Adjoints and automatic (algorithmic) differentiation in computational finance

Cristian Homescu*

Revised version: May 8, 2011†

Two of the most important areas in computational finance: Greeks and, respectively, calibration, are based on efficient and accurate computation of a large number of sensitivities. This paper gives an overview of adjoint and automatic differentiation (AD), also known as algorithmic differentiation, techniques to calculate these sensitivities. When compared to finite difference approximation, this approach can potentially reduce the computational cost by several orders of magnitude, with sensitivities accurate up to machine precision. Examples and a literature survey are also provided.

*Email address: cristian.homescu@gmail.com

†Original version: May 1, 2011

Contents

1 Introduction 3

2 General description of the approach 4
- 2.1 Single function . . . 4
- 2.2 Composite functions . . . 5
- 2.3 Checking the correctness of the implementation . . . 6

3 Simple example 7
- 3.1 Forward (tangent linear) mode . . . 7
- 3.2 Adjoint (reverse) mode . . . 8

4 Example in PDE solver framework 9
- 4.1 Forward (tangent linear) mode . . . 9
- 4.2 Adjoint (reverse) mode . . . 10

5 Example in Monte Carlo framework (evolution of SDE) 11
- 5.1 Forward (tangent linear) mode . . . 11
- 5.2 Adjoint (reverse) mode . . . 13

6 Example in Monte Carlo framework (copula) 13
- 6.1 Forward (tangent linear) mode . . . 14
- 6.2 Adjoint (reverse) mode . . . 15

7 Example in calibration/optimization framework 16

8 Computational finance literature on adjoint and AD 16
- 8.1 Computation of Greeks . . . 16
- 8.2 Calibration . . . 19

9 AD and adjoint applied within a generic framework in practitioner quant libraries 19
- 9.1 Block architecture for Greeks . . . 19
- 9.2 Real Time Counterparty Risk Management in Monte Carlo . . . 20

10 Additional resources 20

11 Conclusion 20

1 Introduction

Two of the most important areas in computational finance: Greeks and, respectively, calibration, are based on efficient and accurate computation of a large number of sensitivities. This paper gives an overview of adjoint and automatic(algorithmic) differentiation techniques to calculate those sensitivities. While only recently introduced in the field of computational finance, it was successfully employed in the last 20 years in other areas, such as computational fluid dynamics, meteorology and atmospheric sciences, engineering design optimization, etc: [25, 27, 42, 40, 46, 47, 33, 61, 68], to mention but a few.

The computation of the sensitivities is done by performing differentiation at either continuous level or, respectively, at discrete level. The "continuous" adjoint approach corresponds to the case where the adjoint equation is formulated at the differential equation level, and then discretized. In contrast, the "discrete" adjoint approach starts with discretized equations, and then formulates the corresponding discrete adjoint equations. Consequently, the continuous adjoint is also known under "differentiate then discretize" moniker, while the discreet adjoint corresponds to "discretize then differentiate" moniker.

The discrete adjoint approach is preferred in many occasions, for several practical reasons:

- constructing the discrete adjoint equations is a more straightforward and systematic process
- Automatic Differentiation software can be employed to greatly reduce the development time

For this reason we will concentrate in this paper on the discrete adjoint approach, and for ease of presentation we refer to it as the adjoint approach. However, since a few papers that are reviewed here are also based on continuous adjoint, we will make that information explicit when we refer to those papers, and use the abbreviation ContAdj for the "continuous adjoint" approach.

Automatic Differentiation (AD), also known as Algorithmic Differentiation, is a chain-rule-based technique for evaluating the derivatives with respect to the input variables of functions defined by a high-level language computer program. AD relies on the fact that all computer programs, no matter how complicated, use a finite set of elementary (unary or binary, e.g. $\sin(\cdot)$, $\mathrm{sqrt}(\cdot)$) operations as defined by the programming language. The value or function computed by the program is simply a composition of these elementary functions. The partial derivatives of the elementary functions are known, and the overall derivatives can be computed using the chain rule.

AD has two basic modes of operations, the forward mode and the reverse mode. In the forward mode the derivatives are propagated throughout the computation using the chain rule, while the reverse mode computes the derivatives for all intermediate variables backwards (i.e., in the reverse order) through the computation. In the literature, AD forward mode is sometimes referred to as *tangent linear* mode, while AD reverse mode is denoted as *adjoint* mode.

The adjoint method is advantageous for calculating the sensitivities of a small number of outputs with respect to a large number of input parameters. The for- ward method is advantageous in the opposite case, when the number of outputs (for which we need sensitivities) is larger compared to the number of inputs.

When compared to regular methods (such as finite differencing) for computing sensitivities, AD has 2 main advantages: reduction in computational time and accuracy. The reduction in computational time is assured by a theoretical result [43]that states that the cost of the reverse mode is smaller than five times the computational cost of a regular run. The computational cost of the adjoint approach is independent of the number of inputs for which we want to obtains the sensitivities with respect to, whereas the cost of thetangent linear approach increases linearly with the number of inputs. Regarding accuracy, AD computes the derivatives exactly (up to machine precision) while finite differences incur truncation errors.

The size of the step h needed for finite difference varies with the current value of of input parameters, making the problem of choosing h, such that it balances accuracy and stability, a challenging one. AD on the other hand, is automatic and time need not be spent in choosing step-size parameters, etc.

AD software packages can also be employed to speed up the development time. Such tools implement the semantic transformation that systematically applies the chain rule of differential calculus to source code written in various programming languages. The generated code can operate in forward or reverse mode (tangent linear or adjoint model).

2 General description of the approach

Let us assume that we have a model dependent on a set of input parameters which produces an output Y. We also denote by $Z = \{X_1, X_2, ..., X_N\}$ the vector of input parameters with respect to which we want to compute sensitivities.

2.1 Single function

We assume we have a single function of the model output, denoted $F(Y)$. The goal is to obtain sensitivities of F with respect to the components of Z.

Computation of the vector of sensitivities

$$\dot{F} = \frac{\partial F}{\partial Z} = \frac{\partial F}{\partial X_1}\dot{X}_1 + \cdots + \frac{\partial F}{\partial X_N}\dot{X}_N \tag{2.1}$$

is done using the forward (tangent linear) mode. If we include the trivial equations $\dot{X}_1 = \dot{X}_1, \cdots, \dot{X}_N = \dot{X}_N$, then we can rewrite the combined expressions in matrix notation. This will prove helpful when constructing the adjoint mode, using the fact that adjoint of a matrix A is defined as its conjugate transpose (or simply the transpose, if the matrix A has only real elements)

$$\begin{pmatrix} \dot{X}_1 \\ \cdots \\ \dot{X}_N \\ \dot{F} \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & 1 \\ \frac{\partial F}{\partial X_1} & \cdots & \frac{\partial F}{\partial X_N} \end{pmatrix} \begin{pmatrix} \dot{X}_1 \\ \cdots \\ \dot{X}_1 \end{pmatrix} \tag{2.2}$$

To compute each component of the vector $\frac{\partial F}{\partial Z}$ we need to evaluate the expression (2.1) a number of N times, every time with a different input vector $\dot{Z} = \left\{\dot{X}_1, \ldots, \dot{X}_N\right\}$.For example, to compute the derivative with respect to X_j, the vector $\dot{Z}$ has the value $\dot{Z} = (0, 0, \ldots, 1, 0, \ldots, 0)$, with the only nonzero element in the $j - th$ position.

To construct the reverse (adjoint) mode, we start with the transposed matrix equation (2.2) With the notation of $\bar{A}$ for the adjoint variable corresponding to an original variable A, we have

$$\begin{pmatrix} \bar{X}_1 \\ \cdots \\ \bar{X}_N \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 & \frac{\partial F}{\partial X_1} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 1 & \frac{\partial F}{\partial X_N} \end{pmatrix} \begin{pmatrix} \bar{X}_1 \\ \cdots \\ \bar{X}_N \\ \bar{F} \end{pmatrix}$$

Consequently, we obtain the following expressions

$$\begin{array}{lll} \bar{X}_1 & = \bar{X}_1 & + \dfrac{\partial F}{\partial X_1}\bar{F} \\ \cdots & \cdots & \cdots \\ \bar{X}_N & = \bar{X}_N & + \dfrac{\partial F}{\partial X_N}\bar{F} \end{array} \tag{2.3}$$

Thus we can obtain the sensitivities in one single run of (2.3), with $\bar{F} = 1$ and the adjoint variables $\bar{X}_1, \ldots, \bar{X}_N$ initialized to zero.

2.2 Composite functions

We can generalize this procedure if the output is obtained through evaluation of a composite function of P single functions (which is in fact how the computer codes are represented):

$$F = F^P \circ F^{P-1} \circ \cdots \circ F^1(Z)$$

We apply the tangent linear mode to each F^j, and we combine them in the recursion relationship. For the adjoint (reverse mode), we construct the adjoint for each F^j, and we combine them in reverse order.

Let us describe the process using the matrix notation. If we view the tangent linear as the result of the multiplication of a multiplication of a number of operator matrices

$$MAT = Mat_1 \cdot Mat_2 \cdots Mat_P$$

where each matrix Mat_j represents either a subroutine or a single statement, then the adjoint approach can be viewed as a product of adjoint subproblems

$$Mat^T = Mat_P^T \cdot Mat_{P-1}^T \cdots Mat_1^T$$

Let us describe how it works through an example for P=2. The computational flow is described by the following diagram

$$Z \to F^1(Z) \to F^2\left(F^1(Z)\right) \to Y$$

For simplicity, we denote by A the output of $F^1(Z)$ and by B the output of $F^2\left(F^1(Z)\right)$. Y is the scalar that is the final output. For example, Y can be the value of the function to be calibrated (in the calibration setup) or, respectively, the price of the option (in the pricing setup, e..g. using Monte Carlo or finite differences). With these notations, we have

$$Z \to A \to B \to Y$$

Applying tangent linear methodology (essentially differentiation line by line) we have

$$\begin{array}{lll} \dot{A} & = & \dfrac{\partial A}{\partial Z}\dot{Z} \\ \dot{B} & = & \dfrac{\partial B}{\partial A}\dot{A} \\ \dot{Y} & = & \dfrac{\partial Y}{\partial B}\dot{B} \end{array}$$

Putting everything together, we get

$$\dot{Y} = \frac{\partial Y}{\partial B}\frac{\partial B}{\partial A}\frac{\partial A}{\partial Z}\dot{Z}$$

Using notation from AD literature, the adjoint quantities $\bar{Z}, \bar{A}, \bar{B}, \bar{Y}$ denote the derivatives of Y with respect to Z, A, B and, respectively, to Y . We note that this implies that $\bar{Y} = 1$. Differentiating again, and with a superscript ^{T} denoting a matrix or vector transpose, we obtain

$$\bar{Z} = \left(\frac{\partial Y}{\partial Z}\right)^T = \left(\frac{\partial Y}{\partial A}\frac{\partial A}{\partial Z}\right)^T = \left(\frac{\partial A}{\partial Z}\right)^T \bar{A}$$

In a similar way, we obtain

$$\bar{A} = \left(\frac{\partial Y}{\partial A}\right)^T = \left(\frac{\partial Y}{\partial B}\frac{\partial B}{\partial A}\right)^T = \left(\frac{\partial B}{\partial A}\right)^T \bar{B}$$
$$\bar{B} = \left(\frac{\partial Y}{\partial B}\right)^T = \left(\frac{\partial Y}{\partial B}\right)^T \cdot 1 = \left(\frac{\partial Y}{\partial B}\right)^T \bar{Y}$$

Putting everything together, we get

$$\bar{Z} = \left(\frac{\partial A}{\partial Z}\right)^T \left(\frac{\partial B}{\partial A}\right)^T \left(\frac{\partial Y}{\partial B}\right)^T \bar{Y}$$

We notice that the tangent linear approach proceeds forward (in forward mode)through the process

$$\dot{Z} \to \dot{A} \to \dot{B} \to \dot{Y}$$

while the adjoint approach proceeds backwards (in reverse mode)

$$\bar{Y} \to \bar{B} \to \bar{A} \to \bar{Z}$$

2.3 Checking the correctness of the implementation

There are several checks that need to be made [61, 4, 39].

First, at any level of the code, the development of the discrete adjoint model can be checked by appling the following identity

$$(AQ)^T (AQ) = Q^T \left[A^T (AQ)\right]$$

where Q represents the input to original code, and A represents either a single statement or a subroutine

We compare the gradient computed using AD to the gradient computed by Finite Difference (with a step size such that sufficient convergence is obtained). We also compare the gradient computed using AD in Forward mode versus the gradient computed using AD in Adjoint mode. We expect those two gradients to be identical up to machine precision.

If possible, we may use complex numbers [39, 66], to avoid roundoff errors due to computations with finite difference.

3 Simple example

We want to compute the gradient of the function $f(a,b,c) = (w - w_0)^2$, where w is obtained using the following sequence of statements

$$\begin{aligned} u &= \sin(ab) + cb^2 + a^3c^2 \\ v &= \exp\left(u^2 - 1\right) + a^2 \\ w &= ln\left(v^2 + 1\right) + cos\left(c^2 - 1\right) \end{aligned}$$

The input vector is denoted by by $z = \{a, b, c\}$, intermediate variables u, w, v and output f.
We show how the sensitivities with respect to a,b,c, namely

$$\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}, \frac{\partial f}{\partial c}$$

are computed in both forward and adjoint mode. For forward mode we also write the expressions in matrix notation, to make it easier to understand how the adjoint mode is constructed.

3.1 Forward (tangent linear) mode

We follow the computational flow, and thus we start by getting the sensitivities with respect to the intermediate variables, denoted by $\dot{u}, \dot{v}, \dot{w}$

$$\begin{aligned} \dot{u} &= \frac{\partial u}{\partial a}\dot{a} + \frac{\partial u}{\partial b}\dot{b} + \frac{\partial u}{\partial c}\dot{c} \\ \frac{\partial u}{\partial a} &= b\cos(ab) + 3a^2c^2 \\ \frac{\partial u}{\partial b} &= a\cos(ab) + 2cb \\ \frac{\partial u}{\partial c} &= b^2 + 2a^3c \end{aligned}$$

Hence we obtain

$$\dot{u} = \left[b\cos(ab) + 3a^2c^2\right]\dot{a} + \left[a\cos(ab) + 2cb\right]\dot{b} + \left[b^2 + 2a^3c\right]\dot{c}$$

In matrix notation

$$\begin{pmatrix} \dot{a} \\ \dot{b} \\ \dot{c} \\ \dot{u} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ b\cos(ab) + 3a^2c^2 & a\cos(ab) + 2cb & b^2 + 2a^3c \end{pmatrix} \begin{pmatrix} \dot{a} \\ \dot{b} \\ \dot{c} \end{pmatrix}$$

In a similar way we obtain

$$\begin{aligned} \dot{v} &= 2u\exp\left(u^2 - 1\right)\dot{u} + 2a\dot{a} \\ \dot{w} &= \frac{2v}{v^2 + 1}\dot{v} - 2c\sin\left(c^2 - 1\right)\dot{c} \\ \dot{f} &= 2\left(w - w_0\right)\dot{w} \end{aligned}$$

In matrix notation

$$\begin{aligned}
(\dot{v}) &= \begin{pmatrix} 2a & 2u\exp\left(u^2-1\right) \end{pmatrix} \begin{pmatrix} \dot{a} \\ \dot{u} \end{pmatrix} \\
(\dot{w}) &= \begin{pmatrix} -2c\sin\left(c^2-1\right) & \frac{2v}{v^2+1} \end{pmatrix} \begin{pmatrix} \dot{c} \\ \dot{v} \end{pmatrix} \\
\left(\dot{f}\right) &= \left(2\left(w-w_0\right)\right)(\dot{w})
\end{aligned}$$

To obtain the required sensitivity with respect to $j-th$ component of input vector$z$, we evaluate the above expressions starting with $\dot{z}$ which has the $j-th$ component set to 1 and all the other components set to 0.

More specifically, the sensitivities at a given set of variables $z^{(0)} = \left\{a^{(0)}, b^{(0)}, c^{(0)}\right\}$ are computed by calling the forward mode with the initial value $\dot{z}$ defined as follows:

$$\begin{aligned}
\frac{\partial f}{\partial a}\left(z^{(0)}\right) &= \dot{f} \quad computed\,with \quad \dot{z} = \left(\dot{a}, \dot{b}, \dot{c}\right) = (1,0,0) \\
\frac{\partial f}{\partial b}\left(z^{(0)}\right) &= \dot{f} \quad computed\,with \quad \dot{z} = \left(\dot{a}, \dot{b}, \dot{c}\right) = (0,1,0) \\
\frac{\partial f}{\partial c}\left(z^{(0)}\right) &= \dot{f} \quad computed\,with \quad \dot{z} = \left(\dot{a}, \dot{b}, \dot{c}\right) = (0,0,1)
\end{aligned}$$

3.2 Adjoint (reverse) mode

To write the Adjoint, we need to take statements in reverse order. Employing this approach to the statements of the forward mode yields

$$\begin{aligned}
(\bar{w}) &= \left(2\left(w-w_0\right)\right)\left(\bar{f}\right) \\
\begin{pmatrix} \bar{c} \\ \bar{v} \end{pmatrix} &= \begin{pmatrix} -2c\sin(c^2-1) \\ \frac{2v}{v^2+1} \end{pmatrix} (\bar{w}) \\
\begin{pmatrix} \bar{a} \\ \bar{u} \end{pmatrix} &= \begin{pmatrix} 2a \\ 2u\exp(u^2-1) \end{pmatrix} (\bar{v}) \\
\begin{pmatrix} \bar{a} \\ \bar{b} \\ \bar{c} \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 & b\cos(ab)+3a^2c^2 \\ 0 & 1 & 0 & a\cos(ab)+2cb \\ 0 & 0 & 1 & b^2+2a^3c \end{pmatrix} \begin{pmatrix} \bar{a} \\ \bar{b} \\ \bar{c} \\ \bar{u} \end{pmatrix}
\end{aligned}$$

Thus the adjoint (reverse) mode is constructed using the following sequence of statements

$$\begin{aligned}
\bar{w} &= 2\left(w-w_0\right)\bar{f} \\
\bar{c} &= -2c\sin(c^2-1)\bar{w} \\
\bar{v} &= \frac{2v}{v^2+1}\bar{w} \\
\bar{a} &= 2a\bar{v} \\
\bar{u} &= 2u\exp(u^2-1)\bar{v} \\
\bar{a} &= \bar{a}+(b\cos(ab)+3a^2c^2)\bar{u} \\
\bar{b} &= \bar{b}+(a\cos(ab)+2cb)\bar{u} \\
\bar{c} &= \bar{c}+(b^2+2a^3)c\bar{u}
\end{aligned} \tag{3.1}$$

We can compute all 3 sensitivities of function f with respect to a, b, c by a single application of the adjoint mode (3.1), with starting point $\bar{z} = 1$. More specifically, we have

$$\begin{aligned}
&\frac{\partial f}{\partial a}\left(z^{(0)}\right) = \bar{a} \quad computed\,with \quad \bar{a} = 1 \\
&\frac{\partial f}{\partial b}\left(z^{(0)}\right) = \bar{b} \quad computed\,with \quad \bar{b} = 1 \\
&\frac{\partial f}{\partial c}\left(z^{(0)}\right) = \bar{c} \quad computed\,with \quad \bar{c} = 1
\end{aligned}$$

The reader is encouraged to verify that the gradient computed via the Forward mode or, respectively, the Reverse mode has identical value.

4 Example in PDE solver framework

The following PDE is considered for $u(t, x)$, on the spatial domain$A \leq x \leq B$

$$\begin{cases}
\frac{\partial u}{\partial t} & = \frac{\partial u}{\partial x} \\
u(0, x) & = u_0(x) \\
u(t, A) & = f(t) \\
u(t, B) & = g(t)
\end{cases}$$

We discretize the PDE in space and time, for a spatial grid $\{x_j\}$and time grid $\left\{T^k\right\}$.We use notation of superscript for time index and subscript for spatial index. For simplicity of exposition, we discretize using a central difference scheme in space, a first order scheme in time, and we consider that both spatial grid and, respective, temporal grid are constant, with Δx and Δt

$$u_j^{k+1} = u_j^k + \frac{\Delta t}{2\Delta x}\left(u_{j+1}^k - 2u_j^k + u_j^{k-1}\right)$$

We denote by c the ratio $\frac{\Delta t}{2\Delta x}$. With that, and incorporating the boundary conditions, we have

$$\begin{aligned}
u_1^{k+1} &= f(T^{k+1}) \triangleq f_{k+1} \\
u_j^{k+1} &= u_j^k + c\left(u_{j+1}^k - 2u_j^k + u_j^{k-1}\right) \quad j = 2...N \\
u_N^{k+1} &= g(T^{k+1}) \triangleq fg_{k+1}
\end{aligned}$$

We want to minimize the difference $F \triangleq \sum_{j=1}^{N}\left(u_j^M - Y_j\right)^2$, with Y_j desired values to get at time T

We want to compute sensitivities of F with respect to the discretized initial condition $\left\{u_j^0\right\}$,where $u_j^0 \triangleq u_0(x_j)$

4.1 Forward (tangent linear) mode

The tangent linear approach has the expression

$$\dot{u}_j^{k+1} = (1 - 2c)\,\dot{u}_j^k + c\left(\dot{u}_{j+1}^k + \dot{u}_{j-1}^k\right) \quad j = 2...N$$

In matrix notation

$$\begin{pmatrix} \dot{u}_1^{k+1} \\ \dots \\ \dot{u}_{j-1}^{k+1} \\ \dot{u}_j^{k+1} \\ \dot{u}_{j+1}^{k+1} \\ \dots \\ \dot{u}_N^{k+1} \end{pmatrix} = \begin{pmatrix} 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ c & 1-2c & c & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & c & 1-2c & c & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & c & 1-2c & c \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \end{pmatrix} \begin{pmatrix} \dot{u}_1^{k} \\ \dots \\ \dot{u}_{j-1}^{k} \\ \dot{u}_j^{k} \\ \dot{u}_{j+1}^{k} \\ \dots \\ \dot{u}_N^{k} \end{pmatrix} \tag{4.1}$$

The last step is

$$\dot{F} = 2\left(u_1^M - Y_1\right)\dot{u}_1^M + \dots + 2\left(u_j^M - Y_j\right)\dot{u}_j^M + \dots + 2\left(u_N^M - Y_N\right)\dot{u}_N^M$$

In matrix notation

$$\left(\dot{F}\right) = \begin{pmatrix} 2\left(u_1^M - Y_1\right) & \dots & 2\left(u_j^M - Y_j\right) & \dots & 2\left(u_N^M - Y_N\right) \end{pmatrix} \begin{pmatrix} \dot{u}_1^M \\ \dots \\ \dot{u}_j^M \\ \dots \\ \dot{u}_N^M \end{pmatrix} \tag{4.2}$$

4.2 Adjoint (reverse) mode

We go backwards. and the starting point is (4.2). The corresponding adjoint statements are

$$\begin{aligned} \bar{u}_1^k &= 2\left(u_1^M - Y_1\right) \\ \dots \quad &\dots \quad \dots \\ \bar{u}_j^M &= 2\left(u_j^M - Y_j\right)\bar{F} \\ \dots \quad &\dots \quad \dots \\ \bar{u}_N^M &= 2\left(u_N^M - Y_N\right)\bar{F} \end{aligned}$$

Then we take the transpose of the matrix operator in (4.1). The corresponding adjoint statements are, for $k = M, M-1, \dots, 1, 0$

$$\begin{aligned} \bar{u}_1^k &= c\bar{u}_2^{k+1} \\ \bar{u}_2^k &= (1-2c)\bar{u}_2^{k+1} + c\bar{u}_3^{k+1} \\ \dots \quad &\dots \quad \dots \\ \bar{u}_j^k &= c\bar{u}_{j-1}^{k+1} + (1-2c)\bar{u}_j^{k+1} + c\bar{u}_{j+1}^{k+1} \\ \dots \quad &\dots \quad \dots \\ \bar{u}_{N-1}^k &= c\bar{u}_{N-2}^{k+1} + (1-2c)\bar{u}_{N-1}^{k+1} \\ \bar{u}_N^k &= c\bar{u}_N^{k+1} \end{aligned}$$

The required sensitivities are given by $\bar{u}_j^0$, for $j = 1...N$

5 Example in Monte Carlo framework (evolution of SDE)

We consider the SDE for a N-dimensional X vector

$$dX = a(X,t)dt + \sigma(X,t)dW$$

The initial value for X, at time t=0, is denoted by$X^0 = \left(X_1(T^0), ..., X_N(T^0)\right)$

For simplicity of exposition, we consider that we discretize it using Euler-Maruyama scheme, with time points $T^k, k = 1...M$ and $T^0 = 0$

$$\begin{aligned} X(T^{k+1}) - X(T^k) &= a(X(T^k), T^k)\Delta t + \sigma(X(T^k), T^k)\Delta W^k \\ \Delta T^k &= T^{k+1} - T^k \\ \Delta W^k &= \varepsilon \cdot \sqrt{T^{k+1} - T^k} \end{aligned}$$

where the random number is chosen from $\mathcal{N}$(0,1)

We can recast this discretization scheme into the following expression

$$X(T^{k+1}) = \Phi(X(T^k), \Theta) \tag{5.1}$$

where Φ may also depend on a set of model parameters $\Theta = \{\theta_1, ..., \theta_P\}$

We also consider that we have to price an option with payoff $G(X(T))$

We want to compute price sensitivities ("Greeks") with respect to X^0 and, respectively, to the model parameters Θ

5.1 Forward (tangent linear) mode

We consider first the sensitivities with respect to initial conditions

$$\frac{\partial G(X(T))}{\partial X_i(T^0)} = \sum_{j=1}^{N} \frac{\partial G(X(T))}{\partial X_j(T)} \frac{\partial X_j(T)}{\partial X_i(T^0)} = \sum_{j=1}^{N} \frac{\partial G(X(T))}{\partial X_j(T)} \Delta_{ij}(T^0, T) \tag{5.2}$$

where we have used notation $\Delta_{ij}(T^0, T^k) \triangleq \frac{\partial X_j(T^k)}{\partial X_i(T^0)}$

We rewrite (5.2) in vector matrix notation

$$\left[\frac{\partial G(X(T))}{\partial X(T^0)}\right]^{\mathrm{T}} = \left[\frac{\partial G(X(T))}{\partial X(T)}\right]^{\mathrm{T}} \cdot \Delta(T^0, T)$$

For simplicity we write the previous relationship in the following format, using the fact that $T^M = T$

$$\left[\frac{\partial G}{\partial X}[0]\right]^{\mathrm{T}} = \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot \Delta[M]$$

where the superscript ^{T} denotes the transpose

Differentiating (5.1) yields

$$\begin{aligned} \frac{\partial X\left(T^{k+1}\right)}{\partial X\left(T^k\right)} &= \frac{\partial \Phi\left(X(T^k), \Theta\right)}{\partial X\left(T^k\right)} \\ \Rightarrow \frac{\partial X\left(T^{k+1}\right)}{\partial X\left(T^0\right)} = \frac{\partial X\left(T^{k+1}\right)}{\partial X\left(T^k\right)} \frac{\partial X\left(T^k\right)}{\partial X\left(T^0\right)} &= \frac{\partial \Phi\left(X(T^k), \Theta\right)}{\partial X(T^k)} \frac{\partial X(T^k)}{\partial X(T^0)} = D[k] \frac{\partial X\left(T^k\right)}{\partial X\left(T^0\right)} \end{aligned} \tag{5.3}$$

where $D[k] \triangleq \frac{\partial \Phi\left(X(T^k),\Theta\right)}{\partial X\left(T^k\right)}$

We rewrite (5.3) as

$$\Delta[k+1] = D[k] \cdot \Delta[k]$$

Then we have an iterative process

$$\begin{aligned}
\left[\frac{\partial G}{\partial X}[0]\right]^{\mathrm{T}} &= \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot \Delta[M] = \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot D[M-1] \cdot \Delta[M-1] \\
&= \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot D[M-1] \cdot D[M-2] \cdot \Delta[M-2] = \dots \\
&= \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot D[M-1] \cdot \dots \cdot D[1] \cdot \Delta[1] \\
&= \left[\frac{\partial G}{\partial X}[M]\right]^{\mathrm{T}} \cdot D[M-1] \cdot \dots \cdot D[0] \cdot \Delta[0]
\end{aligned} \tag{5.4}$$

The matrix $\Delta[0]$ is the identity matrix, since

$$\Delta[0] = \left(\frac{\partial X_j(T^0)}{\partial X_k(T^0)}\right)_{jk} = \begin{pmatrix} 1 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & 1 & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{5.5}$$

The iterations in (5.4) employ matrix-matrix product. We will see later that the adjoint mode will involve matrix-vector product instead, which will offer important computational savings.

Now we move to the sensitivities with respect to model parameters

$$\frac{\partial G(X(T))}{\partial \theta_p} = \sum_{j=1}^{N} \frac{\partial G(X(T))}{\partial X_j(T)} \frac{\partial X_j(T)}{\partial \theta_p} = \sum_{j=1}^{N} \frac{\partial G(X(T))}{\partial X_j(T)} \Psi_{jp}(T) \tag{5.6}$$

where we have used notation $\Psi_{jp}(T^k) \triangleq \frac{\partial X_j(T^k)}{\partial \theta_p}$

Differentiating (5.1) with respect to parameters yields

$$\frac{\partial X\left(T^{k+1}\right)}{\partial \Theta} = \frac{\partial \Phi\left(X(T^k),\Theta\right)}{\partial X\left(T^k\right)} \frac{\partial X(T^k)}{\partial \Theta} + \frac{\partial \Phi\left(X(T^k),\Theta\right)}{\partial \Theta} \tag{5.7}$$

Making the notations $D[k] \triangleq \frac{\partial \Phi\left(X(T^k),\Theta\right)}{\partial X\left(T^k\right)}$ and $B[k] \triangleq \frac{\partial \Phi\left(X(T^k),\Theta\right)}{\partial \Theta}$ for the corresponding matrices, we rewrite (5.7) as

$$\Psi[k+1] = D[k] \cdot \Psi[k] + B[k]$$

Then we have an iterative process

$$\begin{aligned}
\left[\frac{\partial G(X(T))}{\partial \Theta}\right]^{\mathrm{T}} &= \left[\frac{\partial G(X(T))}{\partial X(T)}\right]^{\mathrm{T}} \cdot \Psi[M] = \left[\frac{\partial G(X(T))}{\partial X(T)}\right]^{\mathrm{T}} \cdot (D[M-1] \cdot \Psi[M-1] + B[M-1]) \\
&= \left[\frac{\partial G(X(T))}{\partial X(T)}\right]^{\mathrm{T}} \cdot (D[M-1] \cdot (D[M-1] \cdot \Psi[M-1] + B[M-1]) + B[M-1]) = \dots = \\
&= \left[\frac{\partial G(X(T))}{\partial X(T)}\right]^{\mathrm{T}} \cdot (B[M-1] + D[M-1] \cdot B[N-2] + \dots + D[M-1] \cdots D[1] \cdot B[0])
\end{aligned} \tag{5.8}$$

5.2 Adjoint (reverse) mode

We construct first the adjoint for computing the sensitivities with respect to initial condition. We start with the adjoint equation

$$V[k] = D^T[k] \cdot V[k+1] \tag{5.9}$$

where the superscript ^{T} denotes the transpose

In a recursive manner we obtain

$$V[0] = D^T[0] \cdot V[1] = D^T[0] \cdot D^T[1] \cdot V[2] = \cdots = D^T[0] \cdot D^T[1] \cdot \cdots D^T[M-1] \cdot V[M] \tag{5.10}$$

By taking transpose of (5.10) we have

$$V^T[0] = V^T[M] \cdot D[M-1] \cdot D[M-2] \cdot \cdots D[1] \cdot D[0] \tag{5.11}$$

We set $V[M]$ to the value $\left(\frac{\partial G}{\partial X}[M]\right)^T$ and we combine (5.11) and (5.4), which gives

$$\frac{\partial G}{\partial X}[0] = V^T[0] \cdot \Delta[0]$$

But the matrix $\Delta[0]$ is the identity matrix, according to 5.5

Thus we obtain the sensitivities with respect to initial conditions, namely $\frac{\partial G}{\partial X}[0]$ by applying the recursive relationship (5.11) to find $V^T[0]$. We note that the product in this iterative process is of the matrix-vector type, not matrix-matrix as it was for tangent linear mode

Now we move to sensitivities with respect to model parameters

We use again the adjoint equation (5.9). With same initial condition for $V[M]$, namely $V[M] = \left(\frac{\partial G}{\partial X}[M]\right)^T$, and evolving from time T^{M-k} to time T^M we have that

$$\frac{\partial G(X(T))}{\partial \Theta} \cdot D[M-1] \cdot D[M-2] \cdot \cdots D[M-k] \cdot B[M-k-1] = V[M-k]$$

Thus we can rewrite (5.8) as

$$\frac{\partial G(X(T))}{\partial \Theta} = \sum_{k=0}^{M-1} V^T[k+1] \cdot B[k] \tag{5.12}$$

The values of adjoint vectors $V[k]$ were computed as part of the adjoint approach for sensitivities with respect to initial conditions.

The values of $B[k]$ can be precomputed. We may be able to compute them analytically (e.g., if the parameters are volatilities and vol surface is assumed to have a certain parametric representation, such as cubic spline), otherwise we can employ the adjoint procedure.

6 Example in Monte Carlo framework (copula)

We follow a procedure similar to the one described in [20]

Let us describe the setup. We have a state vector $\mathbb{X}$ of N components, an instrument with a payoff $P(\mathbb{X})$, and the probability distribution $\mathbb{Q}(\mathbb{X})$ according to which the components of $\mathbb{X}$ are distributed. For simplicity we consider a N-dimensional Gaussian copula to model the co-dependence between the components of the state vector, namely a joint cumulative density function of the form

$$\Phi_N \left[\Phi_1 \left(\varphi_1 \left(X_1\right), \cdots, \varphi_N \left(X_N\right)\right); \rho\right]$$

where $\Phi_N[Z_1, \cdots, Z_N; \rho]$ is a N-dimensional multivariate Gaussian distribution with zero mean and a correlation matrix ρ, Φ^{-1} is the inverse of the standard normal cumulative distribution and $\varphi_i(X_i)$, i=1...N are the marginal distributions of the underlying components.

The option value is obtained through averaging of Monte Carlo sampling

$$V = \frac{1}{N_{MC}} \sum_{j=1}^{N_{MC}} P\left(\mathbb{X}^{(i)}\right)$$

where the superscript $^{(i)}$ denotes the $i-$th Monte Carlo path.

The sampling of the N jointly distributed normal random variables $(Z_1, Z_2, \cdots, Z_N)$ can be done using several approaches, such as Cholesky factorization, spectral or singular value decomposition. We select Cholesky factorization

$\rho = C \cdot C^{\mathrm{T}}$ because it will enable us to use an existing procedure for its adjoint. Starting from a vector $\hat{Z}$ of independent standard normal variables, we obtain a vector $Z$of jointly normal random variables distributed according to $\Phi_N[Z_1, \cdots, Z_N; \rho]$ through the product

$$Z = C \cdot \hat{Z}$$

We also use the following:

- if Z_i is sampled from a standard normal distribution then $\Phi(Z_i)$ is in $\mathcal{U}$ [0,1]
- if X_i is distributed according to marginal distribution φ_i, then $\varphi(X_i)$ is in $\mathcal{U}$ [0,1]

Then $X_i = \varphi^{-1}(\Phi(Z_i))$ is distributed according to marginal distribution φ_i

Therefore we have the following algorithm [20]

1. Generate a vector Ξ of independent standard normal variables
2. Correlate the components through $Z = C \cdot \Xi$
3. Set $U_i = \Phi(Z_i)$, $i = 1...N$
4. Set $X_i = \varphi_i^{-1}(\Phi(Z_i)) = \varphi^{-1}(U_i)$, $i = 1...N$
5. Calculate the payoff estimator $P(X_1, ..., X_N)$

We now show how sensitivities can be computed in this setup.

The correlation matrix is an input to the procedure, so we may compute correlation risk, i.e., sensitivities of the price with respect to entries in the correlation matrix

These marginal distributions may be obtained from a set of N^{MARG} discrete call, put, digital call and digital put values (which may be given as corresponding implied volatilities).We may also compute sensitivities with respect to those inputs, denoted by $\varpi_j, j = 1...N^{MARG}$

6.1 Forward (tangent linear) mode

Assuming that the payoff function is regular enough (e.g., Lipschitz continuous) the standard pathwise differentiation corresponds to forward (tangent linear) mode. The differentiation is applied to steps 1-5 in the above algorithm. We need to pay attention if any given step is dependent (implicitly or explicitly) on the input parameters with respect to which we want to compute the sensitivities. Step 1 stays the same.

Step 2 and 3 are differentiated if we want correlation risk, otherwise they remain unchanged. Steps 4 and 5 are differentiated regardless which of the two types of sensitivities of sensitivities we want to compute.

To differentiate Step 4 we start from $X_i = \varphi_i^{-1}(U_i) \Rightarrow U_i = \varphi_i(X_i)$.

Differentiating the last equality gives the following formula, if we have the propagated derivative $\dot{U}_i$ of the variable U_i (i.e., we compute the correlation risk)

$$\dot{U}_i = \frac{\partial \varphi_i}{\partial x}(X_i)\,\dot{X}_i \Rightarrow \dot{X}_i = \frac{\dot{U}_i}{\frac{\partial \varphi_i}{\partial x}(X_i)}$$

If we need to compute sensitivities with respect to ϖ_j, then differentiating the same equality gives

$$\dot{X}_i = \frac{\dot{\varpi}_j}{\frac{\partial \varphi_i}{\partial x}(X_i)}$$

We now present the algorithm for tangent linear mode. For ease of presentation we write explicitly the algorithm only for the case of correlation sensitivities.

We assume that we want to compute the sensitivity with respect to entry ρ_{lk} of the correlation matrix

1. Generate a vector Ξ of independent standard normal variables
2. Calculate $\dot{Z} = \dot{C}_{lk} \cdot \dot{\Xi}$, where $\dot{C}_{lk}$ is the sensitivity of Cholesky factor C with respect to ρ_{lk}
3. Set $\dot{U}_i = \frac{\partial \Phi}{\partial x}(Z_i),\ i = 1...N$
4. Set $\dot{X}_i = \frac{\dot{U}_i}{\frac{\partial \varphi_i}{\partial x}(X_i)},\ i = 1...N$
5. Calculate $\dot{P} = \sum_{i=1}^{N} \frac{\partial P}{\partial X_i}(X_i) \cdot \dot{X}_i$

We note that the derivative of the marginal distribution, denoted by $\frac{\partial \varphi_i}{\partial x}(X_i)$, is the probability density function associated with the marginal distribution φ_i, while the derivative $\frac{\partial \Phi}{\partial x}$ is the standard normal probability density function

6.2 Adjoint (reverse) mode

The adjoint mode consists of the adjoint counterparts for each of the steps in the forward mode, plus the adjoint of Cholesky factorization [63]. The computation is done in reverse order

The resulting algorithm consists of the 5 steps described above (in the forward mode) plus the following steps corresponding to adjoint counterparts

1. Calculate the adjoint of the payoff estimator $\bar{X}_k = \frac{\partial P}{\partial X_k}(X_k),\ k = 1...N$
2. Calculate $\bar{U}_k = \frac{\bar{X}_k}{\frac{\partial \varphi_i}{\partial x}\left(\varphi_k^{-1}(U_k)\right)},\ k = 1...N$
3. Calculate $\bar{Z}_k = \bar{U}_k \frac{\partial \Phi}{\partial x}(Z_k),\ k = 1...N$
4. Calculate $\bar{C} = \bar{Z} \cdot \Xi^{\mathrm{T}}$

The matrix $\bar{C} = \left(\bar{C}_{ij}\right)_{i,j=1..N}$ obtained at the end of the procedure contains all derivatives of payoff estimator with respect to entries ϱ_{ij} of the correlation matrix. We can see that all these sensitivities were computed by running the adjoint mode only once, as opposed to the forward (tangent linear) mode, which had to be run separately for each entry in the correlation matrix (with a total number of runs of N^2)

7 Example in calibration/optimization framework

Let us consider that we have the same SDE as in the previous chapter. We want to minimize a cost functional of the type

$$F = \sum \left(ModelPrice[j] - MarketPrice[j] \right)^2$$

with variables to be optimized being given by model parameters $\Theta = \{\theta_1, ..., \theta_P\}$

The gradient of the cost functional with respect to the model parameters would have the expression

$$\begin{pmatrix} \cdots \\ \frac{\partial F}{\partial \theta_k} \\ \cdots \end{pmatrix} = \begin{pmatrix} \cdots \\ 2 \sum \left(ModelPrice[j] - MarketPrice[j] \right) \frac{\partial (ModelPrice[j])}{\partial \theta_k} \\ \cdots \end{pmatrix}$$

The adjoint procedure enables us to compute $\frac{\partial (ModelPrice[j])}{\partial \theta_k}$ in a similar way to (5.12)

We note here that the above procedure assumes implicitly the existence of the gradient of the functional. It is our experience [47]that the discrete adjoint can still be applied for cases such gradient doe not exist; in those cases the numerical adjoint code will provide us not the gradient, but rather subgradients. Consequently, one will have to employ optimization algorithms that are especially designed to use subgradients instead of gradient

8 Computational finance literature on adjoint and AD

In the last several years quite a few papers were added to the literature on adjoint/AD applied to computational finance [11, 22, 20, 21, 24, 23, 32, 30, 31, 41, 50, 48, 49, 54, 52, 55, 56, 65, 3, 9, 18, 67] . For selected papers we give an overview in the following sections

8.1 Computation of Greeks

A major part of the literature related to this topic is due to Giles, Capriotti and, respectively, Joshi (and their collaborators).

The seminal paper of [41] applied the adjoint approach to the computation of Greeks (Delta and Vega) for swaptions using pathwise differentiation method in the context of LIBOR Market Model (LMM). The portfolio considered had portfolio had 15 swaptions all expiring at the same time, N periods in the future, involving payments/rates over an additional 40 periods in the future. Interested in computing Deltas, sensitivity to initial N+40 forward rates, and Vegas, sensitivity to initial N+40 volatilities. The computational efficiency was improved by a factor of 10 when the forward method was employed instead of finite differences. Then the adjoint approach reduced the cost by several orders of magnitude, compared to the forward approach: for N=80 periods, by a factor of 5 for computation of Deltas only, and respectively by a factor of 25 for computation of both Deltas and Vegas.

This was extended in [58] to the pricing of Bermudan-style derivatives. For testing they have used five 1xM receiver Bermudan swaptions (M = 4, 8, 12, 16, 20) with half-year constant tenor distances. The speedup was as follows (for M=20): by a factor of 4 for only Deltas, and by factor of 10 for both Deltas and Vegas.

The pathwise approach is not applicable when the financial payoff function is not differentiable. To address these limitations, a combination the adjoint pathwise approach for the stochastic path evolution with likelihood ratio method (LRM) for the payoff evaluation is presented in [36, 38]. This combination

is termed "Vibrato" Monte Carlo. The Oxford English Dictionary describes "vibrato" as "a rapid slight variation in pitch in singing or playing some musical instruments". The analogy to Monte Carlo methods is the following; whereas a path simulation in a standard Monte Carlo calculation produces a precise value for the output values from the underlying stochastic process, in the vibrato Monte Carlo approach the output values have a narrow probability distribution.

Applying concepts of adjoint/AD for correlation Greeks were considered in [20]. The pricing of an instrument based on N underlyings is done with Monte Carlo within a Gaussian copula framework, which connects the marginal distributions of the underlying factors. The sampling of the N jointly normal random variables is efficiently implemented by means of a Cholesky factorization of the correlation matrix. Correlation risk is obtained in a highly efficient way by implementing the pathwise differentiation approach in conjunction with AD, using the adjoint of Cholesky factorization. Numerical tests on basket default options shows a speedup of 1-2 orders of magnitude (100 times faster than bumping for 20 names, and 500 times for 40 names).

Examples of pathwise differentiation combined with AD are shown in [18]. For a basket option priced in a multidimensional lognormal model, the savings are already larger than one order of magnitude for medium sized baskets (N=10). For "Best of" Asian options in a local volatility setting, the savings are reported to over one order of magnitude for a relatively small number (N=12) of underlying assets.

Adjoint algorithmic differentiation can be used to implement efficiently the calculation of counterparty credit risk [22]. Numerical results show a reduction by more than two orders of magnitude in the computational cost of Credit Valuation Adjustment (CVA). The example considered is a portfolio of 5 swaps referencing distinct commodities Futures with monthly expiries with a fairly typical trade horizon of 5 years, the CVA bears non-trivial risk to over 600 parameters: 300 Futures prices, and at the money volatilities, (say) 10 points on the zero rate curve, and 10 points on the Credit Default Spread(CDS) curve of the counterparty used to calibrate the transition probabilities of the rating transition model. required for the calculation of the CVA. The computational time for CVA sensitivities is less than 4 times the computational time time spent for the computation of the CVA alone, as predicted by AD theory. As a result, even for this very simple application, adjoint/AD produces risk over 150 times faster than finite differences: for a CVA evaluation taking 10 seconds, adjoint approach produces the full set of sensitivities in less than 40 seconds, while finite differences require approximately 1 hour and 40 minutes.

In the framework of co-terminal swap-rate market model [51] presented an efficient algorithm to implement the adjoint method that computes sensitivities of an interest rate derivative (IRD) with respect to different underlying rates, yielding a speedup of a factor of 15 for Deltas of a Bermudan callable swaption with rates driven by a 5-factor Brownian motion. They show a total computational order of the adjoint approach of order $O(nF)$, with n the number of co-terminal swap rates, assumed to be driven by number$F$ Brownian motions, compared to the order of standard pathwise method which is $O(n^3)$. It was extended to Vegas in [55], in the context of three displaced diffusions swap-rate market models. A slight modification of the method of computing Deltas in generic market models will compute market Vegas with order $O(nF)$ per step. Numerical results for the considered tests show that computational cots for Deltas and Vegas will be not more than 1.5 times the computational cots of only Deltas. CMS spread options in the displaced-diffusion co-initial swap market model are priced and hedged in [53]. The evolution of the swap-rates is based on an efficient two-factor predictor-corrector(PC) scheme under an annuity measure. Pricing options using the new method takes extremely low computational times compared with traditional Monte Carlo simulations. The additional time for computing model Greeks using the adjoint method can be measured in milliseconds. Mostly importantly, the enormous reduction in computational times does not come at a cost of poorer accuracy. In fact, the prices and Greeks produced by the new method are sufficiently close to those produced by a one-step two-factor PC scheme with 65,535 paths.

In the framework of the displaced-diffusion LIBOR market model, [49] compared the discretization bias obtained when computing Greeks with pathwise adjoint method for the iterative predictor-corrector and Glasserman-Zhao drift approximations in the spot measure to those obtained under the log-Euler and predictor-corrector approximations by performing tests with interest rate caplets and cancellable receiver swaps. They have found the iterative predictor-corrector method to be more accurate and slightly faster than the predictor-corrector method, the Glasserman-Zhao method to be relatively fast but highly inconsistent, and the log-Euler method to be reasonably accurate but only at low volatilities.

In the framework of the cross-currency displaced-diffusion LIBOR market model, [11] employs adjoint techniques to compute Greeks for two commonly traded cross-currency exotic interest rate derivatives: cross-currency swaps (CCS) and power reverse dual currency (PRDC) swaps. They measure the computational times relative to the basic implementation of the crosscurrency LIBOR market model, that is with standard least squares regression used to compute the exercise strategy. It was reported that, using the adjoint pathwise method, the computational time for all the Deltas and Vega for each step and the exchange Vega is only slightly larger compared to the computational time required to compute one Delta using finite differences.

Adjoint approach was applied for computation of higher order Greeks (such as Gammas) in [52] and [54], where they show that the Gamma matrix (i.e. the Hessian) can be computed in AM + B times the number of operations where M is the maximum number of state variables required to compute the function, and A,B are constants that only depend on the set of floating point operations allowed.. In the first paper numerical results demonstrate that the computing all $n(n+1)/2$ Gammas for Bermudan cancellable swaps in the LMM takes roughly $n/3$ times as long as computing the price. In the second paper numerical results are shown for an equity tranche of a synthetic CDO with 125 names. To compute all the 125 finite-difference Deltas, the computational time for doing so will be 125 times of the original pricing time (if we use forward difference estimates). However, it only takes up to 1.49 times of the original pricing time if we use algorithmic differentiation. There are 7,875 Gammas to estimate. If one uses central difference estimates, it will take 31,250 times of the original pricing time to achieve this. It only takes up to 195 times to estimate all the Gammas with algorithmic Hessian methods.

In the framework of Markov-functional models, [30] demonstrates how the adjoint PDE method can be used to compute Greeks. This paper belongs to the ContAdj category. It obtains an adjoint PDE, which is solved once to obtain all of the model Greeks. In the particular case of a separable LIBOR Markov-functional model the price and 20 Deltas, 40 mapping Vegas and 20 skew sensitivities (80 Greeks in total) of a 10-year product are computed in approximately the same time it takes to compute a single Greek using finite difference. The instruments considered in the paper were: interest rate cap, cancellable inverse floater swap and callable Bermudan swaption.

In the framework of Heston model, the algorithmic differentiation approach was considered in [24] to compute the first and the second order price sensitivities.. Issues related to the applicability of the pathwise method are discussed in this paper as most existing numerical schemes are not Lipschitz in model inputs. While AD/adjoint approach is usually considered for primarily providing a very substantial reduction in computational time of the Greeks, this paper also shows how its other major characteristic, namely accuracy, can be extremely relevant in some cases. Computing price sensitivities is done using the Lognormal (LN) scheme, the Quadratic- Exponential (QE) scheme, the Double Gamma (DG) scheme and the Integrated Double gamma (IDG) scheme. Numerical tests show that the sample means of price sensitivities obtained using the Lognormal scheme and the Quadratic-Exponential scheme can be highly skewed and have fat-tailed distribution while price sensitivities obtained using the Integrated Double Gamma scheme and the Double Gamma scheme remain stable.

The adjoint/AD approach is also mentioned in the very recent "opus magna" on interest rate derivatives

[5, 6, 7], written by 2 well-known practitioners.

8.2 Calibration

To the best of our knowledge, [3] was the first time that an adjoint approach was presented for calibration in the framework of computational finance

In [57] adjoint methods are employed to speed up the Monte Carlo-based calibration of financial market models. They derive the associated adjoint equation /(and thus are in ContAdj category) and propose its application in combination with a multi-layer method. They calibrate two models: the Heston model is used to confirm the theoretical viability of the proposed approach, while for a lognormal variance model the adjoint-based Monte Carlo algorithm reduces computation time from more than three hours (for finite difference approximation of the gradient)to less than ten minutes

The adjoint approach is employed in [59, 60] to construct a volatility surface and for calibration of local stochastic volatility models. The fitting of market bid/ask-quotes is not a trivial task, since the quotes differ in quality as well as quantity over timeslices and strikes, and are furthermore coupled by hundreds of arbitrage constraints. Nevertheless, the adjoint approach enabled on the fly fitting (less than 1 second) for real-life situations: a volatility matrix with 17 strikes and 8 maturities.

AD is applied to calibration of Heston model in[44], through differentiation of vanilla pricing using characteristic functions. It takes advantage of the fact that it can be applied to complex numbers (if they are "equipped" with auto-differentiation). Applying AD to numerical integration algorithm is straightforward, and the resulting algorithm is fast and accurate.

In [69] the adjoint approach (of ContAdj type) is applied to calibrate a local volatility model

Calibration of a local volatility model is also shown in [65]

9 AD and adjoint applied within a generic framework in practitioner quant libraries

As more financial companies become familiarized with the advantages offered adjoint/AD approach when applied to computational finance, this approach will be more and more integrated in their quant libraries. Very recent presentations [19, 17, 44, 60, 10] at major practitioner quant conferences indicate that this effort is under way at several banks, such as Credit Suisse, Unicredit, Nomura, etc. Within this framework one may imagine a situation where Greeks are computed (using AD techniques) in real time to provide hedging with respect to various risk factors: interest rate, counterparty credit, correlation, model parameters etc.

Due to complexity of quant libraries, AD tools and special techniques of memory management, checkpointing etc can prove extremely useful, potentially leveraging knowledge acquired during AD applications to large software packages in other areas: meteorology, computational fluid dynamics etc where such techniques were employed [43, 25, 45, 4, 39, 15, 8]

9.1 Block architecture for Greeks

[10] presents a "Block architecture for Greeks" , using calculators and allocators. Sensitivities are computed for each block component, including root-finding and optimization routines, trees and latices.

Here is an example included in the presentation. First some notations: every individual function (routine), say $f : \mathbb{R}^{\mathrm{m}} \to \mathbb{R}^{\mathrm{n}}$ y=f(x)is termed a "calculator" function, and a corresponding "allocator" function is defined as

$$G_f : \mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}} \quad \mathbb{G}_{\mathrm{f}}\left(\frac{\partial V}{\partial y}\right) = \frac{\partial V}{\partial y} \cdot \mathrm{J}_f$$

If we have a sequence of functions f, g, h, then we calculate the Greeks by starting with the end point $\frac{\partial V}{\partial V} = 1$ and working backwards by calling the allocator functions one-by-one in reverse order to the original order of the function calls.

Suppose we have a pricing routine with three subroutines

- f : Interpolates market data onto the relevant grid from input market objects
- g : performs some calibration step on the gridded market data
- h : core pricer which uses the calibrated parameters

We need to write the three risk allocator functions for these three subroutines. Then we feed the starting point $\frac{\partial V}{\partial V} = 1$ into the pricer's allocator to get back the vector of risks to the calibrated parameters. The next step is to feed this into the calibrator's allocator to get back the vector of risks to the gridded market data. Finally, we feed that risk vector into the interpolator's allocator to get back risk to the original input market objects.

9.2 Real Time Counterparty Risk Management in Monte Carlo

[19] presents a framework for Real Time Counterparty Risk Management, employing AD to compute Credit Valuation Adjustment (CVA), with a reduction in computational time from 100 min (using bump and reprice) to 10 seconds, for a a portfolio of 5 commodity swaps over a 5 years horizon (over 600 risks)

10 Additional resources

Additional resources include reference books [13, 16, 26, 43], results on matrix differentiation [37, 62, 63], the online community portal for Automatic Differentiation [2], various AD tools such as FASTOPT, FASTBAD, ADMAT, FAD, ADOL-C, RAPSODIA, TAPENADE [1], papers on rules to construct the adjoint code [43, 12, 34, 35, 28, 29, 14, 3, 64]

11 Conclusion

We have presented an overview of adjoint and automatic(algorithmic) differentiation techniques in the framework of computational finance, and illustrated the major advantages of this approach for computation of sensitivities: great reduction of computational time, improved accuracy, and a systematic process to construct the corresponding "adjoint" code from existing codes

References

[1] Automatic Differentiation tools. http://www.autodiff.org/?module=Tools.

[2] Community portal for Automatic Differentiation. http://www.autodiff.org.

[3] Y. Achdou and O. Pironneau. *Computational methods for option pricing*. SIAM, 2005.

[4] M. Alexe, O. Roderick, J. Utke, M. Anitescu, P. Hovland, and T. Fanning. Automatic differentiation of codes in nuclear engineering applications. Technical report, Mathematics and Computer Science Division. Argonne National Lab, 2009.

[5] L.B.G. Andersen and V. V. Piterbarg. *Interest Rate Modeling Volume I: Foundations and Vanilla Models*. Atlantic Financial Press, 2010.

[6] L.B.G. Andersen and V. V. Piterbarg. *Interest Rate Modeling Volume II: Term Structure Modeling*. Atlantic Financial Press, 2010.

[7] L.B.G. Andersen and V. V. Piterbarg. *Interest Rate Modeling Volume III: Products and Risk Management*. Atlantic Financial Press, 2010.

[8] R.A. Bartlett, D.M. Gay, and E.T. Phipps. Automatic Differentiation of C++ codes for large-scale scientific computing. In V.N. Alexandrov, G.D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *Computational Science – ICCS 2006*, volume 3994 of *Lecture Notes in Computer Science*, pages 525–532, Heidelberg, 2006. Springer.

[9] H. Bastani and L. Guerrieri. On the application of Automatic Differentiation to the likelihood function for dynamic general equilibrium models. http://www.federalreserve.gov/pubs/ifdp/2008/920/ifdp920.pdf, 2008.

[10] M. Baxter. Practical implementation of fast Greeks in your analytics library. In *The 6th Fixed Income Conference, Madrid, September 22-24*, 2010.

[11] C. Beveridge, M. Joshi, and W. M. Wright. Efficient pricing and Greeks in the cross-currency LIBOR market model. http://ssrn.com/paper=1662229, 2010.

[12] C. Bischof, B. Lang, and Andre Vehreschild. Automatic differentiation for matlab programs. *Proceedings in Applied Mathematics and Mechanics*, 2(1):50–53, 2003.

[13] C. H. Bischof, H. M. Bücker, P. Hovland, and U. Naumann. *Advances in Automatic Differentiation*. Springer, 2008.

[14] C.H. Bischof. Automatic differentiation, tangent linear models and pseudo-adjoints. In François-Xavier Le Dimet, editor, *High-Performance Computing in the Geosciences*, volume 462 of *Mathematical and Physical Sciences*, pages 59–80. Kluwer Academic Publishers, Boston, Mass., 1995.

[15] C.H. Bischof, H.M. Bücker, B. Lang, A. Rasch, and E. Slusanschi. Efficient and accurate derivatives for a software process chain in airfoil shape optimization. *Future Generation Computer Systems*, 21(8):1333–1344, 2005.

[16] H.M Bucker, G. Corliss, P. Hovland, and U. Naumann. *Automatic Differentiation: Applications, Theory, and Implementations*. Springer, 2006.

[17] L. Capriotti. Adjoint algorithmic differentiation: a new paradigm in risk management. In *Quant Congress Europe, London, Nov 9-11*, November 2010.

[18] L. Capriotti. Fast Greeks by algorithmic differentiation. *Journal of Computational Finance*, 14(3):3–35, 2011.

[19] L. Capriotti. Making the calculation of risk through Monte Carlo methods more efficient by using adjoint algorithmic differentiation. In *Global Derivatives Trading & Risk Management, Paris, April 11-15*, 2011.

[20] L. Capriotti and M.B. Giles. Fast correlation Greeks by adjoint algorithmic differentiation. *RISK*, March, 2010.

[21] L. Capriotti and M.B. Giles. Algorithmic differentiation: Adjoint Greeks made easy. http://ssrn.com/paper=1801522, 2011.

[22] L. Capriotti, S. Lee, and M. Peacock. Real time counterparty credit risk management in Monte Carlo. http://ssrn.com/paper=1824864, 2011.

[23] J.H. Chan and M. Joshi. Fast Monte-Carlo Greeks for financial products with discontinuous pay-offs. http://ssrn.com/paper=1500343, 2009.

[24] J.H. Chan and M. Joshi. First and second order Greeks in the Heston model. http://ssrn.com/paper=1718102, 2010.

[25] I. Charpentier and M. Ghemires. Efficient adjoint derivatives: application to the meteorological model Meso-NH. *Optim. Methods Software*, 13:35–63, 2000.

[26] G Corliss, Christele Faure andAndreas Griewank, and Laurent Hascoet. *Automatic Differentiation of Algorithms*. Springer, 2002.

[27] P. Courtier and O. Talagrand. Variational assimilation of meteorological observations with the adjoint vorticity equation, Ii, numerical results. *Q. J. R. Meteorol. Soc.,*, 113:1329–1347, 1987.

[28] P. Cusdin and J.-D. Müller. Deriving linear and adjoint codes for CFD using Automatic Differentiation. Technical Report QUB-SAE-03-06, QUB School of Aeronautical Engineering, 2003. Submitted to AIAA http://www.ea.qub.ac.uk/pcusdin/index.php.

[29] P. Cusdin and J.-D. Müller. Generating efficient code with Automatic Differentiation. In *European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS*, 2004.

[30] N Denson and Joshi. Fast Greeks for Markov-functional models using adjoint PDE methods. http://ssrn.com/paper=1618026, 2010.

[31] N Denson and M Joshi. Fast and accurate Greeks for the LIBOR market model. http://ssrn.com/paper=1448333, 2009.

[32] N Denson and M Joshi. Flaming logs. *Wilmott Journal*, 1:259–262, 2009.

[33] F. X. Le Dimet, I. M. Navon, and D. N. Daescu. Second order information in data assimilation. *Monthly Weather Review*, 130(3):629–648, 2002.

[34] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Transactions on Mathematical Software*, 24:437–474, 1998.

[35] R. Giering and T. Kaminski. *Recomputations in reverse mode AD*, chapter Automatic differentiation of algorithms. Springer, 2002.

[36] M.B. Giles. Monte Carlo evaluation of sensitivities in computational finance. In *HERCMA*, 2007.

[37] M.B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Lecture Notes in Computational Science and Engineering, Volume 64*, pages 35–44. Springer, 2008.

[38] M.B. Giles. Vibrato Monte Carlo sensitivities. In *Monte Carlo and Quasi Monte Carlo Methods 2008*, pages 369–382. Springer, 2008.

[39] M.B. Giles, D. Ghate, and M.C. Duta. Using automatic differentiation for adjoint CFD code development. Technical report, Oxford University Computing Laboratory,, 2005.

[40] M.B. Giles, D. Ghate, and M.C. Duta. Using automatic differentiation for adjoint CFD code development. In *Recent Trends in Aerospace Design and Optimization*. Tata McGraw-Hill, New Delhi, 2006.

[41] M.B. Giles and P. Glasserman. Smoking adjoints: fast Monte Carlo Greeks. *RISK*, January, 2006.

[42] M.B. Giles and N.A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Control*, 65(3-4):393–415, 2000.

[43] A. Griewank and A. Walther. *Evaluating derivatives : principles and techniques of algorithmic differentiation, 2nd edition*. SIAM, 2008.

[44] J Hakala. Auto-differentiation in practice. In *Global Derivatives Trading & Risk Management, Paris, April 11-15*, 2011.

[45] L. Hascoet and B. Dauvergne. Adjoints of large simulation codes through automatic differentiation. *REMN Revue Europeenne de Mecanique Numerique / European Journal of Computational Mechanics*, pages 63–86, 2008.

[46] C. Homescu and I. M. Navon. Numerical and theoretical considerations for sensitivity calculation of discontinuous flow. *Systems & Control Letters on Optimization and Control of Distributed Systems*, 48(3):253–260, 2003.

[47] C. Homescu and I. M. Navon. Optimal control of flow with discontinuities. *Journal of Computational Physics*, 187:660–682, 2003.

[48] M Joshi and D Pitt. Fast sensitivity computations for Monte Carlo valuation of pension funds. http://ssrn.com/paper=1520770, 2010.

[49] M Joshi and A Wiguna. Accelerating pathwise Greeks in the LIBOR market model. http://ssrn.com/paper=1768409, 2011.

[50] M Joshi and C Yang. Efficient Greek estimation in generic market models. http://ssrn.com/paper=1437847, 2009.

[51] M Joshi and C Yang. Fast Delta computations in the swap-rate market model. http://ssrn.com/paper=1401094, 2009.

[52] M Joshi and C Yang. Algorithmic Hessians and the fast computation of cross-Gamma risk. http://ssrn.com/paper=1626547, 2010.

[53] M. Joshi and C. Yang. Fast and accurate pricing and hedging of long-dated CMS spread options. *International Journal of Theoretical and Applied Finance*, 13(6):839–865, 2010.

[54] M Joshi and C Yang. Fast Gamma computations for CDO tranches. http://ssrn.com/paper=1689348, 2010.

[55] M Joshi and C Yang. Efficient Greek estimation in generic swap-rate market models. http://ssrn.com/paper=1773942, 2011.

[56] C Kaebe. *Feasibility and Efficiency of Monte Carlo Based Calibration of Financial Market Models*. PhD thesis, University of Trier, 2010.

[57] C. Kaebe, J.H. Maruhn, and E.W. Sachs. Adjoint-based Monte Carlo calibration of financial market models. *Finance and Stochastics*, 13:351–379, 2009.

[58] M. Leclerc, Q. Liang, and I. Schneider. Fast Monte Carlo Bermudan Greeks. *Risk*, July:84–88, 2009.

[59] J. Maruhn. On-the-fly bid/ask-vol fitting with applications in model calibration. In *SIAM Conference on Financial Mathematics & Engineering, San Francisco, USA, November 19-20*, 2010.

[60] J. Maruhn. Calibration of stochastic and local stochastic volatility models. In *Global Derivatives Trading & Risk Management, Paris, April 11-15*, 2011.

[61] I. M. Navon, X. Zou, J. Derber, and J. Sela. Variational data assimilation with an adiabatic version of the NMC spectral model. *, Monthly Weather Review*, 120(7):1433–1446, 1992.

[62] K.B. Petersen and M.S. Pedersen. Matrix cookbook. http://www.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf, 2008.

[63] S. Smith. Differentiation of the Cholesky algorithm. *Journal of Computational and Graphical Statistics*, 4(2):134–147, 1995.

[64] S.P. Smith. A tutorial on simplicity and computational differentiation for statisticians. http://hep.physics.ucdavis.edu/~jrsmith/backwards_differentiation/nonad3.pdf, 2000.

[65] J. Spilda. Adjoint methods for computing sensitivities in local volatility surfaces. Master's thesis, University of Oxford, 2010.

[66] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 10(1):110–112, 1998.

[67] B. Stehle. Proxy scheme and Automatic Differentiation: Computing faster Greeks in Monte Carlo simulations. Master's thesis, Imperial College, 2010.

[68] O. Talagrand and P. Courtier. Variational assimilation of meteorological observations with the adjoint vorticity equation, I, Theory. *Q. J. R. Meteorol. Soc.*, 113:1311–1328, 1987.

[69] G. Turinici. Calibration of local volatility using the local and implied instantaneous variance. *Journal of Computational Finance*, 13(2):1–18, 2009.