Monte Carlo evaluation of sensitivities in computational finance

Mike Giles

giles@comlab.ox.ac.uk

Oxford University Computing Laboratory

Oxford-Man Institute of Quantitative Finance

Outline

- stochastic differential equations and Monte Carlo estimation
- different approaches to computing sensitivities
- adjoint pathwise sensitivities
- use of automatic differentiation
- “vibrato” Monte Carlo for non-differentiable payoffs

SDEs in Finance

In computational finance, stochastic differential equations are used to model the behaviour of

- stocks
- interest rates
- exchange rates
- weather
- electricity/gas demand
- crude oil prices
- …

The stochastic term accounts for the uncertainty of unpredictable day-to-day events.

SDEs in Finance

These models are then used to calculate “fair” prices for a huge range of financial options:

- an option to sell a stock portfolio at a specific price in 2 years time
- an option to buy aviation fuel at a specific price in 6 months time
- an option to sell US dollars at a specific exchange rate in 3 years time

In most cases, the buyer of the financial option is trying to reduce their risk.

SDEs in Finance

Examples:

- Geometric Brownian motion (Black-Scholes model for stock prices)

$$\mathrm{d}S = r\,S\,\mathrm{d}t + \sigma\,S\,\mathrm{d}W$$

- Cox-Ingersoll-Ross model (interest rates)

$$\mathrm{d}r = \alpha(b - r)\,\mathrm{d}t + \sigma\,\sqrt{r}\,\mathrm{d}W$$

- Heston stochastic volatility model (stock prices)

$$\begin{aligned} \mathrm{d}S &= r\,S\,\mathrm{d}t + \sqrt{V}\,S\,\mathrm{d}W_1 \\ \mathrm{d}V &= \lambda\,(\sigma^2 - V)\,\mathrm{d}t + \xi\,\sqrt{V}\,\mathrm{d}W_2 \end{aligned}$$

with correlation ρ between $\mathrm{d}W_1$ and $\mathrm{d}W_2$

Generic Problem

Stochastic differential equation with general drift and volatility terms:

$$\mathrm{d}S(t) = a(S,t)\,\mathrm{d}t + b(S,t)\,\mathrm{d}W(t)$$

$W(t)$ is a Wiener variable with the properties that for any $q<r<s<t$, $W(t)-W(s)$ is Normally distributed with mean 0 and variance $t-s$, independent of $W(r)-W(q)$.

In many finance applications, we want to compute the expected value of an option dependent on the terminal state

$$V = \mathbb{E}[f(S(T))].$$

Note: the drift and volatility functions are almost always differentiable, but the payoff $f(S)$ is often not.

Monte Carlo Estimation

Euler discretisation with timestep h:

$$\widehat{S}_{n+1} = \widehat{S}_n + a(\widehat{S}_n, t_n)\, h + b(\widehat{S}_n, t_n)\, \Delta W_n$$

Simplest Monte Carlo estimator for expected payoff is an average of M independent path simulations:

$$M^{-1} \sum_{i=1}^{M} f(\widehat{S}^{(i)}_{T/h})$$

Key idea is very simple, but in practice it gets much more complicated through enhancements to reduce the variance of the estimator.

Monte Carlo Estimation

In the simplest case of Geometric Brownian motion

$$\mathrm{d}S(t) = r\,S\,\mathrm{d}t + \sigma\,S\,\mathrm{d}W(t),$$

the SDE can be solved analytically to give

$$S(T) = S_0 \exp\left(\left(r - \tfrac{1}{2}\sigma^2\right)T + \sigma\,W(T)\right).$$

In this case, we can avoid the Euler discretisation and directly sample $W(T)$ to get

$$V \equiv \mathbb{E}\left[f(S(T))\right] \approx M^{-1}\sum_{i=1}^{M} f(S^{(i)})$$

– will use this to explain approaches to calculating sensitivities

Hedging

- Casinos make money consistently by having lots of customers on different games (statistically unlikely to all be lucky) and a large house margin (expected profit)
- Banks don’t have as many customers/markets, and the margins are smaller, so to get consistent profits they use hedging
- Basic idea: hold a portfolio of options so that the value of the portfolio isn’t affected (to leading order) by changes in stock prices, interest rates, etc.
- This risk management is very important and affects the financial reserves which the banks are forced to maintain

Hedging

Suppose a portfolio has amount c_j of option with payoff f_j:

$$P = \sum_j c_j \, f_j$$

What is the expected sensitivity to changes in interest rate?

$$\frac{\partial}{\partial r} \mathbb{E}[P] = \sum_j c_j \, \frac{\partial}{\partial r} \mathbb{E}[f_j]$$

By choosing the c_j so this is zero, the bank eliminates its risk to unexpected interest rate changes.

To eliminate its risk to other unpredictable changes the bank needs first (and sometimes second) derivatives of the expected value with respect to various parameters

Evaluating sensitivities

Simplest approach is to use a finite difference approximation,

$$\frac{\partial V}{\partial \theta} \approx \frac{V(\theta+\Delta\theta) - V(\theta-\Delta\theta)}{2\,\Delta\theta},$$

$$\frac{\partial^2 V}{\partial \theta^2} \approx \frac{V(\theta+\Delta\theta) - 2\,V(\theta) + V(\theta-\Delta\theta)}{(\Delta\theta)^2}.$$

– very simple, but expensive and inaccurate if $\Delta\theta$ is too big or too small

Evaluating sensitivities

For simple cases where we know the terminal probability distribution and so

$$V \equiv \mathbb{E}\left[f(S(T))\right] = \int f(S)\, p_S(S)\, \mathrm{d}S,$$

we can differentiate this to get

$$\frac{\partial V}{\partial \theta} = \int f\, \frac{\partial p_S}{\partial \theta}\, \mathrm{d}S = \int f\, \frac{\partial (\log p_S)}{\partial \theta}\, p_S\, \mathrm{d}S = \mathbb{E}\left[f\, \frac{\partial (\log p_S)}{\partial \theta}\right].$$

This is the Likelihood Ratio Method – can handle non-differentiable payoffs, but doesn't generalise well to cases where we have to simulate the whole path

Evaluating sensitivities

Alternatively, for simple Geometric Brownian Motion

$$V \equiv \mathbb{E}\left[f(S(T))\right] = \int f(S(T))\, p_W(W)\, \mathrm{d}W,$$

and differentiating this gives

$$\frac{\partial V}{\partial \theta} = \int \frac{\partial f}{\partial S}\, \frac{\partial S(T)}{\partial \theta}\, p_W\, \mathrm{d}W = \mathbb{E}\left[\frac{\partial f}{\partial S}\, \frac{\partial S(T)}{\partial \theta}\right],$$

with $\partial S(T)/\partial\theta$ being evaluated at fixed W.

This is the pathwise sensitivity approach – it can't handle non-differentiable payoffs, but does generalise well to cases where we have to simulate the whole path

Evaluating sensitivities

The generalisation involves differentiating the Euler path discretisation, holding fixed the Brownian increments, to get

$$\frac{\partial \widehat{S}_{n+1}}{\partial \theta} = \left(1 + \frac{\partial a}{\partial S}\, h + \frac{\partial b}{\partial S}\, \Delta W_n\right) \frac{\partial \widehat{S}_n}{\partial \theta} + \frac{\partial a}{\partial \theta}\, h + \frac{\partial b}{\partial \theta}\, \Delta W_n$$

leading to

$$\frac{\partial \widehat{V}}{\partial \theta} = M^{-1} \sum_m \frac{\partial f}{\partial S}(\widehat{S}_N^{(m)})\ \frac{\partial \widehat{S}_N^{(m)}}{\partial \theta}.$$

Adjoint sensitivities

The adjoint approach is an efficient implementation of pathwise sensitivities.

Consider a process in which a vector input α leads to a final state vector S which is used to compute a scalar payoff P

$$\alpha \quad \longrightarrow \quad S \quad \longrightarrow \quad P$$

Taking $\dot{\alpha}, \dot{S}, \dot{P}$ to be the derivatives w.r.t. j^{th} component of α, then

$$\dot{S} = \frac{\partial S}{\partial \alpha}\, \dot{\alpha}, \qquad \dot{P} = \frac{\partial P}{\partial S}\, \dot{S},$$

and hence

$$\dot{P} = \frac{\partial P}{\partial S}\, \frac{\partial S}{\partial \alpha}\, \dot{\alpha}.$$

Adjoint sensitivities

Alternatively, defining $\overline{\alpha}, \overline{S}, \overline{P}$ to be the derivatives of P with respect to α, S, P, then

$$\overline{\alpha} \stackrel{\text{def}}{=} \left(\frac{\partial P}{\partial \alpha}\right)^T = \left(\frac{\partial P}{\partial S}\frac{\partial S}{\partial \alpha}\right)^T = \left(\frac{\partial S}{\partial \alpha}\right)^T \overline{S},$$

and similarly

$$\overline{S} = \left(\frac{\partial P}{\partial S}\right)^T \overline{P},$$

giving

$$\overline{\alpha} = \left(\frac{\partial S}{\partial \alpha}\right)^T \left(\frac{\partial P}{\partial S}\right)^T \overline{P}.$$

Adjoint sensitivities

The two are mathematically equivalent, since

$$\dot{P} = \frac{\partial P}{\partial \alpha}\,\dot{\alpha} = \overline{\alpha}^T \dot{\alpha} = \overline{\alpha}_j$$

but the adjoint approach approach is much cheaper because a single calculation gives $\overline{\alpha}$, the sensitivity of P to each one of the elements of α, whereas in the standard approach a separate calculation would have to be performed for each element.

Adjoint sensitivities

Note that the standard approach goes forward

$$\dot{\alpha} \quad \longrightarrow \quad \dot{S} \quad \longrightarrow \quad \dot{P}$$

while the adjoint approach does the reverse

$$\overline{\alpha} \quad \longleftarrow \quad \overline{S} \quad \longleftarrow \quad \overline{P}.$$

These correspond to the forward and reverse modes of AD (Automatic Differentiation) and we'll now see how these extend to whole computer programs

Automatic Differentiation

A computer instruction creates an additional new value:

$$\mathbf{u}^n \;=\; \mathbf{f}^n(\mathbf{u}^{n-1}) \;\equiv\; \begin{pmatrix} \mathbf{u}^{n-1} \\ f_n(\mathbf{u}^{n-1}) \end{pmatrix},$$

A computer program is the composition of N such steps:

$$\mathbf{u}^N = \mathbf{f}^N \circ \mathbf{f}^{N-1} \circ \ldots \circ \mathbf{f}^2 \circ \mathbf{f}^1(\mathbf{u}^0).$$

Automatic Differentiation

In forward mode, differentiation w.r.t. one element of the input vector gives

$$\dot{\mathbf{u}}^n = D^n \, \dot{\mathbf{u}}^{n-1}, \quad D^n \equiv \begin{pmatrix} I^{n-1} \\ \partial f_n / \partial \mathbf{u}^{n-1} \end{pmatrix},$$

and hence

$$\dot{\mathbf{u}}^N = D^N \, D^{N-1} \ldots \, D^2 \, D^1 \, \dot{\mathbf{u}}^0$$

Automatic Differentiation

In reverse mode, we consider the sensitivity of one element of the output vector, to get

$$\left(\overline{\mathbf{u}}^{n-1}\right)^T \equiv \frac{\partial u_i^N}{\partial \mathbf{u}^{n-1}} = \frac{\partial u_i^N}{\partial \mathbf{u}^n}\,\frac{\partial \mathbf{u}^n}{\partial \mathbf{u}^{n-1}} = \left(\overline{\mathbf{u}}^n\right)^T D^n,$$

$$\Longrightarrow \quad \overline{\mathbf{u}}^{n-1} = \left(D^n\right)^T \overline{\mathbf{u}}^n.$$

and hence

$$\overline{\mathbf{u}}^0 = \left(D^1\right)^T \left(D^2\right)^T \ldots \left(D^{N-1}\right)^T \left(D^N\right)^T \overline{\mathbf{u}}^N.$$

Note: need to go forward through original calculation to compute/store the D^n, then go in reverse to compute $\overline{\mathbf{u}}^n$

Automatic Differentiation

At the level of a single instruction

$$c = f(a, b)$$

the forward mode is

$$\begin{pmatrix} \dot{a} \\ \dot{b} \\ \dot{c} \end{pmatrix}^{n} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial f}{\partial a} & \frac{\partial f}{\partial b} \end{pmatrix} \begin{pmatrix} \dot{a} \\ \dot{b} \end{pmatrix}^{n-1}$$

and so the reverse mode is

$$\begin{pmatrix} \overline{a} \\ \overline{b} \end{pmatrix}^{n-1} = \begin{pmatrix} 1 & 0 & \frac{\partial f}{\partial a} \\ 0 & 1 & \frac{\partial f}{\partial b} \end{pmatrix} \begin{pmatrix} \overline{a} \\ \overline{b} \\ \overline{c} \end{pmatrix}^{n}$$

Automatic Differentiation

This gives a prescriptive algorithm for reverse mode differentiation.

Again the reverse mode is much more efficient if we want the sensitivity of a single output to multiple inputs.

The storage of the D^n is minor for SDEs – much more of a concern for PDEs. There are also extra complexities when solving implicit equations through a fixed point iteration.

Automatic Differentiation

Manual implementation of the forward/reverse mode algorithms is possible but tedious.

Fortunately, automated tools have been developed, following one of two approaches:

- operator overloading (ADOL-C, FADBAD++)
- source code transformation (Tapenade, TAF/TAC++, ADIFOR)

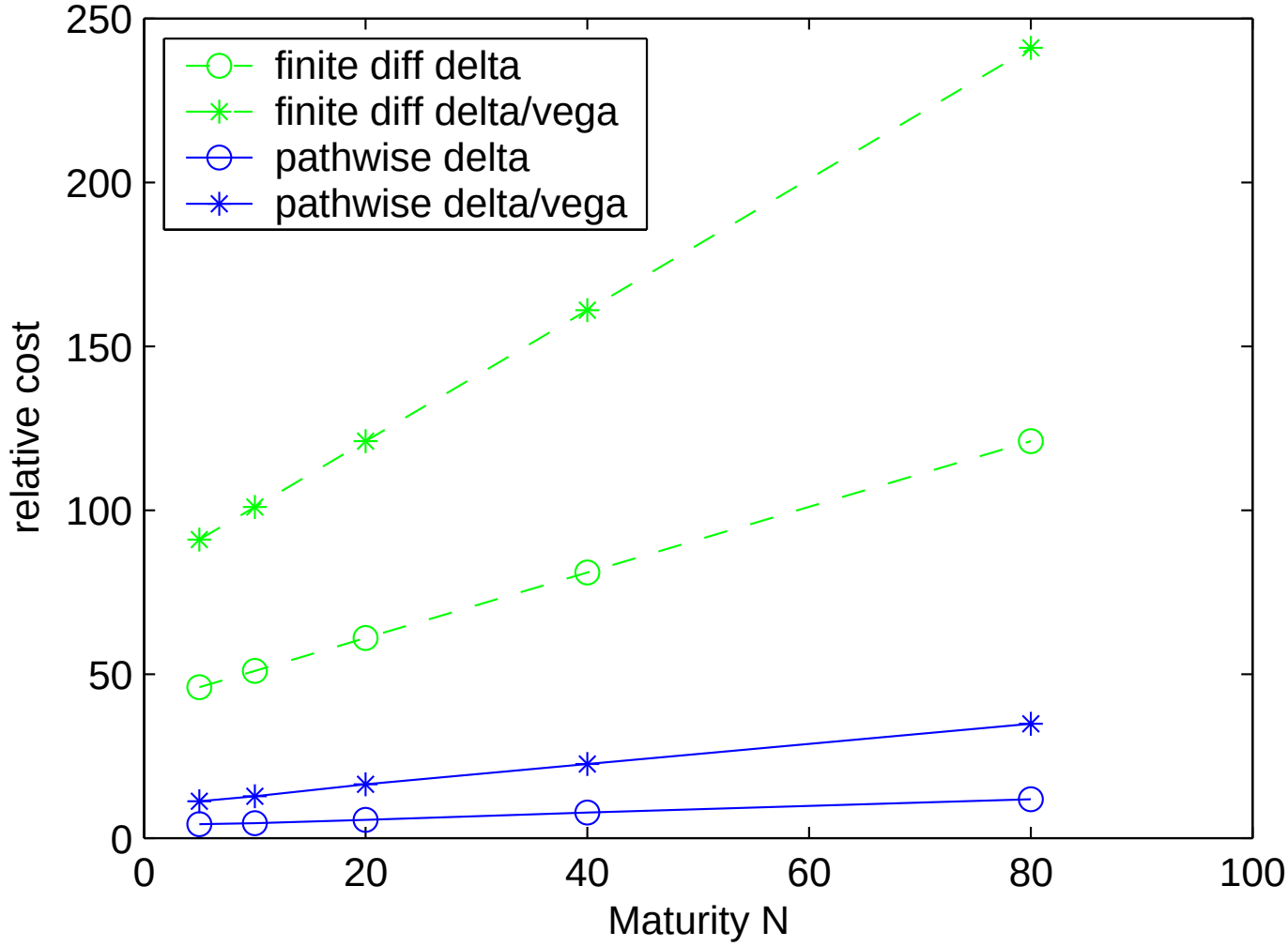
My personal experience is with Tapenade for Fortran, and FADBAD++ for C++. Both are easy to use, Tapenade is as efficient as hand-coded, FADBAD++ less so.

LIBOR Test Application

- model for behaviour of London InterBank Overnight interest Rates
- used to price various options which depend on current and future interest rates
- test problem performs N timesteps with a vector of $N+40$ forward rates, and computes the sensitivity of a portfolio of swaptions to changes in both initial forward rates and volatilities

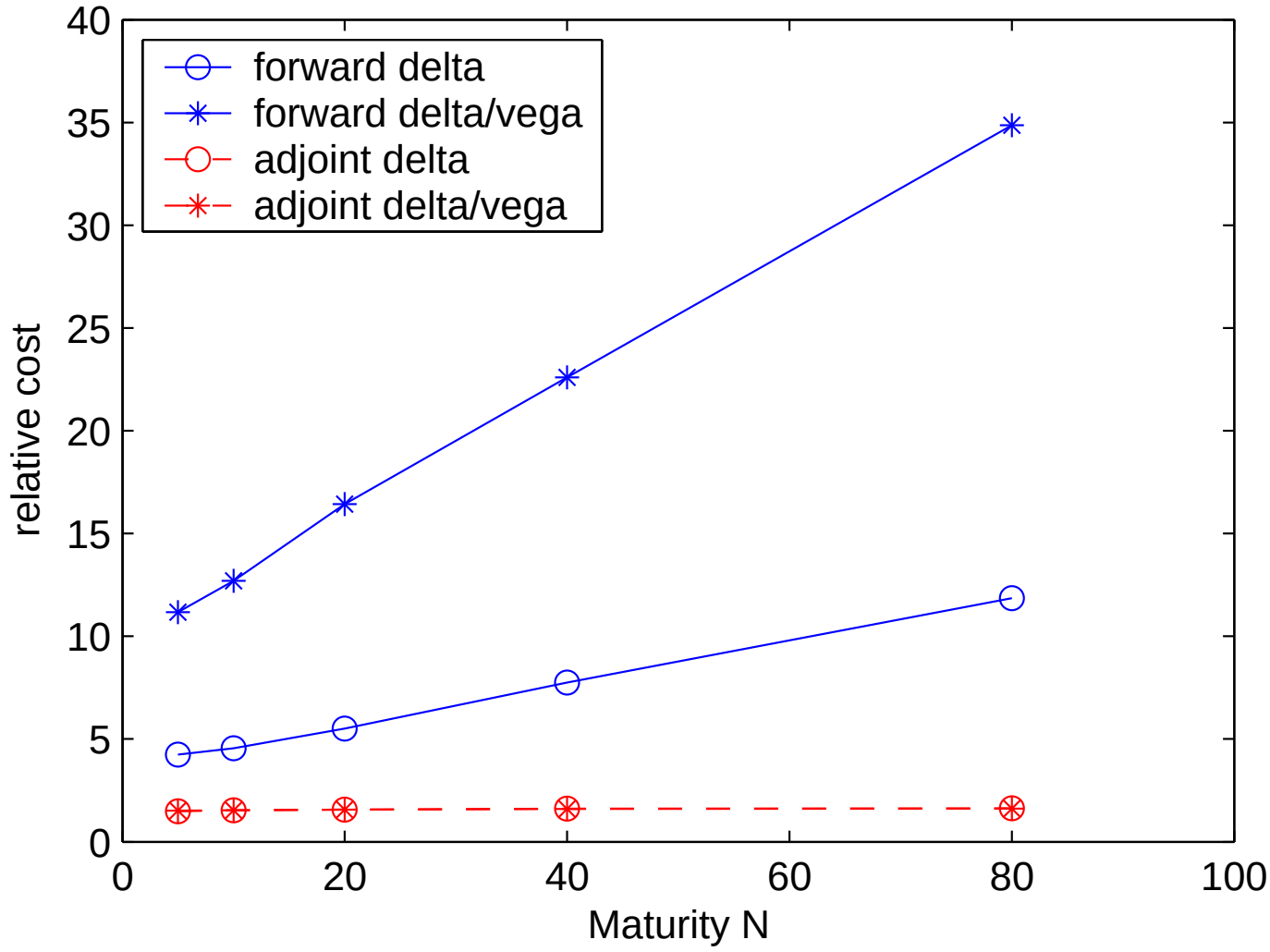
LIBOR Test Application

Finite differences versus forward pathwise sensitivities:



LIBOR Test Application

Hand-coded forward versus adjoint pathwise sensitivities:



LIBOR Test Application

Timings per path for $N = 40$ – the hybrid version uses hand-coded for the path and FADBAD++ for the payoff

milliseconds/path	Gnu g++	Intel `icc`
original	0.37	0.10
hand-coded forward	0.97	0.52
hand-coded reverse	0.47	0.19
FADBAD++ forward	4.30	5.00
FADBAD++ reverse	6.20	4.86
hybrid forward	1.02	0.63
hybrid reverse	0.65	0.35
TAC++ forward	1.36	0.85
TAC++ reverse	1.28	0.45

Conclusions so far

- an adjoint implementation of pathwise sensitivities can be very efficient – gives all first derivatives for less than double the cost of the original portfolio evaluation
- automatic differentiation tools can be used to avoid tedious programming, or to check its correctness
- one remaining problem – what if payoff is not differentiable?
  - Likelihood Ratio Method doesn't extend well to path-dependent applications
  - Malliavin calculus approach is complex and expensive for multiple sensitivities

Vibrato Monte Carlo

- new idea, based on Glasserman example of conditional expectation
- output of each SDE path calculation becomes a narrow Normal distribution rather than a point value
- combine pathwise sensitivity for the differentiable SDE, with LRM for the non-differentiable payoff
- avoiding the differentiation of the payoff also simplifies the implementation in real-world setting

Vibrato Monte Carlo

Final timestep of Euler path discretisation is

$$\widehat{S}_N = \widehat{S}_{N-1} + a(\widehat{S}_{N-1}, t_{N-1})\, h + b(\widehat{S}_{N-1}, t_{N-1})\, \Delta W_{N-1}$$

Instead of using random number generator to get a value for ΔW_{N-1}, consider the whole distribution of possible values, so $\widehat{S}_N$ has a Normal distribution with mean

$$\mu(W) = \widehat{S}_{N-1} + a(\widehat{S}_{N-1}, t_{N-1})\, h$$

and standard deviation

$$\sigma(W) = b(\widehat{S}_{N-1}, t_{N-1})\, \sqrt{h}$$

where $W \equiv (\Delta W_0, \Delta W_1, \ldots \Delta W_{N-2})$.

Vibrato Monte Carlo

For a particular path given by a particular vector W, the expected payoff is

$$\mathbb{E}_Z[f(\mu+\sigma\, Z)]$$

where Z is a Normal random variable with zero mean and unit variance.

Averaging over all W then gives the same overall expectation as before.

Note also that, for given W, $\widehat{S}_N$ has a Normal distribution with

$$p_S(\widehat{S}_N) = \frac{1}{\sqrt{2\pi}\,\sigma}\ \exp\left(-\frac{(\widehat{S}_N-\mu)^2}{2\,\sigma^2}\right)$$

Vibrato Monte Carlo

The novelty comes in calculating the sensitivity. For a particular W, we have a Normal probability distribution for $\widehat{S}_N$ and can apply the Likelihood Ratio method to get

$$\frac{\partial}{\partial\theta}\,\mathbb{E}_Z\left[f(\widehat{S}_N)\right] \;=\; \mathbb{E}_Z\left[f(\widehat{S}_N)\;\frac{\partial(\log p_S)}{\partial\theta}\right],$$

where

$$\begin{aligned}\frac{\partial(\log p_S)}{\partial\theta} &= \frac{\partial(\log p_S)}{\partial\mu}\;\frac{\partial\mu}{\partial\theta}+\frac{\partial(\log p_S)}{\partial\sigma}\;\frac{\partial\sigma}{\partial\theta}\\ &= \frac{Z}{\sigma}\;\frac{\partial\mu}{\partial\theta}+\frac{Z^2-1}{\sigma}\;\frac{\partial\sigma}{\partial\theta}.\end{aligned}$$

Averaging over all W then gives the expected sensitivity. Written paper gives ways of reducing the variance.

Vibrato Monte Carlo

For each W, in forward mode we have

$$\alpha,\ \dot{\alpha} \quad \longrightarrow \quad \mu,\ \dot{\mu},\ \sigma,\ \dot{\sigma} \quad \longrightarrow \quad \text{payoff + sensitivity}$$

- first bit – pathwise sensitivity calculation
- second bit – Likelihood Ratio Method

For maximum efficiency can use adjoint/reverse mode

$$\alpha \quad \longrightarrow \quad \mu,\ \sigma \quad \longrightarrow \quad \text{payoff}$$

$$\overline{\alpha} \quad \longleftarrow \quad \overline{\mu},\ \overline{\sigma} \quad \longleftarrow \quad \text{sensitivity}$$

$\overline{\mu},\ \overline{\sigma}$ are coefficients multiplying $\dot{\mu},\ \dot{\sigma}$ in forward mode

Final Conclusions

Monte Carlo estimation of sensitivities is an important problem in computational finance

Improved methods need ideas from both mathematics

- adjoint technique
- vibrato Monte Carlo
- (multilevel Monte Carlo)

. . . and computer science

- automatic differentiation
- (parallel execution on graphics cards)

Acknowledgements

- Paul Glasserman for collaboration on adjoint technique and discussion on vibrato Monte Carlo
- Ole Stauning for help with FADBAD++
- Michael Vossbeck for TAC++ results

Further information

- `www.comlab.ox.ac.uk/mike.giles/`
- Email: `giles@comlab.ox.ac.uk`