Smoking adjoints: fast Monte Carlo Greeks

Monte Carlo calculation of price sensitivities for hedging is often very time-consuming. Michael Giles and Paul Glasserman develop an adjoint method to accelerate the calculation. The method is particularly effective in estimating sensitivities to a large number of inputs, such as initial rates on a forward curve or points on a volatility surface. The authors apply the method to the Libor market model and show that it is much faster than previous methods

The efficient calculation of price sensitivities continues to be among the greatest practical challenges facing users of Monte Carlo methods in the derivatives industry. Computing Greeks is essential to hedging and risk management, but typically requires substantially more computing time than pricing a derivative. This article shows how an adjoint formulation can be used to accelerate the calculation of the Greeks. This method is particularly well suited to applications requiring sensitivities to a large number of parameters. Examples include interest rate derivatives requiring sensitivities to all initial forward rates and equity derivatives requiring sensitivities to all points on a volatility surface.

The simplest methods for estimating Greeks are based on finite difference approximations, in which a Monte Carlo pricing routine is rerun multiple times at different settings of the input parameters in order to estimate sensitivities to the parameters. In the fixed-income setting, for example, this would mean perturbing each initial forward rate and then rerunning the Monte Carlo simulation to re-price a security or a whole book. The main virtues of this method are that it is straightforward to understand and requires no additional programming. But the bias and variance properties of finite difference estimates can be rather poor, and their computing time requirements grow with the number of input parameters.

Better estimates of price sensitivities can often be derived by using information about model dynamics in a Monte Carlo simulation. Techniques for doing this include the pathwise method and likelihood ratio method, both of which are reviewed in chapter 7 of Glasserman (2004). When applicable, these methods produce unbiased estimates of price sensitivities from a single set of simulated paths, that is, without perturbing any parameters. The pathwise method accomplishes this by differentiating the evolution of the underlying assets or state variables along each path; the likelihood ratio method instead differentiates the transition density of the underlying assets or state variables. In comparison with finite difference estimates, these methods require additional model analysis and programming, but the additional effort is often justified by the improvement in the quality of calculated Greeks.

The adjoint method we develop here applies ideas used in computational fluid dynamics (Giles & Pierce, 2000) to the calculation of pathwise estimates of Greeks. The estimate calculated using the adjoint method is identical to the ordinary pathwise estimate; its potential advantage is therefore computational, rather than statistical. The relative merits of the ordinary (forward) calculation of pathwise Greeks and the adjoint calculation can be summarised as follows: a) the adjoint method is advantageous for calculating the sensitivities of a small number of securities with respect to a large number of parameters; and b) the forward method is advantageous for calculating the sensitivities of many securities with respect to a small number of parameters. The 'small number of securities' in this dichotomy could be an entire book, consisting of many individual securities, so long as the sensitivities to be calculated are for the book as a whole and not for the constituent securities.

The rest of this article is organised as follows. The next section reviews the usual forward calculation of pathwise Greeks and the subsequent section illustrates its application in the Libor market model. We then develop the adjoint method for delta estimates, and extend it to applications such as vega estimation requiring sensitivities to parameters of model dynamics, rather than just sensitivities to initial conditions. We then extend it to gamma estimation. We use the Libor market model as an illustrative example in both settings. Lastly, we present numerical results that illustrate the computational savings offered by the adjoint method.

Pathwise delta: forward method

We start by reviewing the application of the pathwise method for computing price sensitivities in the setting of a multi-dimensional diffusion process satisfying a stochastic differential equation:

$$d\tilde{X}(t) = a\left(\tilde{X}(t)\right)dt + b\left(\tilde{X}(t)\right)dW(t) \tag{1}$$

The process $\tilde{X}$ is m-dimensional, W is a d-dimensional Brownian motion, $a(\cdot)$ takes values in R^m and $b(\cdot)$ takes values in $R^{m\times d}$. For example, $\tilde{X}$ could record a vector of equity prices or – as in the case of the Libor market model, below – a vector of forward rates. We take (1) to be the risk-neutral or otherwise risk-adjusted dynamics of the relevant financial variables. A derivative maturing at time T with discounted payout $g(\tilde{X}(T))$ has price $E[g(\tilde{X}(T)]$, the expected value of the discounted payout.

In a Monte Carlo simulation, the evolution of the process $\tilde{X}$ is usually approximated using an Euler scheme. For simplicity, we take a fixed time step $h = T/N$, with N an integer. We write $X(n)$ for the Euler approximation at time nh, which evolves according to:

$$X(n+1) = X(n) + a\left(X(n)\right)h + b\left(X(n)\right)Z(n+1)\sqrt{h}, \quad X(0) = \tilde{X}(0) \tag{2}$$

where $Z(1)$, $Z(2)$, ... are independent d-dimensional standard normal random vectors. With the normal random variables held fixed, (2) takes the form:

$$X(n+1) = F_n\left(X(n)\right) \tag{3}$$

with F_n a transformation from R^m to R^m.

The price of the derivative with discounted payout function g is estimated using the average of independent replications of $g(X(N))$, $N = T/h$.

Now consider the problem of estimating:

$$\frac{\partial}{\partial X_j(0)} E\left[g\left(\tilde{X}(T)\right)\right]$$

the delta with respect to the jth underlying variable. The pathwise method estimates this delta using:

$$\frac{\partial}{\partial X_j(0)} g\left(\tilde{X}(T)\right)$$

the sensitivity of the discounted payout along the path. This is an unbiased estimate if:

$$E\left[\frac{\partial}{\partial X_j(0)} g\left(\tilde{X}(T)\right)\right] = \frac{\partial}{\partial X_j(0)} E\left[g\left(\tilde{X}(T)\right)\right]$$

that is, if the derivative and expectation can be interchanged.

Conditions for this interchange are discussed in Glasserman (2004), on pages 393–395. Convenient sufficient conditions impose some modest restrictions on the evolution of $\tilde{X}$ and some minimal smoothness on the discounted payout g, such as a Lipschitz condition. If g is Lipschitz, it is differentiable almost everywhere and we may write:

$$\frac{\partial}{\partial X_j(0)} g\left(\tilde{X}(T)\right) = \sum_{i=1}^{m} \frac{\partial g\left(\tilde{X}(T)\right)}{\partial \tilde{X}_i(T)} \frac{\partial \tilde{X}_i(T)}{\partial \tilde{X}_j(0)}$$

Conditions under which $\tilde{X}_i(T)$ is in fact differentiable in $\tilde{X}_i(0)$ are discussed in Protter (1990), on page 250.

Using the Euler scheme (2), we approximate the pathwise derivative estimate using:

$$\sum_{i=1}^{m} \frac{\partial g\left(X(N)\right)}{\partial X_i(N)} \Delta_{ij}(N) \tag{4}$$

with:

$$\Delta_{ij}(n) = \frac{\partial X_i(n)}{\partial X_j(0)}, \quad i, j = 1, \ldots, m$$

Thus, in order to evaluate (4), we need to calculate the state sensitivities $\Delta_{ij}(N)$. We simulate their evolution by differentiating (2) to get:

$$\Delta_{ij}(n+1) = \Delta_{ij}(n) + \sum_{k=1}^{m} \frac{\partial a_i}{\partial x_k} \Delta_{kj}(n) h + \sum_{\ell=1}^{d} \sum_{k=1}^{m} \frac{\partial b_{i\ell}}{\partial x_k} \Delta_{kj}(n) Z_\ell(n+1)\sqrt{h}$$

with a_i denoting the ith component of $a(X(n))$ and $b_{i\ell}$ denoting the (i, ℓ) component of the $b(X(n))$.

We can write this as a matrix recursion by letting $\Delta(n)$ denote the $m \times m$ matrix with entries $\Delta_{ij}(n)$. Let $D(n)$ denote the $m \times m$ matrix with entries:

$$D_{ik}(n) = \delta_{ik} + \frac{\partial a_i}{\partial x_k} h + \sum_{\ell=1}^{d} \frac{\partial b_{i\ell}}{\partial x_k} Z_\ell(n+1)\sqrt{h}$$

where δ_{ik} is one if $i = k$ and zero otherwise. The evolution of Δ can now be written as:

$$\Delta(n+1) = D(n)\Delta(n) \tag{5}$$

with initial condition $\Delta(0) = I$ where I is the $m \times m$ identity matrix. The matrix $D(n)$ is the derivative of the transformation F_n in (3). For large m, propagating this $m \times m$ recursion may add substantially to the computational effort required to simulate the original vector recursion (2).

Libor market model

To help fix ideas, we now specialise to the Libor market model of Brace, Gatarek & Musiela (1997). Fix a set of $m + 1$ bond maturities T_i, $i = 1, \ldots, m + 1$, with spacings $T_{i+1} - T_i = \delta_i$. Let $\tilde{L}_i(t)$ denote the forward Libor rate fixed at time t for the interval $[T_i, T_{i+1})$, $i = 1, \ldots, m$. Let $\eta(t)$ denote the index of the next maturity date as of time t, $T_{\eta(t)-1} \le t < T_{\eta(t)}$. The arbitrage-free dynamics of the forward rates take the form:

1. Structure of the matrix D and its transpose

$$D = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & \times & & & \\ & & & \times & \times & & \\ & & & \times & \times & \times & \\ & & & \times & \times & \times & \times \end{pmatrix}, \quad D^{\mathsf{T}} = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{pmatrix}$$

× is a non-zero entry, blanks are zero

$$\frac{d\tilde{L}_i(t)}{\tilde{L}_i(t)} = \mu_i\left(\tilde{L}(t)\right)dt + \sigma_i^{\mathsf{T}} dW(t), \quad 0 \le t \le T_i, \quad i = 1, \ldots, m$$

where W is a d-dimensional standard Brownian motion under a risk-adjusted measure and:

$$\mu_i\left(\tilde{L}(t)\right) = \sum_{j=\eta(t)}^{i} \frac{\sigma_i^{\mathsf{T}} \sigma_j \delta_j \tilde{L}_j(t)}{1 + \delta_j \tilde{L}_j(t)}$$

Although μ_i has an explicit dependence on t through $\eta(t)$, we suppress this argument. To keep this example as simple as possible, we take each σ_i (a d-vector of volatilities) to be a function of time to maturity:

$$\sigma_i(t) = \sigma_{i-\eta(t)+1}(0) \tag{6}$$

as in Glasserman & Zhao (1999). However, the same ideas apply if σ_i is itself a function of $\tilde{L}(t)$, as it often would be in trying to match a volatility skew.

To simulate, we apply an Euler scheme to the logarithms of the forward rates, rather than the forward rates themselves. This yields:

$$L_i(n+1) = L_i(n)\exp\left(\left[\mu_i\left(L(n)\right) - \|\sigma_i\|^2/2\right]h + \sigma_i^{\mathsf{T}} Z(n+1)\sqrt{h}\right), \quad i = \eta(nh), \ldots, m \tag{7}$$

Once a rate settles at its maturity it remains fixed, so we set $L_i(n + 1) = L_i(n)$ if $i < \eta(nh)$. The computational cost of implementing (7) is minimised by first evaluating the summations:

$$S_i(n) = \sum_{j=\eta(nh)}^{i} \frac{\sigma_j \delta_j L_j(n)}{1 + \delta_j L_j(n)}, \quad i = \eta(nh), \ldots, m \tag{8}$$

This then gives $\mu_i = \sigma_i^{\mathsf{T}} S_i$ and hence the total computational cost is $O(m)$ per time step.

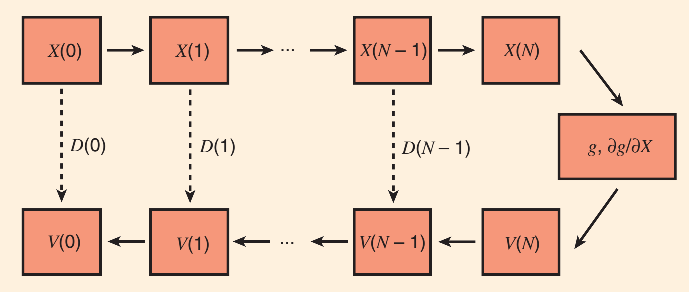
A simple example of a derivative in this context is a caplet for the interval $[T_m, T_{m+1})$ struck at K. It has a discounted payout:

$$\left(\prod_{i=0}^{m} \frac{1}{1 + \delta_i \tilde{L}_i(T_i)}\right) \delta_m \max\left\{0, \tilde{L}_m(T_m) - K\right\}$$

We can express this as a function of $\tilde{L}(T_m)$ (rather than $\tilde{L}(T_i)$, $i = 1, \ldots, m$) by freezing $\tilde{L}_i(t)$ at $\tilde{L}_i(T_i)$ for $t > T_i$. It is convenient to include the maturities T_i among the simulated dates of the Euler scheme, introducing unequal step sizes if necessary.

Glasserman & Zhao (1999) develop (and rigorously justify) the application of the pathwise method in this setting. Their application includes the evolution of the derivatives:

2. Data flow showing relationship between forward and adjoint calculations



$$\Delta_{ij}(n) = \frac{\partial L_i(n)}{\partial L_j(0)}, \quad i = 1,\ldots,m, \quad j = 1,\ldots,i$$

which can be found by differentiating (7). In the notation of (5), the matrix $D(n)$ has the structure shown in figure 1, with diagonal entries:

$$D_{ii}(n) = \begin{cases} 1 & i < \eta(nh) \\ \dfrac{L_i(n+1)}{L_i(n)} + \dfrac{L_i(n+1)\|\sigma_i\|^2 \delta_i h}{\left(1+\delta_i L_i(n)\right)^2} & i \geq \eta(nh) \end{cases}$$

and, for $j \neq i$:

$$D_{ij}(n) = \begin{cases} \dfrac{L_i(n+1)\sigma_i^{\mathsf{T}}\sigma_j \delta_j h}{\left(1+\delta_j L_j(n)\right)^2} & i > j \geq \eta(nh) \\ 0 & \text{otherwise} \end{cases}$$

The efficient implementation used in the numerical results of Glasserman & Zhao (1999) uses $\Delta_{ij}(n+1) = \Delta_{ij}(n)$ for $i < \eta(nh)$, while for $i \geq \eta(nh)$:

$$\Delta_{ij}(n+1) = \frac{L_i(n+1)}{L_i(n)}\Delta_{ij}(n) + L_i(n+1)\sigma_i^{\mathsf{T}} \sum_{k=\eta(nh)}^{i} \frac{\sigma_k \delta_k h \Delta_{kj}(n)}{\left(1+\delta_k L_k(n)\right)^2}$$

The summations on the right can be computed at a cost that is $O(m)$ for each j, and hence the total computational cost per time step is $O(m^2)$ rather than the $O(m^3)$ cost of implementing (5) in general.

Despite this, the number of forward rates m in the Libor market model can easily be 20–80, making the numerical evaluation of $\Delta_{ij}(n)$ rather costly. To get around this problem, Glasserman & Zhao (1999) proposed faster approximations to (5). The adjoint method in the next section can achieve computational savings without introducing any approximation beyond that already present in the Euler scheme.

Pathwise delta: adjoint method

Consider again the general setting of (1) and (2) and write $\partial g/\partial X(0)$ for the row vector of derivatives of $g(X(N))$ with respect to the elements of $X(0)$. With (4) and (5), we can write this as:

$$\begin{aligned} \frac{\partial g}{\partial X(0)} &= \frac{\partial g}{\partial X(N)}\Delta(N) \\ &= \frac{\partial g}{\partial X(N)} D(N-1)D(N-2)\cdots D(0)\Delta(0) \\ &= V(0)^{\mathsf{T}}\Delta(0) \end{aligned} \qquad (9)$$

where $V(0)$ can be calculated recursively using:

$$V(n) = D(n)^{\mathsf{T}} V(n+1), \quad V(N) = \left(\frac{\partial g}{\partial X(N)}\right)^{\mathsf{T}} \qquad (10)$$

The key point is that the adjoint relation (10) is a vector recursion whereas (5) is a matrix recursion. Thus, rather than update m^2 variables at each time step, it suffices to update the m entries of the adjoint variables $V(n)$. This can represent a substantial saving.

The adjoint method accomplishes this by fixing the payout g in the initialisation of $V(N)$, whereas the forward method allows calculation of pathwise deltas for multiple payouts once the $\Delta(n)$ matrices have been simulated. Thus, the adjoint method is beneficial if we are interested in calculating sensitivities of a single function g with respect to multiple changes in the initial condition $X(0)$ – for example, if we need sensitivities with respect to each $X_i(0)$. The function g need not be associated with an individual security; it could be the value of an entire portfolio.

Equation (10) is a discrete adjoint to equation (5) in the same way that the differential equation $-dv/dt = A^{\mathsf{T}}v$ is the adjoint counterpart to $du/dt = Au$, where u and v are in R^m and A is in $R^{m\times m}$. The terminology 'discrete adjoint' is used in computational engineering (Giles & Pierce, 2000), but in the computational finance context it might also be referred to as a 'backward' counterpart to the original forward pathwise sensitivity calculation since the adjoint recursion in (10) runs backward in time, starting at $V(N)$ and working recursively back to $V(0)$. To implement it, we need to store the vectors $X(0), \ldots, X(N)$ as we simulate forward in time so that we can evaluate the matrices $D(N-1), \ldots, D(0)$ as we work backward as illustrated in figure 2. This introduces some additional storage requirements, but these requirements are relatively minor because it suffices to store just the current path. The final calculation $V(0)^{\mathsf{T}}\Delta(0)$ produces exactly the same result as the forward calculations (4)–(5), but it does so with $O(Nm^2)$ operations rather than $O(Nm^3)$ operations.

To help fix ideas, we unravel the adjoint calculation in the setting of the Libor market model. After initialising $V(N)$ according to (10), we set

3. Structure of the matrix B and its transpose

$$B=\begin{pmatrix} & & & \\ & & & \\ & & & \\ \times & & & \\ \times & \times & & \\ \times & \times & \times & \\ \times & \times & \times & \times \end{pmatrix}, \quad B^{\mathsf{T}}=\begin{pmatrix} & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{pmatrix}$$

× is a non-zero entry, blanks are zero

$V_i(n) = V_i(n+1)$ for $i < \eta(nh)$ while for $i \geq \eta(nh)$:

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

The summations on the right can be computed at a cost that is $O(m)$, so the total cost per time step is $O(m)$, which is better than in the general case.

This is an example of a general feature of adjoint methods; whenever there is a particularly efficient way of implementing the original calculation there is also an efficient implementation of the adjoint calculation. This comes from a general result in the theory of algorithmic differentiation (Griewank, 2000), proving that the computational complexity of the adjoint calculation is no more than four times greater than the complexity of the original algorithm. There are various tools available for the automatic generation of efficient adjoint implementations, given an implementation of the original algorithm in C or C++ (see Automatic Differentiation website, *www.autodiff.org*).

In practical implementations, the payout evaluation may be performed by separate code or scripts to that used to perform the Monte Carlo path calculations. In such cases, it may be inconvenient or impractical to differentiate these to calculate $\partial g/\partial X(T)$ and a more pragmatic approach may be to use finite differences to approximate $\partial g/\partial X(T)$ and then use this to initialise the adjoint pathwise calculation.

Pathwise vegas

The previous section considers only the case of pathwise deltas, but similar ideas apply in calculating sensitivities to volatility parameters. The key distinction is that volatility parameters affect the evolution equation (3), and not just its initial conditions. Indeed, although we focus on vega, the same ideas apply to other parameters of the dynamics of the underlying process.

To keep the discussion generic, let θ denote a parameter of F_n in (3). For example, θ could parameterise an entire volatility surface or it could be the volatility of an individual rate at a specific date. The pathwise estimate of sensitivity to θ is:

$$\frac{\partial g}{\partial\theta}=\sum_{i=1}^{m}\frac{\partial g}{\partial X_i(N)}\frac{\partial X_i(N)}{\partial\theta}$$

If we write $\Theta(n)$ for the vector $\partial X(n)/\partial\theta$, we get:

$$\begin{aligned}\Theta(n+1)&=\frac{\partial F_n}{\partial X}\left(X(n),\theta\right)\Theta(n)+\frac{\partial F_n}{\partial\theta}\left(X(n),\theta\right)\\&=D(n)\Theta(n)+B(n)\end{aligned} \qquad (11)$$

with initial conditions $\Theta(0) = 0$. The sensitivity to θ can then be evaluated as:

$$\begin{aligned}\frac{\partial g}{\partial\theta}&=\frac{\partial g}{\partial X(N)}\Theta(N)\\&=\frac{\partial g}{\partial X(N)}\left\{B(N-1)+D(N-1)B(N-2)+\ldots+D(N-2)\ldots D(1)B(0)\right\}\\&=\sum_{n=0}^{N-1}V(n+1)^{\mathsf{T}}B(n)\end{aligned}$$

where $V(n)$ is the same vector of adjoint variables defined by (10).

In applying these ideas to the Libor market model, B becomes a matrix, with each column corresponding to a different element of the initial volatility vector $\sigma_j(0)$. The derivative of the ith element of $F_n(X_n)$ with respect to $\sigma_j(nh)$ is:

$$\frac{\partial(F_n)_i}{\partial\sigma_j(nh)}=\begin{cases}\dfrac{L_i(n+1)\sigma_i\delta_i L_i(n)h}{1+\delta_i L_i(n)} & \\ \quad+\left(S_i h-\sigma_i h+Z(n+1)\sqrt{h}\right)L_i(n+1) & i=j\geq\eta(nh)\\ \dfrac{L_i(n+1)\sigma_i\delta_j L_j(n)h}{1+\delta_j L_j(n)} & i>j\geq\eta(nh)\\ 0 & \textit{otherwise}\end{cases}$$

where S_i is as defined in (8). This has a similar structure to that of the matrix D in figure 1, except for the leading diagonal elements, which are now zero. However, the matrix B is the derivative of $F_n(X_n)$ with respect to the initial volatilities $\sigma_j(0)$, so given the definition (6), the entries in the matrix B are offset so that it has the structure shown in figure 3.

From (12), the column vector of vega sensitivities is equal to:

$$\left(\frac{\partial g}{\partial\sigma(0)}\right)^{\mathsf{T}}=\sum_{n=0}^{N-1}B(n)^{\mathsf{T}}V(n+1)$$

The ith element of the product $B(n)^{\mathsf{T}}V(n+1)$ is zero except for $1 \leq i \leq N-\eta(nh)+1$, for which it has the value:

$$\begin{aligned}&\left(S_{i^*}h-\sigma_{i^*}h+Z(n+1)\sqrt{h}\right)L_{i^*}(n+1)V_{i^*}(n+1)\\&+\frac{\delta_{i^*}L_{i^*}(n)h}{1+\delta_{i^*}L_{i^*}(n)}\sum_{j=i^*}^{m}L_j(n+1)V_j(n+1)\sigma_j\end{aligned}$$

where $i^* \equiv i+\eta(nh)-1$. The summations on the right for the different values of i^* are exactly the same summations performed in the efficient implementation of the adjoint calculation described in the previous section. Hence, the computational cost is $O(m)$ per time step.

Pathwise gamma

The second-order sensitivity of g to changes in $X(0)$ is:

$$\begin{aligned}\frac{\partial^2 g}{\partial X_j(0)\partial X_k(0)}&=\sum_{i=1}^{m}\frac{\partial g}{\partial X_i(N)}\Gamma_{ijk}(N)\\&+\sum_{i=1}^{m}\sum_{\ell=1}^{m}\frac{\partial^2 g}{\partial X_i(N)\partial X_\ell(N)}\Delta_{ij}(N)\Delta_{\ell k}(N)\end{aligned} \qquad (13)$$

where:

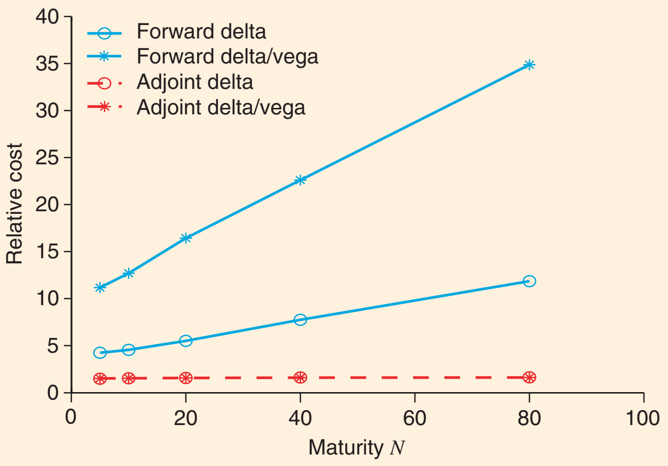
$$\Gamma_{ijk}(n)=\frac{\partial^2 X_i(n)}{\partial X_j(0)\partial X_k(0)}$$

Differentiating (3) twice yields:

$$\Gamma_{ijk}(n+1)=\sum_{\ell=1}^{m}D_{i\ell}(n)\Gamma_{\ell jk}(n)+\sum_{\ell=1}^{m}\sum_{m=1}^{m}E_{i\ell m}(n)\Delta_{\ell j}(n)\Delta_{mk}(n)$$

where $D_{i\ell}(n)$ is as defined previously, and:

4. Relative CPU cost of forward and adjoint delta and vega evaluation for a portfolio of 15 swaptions



$$E_{i\ell m}(n) = \frac{\partial^2 F_i(n)}{\partial X_\ell(n)\partial X_m(n)}$$

For a particular index pair (j, k), by defining:

$$G_i(n) = \Gamma_{ijk}(n), \quad C_i(n) = \sum_{\ell=1}^{m}\sum_{m=1}^{m} E_{i\ell m}(n)\Delta_{\ell j}(n)\Delta_{mk}(n)$$

this may be written as:

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

This is now in exactly the same form as the vega calculation, and so the same adjoint approach can be used. Option payouts ordinarily fail to be twice differentiable, so using (13) requires replacing the true payout g with a smoothed approximation; this is the subject of current research.

The computational operation count is $O(Nm^3)$ for the forward calculation of $L(n)$ and $\Delta(n)$ (and hence $D(n)$ and the vectors $C(n)$ for each index pair (j, k)) plus $O(Nm^2)$ for the backward calculation of the adjoint variables $V(n)$, followed by an $O(Nm^3)$ cost for evaluating the final sums in (12) for each (j, k). This is again a factor $O(m)$ less expensive than the alternative approach based on a forward calculation of $\Gamma_{ijk}(n)$.

Numerical results

Since the adjoint method produces exactly the same sensitivity values as the forward pathwise approach, the numerical results address the computational savings given by the adjoint approach applied to the Libor market model. The calculations are performed using one time step per Libor interval (that is, the time step h equals the spacing $\delta_i \equiv \delta$, which we take to be a quarter of a year). We take the initial forward curve to be flat at 5% and all volatilities equal to 20% in a single-factor ($d = 1$) model. Our test portfolio consists of options on one-year, two-year, five-year, seven-year and 10-year swaps with quarterly payments and swap rates of 4.5%, 5.0% and 5.5%, for a total of 15 swaptions. All swaptions expire in N periods, with N varying from one to 80.

Figure 4 plots the execution time for the forward and adjoint evaluation of both deltas and vegas, relative to the cost of simply valuing the swaption portfolio. The two curves marked with circles compare the forward and adjoint calculations of all deltas; the curves marked with stars compare the combined calculations of all deltas and vegas.

As expected, the relative cost of the forward method increases linearly with N, whereas the relative cost of the adjoint method is approximately constant. Moreover, adding the vega calculation to the delta calculation substantially increases the time required using the forward method, but this has virtually no impact on the adjoint method because the deltas and vegas use the same adjoint variables.

It is also interesting to note the actual magnitudes of the costs. For the forward method, the time required for each delta and vega evaluation is approximately 10% and 20%, respectively, of the time required to evaluate the portfolio. This makes the forward method 10–20 times more efficient than using central differences, indicating a clear superiority for forward pathwise evaluation compared with finite differences for applications in which one is interested in the sensitivities of a large number of different financial products. For the adjoint method, the observation is that one can obtain the sensitivity of one financial product (or a portfolio) to any number of input parameters for less than the cost of the original product evaluation.

The reason for the forward and adjoint methods having much lower computational cost than one might expect, relative to the original evaluation, is that in modern microprocessors, division and exponential function evaluation are 10–20 times more costly than multiplication and addition. By reusing quantities such as $L_i(n+1)/L_i(n)$ and $(1 + \delta_i L_i(n))^{-1}$, which have already been evaluated in the original calculation, the forward and adjoint methods can be implemented using only multiplication and addition, making their execution very rapid.

Conclusions

We have shown how an adjoint formulation can be used to accelerate the calculation of Greeks by Monte Carlo simulation using the pathwise method. The adjoint method produces exactly the same value on each simulated path as would be obtained using a forward implementation of the pathwise method, but it rearranges the calculations – working backward along each path – to generate potential computational savings.

The adjoint formulation outperforms a forward implementation in computing the sensitivity of a small number of outputs to a large number of inputs. This applies, for example, in a fixed-income setting, in which the output is the value of a derivatives book and the inputs are points along the forward curve. We have illustrated the use of the adjoint method in the setting of the Libor market model and found it to be fast – very fast. ■

Michael Giles is professor of scientific computing at the Oxford University Computing Laboratory. Paul Glasserman is the Jack R Anderson professor at Columbia Business School. **The research was supported in part by NSF grant DMS0410234.** **Email: giles@comlab.ox.ac.uk, pg20@columbia.edu,**

REFERENCES

Brace A, D Gatarek and M Musiela, 1997
The market model of interest rate dynamics
Mathematical Finance 7, pages 127–155

Giles M and N Pierce, 2000
An introduction to the adjoint approach to design
Flow, Turbulence and Control 65, pages 393–415

Glasserman P, 2004
Monte Carlo methods in financial engineering
Springer-Verlag, New York

Glasserman P and X Zhao, 1999
Fast Greeks by simulation in forward Libor models
Journal of Computational Finance 3, pages 5–39

Griewank A, 2000
Evaluating derivatives: principles and techniques of algorithmic differentiation
Society for Industrial and Applied Mathematics

Protter P, 1990
Stochastic integration and differential equations
Springer-Verlag, Berlin