Automatic Backward Differentiation for American Monte-Carlo Algorithms - ADD for Conditional Expectations and Indicator Functions

Christian P. Fries
email@christian-fries.de

June 27, 2017

Version 0.8.3
(Extended Version)

Abstract

In this note we derive a modified backward automatic differentiation (a.k.a. adjoint automatic differentiation, adjoint algorithmic differentiation) for algorithms containing conditional expectation operators and/or indicator functions. Bermudan option and xVA valuation are prototypical examples. We consider the Bermudan product valuation, but the method is applicable in full generality.

Featuring a clean and simple implementation, the method improves accuracy and performance.

For conditional expectation operators it offers the ability to use different estimators in the valuation and the differentiation.

For the indicator function, the method allows to use "per-operator"-differentiation of the indicator function, enabling an accurate treatment of each individual exercise boundary - which is not possible in a classic finite difference applied to the Bermudan valuation.

Disclaimer: The views expressed in this work are the personal views of the authors and do not necessarily reflect the views or policies of current or previous employers.
Feedback welcomed at email@christian-fries.de.

Electronic copy available at: https://ssrn.com/abstract=3000822

Contents

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

1 Introduction

The Monte-Carlo valuation of Bermudan-like products or any valuation requiring the calculation of conditional expectations of future values (CVA and MVA are common examples) is a non-trivial problem. The standard solution is to use regression methods to estimate the conditional expectation, often referred to as "American Monte-Carlo".

Another numerical intensive problem is the calculation of sensitivities (a.k.a. Greeks), i.e., partial derivatives with respect to model parameters. This problem can be solved efficiently by adjoint automatic differentiation, c.f. [12].

If the valuation algorithm involves a conditional expectation operator the calculation of a sensitivities then seemingly requires the differentiation of the conditional expectation estimator. It is known that in some cases, the differentiation of the conditional expectation can be omitted, e.g. if the conditional expectation is the input of an optimal exercise criteria *and* the sensitivity is a first order sensitivity, see [13]. However, in general, the differentiation cannot be omitted, see Section 6 for an examples.

Since the exercise criteria is essentially an indicator function, the differentiation of the indicator function is another numerically demanding problem.

We consider the application of *adjoint automatic differentiation* to calculate the sensitivities of the valuation of a general product involving a conditional expectation operator and indicator functions. The automatic differentiation of valuations involving conditional expectation has been discussed in, e.g., [1, 2, 3, 5].

To understand the different approaches it is important to understand that automatic differentiation comes in essentially two different flavours: the automatic differentiation can operate in forward mode (forward AD, sometimes called AD), where the derivatives are propagated forward from input (parameters) to results (values), along-side the valuation; or the automatic differentiation can operate in the backward mode (backward AD, also called "adjoint AD", "AAD"), where one propagates derivatives backward from values to input. The numerical performance of the forward mode scales with the number of parameters and is roughly independent of the number of results, while the performance of the backward mode scales with the number of results and is roughly independent of the number of parameters.

Adjoint algorithmic (backward mode) differentiation is the method of choice when sensitivity have to be calculated for a single (or few) results depending on many parameters. A striking example would be the valuation of an MVA from ISDA-SIMM initial margins, see [10].

Applying an automatic differentiation and in particular an adjoint automatic differentiation to a valuation algorithm containing a conditional expectation reveals some issues:

While in a Monte-Carlo simulation most operators are pathwise and differentiation can be applied on a path-by-path basis, the conditional expectation operator is non-pathwise, aggregating information of adjoint future paths. A brute force

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

application of (A)AD to the conditional expectation regression will differentiate the regression basis functions (see [1, 5]).

However, differentiating the regression basis function can always be avoided, as long as the filtration does not depend on the model parameters.[1] Then the differentiation of a conditional expectation is the conditional expectation of the differentiation

$$\frac{\mathrm{d}}{\mathrm{d}x}\mathrm{E}\left(Z\,|\mathcal{F}_t\right) \;=\; \mathrm{E}\left(\frac{\mathrm{d}}{\mathrm{d}x}Z\,|\mathcal{F}_t\right). \tag{1}$$

This result offers another striking optimization: we may just check if $\frac{\mathrm{d}}{\mathrm{d}x}Z$ is an $\mathcal{F}_t$-measurable random variable. If that is the case, we can omit the outer conditional expectation operator on the right hand side. Our implementation in [6] contains an automatic tracking of the measurability of random variables, which enables us to drop the conditional expectation whenever possible.[2]

If the conditional expectation operator cannot be omitted (for examples see below), the application of (1) still imposes a difficulty for the implementation of adjoint automatic differentiation: since the AAD operates *backward* through the operators we cannot apply (1) during the backward propagation, since we have to calculate the inner derivative $\frac{\mathrm{d}}{\mathrm{d}x}Z$ first, which is only available after the backward sweep has been completed.

This issue can be solved by splitting the algorithm into two independent backward differentiation algorithms, which then have to be combined, see Figures 1.

In [2] the authors present a method called "BD", "backward differentiation", which calculates the derivatives of the arguments of conditional expectations (the so called continuation values) along-side the valuation. This algorithm has the advantage that the derivatives can be calculated in a single sweep. To do so they directly modify the valuation code. The name "backward differentiation" is somewhat misleading: the BD-algorithm in [2] is propagating derivatives forward (namely alongside the valuation algorithm). It is just that the algorithm itself goes backward over the time-steps since this is the way a Bermudan callable is valued. While this allows to calculate the derivative in a single sweep, it comes at the price that the performance and complexity of the algorithms scales with the number of parameters (see N_θ in Section 7.4 of [2]) - as it is typical for a forward mode AD.

In contrast to this we will show how to efficiently apply a true backward mode differentiation (i.e., AAD backward differentiation) to a Bermudan callable - without modification of the original valuation algorithm. The trick that enables us to treat the conditional expectation in a single backward sweep relies on the following observations:

[1] This is the case in supposedly all practical Monte-Carlo applications: the filtration is represented by the generated random numbersand differentiating the filtration would correspond to differentiating the random number generator.

[2] In [2] the authors illustrate that for in their algorithm the conditional expectation can be avoided given the absence of path-dependency, but this condition may be hard to check.

Electronic copy available at: https://ssrn.com/abstract=3000822

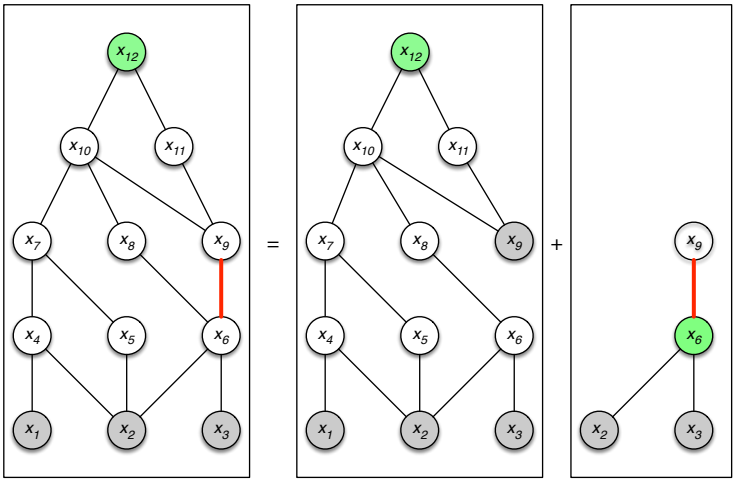

Figure 1: Operator tree where x_9 is a conditional expectation of x_6. The differentiation algorithm can be decomposed in the differentiation of x_{12} considering x_9 as an independent variable and into the differentiation of x_9.

- The application of an automatic differentiation to an algorithm containing a conditional expectation results in a linear combination of conditional expectation operators (due to (1)).
- The valuation is an expectation, that is, the last operator is an expectation operator.[3]
- For any two random variables A, B we have that

$$\mathrm{E}\left(A \cdot \mathrm{E}\left(B \,|\mathcal{F}_t\right)\right) \;=\; \mathrm{E}\left(\mathrm{E}\left(A \,|\mathcal{F}_t\right) \cdot B\right). \tag{2}$$

Equation (2) allows us to apply the conditional expectation to the adjoint differential, which is available during the backward differentiation when the algorithm reaches the conditional expectation operator.

It remains to show that (2) and (1) can be applied in an algorithm featuring multiple, possibly iterative conditional expectation, as it is common in a Bermudan backward algorithm.

1.1 Setup and Notation

Given a filtered probability space $(\Omega, \mathbb{Q}, \{\mathcal{F}_t\})$ we consider a Monte-Carlo simulation of a (time-discretised) stochastic process, i.e., we simulate sample path ω_i of sequences of random variables. Assuming that the drawings are uniform with

[3] It is actually sufficient that there is an outer operator which is an expectation or a conditional expectation on a coarser filtration.

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

respect to the measure $\mathbb{Q}$, this allows to approximate the unconditional expectation $\mathrm{E}^{\mathbb{Q}}(X)$ of a random variable X via

$$\mathrm{E}^{\mathbb{Q}}(X) \approx \frac{1}{n} \sum_{k=0}^{n-1} X(\omega_k).$$

However, the calculation of conditional expectations $\mathrm{E}^{\mathbb{Q}}(X|\mathcal{F}_{T_i})$ is more involved, since the Monte-Carlo simulation does not provide a discretization of the filtration $\{\mathcal{F}_t\}$. For the estimation of conditional expectation numerical approximations like least-square regressions can be used, c.f. [8].

Let z denote a given model parameter used in the generation of the Monte-Carlo simulation. Given a valuation algorithm which calculates the unconditional expectation $\mathrm{E}^{\mathbb{Q}}(V)$ of a random variable V, where the calculation of V involves one or more conditional expectations, we consider the calculation of the derivative $\frac{d}{dz}\mathrm{E}^{\mathbb{Q}}(V)$.

Electronic copy available at: https://ssrn.com/abstract=3000822

2 American Monte-Carlo and Bermudan Option Valuation

We shortly give the definition of the backward algorithm. For details see [8], we use a similar notation here.

Let $0 = T_0 < T_1 < \ldots < T_n$ denote a given time discretization. Let V_i, $i = 1, \ldots, n$ denote the time-T_i numéraire relative values of given underlyings. Here V_i are $\mathcal{F}_{T_i}$-measurable random variables. Then let U_i be defined as follows:

$$\begin{aligned} U_{n+1} &:= 0 \\ U_i &:= B_i \left(\mathrm{E}^{\mathbb{Q}} \left(U_{i+1} \,|\mathcal{F}_{T_i} \right), U_{i+1}, V_i \right), \end{aligned} \tag{3}$$

where B_i are arbitrary function.

2.1 Bermudan Option Valuation

For a Bermudan option B_i is given by the optimal exercise criteria, i.e., $B_i(x,u,v) := G(x-v,u,v)$ with

$$G(y,u,v) := \left\{ \begin{array}{ll} u & \text{if y>0} \\ v & \text{else} \end{array} \right\}.$$

This defines a backward induction $i = n, n-1, \ldots, 1$ for U_i. For a Bermudan option we have that the *unconditional* expectation

$$\mathrm{E}^{\mathbb{Q}} \left(U_1 \right)$$

is the (risk) neutral (numéraire relative) value of the Bermudan option with exercise dates $T_1 < \ldots < T_n$ and exercise values V_i.

2.2 Bermudan Digital Option Valuation

For a Bermudan option the exercise is optimal. This allows for a popular "trick" to ignore the first order derivative of $G(y,u,v)$ with respect to y. Hence, for a Bermudan option it is not necessary to differentiate B_i with respect to x (see [13]).

However, this is just a special property of the function G and holds for first order derivatives only. For example, the trick cannot be applied to a Bermudan *digital option*, i.e., $B_i(x,u,v) := H(x-v,u,v)$ with

$$H(y,u,v) := \left\{ \begin{array}{ll} u & \text{if y>0} \\ 1 & \text{else} \end{array} \right\}.$$

We will use this product as a test case of the methodology in Section 6.

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

Remarks: The Bermudan *digital* option shows a dependency on the first order derivative with respect to the indicator condition y, since the expectation of the two outcomes u and 1 differs, i.e., we have a jump at $y = 0$. For the classic Bermudan option we have that the expectation of u and v agree at $y = 0$.

The (automatic) differentiation of a jump may appear as an issue, but it is possible to solve this elegantly in an stochastic automatic differentiation [9], see Section 4.

3 Automatic Differentiation of the American-Monte-Carlo Backward Algorithm

Let z denote an arbitrary model parameter. We assume that the underlying values V_i depend on z.[4]

We differentiate the Backward algorithm (3): For $i = 1, \dots, n$ we have

$$\frac{\mathrm{d}}{\mathrm{d}z} U_i \;=\; \frac{\mathrm{d}}{\mathrm{d}z} B_i \left(\mathrm{E}^{\mathbb{Q}} \left(U_{i+1} \,|\mathcal{F}_{T_i} \right), U_{i+1}, V_i \right).$$

Applying the chain rule to $B_i(x, u, v)$ with $x = \mathrm{E}^{\mathbb{Q}} \left(U_{i+1} \,|\mathcal{F}_{T_i} \right)$, $u = U_{i+1}$, $v = V_i$ and writing $B_i = B_i \left(\mathrm{E}^{\mathbb{Q}} \left(U_{i+1} \,|\mathcal{F}_{T_i} \right), U_{i+1}, V_i \right)$ to avoid the lengthy argument list we get

$$\frac{\mathrm{d}}{\mathrm{d}z} U_i \;=\; \frac{\mathrm{d}B_i}{\mathrm{d}x} \cdot \left(\frac{\mathrm{d}}{\mathrm{d}z} \mathrm{E}^{\mathbb{Q}} \left(U_{i+1} \,|\mathcal{F}_{T_i} \right) \right) + \frac{\mathrm{d}B_i}{\mathrm{d}u} \cdot \frac{\mathrm{d}U_{i+1}}{\mathrm{d}z} + \frac{\mathrm{d}B_i}{\mathrm{d}v} \cdot \frac{\mathrm{d}V_i}{\mathrm{d}z}$$

and using $\frac{\mathrm{d}}{\mathrm{d}z} \mathrm{E}^{\mathbb{Q}} \left(U_i \right) \;=\; \mathrm{E}^{\mathbb{Q}} \left(\frac{\mathrm{d}}{\mathrm{d}z} U_i \right)$

$$\frac{\mathrm{d}}{\mathrm{d}z} U_i \;=\; \frac{\mathrm{d}B_i}{\mathrm{d}x} \cdot \mathrm{E}^{\mathbb{Q}} \left(\frac{\mathrm{dU_{i+1}}}{\mathrm{d}z} \,|\mathcal{F}_{T_i} \right) + \frac{\mathrm{d}B_i}{\mathrm{d}u} \cdot \frac{\mathrm{d}U_{i+1}}{\mathrm{d}z} + \frac{\mathrm{d}B_i}{\mathrm{d}v} \cdot \frac{\mathrm{d}V_i}{\mathrm{d}z}. \tag{4}$$

3.1 Forward Differentiation

Applying this relation iteratively (plugging the expression of $\frac{\mathrm{d}U_{j+1}}{\mathrm{d}z}$ into the equation of $\frac{\mathrm{d}U_i}{\mathrm{d}z}$ for $j = i, \dots, n-1$) gives

$$\frac{\mathrm{d}}{\mathrm{d}z} U_i \;=\; \sum_{j=i}^{n} \left(\left(\prod_{k=i}^{j-1} \frac{\mathrm{d}B_k}{\mathrm{d}u} \right) \frac{\mathrm{d}B_j}{\mathrm{d}x} \cdot \mathrm{E}^{\mathbb{Q}} \left(\frac{\mathrm{d}U_{j+1}}{\mathrm{d}z} \,|\mathcal{F}_{T_j} \right) + \left(\prod_{k=i}^{j-1} \frac{\mathrm{d}B_k}{\mathrm{d}u} \right) \frac{\mathrm{d}B_j}{\mathrm{d}v} \cdot \frac{\mathrm{d}V_j}{\mathrm{d}z} \right). \tag{5}$$

[4] Think of z being an initial value of the interest rate curve (e.g., calculating delta in an LMM) or a volatility parameter (calculating vega).

Electronic copy available at: https://ssrn.com/abstract=3000822

Indeed, plugging (4) for $i+1$ into the right hand side of (4) for i we get

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}z}U_i &= \frac{\mathrm{d}B_i}{\mathrm{d}x}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{dU_{i+1}}}{\mathrm{d}z}\,|\mathcal{F}_{T_i}\right)+\frac{\mathrm{d}B_i}{\mathrm{d}v}\cdot\frac{\mathrm{d}V_i}{\mathrm{d}z}\\ &\quad+\frac{\mathrm{d}B_i}{\mathrm{d}u}\cdot\left[\frac{\mathrm{d}B_{i+1}}{\mathrm{d}x}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{dU_{i+2}}}{\mathrm{d}z}\,|\mathcal{F}_{T_{i+1}}\right)+\frac{\mathrm{d}B_{i+1}}{\mathrm{d}u}\cdot\frac{\mathrm{d}U_{i+2}}{\mathrm{d}z}+\frac{\mathrm{d}B_{i+1}}{\mathrm{d}v}\cdot\frac{\mathrm{d}V_{i+1}}{\mathrm{d}z}\right]\\ &=\sum_{j=i}^{i+1}\left(\left(\prod_{k=i}^{j-1}\frac{\mathrm{d}B_k}{\mathrm{d}u}\right)\frac{\mathrm{d}B_j}{\mathrm{d}x}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}U_{j+1}}{\mathrm{d}z}\,|\mathcal{F}_{T_j}\right)+\left(\prod_{k=i}^{j-1}\frac{\mathrm{d}B_k}{\mathrm{d}u}\right)\frac{\mathrm{d}B_j}{\mathrm{d}v}\cdot\frac{\mathrm{d}V_j}{\mathrm{d}z}\right)+\left(\prod_{k=i}^{i+1}\frac{\mathrm{d}B_k}{\mathrm{d}u}\right)\frac{\mathrm{d}U_{i+2}}{\mathrm{d}z}.\end{aligned}$$

Repeating this iteration we get $i+1\to n$ on the right hand side. Using $\frac{\mathrm{d}U_{n+1}}{\mathrm{d}z}=0$ gives (5).

To shorten notation, let

$$A_{i,j}=\left(\prod_{k=i}^{j-1}\frac{\mathrm{d}B_k}{\mathrm{d}u}\right)\frac{\mathrm{d}B_j}{\mathrm{d}x}\qquad C_{i,j}=\left(\prod_{k=i}^{j-1}\frac{\mathrm{d}B_k}{\mathrm{d}u}\right)\frac{\mathrm{d}B_j}{\mathrm{d}v}$$

so that (5) becomes

$$\frac{\mathrm{d}U_i}{\mathrm{d}z}=\sum_{j=i}^{n}\left(A_{i,j}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}U_{j+1}}{\mathrm{d}z}\,|\mathcal{F}_{T_j}\right)+C_{i,j}\cdot\frac{\mathrm{d}V_j}{\mathrm{d}z}\right).\tag{6}$$

This last equation would be natural in a forward (automatic) differentiation, since we calculate $0=\frac{\mathrm{d}U_{n+1}}{\mathrm{d}z}$, $\frac{\mathrm{d}U_n}{\mathrm{d}z}$, $\frac{\mathrm{d}U_{n-1}}{\mathrm{d}z}$, …, together with $\frac{\mathrm{d}V_j}{\mathrm{d}z}$ forwardly. Note that forward here refers to the order of the operations in the algorithms, which runs backward over the indices j.

3.2 Backward Differentiation

For the application of a backward (adjoint) automatic differentiation equation (6) would require a mixture of backward differentiation for that we first calculate $\frac{\mathrm{d}U_j}{\mathrm{d}z}$ then followed by a forward application of (6). However, we can calculate the derivative in a single backward differentiation sweep:

we start with (4) for $i=1$. Since we are only interest in $\frac{\mathrm{d}}{\mathrm{d}z}\mathrm{E}^{\mathbb{Q}}(U_1)=\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}}{\mathrm{d}z}U_1\right)$, we can take expectation and get

$$\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}}{\mathrm{d}z}U_1\right)=\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}B_1}{\mathrm{d}x}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{dU_2}}{\mathrm{d}z}\,|\mathcal{F}_{T_i}\right)+\frac{\mathrm{d}B_1}{\mathrm{d}u}\cdot\frac{\mathrm{d}U_2}{\mathrm{d}z}+\frac{\mathrm{d}B_1}{\mathrm{d}v}\cdot\frac{\mathrm{d}V_1}{\mathrm{d}z}\right).$$

Now we may use

$$\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}B_1}{\mathrm{d}x}\cdot\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{dU_2}}{\mathrm{d}z}\,|\mathcal{F}_{T_1}\right)\right)=\mathrm{E}^{\mathbb{Q}}\left(\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}B_1}{\mathrm{d}x}\,|\mathcal{F}_{T_1}\right)\cdot\frac{\mathrm{d}U_2}{\mathrm{d}z}\right)\tag{7}$$

to get

$$\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}}{\mathrm{d}z}U_1\right)=\mathrm{E}^{\mathbb{Q}}\left(\left(\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}B_1}{\mathrm{d}x}\,|\mathcal{F}_{T_1}\right)+\frac{\mathrm{d}B_1}{\mathrm{d}u}\right)\cdot\frac{\mathrm{d}U_2}{\mathrm{d}z}+\frac{\mathrm{d}B_1}{\mathrm{d}v}\cdot\frac{\mathrm{d}V_1}{\mathrm{d}z}\right).\tag{8}$$

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

Plugging (4) into (8) and repeating the previous argument for $i = 2, \ldots, k-1$ we get iteratively the forward equation

$$\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}}{\mathrm{d}z}U_1\right) = \mathrm{E}^{\mathbb{Q}}\left(A^*_{1,k}\cdot\frac{\mathrm{d}U_{k+1}}{\mathrm{d}z} + \sum_{j=1}^{k} C^*_{1,j}\cdot\frac{\mathrm{d}V_j}{\mathrm{d}z}\right).$$

where

$$\begin{aligned} A^*_{1,i} &= \mathrm{E}^{\mathbb{Q}}\left(A^*_{1,i-1}\frac{\mathrm{d}B_i}{\mathrm{d}x}\,|\mathcal{F}_{T_i}\right) + A^*_{1,i-1}\frac{\mathrm{d}B_i}{\mathrm{d}u} \\ C^*_{1,i} &= A^*_{1,i-1}\frac{\mathrm{d}B_i}{\mathrm{d}v} \\ A^*_{1,0} &= 1 \end{aligned}$$

Using $i = n$ we have with $U_{n+1} = 0$ that

$$\mathrm{E}^{\mathbb{Q}}\left(\frac{\mathrm{d}}{\mathrm{d}z}U_1\right) = \mathrm{E}^{\mathbb{Q}}\left(\sum_{j=1}^{n} C^*_{1,j}\cdot\frac{\mathrm{d}V_j}{\mathrm{d}z}\right).$$

The recursive definitions of $A^*_{1,i}$, $C^*_{1,i}$ have an intuitive interpretation in a backward (automatic) differentiation algorithm: in the algorithm the conditional expectation operator on U_{i+1} is replaced by taking the conditional expectation of the adjoint differential.

Remark: The important improvement (which highly simplifies the implementation) is that the calculation of the coefficients A^*, C^* does not involve the direct (automatic) differentiation of the conditional expectation operator. In addition, we can calculate the coefficients in a single backwards differentiation sweep, due to the application of (7).

3.3 Automatic Tracking of Measurability

We can augment our implementation of random variables and operators adding a property $\mathcal{T}(X)$ to a random variable X, such that X is $\mathcal{F}_s$-measurable for $s \geq \mathcal{T}(X)$. We set $\mathcal{T}(W(t)) := t$ for the Brownian driver W of our model, $\mathcal{T}(c) := -\infty$ for deterministic random variables c and for any operator $Z = f(X,Y)$ we set $\mathcal{T}(Z) := \max(\mathcal{T}(X), \mathcal{T}(Y))$. Then $\mathcal{T}$ enables the optimization

$$\mathrm{E}^{\mathbb{Q}}(X \mid \mathcal{F}_t) = X \qquad \text{if } \mathcal{T}(X) \leq t.$$

4 Differentiation of the Indicator Function

For a Bermudan digital option, the differentiation of B also contains the differentiation of the indicator function. Differentiation of the indicator function also

Electronic copy available at: https://ssrn.com/abstract=3000822

appears in other products and even for the valuation of a Bermudan option the differentiation of the indicator function is relevant, given that the estimation of the exercise boundary is not optimal. Hence we have to consider

$$\frac{\partial}{\partial X}\mathbf{1}(X>0),$$

where

$$\mathbf{1}(X>0)(\omega) := \begin{cases} 1 & \text{for } X(\omega)>0, \\ 0 & \text{else.} \end{cases}.$$

Automatic differentiation applied to algorithms involving indicator functions results in a linear combination of differentiations of the indicator functions. Hence we have to evaluate expressions of the form

$$A\cdot\frac{\partial}{\partial X}\mathbf{1}(X>0),$$

where A is a linear operator (the adjoint differential).

If we are only interested in the expectation of the final result, it is sufficient to consider

$$\mathrm{E}\left(A\frac{\partial}{\partial X}\mathbf{1}(X>0)\right),$$

which evaluates to

$$\mathrm{E}\left(A\frac{\partial}{\partial X}\mathbf{1}(X>0)\right) = \mathrm{E}\left(A \mid \{X=0\}\right),$$

which can be approximated by

$$\approx \mathrm{E}\left(A\frac{1}{2\delta}\mathbf{1}(|X|<\delta)\right). \tag{9}$$

In other words, if we are only interested in the expectation of the final result, we can approximate

$$\frac{\partial}{\partial X}\mathbf{1}(X>0) \approx \frac{1}{2\delta}\mathbf{1}(|X|<\delta).$$

This approximation has an intuitive interpretation. First, we may observe that this approximation agrees with the result of a so-called payoff-smoothing, where the indicator function is approximated by a call spread

$$\mathbf{1}(X>0)[\omega] \approx \begin{cases} \frac{X(\omega)+\delta}{2\delta} & \text{for } |X(\omega)|<\delta, \\ 0 & \text{else.} \end{cases}.$$

But more strikingly, the approximation is just a central finite difference approximation of the derivative. It is

$$\frac{\partial}{\partial X}\mathbf{1}(X>0) \approx \frac{\mathbf{1}(X+\delta>0)-\mathbf{1}(X-\delta>0)}{2\delta}.$$

Electronic copy available at: https://ssrn.com/abstract=3000822

Here we have locally replaced the automatic differentiation by a (local)n finite-difference approximation.

Since the implementation in [9, 7] has access to the full random variable X, we can achieve an important improvement in the numerical algorithm: we can choose the δ shift appropriate for the random variable under consideration.

For example, we can choose size of the *bin* δ in (9) as a multiple of the standard deviation of X

$$\mathrm{E}\left(A\frac{\partial}{\partial X}\mathbf{1}(X>0)\right) \approx \mathrm{E}\left(A\frac{1}{2\delta}\mathbf{1}(|X|<\delta)\right) \qquad 2\delta = \epsilon\cdot\left(\mathrm{E}\left(X^2\right)\right)^{\frac{1}{2}}, \quad (10)$$

which essentially determines the number of paths used to approximate the conditional expectation (9) by a binning.[5] The effect of ϵ (or δ) is illustrated in our numerical results.

Our numerical results show that it is an important advantage to apply the approximation (10) on a per-operator basis with an appropriate bin size δ, see Figure 3 and 4.

5 Higher-Order Sensitivities

Higher order sensitivities are possible by applying the automatic differentiation to the lower order automatic differentiation. This is immediately available in the implementation [7] (see the test cases there for examples).

The results presented here, remain valid for higher order sensitivities, since the automatic differentiation results in "admissible" operators, i.e. operators on which the method is applicable: for example, the first order differentiation of the indicator function results in a conditional expectation.

[5] Assuming a normal distributed X, a value of $\epsilon = 0.2$, would result in approximately 8% of the paths used for the estimation of conditional expectation (± 0.1 std. dev).

Electronic copy available at: https://ssrn.com/abstract=3000822

6 Numerical Results

6.1 AAD Delta of a Bermudan Digital Option

As a first test case we calculate the delta of a Bermudan digital option paying

$$\begin{cases} 1 & \text{if } S(T_i) - K_i > \tilde{U}(T_i) \text{ in } T_i \text{ and if no payout had been occurred before,} \\ 0 & \text{otherwise,} \end{cases}$$

where $\tilde{U}(T)$ is the time T value of the future payoffs, for $T_1, \dots, T_n$. Note that $\tilde{U}(T_n) = 0$, such that the last payment is a digital option.

This product is an ideal test-case: the valuation of $\tilde{U}(T_i)$ is a conditional expectation. In addition, the conditional expectation only appears in the indicator function. The delta of a digital option payoff is only driven by the movement of the indicator function, since

$$\frac{\mathrm{d}}{\mathrm{d}S_0}\mathrm{E}(\mathbf{1}(f(S(T)) > 0)) = \phi(f^{-1}(0))\frac{\mathrm{df(S)}}{\mathrm{d}S_0},$$

where ϕ is the probability density of S and $\mathbf{1}(\cdot)$ the indicator function (see [8]). Hence, keeping the exercise boundary fixed, would result in a delta of 0.

We calculate the delta of a Bermudan digital option under a Black-Scholes model ($S_0 = 1.0$, $r = 0.05$, $\sigma = 0.30$ using 1000000 Monte-Carlo paths. We consider an option with $T_1 = 1$, $T_2 = 2$, $T_3 = 3$, $T_4 = 4$, and $K_1 = 0.5$, $K_2 = 0.6$, $K_3 = 0.8$, $K_4 = 1.0$. The implementation of this test case is available in [7]. The results are depicted in Figure 2.

We depicts the finite difference approximation as red dots with different shift sizes on the x-axis. The finite difference approximation was performed with different Monte-Carlo seeds. We see the well-known effect that a finite-difference approximation of the derivative is biased for large shift sizes and unstable / unreliable for small shift sizes. We see stable and unbiased results only for shift sizes h in the range 0.01 – 0.1.

We then depict the result of our method in green.[6] Our method reproduces the stable and unbiased result. Its Monte-Carlo error is indistinguishable in the graph.

To illustrate that taking the conditional expectation of the adjoint differential is required, we repeat the calculation without this step: In blue we depict the value of an automatic differentiation if the conditional expectation is omitted in the calculation of $A^*_{1,i}$ and see that this would give a wrong result. And even worse, if we would keep the exercise boundary fixed, the result would be 0.

6.2 AAD Vega of a Bermudan Option (under LIBOR Market Model)

As a second test case we consider the AAD calculation of the vega of a Bermudan Swaption (30Y in 40Y, with semi-annual exercise) under a LIBOR Market Model

[6] Since there is no shift size, the is no dependency on the shift size and we get a horizontal line.

Electronic copy available at: https://ssrn.com/abstract=3000822

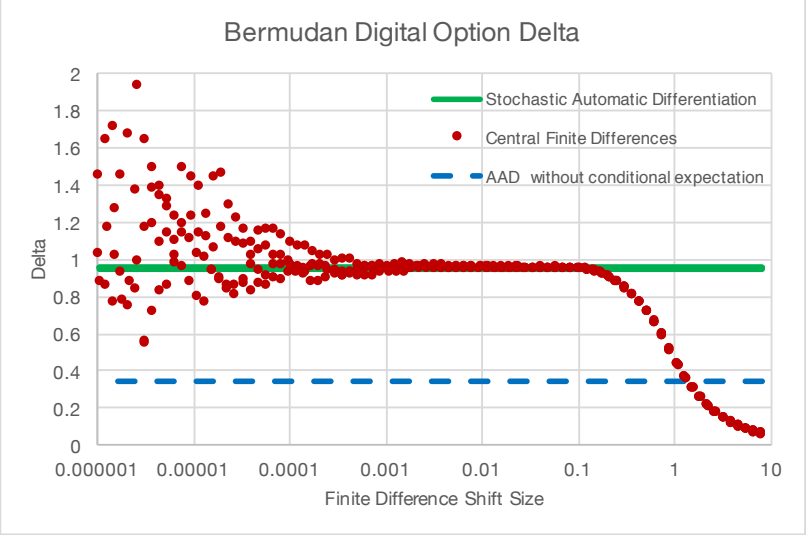

Figure 2: Delta of a Bermudan digital option using finite differences (red, different finite difference shit sized) and stochastic AAD (green). The calculations were repeated with different Monte-Carlo random number seeds. The finite difference approximation is biased for large shifts size. The blue dashed line depicts the AAD delta when the conditional expectation operator is ignored. This generates a bias due to the correlation of the derivative and the underlying. Ignoring differentiation of the conditional expectation would result in delta being 0.

Electronic copy available at: https://ssrn.com/abstract=3000822

(40Y, semi-annual, 320 time-steps, 15000 paths). The model has 25600 independent instantaneous volatility parameters, resulting in 12640 effective vega sensitivities. The performance of the algorithm is given in Table 1.

Vega of a Bermudan Swaption in the LIBOR Market Model			
algorithm	evaluation	derivative	memory usage
finite difference	2.0 s	25600 s (7 hours)	1.5 GB
efficient stochastic AAD	4.6 s	27 s	8.7 GB

Table 1: Calculation times and memory usages for finite differences and backward algorithmic differentiation algorithms. LIBOR market model with 40 years of simulation, 15000 paths, 80 forward rates, 320 time discretization points (1.5 month simulation time step), resulting in 25600 theoretical model vegas.

To analyse the numerical stability, we depict the values of a parallel vega of the Bermudan swaption, where the conditional expectation is estimated by a least-square regression. In theory the differentiation of the conditional expectation could be ignored, since the Bermudan swaption valuation could assume optimal exercises (see [13]). However, the Monte-Carlo error and the regression error will result in a non-optimal exercise (see [11]), which results in (slightly) a different vega. Assuming optimal exercise will result in wrong hedge ratios, since an exercise will be effectively sub-optimal due to the biased valuation of the future exercise dates. In addition, a comparison of the two different vegas (with and without ignoring differentiation of the indicator function) gives an indication of the quality of the conditional expectation estimator.

We may control the differentiation of the exercise boundary through the parameter ϵ in (10) (where $\epsilon = 0$ is interpreted as ignoring the differentiation of the indicator function). The parameter ϵ thus enables us to investigate the optimality of each exercise boundary.

In Figure 3 we show the AAD vega for different values ϵ for the size of the bin estimating the derivative of the indicator function. Since ϵ determines the number of Monte-Carlo paths used for sampling the differentiation of the indicator functions, we see a larger Monte-Carlo error for very small epsilon. The value $\epsilon = 0$ will switch off the differentiation of the indicator function and hence result in the AAD differentiation ignoring the differentiation of the indicator function.

In Figure 4 we depict the classic, brute-force finite difference approximation of the vega for different shift sizes h and the stochastic AAD calculation of vega for $\epsilon = 0$ (green) and $\epsilon = 0.2$ (red). We recover the effect that the finite difference approximation ignores the differentiation of the exercise boundary for very small shifts. This is due to the fact that under a small shift no path is crossing the exercise boundary. For larger shifts the finite difference approximation includes the differentiation of the exercise boundary, but results in a huge Monte-Carlo variance.

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

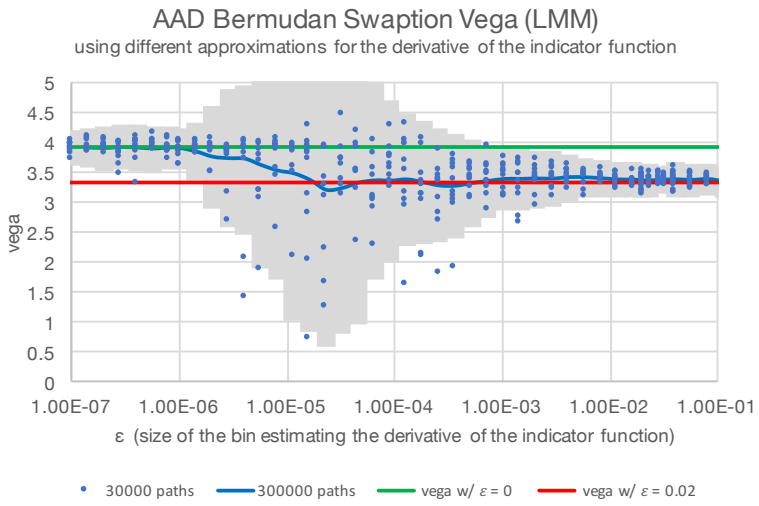

Figure 3: Vega of a Bermudan swaption using AAD for different values of ϵ for the differentiation of the indicator function in (10). The calculations were performed for 30000 paths with different bin sizes ϵ (blue). We depict a mean (blue line) and standard deviation (grey). The green line marks the vega for $\epsilon = 0$ (ignoring the differentiation of the indicator function). The red line marks the vega for $\epsilon = 0.2$. Small values of epsilon result in unstable vegas, since only few paths are used for the estimation. The value 0.2 can be interpreted statistically using 8% of the paths for the estimation of the derivative.

Electronic copy available at: https://ssrn.com/abstract=3000822

It is almost impossible to obtain a reliable estimate for the Bermudan swaption vega by finite differences, which includes the effect of a sub-optimal exercise by differentiation the indicator functions. AAD gives a reliable estimate (red). The finite difference sample points show some convergence to the correct solution for a narrow range of shifts sized h around 0.005.

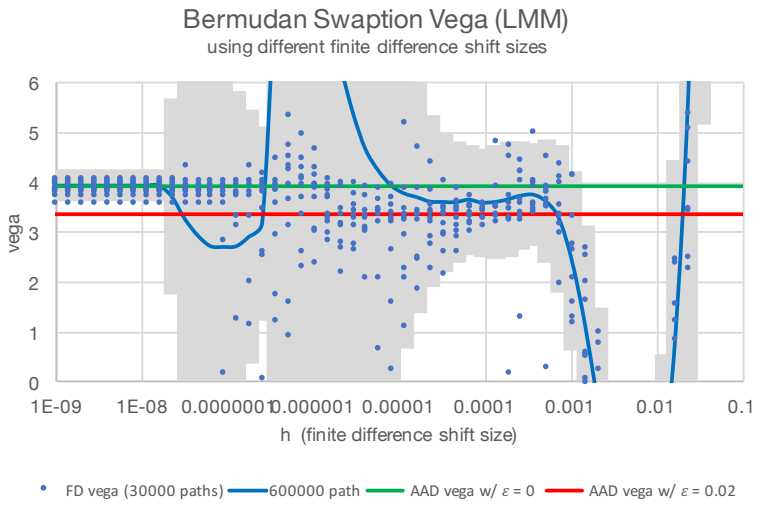

Figure 4: Vega of a Bermudan swaption using finite differences (blue, different random number seeds and finite difference shifts). The red and green line mark the values obtained by AAD, where the differentiation of the indicator function was performed using (10) with $\epsilon = 0$, ignoring differentiation of the indicator function (green), and $\epsilon = 0.2$ (red), resulting in two different values for vega. The patterns visible for finite difference vegas are due to the same Monte-Carlo seed used for different shift sizes, generating a jump in vega once a Monte-Carlo path crosses the exercise boundary.

A comparison of Figure 4 and 3 highlights an advantage of the AAD differentiation of the indicator function: while a finite difference shift on the parameter acts on all indicator functions (e.g., all exercise dates) and an optimal or good choice of the finite difference shift is unclear, the AAD differentiation allows for a per-operator determination of the size of the estimation bin.

Electronic copy available at: https://ssrn.com/abstract=3000822

7 Implementation Design

7.1 Conditional Expectation Operators

Using the framework described in [9] we are able to implement the methodology with a minimum of code complexity: we require only two additional lines of code: In [6] all random variable objects of a Monte-Carlo simulation are implementing a common interface `RandomVariableInterface`. This interface offers methods for arithmetic operators like `add`, `sub`, `mult`, `exp`, but also an operator

```
RandomVariableInterface getConditionalExpectation(
    ConditionalExpectationEstimatorInterface estimator);
```

The default implementation of this method is just

```
default RandomVariableInterface getConditionalExpectation(
    ConditionalExpectationEstimatorInterface estimator)
{
 return estimator.getConditionalExpectation(this);
}
```

In other words: all conditional expectations of random variables have to be called through this method - which enables us to track conditional expectation in the operator tree.

To enable adjoint automatic differentiation we inject a special implementation of `RandomVariableInterface` which records the operator tree and can perform the AAD. The implementation of the differential of the operator `getConditionalExpectation` now consists of two parts: First, the partial derivative of the operator with respect to its arguments is set to 1.0:

```
case CONDITIONAL_EXPECTATION:
 return new RandomVariable(1.0);
```

- this implies that for the arguments the conditional expectation is replaced by the identity operator. Second, the backward propagation of the differential contains the additional check for the operator, applying it to the differential if required:

```
if(operatorType == OperatorType.CONDITIONAL_EXPECTATION) {
 ConditionalExpectationEstimatorInterface estimator = (
     ConditionalExpectationEstimatorInterface)operator;
 derivative = estimator.getConditionalExpectation(derivative);
}
```

7.2 Indicator Functions

The treatment of the indicator function is implemented on a per-operator level. In [6] the indicator function operator $\mathbf{1}(X \geq 0) \cdot Y + \mathbf{1}(X < 0) \cdot Z$ is called `BARRIER` and the stochastic AD derivative with respect to X is implemented by

```
case BARRIER:
 /*
  * Approximation of derivative of indicator function via local finite difference
  */
 result = Y.sub(Z);
 double epsilon = getBarrierDiracWidth() * X.getStandardDeviation();
```


Electronic copy available at: https://ssrn.com/abstract=3000822

```
if(epsilon > 0) {
  result = result.mult(X.barrier(X.add(epsilon/2), one, zero));
  result = result.mult(X.barrier(X.sub(epsilon/2), zero, one));
  result = result.div(epsilon);
}
else {
  result = zero;
}
```

7.3 Automatic Tracking of Measurability

Like an automatic differentiation we can implement (or "overload") the operators to automatically track the filtration time of a random variable: The filtration time of the result of an operator is the maximum of the filtration time[7] of its argument. The filtration time of a constant is set to $-\infty$. The filtration time of a Brownian increment $W(t+\Delta t) - W(t)$ is set to $t + \Delta t$. If a random variable provides its filtration time as `getFiltrationTime()`, then we can optimize the conditional expectation estimator by a pre-check, ignoring the conditional expectation, when admissible:

```
RandomVariableInterface getConditionalExpectation(RandomVariableInterface
    value) {
  if(value.getFiltrationTime() <= conditionTime) return value;

  //...
```

See [6].

[7] By the filtration time we denote a time t, such that the random variable is $\mathcal{F}_t$-measurable. In most applications is is possible to automatically track the minimal filtration time.

Electronic copy available at: https://ssrn.com/abstract=3000822

8 Conclusion

In this note we presented a modification of the backward automatic differentiation to apply a true adjoint algorithmic differentiation to algorithms which require the calculation (estimation) of a conditional expectation: the conditional expectation operator has to be applied to the adjoint differential, while the remaining algorithm remains unchanged.

Our method has important advantages over a direct differentiation of the conditional expectation estimator ([3, 5]):

1. Do not differentiate an approximation: It is often not clear that differentiation of an approximation results in a good approximation of the differential. In our approach it is not necessary to differentiate the approximation of the conditional expectation operator. A simple example is if the conditional expectations is approximated by fixed bins, which results in a piecewise constant approximation (see [8]) and the (automatic) differentiation of the conditional expectation would be zero.

2. The approach allows to use a different approximation of the conditional expectation operator in the valuation and in the algorithmic differentiation. This can improve the accuracy of the differentiation. For example, it is possible to choose different regression basis functions.

Most importantly, the approach greatly simplifies the implementation of the algorithm. In [7] the implementation basically consists of adding two lines of code.

In addition we discussed the treatment of the indicator function within this algorithm, where the differentiation of indicator function can be understood as a conditional expectation. This allows to fine-tune the treatment of the differentiation of the indicator function. It is possible to either ignore the differentiation of the indicator function or to use a numerically robust approximation.

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822

References

[1] ANDREASEN, JESPER: CVA on an iPad Mini. Global Derivatives, ICBI, Amsterdam 2014.

[2] ANTONOV, ALEXANDRE; ISSAKOV, S.;KONIKOV M.; MCCLELLAND, A. ; MECHKOV, S.: PV and XVA Greeks for Callable Exotics by Algorithmic Differentiation. (February, 2107) SSRN. http://ssrn.com/abstract=2881992

[3] ANTONOV, ALEXANDRE: Algorithmic Differentiation for Callable Exotics (April 4, 2017). http://ssrn.com/abstract=2839362.

[4] CAPRIOTTI, LUCA; GILES, MIKE: Algorithmic Differentiation: Adjoint Greeks Made Easy (2011). http://ssrn.com/abstract=1801522.

[5] CAPRIOTTI, LUCA; JIANG, YUPENG; MACRINA, ANDREA: AAD and Least Squares Monte Carlo: Fast Bermudan-Style Options and XVA Greeks (September 23, 2016). https://ssrn.com/abstract=2842631 or http://dx.doi.org/10.2139/ssrn.2842631

[6] FINMATH.NET: finmath-lib: Mathematical Finance Library: Algorithms and methodologies related to mathematical finance. https://github.com/finmath/finmath-lib

[7] FINMATH.NET: finmath-lib automatic differentiation extentions: Enabling finmath lib to utilize automatic differentiation algorithms (e.g. AAD). https://github.com/finmath/finmath-lib-automaticdifferentiation-extensions

[8] FRIES, CHRISTIAN P.: Mathematical Finance. Theory, Modeling, Implementation. John Wiley & Sons, 2007. ISBN 0-470-04722-4. http://www.christian-fries.de/finmath/book.

[9] FRIES, CHRISTIAN P.: Stochastic Automatic Differentiation: Efficient Implementation of Automatic Differentiation for Monte-Carlo Simulations. (June, 2017). https://ssrn.com/abstract=2995695

[10] FRIES, CHRISTIAN P.: Fast stochastic forward sensitivities in Monte-Carlo simulations using stochastic automatic differentiation (with applications to initial margin valuation adjustments (MVA)). *SSRN*, August 2017. http://ssrn.com/abstract=3018165.

[11] FRIES, CHRISTIAN P.: Foresight Bias and Suboptimality Correction in Monte-Carlo Pricing of Options with Early Exercise: Classification, Calculation and Removal. *SSRN*, 2005. http://ssrn.com/abstract=839105.

Electronic copy available at: https://ssrn.com/abstract=3000822

[12] GILES, MICHAEL; GLASSERMAN, PAUL: Smoking adjoints: fast Monte Carlo Greeks (2006). Risk, January 2006. https://www.risk.net

[13] PITERBARG, VLADIMIR: Computing Deltas of callable LIBOR exotics in forward LIBOR models (2004). *Journal of Computational Finance*, January 2004, Vol 7, 107-144.

Notes

Suggested Citation

FRIES, CHRISTIAN P.: Automatic Differentiation of the American-Monte-Carlo Backward Algorithm. (June, 2017).
https://ssrn.com/abstract=3000822
http://www.christian-fries.de/finmath/stochasticautodiff

Classification

MSC-class: 65C05 (Primary)

ACM-class: G.1.4; G.3; I.6.8.

JEL-class: C15, G13.

22 pages. 4 figures. 1 tables.

©2017 Christian Fries

Electronic copy available at: https://ssrn.com/abstract=3000822