimised AD algorithm)**.** An expert eye, would see (running the code) that $\overline{d_+}$ always equals 0 as highlighted in *Algorithmic Differenciation in Finance Explained* by M. Henrard [19]. Therefore we can simplify the previous algorithm, setting $\overline{d_+} = 0$. Hence, we get an *optimised* AD algorithm (see [19, page 34]).

3.3.2 Computation and results

All the previous mathematical development have been coded in C++. In addition to the implementation of the simple and optimised AD method, we reused a previous algorithm to compute price and greeks closed forms as well as Monte-Carlo and Likelihood Ratio methods (see Simulation Methods for Finance coursework [14]). In the table below, we give the price and greeks obtained with each methods and the computation time needed to estimate all these quantities.

	Closed Form	AAD method	Optimized AAD	Likelihood Ratio Method
Price	19.9077	19.9077	19.9077	19.9009
Delta	0.702412	0.702412	0.702412	0.701676
Vega	48.9911	48.9911	48.9911	49.0197
Rho	100.667	100.667	100.667	100.697
Theta	-3.56528	-3.56528	-3.56528	-3.6345
Computation Time (s)	0.002	0.003	0.002	3.254

Table 5: European Call Price and Greeks for $S_0 = 100$, $r = 0.01$, $T = 2$, $\sigma = 0.25$, $K = 90$ and $N_{MC} = 10^6$ simulations for Likelihood Ratio Method

Even if there exist closed forms for the greeks in this case (and therefore there is no need to use AD to have accurate and fast results), we can already see with this example the precision and speed of such method. A more complex payoff would involve more work but the idea remains the same: first decompose the payoff of the option, then compute the adjoint of each variable involved in the decomposition to get estimates.

4 AD method on financial derivatives

In most of the cases, there are no closed-form formulae for the price of an option. Therefore, the AD procedure is slightly different and involves a Monte-Carlo simulation on the estimates that stem from the application of the AD method on the option's payoff.

4.1 AD method on basket option

In this section we consider a basket option as described in L. Capriotti's paper [12, page 19].

Definition 4.1 (Simple basket call option)**.** Let's consider a basket of n stocks $S^1, \dots, S^n$ with respective weights $\omega_1, \dots, \omega_n$. The payoff for a simple basket call option is given by:

$$\mathrm{e}^{-rT}(\sum_{i=1}^{n} \omega_i S_T^i - K)^+.$$

4.1.1 Payoff decomposition and application of the AD method

In his paper, L. Capriotti provides a pseudo code to compute the AD method on a basket option ([12, page 20]). Nevertheless, this pseudo-code focuses on the payout function which, in general, has no notion of how the underlyings have been derived (e.g. what SDE was used) and in particular their possible dependence on interest rate, dividend yield and volatilities etc.

In our framework, whatever model one chooses, the drift will always be the risk free rate r, so it is model independent to some extent. Hence, under the risk pricing measure, the underlyings S^i depend on r and therefore an additional term has to be added to the computation of $\bar{r}$ and $\overline{T}$ performed in Mr. Capriotti's paper [12].

Hence we suggest the following application of the AD method on a basket option:

Payoff decomposition	AAD method
$B = 0.0$	$\overline{price} = 1.0$
$B = \sum_{i=1}^{n} \omega_i S_T^i$	$\overline{D} = max(x, 0.0) * \overline{price}$
$x = B - K$	$\bar{x} = D * \mathbb{1}_{\{x>0\}} * \overline{price}$
$D = \mathrm{e}^{-rT}$	$\overline{B} = \bar{x}$
$price = D * max(x, 0.0)$	$\overline{S_T^i} = \omega_i \overline{B}$

Table 6: Adjoint Algorithmic Differentiation on Basket option

Then we get the following expression for rho, -thetha and vega:

$$\bar{r} = -TD\overline{D} + \sum_{i=1}^{n} \omega_i \frac{\delta S_T^i}{\delta r} \overline{B},$$

$$\overline{T} = -rD\overline{D} + \sum_{i=1}^{n} \omega_i \frac{\delta S_T^i}{\delta T} \overline{B},$$

$$\bar{\sigma} = \sum_{i=1}^{n} \omega_i \frac{\delta S_T^i}{\delta \sigma_i} \overline{B}.$$

Remark 4.2 (Model dependence)**.** In the previous application of the AAD method, $\frac{\delta S_T^i}{\delta r}$, $\frac{\delta S_T^i}{\delta T}$ and $\frac{\delta S_T^i}{\delta \sigma_i}$ depend on the dynamics of the inherent underlying.

In our implementation, we assume that the risk free rate r and the volatility σ are constant. We consider only two possible dynamics for the underlying stocks (a basket can possibly contain stocks in different models). The underlying can follow a Bachelier or a Black and Scholes model:

Bachelier model:

$$\mathrm{d}S_t^i = r\mathrm{d}t + \sigma_i \mathrm{d}W_T^i, \qquad S_T^i = S_0^i + rT + \sigma_i\sqrt{T}Z_i, \tag{4.1}$$

$$\frac{\delta S_T^i}{\delta r} = T, \qquad \frac{\delta S_T^i}{\delta T} = r + \frac{\sigma_i Z_i}{2\sqrt{T}}, \qquad \frac{\delta S_T^i}{\delta \sigma_i} = \sqrt{T}Z_i. \tag{4.2}$$

Black and Scholes:

$$\mathrm{d}S_t^i = rS_t^i\mathrm{d}t + \sigma_i S_t^i \mathrm{d}W_T^i, \qquad S_T^i = S_0^i \exp((r - \frac{\sigma_i^2}{2})T + \sigma_i\sqrt{T}Z_i), \tag{4.3}$$

$$\frac{\delta S_T^i}{\delta r} = TS_T^i, \qquad \frac{\delta S_T^i}{\delta T} = [(r - \frac{\sigma_i^2}{2} + \frac{\sigma_i Z_i}{2\sqrt{T}})]S_T^i, \qquad \frac{\delta S_T^i}{\delta \sigma_i} = [\sqrt{T}Z_i - \sigma_i T]S_T^i. \tag{4.4}$$

where Z_1, ... , Z_n are iid standard normal variables.

Remark 4.3 (Computation of delta)**.** The decomposition of the payoff using the AD method gives $\overline{S_T^i} = \frac{\delta price}{\delta S_T^i}$. We would like to have *delta* (i.e $\frac{\delta price}{\delta S_0^i}$) instead. To get it, we use the following property:

$$\frac{\delta price}{\delta S_0^i} = \frac{\delta price}{\delta S_T^i}\frac{\delta S_T^i}{\delta S_0^i} = \overline{S_T^i}\frac{\delta S_T^i}{S_0^i}.$$

We just have to replace the quantity $\frac{\delta S_T^i}{\delta S_0^i}$ (which is model dependent) by the relevant expression given by: $\frac{S_T^i}{S_0^i}$ in the Black and Scholes model and 1.0 in the Bachelier model.

Remark 4.4 (Monte-Carlo methodology)**.** As we work on the option's payoff and no longer on the price as in 3.3, we need to perform Monte-Carlo simulations on this previous decomposition to have the price and greeks of the option.

- 1. Simulate the n underlying stocks S_T^1, ... , S_T^n feeding the basket.
- 2. Decompose the payoff of the inherent basket option as explained before.
- 3. Repeat M times 1. and 2. and average the $\overline{parameter}$ obtained on these M simulations i.e $\overline{parameter} = \frac{1}{M}\sum_{i=1}^{M}\overline{parameter}^i$.

4.1.2 Closed-form formulae in the normal case

As a benchmark, we compute the closed-form obtained for a basket option where all underlyings follow independent Bachelier models (ie independent normal laws).

Let assume that underlyings S_t^1, ... , S_t^n follow respectively independent normal laws with mean $S_0 + rt$ and variance $\sigma_i^2 t$ under the pricing measure (see 4.1). Then, using the fact that the sum of independent normals is a normal variable, we have:

$$Y = \sum_{i=1}^{n}\omega_i S_T^i \rightsquigarrow \mathcal{N}(\sum_{i=1}^{n}\omega_i(rT + S_0^i), \sum_{i=1}^{n}\omega_i^2\sigma_i^2 T).$$

We define the following quantities:

$$\mu_y = \sum_{i=1}^{n}\omega_i(rT + S_0^i), \qquad \sigma_y = \sqrt{\sum_{i=1}^{n}\omega_i^2\sigma_i^2 T}, \qquad \bar{x} = \frac{K-\mu_y}{\sigma_y}.$$

Then the price of the basket option is given by:

$$\begin{aligned}
C_0 &= \mathbb{E}^Q[\mathrm{e}^{-rT}(\sum_{i=1}^{n}\omega_i S_T^i - K)^+] \\
&= \mathbb{E}^Q[\mathrm{e}^{-rT}(Y-K)^+] \\
&= \mathbb{E}^Q[\mathrm{e}^{-rT}(\sigma_y Z - (K-\mu_y))^+], \qquad \textit{where Z is a standard normal random variable}, \\
&= \mathrm{e}^{-rT}\int_{\bar{x}}^{\inf}\frac{\sigma_y \mathrm{e}^{\frac{-z^2}{2}}z}{\sqrt{2\pi}} - \mathrm{e}^{-rT}(K-\mu_y)\mathcal{N}(-\bar{x}) \\
&= \mathrm{e}^{-rT}\sigma_y\phi(-\bar{x}) - \mathrm{e}^{-rT}(K-\mu_y)\mathcal{N}(-\bar{x}),
\end{aligned}$$

where $\mathcal{N}$ denotes the cumulative distribution function of the Gaussian random variable and ϕ denotes the probability density function of the Gaussian random variable.

4.1.3 Greeks using pathwise derivatives

Following the methodology given in Simulation Methods Lectures for Finance (SMF) [22], we have a recipe to compute *delta* with respect to the i^{th} underlying:

$$\begin{aligned}\frac{\delta C_0}{\delta S_0^i} &= \mathbb{E}^Q[\mathrm{e}^{-rT}\frac{\delta(Y-K)^+}{\delta S_0^i}] \\ &= \mathbb{E}^Q[\mathrm{e}^{-rT}\frac{\delta(Y-K)^+}{\delta Y}\frac{\delta Y}{\delta S_0^i}] \\ &= \mathbb{E}^Q[\mathrm{e}^{-rT}\mathbb{1}_{\{Y>K\}}\omega_i]\, under\, Bachelier's\, model.\end{aligned}$$

We can also compute *rho* as follows:

$$\begin{aligned}\frac{\delta C_0}{\delta r} &= \mathbb{E}^Q[\mathrm{e}^{-rT}\frac{\delta(Y-K)^+}{\delta r}] - T\mathbb{E}^Q[\mathrm{e}^{-rT}(Y-K)^+] \\ &= \mathbb{E}^Q[\mathrm{e}^{-rT}\frac{\delta(Y-K)^+}{\delta Y}\frac{\delta Y}{\delta r}] - T\mathbb{E}^Q[\mathrm{e}^{-rT}(Y-K)^+] \\ &= \mathbb{E}^Q[\mathrm{e}^{-rT}\mathbb{1}_{\{Y>K\}}\sum_{i=1}^n \omega_i\frac{\delta S_T^i}{\delta r}] - T\mathbb{E}^Q[\mathrm{e}^{-rT}(Y-K)^+] \\ &= \mathrm{e}^{-rT}\mathbb{E}^Q[\mathbb{1}_{\{Y>K\}}\sum_{i=1}^n \omega_i T - T(Y-K)^+]\, under\, Bachelier's\, model.\end{aligned}$$

We can then apply Monte-Carlo framework to have a good benchmark for our greeks' computations.

4.1.4 Numerical results

We performed $N_{MC} = 1.000.000$ Monte-Carlo simulations to compute the price and greeks of a basket option with expiry $T = 2.0$ and strike price $K = 88$ built on 3 stocks following independent Bachelier's model under a constant risk free rate $r = 0.01$. We give the volatility and initial price of each underlying stocks:

$$\sigma_1 = 0.25 \qquad \sigma_2 = 0.3 \qquad \sigma_3 = 0.1,$$

$$S_0^1 = 100.0 \qquad S_0^2 = 82.0 \qquad S_0^3 = 97.0.$$

The results are presented in the table next page and highlight the strength of the AD method: no loss in precision and a speed which does not depend on the number of Greeks computed !

	AAD method	Closed Form
Price	478.453	478.455
Sensitivity wrt 1th stock in Basket	2.9406	2.9406
Sensitivity wrt 2nd stock in Basket	0.980199	0.980199
Sensitivity wrt 3rd stock in Basket	1.9604	1.9604
Rho	-945.143	-945.141
Theta	4.72615	not computed
Vega	-0.00810242	not computed
Computation Time (s)	8.046	10.127

Table 7: Basket Price and Greeks

In this simulation we see that the AD method is very accurate and extremely fast. Indeed, to compute the price and 6 Greeks, the AD method is faster than the Pathwise Derivative approach to compute only 2 Greeks !

4.2 AD method on Bermudan option

Luca Capriotti ([11]) gives a framework to understand the computation of the AD method on any Bermudan option. We recall that a Bermudan option can be exercised at several fixed dates determined in the contract. The idea is similar to what we have done previously on basket options but the optimality of the boundary makes the decomposition of the payoff and therefore the Monte-Carlo computation far more complicated. The balance between computation time and memory management is also very challenging and will be discussed.

Hence, this section follows Mr Capriotti's work and we use mostly the notations introduced in his paper [11, page 38-42]. Nevertheless, we try to highlight simplifications that can be performed and particular choices that are made in our case.
As always, when using AD method, there are two steps: the payoff decomposition and then the backward induction.

As in the previous section, the code inherent to this section enables to choose the dynamic of the underlying. For sake of clarity, we only detail here the particular case of an underlying following a Black-Scholes model.

4.2.1 Forward sweep: pricing techniques

Let's consider an arbitrary Bermudan option built on an underlying asset S_t following a Geometric Brownian motion

$$\mathrm{d}S_t = rS_t\mathrm{d}t + \sigma S_t\mathrm{d}W_t, \qquad X_0 = x > 0.$$

We consider that this option expires at time T. The option can be exercised at M discrete times T_1, ... , T_M which constitute our discretized time grid of $[0, T]$.
In what follows, E_m denotes the payoff of the function at time T_m. We also denote by V_m^n the price of the Bermudan option at time T_m for the n^{th} Monte-Carlo simulation.

Remark 4.5. We work under a constant risk free rate r. Therefore the numeraire N_i at time T_i does not depend on the simulation nor the time grid i.e:

$$N_1^n = ... = N_M^n = \mathrm{e}^{\frac{-rT}{M}}.$$

The main idea to price Bermudan option is simple (and is developed in greater details in the literature [22]). One has to find the hold value at all times T_m i.e the continuation value if the Bermudan option is not exercised at time 0. Then, the option is exercised at time T_m only if the exercise value E_m is greater than the hold value H_m. Therefore, the price of the option at time T_m is only the maximum between the hold value H_m and the exercise value E_m.

Regression based algorithms are meant to estimate the hold value H_m at any time T_m rather than finding its exact value. As stated in Capriotti's paper [11], the hold value is estimated by a regression on a vector of basis functions $\psi(x)$. Therefore, β being the vector of regression coefficients, the estimate of the hold value will be:

$$H_m(x) = \beta^T\psi(x).$$

The main challenge of such method is to find basis functions giving accurate results on a large range of assets. This topic will be discussed later in 4.2.2. Below, we state the methodology to price such Bermudan option, following Mr. Capriotti's paper [11].

1. We first simulate N_{MC} independent paths of an underlying following a GBM [11, step (R1) page 38]. To do this, we use the Euler Method to generate paths i.e:

$$\forall m \in [0, M],\ S_{m+1}^n = S_m^n + rS_m^n h + \sigma S_m^n\sqrt{h}Z_m^n,$$

where $h = T/M$, n is the n^{th} Monte-Carlo simulation, Z_m^n are iid standard normal random variables.

2. We then work backward for every simulated paths (ie for every n=1, ..., N_{MC}) [11, steps (R2) and (R3) page 38].

- We first set $V_M^n = E_M^n$.
  Then working backward for $m = M-1, .., 1$, we do the following:

$$\Psi_m = \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \psi_m^n (\psi_m^n)^T,$$

$$\Omega_m = \frac{\mathrm{e}^{\frac{-rT}{M}}}{N_{MC}} \sum_{i=1}^{N_{MC}} \psi_m^n V_{m+1}^n,$$

  where ψ_m^n denotes the value of the vector of basis functions used to perform the regression evaluated at S_m^m i.e at time T_m and for the n^{th} simulation.

- We then compute the regression coefficients:

$$\beta_m = \Psi_m^{-1} \Omega_m.$$

- And estimate the hold value:

$$H_m^n = \beta_m^T \psi_m^n.$$

3. Finally, we have to choose one of the 3 following estimates of the option's value [11, step (R4) page 38]):

- Longstaff and Schwartz [11]:

$$V_m^n = \begin{cases} E_m^n & if\ E_m^n > H_m^n \\ V_{m+1}^n \mathrm{e}^{\frac{-rT}{M}} & otherwise. \end{cases}$$

  Then the Monte-carlo estimate of the option price is given by:

$$V_0 = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} V_1^n \mathrm{e}^{\frac{-rT}{M}}.$$

- Classic Regression based Monte-Carlo [11]:

$$V_m^n = max(E_m^n, H_m^n).$$

  Then the Monte-carlo estimate of the option price is given by:

$$V_0 = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} V_1^n \mathrm{e}^{\frac{-rT}{M}}.$$

- Lower bound algorithm for Bermudan-style options [11]:

  Compute the path-wise estimator for the discounted cashflows of the option:

$$P^n = \sum_{m=1}^{M} [(\prod_{m=1}^{m-1} \mathbb{1}_{H_i^n > E_i^n}) \mathbb{1}_{H_m^n < E_m^n} \mathrm{e}^{\frac{-mrT}{M}} E_m^n].$$

  Then the Monte-carlo estimate of the option price is given by:

$$V_0 = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} P^n.$$

4.2.2 Choice of basis functions

We use regression based methods in order to price Bermudan-style options. The first issue triggered by such methods is the choice of the basis functions denoted by the vector ψ. One needs to take an orthogonal polynomial basis in order to be able to price complicated options.

An Hermite polynomials basis was chosen as it fulfills all requirements. We constructed the basis as follows:

$$H_0(x) = 1.0 \qquad H_1(x) = 2x \qquad H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-1}(x)$$

One needs to be careful when choosing the number of basis functions to find the best fit. On the one hand, a lack of basis function triggers underfitting issues and therefore lead to inaccurate regression results. On the other hand, too many basis functions brings overfitting problems. Moreover, the more basis functions the slower the algorithm will be.

Therefore, an analysis had been performed separately on every options treated in order to find the right balance between goodness of fit and reasonable computation time. In the Bermudan vanilla case, the best trade-off is obtained with 3 basis functions while for the best of two assets case 13 basis functions are needed.

Results which led us to the choice of 3 basis functions are given in the table below for the Bermudan Put case using a Binomial tree as benchmark [14]:

	2 basis	3 basis	5 basis	10 basis	Binomial Tree
Price	12.3294	13.0096	11.5158	11.3083	13.0676

Table 8: Bermudan Put price for $S_0 = 100.0$, $r = 0.05$, $T = 1.0$, $\sigma = 0.4$, $K = 100.0$

4.2.3 Comparison of the different pricing techniques on Bermudan put

We consider the previous framework and work on a Bermudan put option with payoff $E_m^n = (K - S_m^n)^+$.

We have implemented the 3 methods explained previously [11] for this option and compared the results. We have chosen 3 basis functions as it appeared to be the best balance between accuracy and computation time.

In our simulations, the expiry date is three years ($T = 3.0$) and we can exercise the option every 3 months ($M = 12$). We consider also that $S_0 = 1.0$, $\sigma = 0.2$, $r = 0.15$.
We then run simulations for $K = 0.9$, $K = 1.0$, $K = 1.1$ and for $N_{MC} = 500.000$ Monte Carlo simulations.

	Option Value	Computation Time
$K = 0.9$	0.0195459	50.721 seconds
$K = 1.0$	0.0434925	51.521 seconds
$K = 1.1$	0.0907178	52.862 seconds

Table 9: Classic Regression Based-Monte Carlo

	Option Value	Computation Time
$K = 0.9$	0.0159796	50.531 seconds
$K = 1.0$	0.0410524	51.611 seconds
$K = 1.1$	0.0894568	52.141 seconds

Table 10: Longstaff and Schwartz

	Option Value	Computation Time
$K = 0.9$	0.0162559	51.81 seconds
$K = 1.0$	0.0410968	53.767 seconds
$K = 1.1$	0.0893994	54.998 seconds

Table 11: Lower-bound algorithm

	Option Value
$K = 0.9$	0.0157926
$K = 1.0$	0.0421816
$K = 1.1$	0.1

Table 12: Binomial tree

We used the implementation of the Binomial Tree method from the coursework submitted in SMF [14] in order to benchmark our results. If the number of nodes is not too important, the Binomial Tree method is of course faster than Monte-Carlo based methods. Therefore on this example with 12 exercise dates, comparing the computation time of Monte-Carlo methods to the Binomial Tree is totally irrelevant.

The previous tables highlight that Longstaff and Schwartz estimate seems to give slightly more accurate results than the other methods. Therefore we use this technique in our implementation of the forward sweep in the AD method.

4.2.4 Backward sweep

Even if all the previous pricing techniques can be used in the AD method (and have been implemented), as the Longstaff and Schwartz is the most accurate method, we will only develop in this section the backward sweep on this estimate of the option value to obtain the Greeks.

To have further details regarding the methodology underlying the use of the two other estimates in the backward sweep of the AD method (especially using the Lower-Bound algorithm), please refer to Mr. Capriotti's paper [11, pages 42-44]. The three resulting AD methods have been implemented and the code architecture enables to choose which method the user wants to use.

The following steps have been also largely inspired by Capriotti's paper [11, page 44] and give the methodology to obtain the adjoint of models parameters.

- First, we initialize the adjoint of the option value, models parameters and regression coefficients. Their values are given by: $\overline{V_0} = 1.0$, $\bar{\sigma} = 0$, $\overline{S_0} = 0$ and $\overline{\beta_m} = 0$ for $m = 1, ..., M-1$. [11, step ($\overline{R4}$) page 44]

  We then set for $n = 1, ..., N_{MC}$:

$$\begin{cases} \overline{V_1^n} = \frac{\overline{V_0}}{N_{MC}} \mathrm{e}^{\frac{-rT}{M}} \\ \overline{N_1^n} = -\frac{\overline{V_0}}{N_{MC}} V_1^n \mathrm{e}^{-\frac{2rT}{M}}. \end{cases}$$

- For $m = 1, ..., M-1$, we compute [11, step ($\overline{R3}$) page 44]:

$$\forall n, \begin{cases} \overline{E_m^n} = \overline{V_m^n} \mathbb{1}_{E_m^n > H_m^n} \\ \overline{H_m^n} = \overline{V_m^n} \mathbb{1}_{E_m^n < H_m^n}, \end{cases}$$

$$\overline{S_m^n} = \overline{E_m^n} \frac{\delta E_m^n}{\delta S_m^n}.$$

  Remark 4.6. The quantity $\frac{\delta E_m^n}{\delta S_m^n}$ depends on the payoff of the option considered. In the numerical examples, we will explain how we obtain this quantity using the AD method on the payoff E_m^n.

- We then compute, as in [11, ($\overline{R3}$) page 44], the adjoint the basis functions ψ_m^n and update the adjoint of β as follows:

$$\bar{\beta_m} += \sum_{n=1}^{N_{MC}} \psi_m^n \overline{H_m^n},$$

$$\overline{\psi_m^n} = \beta_m \overline{H_m^n}.$$

Finally we compute the adjoint of the variables Ω_m and Ψ_m setting:

$$\overline{\Omega_m} = \Psi_m^{-T}\overline{\beta_m}, \qquad \overline{\Psi_m} = -\overline{\Omega_m}\beta_m^T.$$

- For $n = 1, ..., N_{MC}$, as in [11, ($\overline{R}3$) page 44] compute or update:

$$\begin{aligned}
\overline{\psi_m^n} += & \frac{1}{N_{MC}} \mathrm{e}^{\frac{-rT}{M}} V_{m+1}^n \overline{\Omega_m},\\
\overline{V_{m+1}^n} = & \frac{1}{N_{MC}} \mathrm{e}^{\frac{-rT}{M}} (\psi_m^n)^T \overline{\Omega_m},\\
\overline{\psi_m^n} += & \frac{1}{2N_{MC}} (\overline{\Psi_m} + (\overline{\Psi_m})^T)\psi_m^n,\\
\overline{S_m^n} += & \overline{\Psi_m} \frac{\delta\overline{\Psi_m}}{\delta S_m^n}.
\end{aligned}$$

Remark 4.7. Computing efficiently the quantity $\frac{\delta\overline{\Psi_m}}{\delta S_m^n}$ is quite challenging if we want to save memory. We decided to compute it as follow:

$$\begin{aligned}
\frac{\delta\overline{\Psi_m}}{\delta S_m} &= \frac{\delta\overline{\Psi_m}}{\delta V_{m+1}} \frac{\delta V_{m+1}}{\delta E_{m+1}} \frac{\delta E_{m+1}}{\delta S_{m+1}} \frac{\delta S_{m+1}}{\delta S_m},\\
\frac{\delta V_{m+1}}{\delta E_{m+1}} &= \mathbb{1}_{E_{m+1} > H_{m+1}},\\
\frac{\delta\overline{\Psi_m}}{\delta V_{m+1}} &= \frac{1}{N_{MC}} \mathrm{e}^{\frac{-rT}{M}} \overline{\Omega_m},\\
\frac{\delta S_{m+1}}{\delta S_m} &= 1.0 + rh + \sigma\sqrt{h} Z_{m+1}.
\end{aligned}$$

Again the quantity $\frac{\delta E_{m+1}}{\delta S_{m+1}}$ depends on the payoff of the option and is treated in numerical examples. Note that the quantity $\frac{\delta S_{m+1}^n}{\delta S_m^n}$ will in general depend on the dynamic of the underlying (and therefore of the Euler Path expression).

- Finally, as in [11, ($\overline{R}3$) page 44] we compute:

$$\overline{S_M^n} = \overline{V_M^n} \frac{\delta E_M^n}{\delta S_M^n}.$$

- We then get the greeks [11, ($\overline{R}1$) page 44], by differentiating the Euler path with respect to the desired parameter as follows:

$$\forall m \in [|0, M|], \ \overline{S_m^n} += \overline{S_{m+1}^n}[1.0 + rh + \sigma\sqrt{h} Z_m^n].$$

Then,

$$\overline{S_0} = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \overline{S_0^n},$$

Similarly,

$$\forall m \in [|0, M|], \ \bar{\sigma} += \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \overline{S_{m+1}}^n [\sqrt{h} S_m^n Z_m^n].$$

4.2.5 Example of a Bermudan put

We coded carefully the general steps developed above, the only step "specific" to the option's payoff being the computation of $\frac{\delta E_{m+1}}{\delta S_{m+1}}$. We computed this quantity using the AD method on the payoff (to cope with the rules of *Automatic Algorithmic Differentiation*) as follows:

Payoff decomposition	AAD method
$u = S_m^n$	$\bar{p} = 1.0$
$v = K - u$	$\bar{v} = \mathbb{1}_{\{v>0\}}\bar{p}$
$p = max(v, 0)$	$\bar{u} = \frac{\delta E_m}{\delta S_m} = -\bar{v}$

Table 13: Adjoint Algorithmic Differentiation on Put payoff

In our simulations, the expiry date is 3 years ($T = 3.0$) and we can exercise the option every three months ($M = 12$). We consider also that $S_0 = 1.0$, $\sigma = 0.2$, $r = 0.15$, K=0.9. Moreover we have used an extended binomial tree as a benchmark of our results [14]:

	AD method	Extended binomial tree
Price	0.0159705	0.0157926
Delta	-0.191629	-0.195008
Vega	0.285818	0.293983

Table 14: AD method on Bermudan Put with 50.000 Monte-Carlo simulations

4.2.6 Example of a Bermudan best of two assets (put/call)

We also considered a Bermudan call best of two assets option [11, page 44] built on two underlyings S_1 and S_2. The payoff of this option at exercise date T_m is given by:

$$(max(S_1(T_m), S_2(T_m)) - K, 0.0)^+.$$

The method previously detailed is very general and therefore can be applied to such payoff. Only two steps should be modified to take into account the fact that this payoff implies the use of two underlyings.

First, Hermite polynomials cannot be used anymore as basis for the regression, therefore, we decided to use the following polynomial basis as in [11, page 45]:

$$1,\ S_1,\ S_2,\ S_1S_2,\ S_1^2,\ S_2^2,\ S_1^3,\ S_2^3,\ S_1S_2^2,\ S_2S_1^2.$$

And we added to this basis the following polynomial functions of the payoff [11, page 45]:

$$(max(S_1, S_2) - K, 0.0)^+, \ ((max(S_1, S_2) - K, 0.0)^+)^2, \ ((max(S_1, S_2) - K, 0.0)^+)^3.$$

Finally, we needed to adapt the computation of $\frac{\delta E_m}{\delta S_m}$ using AD as follows:

Payoff decomposition	Adjoint method
$u = S^1_m$	$\bar{p} = 1.0$
$v = S^2_m$	$\bar{y} = \mathbb{1}_{y>0}\bar{p}$
$w = max(u, v)$	$\overline{w} = \bar{y}$
$y = w - K$	$\bar{v} = \frac{\delta E_m}{\delta S^2_m} = \mathbb{1}_{\{v>u\}}\overline{w}$
$p = max(y, 0)$	$\bar{u} = \frac{\delta E_m}{\delta S^1_m} = \mathbb{1}_{\{u>v\}}\bar{v}$

Table 15: Adjoint Algorithmic Differentiation on Put payoff

Similar work (modulo a change of basis and of sign in the computation of $\frac{\delta E_m}{\delta S_m}$) has been performed on the Bermudan put best of two assets option. We performed our simulations with the following parameters: $S^1_0 = 90$, $S^2_0 = 100$, $K = 100$, $r = 0.04$, $T = 1$, $\sigma = 0.4$, $M = 50$ and 13 basis functions. The results obtained with these two options after 100.000 Monte-Carlo simulations are given in the tables below:

	AD method	Monte-Carlo/FD
$Price$	7.98615	8.04129
$\frac{\delta Price}{\delta S^1_0}$	-0.207099	-0.202373
$Vega$	18.0393	18.18032

Table 16: Bermudan Best of two Put

	AD method	Monte-Carlo/FD
$Price$	26.1643	26.1464
$\frac{\delta Price}{\delta S^1_0}$	0.155386	0.158404
$Vega$	47.3297	47.1026

Table 17: Bermudan Best of two Call

To benchmark our results, we also simulate the Bermudan best of two assets put with $S^1_0 = S^2_0 = 1.0$, and using the same random numbers to simulate both paths. Hence, the paths generated for S^1 and S^2 are identical. Therefore, in this settings we get a Bermudan put option built on S^1 for example. The following table gives the results obtained:

	Best of two Assets Put	Bermudan Put	Extended Binomial Tree
$Price$	0.0157612	0.0157905	0.0157926
$Delta$	-0.188449	-0.191629	-0.195008
$Vega$	0.300706	0.285818	0.293983

Table 18: Comparison of the different methods with $S^1_0 = 1.0$, $S^2_0 = 1.0$, $K = 0.9$, $r = 0.15$, $T = 3.0$, $\sigma = 0.2$, $M = 12$ and 13 basis functions.

Results are accurate and stable. The computation time needed to compute the price and the greeks is approximatively equivalent to 3 times the time needed to just price the option. This fact remains true if we increase the number of greeks computed as the only limit is the memory of your computer ! Therefore, AD is extremely powerful when one wants to compute more than 10 greeks: the computation time involved remains the same when computing one or ten thousands greeks.

We worked on Bermudan vanilla options in one or two dimensions. One should notice that the framework created would remain the same for more complex Bermudan options and the AD method would become the only reasonable way to price and quantify sensitivities of such options as Finite Difference algorithms would involve hours of computations.

We extend our work on options' prices in the next sections, using similar AD framework on Credit Valuation Adjustments (CVA). To be able to perform AD computations under such financial instrument, we will work on strong assumptions namely:

- Unilateral counterparty risk: only one counterparty is risky, the other is default-free.
- Independence between the time of default and the equity payoff.

Even if one could argue that this framework is not realistic while applied to "real" financial markets, it enables to obtain interesting theoretical results and convincing numerical computations.

5 AD method on CVA

The AD method can be applied very similarly to CVA. However, the complex structure of this financial instrument makes the decomposition of the CVA expression for the forward sweep more challenging than for options' payoff. Therefore, we first begin with the simplest case of a non-Bermudan option with a positive payoff giving birth to many simplifications in our calculations. Then, we will treat the general Bermudan case. In this section, we always consider a unilateral CVA setting.

5.1 CVA of non-Bermudan options with positive payoff

5.1.1 Hypothesis and notations

The notations used in this section stem from D. Brigo lectures notes [1]. As in the latter, we will call the "default-free" inverstor "B" and "C" the risky counterparty who may default. Therefore, we decide to work in an unilateral counterparty risk setting.

$\pi_B(t,T)$ will denote the discounted payoff without default risk seen by "B". Therefore it will be the sum of all future cashflows between t and T discounted back to t (see [1]).

In what follows, we consider European call or Basket options with maturity T so that $\pi_B \geq 0$. Indeed,

$$\pi_B(t,T) = \mathrm{e}^{-r(T-t)}(S_T - K)^+ \qquad or \qquad \pi_B(t,T) = \mathrm{e}^{-r(T-t)}(\sum_{i=1}^{n} S_T^i - K)^+.$$

Hence we write $\pi_B(t,T) = \mathrm{e}^{-r(T-t)}Payoff^+$.

We model the default time of "C" using a stochastic intensity model (without jumps) and assume independence between the counterparty default time and the contract underlying.

5.1.2 Intensity model used

We make the assumption that the default time τ is exponentially distributed. We denote the default intensity at time t by λ_t and the associated cumulated intensity by $\Lambda_t = \int_0^t \lambda_s ds$.

Then, as defined in [1, slide 666], τ is the inverse of the cumulated intensity on an exponential random variable ξ with mean 1 and independent of λ.

$$\tau = \Lambda^{-1}(\xi).$$

We also recall that:

$$Q(\xi > x) = \mathrm{e}^{-x} \qquad \textit{and} \qquad Q(\xi < x) = 1 - \mathrm{e}^{-x}. \tag{5.1}$$

5.1.3 Useful property

The following property will be of great help in our computations and stem from [1, slide 669]:

$$\begin{aligned} Q(\tau > t) &= Q(\Lambda^{-1}(\xi) > t) \\ &= Q(\xi > \Lambda(t)) \\ &= \mathbb{E}[Q(\xi > \Lambda(t)|\Lambda(t)] \ \textit{using Tower and independence properties} \\ &= \mathbb{E}[\mathrm{e}^{-\Lambda(t)}] \\ &= \mathbb{E}[\mathrm{e}^{-\int_0^t \lambda_s ds}]. \end{aligned}$$

5.1.4 Default intensity SDE

We assume that λ follows a CIR process without jumps (to ensure positiveness of λ_t as it can be seen as the local probability of defaulting around t).

$$d\lambda_t = k(\mu - \lambda_t)dt + \nu\sqrt{\lambda_t}d\omega_t \quad \textit{with} \ \ 2k\mu > \nu^2. \tag{5.2}$$

Therefore, we discretize $[0, T]$ and, using Euler scheme to simulate paths on the discrete time grid $T_1, \dots, T_M$, we simulate the intensity process as follows:

$$\lambda_{t_{i+1}} = \lambda_{t_i} + k(\mu - \lambda_{t_i})h + \nu\sqrt{\lambda_{t_i}}\sqrt{h}Z_i,$$

where $h = \frac{T}{M}$ and Z_i are iid standard normal variables.

5.1.5 Default Bucketing approximation

The default bucketing technique will enable to benchmark our CVA results. Again, this technique relies on our discretization of the interval $[0,T]$ into M segments of equal length determined by the following times: $T_1, \dots, T_M = T$. Then we proceed following Mr. Brigo's guidance [1, slide 860]:

$$\begin{aligned} CVA(0) &= L_{GD}\mathbb{E}_0[\mathbb{1}_{\tau<T}\mathrm{e}^{-r\tau}(\mathbb{E}_\tau[\pi(\tau,T)])^+] \\ &= L_{GD}\mathbb{E}_0[\sum_{j=1}^{M}\mathbb{1}_{\tau\in(T_{j-1},T_j]}\mathrm{e}^{-r\tau}\mathbb{E}_\tau[\pi(\tau,T)]] \quad , as \ \ \pi(\tau,T) \geq 0 \\ &\approx L_{GD}\sum_{j=1}^{M}\mathbb{E}_0[\mathbb{1}_{\tau\in(T_{j-1},T_j]}\mathrm{e}^{-rT_j}\mathbb{E}_{T_j}[\pi(T_j,T)]] \\ &= L_{GD}\sum_{j=1}^{M}\mathbb{E}_0[\mathbb{E}_{T_j}[\mathbb{1}_{\tau\in(T_{j-1},T_j]}\mathrm{e}^{-rT}Payoff^+]] \end{aligned}$$

$$\begin{aligned} CVA(0) &= L_{GD} \sum_{j=1}^{M} \mathbb{E}_0[\mathbb{1}_{\tau \in (T_{j-1},T_j]} \mathrm{e}^{-rT} Payoff^+] \\ &= L_{GD} \sum_{j=1}^{M} \mathbb{E}_0[\mathbb{1}_{\tau \in (T_{j-1},T_j]}] \mathbb{E}_0[\mathrm{e}^{-rT} Payoff^+], \ \ \tau \perp\!\!\!\perp Payoff^+ \\ &= L_{GD} \sum_{j=1}^{M} Q(\tau \in (T_{j-1}, T_j]) PriceOption_0. \end{aligned}$$

Therefore we just need to estimate $Q(\tau \in (T_{j-1}, T_j]) = Q(\tau > T_{j-1}) - Q(\tau > T_j)$. This is straightforward as,

$$\begin{aligned} \forall m \leq M, \quad Q(\tau > T_m) &= \mathbb{E}_0[\mathrm{e}^{-\int_0^{T_m} \lambda_s ds}] \\ &\approx \mathbb{E}_0[\mathrm{e}^{-\sum_{j=0}^{m-1} \lambda_j (T_{j+1} - T_j)}] \\ &\approx \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \mathrm{e}^{-\sum_{j=0}^{m-1} \lambda_j^i (T_{j+1} - T_j)}. \end{aligned}$$

Hence using the relevant value for $PriceOption_0$ we get easily the CVA of Call, Basket or other options with positive payout.

5.1.6 Application on the CVA of an European call option

We start with the following expression obtained while computing CVA using default bucketing method:

$$CVA(0) = \mathbb{E}_0[L_{GD} \sum_{j=1}^{M} \mathbb{1}_{\tau \in (T_{j-1},T_j]} \mathrm{e}^{-rT} (S_T - K)^+].$$

Before being able to perform the AD method on this expectation, we need a little bit of work to obtain an adequate form to perform the forward decomposition (first step of the AD method). First, we need to exhibit the dependence of $\mathbb{1}_{\tau \in (T_{i-1},T_i]}$ to λ in order to compute $\frac{\delta CVA}{\delta \lambda}$.

First note that,

$$\mathbb{1}_{\{\tau \in (T_{i-1},T_i]\}} = \mathbb{1}_{\{\tau > T_{i-1}\}} \mathbb{1}_{\{\tau < T_i\}}.$$

Moreover,

$$\begin{aligned} \mathbb{1}_{\{\tau > T_{i-1}\}} &= \mathbb{1}_{\{\Lambda^{-1}(\xi) > T_{i-1}\}} \\ &= \mathbb{1}_{\{\xi > \Lambda(T_{i-1})\}} \\ &= \mathbb{1}_{\{\xi > \int_0^{T_{i-1}} \lambda_s ds\}} \\ &\approx \mathbb{1}_{\{\xi > \sum_{j=1}^{i-1} (T_j - T_{j-1}) \lambda_j\}} = \mathbb{1}_{\{\xi > b_i\}}. \end{aligned}$$

Similarly,

$$\mathbb{1}_{\{\tau<T_i\}} = \mathbb{1}_{\{\xi<a_i\}}$$

with $a_i = \sum_{j=1}^{i}(T_j - T_{j-1})\lambda_j$ and $b_i = \sum_{j=1}^{i-1}(T_j - T_{j-1})\lambda_j$.

Remark 5.1 (Simulation of exponential variable)**.** We simulate the exponential variable ξ using the inverse transform method. First, we simulate a uniform random variable on [0,1] called U. Then,

$$\xi = F^{-1}(U) = -\log(U).$$

Hence we can estimate $\mathbb{1}_{\xi>b_i}$ and $\mathbb{1}_{\xi<a_i}$ for every i.

The idea behind this decomposition of the indicator function is to exhibit its dependence to λ. Then, we will be able to calculate the adjoint of a_i and b_i in order to get the adjoint of λ and therefore $\frac{\delta CVA}{\delta\lambda}$.

The AD method applies as follows to compute $\frac{\delta CVA}{\delta\lambda}$:

$$\begin{aligned}
u &= S_T \\
v &= u - K \\
w &= max(v, 0.0) \\
p &= \mathrm{e}^{-rT} w \\
a_i &= \sum_{j=1}^{i}(T_j - T_{j-1})\lambda_j \quad \forall i \\
b_i &= \sum_{j=1}^{i-1}(T_j - T_{j-1})\lambda_j \quad \forall i \\
x &= \sum_{i=1}^{M} \mathbb{1}_{\xi<a_i}\mathbb{1}_{\xi>b_i} \\
y &= L_{GD}xp \\
\bar{y} &= 1.0 \\
\bar{x} &= L_{GD}p\bar{y} \\
\overline{b_i} &= \begin{cases} \frac{-1}{2\delta}\mathbb{1}_{\xi<a_i}\bar{x} & if \;\; b_i \in [\xi-\delta, \xi+\delta] \\ 0 & otherwise \end{cases} \\
\overline{a_i} &= \begin{cases} \frac{1}{2\delta}\mathbb{1}_{\xi>b_i}\bar{x} & if \;\; a_i \in [\xi-\delta, \xi+\delta] \\ 0 & otherwise \end{cases} \\
\overline{\lambda_0} &= \sum_{i=1}^{M}(T_1 - T_0)[\overline{a_i} + \overline{b_i}]
\end{aligned}$$

with δ the regularization parameter to approximate the indicator function by the following:

$$\mathbb{1}_{a_i > \xi} \approx \begin{cases} 0 \quad if \;\; a_i \leq \xi - \delta \\ \frac{a_i}{2\delta} + c \quad if \;\; a_i \in [\xi - \delta, \xi + \delta] \\ 1 \quad if \;\; a_i \geq \xi + \delta \end{cases}$$

and similar work have been done for $\mathbb{1}_{b_i < \xi}$

5.1.7 CVA closed-form

Limitation of the method previously used

To compute $\overline{a_i}$ and $\overline{b_i}$ we had to smoothen the indicator function so that it becomes continuous and we can somehow differentiate it.

The main concern with this approach is that we have to choose the regularization parameter δ so that the function is "continuous enough" but choices of "big δ" would not represent an indicator function anymore. Therefore the result is very dependent on the choice of δ and thus not really satisfactory.

Even though the AD method is very fast compared to the use of the finite difference method to compute $\frac{\delta CVA}{\delta\lambda}$, it is very dependent on the choice of the regularization parameter δ and a good choice of δ seems purely qualitative.

To get rid of this dependence on the regularization factor, one could suggest to approximate the Dirac function by a gaussian with very little variance but again the results are very dependent on the variance chosen and therefore results are not satisfactory.

The idea is to change the filtration under which we compute the CVA in order to get rid of the indicator function in the expression of the CVA.

Change of filtration and closed-form computation

Let's denote the filtration of default-free market variables by $\mathbb{F}_t$ and assume:

$$\mathbb{G}_t = \mathbb{F}_t \cup (\cup_i \sigma(\tau_i \leq u, u \leq t)$$

with i indexing the default times of the system [1, slide 837].

Hence, following the notations from Mr. Brigo's lecture notes [1] and using the same type of intensity model 5.1.2, we get the following expression for the CVA by using the tower property:

$$\begin{aligned} CVA(0) &= L_{GD}\mathbb{E}_0[\mathbb{1}_{\tau<T}\mathrm{e}^{-r\tau}(\mathbb{E}_\tau[\pi(\tau,T)])^+] \\ &= L_{GD}\mathbb{E}_0[\mathbb{1}_{\tau<T}\mathrm{e}^{-r\tau}(\mathbb{E}_\tau[\mathrm{e}^{-r(T-\tau)}Payoff^+])^+] \\ &= L_{GD}\mathbb{E}_0[\mathbb{E}_\tau[\mathbb{1}_{\tau<T}\mathrm{e}^{-r\tau}\mathrm{e}^{-r(T-\tau)}Payoff^+]] \\ &= L_{GD}\mathbb{E}_0[\mathbb{1}_{\tau<T}\mathrm{e}^{-rT}Payoff^+]. \end{aligned}$$

We then use the immersion hypothesis meaning that we switch from filtration $\mathbb{G}$ to filtration $\mathbb{F}$. Indeed switching to the filtration $\mathbb{F}$ will transform $\mathbb{1}_{\tau>t}$ into its $\mathbb{F}$ expectation $\mathrm{e}^{-\Lambda(T)}$ as explain in 5.1.3 [1].

$$\begin{aligned} CVA(0) &= L_{GD}\mathbb{E}_0[\mathbb{E}[\mathbb{1}_{\tau<T}\mathrm{e}^{-rT}Payoff^+|\mathbb{F}_T]] \\ &= L_{GD}\mathbb{E}_0[\mathbb{E}[\mathbb{1}_{\tau<T}|\mathbb{F}_T]\mathrm{e}^{-rT}Payoff^+] \\ &= L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\Lambda(T)})\mathrm{e}^{-rT}Payoff^+] \\ &= L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\int_0^T\lambda_s ds})\mathrm{e}^{-rT}Payoff^+] \\ &\approx L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\sum_{i=0}^M\lambda_i(T_i-T_{i-1})})\mathrm{e}^{-rT}Payoff^+]. \end{aligned}$$

We can then apply the AD method on this new form (in the case of a European Call to fix ideas) and get the following pseudo code:

```
$u = S_T$
$v = u - K$
$w = max(v, 0.0)$
$P = \mathrm{e}^{-rT}w$
$x = 1 - \mathrm{e}^{-\sum_{i=0}^M\lambda_i(T_i-T_{i-1})}$
$y = L_{GD}\,P\,x$
$\bar{y} = 1.0$
$\bar{x} = L_{GD}\,\bar{y}\,P$
$\overline{\lambda_0} = (T_1-T_0)\mathrm{e}^{-\sum_{i=0}^M\lambda_i(T_i-T_{i-1})}\bar{x}.$
```

Again, this formulation is not good enough as the result for the sensitivity with respect to λ_0 would be dependent on the discretization step while approximating the integral.

Nevertheless, as the default intensity is independent of the option's discounted payoff and follows a CIR process, we can obtain the following closed-form solution:

$$\begin{aligned} CVA(0) &= L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\int_0^T \lambda_s ds})\mathrm{e}^{-rT}Payoff^+] \\ &= L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\int_0^T \lambda_s ds})]\mathbb{E}_0[\mathrm{e}^{-rT}Payoff^+]\,, as\, \lambda \perp\!\!\!\perp S_T. \end{aligned}$$

We then use the fact that $\mathbb{E}_0[\mathrm{e}^{-\int_0^T \lambda_s ds}] = A(0,T)\mathrm{e}^{-B(0,T)\lambda_0}$ with:

$$A(0,T) = (\frac{2h\mathrm{e}^{(k+h)\frac{T}{2}}}{2h+(k+h)(\mathrm{e}^{hT}-1)})^{\frac{2k\mu}{\nu^2}}, \qquad B(0,T) = \frac{2h(\mathrm{e}^{hT}-1)}{2h+(k+h)(\mathrm{e}^{hT}-1)}, \qquad h = \sqrt{k^2+2\nu^2}.$$ [20]

Finally we obtain the following closed-form expression:

$$\begin{aligned} CVA(0) &= L_{GD}\mathbb{E}_0[(1-\mathrm{e}^{-\int_0^T \lambda_s ds})]Price_0 \\ &= L_{GD}(1-A(0,T)\mathrm{e}^{-\lambda_0 B(0,T)})Price_0. \end{aligned}$$

This expression enables to compute $\frac{\delta CVA}{\delta \lambda_0}$ and all other sensitivities using the relevant decomposition of $Price_0$ to perform the AD method on it as detailed in 3.3 for European vanilla options or 4.1 for Basket options. Therefore we get a proper *Automatic Algorithmic Differentiation* algorithm for CVAs on options with positive payoffs, given by the following pseudo code:

```
$u = 1 - A(0,T)\mathrm{e}^{-B(0,T)\lambda_0}$
$p = LGD\, Price_0\, u$
$\bar{p} = 1.0$
$\bar{u} = LGD Price_0\, \bar{p}$
$\overline{\lambda_0} = A(0,T)B(0,T)\mathrm{e}^{-B(0,T)\lambda_0}\bar{u}.$
```

5.1.8 Numerical results

We compare the two approaches on an European Call option using the following parameters for the underlying stock: $S_0 = 100.0$, $r = 0.01$, $T = 2.0$, $\sigma = 0.25$ and $K = 90.0$.

In our simulations we use the following intensity model parameters (using 5.2 notations) for the time of default: $\lambda_0 = 1.0$, $k = 0.5$, $\mu = 1.0$, $\nu = 0.25$ and a $L_{GD} = 0.6$.

To benchmark our results, we calculate CVA values using default bucketing method and CVA's sensitivities using a finite difference approach (FD). We also use a regularization parameter for the inherent method which value is given by $\delta = 0.001$. Hence, we get the following table:

	AD with reg.	AD with change of filt.	Default bucketing/FD	FD on closed-form
CVA	10.24	10.2608	10.2734	10.2608
$\frac{\delta CVA}{\delta \lambda_0}$	1.8278	2.07621	1.976	2.07626
Computation Time	405 seconds	0.001 seconds	$\geq$ 45 minutes	398 seconds

Table 19: AD method on European Call CVA with 500.000 Monte-Carlo simulations

Even optimizing "manually" the choice of the best regularization parameter δ for the first method, we obtain poor results for $\frac{\delta CVA}{\delta \lambda_0}$. However, with the closed-form obtained under the immersion hypothesis we get perfectly accurate and instantaneous results (less than one second to get all the sensitivities !). Indeed, the fact that we apply AD on a closed form enables to have a computation time below 0.001 seconds.

To highlight the power of this change of filtration, we present further computations using the AD method on this CVA closed-form for options with positive payoff. This time, we only use a finite difference approach (FD) to benchmark our results for $\frac{\delta CVA}{\delta \lambda_0}$ (as in the 4^{th} column of Table 19). We start with the same parameters as those used above. We only specify the value of the parameter which has changed compared to Table 19.

	FD on the closed form	AD with change of filtration
CVA ($S_0 = 100$)	10.2608	10.2608
$\frac{\delta CVA}{\delta \lambda_0}$ ($S_0 = 100$)	2.07626	2.07621
CVA ($S_0 = 110$)	14.113	14.113
$\frac{\delta CVA}{\delta \lambda_0}$ ($S_0 = 110$)	2.85566	2.85574
CVA($S_0 = 90$)	6.91399	6.91399
$\frac{\delta CVA}{\delta \lambda_0}$ ($S_0 = 90$)	1.399	1.39903
CVA($K = 100$)	7.68221	7.68221
$\frac{\delta CVA}{\delta \lambda_0}$ ($K = 100$)	1.55444	1.55448
CVA($\sigma = 0.4$)	14.0955	14.0955
$\frac{\delta CVA}{\delta \lambda_0}$ ($\sigma = 0.4$)	2.8522	2.85212
CVA($T = 1.0$)	5.97897	5.97897
$\frac{\delta CVA}{\delta \lambda_0}$ ($T = 1.0$)	2.74747	2.74744

Table 20: AD method on European Call CVA with 500.000 Monte-Carlo simulations

5.2 AD method on CVA of Bermudan options

In this section we follow carefully the work made by Luca Capriotti and use his notations [11, page 40-43]. As in section 5.1, we only work in a unilateral counterparty risk framework, assuming that the default time τ of the risky counterparty is independent of the portfolio values V(τ).

5.2.1 Forward sweep: CVA computation using Least-Square Monte-Carlo

We start from the following well-known form of CVA under default-bucketing assumption and with the same notations introduced in 4.2 and 5.1 [11, page 38]:

$$CVA(0) = \mathbb{E}_0[L_{GD}\sum_{i=1}^{M}(Q(\tau > T_{i-1}) - Q(\tau > T_i))\mathrm{e}^{-rT_i}(V(T_i))^+] \tag{5.3}$$

The computation of CVA values using regression based algorithm is very similar to the method introduced to price Bermudan options. Here we consider that the option can be exercised p times at time T_1, ... , T_p and its payoff is given by E_{T_m} at time T_m.

One has to be careful as many indices are involved. Indeed, using the default bucketing assumption we discretize $[0, T]$ according to the discretization time-grid T_1, ..., T_M where T_i is not necessarily one of the exercise dates T_1, ... , T_p. Thus, the two discrete time grids can be different and this fact needs to be taken into account while performing the computations.

The following steps stem from Luca Capriotti's paper [11, page 43] and enable to compute CVA values. They constitute the forward sweep of the AD method. We use the index $n = 1$, ... , N_{MC} to denote each Monte-Carlo simulations and proceed as follows:

1. We first simulate the paths S^n_m of the underlying asset (respectively λ^n_m of the counterparty default intensity) for all time horizon T_1, ... T_m, ... , T_M . Note that the code created enables to choose the dynamic of the underlying and therefore to choose between the normal or lognormal dynamic for the underlying stock while using Euler paths approximation. However, for the sake of clarity (and without loss of generality), we here discuss the lognormal case (where the underlying follows a Black Scholes model).

$$\forall m \in [0, M-1],\ S^n_{m+1} = S^n_m + rS^n_m h + \sigma S^n_m\sqrt{h}Z^n_m,$$

$$\forall m \in [0, M-1],\ \lambda^n_{m+1} = \lambda^n_m + k(\mu - \lambda^n_m)h + \nu\sqrt{\lambda^n_m}\sqrt{h}Z_i,$$

with obvious notations [11, (X1) page 40].

2. Then, we compute the paths of the survival probabilities [11, (X2) page 40] for $m \in [0, M]$.

$$Q^n(\tau > T_m) = \mathrm{e}^{-\sum_{j=0}^{m-1}\lambda^n_j(T_{j+1}-T_j)}.$$

3. Following exactly the same steps as in 4.2.1, we get the following estimate of the hold value:

$$\forall m \in [1, M-1],\ \ H_m^n = \beta_m^T \psi_m \qquad H_M^n = 0.$$

Then, we get the estimate of the value of the option [11, (X3) page 40] as follows:

$$\forall m \in [1, M],\ \ V_m^n = \begin{cases} max(E_m^n, H_m^n) & if\, T_m\, is\, an\, exercise\, date \\ H_m^n \quad otherwise, & \end{cases}$$

with E_m^n the exercise value of the value for the n^{th} MC simulation at T_m.

4. The estimator of the CVA value for the n^{th} simulation [11, (X4) page 40] is then given by:

$$CVA^n(0) = \sum_{m=1}^{M} L_{GD}(Q(\tau > T_{m-1}) - Q(\tau > T_m))\mathrm{e}^{-rT_m}(V_m^n)^+.$$

5. Finally, we get the MC estimate as follows:

$$CVA(0) = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} CVA^n(0).$$

5.2.2 Backward sweep: Computation of CVA's sensitivities

Exactly as in 4.2, the backward sweep enables to compute CVA sensitivities with respect to model parameters and especially with respect to λ_0, ... , λ_M.

In what follows, we denote model parameters by θ and are interested in computing $\frac{\delta CVA}{\delta \theta_i}$. As always, the backward sweep consists in the adjoint of the steps involved in the forward sweep. This methodology fully relies on Mr. Capriotti's paper [11, page 43].

1. First, the initialization step [11, ($\overline{X}5$) page 43] states $\overline{CVA} = 1$ and $\overline{\theta_i} = 0$, and finally sets:

$$\overline{CVA^n} = \frac{\overline{CVA}}{N_{MC}}.$$

2. Then for $m = M, ..., 1$, we proceed backward and get:

$$\overline{V_m^n} = \overline{CVA}^n \mathrm{e}^{-rT_m}[L_{GD}(Q^n(\tau > T_m - 1) - Q^n(\tau > T_m))\mathbb{1}_{V_m^n > 0}],$$

$$\overline{N_m^n} = \overline{CVA}^n \mathrm{e}^{-2rT_m}[L_{GD}(Q^n(\tau > T_{m-1}) - Q^n(\tau > T_m))(V_m^n)^+],$$

$$\overline{Q^n(\tau > T_m)} = \overline{CVA}^n[V_m^n \mathrm{e}^{-rT_m}(1 - \delta_{m,0}) - V_{m+1}^n \mathrm{e}^{-rT_{m+1}}]\mathbb{1}_{V_m^n > 0},$$

with $\delta_{m,0}$ the Kronecker symbol, $N_m^n = \mathrm{e}^{-rT_m}$ the nominal at time T_m, and using the convention $V_{M+1}^n = 0$ [11, ($\overline{X4}$) page 43].

3. Next, we compute adjoints of the hold and exercise values for $m = M, ..., 1$ [11, ($\overline{X3}$) page 43]:

   If T_m is an exercise date we get:

$$\overline{H_m^n} = \overline{V_m^n} \mathbb{1}_{H_m^n > E_m^n},$$

$$\overline{E_m^n} = \overline{V_m^n} \mathbb{1}_{H_m^n < E_m^n}$$

   Otherwise, we set $\overline{H_m^n} = \overline{V_m^n}$ and $\overline{E_m^n} = 0$.

4. Then, we initialize the adjoint of the underlying by:

$$\overline{S_m^n} = \overline{E_m^n} \frac{\delta E_m^n}{\delta S_m^n}.$$

We also initialize the adjoints of the regression coefficients β and of basis functions ψ_m^n as in 4.2.4 and 4.2.4, and we get the following update of the underlying adjoint:

$$\overline{S_m^n} + = (\bar{\psi_m^n})^T \frac{\psi_m^n}{S_m^n}.$$

5. We can therefore update the adjoint of the discretized default intensities as follows [11, ($\overline{X2}$) page 43]:

$$\overline{\lambda_m^n} + = -(T_{m+1} - T_m) \sum_{j=m+1}^{M} \overline{Q^n(\tau > T_m)} Q^n(\tau > T_m).$$

6. Finally, we again update the adjoint of the discretized default intensities and get the adjoint of the model parameters.

$$\overline{\lambda_m^n} + = \overline{\lambda_{m+1}^n} \frac{\delta G}{\delta \lambda_m^n}(T_m, \lambda_m^n, \theta),$$

$$\bar{\theta} + = \overline{\lambda_{m+1}^n} \frac{\delta G}{\delta \theta}(T_m, \lambda_m^n, \theta).$$

with G the function generating the paths of the default intensity process [11, ($\overline{X1}$) page 43]. In our case, λ follows a CIR process which enables the simplifications below:

$$\frac{\delta G}{\delta \lambda_m^n}(T_m, \lambda_m^n, \theta) = 1.0 - kh + \frac{\mu Z_m^n \sqrt{h}}{2\sqrt{\lambda_m^n}}. \tag{5.4}$$

5.2.3 Numerical results

We present our numerical results on both Bermudan put and Bermudan best of two put. For clarity and in order to have a simple benchmark, we again will display results for a Bermudan best of two put where the two underlyings are simulated using the same random numbers starting from $S_0^1 = S_0^2$. Hence, we get an option equivalent to a simple Bermudan put built on one stock.

It is worth noting that while we still use 13 basis functions for our regression on Best of two Bermudan put as in 4.2, due to the more complex structure of the CVA, we need 7 Hermite polynomials in our regression basis to have accurate results on Bermudan put CVA.

In our simulations, the expiry date is 3 years ($T = 3.0$) and we can exercise the option every 3 months ($M = 12$). We consider also that $S_0 = 1.0$, $\sigma = 0.2$, $r = 0.15$, $K = 0.9$. We use as benchmarks an extended binomial tree [14], a default bucketing computation for the CVA and finite differences algorithms for the CVA sensitivities. Finally, we use the following intensity model parameters 5.2 for the time of default: $\lambda_0 = 1.0$, $k = 0.5$, $\mu = 1.0$, $\nu = 0.25$ and a $L_{GD} = 0.6$.

The table below displays the results obtained:

	Price	CVA	$\frac{\delta CVA}{\delta \lambda_0}$	Computation Time
AD method on Bermudan Put	0.0157905	0.00252039	-0.00256843	249 seconds
AD method on Best of two Put	0.00157121	0.00254705	-0.00266047	328 seconds
Benchmarks	0.0157926	0.00253147	-0.00259147	$\geq$ 45 minutes

Table 21: AD method on Bermudan Put CVA with 500.000 Monte-Carlo simulations

The time of computation is again quite impressive as one obtains the price, CVA and CVA greeks in a time equivalent to 3 times the time needed to only price the option (and this remains true no matter the number of greeks !). More than 45 minutes are needed to perform finite difference with this number of Monte-Carlo simulation while the computation time involved using AD is only few seconds ! Again, as we increase the number of sensitivities, the computation time increases dramatically in the case of finite difference while it remains the same with the AD method.

6 Further research on CVA

Our approach regarding CVAs was mainly based on strong assumptions which led to interesting results but often not in line with the "reality" observed on financial markets. It is worth-noting that we worked under the following simplifications:

- Independence between the default's intensity, the interest rate and the option's underlying.
- Unilateral counterparty risk: we only worked in a framework where one counterparty was default-free i.e without the notion of DVA for example.

These hypothesis enabled to obtain good results with the AD methods both on equities with positive payoff and Bermudan options. In the case of options with positive payoff we also obtained a closed form (assuming a CIR dynamic for the intensity of default).

Nevertheless, on financial markets one should introduce Wrong Way Risk (WWR) and collateralization to have a more realistic framework. Therefore, we decided to perform further research on their influence especially regarding the closed form obtained in 5.1.7 when changing filtration on options with positive payoff.

First, introducing correlation between the intensity of default λ and the equity payoff makes computations far more complicated and the existence of a closed-from is not guaranteed. Furthermore, if the intensity of default does not follow a CIR or any other chi-squared dynamic, the closed-form is not applicable anymore. Therefore, even if theoretically the result is extremely interesting, under a more realistic framework, things become more complicated.

One should however notice that under the framework created in 5.1.7, we have a closed-form not only for CVA but also for $\frac{\delta CVA}{\delta \lambda_0}$. Hence, we can access to second order sensitivities, such as the very important quantity $\frac{\delta CVA}{\delta \lambda_0 \delta S_0}$. Therefore, the application of the AD method on this second closed-form is again extremely powerful and leads to interesting theoretical calculations.

Lastly, WWR should be introduced and constitutes a very interesting axis to go beyond the work performed on CVAs in this thesis. Most of the computations performed on Bermudan options' CVA will remain the same but the simulation of default's intensity and underlying assets will involve correlation and hence, modeling and programming difficulties.

7 C++ code architecture

The purpose of this thesis was also to create an application where the user would be able to choose all the parameters of the option of interest i.e to choose the option payoff, the dynamics of the underlyings involved. Therefore, the user is able to fully design the option and then get its price, Greeks, CVA and CVA sensitivities (computed using the AD method).

The goal was to create a code easy to customize by adding more and more modules over time (ie more underlying dynamics or options' payoff). Indeed, the main challenge with the AD method was the coding part: this method was memory consuming and we often needed some manual treatments on the payoff. Therefore, a strong emphasis was placed on the coding part during this thesis and thus, will be detailed in this section.

7.1 Simulation of the behaviour of the options underlyings

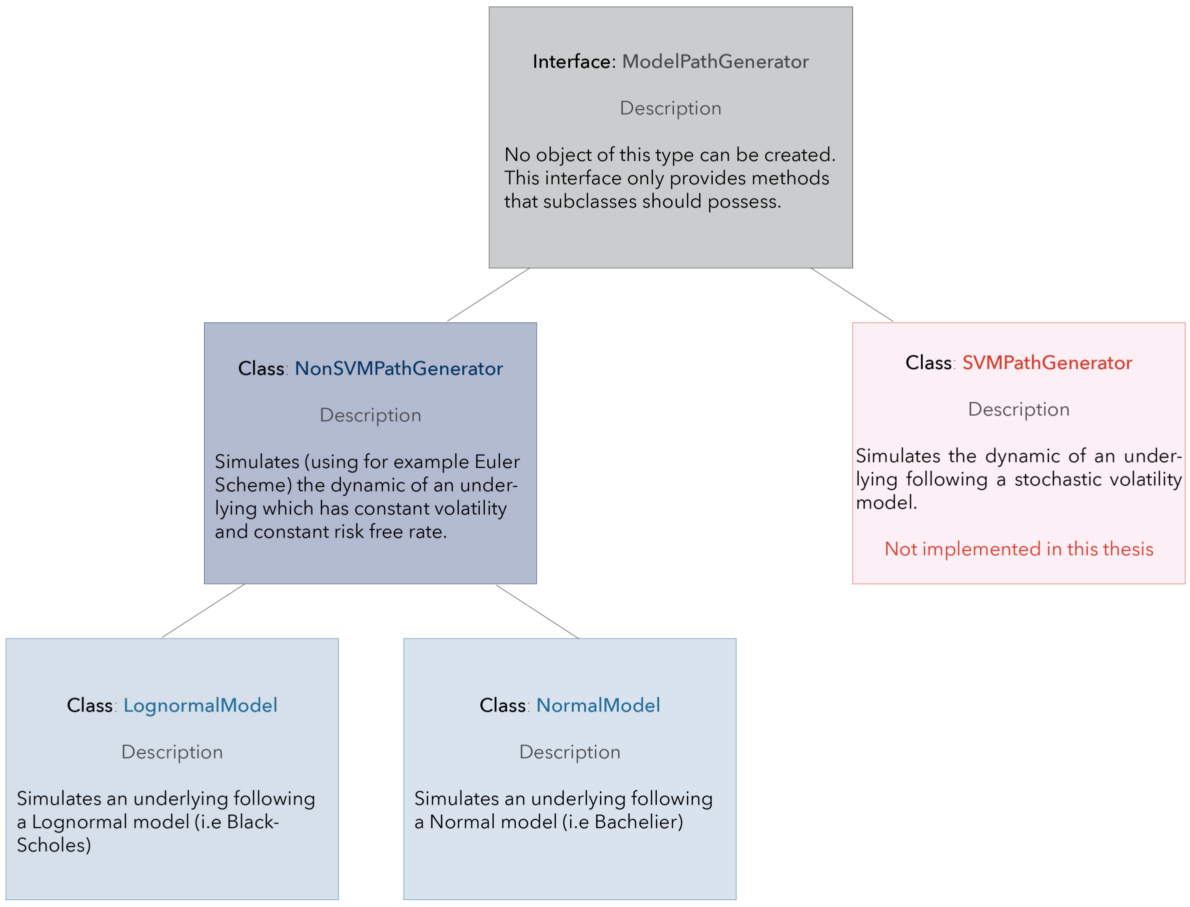
The application should enable the user to choose the dynamic of the underlying(s) of the option of interest in order to be able to apply the AD method on options built on any kind of underlyings. As it was impossible to code all the models an underlying could follow, the code needed to be easy to customize.

Therefore, we created an interface called *ModelPathGenerator* meant to be a template that gives all the functions that a class (used to simulate a special type of underlyings model) needs to have. Then, we created two abstract classes which implement this interface: *SVMPathGenerator* and *NonSVMPathGenerator*.

SVMPathGenerator is meant to simulate the dynamic of an underlying following a stochastic volatility model (such as Heston, SABR or CEV) but these models are not treated in this thesis so no implementations have been done. However, the interface has been created in order to provide a recipe for future implementations of such models.

NonSVMPathGenerator is meant to simulate (using for example Euler or Milstein Schemes) the dynamic of an underlying which has constant volatility σ and constant risk free rate r. We implemented two subclasses of it called *LognormalModel* and *NormalModel* which respectively simulate underlyings following lognormal and normal model. Both can whether simulate the entire path of the underlying using a Euler scheme or give directly the value of the underlying at maturity using the closed form for S_T as a function of S_0 (which is of course faster).

Figure 1: Interface to simulate the behaviour of an underlying


Interface: ModelPathGenerator
Description
No object of this type can be created. This interface only provides methods that subclasses should possess.
Class: NonSVMPathGenerator
Description
Simulates (using for example Euler Scheme) the dynamic of an underlying which has constant volatility and constant risk free rate.
Class: SVMPathGenerator
Description
Simulates the dynamic of an underlying following a stochastic volatility model.
Not implemented in this thesis
Class: LognormalModel
Description
Simulates an underlying following a Lognormal model (i.e Black-Scholes)
Class: NormalModel
Description
Simulates an underlying following a Normal model (i.e Bachelier)

7.2 Choice of the option's payoff

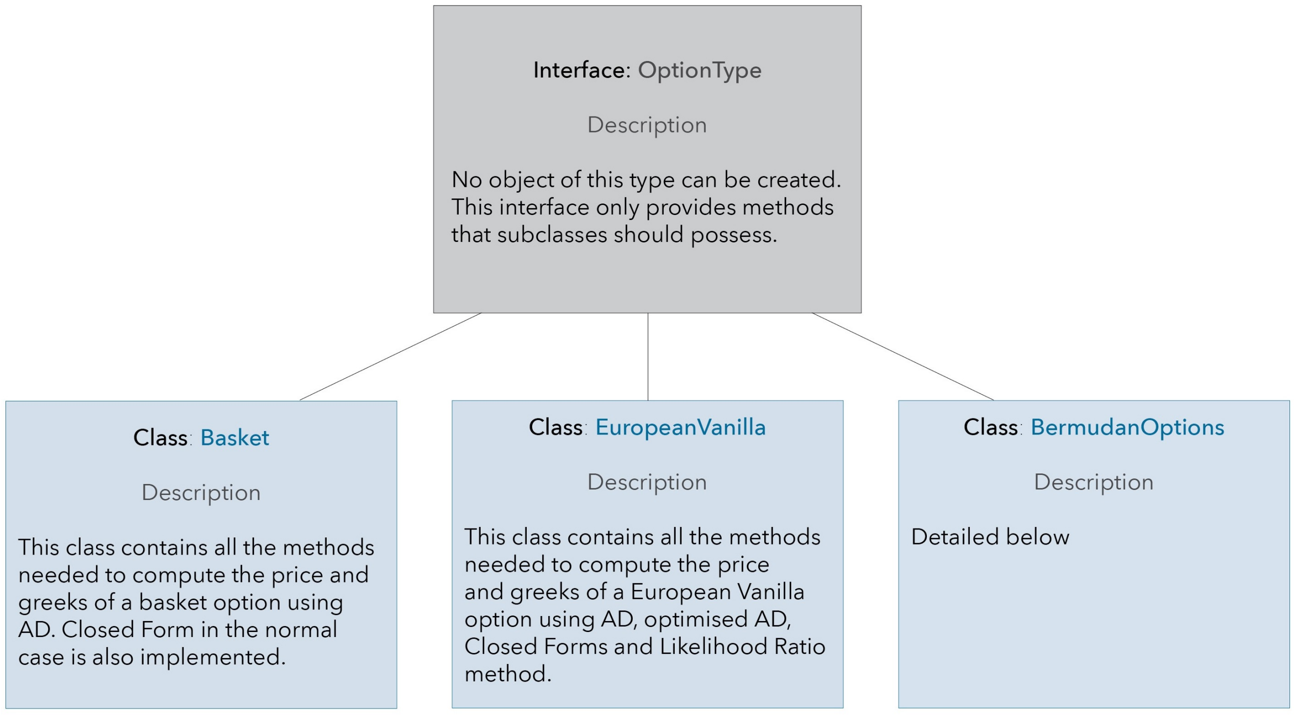
The application should enable to compute the AD method on any payoff. Again, we had to restrict ourselves to some option types as it is impossible to code "every" existing payoffs. However, the framework to easily add new payoffs exists and enables to customize very quickly the application.

Indeed, the interface *OptionType* is meant to be the contract that every class (used to compute AD method on a particular type of option) needs to fill. As part of this thesis, three types of options have been implemented: European and Bermudan vanilla options (in one and two dimensions) as well as Basket options. Hence, three subclasses inherit *OptionType*.

First, *EuropeanVanilla* provides the framework for European vanilla options. It is a class which contains lot of modules: first it possesses all functions required to compute the closed form for vanilla options (greeks and price). It also contains a module to compute every Greeks using the likelihood ratio method (basically a Monte Carlo on path-wise derivative). Finally, based on the book of Marc Henrard [19], it holds modules to compute Price and Greeks using the AD method and an optimized AD method.

Then, *Basket* is a class which enables to compute price and greeks of a basket option with stocks following *NormalModel* or *LognormalModel* dynamics (the user can create mixed basket). The AD method originally [12] gives the sensitivity with respect to S_T (final value of each stocks in the basket) so we implemented functions to have the derivative with respect to S_0 instead of S_T (delta). These functions differ according to which underlying model we are using.

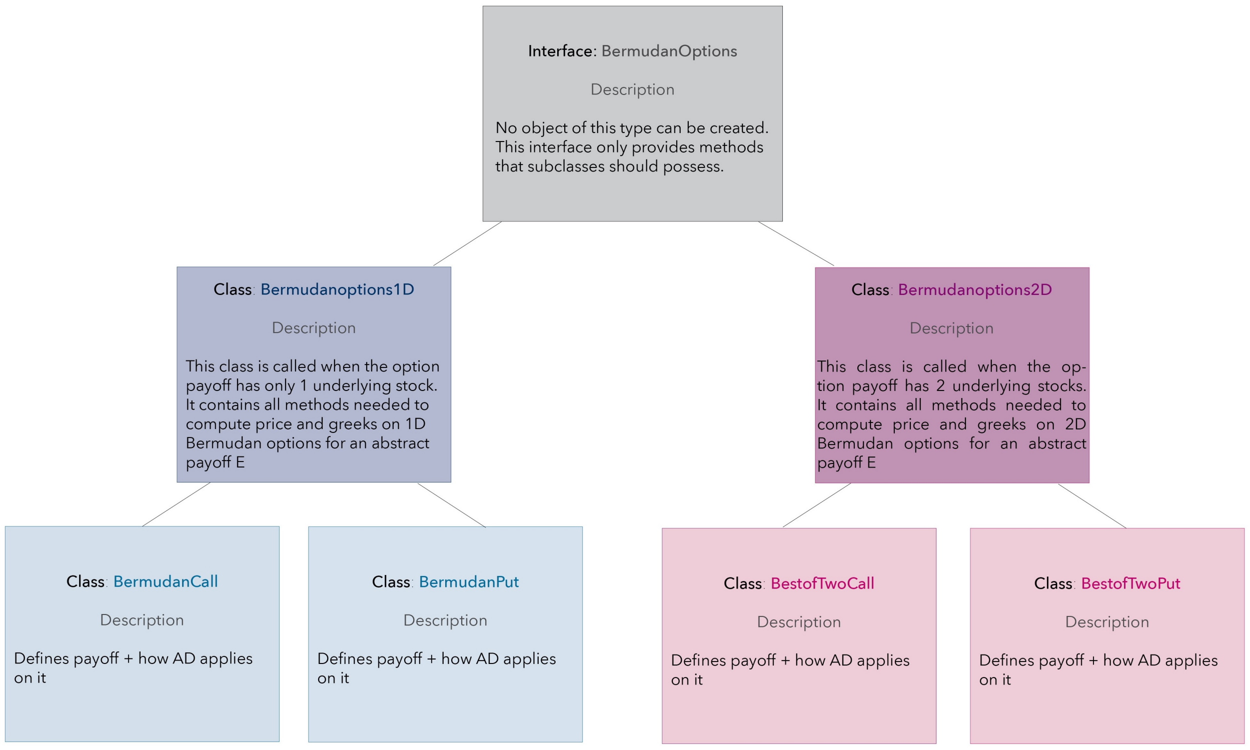
Figure 2: Interface to choose the option on which the AD is applied


Interface: OptionType
Description
No object of this type can be created. This interface only provides methods that subclasses should possess.
Class: Basket
Description
This class contains all the methods needed to compute the price and greeks of a basket option using AD. Closed Form in the normal case is also implemented.
Class: EuropeanVanilla
Description
This class contains all the methods needed to compute the price and greeks of a European Vanilla option using AD, optimised AD, Closed Forms and Likelihood Ratio method.
Class: BermudanOptions
Description
Detailed below

Finally, the interface *BermudanOptions* gives the framework (parameters and methods) that every type of Bermudan options share. This interface is implemented by two abstract classes *BermudanOptions1D* and *BermudanOptions2D* which provide all methods to compute the AD methods on any Bermudan Option (i.e on an abstract payoff E) built respectively on 1 or 2 underlyings.

The subclasses *BermudanCall*, *BermudanPut*, *BestOfTwoCall*, *BestOfTwoPut* then define the payoff of their associated option and how the AD method applies on this payoff. Therefore, it is very easy to add new Bermudan options to the application. Indeed, one just needs to redefine 2 functions: GetPayoff() (which returns the option payoff) and AD_on_payoff() (which computes the AD method on the associated payoff function by performing a "manual" decomposition of the latter).

Figure 3: Interface to apply AD on Bermudan Option


Interface: BermudanOptions
Description
No object of this type can be created. This interface only provides methods that subclasses should possess.
Class: Bermudanoptions1D
Description
This class is called when the option payoff has only 1 underlying stock. It contains all methods needed to compute price and greeks on 1D Bermudan options for an abstract payoff E
Class: Bermudanoptions2D
Description
This class is called when the option payoff has 2 underlying stocks. It contains all methods needed to compute price and greeks on 2D Bermudan options for an abstract payoff E
Class: BermudanCall
Description
Defines payoff + how AD applies on it
Class: BermudanPut
Description
Defines payoff + how AD applies on it
Class: BestofTwoCall
Description
Defines payoff + how AD applies on it
Class: BestofTwoPut
Description
Defines payoff + how AD applies on it

7.3 CVA computation

The architecture of the code regarding CVA computation is similar to the one used to price options. Again, everything has been thought so that the code is easily customizable (in order to add more features over time). The diagram explaining the design of the inherent code can be found in Appendix A (8) but we briefly describe here the role of every classes.

First, the interface *CVAcomputer* is the recipe which provides all the functions needed to apply the AD method on CVA built on any kind of payoff. Two classes implement this interface, respectively *OptionWithPositivePayoff* and *BermudanPayoff*. *OptionWithPositivePayoff* again defines the framework to apply AD method on non-Bermudan options with a positive payoff (such as European vanilla or Basket options) while *BermudanPayoff* does the same job but for Bermudan Payoff.

Then, *BasketCVA* and *EuropeanVanillaCVA* derive from *OptionWithPositivePayoff* and only specify the payoff to be considered while applying *OptionWithPositivePayoff* functions to perform AD.

Similarly, *BermudanPayoff* is implemented by two subclasses which specify the number of underlyings to be considered in our AD computations (namely *Bermudan1DPayoff* and *Bermudan2DPayoff*). Again, these two interfaces are implemented by classes specifying the exact payoff on which the AD is applied (i.e Bermudan Put, Call, Best of two Call, Best of two Put, etc ...).

7.4 Additional helpful modules

In this section, we detail quickly the additional modules used in the application. Some other modules were implemented to benchmark our results (such as Finite Differences algorithms or Binomial Trees) but are not part of the final application created and hence are not cited here.

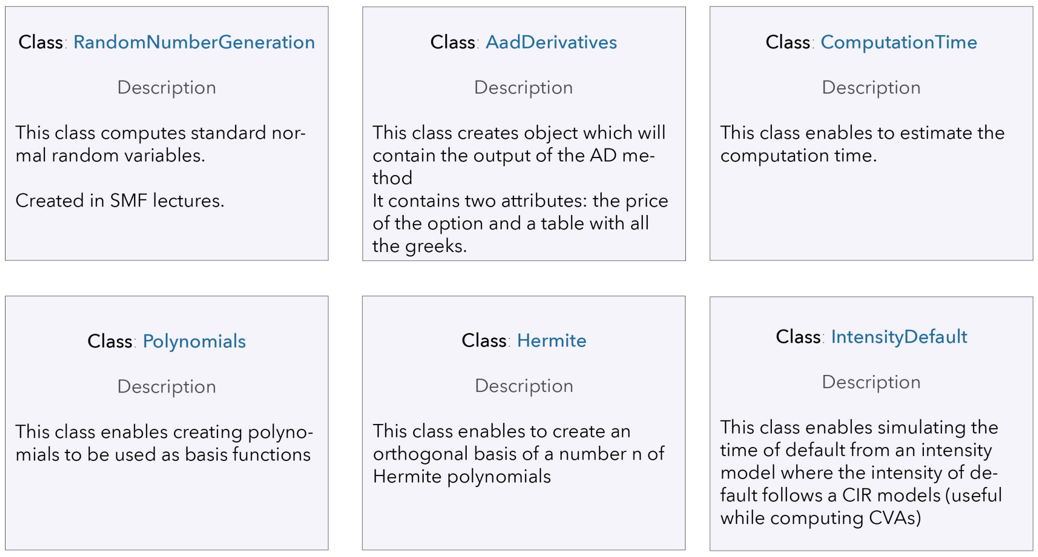
To have an efficient C++ code, it is necessary to stock in memory many quantities and therefore we used the open source C++ library Eigen [18] to keep track of relevant values in vectors and matrix. We also reused a library called *RandomNumbersGeneration* created for the SMF coursework [22]. This module enables to generate random variables (gaussian, exponential, etc...).

Finally, we created some helpful modules that provided objects needed in our AD computations.

The main classes are:

- **Polynomials**: This class enables creating polynomials to be used as basis functions for our regressions.
- **Hermite**: This class enables to create an orthogonal basis of a given number n of Hermite polynomials.
- **IntensityModel**: This class enables simulating the time of default from an intensity model where the intensity of default follows a CIR model (useful while computing CVAs).
- **ComputationTime**: This class enables to estimate the computation time.
- **AadDerivatives**: This class creates objects which contains the output of the AD method [19].

Figure 4: Additional Modules



Class: RandomNumberGeneration

Description

This class computes standard normal random variables.

Created in SMF lectures.

Class: AadDerivatives

Description

This class creates object which will contain the output of the AD method
It contains two attributes: the price of the option and a table with all the greeks.

Class: ComputationTime

Description

This class enables to estimate the computation time.

Class: Polynomials

Description

This class enables creating polynomials to be used as basis functions

Class: Hermite

Description

This class enables to create an orthogonal basis of a number n of Hermite polynomials

Class: IntensityDefault

Description

This class enables simulating the time of default from an intensity model where the intensity of default follows a CIR models (useful while computing CVAs)

Conclusion

The AD method gives a very powerful way to price options, obtain their greeks, quantify their CVAs and CVAs sensitivities. One can easily verify that this method gives accurate and very fast results. Indeed, the computation time needed to estimate **all** the sensitivities of a function (price or CVA) is only 3 times the duration required to evaluate this function.

While Monte-Carlo simulations and Finite Difference algorithms are often the only alternatives to the AD method on complex products (such as Bermudan options), these techniques are extremely time-consuming to obtain precise results. However, the AD method has the drawback to require a clever memory management as the decrease of the computation time necessitates to keep in memory lot of variables.

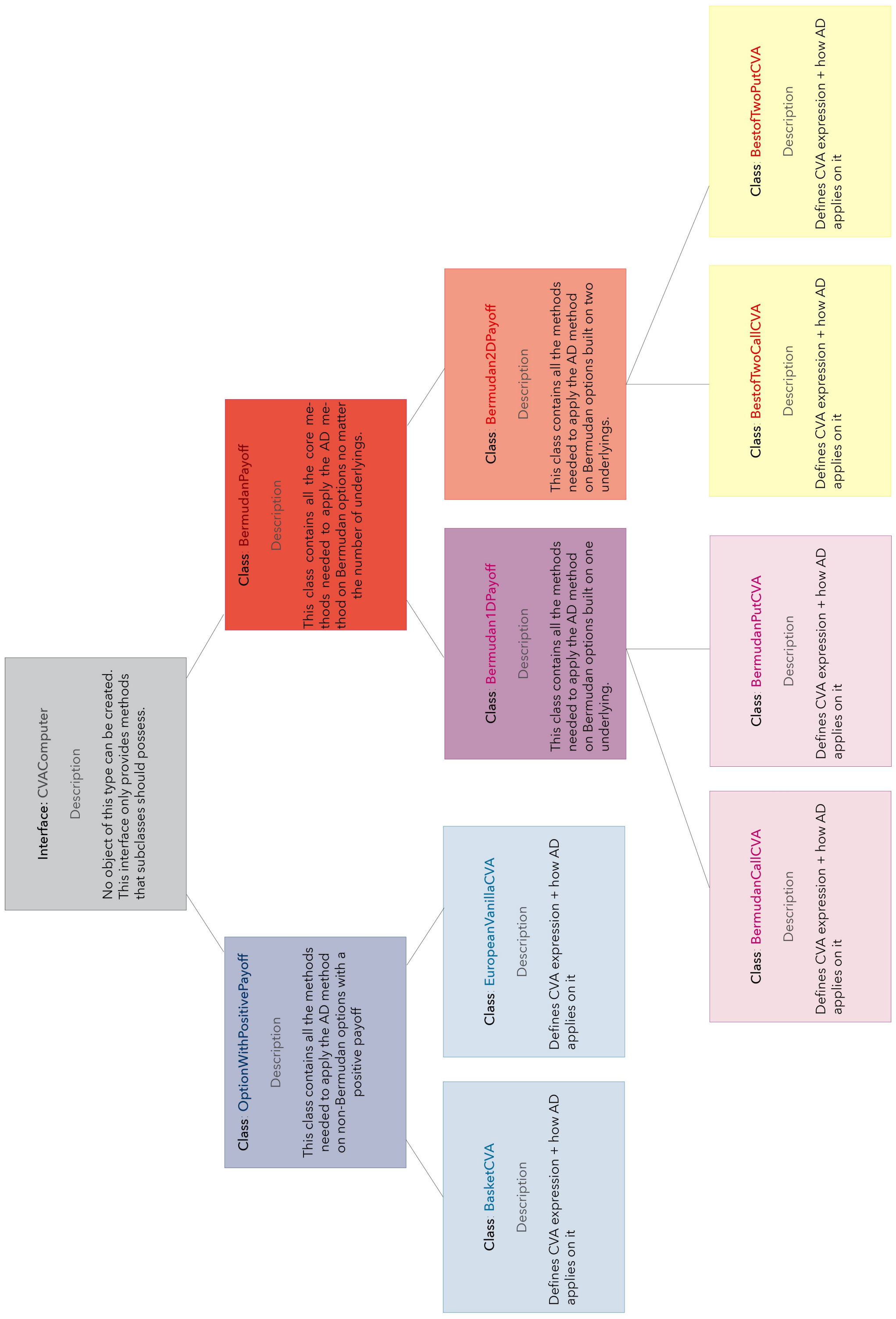
On simple instruments sush as Basket options, results are already impressive but the real power of the AD method can be observed on Bermudan options and CVA sensitivities computations. Indeed, these financial products imply to take into account the optimality of the exercise boundary and therefore Finite Difference algorithms are the most widespread frameworks to evaluate their sensitivities. Nevertheless, the complexity of such products often triggers unreasonable computation times.

In this context, the AD method seems to be one of the best solution to access sensitivities of complex financial instruments. In this thesis, very strong assumptions were made especially regarding CVA computations and therefore the results obtained here are not meant to overthrow the established order on financial markets. One should consider more complex underlying dynamics with SVM models for example, bilateral counterparty risk in CVA computations, Wrong Way Risk, collateralization, etc ... In a more realistic context, AD method is likely to trigger more modeling and programing problems.

However, no matter the simplifications made in this dissertation, the AD method appears to be one good research axis in a financial world where a precise and fast estimation of the sensitivities of financial instruments is becoming a priority.

8 Appendix A

Figure 5: Interface to apply AD on CVA


Interface: CVAComputer
Description
No object of this type can be created. This interface only provides methods that subclasses should possess.
Class: BermudanPayoff
Description
This class contains all the core methods needed to apply the AD method on Bermudan options no matter the number of underlyings.
Class: OptionWithPositivePayoff
Description
This class contains all the methods needed to apply the AD method on non-Bermudan options with a positive payoff
Class: Bermudan2DPayoff
Description
This class contains all the methods needed to apply the AD method on Bermudan options built on two underlyings.
Class: Bermudan1DPayoff
Description
This class contains all the methods needed to apply the AD method on Bermudan options built on one underlying.
Class: EuropeanVanillaCVA
Description
Defines CVA expression + how AD applies on it
Class: BasketCVA
Description
Defines CVA expression + how AD applies on it
Class: BestofTwoPutCVA
Description
Defines CVA expression + how AD applies on it
Class: BestofTwoCallCVA
Description
Defines CVA expression + how AD applies on it
Class: BermudanPutCVA
Description
Defines CVA expression + how AD applies on it
Class: BermudanCallCVA
Description
Defines CVA expression + how AD applies on it

References

[1] Brigo, B. (2018). Interest Rate Models with Credit Risk, Collateral, Funding Liquidity Risk and Multiple Curves, lecture notes, Interest Rate Models with Credit Risk, Collateral, Funding Liquidity Risk and Multiple Curves M5MF30, Imperial College London, Spring Term 2018.

[2] Brigo, D., Morini, M., and Tarenghi, M. (2011). Credit Calibration with Structural Models and Equity Return Swap valuation under Counterparty Risk. In: Bielecki, Brigo and Patras (Editors), *Credit Risk Frontiers: Subprime crisis, Pricing and Hedging, CVA, MBS, Ratings and Liquidity*, Wiley/Bloomberg Press, pp. 457–484. Preprint available at SSRN.com

[3] Brigo D., Morini, M., and Pallavicini, A. (2013). Counterparty Credit Risk, Collateral and Funding, with Pricing cases for all asset classes, Wiley and sons, Chichester.

[4] Brigo, D., Capponi, A., and Pallavicini, A. (2014). Arbitrage-free bilateral counterparty risk valuation under collateralization and application to Credit Default Swaps. Mathematical Finance, Vol. 24, No. 1, pages 125146.

[5] Brigo, D., Capponi, A., Pallavicini, A., and Papatheodorou, V. (2013). Pricing Counterparty Risk Including Collateralization, Netting Rules, Re-hypothecation and Wrong–Way Risk. International Journal of Theoretical and Applied Finance Vol. 16, No. 02

[6] D. Brigo, A. Pallavicini (2008). Counterparty Risk and Contingent CDS under correlation, Risk Magazine, February issue.

[7] Brigo, D., and Bakkar, I. (2009). Accurate counterparty risk valuation for energy-commodities swaps. Energy Risk, March 2009 issue.

[8] Brigo, D. and Chourdakis, K. (2009). Counterparty Risk for Credit Default Swaps:

Impact of spread volatility and default correlation. International Journal of Theoretical and Applied Finance, vol. 12 (07), pages 1007-1026.

[9] Brigo, D., Buescu, C., and Morini, M. (2012). Counterparty Risk Pricing: Impact of closeout and first-to-default time. International Journal of Theoretical and Applied Finance, Vol. 15, No. 06 (2012).

[10] Brigo, D., and Masetti, M., Risk Neutral Pricing of Counterparty Risk. Chapter 11 In: Pykhtin, M. (Editor), Counterparty Credit Risk Modelling: Risk Management, Pricing and Regulation (2006). Risk Books, London.

[11] Capriotti, L., Jiang, Y. and Macrina, A. (2017). AAD and least-square Monte Carlo: Fast Bermudan-style options and XVA Greeks. *Algorithmic Finance 6*: 3549, 2017.

[12] Capriotti, L. (2011). Fast Greeks by algorithmic differentiation. *The Journal of Computational Finance (3 - 35)*: Volume 14/Number 3, Spring 2011.

[13] Capriotti, L. and Giles, M. (2010). Fast correlation Greeks by adjoint algorithmic dierentiation. Risk, 23(3):7983. 1

[14] Collas, A., Groeneweg, K., Liang, X., Patel, C. and Thomas, M. (2018). Simulation Methods for Finance Project Report, Simulation Methods for Finance M5MF4 Imperial College London, Spring Term 2018.

[15] Glasserman, P. (2003). Monte Carlo Methods in Financial Engineering, Springer: New York 2003.

[16] Green, A. (2016). XVA : credit, funding and capital valuation adjustments, *The Wiley finance series*, 2016.

[17] Green, A. and Kenyon, C. (2015). MVA: Initial Margin Valuation Adjustment by Replication and Regression. Risk Magazine.

[18] Guennebaud, G., Jacob, B. and others (2010). Eigen v3, http://eigen.tuxfamily.org, 2010.

[19] Henrard, M. (2017). *Algorithmic Differentiation in Finance Explained.* Palgrave macmillan, 2017.

[20] Lal, B. and Ouoba, M. (2012). A seminar report in Analytical Finance II. Sweden: Malardalen University, 2012.

[21] Wang, Y. and Caflisch, R. (2010). Pricing and Hedging American-Style Options: A Simple Simulation-based Approach. *Journal of Computational Finance.* 2010.

[22] Zheng, H. (2018). Simulation Methods for Finance, lecture notes, Simulation Methods for Finance M5MF4 Imperial College London, Spring Term 2018.