AD in Monte Carlo for finance

Mike Giles

giles@comlab.ox.ac.uk

Oxford University Computing Laboratory

Overview

- overview of computational finance
- stochastic o.d.e.’s
- Monte Carlo simulation
- sensitivity analysis
- use of AD
- future prospects

Computational finance

There are 3 main approaches to the pricing of financial options based on equities, bonds, exchange rates, . . .

- Monte Carlo methods (50%?)
  - simple, flexible
  - efficient for high-dimensional problems
- trees (25%?)
  - simple version of explicit finite differences
- PDE methods (25%?)
  - more complicated
  - efficient for low-dimensional problems
  - excellent for American options (free boundary)

Stochastic ODEs

A generic stochastic ODE is of the form

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

Here $W(t)$ is a Wiener variable (Brownian motion) with the properties:

- for $s < t$, $W(t) - W(s)$ is Normally distributed with mean 0 and variance $t - s$
- for any $q < r < s < t$, $W(t) - W(s)$ is independent of $W(r) - W(q)$

In the multi-dimensional generalisation $X(t)$ and $W(t)$ are both vectors.

Stochastic ODEs

Example: geometric Brownian motion

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

This is the simplest model of the behaviour of a stock price S with $W(t)$ representing the uncertainty of the real world.

The simplest option is a European call (an option to buy at a certain time T and price K) whose "payoff" value is

$$g(S(T)) = e^{-rT}\ \max(0, S-K)$$

What is wanted is the expected (or average) value $E[f]$

- simulate lots of paths
- compute the average payoff

Stochastic ODEs

For the generic stochastic ODE, simplest to use Euler approximation

$$X_{n+1} = X_n + a(X_n, t_n)\,\Delta t + b(X_n, t_n)\,\Delta W_n$$

with each ΔW_n independently Normally distributed with zero mean and variance Δt.

For each path $X_n^{(m)}$ can compute a payoff $g^{(m)}$, and then average these to get

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

Stochastic ODEs

Key foundation is Central Limit Theorem:

If g has mean μ_g and variance σ_g^2, then for large N

$$\overline{g} - \mu_g \sim \; M^{-1/2} \, \sigma_g \, \nu$$

where ν is Normally distributed with zero mean and unit variance.

Hence, there is a 99.9% probability that μ_g lies in the interval

$$\left[\overline{g} - 3M^{-1/2}\sigma_g, \;\; \overline{g} + 3M^{-1/2}\sigma_g\right]$$

with σ_g estimated from the sample variance.

Stochastic ODEs

The $M^{-1/2}$ convergence is independent of dimension (very good for high dimensions) but not very rapid

In practice, lots of techniques are used to reduce the variance:

- antithetic variables
- control variates
- stratified sampling
- importance sampling
- quasi Monte Carlo methods

. . . but these are not relevant to this talk

Greeks

What is relevant is that we don't just want to know the expected value of some payoff

$$V = E[g(S(T)].$$

We also want to know a whole range of "Greeks" corresponding to first (and second) derivatives of V with respect to various parameters:

$$\Delta = \frac{\partial V}{\partial S_0}, \qquad \Gamma = \frac{\partial^2 V}{\partial S_0^2},$$

$$\rho = \frac{\partial V}{\partial r}, \qquad \text{Vega} = \frac{\partial V}{\partial \sigma}.$$

These are needed for hedging (cancels out uncertainty to leading order) and for risk analysis.

Finite difference sensitivities

If $V(\theta) = E[g(S(T))]$ for a particular value of an input parameter θ, and sufficiently differentiable, then the sensitivity $\frac{\partial V}{\partial \theta}$ can be approximated by one-sided finite difference

$$\frac{\partial V}{\partial \theta} = \frac{V(\theta+\Delta\theta) - V(\theta)}{\Delta\theta} + O(\Delta\theta)$$

or by central finite difference

$$\frac{\partial V}{\partial \theta} = \frac{V(\theta+\Delta\theta) - V(\theta-\Delta\theta)}{2\Delta\theta} + O((\Delta\theta)^2)$$

Finite difference sensitivities

The clear advantage of this approach is that it is very simple to implement (hence the most popular in practice?)

However, the disadvantages are:

- expensive (2 extra sets of calculations for central differences)
- significant bias error if $\Delta\theta$ too large
- large variance if $g(S(T))$ discontinuous and $\Delta\theta$ small

Pathwise sensitivities

Under certain conditions (e.g. g, a and b are continuous and piecewise differentiable)

$$\frac{\partial}{\partial\theta}\, E[g(X(T))] = E\left[\frac{\partial g(X(T))}{\partial\theta}\right] = E\left[\frac{\partial g}{\partial X}\,\frac{\partial X(T)}{\partial\theta}\right].$$

with $\dfrac{\partial X(T)}{\partial\theta}$ computed by differentiating the path evolution.

Pros:

- less expensive (1 cheap calculation for each sensitivity)
- no bias

Cons:

- more difficult to implement

Generic adjoint approach

Returning to the generic stochastic o.d.e.

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

an Euler approximation gives

$$X(n{+}1) = F_n(X(n))$$

Defining $\Delta(n) = \dfrac{\partial X(n)}{\partial X(0)}$, then

$$\Delta(n{+}1) = D(n)\,\Delta(n), \qquad D(n) \equiv \frac{\partial F_n(X(n))}{\partial X(n)},$$

and hence

$$\frac{\partial g(X(N))}{\partial X(0)} = \frac{\partial g(X(N))}{\partial X(N)}\,\Delta(N) = \frac{\partial g}{\partial X} D(N{-}1) D(N{-}2) \ldots D(0)\Delta(0)$$

Generic adjoint approach

If X is m-dimensional, then $D(n)$ is an $m \times m$ matrix, and the overall computational cost is $O(Nm^3)$.

Alternatively,

$$\frac{\partial g(X(N))}{\partial X(0)} = \frac{\partial g}{\partial X}\, D(N{-}1)\, D(N{-}2) \cdots D(0)\, \Delta(0) = V(0)^\top \Delta(0),$$

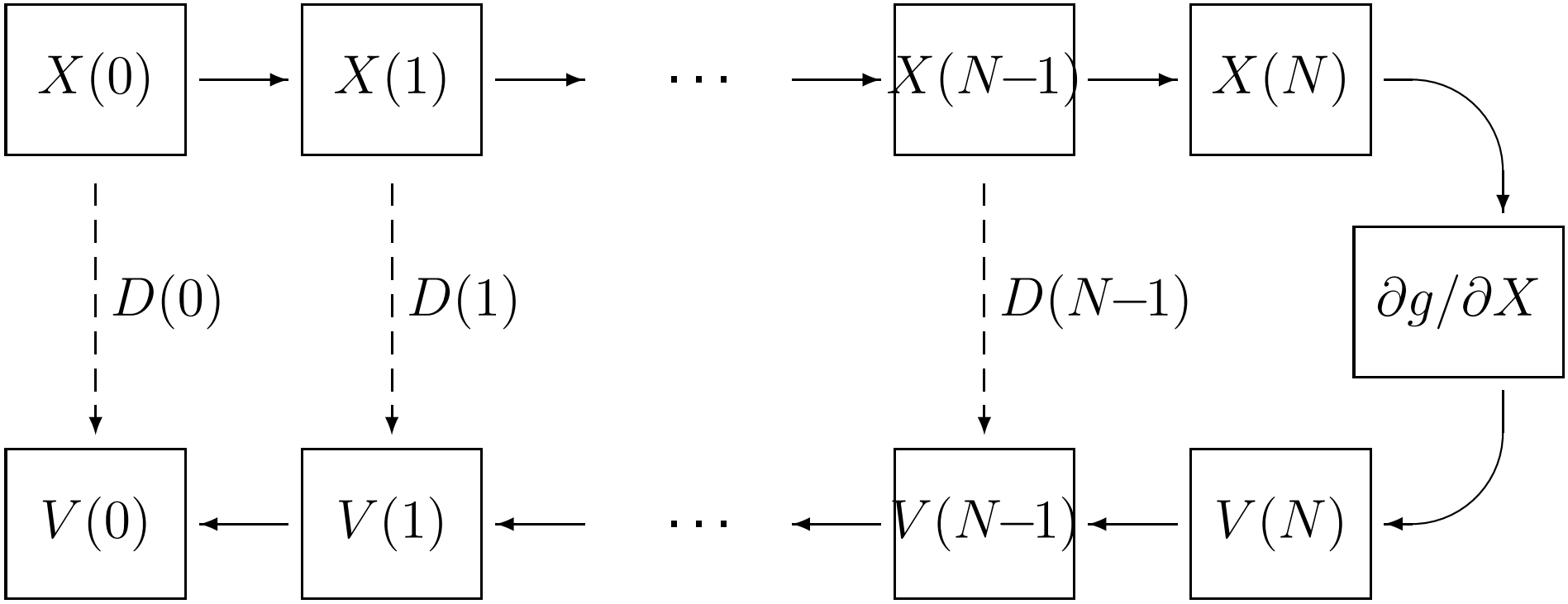
where adjoint $V(n) = \left(\dfrac{\partial g(X(N))}{\partial X(n)} \right)^\top$ is calculated from

$$V(n) = D(n)^\top V(n{+}1), \quad V(N) = \left(\frac{\partial g}{\partial X(N)} \right)^\top,$$

at a computational cost which is $O(Nm^2)$.

Generic adjoint approach

Usual flow of data within the forward/reverse path calculations:



– memory requirements are not significant because data only needs to be stored for the current path.

Generic adjoint approach

To calculate the sensitivity to other parameters, consider a generic parameter θ. Defining $\Theta(n) = \partial X(n)/\partial\theta$, then

$$\Theta(n+1) = \frac{\partial F_n}{\partial X}\,\Theta(n) + \frac{\partial F_n}{\partial \theta} \equiv D(n)\,\Theta(n) + B(n),$$

and hence

$$\begin{aligned}
\frac{\partial g}{\partial \theta} &= \frac{\partial g}{\partial X(N)}\,\Theta(N) \\
&= \frac{\partial g}{\partial X(N)}\Big\{B(N-1) + D(N-1)B(N-2) + \ldots \\
&\qquad\qquad + D(N-1)D(N-2)\ldots D(1)B(0)\Big\} \\
&= \sum_{n=0}^{N-1} V(n+1)^{\top} B(n).
\end{aligned}$$

Generic adjoint approach

Different θ's have different B's, but same V's

$$\Longrightarrow \text{ Computational cost} \simeq Nm^2 \; + \; Nm \times \#\text{ parameters},$$

compared to the standard forward approach for which

$$\text{Computational cost} \simeq Nm^2 \times \#\text{ parameters}.$$

However, the adjoint approach only gives the sensitivity of one output, whereas the forward approach can give the sensitivities of multiple outputs for little additional cost.

Generic adjoint approach

Defining $G(n) = \partial^2 X(n)/\partial X_j(0)\partial X_k(0)$ for a particular (j,k) it can be shown that

$$G(n+1) = D(n)\,G(n) + C(n),$$

where $C(n)$ is a complicated quadratic function of $\Delta(n)$.

Hence, pathwise Gammas can be computed efficiently by doing a forward calculation of Δ, followed by an adjoint calculation to compute

$$\sum_{n=0}^{N-1} V(n+1)^\top C(n),$$

for each pair (j,k), at a savings of factor $O(m)$ relative to a standard forward approach.

LIBOR Market Model

This example models the evolution of future interest rates; an important application and a representative example.

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

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

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

For simplicity, we keep $L_i(t)$ constant for $t > T_i$, and take the volatilities to be a function of the time to maturity,

$$\sigma_i(t) = \sigma_{i-\eta(t)+1}(0).$$

LMM implementation

Applying the Euler scheme to the logarithms of the forward rates yields

$$L_i(n+1) = L_i(n) \exp\left([\mu_i(L(n)) - \|\sigma_i\|^2/2]h + \sigma_i^\top Z(n+1)\sqrt{h} \right).$$

For efficiency, we first compute

$$S_i(n) = \sum_{k=\eta(t)}^{i} \frac{\sigma_k \delta_k L_k(n)}{1 + \delta_k L_k(n)},$$

and then obtain $\mu_i = \sigma_i^\top S_i$.

Each timestep, there is an $O(m)$ cost in computing the S_i's, and then an $O(m)$ cost in updating the L_i's.

LMM implementation

Defining $\Delta_{ij}(n) = \partial L_i(n)/\partial L_j(0)$, differentiating the Euler scheme yields

$$\Delta_{ij}(n+1) = \frac{L_i(n+1)}{L_i(n)}\,\Delta_{ij}(n) + L_i(n+1)\,\sigma_i^\top\,S_{ij}(n)\,h,$$

where

$$S_{ij}(n) = \sum_{k=\eta(nh)}^{i} \frac{\sigma_k\,\delta_k\,\Delta_{kj}(n)}{(1+\delta_k L_k(n))^2}.$$

Each timestep, there is an $O(m^2)$ cost in computing the S_{ij}'s, and then an $O(m^2)$ cost in updating the Δ_{ij}'s.

(Note: programming implementation requires only multiplication and addition – very rapid on modern CPU's).

LMM implementation

Working through the details of the adjoint formulation, one eventually finds that $V_i(n) = V_i(n+1)$ for $i < \eta(nh)$, and

$$V_i(n) = \frac{L_i(n+1)}{L_i(n)} V_i(n+1) + \frac{\sigma_i^\top \delta_i \, h}{(1+\delta_i L_i(n))^2} \sum_{j=i}^{m} L_j(n+1) V_j(n+1) \sigma_j$$

for $i \geq \eta(nh)$.

Each timestep, there is an $O(m)$ cost in computing the summations, and then an $O(m)$ cost in updating the V_i's.

The correctness of the formulation is verified by checking it gives the same sensitivities as the forward calculation.

LMM results

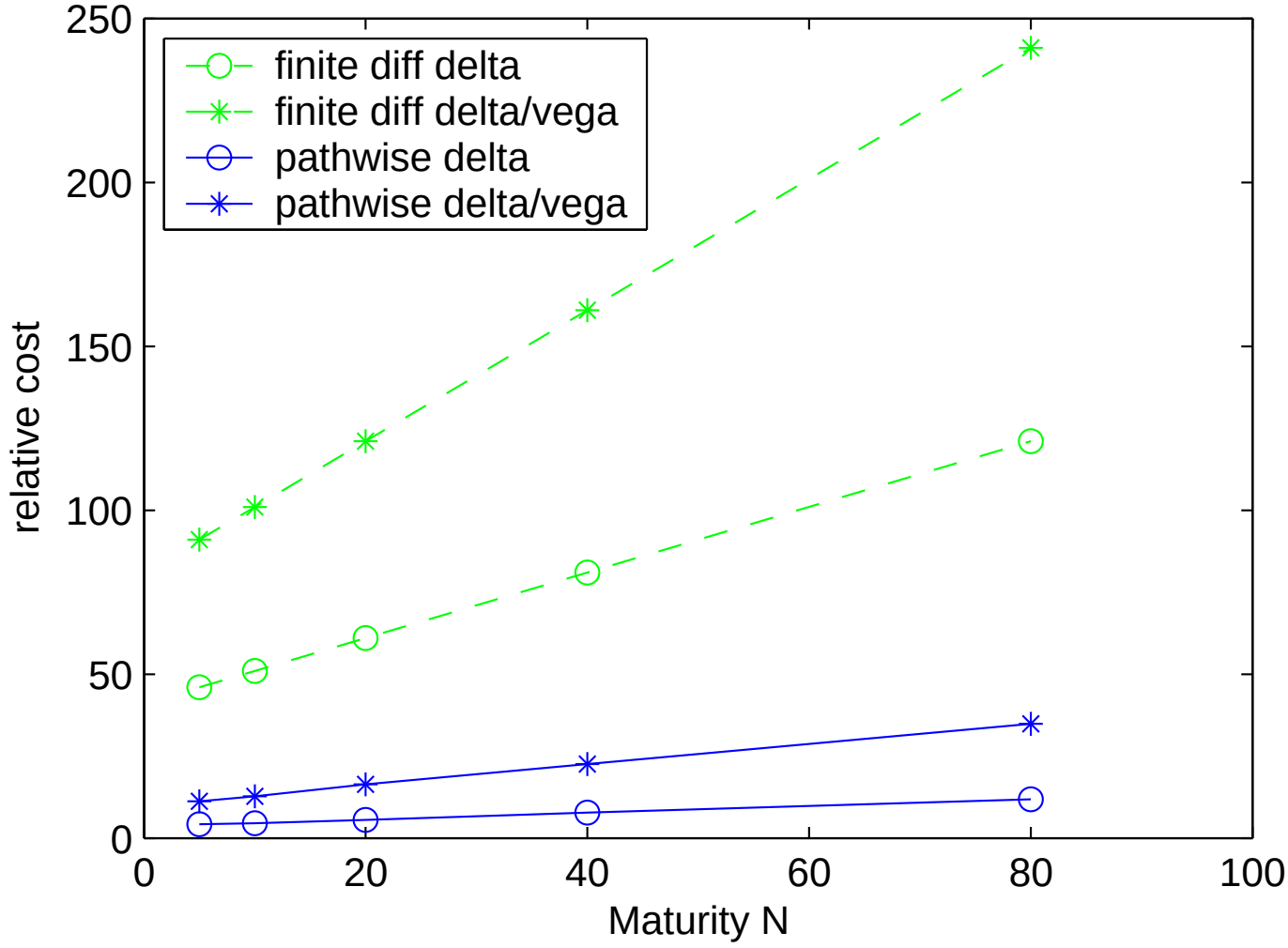
LMM portfolio has 15 swaptions all expiring at the same time, N periods in the future, involving payments/rates over an additional 40 periods in the future.

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.

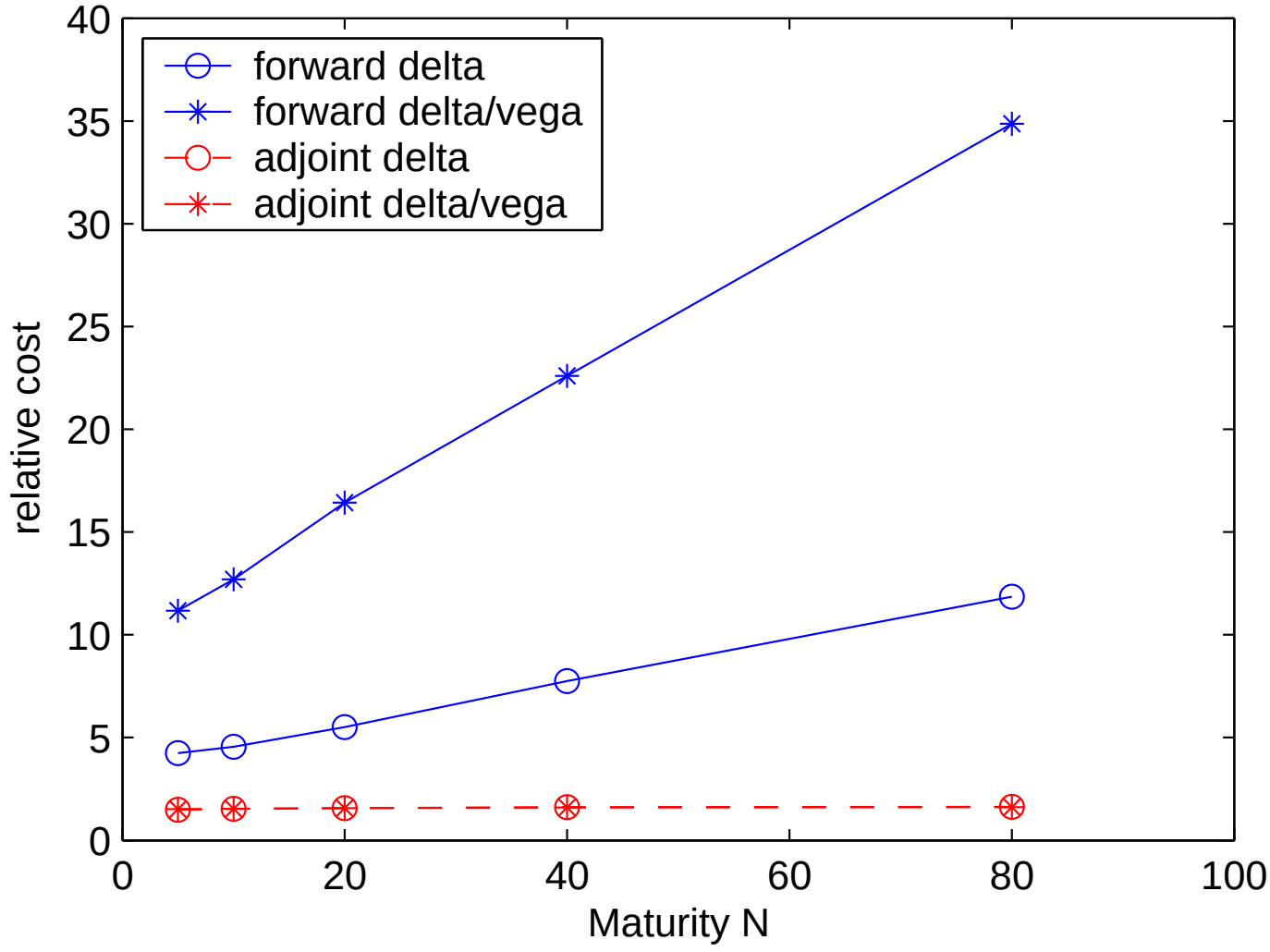
LMM results

Finite differences versus forward pathwise sensitivities:



LMM results

Forward versus adjoint pathwise sensitivities:



AD results

The figures show my hand-coded implementations.

FastOpt have produced preliminary timings using TAC++ (C/C++ version of TAF, still under development)

Timings for 120 deltas and 120 vegas:

	forward	reverse
hand-coded	35	1.3
TAC + gcc	240?	5.3
TAC + icc	???	3.0

AD in future?

First, some numbers:

- 10^4 – 10^6 paths
- 20 – 200 timesteps
- 20 – 2000 operations per timestep
- 1 – 100 state variables

Two good solutions:

- complete taping of each individual path
  (would probably fit within L2/L3 cache)
- store just state variables on initial forward pass,
  then recalculate/tape each timestep
  (would probably fit within L1 cache)

AD in future?

The ideal (which is what I did in hand-coded version)

- in forward pass, for each timestep store
  - state variables
  - all results of “expensive” operations (e.g. exponential, inverse)
- in reverse pass, recalculation only requires “inexpensive” operations (e.g. addition, multiplication)

In principle, a natural tradeoff between memory access and re-computation, and could be automated given some user input on typical values for key loop indices.

However, difficult to develop generic tools which are optimal under widely-differing circumstances.

Prospects for the future

- just given presentations at Quant Congresses in London and New York;
- article with Monte Carlo expert (Paul Glasserman) appearing in December issue of Risk;
- CSFB planning to start an internal project;
- HSBC may be interested too;
- have also talked to some of the software vendors;
- hard to predict but it could be an interesting new application area for AD.

Further Information

`www.comlab.ox.ac.uk/mike.giles/finance.html`
— papers and talks on finance applications

`www.comlab.ox.ac.uk/mike.giles/airfoil/`
— paper and codes for a talk on using Tapenade to generate linear/adjoint versions of a simple airfoil code (to be presented at a workshop in Bangalore in Dec '05)