Algorithmic Differentiation in Finance

Fast Greeks and Beyond

Marc Henrard

Quantitative Research - OpenGamma

OpenGamma Webinar, December 12th, 2012

About OpenGamma

- Open source trading and risk analytics platform for the financial services industry
- Completely modular architecture for custom application development
- Available under Apache License 2.0
- 0penGamma Analytics
  - Cutting-edge analytics library
  - Large range of asset classes, models, numerical techniques

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Introduction

- Research and development time: price.
- CPU time: greeks (derivatives with respect to the input).
- In computer science the art of calculating derivatives is known as Algorithmic Differentiation.
- Algorithmic Differentiation comes in two modes:
  1. Forward/standard
  2. Reverse/adjoint
- The Adjoint mode is often the most efficient in finance.
- The theoretical efficiency can be achieve in practice in a quantitative finance library if it is built into its structure.
- The theoretical efficiency can be improved in practice in specific cases important in finance.

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

Mathematics (1): Differentiation ratio

The derivatives are often computed through the ratio:

Approximation (Differentiation ratio)

One sided:

$$D_{x_i} f(x) \simeq \frac{f(x + \epsilon_i) - f(x)}{\epsilon_i}.$$

Two sided (or symmetrical):

$$D_{x_i} f(x) \simeq \frac{f(x + \epsilon_i) - f(x - \epsilon_i)}{2\epsilon_i}.$$

Note that the approximation converges for $\epsilon \to 0$ only with infinitely precise arithmetic.

Mathematics (2): Chain rule

The main piece of mathematics for Algorithmic Differentiation is

Theorem (Chain rule)

For two differentiable functions f and g, one has

$$D(f \circ g)(x) = Df(g(x)) \cdot Dg(x).$$

Library: function - method

The starting point is the algorithm for

$$z = f(a).$$

The function inputs are: $a = a[0 : p_a]$ (dimension $p_a + 1$).
The function output is z (dimension 1).

The code is

Initialisation $[j = -p_a : 0]$ $b[j] = a[j + p_a]$

Library: function - method

The starting point is the algorithm for

$$z = f(a).$$

The function inputs are: $a = a[0 : p_a]$ (dimension $p_a + 1$).
The function output is z (dimension 1).

The code is

Initialisation	$[j = -p_a : 0]$	$b[j] = a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j] = g_j(b[-p_a : j - 1])$

Library: function - method

The starting point is the algorithm for

$$z = f(a).$$

The function inputs are: $a = a[0 : p_a]$ (dimension $p_a + 1$).
The function output is z (dimension 1).

The code is

Initialisation	$[j = -p_a : 0]$	$b[j] = a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j] = g_j(b[-p_a : j - 1])$
Value		$z = b[p_b]$

This algorithm is supposed to be implemented.

Library: Algorithmic Differentiation

Goal: compute the derivatives of z with respect to a_i:

$$\frac{\partial}{\partial a_i} f(a) = \frac{\partial}{\partial a_i} z.$$

The emphasis can be put on *with respect to* a_i (standard) or on *of* z (adjoint).
The forward or standard mode puts the emphasis on *with respect to* a_i; it is not discussed in this presentation.

Computer: AD reverse/adjoint

Goal: compute the derivatives of z: $\frac{\partial}{\partial b[j]} z = \bar{b}[j]$.

Init	$[j = -p_a : 0]$	$b[j] = a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j] = g_j(b[-p_a : j - 1])$
Value		$z = b[p_b]$

OpenGamma

Computer: AD reverse/adjoint

Goal: compute the derivatives of z: $\frac{\partial}{\partial b[j]} z = \bar{b}[j]$.

Init	$[j = -p_a : 0]$	$b[j] = a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j] = g_j(b[-p_a : j-1])$
Value		$z = b[p_b]$

Value	$\bar{z}$	$= 1.0$
Value	$\bar{b}[p_b]$	$= 1.0$
		$=$

OpenGamma

Computer: AD reverse/adjoint

Goal: compute the derivatives of z: $\frac{\partial}{\partial b[j]} z = \bar{b}[j]$.

Init	$[j = -p_a : 0]$	$b[j] = a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j] = g_j(b[-p_a : j-1])$
Value		$z = b[p_b]$

Value		$\bar{z} = 1.0$
Value		$\bar{b}[p_b] = 1.0$
Algorithm	$[j = p_b - 1 : -1 : -p_a]$	$\bar{b}[j] = \sum_{k=j+1}^{p_b} \frac{\partial}{\partial b_k} z \frac{\partial}{\partial b_j} b_k$
		$= \sum_{k=j+1}^{p_b} \bar{b}[k] \frac{\partial}{\partial b_j} g_k$
Init	$[i = 0 : p_a]$	$\frac{\partial}{\partial a_i} z = \bar{b}[i - p_a + 1]$

OpenGamma

Computer: AD reverse/adjoint

Goal: compute the derivatives of z: $\frac{\partial}{\partial b[j]} z = \bar{b}[j]$.

Init	$[j = -p_a : 0]$	$b[j]$	$= a[j + p_a]$
Algorithm	$[j = 1 : p_b]$	$b[j]$	$= g_j(b[-p_a : j-1])$
Value		z	$= b[p_b]$

Value		$\bar{z}$	$= 1.0$
Value		$\bar{b}[p_b]$	$= 1.0$
Algorithm	$[j = p_b - 1 : -1 : -p_a]$	$\bar{b}[j]$	$= \sum_{k=j+1}^{p_b} \frac{\partial}{\partial b_k} z \frac{\partial}{\partial b_j} b_k$
			$= \sum_{k=j+1}^{p_b} \bar{b}[k] \frac{\partial}{\partial b_j} g_k$
Init	$[i = 0 : p_a]$	$\frac{\partial}{\partial a_i} z$	$= \bar{b}[i - p_a + 1]$

There is one *line of code* for each line in f.

OpenGamma $\text{Cost}(P + D) \leq \omega_A \, \text{Cost}(P) \qquad \omega_A \in [3, 4]$

AD adjoint: example

The function (with 4 inputs)

$$z = (a_0 + \exp(a_1))\,(\sin(a_2) + \cos(a_3)) + (a_1)^2 + a_3.$$

```
public double f(double[] a) {
  double b1 = a[0] + Math.exp(a[1]);
  double b2 = Math.sin(a[2]) + Math.cos(a[3]);
  double b3 = b1 * b2 + Math.pow(a[1], 2) + a[3];
  return b3;
}
```

Note: all the code used in this presentation is "pseudo-code" written to fit on the screen, not actual code.

AD adjoint: example

```
public double f(double[] a, double[] aBar) {
  // Forward sweep
  double b1 = a[0] + Math.exp(a[1]);
  double b2 = Math.sin(a[2]) + Math.cos(a[3]);
  double b3 = b1 * b2 + Math.pow(a[1], 2) + a[3];
  // Backward sweep
  double b3Bar = 1.0;
  double b2Bar = b1 * b3Bar;
  double b1Bar = b2 * b3Bar + 0.0 * b2Bar;
  aBar[3] = 1.0 * b3Bar - Math.sin(a[3]) * b2Bar;
  aBar[2] = Math.cos(a[2]) * b2Bar;
  aBar[1] = 2 * a[1] * b3Bar + Math.exp(a[1]) * b1Bar;
  aBar[0] = 1.0 * b1Bar;
  return b3;
}
```

AD adjoint: example

Example implementation:
1,000,000 - value: 79 ms
1,000,000 - value and 4 derivatives (adjoint): 149 ms
1,000,000 - value and 4 derivatives (adjoint) - $\exp(a_1)$ stored: 126 ms
Ratios: 1.89 - 1.60

AD adjoint: advantages/drawbacks

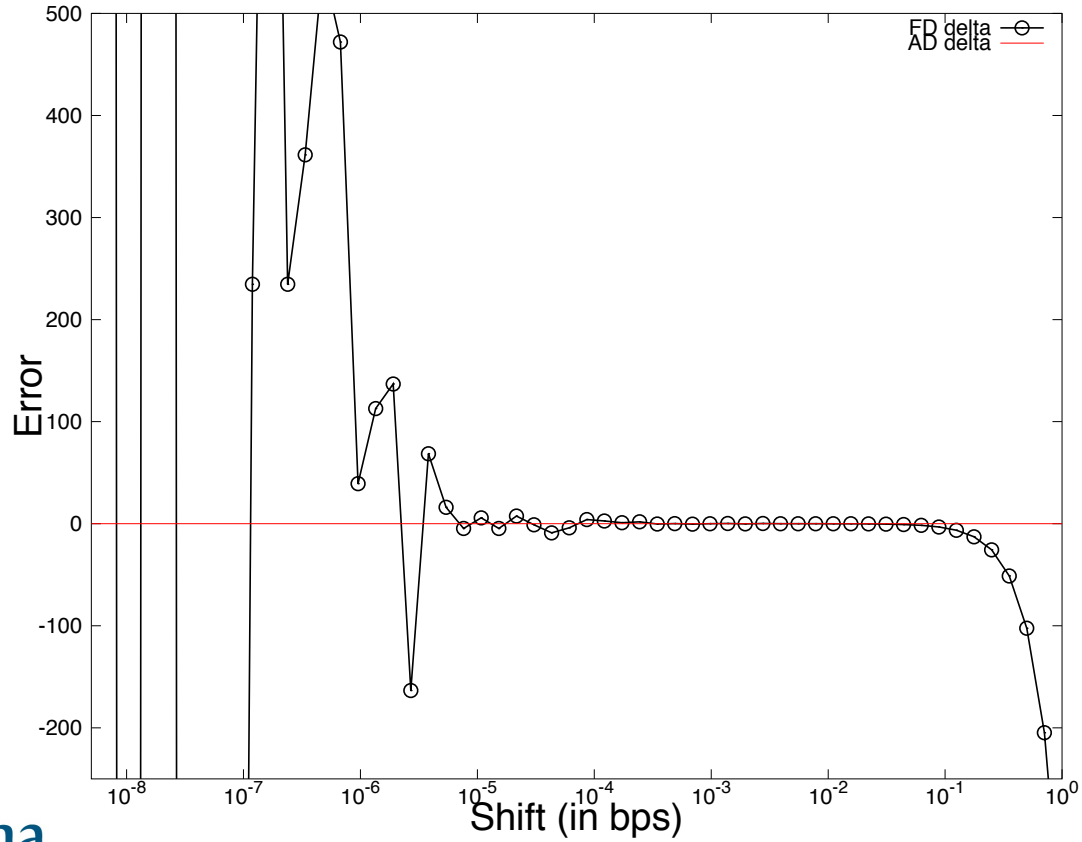
- Requires a bottom-up approach. Can be implemented for an algorithm only if all its components are already implemented.
- Requires more development time.
- When it is correctly implemented, it is very fast.
- Provides results with machine accuracy precision.

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

In Practice: Stability

Delta (curve parallel shift) of a (5Yx5Y) swaption in Hull-White one-factor model.
Delta approximated by one-sided differentiation ratio.



OpenGamma

Our implementation in OG-Analytics

- Library written in Java.
- Built with efficient derivatives in mind from the start.
- AD code manually written.
- Code optimized when domain specific knowledge can improve speed.
- When required for financial reasons, the differentiation is modified.

In Practice: Black formula

The standard option price in the Black framework.
Class name: `BlackPriceFunction`
1,000,000 - value: 156 ms
1,000,000 - value + 3 derivatives: 176 ms
Cost ratio: 1.13
Black formula derivatives are optimised using the domain specific knowledge that the derivative of the price with respect to the (optimal) exercise boundary is 0.

In Practice: Swaption in SABR

The price of European swaptions (5Y quarterly) with physical delivery for a swap rate following a SABR model (Hagan et al. approximation) in a multi-curve framework.
Class name: `SwaptionPhysicalFixedIborSABRMethod`
5,000 - swaptions SABR (price): 75 ms
5,000 - swaptions SABR (price + 20+24 delta + 3 vega): 210 ms
Cost ratio: 2.8

In Practice: Swaption in SABR

```
public double pv(Swaption swpt, SABRData sabr) {
 double maturity = swpt.getMaturityTime();
 double expiry = swpt.getTimeToExpiry();
 FixedCouponSwap swap = swpt.getUnderlyingSwap();
 // Forward sweep
 double fwd = PRC.visit(swap, sabr);
 double pvbp = SwapMethod.pvbp(swap, sabr);
 double strike = SwapMethod.couponEquivalent(swap, pvbp, sabr);
 EuropeanVanillaOption option = new EuropeanVanillaOption(strike,
 double volatility = sabr.vol(expiry, maturity, strike, fwd);
 BlackData dataBlack = new BlackData(forward, 1.0, volatility);
 double black = black.price(dataBlack);
 double pv = pvbp * black;
 return pv
}
```

In Practice: Swaption in SABR

```
public IRSensitivity pvSensi(Swaption swpt, SABRData sabr) {
 // Forward sweep
 double fwd = PRC.visit(swap, sabr);
 double pvbp = SwapMethod.pvbp(swap, sabr);
 double strike = SwapMethod.couponEquivalent(swap, pvbp, sabr);
 EuropeanVanillaOption option = new EuropeanVanillaOption(strike,
 double[] volAdj = sabr.volAdj(expiry, maturity, strike, fwd);
 BlackData dataBlack = new BlackData(forward, 1.0, volAdj[0]);
 double[] bsAdj = black.priceAdj(option, dataBlack);
 double pv = pvbp * bsAdj[0];
 // Backward sweep
 double pvBar = 1.0;
 double volBar = pvbp * bsAdj[2] * pvBar;
 double pvbpBar = bsAdj[0] * pvBar;
 double fwdBar = pvbp * bsAdj[1] * pvBar + volAdj[1] * volBar;
 IRSensi pvbpDr = SwapMethod.pvbpSensi(swap, sabr);
 IRSensi fwdDr = PRSC.visit(swap, sabr);
 return pvbpDr.mult(pvbpBar).plus(fwdDr.mult(fwdBar));
}
```

In Practice: Exotic (Ratchet) in Hull-White

Hull-White one factor with Monte-Carlo in multi-curves framework.
Class name: `HullWhiteMonteCarloMethod`
Ratchet on Libor (20 quarterly coupons over 5 years), 12,500 paths.
Ratchet Hull-White (12,500 paths - price): 56 ms
Ratchet Hull-White (12,500 paths - price + 20+20 delta): 176 ms
Cost ratio: 3.12

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

Speed beyond Speed: Calibration

- the price of an *exotic instrument* is related to a specific basket of *vanilla instruments*;
- the prices of these vanilla instruments are computed in a given *base model*;
- a complex model is selected; its parameters are calibrated on the prices of the options in the base model. This step is usually done through a generic numerical equation solver or optimizer (least squares);
- the exotic instrument is priced with the calibrated model.
- Goal: derivatives of the exotic instrument price with respect to the base model parameters.

Speed beyond Speed: Calibration

- the price of an *exotic instrument* is related to a specific basket of *vanilla instruments*;
- the prices of these vanilla instruments are computed in a given *base model*;
- a complex model is selected; its parameters are calibrated on the prices of the options in the base model. This step is usually done through a generic numerical equation solver or optimizer (least squares);
- the exotic instrument is priced with the calibrated model.
- Goal: derivatives of the exotic instrument price with respect to the base model parameters.

Greeks through calibration

Input C: yield curves
Input Θ: parameters for the base model (SABR parameters)
Intermediary value Φ: parameters for the calibrated model (LMM)
$\mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Base}}$: pv of vanilla instruments in base model.
$\mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Cal}}$: pv of vanilla instruments in calibrated model.
$\mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Cal}}$: pv of exotic instrument in calibrated model.

The calibration procedure (perfect calibration) is

$$0 = f(C, \Theta, \Phi) = \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Base}}(C, \Theta) - \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Cal}}(C, \Phi).$$

Greeks through calibration

The calibration problem looks like:

$$\begin{aligned} b &= g_1(a) \\ c &\text{ s. t. } g_2(b,c) = 0 \\ z &= g_3(c) \end{aligned}$$

with $g_1 : \mathbb{R}^{p_a} \to \mathbb{R}^{p_b}$, $g_2 : \mathbb{R}^{p_b} \times \mathbb{R}^{p_c} \to \mathbb{R}^{p_c}$ and $g_3 : \mathbb{R}^{p_c} \to \mathbb{R}^{p_z}$.
We know how to deal with

$$\begin{aligned} b &= g_1(a) \\ c &= g_4(b) \\ z &= g_3(c) \end{aligned}$$

Mathematics (3): Implicit function theorem

Theorem (Implicit function theorem)

Under mild regularity conditions on f, if

$$f(x_0, y_0) = 0$$

and if $D_y f(x_0, y_0)$ is invertible, then, near x_0, there exists a (implicit) function g such that $f(x, g(x)) = 0$, g is differentiable in x_0 and

$$D_x g(x_0) = -\left(D_y f(x_0, y_0)\right)^{-1} D_x f(x_0, y_0).$$

The term *exists* is in the sense of the mathematicians, not of the computer scientists!

Greeks through Calibration: Implicit AD

The elements of $\mathbb{R}^p$ are represented by column vectors. The derivative $Df(a) \in \mathcal{L}(\mathbb{R}^{p_a}, \mathbb{R}^{p_z})$ is represented by a $p_z \times p_a$ matrix (p_z rows, p_a columns).
The adjoint version of the algorithm is

$$
\begin{aligned}
\bar{z} &= I \quad \text{(with } I \text{ the } p_z \times p_z \text{ identity)} \\
\bar{c} &= (D_c g_3(c))^T \bar{z} \\
\bar{b} &= (D_b g_4(b))^T \bar{c} = -\left((D_c g_2(b,c))^{-1} D_b g_2(b,c) \right)^T \bar{c} \\
\bar{a} &= (D_a g_1(a))^T \bar{b}.
\end{aligned}
$$

Greeks through calibration

Calibration: $\Phi_0 = \Phi(C_0, \Theta_0)$.
Using the calibration:

$$\mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Base}}(C_0, \Theta_0) = \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Cal}}(C_0, \Phi(C_0, \Theta_0))$$

The quantities of interest are

$$D_C \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Base}} \text{ and } D_\Theta \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Base}}.$$

Through composition we have

$$D_C \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Base}} = D_C \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Cal}}(C_0, \Phi_0) + D_\Phi \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Cal}}(C_0, \Phi_0) D_C \Phi(C_0, \Theta_0),$$

and

$$D_\Theta \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Base}}(C_0, \Theta_0) = D_\Phi \mathrm{PV}^{\mathrm{Exotic}}_{\mathrm{Cal}}(C_0, \Phi_0) D_\Theta \Phi(C_0, \Theta_0)$$

where $D_C\Phi$ and $D_\Theta\Phi$ are unknown.

Greeks through calibration

Using the implicit function theorem, the function Φ is differentiable and its derivatives can be computed from the derivative of f:

$$D_\Theta \Phi(C_0, \Theta_0) = \left(D_\Phi \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Cal}}(C_0, \Phi_0) \right)^{-1} D_\Theta \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Base}}(C_0, \Theta_0)$$

and

$$\begin{aligned} D_C \Phi(C_0, \Theta_0) &= \left(D_\Phi \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Cal}}(C_0, \Phi_0) \right)^{-1} \\ &\quad \left(D_C \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Base}}(C_0, \Theta_0) - D_C \mathrm{PV}^{\mathrm{Vanilla}}_{\mathrm{Cal}}(C_0, \Phi_0) \right). \end{aligned}$$

OpenGamma

Amortised swaptions in LMM

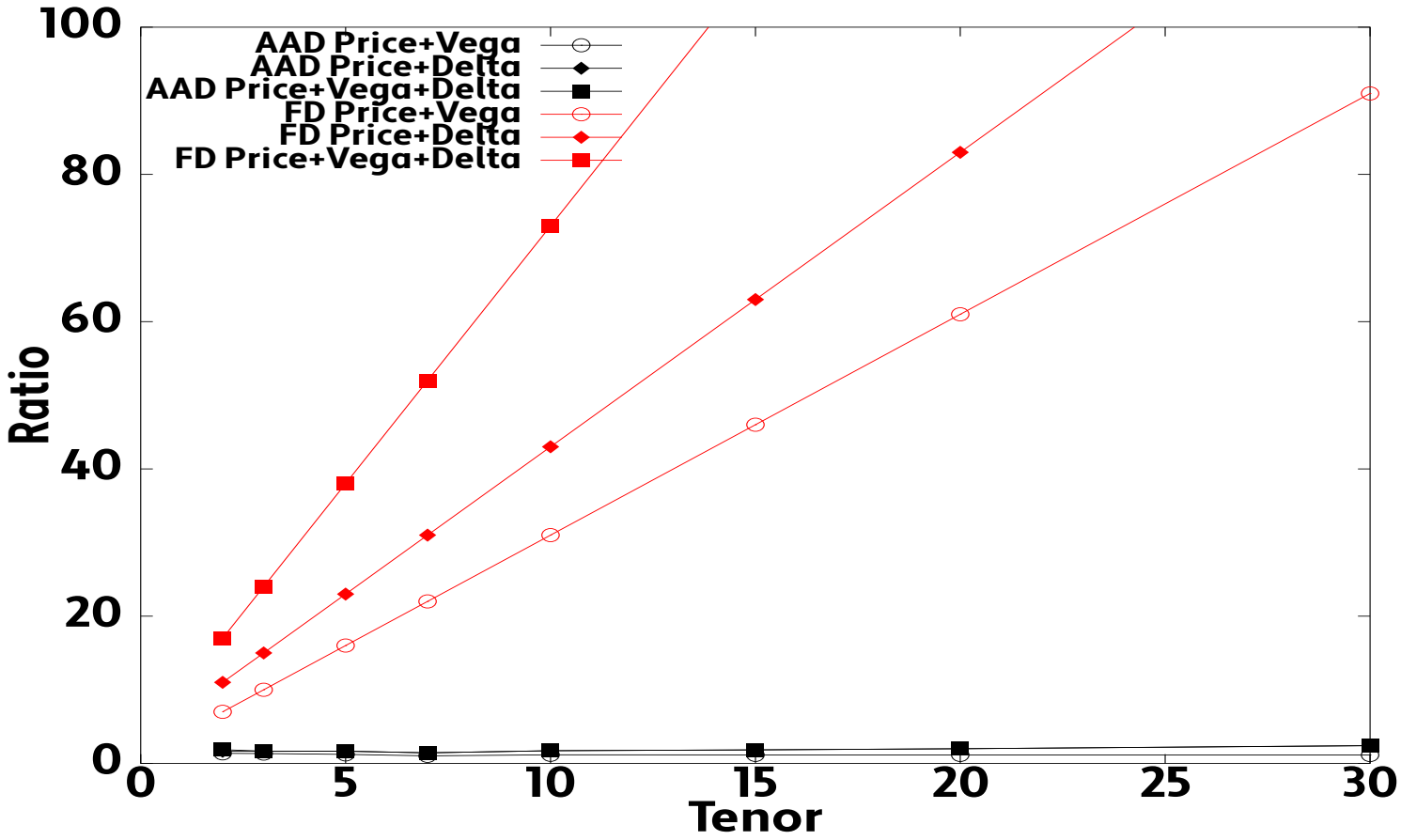
Exotic: 5Yx10Y amortised European swaption (yearly amortisation)
Vanilla: 10 vanilla swaptions with tenors between 1Y and 10Y
Base model: SABR model (one set of 3 parameters for each tenor) and multi-curves framework.
Calibrated model: two-factor LMM with displaced diffusion.
Calibration: for each year the weights of the 4 volatilities are fixed; the weights multiplied by a common parameter (Φ).

Amortised swaptions in LMM

Exotic: 5Yx10Y amortised European swaption (yearly amortisation)
Vanilla: 10 vanilla swaptions with tenors between 1Y and 10Y
Base model: SABR model (one set of 3 parameters for each tenor) and multi-curves framework.
Calibrated model: two-factor LMM with displaced diffusion.
Calibration: for each year the weights of the 4 volatilities are fixed; the weights multiplied by a common parameter (Φ).

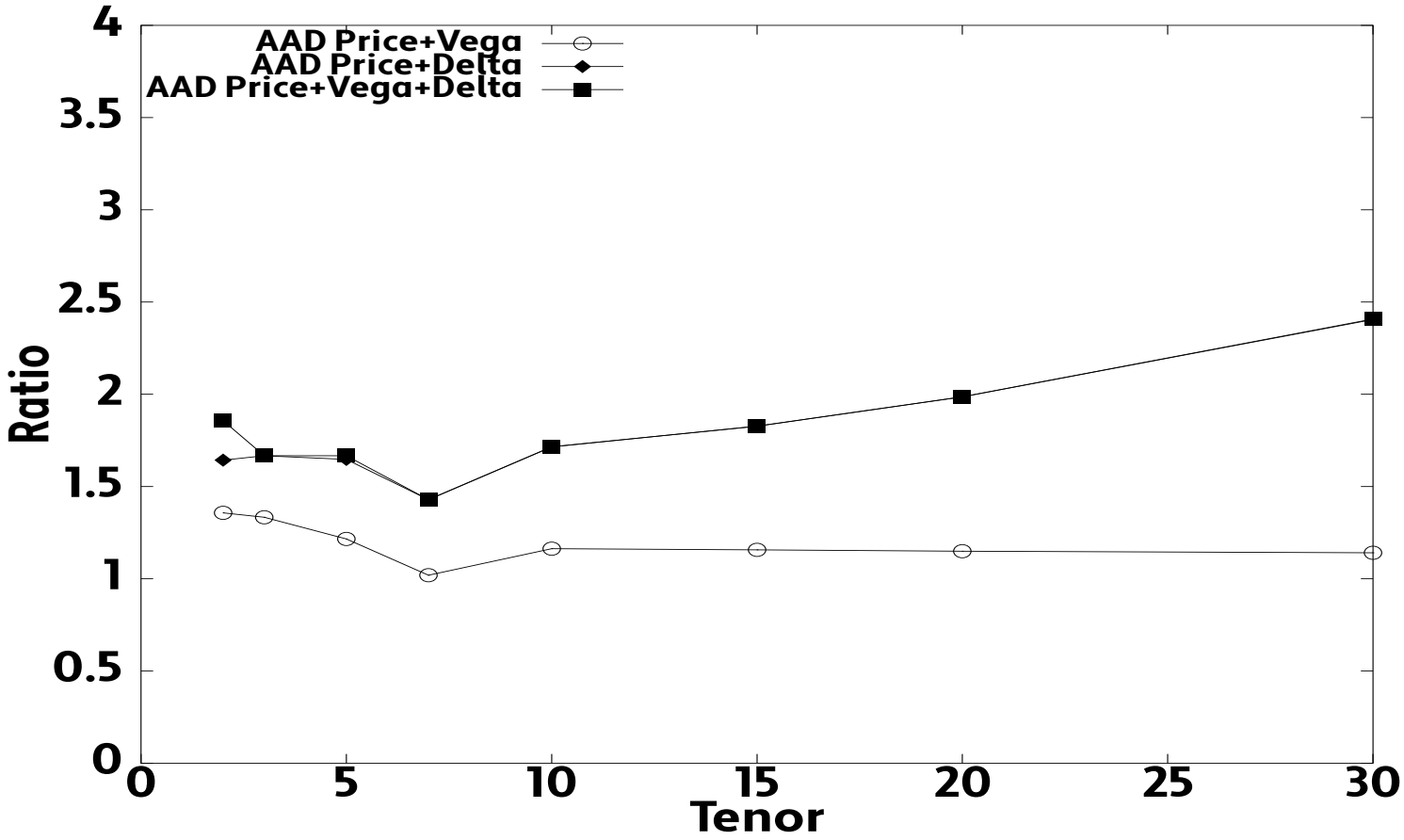
Risk type	Approach	Price time	Risks time	Total
SABR	FD	1.00	30×1.00	31.00
SABR	AAD	1.00	0.18	1.18
Curve	FD	1.00	42×1.00	43.00
Curve	AAD	1.00	0.74	1.74
Curve and SABR	FD	1.00	72×1.00	73.00
Curve and SABR	AAD	1.00	0.75	1.75

Amortised swaptions in LMM (exact calibration)



OpenGamma

Amortised swaptions in LMM (exact calibration)



OpenGamma

Financial sensitivities

Financially meaningful sensitivities can be computed and not only the pure data-flow sensitivities that would be obtained by automatic differentiation.
Example: computing the currency exposure (delta) of a forex option in a sticky strike way when the volatility is stored in a delta dependent way.

Financial sensitivities

```
CurrencyAmount currencyExposureStickyStrike
  (ForexOptionVanilla option, ForexSmileDelta data) {
 double dfDomestic = data.dscFactor(ccy2, option.payTime());
 double dfForeign = data.dscFactor(ccy1, option.payTime());
 double spot = data.fxRate(ccy1, ccy2);
 double fwd = spot * dfForeign / dfDomestic;
 double vol = data.vol(ccy1, ccy2, fwd, option.expiry(), opti
 double[] priceAD = BLACK.getPriceAD(option, fwd, dfDomestic
 double price = priceAD[0] * amount1;
 double delta = priceAD[1] * dfForeign / dfDomestic;
 CurrencyAmount ce = CurrencyAmount.of(ccy1, delta * amount1)
 return ce.plus(ccy2, - delta * amount1 * spot + price);
}
```

OpenGamma

Financial sensitivities

We have

$$P = \text{Black}(S, \sigma(S)).$$

The direct implementation would give

$$\Delta_{\text{Sticky delta}} = \frac{dP}{dS} = \frac{\partial \text{Black}}{\partial S} + \frac{\partial \text{Black}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial S}$$

but we want

$$\Delta_{\text{Sticky strike}} = \frac{\partial \text{Black}}{\partial S}.$$

The currency exposure computation does not take into account the change of volatility wrt the change of forward. The automatic differentiation would allow only sticky delta currency exposure. Both the sticky delta and sticky strike sensitivities can be computed without changing the way the data is stored.

Algorithmic Differentiation: Fast Greeks and Beyond

OpenGamma

Conclusion

- Algorithmic Differentiation: price and derivatives (greeks) at the computation cost of less than 4 times the cost of one price.
- The fast execution time comes with a cost: a (slightly) longer development time (the code length is doubled, not the development time).
- Library needs to be built from the ground with AD in mind.
- When done properly, AD is very efficient.
- Calibrations require equation solving or least square minimum. Using implicit function approach the AD can be made even more efficient. For prices including calibration, the time ratio can be well below two.
- Financially important sensitivities can be computed, not only data-flow sensitivities.

Conclusion

- Algorithmic Differentiation: price and derivatives (greeks) at the computation cost of less than 4 times the cost of one price.
- The fast execution time comes with a cost: a (slightly) longer development time (the code length is doubled, not the development time).
- Library needs to be built from the ground with AD in mind.
- When done properly, AD is very efficient.
- Calibrations require equation solving or least square minimum. Using implicit function approach the AD can be made even more efficient. For prices including calibration, the time ratio can be well below two.
- Financially important sensitivities can be computed, not only data-flow sensitivities.

References


Capriotti, L. (2011).
Fast Greeks by algorithmic differentiation.
The Journal of Computational Finance, 14(3):3--35.


Christianson, B. (1998).
Reverse accumulation and implicit functions.
Optimisation Methods and Software, 9(4):307--322.


Denson, N. and Joshi, M. (2009).
Fast and accurate Greeks for the Libor Market Model.
Journal of Computational Finance., 14(4):115--140.


Giles, M. and Glasserman, P. (2006).
Smoking adjoints: fast Monte Carlo greeks.
Risk, 19:88--92.


Griewank, A. and Walther, A. (2008).
Evaluating derivatives: principles and techniques of algorithmic differentiation.
SIAM, second edition.

References


Henrard, M. (2010a).
The irony in the derivatives discounting - Part II: the crisis.
Wilmott Journal, 2(6):301--316.


Henrard, M. (2010b).
Swaptions in Libor Market Model with local volatility.
Wilmott Journal, 2(3):135--154.


Henrard, M. (2012).
Adjoint algorithmic differentiation: Calibration and implicit function theorem.
Journal of Computational Finance, to appear.
Preprint available at ssrn.com/abstract=1896329.


Schlenkrich, S. (2012).
Efficient calibration of the Hull-White model.
Optimal Control Applications and Methods, 33(3):352--362.

Thank you!

Contact OpenGamma

Web: `www.opengamma.com`
Email: `info@opengamma.com`
Twitter: `@OpenGamma`

Europe
OpenGamma
185 Park Street
London SE1 9BL
United Kingdom

North America
OpenGamma
230 Park Avenue South
New York, NY 10003
United States of America

OpenGamma