Algorithmic Differentiation of the Local Volatility Model

Jacques du Toit (Numerical Algorithms Group), Viktor Mosenkis and Uwe Naumann (RWTH Aachen)

Local Volatility Model

A stochastic process $S = (S_t)_{t\geq 0}$ follows the *local volatility model* if

$$dS_t = \big(r(t) - q(t)\big)S_t\,dt + \sigma(t, S_t)S_t\,dW_t \qquad (1)$$

where r and q are term structures of interest and dividends and $(W_t)_{t\geq 0}$ is a standard Brownian motion. This model is used in many markets including equities and FX. The function $\sigma(t,x)$ is the *local volatility* of S and is *unknown*, but is computable from the Dupire formula by using market quotes of call option prices.

Input Data Requirements

Computing σ in (1) from the Dupire formula requires a *smooth continuum* of market quotes at all strikes and maturities. This does not exist. Instead practitioners take the *implied volatility surface* and interpolate it into a smooth function. Liquid assets can have more than 150 quotes, meaning the local volatility model can easily have more than 200 input parameters.

Model Risk

Risk is typically treated as the sensitivity (derivative) of the price with respect to input parameters. Practitioners are interested in some or all of the following:

$$\begin{array}{cccc} \frac{\partial P}{\partial S_0} & \frac{\partial^2 P}{\partial S_0^2} & \frac{\partial P}{\partial K} & \frac{\partial P}{\partial r(T_k)} \\ \frac{\partial P}{\partial q(T_\ell)} & \frac{\partial P}{\partial \theta(T_i, K_j)} & \frac{\partial^2 P}{\partial \theta^2(T_i, K_j)} & \frac{\partial^2 P}{\partial S_0 \partial \theta(T_i, K_j)} \end{array} \qquad (2)$$

where P is call option price in the local volatility model. Typically these derivatives are computed by finite differences (bumping) which is very slow and inaccurate.

Pricing by PDE: Crank–Nicholson Method

From (1) we know following Andersen, Brotherton-Ratcliffe (1997) that

$$\frac{\partial H(t,x)}{\partial t} + \frac{1}{2}v(t,x)\frac{\partial^2 H(t,x)}{\partial x^2} + b(t,x)\frac{\partial H(t,x)}{\partial x} = r(t)H(t,x) \qquad (3)$$

where $x = \ln(S)$ and $H(t,x) = P(t,S)$ and

$$v(t,x) = \sigma^2(t, e^x), \qquad b(t,x) = r(t) - q(t) - \frac{1}{2}v(t,x). \qquad (4)$$

The PDE (3) can be discretised directly and solved using a Crank–Nicholson scheme to obtain call option prices.

Test Problem

We took implied volatility data on a grid with 10 maturities and 40 strikes and interpolated using cubic splines. We used 10 point yield and dividend curves. This gave a **total of 1,223 derivatives** in (2) to compute. We wrote highly optimised (maximal caching of intermediate results) FD code and compared results and timings with an implementation based on algorithmic differentiation.

Algorithmic Differentiation

$$f : \mathbb{R}^n \to \mathbb{R}^m\,, \qquad \mathbf{y} = f(\mathbf{x})$$

First-Order AD

- **First-Order Tangent-Linear Code** $f^{(1)}$ (F)
  $\mathbf{y}^{(1)} = \nabla f(\mathbf{x}) \cdot \mathbf{x}^{(1)} \quad \Rightarrow \quad \nabla f$ at $O(n) \cdot Cost(f)$
- **First-Order Adjoint Code** $f_{(1)}$ (R)
  $\mathbf{x}_{(1)} = \mathbf{y}_{(1)}^T \cdot \nabla f(\mathbf{x}) \quad \Rightarrow \quad \nabla f$ at $O(m) \cdot Cost(f)$

Second-Order AD ($m = 1$)

- **Second-Order Tangent-Linear Code** $f^{(1,2)}$ (FoF)
  $\underbrace{y^{(1,2)}}_{\in\mathbb{R}} = \underbrace{\mathbf{x}^{(1)^T}}_{\in\mathbb{R}^n} \cdot \underbrace{\nabla^2 f(\mathbf{x})}_{\in\mathbb{R}^{n\times n}} \cdot \underbrace{\mathbf{x}^{(2)}}_{\in\mathbb{R}^n} \quad \Rightarrow \quad \nabla^2 f$ at $O(n^2) \cdot Cost(f)$
- **Second-Order Adjoint Code** $f_{(1)}^{(2)}$ (FoR)
  $\underbrace{\mathbf{x}_{(1)}^{(2)}}_{\in\mathbb{R}^n} = \underbrace{y_{(1)}}_{\in\mathbb{R}} \cdot \nabla^2 f(\mathbf{x}) \cdot \mathbf{x}^{(2)}$
  $\Rightarrow \quad \nabla^2 f \cdot \mathbf{x}^{(2)}$ at $O(1) \cdot Cost(f)$ resp. $\nabla^2 f$ at $O(n) \cdot Cost(f)$

AD Tools

DCO

- first- and higher-order projections
- mathematically rigorous user interface
- exploitation of expression templates
- statement-level preaccumulation
- support for external functions (inclusion of tangent-linear or adjoint user code)
- support for (selected) NAG Library functions
- support for MPI, activity analysis, checkpointing, ...

AD-enabled NAG Library

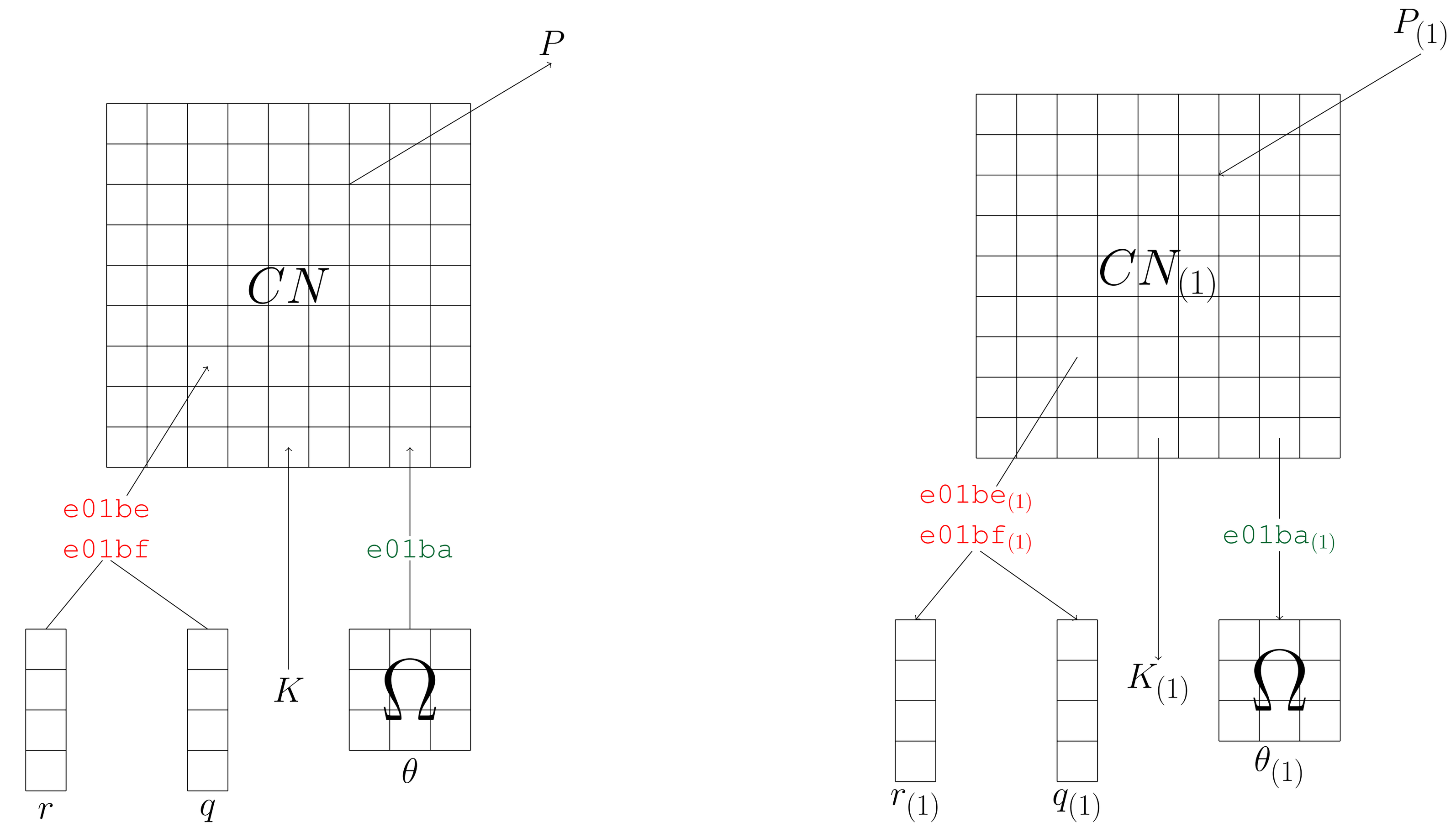
$$s(x) = \sum_{i=1}^{m} c_i N_i(x)$$

```
SUBROUTINE  E01BAF_A1S
    ( ..., X, X_A1S, ! x value of the data points
         , Y, Y_A1S, ! y value of the data points
         , LAMDA, LAMDA_A1S, ! normalized B-spline knots
         , C, C_A1S, ! B-spline coefficients
      ...)
```

Results

Implementation

Primal | Adjoint



Used the AD-enabled versions of the following NAG routines

- e01bef: Interpolating functions, monotonicity-preserving, piecewise cubic Hermite, one variable
- e01bff: Interpolated values, interpolant computed by e01bef, function only, one variable
- e01baf: Interpolating functions, cubic spline interpolant, one variable
- e02bcf: Evaluation of fitted cubic spline, function and derivatives
- f07cef: Solves a real tridiagonal system of linear equations using the LU factorization computed by f07cdf
- f07cdf: LU factorization of real tridiagonal matrix

Performance (Crank–Nicolson mesh size (250×500))

Greeks	Size	FD (s)	AD (s)	Speedup	AD Mode
$\frac{\partial P}{\partial r(T_k)} + \frac{\partial P}{\partial q(T_l)} + \frac{\partial P}{\partial \theta(T_i,K_j)}$	$11+11+400$	$4+4+90$	12	8.1	R
$\frac{\partial^2 P}{\partial S_0 \partial \theta(T_i,K_j)}$	400	200	24	8.3	FoR
$\frac{\partial^2 P}{\partial \theta^2(T_i,K_j)}$	400	100	$< 100?$	1	FoF
All	1222	300	124	2.4	

Jacques du Toit: jacques@nag.co.uk

Viktor Mosenkis: mosenkis@stce.rwth-aachen.de