Adjoint methods in computational finance

Mike Giles

Mathematical and Computational Finance Group,
Mathematical Institute, University of Oxford

Oxford-Man Institute of Quantitative Finance

12th Winter School on Mathematical Finance

Jan 21-23, 2013

Lecture outline

- SDEs and Monte Carlo methods:
  - LRM and pathwise sensitivity approaches
  - adjoint pathwise approach
  - use of automatic differentiation software
  - path dependent options
  - multiple payoffs
  - binning and correlation Greeks
  - local volatility example, revisited
  - discontinuous payoffs

Monte Carlo simulation

If we want the expected value of a payoff function P which depends on an underlying quantity S, so that

$$V \;=\; \mathbb{E}[P(S)] \;=\; \int P(S)\; p_S(\theta; S)\; \mathrm{d}S$$

where p_S is the probability distribution for S which depends on an input parameter θ, then the Monte Carlo estimate is

$$M^{-1} \sum_{m=1}^{M} P(S^{(m)})$$

where the samples $S^{(m)}$ are generated independently.

LRM sensitivity analysis

If p_S is a differentiable function of θ, then

$$\frac{\partial V}{\partial \theta} = \int P(S) \frac{\partial p_S}{\partial \theta} \, \mathrm{d}S = \int P(S) \frac{\partial \log p_S}{\partial \theta} p_S(S) \, \mathrm{d}S$$

which is equivalent to

$$\frac{\partial V}{\partial \theta} = \mathbb{E}\left[P(S) \frac{\partial \log p_S}{\partial \theta} \right]$$

which can be estimated as

$$M^{-1} \sum_{m=1}^{M} P(S^{(m)}) \frac{\partial \log p_S^{(m)}}{\partial \theta}$$

LRM sensitivity analysis

This is the Likelihood Ratio Method for computing sensitivities.

Its great strength is that there are no restrictions on $P(S)$
(e.g. it can be discontinuous) but it has two big drawbacks:

- it requires a known distribution p_S (e.g. log-normal)
- the Monte Carlo estimator usually has a much larger variance than alternative methods

Monte Carlo simulation

Alternatively, suppose S depends on θ and a simple random variable Z which does not depend on θ (e.g. multivariate Normal).

Then we have

$$V \;=\; \mathbb{E}[P(\,S(\theta;Z)\,)] \;=\; \int P(\,S(\theta;Z)\,)\;p_Z(Z)\;\mathrm{d}Z$$

and the Monte Carlo estimate remains

$$M^{-1}\sum_{m=1}^{M} P(\,S(Z^{(m)})\,)$$

where the samples $Z^{(m)}$ are generated independently.

Pathwise sensitivity analysis

If P is a differentiable function of θ, then

$$\frac{\partial V}{\partial \theta} = \int \frac{\partial P}{\partial S} \frac{\partial S}{\partial \theta} \, p_Z(Z) \, \mathrm{d}Z$$

which is equivalent to

$$\frac{\partial V}{\partial \theta} = \mathbb{E}\left[\frac{\partial P}{\partial S} \frac{\partial S}{\partial \theta}\right]$$

This is also OK if P is continuous and piecewise differentiable, but not if P is discontinuous.

For example, for a simple digital option with value 0 or 1 depending on S then locally $\dfrac{\partial P}{\partial S} = 0$, but $\dfrac{\partial V}{\partial \theta} \neq 0$.

Pathwise sensitivity analysis

This gives the pathwise sensitivity estimate

$$M^{-1} \sum_{m=1}^{M} \frac{\partial P}{\partial S} \, \dot{S}^{(m)}$$

where $\dot{S}$ is the sensitivity keeping fixed all of the random numbers.

Note that this corresponds very naturally to applying the forward mode sensitivity analysis to the standard Monte Carlo estimator, but my derivation shows the need for $P(S)$ to be continuous – this is its big weakness.

Monte Carlo sensitivities

For European payoff $P(S_N)$, the forward mode sensitivity calculation is:

- set $S_0(\theta), \dot{S}_0$
- for timestep n from 0 to $N-1$:

  compute random numbers Z_n

  compute $S_{n+1} = f_n(\theta; S_n, Z_n)$

  compute $\dot{S}_{n+1} = \dfrac{\partial f_n}{\partial S_n}\,\dot{S}_n + \dfrac{\partial f_n}{\partial \theta}$

- calculate payoff $P(\theta; S_N)$ and $\dot{P} = \dfrac{\partial P}{\partial S_N}\,\dot{S}_N + \dfrac{\partial P}{\partial \theta}$

Monte Carlo sensitivities

The corresponding adjoint (reverse mode) sensitivity is

$$M^{-1} \sum_{m=1}^{M} \overline{\theta}^{(m)}$$

where $\overline{\theta}^{(m)}$ corresponds to $\left(\dfrac{\partial P}{\partial \theta}\right)^T$ for m^{th} path

Note: the adjoint sensitivity is the same as the standard pathwise sensitivity, so it is valid under the same conditions
(e.g. $P(S)$ Lipschitz and piecewise differentiable)

Monte Carlo sensitivities

For European payoff $P(S_N)$, the adjoint calculation involves:

- set $S_0(\theta)$
- for timestep n from 0 to $N-1$:
  compute and store random numbers Z_n
  compute and store $S_{n+1} = f_n(\theta; S_n, Z_n)$
- calculate payoff $P(\theta; S_N)$

- set $\overline{S}_N = \dfrac{\partial P}{\partial S_N}$ and $\overline{\theta} = \dfrac{\partial P}{\partial \theta}$
- for timestep n from $N-1$ to 0
  compute $\overline{S}_n$ and $\overline{\theta}$ increment
- add final $\overline{\theta}$ increment $\overline{S}_0^T \dot{S}_0$

Trivial example

Geometric Brownian Motion SDE, European call payoff, Euler discretisation:

Given S_0, r, σ, T, K, then

$$S_{n+1} = S_n + r\, S_n\, \Delta t + \sigma\, S_n \sqrt{\Delta t}\, Z_n \quad n = 0, \ldots N-1$$

and the payoff is

$$P = \exp(-rT)\ \max(0, S_N - K)$$

Note that there are 5 different sensitivities which can be computed here.

Trivial example

In the forward mode, given $\dot{S}_0, \dot{r}, \dot{\sigma}, \dot{T}, \dot{K}$, then

$$\begin{aligned}
\dot{S}_{n+1} &= \left(1 + r\,\Delta t + \sigma\sqrt{\Delta t}\,Z_n\right)\dot{S}_n \\
&+ S_n\,\Delta t\;\dot{r} \\
&+ S_n\sqrt{\Delta t}\,Z_n\;\dot{\sigma} \\
&+ \left(r\,S_n + \tfrac{1}{2}\,\sigma\,S_n\,\Delta t^{-1/2}\,Z_n\right) N^{-1}\,\dot{T}
\end{aligned}$$

and the payoff sensitivity is

$$\dot{P} = \exp(-rT)\;\mathbf{1}_{S_N>K}\;\left(\dot{S}_N - \dot{K} - (S_N - K)\;(T\dot{r} + r\,\dot{T})\right)$$

Trivial example

To obtain the sensitivities to all of S_0, r, σ, T, K, in one pass, we can set

$$\dot{S}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \dot{r} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \dot{\sigma} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \dot{T} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \dot{K} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix},$$

and then the final $\dot{P}$ will be a vector of sensitivities:

$$\dot{P} = \begin{pmatrix} \frac{\partial P}{\partial S_0} \\ \frac{\partial P}{\partial r} \\ \frac{\partial P}{\partial \sigma} \\ \frac{\partial P}{\partial T} \\ \frac{\partial P}{\partial K} \end{pmatrix}$$

Trivial example

In the reverse mode, we start with

$$\begin{aligned}
\overline{S}_N &= \exp(-rT)\ \mathbf{1}_{S_N>K} \\
\overline{r} &= -\exp(-rT)\ \mathbf{1}_{S_N>K}\ (S_N-K)\ T \\
\overline{\sigma} &= 0 \\
\overline{T} &= -\exp(-rT)\ \mathbf{1}_{S_N>K}\ (S_N-K)\ r \\
\overline{K} &= -\exp(-rT)\ \mathbf{1}_{S_N>K}
\end{aligned}$$

and then loop over timesteps from $N-1$ to 0 using

$$\begin{aligned}
\overline{S}_n &= \left(1 + r\,\Delta t + \sigma\,\sqrt{\Delta t}\,Z_n\right)\overline{S}_{n+1} \\
\overline{r} &\mathrel{+}= S_n\,\Delta t\ \overline{S}_{n+1} \\
\overline{\sigma} &\mathrel{+}= S_n\,\sqrt{\Delta t}\,Z_n\ \overline{S}_{n+1} \\
\overline{T} &\mathrel{+}= \left(r\,S_n + \tfrac{1}{2}\,\sigma\,S_n\,\Delta t^{-1/2}\,Z_n\right) N^{-1}\ \overline{S}_{n+1}
\end{aligned}$$

to get all 5 sensitivities, $\overline{S}_0$, $\overline{r}$, $\overline{\sigma}$, $\overline{T}$, $\overline{K}$

Monte Carlo sensitivities

In more complicated cases, AD tools can simplify the software development process, especially for the reverse mode.

If a routine

`step(n,theta,S,Z)`

performs the n^{th} timestep calculation, taking θ, S_n, Z_n as input and returning S_{n+1}, then AD tools can generate a routine

`step_b(n,theta,theta_b,S,S_b,Z)`

which takes inputs $\theta, \overline{\theta}, S_n, \overline{S}_{n+1}, Z_n$ and returns $\overline{S}_n$ and an updated $\overline{\theta}$.

AD tools can also generate the adjoint routines for the S_0 initialisation and the payoff evaluation.

Monte Carlo sensitivities

Some more implementation detail:

- first, go forward through the path storing the state S_n at each timestep (corresponds to "checkpointing" in AD terminology)
- then, go backwards through the path, using reverse mode AD for each step – this will re-do the internal calculations for the timestep and then do its adjoint
- when hand-coding for maximum performance, I also store the result of any very expensive operations (typically `exp`) to avoid having to re-do these

Note that this is different from applying AD to the entire path, which would require a lot of storage – it's cheaper to re-calculate the internal variables rather than fetch them from main memory

Monte Carlo sensitivities

By computing $\overline{\theta} \equiv \dfrac{\partial P}{\partial \theta}$ for each path individually, we can also compute a confidence interval for the sensitivity estimate.

If we had applied AD to the entire code which computes:

- the estimate
- the confidence interval

then we would have obtained:

- the sensitivity of the estimate
- the sensitivity of the confidence interval, not the confidence interval of the sensitivity – a subtle but important difference.

LIBOR Market Model

As an example, consider the LIBOR market model of BGM, with $m+1$ bond maturities T_i, with spacings $T_{i+1} - T_i = \delta$.

The forward rate for the interval $[T_i, T_{i+1})$ satisfies

$$\frac{\mathrm{d}L_i(t)}{L_i(t)} = \mu_i(L(t))\,\mathrm{d}t + \sigma_i^\top\,\mathrm{d}W(t), \quad 0 \le t \le T_i,$$

where
$$\mu_i(L(t)) = \sum_{j=\eta(t)}^{i} \frac{\sigma_i^\top \sigma_j\,\delta L_j(t)}{1+\delta L_j(t)},$$

and $\eta(t)$ is the index of the next maturity date.

For simplicity, we keep $L_i(t)$ constant after maturity, and take the volatilities to be a function of time to maturity, $\sigma_i(t) = \sigma_{i-\eta(t)+1}(0)$.

LIBOR Market Model

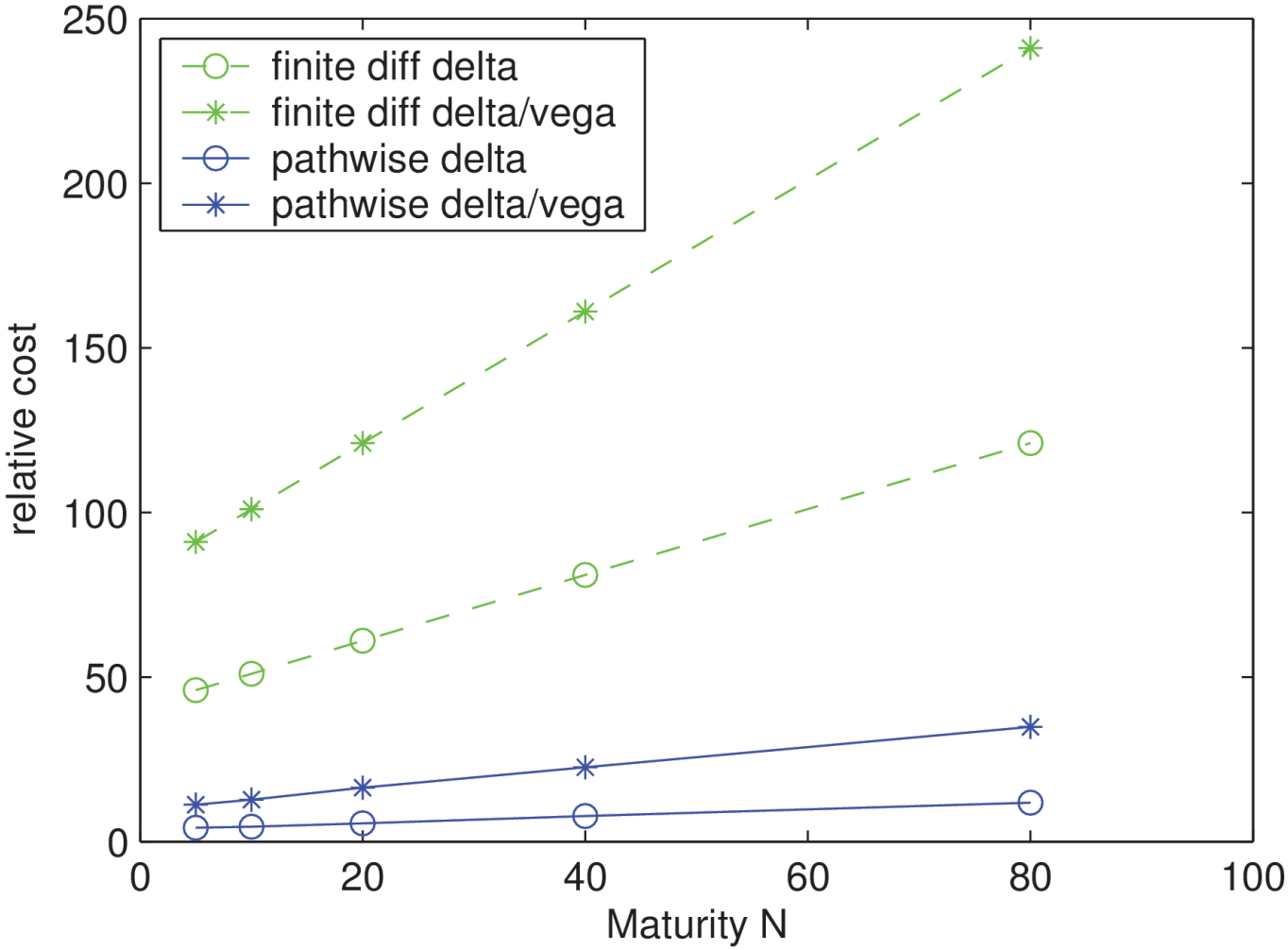
The LMM portfolio has 15 swaptions all expiring at the same time, N periods in the future, involving payments/rates over an additional 40 periods in the future.

Interested in computing Deltas, sensitivity to initial $N+40$ forward rates, and Vegas, sensitivity to initial $N+40$ volatilities.

Focus is on the cost of calculating the portfolio value and the sensitivities, relative to just the value.

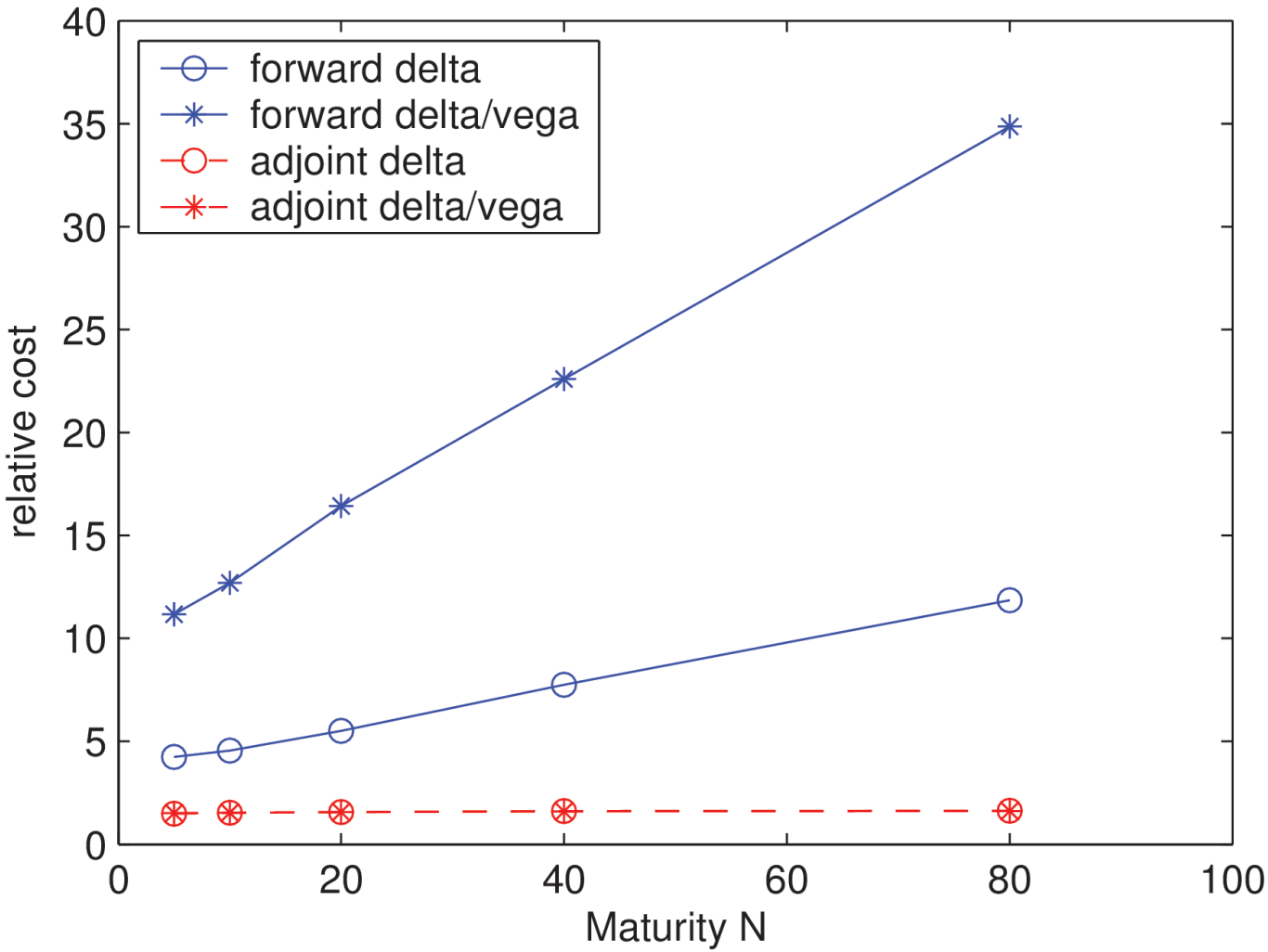
LIBOR Market Model

Finite differences versus forward pathwise sensitivities:



LIBOR Market Model

Forward versus adjoint pathwise sensitivities:



Additional complications

- path-dependent payoffs
- efficiency improvement for handling multiple European payoffs (Christoph Kaebe & Ekkehard Sachs)
- binning for expensive pre-processing steps (Luca Capriotti)
- handling discontinuous payoffs

Path dependent payoffs

The simplest way to handle path dependent payoffs is to make the payoff depend only on the final state S_N by including one or more of the following as part of a larger state $\widehat{S}_n$

- $\displaystyle\sum_{m\le n} S_m$ for Asian options
- $\min\limits_{m\le n} S_m, \ \max\limits_{m\le n} S_m$ for lookback options
- sum of all (discounted) cash payments made so far

We then have an augmented timestep calculation

$$\widehat{S}_{n+1} = \widehat{f}_n(\theta; \widehat{S}_n, Z_n)$$

and can again use AD tools to generate the whole adjoint code.

Multiple European payoffs

Suppose that you have

- n_θ input parameters
- n_P different payoffs
- dimension d path simulation

If n_θ is smallest, use forward mode sensitivity analysis

If n_P is smallest, use reverse mode sensitivity analysis

What if d is smallest?

Multiple European payoffs

Going back to the original matrix question, what is the best way of computing this?

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Multiple European payoffs

The most efficient approach is

- perform d adjoint calculations to determine

$$\frac{\partial S_N}{\partial \theta}$$

- perform d forward sensitivity calculations to determine

$$\frac{\partial P_k}{\partial S_N}$$

- combine these to obtain

$$\frac{\partial P_k}{\partial \theta} = \frac{\partial P_k}{\partial S_N} \frac{\partial S_N}{\partial \theta}$$

Binning

The need for binning is best demonstrated by the case of correlation Greeks – computing the sensitivity of the option value to each element in the correlation matrix.

Given correlation matrix Ω, Cholesky factor L is lower-triangular matrix such that $L\,L^T = \Omega$. If Z is a vector of uncorrelated unit Normals, then $L\,Z$ has variance Ω.

The standard pricing calculation has three stages

- perform Cholesky factorisation
- do M path calculations
- compute average and confidence interval

How do we compute the adjoint sensitivity to the correlation coefficients?

Binning

If we apply the reverse mode AD approach to the entire calculation, then we get an estimate of

- the sensitivity of the price
- the sensitivity of the confidence interval,
  not the confidence interval for the sensitivity!

To get the confidence interval for the sensitivity, for each path we can do the adjoint of the Cholesky factorisation, so we compute $\overline{\theta}$ for each path and then compute an average and confidence interval in the usual way.

However, this greatly increases the computational cost.

Binning

The binning approach splits the M paths into K groups (grouped arbitrarily, unlike "binning" in some other contexts).

For each group, it uses the full AD approach to efficiently compute an estimate of the price sensitivity.

It then uses the variability between the group averages to estimate the confidence interval.

Needs

- $K \gg 1$ to get a good estimate of the confidence interval
- $K \ll M$ for cost of K adjoint Cholesky calculations to be smaller than M path calculations

Local volatility example

The same approach can be used for a Monte Carlo version of the earlier example with local volatility:

- market implied vol $\sigma_I \implies$ local vol σ_L at a few points using Dupire's formula
- local vol σ_L at a few points $\implies \sigma_L, \sigma_L'$ through cubic spline procedure
- M Monte Carlo path calculation, using spline evaluation to obtain local volatility
- compute average and confidence interval

The adjoint of the path calculation will give increments to $\overline{\sigma_L}$ and $\overline{\sigma_L'}$. Then, for each group of paths, can use adjoint of first two stages to get an estimate for the sensitivity to market implied vol data.

Local volatility example

In the PDE case, we assumed the spline evalation location S didn't vary, but here it does. In the forward mode we get

$$\begin{aligned}\dot{\sigma}(S) &= a_j(S)\,\dot{\sigma}_j + b_j(S)\,\dot{\sigma}_{j+1} + c_j(S)\,\dot{\sigma}'_j + d_j(S)\,\dot{\sigma}'_{j+1} \\ &\quad + \left(a'_j(S)\,\sigma_j + b'_j(S)\,\sigma_{j+1} + c'_j(S)\,\sigma'_j + d'_j(S)\,\sigma'_{j+1}\right)\dot{S}\end{aligned}$$

and in the reverse mode we get

$$\begin{aligned}\overline{\sigma}_j &\mathrel{+}= a_j(S)\,\overline{\sigma}(S) \\ \overline{\sigma}_{j+1} &\mathrel{+}= b_j(S)\,\overline{\sigma}(S) \\ \overline{\sigma}'_j &\mathrel{+}= c_j(S)\,\overline{\sigma}(S) \\ \overline{\sigma}'_{j+1} &\mathrel{+}= d_j(S)\,\overline{\sigma}(S) \\ \overline{S} &\mathrel{+}= \left(a'_j(S)\,\sigma_j + b'_j(S)\,\sigma_{j+1} + c'_j(S)\,\sigma'_j + d'_j(S)\,\sigma'_{j+1}\right)\,\overline{\sigma}(S)\end{aligned}$$

Discontinuous payoffs

The biggest limitation of the pathwise sensitivity method (both forward mode and reverse mode) is that it cannot handle discontinuous payoffs.

There are 3 main ways to deal with this:

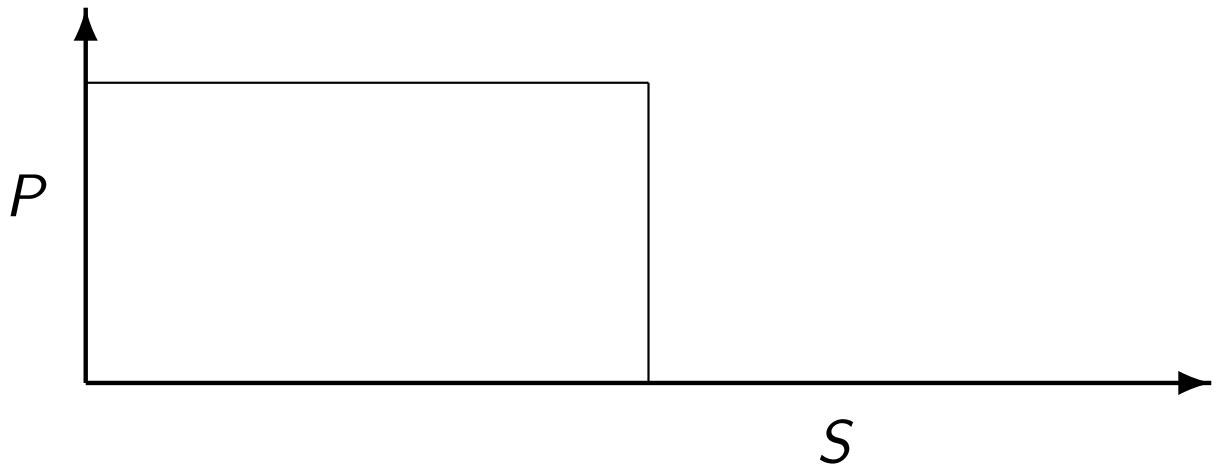
- explicitly smoothed payoffs
- using conditional expectation to smooth the payoff
- "vibrato" Monte Carlo

Of course, we can also switch to Likelihood Ratio Method or Malliavin calculus, but then I don't see how to get the efficiency benefits of adjoint methods.

Discontinuous payoffs

Explicitly-smoothed payoffs replace the discontinuous payoff by a smooth (or at least continuous) alternative.

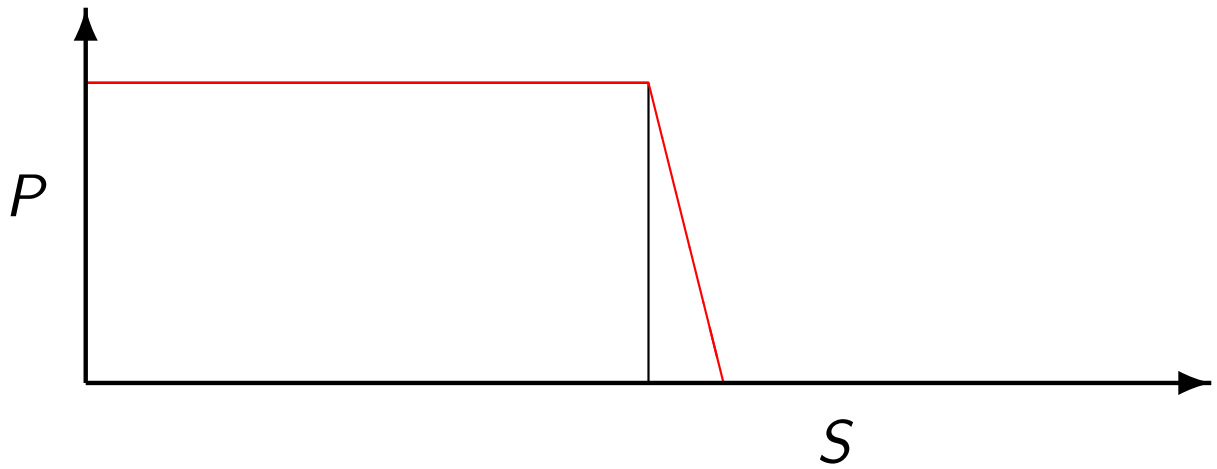
Digital options $P(S) \equiv H(S-K)$ can be replaced by a piecewise linear approximation:



Discontinuous payoffs

Explicitly-smoothed payoffs replace the discontinuous payoff by a smooth (or at least continuous) alternative.

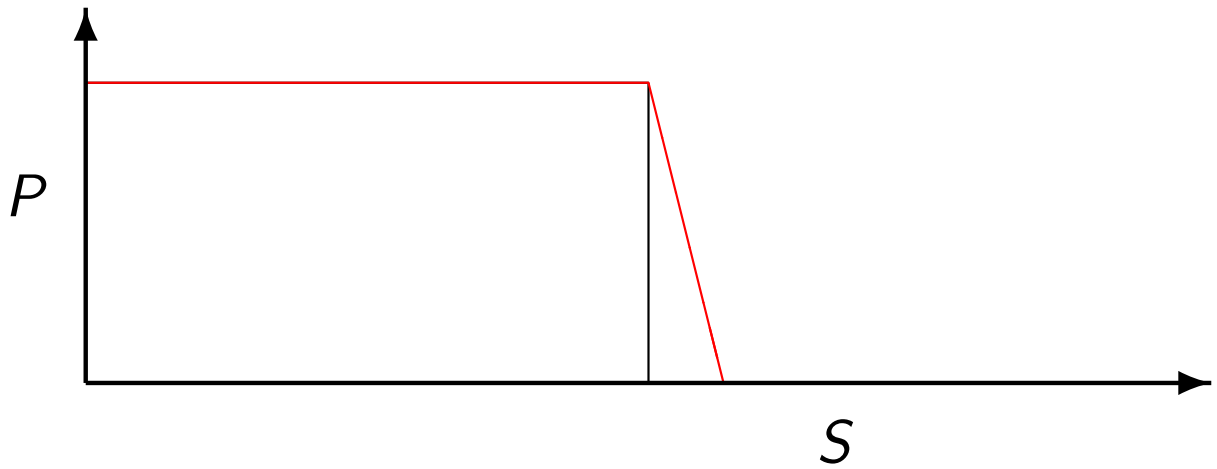
Digital options $P(S) \equiv H(S-K)$ can be replaced by a piecewise linear approximation:



Discontinuous payoffs

Explicitly-smoothed payoffs replace the discontinuous payoff by a smooth (or at least continuous) alternative.

Digital options $P(S) \equiv H(S-K)$ can be replaced by a piecewise linear approximation:



or something much smoother such as $\Phi\left(\frac{S-K}{\delta}\right)$ which has an $O(\delta^2)$ bias

Discontinuous payoffs

Implicitly-smoothed payoffs use conditional expectations.

My favourite is for barrier options, where a Brownian Bridge conditional expectation computes the probability that the path has crossed the barrier within a timestep.
(see Glasserman's book, pp. 366-370)

This improves the weak convergence to first order, and also makes the payoff differentiable.

Discontinuous payoffs

With digital options, can stop the path simulation one timestep before maturity.

Conditional on the value S_{N-1}, an Euler discretisation for the final timestep has the form

$$S_N = S_{N-1} + \mu_{N-1}\,\Delta t + \sigma_{N-1}\,\Delta W_{N-1}$$

which gives a (multi-variate) Normal p.d.f. for S_N

In simple cases it is possible to analytically evaluate

$$\mathbb{E}\left[\, P(S_N) \,|\, S_{N-1} \,\right]$$

and this will be a smooth function of S_{N-1} so we can use the pathwise sensitivity method.

Discontinuous payoffs

Continuing this digital example, in more complicated multi-dimensional cases it is not possible to analytically evaluate the conditional expectation.

Instead, we can apply the Likelihood Ratio Method for the final step – I called this the "vibrato" method because of the uncertainty in the final value S_N

Need to read my paper for full details for multi-dimensional applications – I'll give an outline for a scalar SDE.

Its main weakness is that the variance is $O(\Delta t^{-1/2})$, but it is much better than the $O(\Delta t^{-1})$ variance of the standard Likelihood Ratio Method, and you get the full benefit of adjoints.

Discontinuous payoffs

Conditional on the value of S_{N-1}, the probability distribution for S_N is

$$p_s(\theta, S_{N-1}; S)$$

where p_s is a Normal distribution with a mean and variance which depend on S_{N-1}, and perhaps also directly on θ.

Hence, using the LRM approach,

$$\frac{\partial}{\partial \theta} \mathbb{E}\left[P(S_N) \,|\, S_{N-1} \right] = \mathbb{E}\left[P(S_N) \left(\frac{\partial \log p_s}{\partial \theta} \right)_{total} | \, S_{N-1} \right]$$

with

$$\left(\frac{\partial \log p_s}{\partial \theta} \right)_{total} = \frac{\partial \log p_s}{\partial S_{N-1}} \frac{\partial S_{N-1}}{\partial \theta} + \frac{\partial \log p_s}{\partial \theta}$$

The conditional expectation can be estimated by averaging over a number of samples for Z_{N-1}, the random number used for the final timestep.

Discontinuous payoffs

Re-arranging we get

$$\frac{\partial}{\partial\theta}\,\mathbb{E}\left[\,P(S_N)\,|\,S_{N-1}\,\right] \;=\; \mathbb{E}\left[\,P(S_N)\,\frac{\partial\log p_s}{\partial S_{N-1}}\,|\,S_{N-1}\,\right]\frac{\partial S}{\partial\theta}$$

$$+\;\; \mathbb{E}\left[\,P(S_N)\,\frac{\partial\log p_s}{\partial\theta}\,|\,S_{N-1}\,\right]$$

so for the adjoint version we set

$$\overline{S}_{N-1} \;=\; \mathbb{E}\left[\,P(S_N)\,\frac{\partial\log p_s}{\partial S_{N-1}}\,|\,S_{N-1}\,\right]$$

$$\overline{\theta} \;=\; \mathbb{E}\left[\,P(S_N)\,\frac{\partial\log p_s}{\partial\theta}\,|\,S_{N-1}\,\right]$$

and then continue backwards down the path in the usual way.

Conclusions

- adjoints can be very efficient for option pricing, calibration and sensitivity analysis
- same result as "standard" approach but a much lower computational cost
- basic elements of discrete adjoint analysis are very simple, although real applications can get quite complex
- automatic differentiation ideas are very important, even if you don't use AD software

Further reading

M.B. Giles and P. Glasserman. 'Smoking adjoints: fast Monte Carlo Greeks', RISK, 19(1):88-92, 2006

M.B. Giles and P. Glasserman. 'Smoking Adjoints: fast evaluation of Greeks in Monte Carlo calculations'. Numerical Analysis report NA-05/15, 2005.
— *original RISK paper, and longer version with appendix on AD*

M. Leclerc, Q. Liang, I. Schneider. 'Fast Monte Carlo Bermudan Greeks', RISK, 22(7):84-88, 2009.

L. Capriotti and M.B. Giles. 'Fast correlation Greeks by adjoint algorithmic differentiation', RISK, 23(5):77-83, 2010
— *correlation Greeks and binning*

L. Capriotti and M.B. Giles. 'Adjoint Greeks made easy', RISK, 25(9):96-102, 2012
— *use of AD*

Further reading

M.B. Giles. 'Monte Carlo evaluation of sensitivities in computational finance'. Numerical Analysis report NA-07/12, 2007.
— *use of AD, and introduction of Vibrato idea*

M.B. Giles. 'Vibrato Monte Carlo sensitivities'. In Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer, 2009.
— *Vibrato Monte Carlo for discontinuous payoffs*

C. Kaebe, J.H. Maruhn and E.W. Sachs. 'Adjoint-based Monte Carlo calibration of financial market models'. Finance and Stochastics, 13(3):351-379, 2009.
— *adjoint Monte Carlo sensitivities and calibration*

Further reading

M.B. Giles 'On the iterative solution of adjoint equations', pp.145-152 in Automatic Differentiation: From Simulation to Optimization, G. Corliss, C. Faure, A. Griewank, L. Hascoet, U. Naumann, editors, Springer-Verlag, 2001.
— *adjoint treatment of time-marching and fixed point iteration*

M.B. Giles. 'Collected matrix derivative results for forward and reverse mode algorithmic differentiation'. In Advances in Automatic Differentiation, Springer, 2008.

M.B. Giles. 'An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation'. Numerical Analysis report NA-08/01, 2008.
— *two papers on adjoint linear algebra, second has MATLAB code and tips on code development and validation*