# Computing Sensitivities of CVA Using Adjoint Algorithmic Differentiation

d-fine GmbH
Kellog College
University of Oxford

A thesis submitted in partial fulfillment of the requirements for the M.Sc. in
*Mathematical Finance*
September 27, 2015

## Abstract

In this thesis it is shown how one can use adjoint algorithmic differentiation in order to compute sensitivities of a CVA computation for a portfolio of single currency plain vanilla swaps, where the underlying model is chosen such that it takes into account the phenomenon of wrong way risk. Besides giving a basic introduction to CVA in general and describing the concrete model used for this work, this work focuses on the computational techniques needed to implement the adjoint sensitivity computation for first order interest, credit, model volatility and correlation sensitivities. The implementation presented in this work is based on the library dco-c++, developed by NAG / RWTH Aachen.

# Contents

**1 Introduction** **1**

1.1 Short Review of CVA . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Sensitivities for Monte Carlo based Algorithms and Algorithmic Differentiation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2.1 Pathwise Sensitivities . . . . . . . . . . . . . . . . . . . . . . . 3

1.2.2 Algorithmic Differentiation . . . . . . . . . . . . . . . . . . . . 5

1.3 Scope of this Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

**2 CVA Model for Interest Rate Products** **10**

2.1 Term Structures and Swap Pricing . . . . . . . . . . . . . . . . . . . 10

2.2 Interest Rate Model . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3 Credit Risk Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.4 A Concrete CVA Problem . . . . . . . . . . . . . . . . . . . . . . . . 16

2.5 Remarks on the choice of the model and its calibration . . . . . . . . 17

**3 Algorithmic Realization via Backward Induction** **19**

3.1 Numerical Simulation of SDEs . . . . . . . . . . . . . . . . . . . . . 19

3.2 Estimating Conditional Expectations via Regression . . . . . . . . . . 21

3.3 The Final Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.4 Backward Induction vs Forward Induction . . . . . . . . . . . . . . . 27

3.5 Adjoint Algorithmic Differentiation using dco-c++ . . . . . . . . . . 28

**4 Numerical Results** **32**

4.1 Single IR Swap . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

4.1.1 Monte Carlo Errors . . . . . . . . . . . . . . . . . . . . . . . . 38

4.1.2 Comparison with Forward Induction Algorithm . . . . . . . . 38

4.1.3 AD and Correlation Sensitivities . . . . . . . . . . . . . . . . . 43

4.1.4 Performance Measurements . . . . . . . . . . . . . . . . . . . 44

4.2 Portfolio of IR Swaps . . . . . . . . . . . . . . . . . . . . . . . . . . 45

**5 Summary and Conclusion** **48**

**Bibliography** **49**

# List of Figures

4.1 Plot of initial zero rates. X-axis: time in years, Y-axis: corresponding zero rate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.2 EE(t) for a 10Y Payer Swap. X-axis: time in years, Y-axis: exposure in units of currency . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
4.3 CVA Zero Sensitivities for 10Y Payer Swap. X-axis: time in years, Y-axis: corresponding zero rate sensitivity . . . . . . . . . . . . . . . 35
4.4 CVA Par Sensitivities 10Y Payer Swap. X-axis: time in years, Y-axis: corresponding par rate sensitivity . . . . . . . . . . . . . . . . . . . 35
4.5 Convergence of central difference approximation to AD sensitivities. X-Axis: $-\log_{10}\epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities. . . . . . . . . . . . . . . 36
4.6 Convergence of the AMC based backward induction algorithm. X-axis: number of paths, Y-axis: resulting CVA in units of currency . . . . . 39
4.7 Exposure profiles for backward and forward induction algorithms. X-axis: time in years, Y-axis: exposure in units of currency. . . . . . . . 41
4.8 Zero rate sensitivities for backward and forward induction algorithms. X-axis: time in years, Y-axis: CVA zero rate sensitivity. . . . . . . . 42
4.9 Par rate sensitivities for backward and forward induction algorithms. X-axis: time in years, Y-axis: CVA par rate sensitivity. . . . . . . . . 42
4.10 Convergence of central difference approximation to AD sensitivities for both algorithms. X-Axis: $-\log_{10}\epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities. . . . . . . . 43
4.11 Convergence of central difference approximation to AD sensitivities for both algorithms - $\bar{\rho} = 1$. X-Axis: $-\log_{10}\epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities. . . 44
4.12 Exposure profile for a portfolio of 10 payer swaps X-axis: time in years, Y-axis: exposure in units of currency. . . . . . . . . . . . . . . . . . . 46

4.13 Zero rate sensitivities for a portfolio of 10 payer swaps. X-axis: time in years, Y-axis: CVA zero rate sensitivity. . . . . . . . . . . . . . . . . 47
4.14 Par swap rate sensitivities for a portfolio of 10 payer swaps. X-axis: time in years, Y-axis: CVA par rate sensitivity. . . . . . . . . . . . . . 47

# Chapter 1

# Introduction

## 1.1 Short Review of CVA

Credit Valuation Adjustment - or CVA for short - can be thought of as the risk-neutral price of the counterparty risk of a portfolio of transactions and is dependent on the current and future exposure that an institution has with a specific counterparty. Following [4] we we will work throughout this thesis with the following definition:

**Definition 1** *Given a portfolio of transactions with a single counterparty (netting set), we assume that its risk neutral price process is given by $\Pi_t$. The counterparty may possibly default with random default time $\tau$. Assume that for a suitable payoff $V(t,T)$ we have $\Pi_t = \mathbb{E}[V(t,T)|\mathcal{F}_t]$, where $\mathcal{F}_t$ denotes the filtration given by the joint evolution of the default event and market factors determining $\Pi_t$. Given discount factors $D(0,t)$ and a constant recovery rate $0 \leq R \leq 1$ we define the* ***unilateral CVA*** *as of $t = 0$ as*

$$\begin{aligned} CVA &= (1-R)\mathbb{E}_0\left[\mathbb{1}_{\{0<\tau\leq T\}} D(0,\tau)\Pi_\tau^+\right] \qquad (1.1)\\ &= (1-R)\mathbb{E}_0\left[\mathbb{1}_{\{0<\tau\leq T\}} D(0,\tau)\mathbb{E}[V(\tau,T)|\mathcal{F}_\tau]^+\right], \end{aligned}$$

*where both the inner and outer expectations are taken under the risk neutral measure $\mathbb{Q}$ w.r.t the numeraire corresponding to the discount factors $D$. The expression $EE(t) = \mathbb{E}_0[\Pi_t^+]$ is usually referred to as the* ***Expected Exposure***. *A plot of $EE(t)$ versus $t$ is usually referred to as the* ***Exposure Profile***. *The expression $(1-R)$ is also called loss given default, or LGD for short.*

In order to better understand the above definition we try to rewrite it in different forms. As a first step, we have obviously have

$$\text{CVA} = (1-R)\int_0^T \mathbb{E}_0[D(0,t)\mathbb{E}[V(\tau,T)|\tau = t]^+]d\mathbb{Q}(\tau = t). \qquad (1.2)$$

Assuming independence between the default time $\tau$ and the market evolution determining $\Pi_t$[1], we get

$$\text{CVA} = (1-R)\int_0^T \mathbb{E}_0[D(0,t)\Pi_t]^+\phi(t)dt.$$

with the density $\phi(t)$ belonging to the probability of a default, i.e.

$$\mathbb{Q}(0<\tau<t) = \int_0^t \phi(s)ds.$$

The previous expression can then be further simplified to

$$\begin{aligned}\text{CVA} &= (1-R)\int_0^T \mathbb{E}_0[D(0,t)]\mathbb{E}_0[\Pi_t]^+\phi(t)dt \\ &= (1-R)\int_0^T P(0,t)\mathbb{E}_0[\Pi_t^+]\phi(t)dt, \qquad (1.3)\end{aligned}$$

where $P(0,t)$ are the zero bond prices for maturities $t$ oberserved at time 0. We can thus interpret the above expression in the following way: CVA can be thought of as the discounted expected exposure weighted by probability of default and multiplied by the loss given default.

To get a better feeling for what is to come, we follow again [4] and start to discretize Eq. (1.1). Of course, we could now directly discretize the integral in Eq. (1.3). However, we would like to show that a discretization can also be done at an earlier stage. Thus, we first introduce a timegrid $t = T_0, T_1, \ldots, T_N = T$, such that it forms a partition of $(0,T)$. Bringing this together with Eq. (1.1) we get

$$\text{CVA} = (1-R)\sum_{i=1}^N \mathbb{E}_0[\mathbb{1}_{T_{i-1}<\tau<T_i}D(0,\tau)\Pi_\tau^+].$$

This expression can now be approximated by assumming that a default may only occur on time $T_i$:

$$\text{CVA} \approx (1-R)\sum_{i=1}^N \mathbb{E}_0[\mathbb{1}_{T_{i-1}<\tau<T_i}D(0,T_i)\Pi_{T_i}^+].$$

Supposing again, that default and market evolution are independent we may simplify the previous expressions to

$$\begin{aligned}\text{CVA} &\approx (1-R)\sum_{i=1}^N \mathbb{Q}(t_{i-1}<\tau<t_i)\mathbb{E}_0[D(0,T_i)\Pi_{T_i}^+], \\ &= (1-R)\sum_{i=1}^N \mathbb{Q}(t_{i-1}<\tau<t_i)P(0,T_i)\mathbb{E}_0[\Pi_{T_i}^+], \qquad (1.4)\end{aligned}$$

which is nothing other than a discrete version of Eq. (1.3).

[1]This is in practive not an uncommon assumption.

## 1.2 Sensitivities for Monte Carlo based Algorithms and Algorithmic Differentiation

We give here a brief overview of some standard techniques used for computing sensitivities of Monte Carlo based algorithms for the valuation of financial derivatives. Unless stated otherwise, the material presented here is based on Chapter 7 in Glassermann's book [15].

### 1.2.1 Pathwise Sensitivities

An important task besides pricing a derivative or determining a valuation adjustment such as CVA is to determine sensitivities or "Greeks" of theses values. When this is done for a Monte Carlo based computation, this task thus consists of evaluating a derivative of the form

$$\partial_\theta V(t;\theta) = \partial_\theta \mathbb{E}[P(T;\theta)],$$

for some payoff function $P$. A first shot at this problem could make use of approximating the partial derivative using a finite difference quotient, that is by evaluating

$$\Delta_\theta V(t;\theta) := \frac{V(t;\theta+h) - V(t;\theta)}{h} = \partial_\theta V(t;\theta) + \mathcal{O}(h), \tag{1.5}$$

or the central difference quotient

$$\frac{V(t;\theta+h) - V(t;\theta-h)}{2h} = \partial_\theta V(t;\theta) + \mathcal{O}(h^2), \tag{1.6}$$

which obviously has the advantage of producing at least in theory an approximation error of $h^2$ instead of $h$. While this approach is easily applicable, once a realization of the function $V$ is at hand, it has at least three major drawbacks:

1. It is well known that numerical differentiation using ordinary finite differences is in generally not numerically stable.
2. The evaluation of one partial derivative of $V$ requires at least two full evaluations of $V$. This fact may cause the evaluation of multiple partial derivatives to become prohibitively expensive, especially when one evaluation of $V$ requires a Monte Carlo simulation.
3. Even if both evaluations of $V$ make use of the same set of random numbers, the variance of $\Delta_\theta V(t;\theta)$ may be in the order of $h^{-1}$. Hence, even if the numerical finite difference could be computed in a stable manner, the variance of the finite difference estimator could easily render results unreliable.

One way to improve this situation is to choose an approach called "pathwise sensitivities". One assumes here a sufifcient regularity of the payoff P, such that one may write

$$\partial_\theta \mathbb{E}[P(T;\theta)] = \mathbb{E}[\partial_\theta P(T;\theta)].$$

Let us make this more concrete and assume that we are given an asset $S_\theta(t)$ and payoff $P = P(S_\theta(t))$. In this form, $\theta$ could for example be the initial value of the asset $S_0$, while $P$ could be the payoff of a standard call option on $S$, i.e. $P(S) = (S-K)^+$.

The regularity conditions required for the interchange of the expectation and differentiation operators can then be summarized as follows:

1. For each $\theta$, $\partial_\theta S_\theta$ exists with probability one.
2. For each $\theta$, $S_\theta$ must lie in $P$'s domain of differentiability with probability one.
3. Lipschitz-continuity of the payoff $P$.
4. A similar condition of $S_\theta$, i.e. there has to exist a random variable $\kappa$, such that for all $\theta_1$, $\theta_2$ there holds

   $$|S_{\theta_2} - S_{\theta_1}| \leq \kappa |\theta_2 - \theta_1|,$$

   with $\mathbb{E}[\kappa] < \infty$.

A typical example where these conditions do not hold is the case of digital payoffs. The rest of this section is now devoted to the efficient implementation of the pathwise sensitivities approach. Our standard reference for this is the famous paper [14] by Giles and Glassermann.

In the situation, where $\partial_\theta P(T;\theta)$ is available in explicit form, one thus obtains an estimator for the sensitivity with respect to $\theta$ by directly evaluating $\partial_\theta P(T;\theta)$ along each path and then taking the usual average.

In most applications, $S(t)$ is defined as a diffusion proccess

$$\mathrm{d}S = \mu(S,t)\,\mathrm{d}t + \sigma(S,t)\,\mathrm{d}W.$$

Given the fact that this equation might not be explicitly solvable, one often has to apply a discretization scheme on the previous equation, which gives a discrete version of the process via a relationship of the form

$$S(t_{n+1}) = F(S(t_n)), \tag{1.7}$$

with a suitable function $F$ and timegrid $0 = t_0, t_1, \ldots, t_n = T$. Differentiating both sides gives

$$\partial_\theta S(t_{n+1}) = \partial_S F(S(t_n))\partial_\theta S(t_n), \tag{1.8}$$

which then gives

$$\begin{aligned} \partial_\theta[P(S_\theta(T))] &= \partial_S P(S_\theta(T))\partial_\theta S(T) \\ &= \partial_S P(S_\theta(t_n))\partial_S F(S(t_n))\partial_S F(S(t_{n-1}))\ldots\partial_S F(S(t_0)). \end{aligned}$$

Hence, in order to evaluate $\partial_\theta[P(S_\theta(T))]$ we would make use of the iteration (1.7) to obtain the option value itself and at the same time simulate the "tangent process" (1.8), the result of which is then multiplied by the derivative of the payoff $\partial_S P(S_\theta(t_n))$.

Clearly, there are two ways of algorithmically calculating $\partial_\theta[P(S_\theta(T))]$:

1. Forward mode: In each step evaluate $S(t_{n+1})$ and $\partial_\theta S(t_{n+1})$, where the latter is obtained by evaluating $\partial_S F(S(t_n))$ and multiplying it by $\partial_\theta S(t_n)$ obtained in the previous step. At the final step, multiply $\partial_\theta S(t_n)$ by $\partial_S P(S_\theta(t_n))$.

2. Backward (or adjoint) mode: In each step evaluate $S(t_{n+1})$, $\partial_S F(S(t_{n+1}))$. Save all $\partial_S F(S(t_i))$ in memory and do not perform the multiplication in each individual step. Instead, at the final step evaluate $\partial_S P(S_\theta(t_n))$ and then multiply it successively by $\partial_S F(S(t_n))$, $\partial_S F(S(t_{n-1}))$,...

One might wonder, as to why one would choose either the forward or backward mode. This is, in fact, a question of efficiency. Suppose, that we want to compute sensitivites of an option that is dependent on $m > 1$ assets, i.e. that $S = (S^{(1)}, \ldots, S^{(m)})$ and that we are computing the Delta of this option with respect to $S$, so that $\theta = (S_0^{(1)}, \ldots, S_0^{(m)})$. In this setting, each of the expressions $\partial_S F(S(t_i))$ turns into an $m$ by $m$ Jacobian matrix. This means that in the forward mode we would have to compute in each step a matrix multiplication at the cost of $\mathcal{O}(m^3)$.

In the backward mode, however, since $\partial_S P(S_\theta(t_n))$ is a vector of size $m$, we would only be required to perform one vector-matrix multiplication, which we can do at the cost of $\mathcal{O}(m^2)$. Hence, if we use the backward mode for this situation, we can evaluate the derivative $\partial_\theta[P(S_\theta(T))]$ one order of magnitude cheaper than in the forward mode.

### 1.2.2 Algorithmic Differentiation

Often it can become quite tedious in practice to manually implement either the forward or backward mode for pathwise sensitivities. This is where a technique called *algorithmic* or sometimes *automatic differentiation* (AD) comes into play.

Note: This subsection is partly based on the material already presented by the author in [26]

The idea of automatic differentiation (see [17, 23] for an introduction) is to take a computer program which represents a function $F(x)$ and then "differentiate" the program, i.e. the algorithmic realization of $F(x)$. To this end, the function $F(x)$ is first split up into into its most basic parts, i.e. addition, multiplication, etc. Each atomic operation is then extended to be performed not only on the input values $x$ but also on the corresponding tangent space.

There are two major computational techniques used:

- Source-Code Transformation: Before the source code for $F$ is compiled, a special tool is applied to the code, which then generates source code that evaluates $\partial_x F(x)$.

- Operator Overloading: This technique is applicable for programming languages which support overloading of operators and functions. Here, any arithmetic operator (function resp.) is overloaded such that any operation is performed on the function and the derivative as well, e.g.

  $$+ : (a, b) \to (a + b)$$

  is automatically replaced by

  $$+ : ((a, a'), (b, b')) \to (a + b, a' + b').$$

  Similary, the multiplication operation

  $$\times : (a, b) \to (ab),$$

  would automatically be replaced by

  $$\times : ((a, a'), (b, b')) \to (ab, ab' + a'b).$$

The two major advantages of using AD can be summarized in the following points:

1. Accuracy: By construction, AD always gives exact derivatives of the algorithmic realization.

2. Efficiency: Similar to the case described in the previous subsection, the evaluation of the derivative can be performed in the so called *reverse* or *adjoint* mode. In this mode, all operations during the evaluation are first recorded in memory.

After the evaluation of $F$ is finished, the gradient is then reconstructed by backward execution of the operations saved in memory during the first sweep. It can be shown (see i.e. [17]) that the cost of the derivative evaluation for a function $F : \mathbb{R}^n \to \mathbb{R}$ is then given by a constant factor multiplied by the cost of one function evaluation of $F$. Moreover, the constant is independent of the number of partial derivatives. This behavior gives a very distinct advantage over using, for instance, finite differences where one would need at least two function calls for the evaluation of one single partial derivative, thus leading to a complexity growing linearly with the number of partial derivatives to be computed.

Note that the situation reverses for a function $F : \mathbb{R}^n \to \mathbb{R}^m$ with $m > n$. In this situation, the forward mode of AD will outperform the backward mode.

## 1.3 Scope of this Work

In this thesis we will use AD in order to compute

$$\partial_\theta \mathrm{CVA} = (1-R)\partial_\theta \mathbb{E}_0 \left[\mathbb{1}_{\{0<\tau\leq T\}} D(0,\tau)\Pi_\tau^+\right],$$

where $\Pi_t$ is given by the risk neutral price process of a portfolio of interest rate swaps. This problem has already been adressed in a series of papers by Capriotti et al. in [7, 8] and the article [27] by Savickas et al. Their work is mainly based on the following two assumptions:

1. They assume that their CVA is given by an expression of the form

$$\mathrm{CVA} = (1-R)\sum_{i=1}^{N} \mathbb{Q}(t_{i-1} < \tau < t_i) P(0,T_i) \mathbb{E}_0[\Pi_{T_i}^+],$$

so that

$$\mathrm{CVA} = (1-R)\mathbb{E}_0 \left[\sum_{i=1}^{N} \mathbb{Q}(t_{i-1} < \tau < t_i) P(0,T_i) \Pi_{T_i}^+\right], \tag{1.9}$$

which means nothing other than interpreting CVA as an option with (path-dependent) payoff $P = \sum_{i=1}^{N} \mathbb{Q}(t_{i-1} < \tau < t_i) P(0,T_i)\Pi_{T_i}^+$.

2. They go on by modelling an evolution of the market factors $S_i(t)$ determining $\Pi_t$ (interest rates, asset prices, ...) and assume that $\Pi_t$ is given as an explicit, nonlinear function $F$ depending on the market factors $S_i$, i.e. they suppose that $\Pi_t = F(S_1(t), \dots, S_n(t))$. Here, one could think of $F$ as a simple swap pricing or Black-Scholes-type formula.

Unsurprisingly, they go on and and apply the pathwise method outlined in the previous section: while Savickas [27] manually implements the pathwise approach in forward mode for an implementation on GPUs, Capriotti [7, 8] chooses an adjoint (i.e. backward) implementation supported by AD tools. Moreover, it is easy to imagine that all authors report a successful application of the pathwise/AD approach to CVA sensitivities and emphasize the performance gains compared to the ordinary "bumping" (i.e. finite difference) technique.

In this thesis we will, however, approach the problem of CVA sensitivities under different assumptions: First of all we assume a stochastic model of default probabilities which is correlated with the stochastic evolution of market factors. Secondly, while the asumption that $\Pi_t = F(S_1(t), \ldots, S_n(t))$ is not an unreasonable one, especially when the portfolio in question is made up of plain vanilla products, the situation changes when dealing with a portfolio of more complex products, where prices cannot be obtained using explicit formulas anymore. Often, this is already the case when the products in question exhibit an early exercise feature or involve more than one risk factor. For a portfolio consisting of a single such product, all we know is that $\Pi_t = \mathbb{E}[V(t,T)|\mathcal{F}_t]$ is defined by a conditional expectation, which may have to be computed by another (Sub-)Monte-Carlo simulation or a different similarly expensive procedure (PDEs,...). Implemented naively this could render the overall CVA computation prohibitively expensive, as one would for instance be forced to run a whole Monte Carlo compuation for all time steps along each individual path.

This is where the American Monte Carlo (AMC) approach to CVA comes in, which seems to have been independently proposed by Brigo and Palliavicini in [5], Cesari et al. in [10], as well Schöftner in [28]. Under the AMC approach one first simulates all market factors until maturity $T$ and then determines conditional expectations giving $\Pi_t$ backwards in time by approximating them for each time step using a Longstaff-Schwartz-type regression [22]. This means that the simultaneous evaluation of $\Pi_t$ along all paths for a fixed $t$ is about as expensive as one Longstaff-Schwartz-type Regression. This can lead to a significant speedup in CVA computations and has thus become a popular choice among practitioners in the field.

Of course, a CVA implementation based on AMC drastically changes the way of computing its sensitivities. Although - given the results for pathwise sensitivities of American options outlined in [21] - one might suspect differently, we will not be able anymore to apply the ordinary pathwise method and will have to apply AD on the implementation as a whole. This will be the main topic of Chapter 3. There, we will present the basic underlying AMC algorithm and then show how it can be equipped with AD sensitivities. Before we discuss the actual algorithm and its differentiation, we will first present a concrete model problem for the compuation of CVA for a portfolio of interest rate swaps which includes a possibility for wrong way risk. This is the subject of Chapter 2. After having discussed concrete numerical examples in Chapter 4, we move on to the conclusion of this work in Chapter 5.

Currently there seems to be no literature regarding the combination of AD and regression-based CVA, yet there is evidence (see [18]) that the two are already being combined in practice. This thesis hopes to make this gap between theory and practice a bit smaller.

Finally, we would like to mention that in this thesis, the concrete implementation of AD sensitivities is performed using the software package dco-c++, which is part of the NAG libraries and developed by Naumann et al. at RWTH Aachen [24].

# Chapter 2

# CVA Model for Interest Rate Products

The eventual goal of this thesis is to compute sensitivities for the CVA of a set of interest rate swaps. As we have seen, we thus have to specify a model for the evolution of the interest rate term structure, as well as a credit risk model from which we can obtain default probabilities. Moreover, we will incorporate the phenomenon of Wrong Way Risk into our model. This means that we will have to specify how our interest rate and credit risk model are correlated. Our basic setup will be that of the papers [5] and [6], condensed versions of which have been added as chapters into the monograph [4]. Because our emphasis is on sensitivities, we will however, simplify the setup of Brigo et. al. at certain points. Now, before we move on to the actual CVA modelling, we first review the elementary financial instrument that we will be the basis for our CVA computations

## 2.1 Term Structures and Swap Pricing

We begin this section by fixing some notation. With

$$P(t,T), \qquad P(T,T) = 1,$$

we denote the (random) price of a zero-coupon bond or discount-factor. When used for discounting purposes we will often use the symbol $D$ instead of $P$. Moreover, in the money market (or bank account) numeraire we can express the risk neutral price of a zero coupon bond as

$$P(t,T) = \mathbb{E}_t \left[ \exp \left( - \int_t^T r(s) ds \right) \right].$$

Here, as usual, the random variable $r(s)$ denotes the so called *short rate*, corresponding to the bank account numeraire

$$N_t = \exp\left(\int_0^t r(s)ds\right),$$

and the expectation is taken under the measure coming from this choice of numeraire. Now, if we wanted to calculate the present value (i.e. $t = 0$) of a cashflow $C$, paid to us at time $T$, then this would be equal to

$$P(0,T)\ C,$$

assuming that we are already given an initial term structure $P(0,T)$, which is usually called the *zero curve*. The initial term structure $P(0,T)$ is in general constructed in the following way:

- Observe prices (i.e. interest rates) quoted by market participants for different (mostly simple) financial instruments.
- Make an assumption about how most market participants determine their quoted prices, that is choose a number of models.
- Use these models to compute the initial term structure $P(0,T)$ from the observed prices.

For our purposes we will be content with constructing this term structure from observed swap rates, only. As we will also compute CVA for these very instruments it seems apt to briefly review the details of how to price a simple interest rate swap.

An interest rate swap is a contract between two parties where a fixed or floating interest rate payment on one side is exchanged for a floating rate payment on the other side. A swap trade is entered at par, this means that at inception, the value of the trade is zero for either side. The rates which are quoted for these par swaps are used for the construction of $P(0,T)$ where $T$ is typically greater or equal than one year (the so called *far end* of the curve).

Assume now a swap with start date $T_0$ (used for the first interest rate period) and payment dates $T_1$,...,$T_N$, which at these dates exchanges a fixed payment with interest rate $r$ for a floating payment $L(T_{i-1}, Ti)$ (the letter L chosen in order to resemble LIBOR). We will call $L(T_{i-1}, T_i)$ the forward rate. The day count fraction for each period $[T_{i-1}, T_i]$ is denoted by $\delta_n$. Then, assuming a notional of $X$, the value of this swap is given by

$$\begin{aligned} V_{\text{Swap}} &= PV_{\text{fixed}} - PV_{\text{float}} \\ &= r \sum_{k=1}^{N} \delta_n P(0, T_k) X - \sum_{k=1}^{N} \delta_n P(0, T_k) L(T_{k-1}, T_k) X \end{aligned} \tag{2.1}$$

Thus, knowing the current term structure $P(0,t)$ we already have enough information at hand to price the majority of plain vanilla interest rate swaps. We finish this section by showing how we may use this knowledge to construct the initial ter mstructure from observed quotes for interest rate swaps.

Because all swap quotes refer to par swaps, that is $V_{\text{Swap}} = 0$, and there is typically no exchange of notional, we obtain the following equation:

$$r \sum_{k=1}^{N} \delta_n P(0, T_k) = \sum_{k=1}^{N} \delta_n P(0, T_k) L(T_{k-1}, T_k). \tag{2.2}$$

In pre-crisis text books (see for instance [11]) it was usually assumed that all interest rates, including the forward rates $L(T_{i-1}, T_i)$ could be obtained from one curve $P(0,t)$, which was also used for the discounting of the cashflows. Hence, in this situation,

$$L(T_i, T_{i-1}) = \frac{1}{\delta_n} \left( \frac{P(0, T_{i-1})}{P(0, T_i)} - 1 \right).$$

Plugging this into (2.2) we obtain

$$r \sum_{k=1}^{N} \delta_n P(0, T_k) = P(0, T_0) - P(0, T_N), \tag{2.3}$$

because all the terms on the right hand side of (2.2) turn into differences $P(0, T_{i-1}) - P(0, T_i)$. If we assume now that we already know $P(0, T_0)$ Eq. (2.2) is then the basis for a general procedure of obtaining the values $P(0, T_k)$, which is usually termed *bootstrapping a zero curve.* In the above setting, suppose that one is given a discount factor $P(0, T_0)$, for instance by considering money market rates, zero-coupon bond prices, or similar vanilla instruments. Furthermore, assume that one can observe $N$ par swap rates $r_i$ for the $N$ maturities $T_N$, then there hold the following $N$ equations

$$r_M \sum_{k=1}^{M} \delta_n P(0, T_k) = P(0, T_0) - P(0, T_M), \qquad M = 1, \ldots, N.$$

As we are already given $P(0, T_0)$ and because the $i$-th equation depends only on the variables $P(0, T_0)$, ..., $P(0, T_i)$, we can therefore recursively compute the discount

factors $P(0,T_i)$.
Note: This subsection is partly based on the material already presented by the author in [25].

## 2.2 Interest Rate Model

From the previous section we now know how to price an interest rate as of today. For a CVA computation, it is however necessary to estimate the future distribution of swap prices. Hence, we have to specify a stochastic evolution of the term structure $P(0,t)$. For our purposes, we will again follow Brigo [4–6] and choose the G2++ model.

The G2++ model [3] belongs to the class of two factor additive gaussian short rate models. It is an equivalent formulation of the Hull-White two factor model and is essentially given by the system of SDEs

$$\begin{aligned} \mathrm{d}x &= -ax\,\mathrm{d}t + \sigma\,\mathrm{d}W_1, \\ \mathrm{d}y &= -by\,\mathrm{d}t + \eta\,\mathrm{d}W_2, \end{aligned}$$

where the parameters $a$,$b$, $\sigma$, $\eta$ are assumed to be constant, postive, real numbers. The two Brownian motions are correlated, i.e. $\mathrm{d}W_1\,\mathrm{d}W_2 = \rho\,\mathrm{d}t$, thus adding another parameter $\rho > 0$. In this model, the evolution of the short-rate $r$ is determined in the following way: Assume one has observed the initial term structure at time $t = 0$ in the form of instantaneous forward rates $f(t)$. The simulated short rate in the G2++ model is then given by:

$$r(t) = x(t) + y(t) + f(t), \tag{2.4}$$

where the initial conditions have to be chosen according to

$$x(0) = 0, \qquad y(0) = 0, \qquad r(0) = r_0.$$

From Chapter 4.2. in [3] we can then obtain simulated zero coupon bond prices:

**Proposition 1 (Pricing a zero coupon bond in the G2++ model)** *Under the G2++ model the price of a zero coupon bond*

$$P(t,T) = \mathbb{E}\left[\exp\left(-\int_t^T r(s)ds\right)\right]$$

*is given by*

$$\begin{aligned}
P(t,T) &= \frac{P(0,T)}{P(0,t)}\exp(A(t,T))\\
A(t,T) &= \frac{1}{2}\left[V(t,T)-V(0,T)+V(0,t)\right]\\
&\quad -\frac{1-\exp\left(-a(T-t)\right)}{a}x(t)-\frac{1-\exp\left(-b(T-t)\right)}{b}y(t), \qquad (2.5)
\end{aligned}$$

*where $V$ is given by*

$$\begin{aligned}
V(t,T) &= \frac{\sigma^2}{a^2}\left(T-t+\frac{2}{a}\exp\left(-a(T-t)\right)-\frac{1}{2a}\exp\left(-2a(T-t)\right)-\frac{3}{2a}\right)\\
&\quad +\frac{\eta^2}{b^2}\left(T-t+\frac{2}{b}\exp\left(-b(T-t)\right)-\frac{1}{2b}\exp\left(-2b(T-t)\right)-\frac{3}{2b}\right)\\
&\quad +\frac{2\rho\sigma\eta}{ab}\Bigg(T-t+\frac{1}{a}(\exp(-a(T-t))-1)+\frac{1}{b}(\exp(-b(T-t))-1)\\
&\quad -\frac{1}{a+b}(\exp(-(a+b)(t-t))-1)\Bigg)
\end{aligned}$$

## 2.3 Credit Risk Model

The goal of this section is to define our model for the default of the counterparty. In principle one has the choice between two approaches (see e.g. Chapter 3 in [4]):

1. Structural models
2. Intensity based models

For our purposes we will choose the intensity based approach. Following the seminal paper [20] we suppose that we are given an inhomogeneous Poisson process $N$ with a non-negative intensity function $\lambda$. Then, there holds

$$\mathbb{P}(N_t = 0) = \exp\left(-\int_0^t \lambda(s)ds\right), \qquad N_0 = 0. \qquad (2.6)$$

We may interpret this expression as a survival probability, i.e. the probability that our counterparty does not default in the time interval $[0,t]$. The values $\lambda(t)$ may be interpreted as conditional probabilities, meaning that

$$\mathbb{P}(N_{t+dt} = 1 | N_s = 0, s \leq t) \approx \lambda(t)dt$$

gives the probability of a default occuring in an infinitesimal time interval $dt$ after $t$ under the condition that there has been no default up to $t$.

Given the intensity function $\lambda(t)$, we may already simulate the default time $\tau$: For one simulation of $\tau$ draw a mean one exponentially distributed random variable $E$ and choose $\tau$, such that

$$\tau = \inf_t \left\{ \exp\left( - \int_0^t \lambda(s) ds \right) \geq E \right\} = \inf_t \left\{ \int_0^t \lambda(s) ds \geq - \log E \right\} \tag{2.7}$$

Often, this is taken as the very definition of a default time. In terms of the default time Eq. (2.6) becomes

$$P(\tau > t) = \exp\left( - \int_0^t \lambda(s) ds \right). \tag{2.8}$$

In practice, whenever risk neutral default probababilities are needed for the pricing of a credit risky financial instrument, the default intensity function is bootstrapped from market quotes for Credit Default Swaps. This procedure is similar to that presented in Section 2.1. For more details the reader is referred to [4] and [9].

The approach that we have outlined just now may be readily extended to the setting of Cox processes where $\lambda$ is not a deterministic function but taken as a non-negative stochastic process [20]. In this setting, Eq. (2.8) generalizes to

$$P(\tau > t) = \mathbb{E}\left[ \exp\left( - \int_0^t \lambda(s) ds \right) \right].$$

What remains here is to specify the evolution of $\lambda$. For this thesis, we adopt the same choice as Brigo in [5], that is a CIR++ model. This model is structurally very similar to short rate models for the term structure evolution. To specify the model we start with the SDE

$$\mathrm{d}z = \kappa(\mu - z)\,\mathrm{d}t + \nu\sqrt{z}\,\mathrm{d}W, \qquad z(0) = z_0, \tag{2.9}$$

depending on the three real parameters $\kappa$,$\mu$, $\nu$ and the initial condition $z_0$. Furthermore, assume that we have bootstrapped an initial hazard rate curve $\lambda_0(t)$ from quotes for Credit Default Swaps. Following [3, 4] we set

$$\lambda(t) = z(t) + \lambda_0(t) - \frac{\kappa\mu(\exp(th) - 1)}{2h + (\kappa + h)(\exp(th) - 1)} + z_0 \frac{4h^2 \exp(th)}{(2h + (\kappa + h)(\exp(th) - 1))^2},$$

where $h = \sqrt{\kappa^2 + 2 + \sigma^2}$. This choice of $\lambda$ ensures that our model is consistent with the initially observed hazard rate curve $\lambda_0$. Note that one has to add constraints on the parameters in order to preserve the positivity of $\lambda(t)$ [4].

## 2.4 A Concrete CVA Problem

We now have all the building blocks in place that we need to formulate a concrete CVA model problem. These two building blocks are the G2++ model and the CIR default intensity model as presented earlier in this chapter. Again we now follow our reference paper [5] and combine these models using correlated Brownian motions in order to incorporate the phenomenon of wrong way risk into our model.

The combined system of SDEs reads as:

$$\begin{aligned} \mathrm{d}x &= -ax\,\mathrm{d}t + \sigma\,\mathrm{d}W_1 \\ \mathrm{d}y &= -by\,\mathrm{d}t + \eta\,\mathrm{d}W_2 \\ \mathrm{d}z &= \kappa(\mu - z)\,\mathrm{d}t + \nu\sqrt{z}\,\mathrm{d}W_3, \end{aligned} \tag{2.10}$$

with initial conditions

$$x(0) = 0, \qquad y(0) = 0, \qquad z(0) = z_0.$$

From these equations of motion we obtain the short rate $r$ as:

$$r(t) = x(t) + y(t) + f(t), \qquad r(0) = r_0,$$

where $f(t)$ are the instantaneous forward rates observed in the market. Similarly, we get the default intensity $\lambda$ via

$$\lambda(t) = z(t) + \lambda_0(t) + \gamma(t),$$

with $\lambda_0(t)$ assumed to have been bootstrapped from market quotes for CDS spreads and $\gamma$ defined by

$$\gamma(t) = -\frac{\kappa\mu(\exp(th) - 1)}{2h + (\kappa + h)(\exp(th) - 1)} + z_0 \frac{4h^2 \exp(th)}{(2h + (\kappa + h)(\exp(th) - 1))^2}.$$

Furthermore, there are three correlations entering this model:

$$dW_i dW_j = \rho_{ij} dt.$$

Our goal is to use the above formulation to obtain the CVA for a portfolio of plain vanilla interest rate swaps. Assuming hence that $\Pi_t$ is the risk neutral price process under the above system of SDEs, we want to compute

$$\mathrm{CVA} = (1 - R)\mathbb{E}_0 \left[ \mathbb{1}_{\{0<\tau\leq T\}} D(0, \tau) \Pi_\tau^+ \right].$$

Because we want to avoid explicit sampling of the default indicator and bring the expression into a form that is more suitable for differentation w.r.t parameters entering $\lambda$ (resp. $\tau$), we rewrite the previous expression using $\lambda(t)$. For this end we apply Lemma 1 in [13] and obtain

$$\mathrm{CVA} = (1 - R) \int_0^T \mathbb{E}_0 \left[\lambda(s)\Lambda(s)D(0,s)\Pi_s^+\right] ds,$$

where

$$\Lambda(t) = \exp\left(-\int_0^t \lambda(s)ds\right), \qquad D(0,t) = \exp\left(-\int_0^t r(t)ds\right).$$

Due to the correlation between $\lambda$ and $r$ it is not possible to simplify the expectation under the integral any further. In terms of $\rho$ and $\lambda$ the overall correlation between the interest rate and the default intensity [4] is given by

$$\overline{\rho} = \frac{\sigma\rho_{13} + \eta\rho_{23}}{\sqrt{\sigma^2 + \eta^2 + 2\sigma\eta\rho_{12}}}. \tag{2.11}$$

## 2.5 Remarks on the choice of the model and its calibration

Before we go directly into the matter of an algorithmic realization of the problem outlined in the last section, it seems apt to comment on the choice of our model and its calibration:

1. *Calibration*: Although in the CVA context one deals with a combined credit and market risk model, it appears to be common practice to calibrate the two blocks of the model independently [4, 5]. This means that for the interest rate part we first bootstrap an appropriate zero curve from market quotes for par swap rates (or other suitable instruments) and then determine the remaining parameters using quotes by calibration of the G2++ model to implied volatilities of European swaptions of different tenors and maturities. The initial default intensities are obtained from CDS spread quotes, while the credit spread volatilities and mean reversion parameters are calibrated to implied volatilities for CDS options. The three correlation parameters may be estimated from historical time series data. For the remainder of this thesis we will assume that the calibration procedure has already been dealt with and only work with the final parameter values.

2. *Choice of model*: The main reason for the choice of our model is that is has already been studied in the same context in the paper [5]. However, in order to keep the presentation as streamlined as possible and keep the focus on the computation of sensitivities and its algorithmic realization we have chosen not to model the evolution for $\lambda$ using a jump diffusion but rather an ordinary square root diffusion process.

# Chapter 3

# Algorithmic Realization via Backward Induction

In this chapter we will describe the algorithm that we will use to implement the model described in the previous chapter. It is based on two basic techniques: numerical simulation of a stochastic differential equations and the estimation of conditional expectations via regression. These two techniques will be briefly desribed in the first two sections of this chapter. Both of these sections are based on Glassermann's book [15]. Afterwards we will present the final AMC CVA algorithm, show how it relates to a standard forward Monte-Carlo simulation and finally describe how one can equip the AMC algorithm with sensitivities obtained from adjoint algorithmic differentiation.

Note that the algorithm, that is presented eventually, does not make use of fully optimized individual techniques. Here and there, i.e. regarding the choice of an SDE discretization scheme, the choice of regression basis functions, interpolation schemes, etc. we might not always choose the approach that would be optimal in the sense of accuracy, stability or efficiency. The main reason for this is that we keep focusing more on the eventual goal of producing sensitivities. Furthermore, we note that our simplified choice does not change the overall structure of the algorithm and that we will point out shortcomings of our implemenation wherever it seems necessary.

## 3.1 Numerical Simulation of SDEs

Given a one dimensional diffusion

$$\mathrm{d}S(t) = \mu(S(t))\,\mathrm{d}t + \sigma(S(t))\,\mathrm{d}W,$$

with deterministic functions $\mu$, $\sigma$, a Brownian motion $W$, and a step size $h$ we are interested in finding discrete approximated paths $\widehat{S}(0)$, $\widehat{S}(h)$, $\widehat{S}(2h)$,..., such that for $T = nh$ and constants $c$, $\beta$ there holds

$$\mathbb{E}[|\widehat{S}(nh) - S(T)|] \leq ch^{\beta}.$$

In this case we say that the approximation scheme from which we have obtained $\widehat{S}$ has a strong order of convergence of $\beta$. Alternatively we could require that

$$|\mathbb{E}[f(\widehat{S}(nh))] - \mathbb{E}[f(S(T))]| \leq ch^{\beta},$$

for all sufficiently regular $f$. In this case we speak of a weak order of convergence of $\beta$. As a concrete example for a scheme producing $\widehat{S}$ we consider the so called Euler-Maruyama scheme. For an initial value $S_0$ and a timegrid $t_n = hn$ the scheme reads

$$\widehat{S}(t_{n+1}) = \widehat{S}(t_n) + \mu(\widehat{S}(t_n))h + \sigma(\widehat{S}(t_n))\sqrt{h}\ \ Z_{n+1}. \tag{3.1}$$

The extension to the case where $\mu$ and $\sigma$ are time dependent as well is straightforward:

$$\widehat{S}(t_{n+1}) = \widehat{S}(t_n) + \mu(\widehat{S}(t_n), t_n)h + \sigma(\widehat{S}(t_n), t_n)\sqrt{h}\ \ Z_{n+1}, \tag{3.2}$$

where $Z_i \sim N(0,1)$. It can be shown that this scheme has a strong order of convergence of $\beta = 1/2$ and may produce a weak order of convergence of $\beta = 1$. The situation gets in general more complicated when dealing with a multidimensional state space, i.e. when the vector $S$ of random processes $S_i$ follows the system of SDEs

$$\mathrm{d}S_i = \mu_i(S,t)dt + \Sigma_i(S,t)^T\,\mathrm{d}W.$$

If, however, $\Sigma_i(S,t)^T\,\mathrm{d}W = \sigma_i(S,t)\,\mathrm{d}W_i$, the system of SDEs may be discretized by considering each equation independently from each other. Let us recall the system of SDEs governing our joint evolution of market and credit risk factors:

$$\begin{aligned}
\mathrm{d}x &= -ax\,\mathrm{d}t + \sigma\,\mathrm{d}W_1 \\
\mathrm{d}y &= -by\,\mathrm{d}t + \eta\,\mathrm{d}W_2 \\
\mathrm{d}z &= \kappa(\mu - z)\,\mathrm{d}t + \nu\sqrt{z}\,\mathrm{d}W_3.
\end{aligned}$$

We may easily apply the scheme (3.2) to obtain

$$\begin{aligned}
x_{n+1} &= (1-a)x_n h + \sigma\sqrt{h}\ \ Z_n^1 \\
y_{n+1} &= (1-b)y_n h + \eta\sqrt{h}\ \ Z_n^2 \\
z_{n+1} &= z_n + \kappa(\mu - z_n)h + \nu\sqrt{zh}\ \ Z_n^3,
\end{aligned} \tag{3.3}$$

where $Z_n^i \sim N(0,1)$ and $\mathrm{Corr}(Z_n^i, Z_n^j) = \rho_{ij}$. The above scheme may be readily implemented while taking into account the following two points:

1. The generation of the correlated random numbers $Z_n^i$ is achieved as follows: draw three random, standard uniformly distributed random numbers $Y_i$ and consider the matrix
$$R = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix},$$
together with its Cholesky decomposition $R = LL^T$. Then, the components of the vector $Z = LY$, where $Y = (Y_1, Y_2, Y_3)$ satisfy $\mathrm{Corr}(Z_n^i, Z_n^j) = \rho_{ij}$.

2. Clearly, there is no guarantee that for some $n > 0$ and initial condition $z_0$ we do not have $z_n < 0$, which due to the square root would make the iteration become ill-defined. In [5] Brigo proposes here to use an implicit scheme for the CIR process for $z$ presented by Alfonsi and Brigo in [2]. For this work, however, we will be content with the quick fix of replacing the square-root term in (3.3) with $\sqrt{\max(z_n, 0)}$.

## 3.2 Estimating Conditional Expectations via Regression

The main ingredient in the CVA computation presented in this thesis is a method which allows for a fast estimate of the inner conditional expectation in Eq. (1.1). Unsurprisingly this is the very same approach as the one taken in the famous paper by Longstaff and Schwartz [22]. We stress again that this approach has been put to use in the CVA context in several previous works, see i.e. [5], [10], and [28]. It also appears to be a common choice among practitioners in the field, especially in the context of computing CVA for exotic options which are typically characterized by either a lack of analytic solutions, high dimensional state spaces, early exercise features or a combination of these.

Let us briefly sketch the essentials of this method. A standard reference for this is again the book by Glasserman [15]. Thus, we suppose that we would like to estimate $E = \mathbb{E}[V(X_T)|X_t = x]$, where $V$ is a generic payoff function depending on the process $X_t$. Clearly, the conditional function $E$ can be interpreted as a function of $x$. Naturally, one could try to approximate this function using a linear combination of suitable basis functions $\varphi_i$, that is

$$\mathbb{E}[V(X_T)|X_t = x] \approx \sum_{i=1}^{k} \alpha_i \varphi_i(x) =: \Psi(x, \alpha),$$

with constant coefficients $\alpha_i \in \mathbb{R}$. Assume now, that we are given $N$ realizations of the process $X$ (i.e. generated paths) providing the $2N$ values $X_t^j$, $X_T^j$. We will now use these values to determine the yet unknown coefficients $\alpha_i$ by requiring that they minimize the mean square error, meaning

$$\alpha = \arg\min_\alpha \; \mathbb{E}\left[(\mathbb{E}[V(X_T)|X_t] - \Psi(X_t, \alpha))^2\right].$$

To find $\alpha$ we proceed by considering that the following must hold, if $\alpha$ is indeed minimizing the above expression:

$$\frac{d}{d\alpha}\mathbb{E}\left[\mathbb{E}[(V(X_T)|X_t] - \Psi(X_t, \alpha))^2\right] = 0.$$

Assuming for now that the order of differentiation and expectation may be interchanged we obtain the following $k$ equations:

$$\frac{d}{d\alpha_i}\mathbb{E}\left[(\mathbb{E}[V(X_T)|X_t] - \Psi(X_t, \alpha))^2\right] = -\mathbb{E}\left[2\left(\mathbb{E}[V(X_T)|X_t] - \Psi(X_t, \alpha)\right)\varphi_i(X_t)\right] = 0.$$

These equations can be simplified further using the tower rule for conditional expectations:

$$\mathbb{E}[V(X_T)\varphi_i(X_t)] - \mathbb{E}[\varphi_i(X_t^j)\Psi(X_t, \alpha)] = 0.$$

Introducing the matrix $M$ and the vector $v$, the components of which are defined by

$$M_{ij} = \mathbb{E}[\varphi_i(X_t)\varphi_j(X_t)], \quad v_i = \mathbb{E}[V(X_T)\varphi_i(X_t)], \quad i, j = 1..k,$$

we may finally rewite the above $k$ equations as

$$M\alpha = v.$$

Of course, the expectations defining $M$ and $v$ are usually not available explicitly. To proceed further, we use our $N$ sampled paths $(X_t^l, X_T^l)$ and introduce estimates $\widehat{M}$, $\widehat{v}$ via

$$\widehat{M}_{ij} = \frac{1}{N}\sum_{l=1}^{N} \varphi_i(X_t^l)\varphi_j(X_t^l), \quad \widehat{v}_i = \frac{1}{N}\sum_{l=1}^{N} V(X_T^l)\varphi_i(X_t^l),$$

so that we may readily compute $\alpha$ by

$$\alpha = \widehat{M}^{-1}\widehat{v},$$

which amounts to solving a linear system of $k$ equations. It remains to choose a suitable set of basis functions $\phi$. A natural choice would, of course, be given by the ordinary monomials $1,\ X,\ X^2,\ \ldots$. Alternatives are given by Hermite, Legendre, Chebyshev or LaGuerre polynomials (see e.g. [19]). Note that there are no general rules for an optimal choice of the basis functions.

## 3.3 The Final Algorithm

We are now in a position to formulate an American Monte Carlo based algorithm which lets us compute an approximation to

$$\mathrm{CVA} = (1 - R) \int_0^T \mathbb{E}_0 \left[ \lambda(s) \Lambda(s) D(0, s) \Pi_s^+ \right] ds,$$

using the combined interest rate and credit risk model described at the end of Chapter 2. We first present the algorithm for a single interest rate swap only. Later on we will describe the extension to a whole portfolio of interest rate swaps. This algorithm should be considered as a concrete adaption of the method presented in [1].

**Algorithm: CVA by backward induction for a single IR swap**

**Input**

1. A fixed vs floating interest rate swaps characterized by
   (a) Fixed interest rate $K$
   (b) Fixed start date $T_0$
   (c) Notional $\mathcal{N}$
   (d) Payment dates $T_1,\ \ldots,\ T_N$. Fixings for the following accrual period are assumed to take place on the payment date for the previous period. The first fixing is assumed to take place on $T_0$. Furthermore, for the sake of presentation we assume that payment frequencies and day count conventions for the fixed and floating legs coincide.
   (e) A function $\delta(S,T)$ giving the year count fraction for a period $[T_i, T_{i+1}]$.
2. An initial interest rate term structure $P(0,t)$ together with instantaneous forward rates $f(t)$. We assume that the algorithm is provided with a set of initial zero rates $(t_i, R_i)$ and an interpolation scheme, which eventually lets us compute an approximate value of $f(t)$ for any $t \in [T_0, T_N]$.

3. An initial term structure of hazard rates given by $\lambda_0(t)$. Again, we assume that the algorithm is provided with a set of initial hazard rates $(t_i, \lambda_i)$ and an interpolation scheme.

4. Model parameters $\sigma$, $\eta$, $\kappa$, $\mu$, $z_0$, $\rho_{12}$, $\rho_{23}$, $\rho_{13}$.

5. A step size $h$. For presentational reasons we assume here that $T_i = iJh$ for a fixed integer $J > 0$.

**Algorithm**

1. Initialization: Perform a Cholesky-Decomposition of the correlation matrix

$$R = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix} = LL^T.$$

2. Forward pass: for every path $p = 1, \ldots, M$ execute the following steps:

   (a) Use the scheme

$$\begin{aligned} x_{n+1}^{(p)} &= (1-a)x_n^{(p)}h + \sigma\sqrt{h}\ Z_n^1 \\ y_{n+1}^{(p)} &= (1-b)y_n^{(p)}h + \eta\sqrt{h}\ Z_n^2 \\ z_{n+1}^{(p)} &= z_n + \kappa(\mu - z_n^{(p)})h + \nu\sqrt{\max(z_n^{(p)}, 0)}\sqrt{h}\ Z_n^3, \end{aligned} \tag{3.4}$$

   where $Z = (Z_1, Z_2, Z_3)^T = L(Y_1, Y_2, Y_3)^T$ and $Y_i \sim N(0,1)$ in order to obtain for $k = 0, \ldots, N \cdot J$

$$r_k^{(p)} = x_k^{(p)} + y_k^{(p)} + f(k \cdot h), \qquad \lambda_k^{(p)} = z_k^{(p)} + \lambda_0(k \cdot h) + \gamma(k \cdot h)$$

   as well as approximate discount factors

$$D_k^{(p)} = \exp\left(-h\sum_{j=1}^{k} r_j^{(p)}\right), \quad \Lambda_k^{(p)} = \exp\left(-h\sum_{j=1}^{k} \lambda_j^{(p)}\right),$$

   so that we may set

$$r^{(p)}(T_i) = r_{i\cdot J}^{(p)}, \qquad \lambda^{(p)}(T_i) = \lambda_{i\cdot J}^{(p)},$$

$$D^{(p)}(T_i) = \exp\left(-h\sum_{j=1}^{i\cdot J} r_j^{(p)}\right), \quad \Lambda^{(p)}(T_i) = \exp\left(-h\sum_{j=1}^{i\cdot J} \lambda_j^{(p)}\right).$$

Furthermore we apply Proposition 1 in order to obtain $P^{(p)}(k \cdot h, T_i)$ and $P^{(p)}(k \cdot h, T_{i+1})$, so that we may compute the simulated Libor fixings along path $p$ via

$$L^{(p)}(T_i, T_{i+1}) = \frac{1}{\Delta}\left(\frac{P^{(p)}(k \cdot h, T_i)}{P^{(p)}(k \cdot h, T_{i+1})} - 1\right),$$

where $\Delta$ denotes the year fraction suitable for the tenor of the floating leg of the swap in question. Additionally, we compute approximate discount factors for periods $[k \cdot h, l \cdot h]$ by

$$D_{k,l}^{(p)} = \exp\left(-h \sum_{j=k}^{l} r_j^{(p)}\right).$$

3. Backward pass

   (a) Evaluate swap payoff at maturity, that is for every path compute

   $$\Pi_N^{(p)} = \phi\delta(T_{N-1}, T_N)\mathcal{N}(L^{(p)}(T_{N-1}, T_N) - K),$$

   where $\phi \in \{-1, 1\}$ depending on whether we have a payer or receiver swap.

   (b) for $k = JN - 1, \ldots, 1$

      i. Determine regression coefficients $\alpha_i$ by regressing on the discounted payoffs $D_{k,k+1}^{(p)}\Pi_{k+1}^{(p)}$ and regression variables $(x_k^{(p)}, y_k^{(p)})$ using the basis functions $\varphi(X, Y) = (1, X, Y, XY, X^2, Y^2)$.

      ii. Set

      $$\Pi_k^{(p)} = \sum_{i=1}^{6} \alpha_i \varphi_i(x_k^{(p)}, y_k^{(p)}) + \beta,$$

      where $\beta = 0$ for $k \neq J \cdot l$ for some $l$ (meaning $t = k \cdot h$ is not a payment date) and

      $$\beta = \phi\delta(T_{l-1}, T_l)\mathcal{N}(L^{(p)}(T_{l-1}, T_l) - K),$$

      for $k = J \cdot l$ (meaning a payment occurs on $t = k \cdot h$) .

   (c) Evaluate the swap at $t = 0$ using the regular pricing formula and initial term structure $P(0, t)$ and use this value as $\Pi_0^{(p)}$ for all $p$.

4. Final forward aggregation sweep

   (a) For every $k = 0, \ldots, N$

i. Compute the expected exposure profile by

$$\mathrm{EE}(kh) = \frac{1}{M}\sum_{j=1}^{M} \max(\Pi_k^{(j)}, 0).$$

ii. Compute the discounted and probability weighted expected exposure profile via

$$\widetilde{\mathrm{EE}}(kh) = \frac{1}{M}\sum_{j=1}^{M} \lambda_k^{(j)} \Lambda_k^{(j)} D_k^{(j)} \max(\Pi_k^{(j)}, 0).$$

(b) Apply the trapezoidal rule to $\widetilde{\mathrm{EE}}(kh)$ and multiply with $(1-R)$ to get the final CVA figure.

As output we have thus obtained the CVA, as well as an exposure profile $\mathrm{EE}(kh)$.

At this point we should make a few remarks regarding the algorithm above. Firstly, it uses the same set of paths for determining the regression coefficients and the conditional expectations itself. One can assume that this would allow for a potential bias on top of the bias introduced by the regression itself. Secondly, as basis functions we have chosen ordinary monomials up to degree 2. This is again another possible source for an increased bias in the estimated conditional expectation.

As this thesis is mostly concerned with the question of how to equip an algorithm structured like the above with AD sensitivities, we shall be content with the formulation as it stands. For a possible list of improvements, the reader is referred to the final Chapter.

It remains to show how the above method may be extended to a portfolio of interest rate swaps. Luckily, this is a rather straightforward exercise which amounts to the following modifications:

1. Step 3. a.: Substitute the individual payoff by the sum of the final payoffs of all swaps.
2. Step 3. b.: Similarly, substitute $\beta$ with the sum of the payoffs of all swaps in the portfolio.
3. Step 3. c.: Instead of pricing the single swap only, compute the sum of present values of all swaps in the portfolio.

## 3.4 Backward Induction vs Forward Induction

As we have mentioned in Section 1.3, there exists an alternative to the algorithm presented in the previous subsection, which is based on a single pass forward Monte Carlo simulation. Later on, we will compare both algorithms.

**Algorithm: CVA by forward induction for a single IR swap**

**Input: See backward induction algorithm**

**Algorithm**

1. Initialization: Perform a Cholesky-Decomposition of the correlation matrix

$$R = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix} = LL^T.$$

2. For every path $p = 1, \dots, M$ execute the following steps:

   (a) Use the scheme

$$\begin{aligned} x_{n+1}^{(p)} &= (1-a)x_n^{(p)}h + \sigma\sqrt{h}\ Z_n^1 \\ y_{n+1}^{(p)} &= (1-b)y_n^{(p)}h + \eta\sqrt{h}\ Z_n^2 \\ z_{n+1}^{(p)} &= z_n + \kappa(\mu - z_n^{(p)})h + \nu\sqrt{\max(z_n^{(p)}, 0)}\sqrt{h}\ Z_n^3, \end{aligned}$$

   where $Z = (Z_1, Z_2, Z_3)^T = L(Y_1, Y_2, Y_3)^T$ and $Y_i \sim N(0,1)$ in order to obtain for $k = 0, \dots, N \cdot J$

$$r_k^{(p)} = x_k^{(p)} + y_k^{(p)} + f(k \cdot h), \qquad \lambda_k^{(p)} = z_k^{(p)} + \lambda_0(k \cdot h) + \gamma(k \cdot h),$$

   as well as approximate discount factors

$$D_k^{(p)} = \exp\left(-h \sum_{j=1}^{k} r_j^{(p)}\right), \quad \Lambda_k^{(p)} = \exp\left(-h \sum_{j=1}^{k} \lambda_j^{(p)}\right).$$

   (b) Use Proposition 1 in order to compute the zero bonds $P^{(p)}(k \cdot h, T_i)$ for all $T_i \geq kh$ and use them in turn to determine the price $\Pi_k^{(p)}$ of the underlying swap at time $t = k \cdot h$ using (2.1).

3. Aggregation and integration

(a) For every $k = 0, \dots, N$

i. Compute the expected exposure profile by

$$\mathrm{EE}(kh) = \frac{1}{M}\sum_{j=1}^{M} \max(\Pi_k^{(j)}, 0).$$

ii. Compute the discounted and probability weighted expected exposure profile via

$$\widetilde{\mathrm{EE}}(kh) = \frac{1}{M}\sum_{j=1}^{M} \lambda_k^{(j)}\Lambda_k^{(j)}D_k^{(j)}\max(\Pi_k^{(j)}, 0).$$

(b) Apply the trapezoidal rule to $\widetilde{\mathrm{EE}}(kh)$ and multiply with $(1-R)$ to get the final CVA figure.

We note the following:

1. The algorithm based on forward induction avoids the computation of conditional expectations. We thus expect it to exhibit less bias and a lower variance.

2. Except for the final aggregation, all computations are performed within individual single paths and fit within the structure of Eq. (1.9). Thus, computing sensitivities can be achieved using the traditional path-wise approach as outlined in Chapter 1.

3. Instead of $j \cdot N$ regressions steps it involves the computation of $j \cdot N \cdot M$ present values. While this might not cause major problems in the case of vanilla instruments such as interest rate swaps, it can lead to significantly longer computation times for more complicated instruments when compared to the algorithm based on backward induction, which always relies on the same method (regression) to obtain present values for the underlying instruments.

Clearly, the concrete choice of whether to use a forward or backward induction based CVA computation depends on the nature of the underlying portfolio of trades.

## 3.5 Adjoint Algorithmic Differentiation using dco-c++

Both of the two algorithms described previously have been implemented in C++ code. In this section we will now describe how both of the implementations can be equipped with sensitivities using the AAD tool dco-c++.

However, before we go directly into the matter, we first specify which derivatives we want to compute exactly. These are the following:

1. $\partial\mathrm{CVA}/\partial R_i$ and $\partial\mathrm{CVA}/\partial\lambda_i$: The sensitivities w.r.t. the initial zero rates and initial hazard rates. We may view these as Delta sensitivities.

2. $\partial\mathrm{CVA}/\partial\sigma$ and $\partial\mathrm{CVA}/\partial\eta$: The sensitivities w.r.t. the interest rate volatility model parameters. We may view these as Vega sensitivities.

3. $\partial\mathrm{CVA}/\partial\rho_{ij}$: The sensitivities w.r.t. the correlations.

Depending on the concrete parametrization of the initial yield curve and hazard rate curve this can make for about 50 individual derivatives of the single real valued function CVA. Thus, it seems like a good choice to use the backward mode of automatic differentiation. Furthermore, we will restrain ourselves to first order sensitivities only.

dco-c++ is based on the concept of operator and function overloading. In the adjoint mode this means that all operations performed on the special dco-c++ datatypes are recorded on a so-called *tape*, meaning that dco-c++ is keeping track of all individual jacobians produced along the execution of individual operations (compare this to the example in Sect. 1.2.1). Once the execution of the function has finished, the flow of operations is automatically reversed and the previously recorded tape is executed backwards, eventually giving the desired derivatives.

For concrete purposes this means that all occurences of floating point variables (i.e. C++ doubles) have to be substituted by the correct dco-c++ datatype. Then, the code has to be inspected for statements which can affect the differentiability of the function of part of it. While this is in general no problem for loops, special care has to be taken whenever an if-statement involves a numerical variable. Finally, the function call that is to be differentiated needs to be wrapped into a few lines of driver code needed to mark independent and dependent variables, as well as performing the backward execution of the tape. To develop a better feeling for this, one should inspect an example in the dco-c++ documentation [24].

While it seems that this approach will quickly lead to a fully differentiated piece of source code, one should be aware of the following points:

1. Source code of all numerical methods need to be available as C++ code as well and has to be adjusted regarding the numerical datatype. For our two algorithms this means that we need to have C++ source code available for

(a) Interpolation of yield curves.

(b) LU decomposition for the solution of linear equations.

(c) Cholesky decomposition for the generation of correlated Brownian increments.

2. The backward mode of AD requires recording operations on a tape which is kept in memory during the execution of the program. This means that, when one naively applies AD, memory consumption can quickly reach critical levels. This is where a technique called "checkpointing" (see for instance [17]), which is also supported by dco-c++, comes into play.

3. Any code differentiated using AD still needs to be validated using standard bumping methods. To facilitate this, it is advisable to use C++ templates for all code whereever possible in order to be able to quickly exchange one datatype for another (i.e. double and the corresponding c++ datatype).

For the remainder of this section we now proceed by showing how we have dealt with the first two points.

Dealing with the first item is not overly difficult. For this work, we have manually implemented a linear interpolation and dealt with the two matrix decompositions using standard, publicly available, code [29].

The second point, however, requires more care. Before describing how it has been dealt with concretely, we briefly sketch the idea behind the checkpointing technique. Suppose for instance, that a numerical algorithm that is to be differentiated using the backward mode of AD requires repeated execution of a function $g : \mathbb{R}^n \to \mathbb{R}$. One can think of $g$ as all operations that are performed for an indivdual path in a Monte Carlo simulation. Now, instead of having dco-c++ record all operations coming from the repeated execution of $g$ we "wrap" $g$ inside a dco checkpoint:

1. Before the execution of $g$ in the forward pass, the input $x$ to $g$ is saved in a place in memory reserved for the checkpoint. In practice the input to $g$ may consist of actual independent variables, as well as passive parameters. Thus, we are actually saving the whole state of the computation at this particular point in time.

2. We execute $g$ in AD disabled mode, e.g. by calling another version of $g$ implemented using standard C++ doubles. Thus, no operations are recorded on the tape, which obviously limits its growth.

3. After the execution of $g$ we save the output of $g$ in the data structure reserved for the checkpoint.

4. Once, the AD backward pass is executed and control flow reaches the point where the derivative of $g$ is to be computed, all information is read from the corresponding checkpoint. Then, a "sub AD process" is run for every checkpointed created for $g$ on a local tape, the memory occupied by which is freed afterwards. This leads to a oscillating amount of memory being consumed by the process.

Given an ordinary Monte Carlo option pricing problem, we can think of this approach as a concrete technical realization of the pathwise sensitivities approach, where the derivatives are computed individually along each path. This is also the approach that we have applied for differentiating the forward induction based algorithm, which is mostly based on independent path-wise computations.

While checkpointing clearly reduces memory consumption it introduces an extra computational overhead. Thus, for most concrete implementations there will always be a trade off between execution speed and memory requirements.

For the differentiation of the backward induction based algorithm we have applied the following checkpointing strategy:

1. All pathwise computations in the forward pass are wrapped in one individual checkpoint.

2. Every regression corresponds to another checkpoint.

3. Summation over all paths for estimating $\widetilde{\mathrm{EE}}(t_i)$ is wrapped in checkpoints for every $t_i$.

This strategy has been successively developed in the form of a large number of numerical experiments. It would be of great interest to see how it would fit with an optimal checkpinting strategy produced by the algorithm Revolve [16].

# Chapter 4

# Numerical Results

In this chapter we present a series of numerical results obtained using our implementations of the backward and forward induction based algorithms. Unless stated otherwise, we will use the set of parameters as displayed in Table 4.1.

| Parameter | Value |
|---|---|
| $\sigma$ | 0.0093 |
| $\eta$ | 0.0138 |
| $\rho_{12}$ | -0.7 |
| $\rho_{23}$ | 0.0 |
| $\rho_{13}$ | 0.0 |
| $\lambda$ | 0.07 |
| $a$ | 0.058 |
| $b$ | 0.5493 |
| $\kappa$ | 0.4 |
| $\mu$ | 0.14 |
| $\nu$ | 0.14 |
| $z_0$ | 0.0165 |

Table 4.1: Standard parameters used for the examples in this Chapter

To streamline the presentation we have chosen to use a constant hazard rate given by $\lambda_0(t) = \lambda = 0.07$ which corresponds to a stylized credit spread of 700bps. The initial interest rate term structure, defined in continously compunded zero rates with ACT/ACT day count convention can be seen in Table 4.2 and Figure 4.1.

The zero rates have been bootstrapped using the library QuantLib [12] from fixed for float par swap rates as displayed in Table 4.3. The underlying swaps are assumed to have a 6M payment frequency for the float the floating leg and annual payments for the fixed leg.

| $t_{1..9}$ | $R_{1..9}$ | $t_{10..19}$ | $R_{10..19}$ | $t_{20..29}$ | $R_{20..29}$ | $t_{30..33}$ | $R_{30..33}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0.0177558 | 4.49513 | 0.0223713 | 8.99924 | 0.0280456 | 13.4951 | 0.0316604 |
| 0.495134 | 0.0177558 | 4.99924 | 0.0228294 | 9.49513 | 0.0286111 | 13.9992 | 0.031921 |
| 0.999244 | 0.0177558 | 5.49513 | 0.0239433 | 9.99924 | 0.0291297 | 14.4951 | 0.0321597 |
| 1.49513 | 0.0190884 | 5.99924 | 0.0248948 | 10.4951 | 0.0295913 | 14.9992 | 0.0323862 |
| 1.99924 | 0.0197884 | 6.49513 | 0.0254371 | 10.9992 | 0.0300179 | | |
| 2.49513 | 0.0203974 | 6.99924 | 0.025907 | 11.4973 | 0.0304026 | | |
| 2.99924 | 0.0208129 | 7.49727 | 0.0264736 | 12 | 0.0307585 | | |
| 3.49727 | 0.02138 | 8 | 0.0269776 | 12.4951 | 0.0310811 | | |
| 4 | 0.0218119 | 8.49513 | 0.0275375 | 12.9992 | 0.0313842 | | |

Table 4.2: Input zero rates defining the initial term structure of interest rates.

| Maturity | Rate |
|---|---|
| 1Y | 0.018 |
| 2Y | 0.02 |
| 3Y | 0.021 |
| 4Y | 0.022 |
| 5Y | 0.023 |
| 6Y | 0.025 |
| 7Y | 0.026 |
| 8Y | 0.027 |
| 9Y | 0.028 |
| 10Y | 0.029 |

Table 4.3: Input swap rates

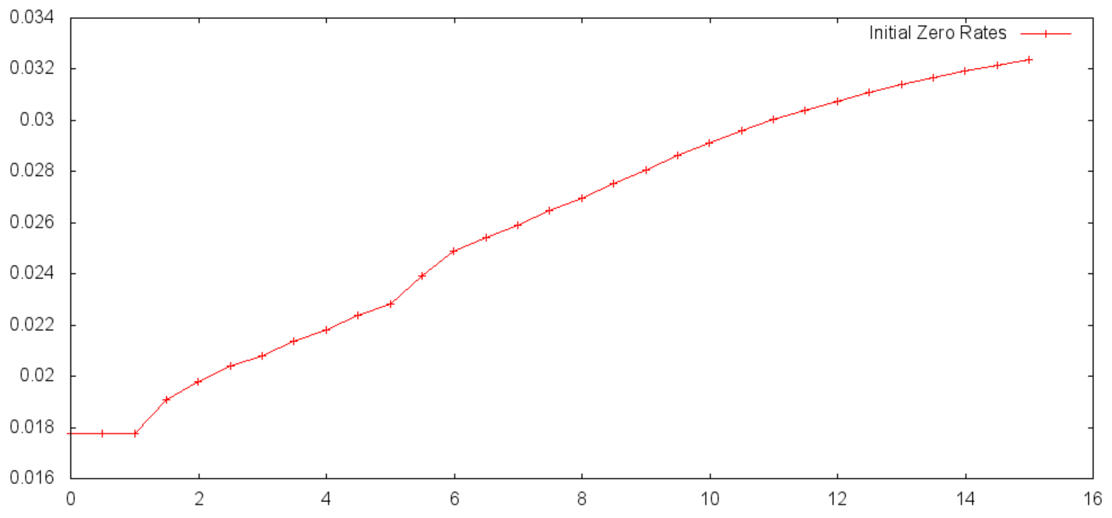

Figure 4.1: Plot of initial zero rates. X-axis: time in years, Y-axis: corresponding zero rate

## 4.1 Single IR Swap

We first test our backward induction CVA algorithm on a single interest rate swap that is defined by the following characteristics:

- Payer swap with notional of 1,000,000. Furthermore there is no exchange of notional at start and end of the swap.
- Fixed rate $K = 0.029$. This ensures that the swap's PV as of $t = 0$ equals zero.
- The payment frequency of both swap legs equals $6M$. Fixing Dates are assumed to coincide with payment dates.

The exposure profile is evaluated on a bi-monthly (60 days) time grid. Initially, we use $N = 1000$ paths. The resulting exposure profile can be seen in Figure 4.2.

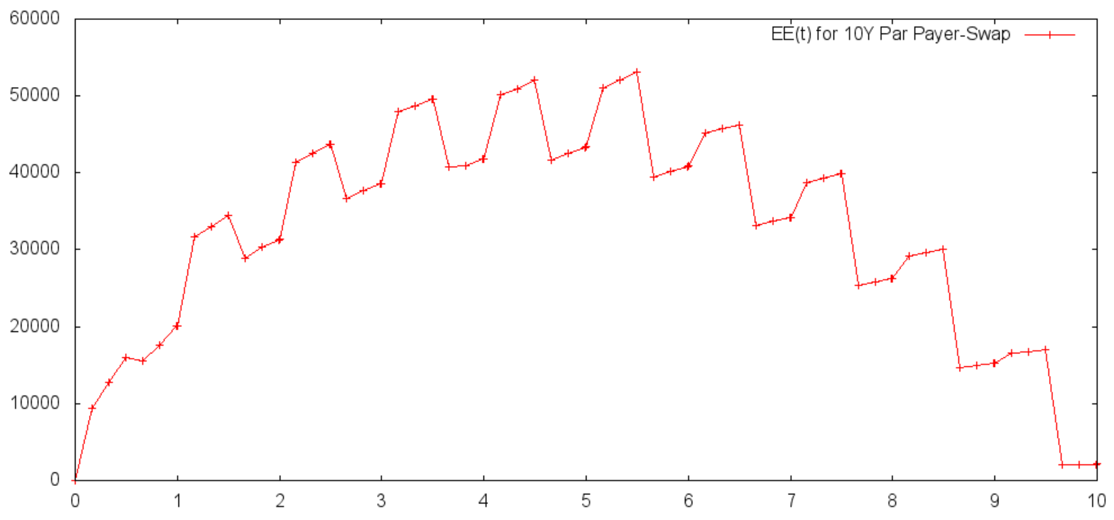

Figure 4.2: EE(t) for a 10Y Payer Swap. X-axis: time in years, Y-axis: exposure in units of currency

A plot of the resulting sensitivities $(t_i, \partial\text{CVA}/\partial R_i)$ is shown in Figure 4.3. Often, one is interested how these sensitivities correspond to par swap rate sensitivities. We therefore numerically approximate the Jacobian of the QuantLib curve bootstrapping procedure using ordinary finite differences and multiply it on the vector $\partial\text{CVA}/\partial R$. This gives us the CVA sensitivities w.r.t. to par swap rates. They are displayed in Figure 4.4. Finally, we report concrete figures for the CVA and its remaining sensitivities in Table 4.4. Error estimations are presented in the next subsection.

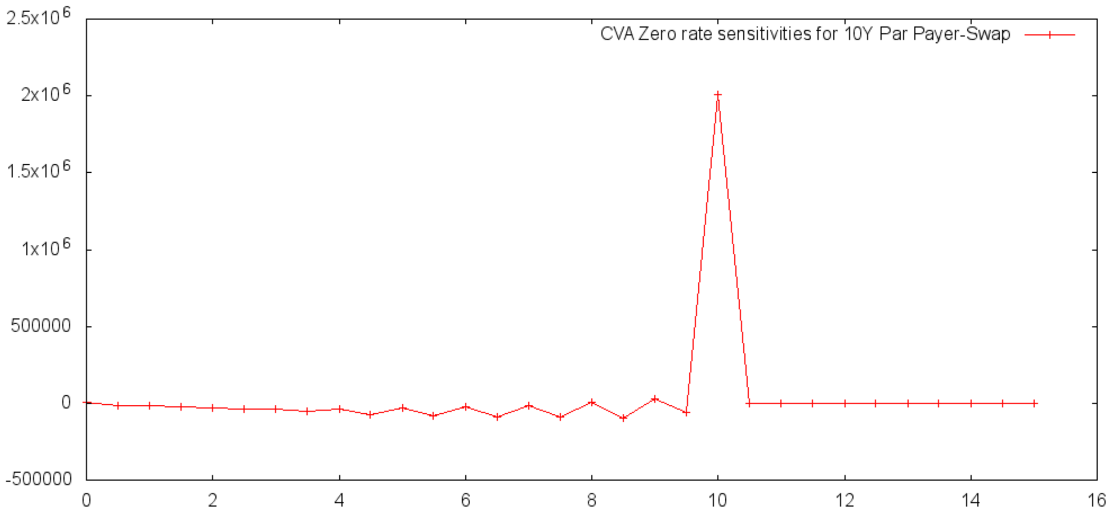

Figure 4.3: CVA Zero Sensitivities for 10Y Payer Swap. X-axis: time in years, Y-axis: corresponding zero rate sensitivity

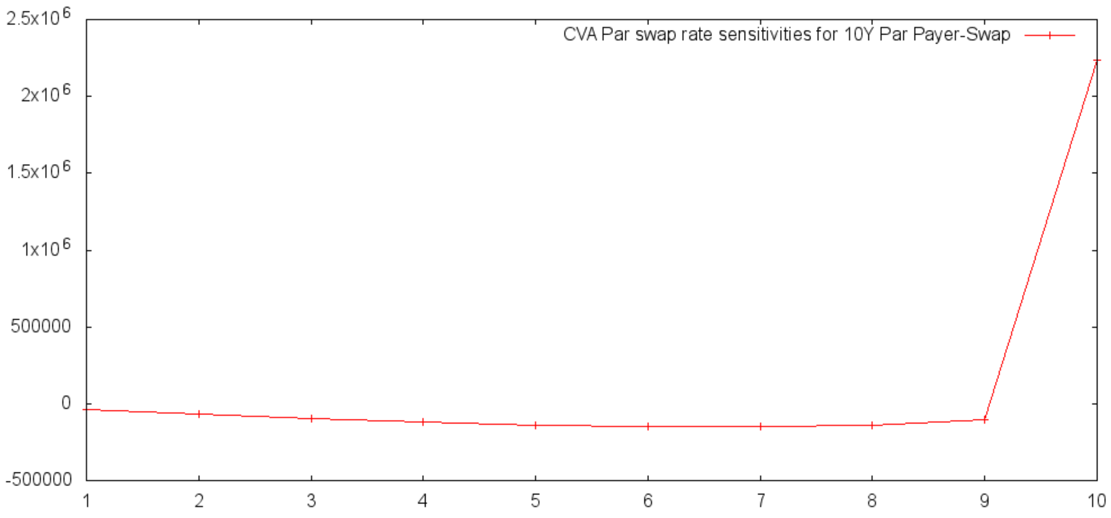

Figure 4.4: CVA Par Sensitivities 10Y Payer Swap. X-axis: time in years, Y-axis: corresponding par rate sensitivity

| CVA | 14581.8 |
|---|---|
| Parameter | $\partial \text{CVA}/\partial\theta$ |
| $\rho_{23}$ | 907.19 |
| $\rho_{12}$ | 1752.2 |
| $\rho_{13}$ | 2816.56 |
| $\sigma$ | 1080790 |
| $\eta$ | -92566.5 |
| $\lambda$ | 146525 |

Table 4.4: CVA and sensitivities for single IR swap

Before proceeding to the next subsection we check that the AD sensitivities of the backward induction algorithm can be reproduced using standard bumping techniques. To achieve this we run the same algorithm but approximate all sensitivities using ordinary central differences for an increasingly smaller $\epsilon$. The result is displayed in Figure 4.5 which shows a log-log plot of the average relative error of the finite difference and AD sensitivities vs. the precision used for the finite difference approximation. (i.e. a x-value of 4 means that $\epsilon = 10^{-4}$ was used). Some numbers used to produce this plot are found in Table 4.5.

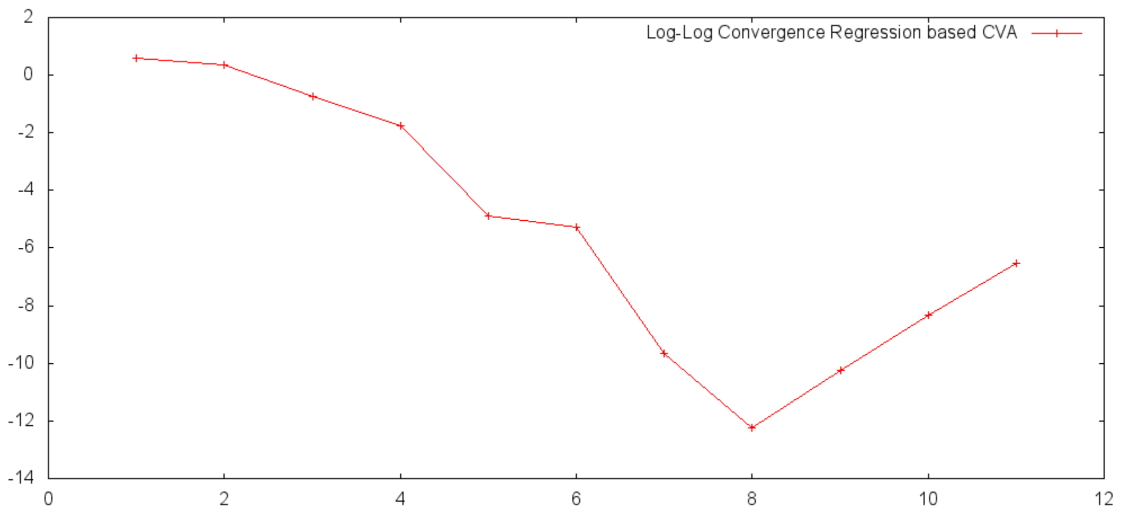

Figure 4.5: Convergence of central difference approximation to AD sensitivities. X-Axis: $-\log_{10}\epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities.

We observe convergence up to $\epsilon = 10^{-8}$. For smaller $\epsilon$ we conjecture that finite precision arithmetic errors prevent further convergence. To further validate correctness of the implementation we compare the sensitivities obtained using backward AD with sensitivities computed using the forward mode. For that end we have implemented an additional driver routine which makes use of the dco-c++ forward mode implementation instead of the adjoint mode implementation. The relative errors between the sensitivities obtained with backward and forward mode AD are reported in Table 4.6.

| | $\epsilon = 10^{-4}$ | $\epsilon = 10^{-8}$ | Backward AD |
|---|---|---|---|
| $\partial CVA/\partial R_1$ | 6459.68159386939 | 6459.61727059329 | 6459.62012485053 |
| $\partial CVA/\partial R_2$ | -15953.1833132132 | -15959.0430484968 | -15959.0443319228 |
| $\partial CVA/\partial R_3$ | -16815.3796103069 | -16809.0329680126 | -16809.0348441664 |
| $\partial CVA/\partial R_4$ | -24884.2406715721 | -24884.6655267698 | -24884.668363348 |
| $\partial CVA/\partial R_5$ | -27796.3870959865 | -27805.9337688318 | -27805.9286101873 |
| $\partial CVA/\partial R_6$ | -41151.188514068 | -41125.4265600291 | -41125.424055725 |
| $\partial CVA/\partial R_7$ | -35330.4923195901 | -35312.3530658194 | -35312.3481072624 |
| $\partial CVA/\partial R_8$ | -55376.323912942 | -55373.4702407382 | -55373.4719915871 |
| $\partial CVA/\partial R_9$ | -35416.3345479264 | -35457.2080141224 | -35457.2053699844 |
| $\partial CVA/\partial R_{10}$ | -74702.4308028267 | -74661.6597098181 | -74661.6578406115 |
| $\partial CVA/\partial R_{11}$ | -31093.3809445396 | -31112.1876620745 | -31112.1909230498 |
| $\partial CVA/\partial R_{12}$ | -85100.7503516211 | -85078.6434057226 | -85078.6419083269 |
| $\partial CVA/\partial R_{13}$ | -19942.1472272661 | -19935.5839868076 | -19935.5836477524 |
| $\partial CVA/\partial R_{14}$ | -91662.0051957397 | -91756.9719604216 | -91756.9707908714 |
| $\partial CVA/\partial R_{15}$ | -12777.0615469034 | -12680.7913147786 | -12680.7878198266 |
| $\partial CVA/\partial R_{16}$ | -87527.05119673 | -87497.2110977978 | -87497.2092993284 |
| $\partial CVA/\partial R_{17}$ | 8920.83498333704 | 8993.40393516467 | 8993.40087037561 |
| $\partial CVA/\partial R_{18}$ | -96237.2185331969 | -96216.0882409079 | -96216.0876020922 |
| $\partial CVA/\partial R_{19}$ | 28779.9966418151 | 28911.3268081564 | 28911.3254769239 |
| $\partial CVA/\partial R_{20}$ | -59461.9816121939 | -59468.6008298594 | -59468.6006341257 |
| $\partial CVA/\partial R_{21}$ | 2281035.8563757 | 2011188.28235848 | 2011188.28064756 |
| $\partial CVA/\partial \rho_{23}$ | 907.341823212846 | 907.194407773204 | 907.194435116284 |
| $\partial CVA/\partial \rho_{12}$ | 1752.2060828287 | 1752.19811353599 | 1752.19818535462 |
| $\partial CVA/\partial \rho_{13}$ | 2816.8728682067 | 2816.55466096708 | 2816.55546201239 |
| $\partial CVA/\partial \sigma$ | 1080926.121053 | 1080788.28352518 | 1080788.28375513 |
| $\partial CVA/\partial \eta$ | -92668.9697165147 | -92566.533567151 | -92566.5337267274 |
| $\partial CVA/\partial \lambda$ | 146524.545353213 | 146524.525371205 | 146524.525292698 |

Table 4.5: Comparison of sensitivites obtained using central difference approximation and backward mode AD.

During the implementation of the forward mode sensitivites an interesting phenomen was observed: while the non-differentiability of the term $\sqrt{\max(z_n^{(p)}, 0)}$ in Eq. (3.4) did not have any effect on the sensitivity computation in backward mode, it did not allow for reproducing these sensitivites in the forward mode. After changing the term to $\sqrt{\max(z_n^{(p)}, \epsilon)}$, with $\epsilon = 10^{-8}$, forward and backward sensitivities were identical up to the errors reported in 4.6.

| Parameter | Relative error |
|---|---|
| $R_i$ | $5.7217026740043400 \cdot 10^{-15}$ (avg.) |
| $\rho_{23}$ | $7.6192207510671100 \cdot 10^{-15}$ |
| $\rho_{12}$ | $5.7096544498594800 \cdot 10^{-15}$ |
| $\rho_{13}$ | $3.5542562881378500 \cdot 10^{-15}$ |
| $\sigma$ | $2.7789982305600400 \cdot 10^{-14}$ |
| $\eta$ | $1.1004241600292700 \cdot 10^{-15}$ |
| $\lambda$ | $3.4164017281807400 \cdot 10^{-14}$ |

Table 4.6: Comparison of sensitivites obtained using forward and backward mode AD

Overall, we may assume that our implementation provides correct sensitivity figures.

### 4.1.1 Monte Carlo Errors

We now investigate, whether our implementation of the backward induction algorithm actually convergences with the number of MC paths increasing. To check this, we have run the algorithm on an increasing number of paths. To limit the execution time and memory growth for this experiment we have chosen the underlying swap to have maturity of 5Y. The timegrid has been adjusted to coincide with the payment dates of the swap. The resulting plot is shown in Figure 4.6.

To estimate the variance of the estimator we have run the setup as described in the previous subsection using $N = 1000$ and $N = 10000$ paths creating 1000 samples each using a different random seed. Results are presented in Tables 4.7, 4.8 and 4.9. It appears there that the standard deviations seem large compared to change in means for $N = 1000$, $N = 10000$. However, repeating the experiment with different sets of parameters we find that the difference in means may actually be much larger, indicating an isolated numerical curiosity.

### 4.1.2 Comparison with Forward Induction Algorithm

We proceed by comparing the backward and forward induction algorithms. First, we compare the two exposure profiles generated by both methods. This is shown in Figure 4.7. While there is little qualitative difference between the exposure profiles we observe differences in the resulting sensitivities for the zero rates (See Figure 4.8).

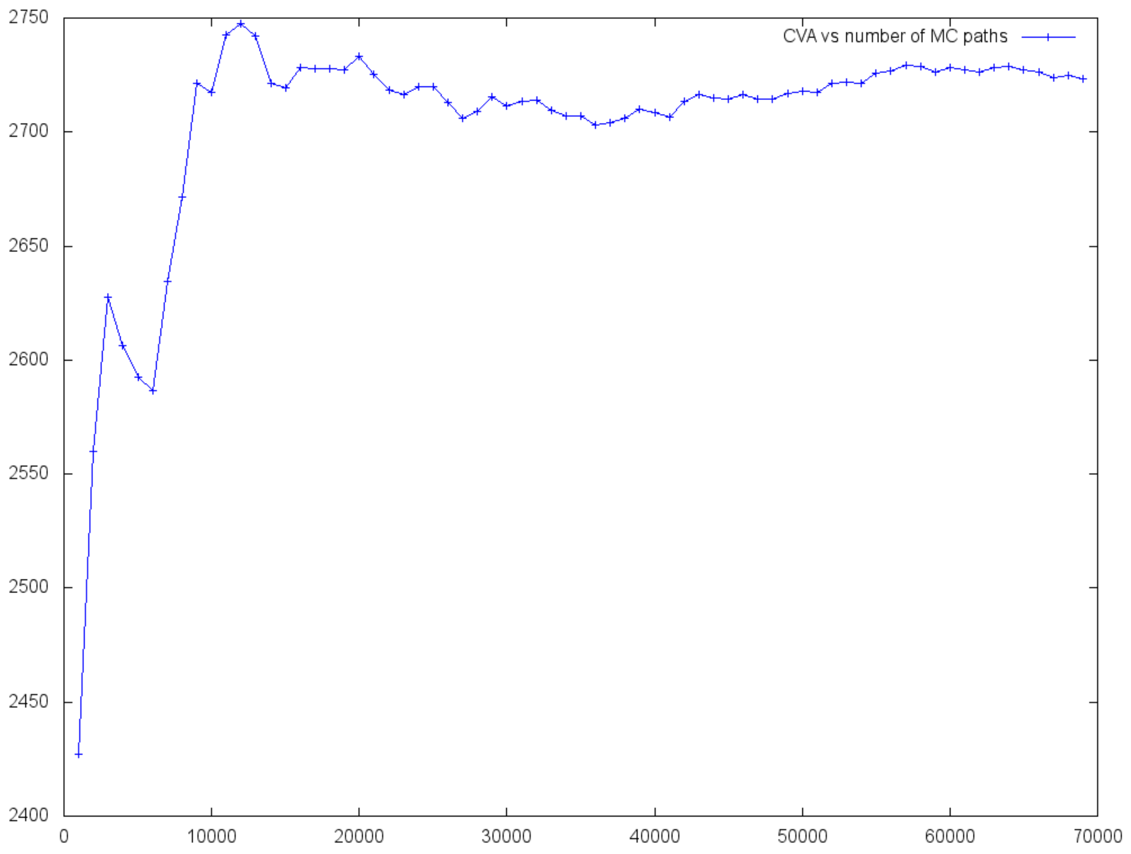

Figure 4.6: Convergence of the AMC based backward induction algorithm. X-axis: number of paths, Y-axis: resulting CVA in units of currency

| | | CVA | $\partial\text{CVA}/\partial R_i$ (average) | $\partial\text{CVA}/\partial r_{12}$ |
|---|---|---|---|---|
| N=1000 | Mean | 14226.3618 | 57331.03778 | 929.366993 |
| N=1000 | Standard Deviation | 700.0110557 | 4340.881238 | 172.5297199 |
| N=1000 | Standard Deviation (percent w.r.t swap notional) | 0.070001106 | 0.434088124 | 0.017252972 |
| N=10000 | Mean | 14226.8469 | 57370.65987 | 937.17208 |
| N=10000 | Standard Deviation | 217.8726233 | 1388.533872 | 52.86481342 |
| N=10000 | Standard Deviation (percent w.r.t swap notional) | 0.021787262 | 0.138853387 | 0.005286481 |

Table 4.7: Mean and Standard Deviation of CVA and its sensitivities (1)

| | | $\partial \text{CVA}/\partial r_{23}$ | $\partial \text{CVA}/\partial r_{13}$ | $\partial \text{CVA}/\partial \sigma$ |
|---|---|---|---|---|
| N=1000 | Mean | 2049.43661 | 2761.77101 | 1024517.742 |
| N=1000 | Standard Deviation | 369.3475393 | 217.6215131 | 70222.60755 |
| N=1000 | Standard Deviation (percent w.r.t swap notional) | 0.036934754 | 0.021762151 | 7.022260755 |
| N=10000 | Mean | 2038.19709 | 2765.13661 | 1023295.59 |
| N=10000 | Standard Deviation | 111.5309582 | 68.35470611 | 22246.94653 |
| N=10000 | Standard Deviation (percent w.r.t swap notional) | 0.011153096 | 0.006835471 | 2.224694653 |

Table 4.8: Mean and Standard Deviation of CVA and its sensitivities (2)

| | | $\partial \text{CVA}/\partial \eta$ | $\partial \text{CVA}/\partial \lambda$ |
|---|---|---|---|
| N=1000 | Mean | -68714.40264 | 143001.493 |
| N=1000 | Standard Deviation | 22330.50033 | 7455.073386 |
| N=1000 | Standard Deviation (percent w.r.t swap notional) | 2.233050033 | 0.745507339 |
| N=10000 | Mean | -68232.5249 | 142937.481 |
| N=10000 | Standard Deviation | 6761.467513 | 2339.201592 |
| N=10000 | Standard Deviation (percent w.r.t swap notional) | 0.676146751 | 0.233920159 |

Table 4.9: Mean and Standard Deviation of CVA and its sensitivities (3)

After transforming the zero rate sensitivities back to par rate sensitivities (See Figure 4.9) these differences appear to become smaller.

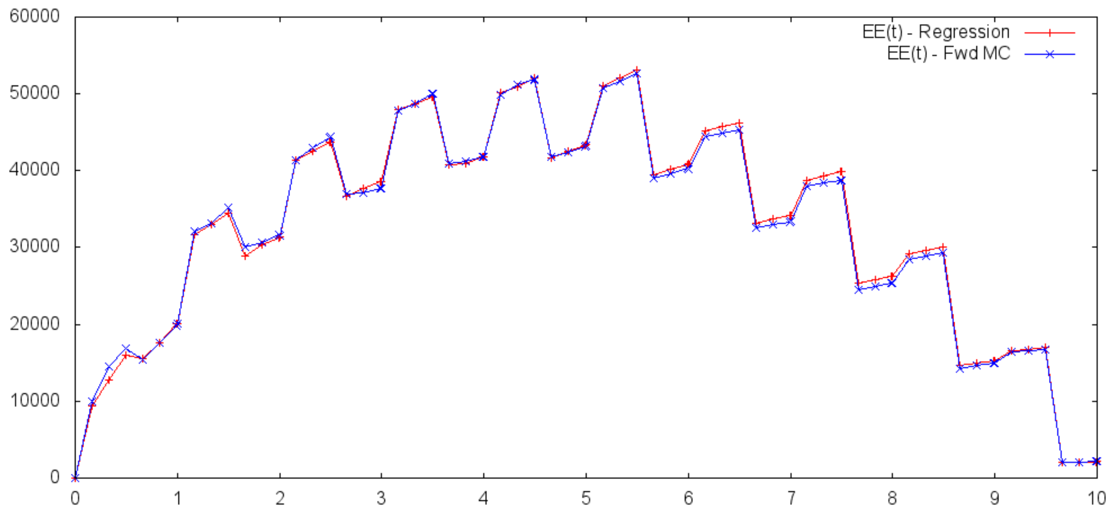

Figure 4.7: Exposure profiles for backward and forward induction algorithms. X-axis: time in years, Y-axis: exposure in units of currency.

Again, we verify the sensitivities of the forward induction algorithm using central differences. The convergence behaviour is similar to that of the backward induction algorithm (See Figure 4.10).

It may be conjectured that the visible differences for the zero rate sensitivities are due to an increased variance of the backward induction based algorithm. To provide evidence for this conjecture we estimate the variance of CVA and the zero rate sensitivites produced by the forward induction algorithm. The result is shown in Table 4.10. Supporting our evidence, we find that the forward induction algorithm exhibits a standard deviation which is about half of that of the backward induction based algorithm.

| | | CVA | $\partial\text{CVA}/\partial R_i$ (average) |
|---|---|---|---|
| N=1000 | Mean | 14181.94855 | 55924.6097 |
| N=1000 | Standard Deviation | 463.4188233 | 2806.406929 |
| N=1000 | Standard Deviation (percent w.r.t swap notional) | 0.046341882 | 0.280640693 |

Table 4.10: Mean and Standard Deviation of CVA and its sensitivities produced by forward induction algorithm.

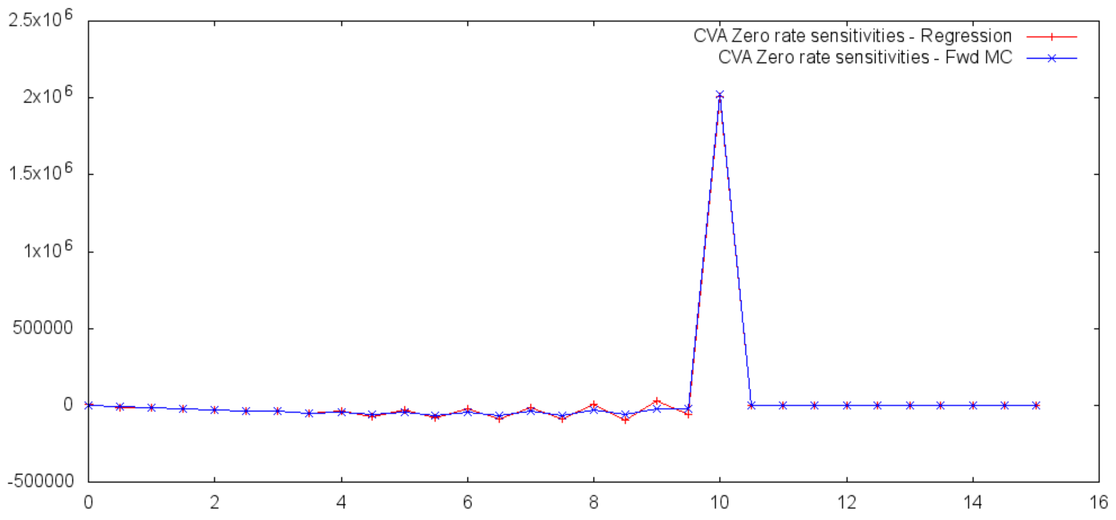

Figure 4.8: Zero rate sensitivities for backward and forward induction algorithms. X-axis: time in years, Y-axis: CVA zero rate sensitivity.

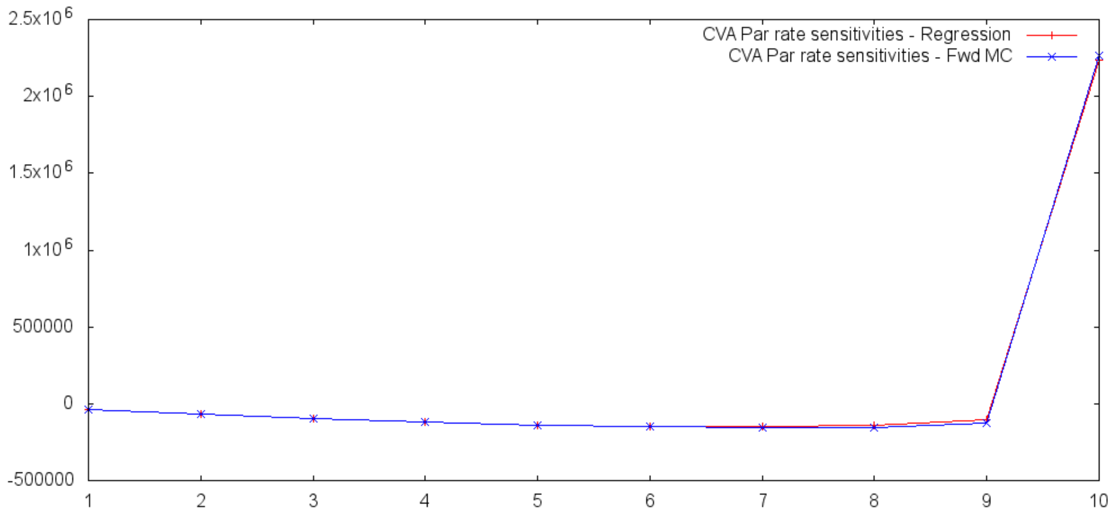

Figure 4.9: Par rate sensitivities for backward and forward induction algorithms. X-axis: time in years, Y-axis: CVA par rate sensitivity.

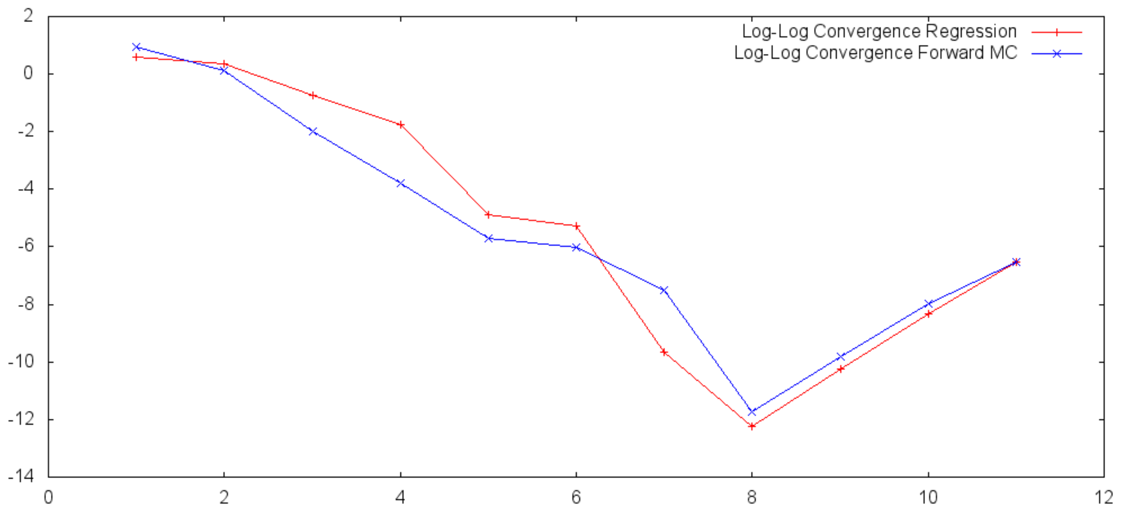

Figure 4.10: Convergence of central difference approximation to AD sensitivities for both algorithms. X-Axis: $-\log_{10}\epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities.

### 4.1.3 AD and Correlation Sensitivities

A central feature of our model is that it accounts for the correlation between interest rate and credit risk. We thus expect the sensitivities $\partial\text{CVA}/\partial\rho_{ij}$ to be non-zero. As we have seen previously, this is indeed the case (See Tables 4.7 and 4.8). Also, for the parameter choice $\rho_{12} = -0.7$, $\rho_{23} = 0.0$, $\rho_{13} = 0.0$, which means essentially that the short rate and hazard rate process are correlated with $\bar{\rho} = 0$, we observed no problems when approximating the correlation sensitivities using finite differences. This may, however, not always be the case. If we choose for instance $\rho_{12} = -0.7$, $\rho_{23} = -0.7419$, $\rho_{13} = 0.045$, giving $\bar{\rho} = 1$ (see [5]), we are in the situation that the correlation matrix $R$ may not be positive semidefinite anymore, when the individual entries are shifted by a small amount $\epsilon$. In turn, the Cholesky decomposition in our implementations fail. This situation is visible in Figure 4.11: for the above choice of parameters we may only be able to perform the Cholesky decomposition starting at a bump size of $\epsilon = 10^{-6}$. Moreover, even at this scale errors are still several orders of magnitude higher when compared to the case of $\bar{\rho} = 0$. This clearly demonstrates that the usage of algorithmic differentiation may in this case give much more accurate figures for the correlation sensitivities.

In particular, we have observed that

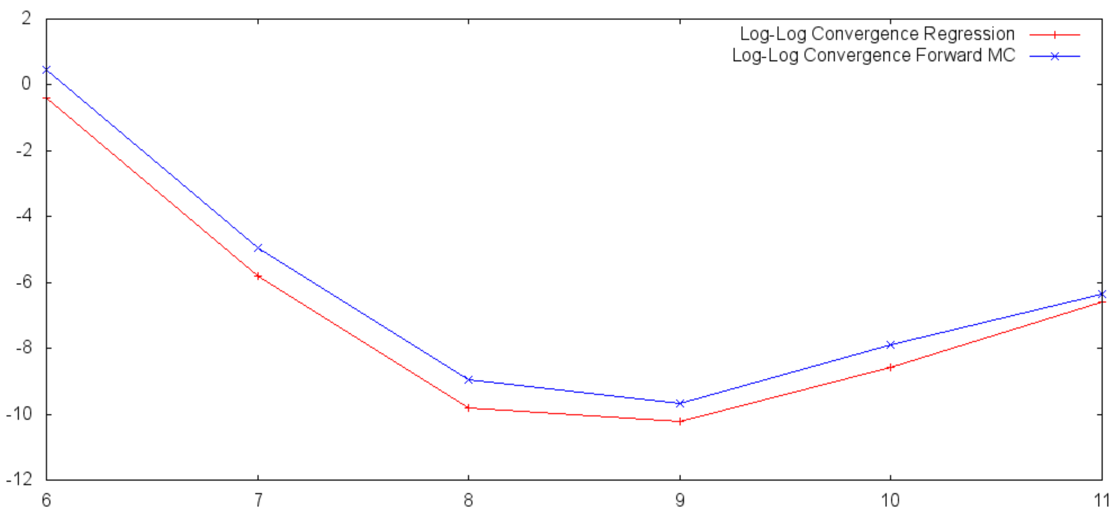

Figure 4.11: Convergence of central difference approximation to AD sensitivities for both algorithms - $\bar{\rho} = 1$. X-Axis: $-\log_{10} \epsilon$, where $\epsilon$ is the bump size. Y-Axis: Logarithm of average relative error across all sensitivities.

### 4.1.4 Performance Measurements

Of course, the most obvious motivation for implementing adjoint AD sensitivities is the potential performance gain compared to the ordinary bumping approach. In the case of a single interest rate swap we report for this work the following perfomance measurements as displayed in Table 4.11. In order to increase the number of sensitivities in our example calculations we have added more points on to the zero rate curve.

We may summarize our observations in the following points:

1. The backward algorithm outperforms the forward algorithm in general by about one order of magnitude. Although this is expected to some extent, we note, however, that our implementation of the forward algorithm is by no means optimal. Further work in this direction could also lead to significantly lower differences in performance.

2. For the backward algorithm we observe another performance gain of one order of magnitude by using AAD instead of finite differences for the sensitivity computation.

3. As expected, the number of sensitivities computed has little or no impact on the execution times of the dco based implementation.

| N | Algorithm | Sensitivities | Number of sensitivities | Execution time in secs. |
|---|---|---|---|---|
| 1000 | Backward | dco | 37 | 0.7 |
| 1000 | Forward | dco | 37 | 11 |
| 1000 | Backward | bumping | 37 | 11 |
| 1000 | Forward | bumping | 37 | 81 |
| 10000 | Backward | dco | 37 | 7 |
| 10000 | Forward | dco | 37 | 1879 |
| 10000 | Backward | bumping | 37 | 114 |
| 10000 | Forward | bumping | 37 | 837 |
| 1000 | Backward | dco | 67 | 0.7 |
| 1000 | Forward | dco | 67 | 11 |
| 1000 | Backward | bumping | 67 | 20 |
| 1000 | Forward | bumping | 67 | 152 |
| 10000 | Backward | dco | 67 | 7 |
| 10000 | Forward | dco | 67 | 1836 |
| 10000 | Backward | bumping | 67 | 215 |
| 10000 | Forward | bumping | 67 | 1567 |

Table 4.11: Execution times for CVA sensitivity calculations - single interest rate swap.

4. For the forward algorithm we observe this kind of performance gain only for $N = 1000$. Surprisingly, for $N = 1000$ the finite difference version of the code seems to outperform the dco version. We conjecture that this is due to a suboptimal implementation of the checkpointing technique in our code.

Finally we note that in our implementations, memory consumption of the AD disabled versions was observed to be about ten times lower when compared to the AD enabled versions implemented using dco. This is a typical situation encountered when using adjoint algorithmic differentiation.

## 4.2 Portfolio of IR Swaps

So far, we have only applied our algorithms to an individual trade. In practice, however, CVA has to be computed for a whole "netting set" of trades that all belong to the same counterparty. To see how our algorithm performs in this case we apply it to a portfolio of 10 swap trades. Each trade has the same basic characteristics as the single interest rate swap used in the previous section. The only differences are that we choose our swaps such that $i$-th swap matures after $i$ years and uses as fixed rate the par swap rate for a maturity of $i$ years.

The exposure profile of the swap portfolio is displayed in Figure 4.12, and plots of the zero rate and par rate sensitivities can be seen in Figures 4.13 and 4.14.

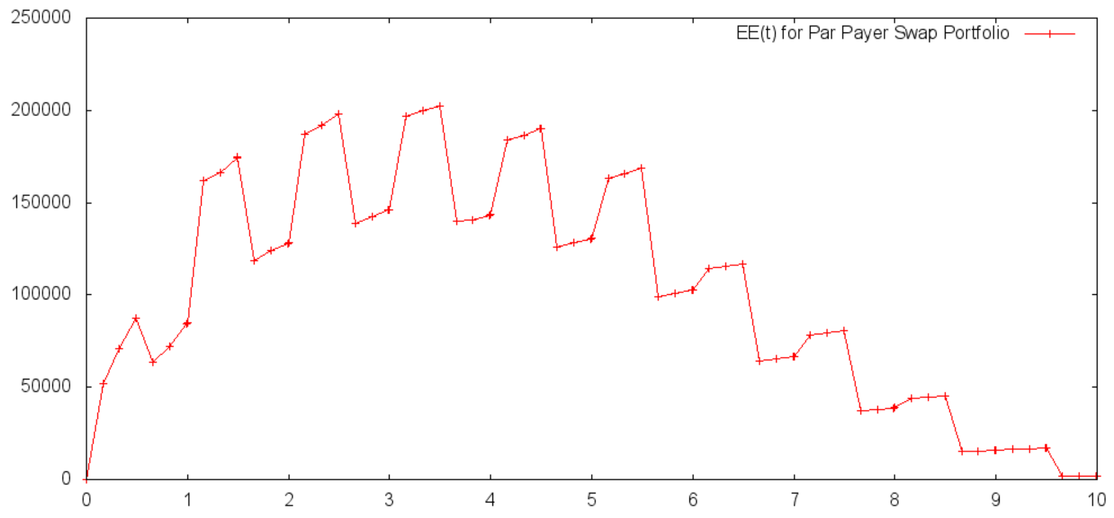

Figure 4.12: Exposure profile for a portfolio of 10 payer swaps X-axis: time in years, Y-axis: exposure in units of currency.

We notice that the exposure profile looks less symmetric when compared to that of the single interest rate swap. This comes to no surprise, since the individual swaps all have maturities less than or equal to 10 years.

Similarly, we can observe that for the swap portfolio the absolute values of the zero rate sensitivities corresponding to times smaller than 10 years have increased (Figure 4.13 ). Similarly, we can see more sensitivity on par swap rates with lower maturities (Figure 4.14 ).

Finally, we report performance measurements for the case of the swap portfolio in Table 4.12. We notice that the backward induction seems to scale better with an increasing number of swaps.

| N | Algorithm | Sensitivities | Number of sensitivities | Execution time in secs. |
|---|---|---|---|---|
| 1000 | Backward | dco | 37 | 0.7 |
| 1000 | Forward | dco | 37 | 20 |
| 10000 | Backward | dco | 37 | 7 |
| 10000 | Forward | dco | 37 | 1916 |

Table 4.12: Execution times for CVA sensitivity calculations - portfolio of 10 interest rate swaps.

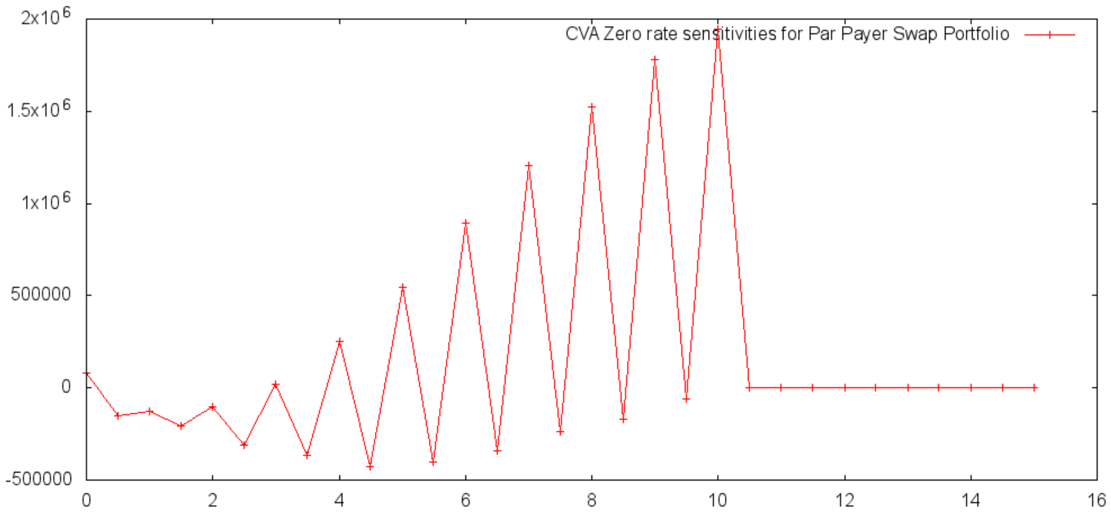

Figure 4.13: Zero rate sensitivities for a portfolio of 10 payer swaps. X-axis: time in years, Y-axis: CVA zero rate sensitivity.

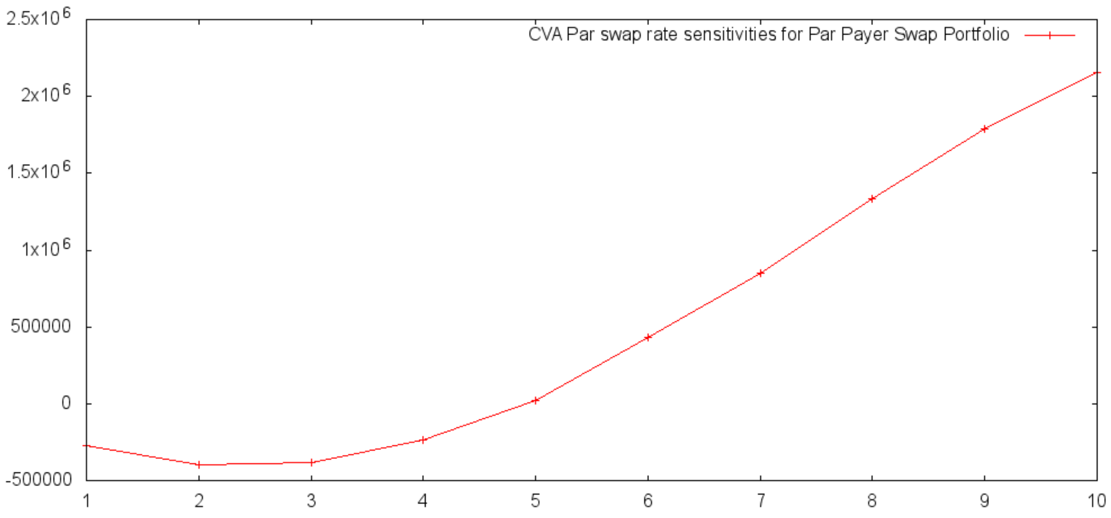

Figure 4.14: Par swap rate sensitivities for a portfolio of 10 payer swaps. X-axis: time in years, Y-axis: CVA par rate sensitivity.

# Chapter 5

# Summary and Conclusion

At this point it should have become apparent that a CVA computation, even for seemingly simple trade portfolios und underlying models, can become a rather challenging task. Implementing CVA sensitivities on top of an existing code becomes even more difficult, when faced with constraints on performance in general and the number of sensitivities in particular.

In this thesis we have presented two algorithms that can be used to implement CVA for a combined interest rate and credit risk model exhibiting the phenomenon of wrong way risk. The first algorithm is based on a backward induction procedure using linear regression in order to estimate conditional expectations (American Monte Carlo). This algorithm has been the main object of study in this thesis. For comparison we have presented another algorithm, which represents the traditional approach to CVA and is based on a pure forward Monte Carlo simulation, similar to the pricing of a path dependent European option. We have implemented both of these algorithms in C++ and computed their sensitivities using the AD tool dco-c++. All sensitivities were computed using the backward (adjoint) mode of AD and have been validated against ordinary finite difference sensitivities.

The main results of this work can be summarized as follows:

1. We have shown that an American Monte Carlo based CVA computation may be equipped with AAD sensitivities, if one has all of the underlying source code available. However, the concrete technical implementation is highly non-trivial and requires extensive usage of the checkpointing technique. Yet, the performance gain for the sensitivity computation (compared to finite differences) is in the order of one magnitude.

2. Comparing the forward and backward algorithms we have found that the latter clearly outperforms the former in terms of speed. The price for this is an increase

in variance.

3. The implementation of the backward algorithm scaled well w.r.t. the number of sensitivities, as well as trades and MC paths. Our implementation of the forward algorithm did, however, not scale well w.r.t. the number of Monte Carlo paths: an increase in the number of paths did not lead to a linear, but rather quadratic increase in computation time, most likely due to the extra overhead caused by the suboptimal checkpointing strategy chosen for the forward algorithm.

4. Usage of AAD is especially advatangeous for correlation sensitivities.

Concluding this thesis, we mention possibilities for future extension of this work:

1. Improve details of the algorithms, such as SDE discretization schemes (use an implicit scheme for the square root process) or regression basis functions (experiment with non-polynomial basis functions).

2. Find an optimal checkpointing strategies for both algorithms with the help of the algorithm Revolve [16].

3. Apply the algorithm to more exotic products, e.g. characterized by American or Bermudan exercise features or priced using hybrid models. A straightforward extension with little adjustments to the implementation could for example consist of the CVA sensitivity computation for a Bermudan swaption.

4. Extend the implementation to take into account collateralization: this would mean that one would have to model an additional stochastic process which describes the amount of collateral received or posted during the lifetime of the trades being considered. An interesting point here would be to compute the sensitivity of CVA with respect to the margin period of risk, i.e. the frequency at which the two parties involved in the trade post or receive collateral.

5. Use AAD for calibration of the underlying models: So far, we have set aside the question of how to calibrate the underlying model(s) in this work. However, as almost all calibration routines amount to numerical optimization or root finding, this appears to be a well fitting application for AAD sensitivities.

6. Use AAD for second order derivatives: In the case of CVA sensitivites this would be of particular interest, especially because it would allow for a systematic study of so-called "cross-gamma" sensitivities, i.e. second order mixed interest rate and credit sensitivities.

# References

[1] Alexandre Antonov, Serguei Issakov, and Serguei Mechkov. Algorithmic exposure and CVA for exotic derivatives. *Available at SSRN 1960773*, 2011.

[2] Damiano Brigo and Aurelien Alfonsi. Credit default swap calibration and derivatives pricing with the SSRD stochastic intensity model. *Finance and Stochastics*, 9(1):29–42, 2005.

[3] Damiano Brigo and Fabio Mercurio. *Interest rate models-theory and practice: with smile, inflation and credit*. Springer Science & Business Media, 2007.

[4] Damiano Brigo, Massimo Morini, and Andrea Pallavicini. *Counterparty Credit Risk, Collateral and Funding: With Pricing Cases For All Asset Classes*. John Wiley & Sons, 2013.

[5] Damiano Brigo and Andrea Pallavicini. Counterparty risk and contingent CDS valuation under correlation between interest-rates and default. *Available at SSRN 926067*, 2006.

[6] Damiano Brigo, Andrea Pallavicini, and Vasileios Papatheodorou. Bilateral counterparty risk valuation for interest-rate products: impact of volatilities and correlations. Technical report, 2010.

[7] Luca Capriotti and Shinghoi Jacky Lee. Adjoint credit risk management. *Available at SSRN 2342573*, 2013.

[8] Luca Capriotti, Shinghoi Jacky Lee, and Matthew Peacock. Real time counterparty credit risk management in Monte Carlo. *Risk Magazine*, May, 2011.

[9] Giuseppe Castellacci. Bootstrapping credit curves from CDS spread curves. *Available at SSRN 2042177*, 2008.

[10] Giovanni Cesari, John Aquilina, Niels Charpillon, Zlatko Filipovic, Gordon Lee, and Ion Manda. *Modelling, pricing, and hedging counterparty credit exposure: A technical guide.* Springer Science & Business Media, 2009.

[11] Hans-Peter Deutsch. *Derivate und interne Modelle.* Schäffer-Poeschel, 2004.

[12] Luigi Ballabio Ferdinando Ametrano. QuantLib. `http://quantlib.org/`. [Online; last accessed 02-September-2015].

[13] Samim Ghamami and Lisa R Goldberg. Stochastic intensity models of wrong way risk: Wrong way CVA need not exceed independent CVA. *The Journal of Derivatives*, 21(3):24–35.

[14] Mike Giles and Paul Glasserman. Smoking adjoints: Fast Monte Carlo greeks. *Risk*, 19(1):88–92, 2006.

[15] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2003.

[16] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

[17] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, 2008.

[18] Chris Kenyon and Andrew Green. Efficient XVA management: Pricing, hedging, and allocation using trade-level regression and global conditioning. *arXiv preprint arXiv:1412.5332*, 2014.

[19] Michael Kohler. A review on regression-based Monte Carlo methods for pricing American options. In *Recent Developments in Applied Probability and Statistics*, pages 37–58. Springer, 2010.

[20] David Lando. On Cox processes and credit risky securities. *Review of Derivatives Research*, 2(2-3):99–120, 1998.

[21] Matthias Leclerc, Qian Liang, and Ingo Schneider. Fast Monte Carlo bermudan greeks. *Risk magazine*, 2009.

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

[23] Uwe Naumann. *The art of differentiating computer programs: an introduction to algorithmic differentiation*, volume 24. SIAM, 2012.

[24] Uwe Naumann, Klaus Leppkes, and Johannes Lotz. dco/c++ user guide. Technical Report AIB-2014-03, RWTH Aachen, January 2014.

[25] Andreas Pfadler. On the impact of collateralization on derivative pricing and curve construction: a review of recent results. Submitted as Assignment to Module 5 in the part-time MSc program in mathematical finance at the University of Oxford, 2013.

[26] Andreas Pfadler. Var sensitivity estimation using automatic differentation. Submitted as Assignment to Module 7 in the part-time MSc program in mathematical finance at the University of Oxford, 2014.

[27] Vytautas Savickas, Norbert Hari, Tim Wood, and Drona Kandhai. Super fast greeks: An application to counterparty valuation adjustments. *Wilmott*, 2014(69):76–81, 2014.

[28] Robert Schöftner and WU Wien. On the estimation of credit exposures using regression-based Monte Carlo simulation. *Journal of Credit Risk*, 4(4):37–62, 2008.

[29] Philip Wallstedt. C++ Code for Matrix Decompositions. `http://www.sci.utah.edu/~wallstedt/LU.htm`. [Online; last accessed 05-April-2015].