Fast correlation Greeks by adjoint algorithmic differentiation

Adjoint methods have recently been proposed as an efficient way to calculate risk through Monte Carlo simulation. ***Luca Capriotti*** *and* ***Mike Giles*** *extend these ideas and show how adjoint algorithmic differentiation allows for fast calculation of price sensitivities in full generality. They illustrate the method for the calculation of correlation risk and test it numerically for portfolio default options*

One of the consequences of the recent crisis of the financial markets is a renewed emphasis on rigorous risk management practices. To quantify the financial exposure of financial firms, and to ensure efficient capital allocation and more effective hedging practices, regulators and senior management alike are insisting more and more on a thorough monitoring of risk. Among all businesses, those dealing with complex, over-the-counter derivatives are the ones receiving the most attention.

A thorough calculation of the risk exposure of portfolios of structured derivatives comes with a high operational cost because of the large amount of computer power required. Indeed, highly time-consuming Monte Carlo (MC) simulations are very often the only tool available for pricing and hedging complex securities. Calculating the Greeks, or price sensitivities, by 'bumping' – that is, by perturbing in turn the underlying model parameters, repeating the simulation and forming finite difference approximations – results in a computational burden increasing linearly with the number of sensitivities computed. This easily becomes very significant when the models employed depend on a large number of parameters, as is typically the case.

A particularly challenging task is the calculation of correlation risk, that is, the calculation of the sensitivities of a security with respect to some measure of the correlations among the random factors it depends on. Indeed, calculating risk with respect to all the independent pairwise correlations by bumping requires repeating the MC simulation many times, increasing quadratically with the number of random factors, and it is often unfeasible because of its high computational cost.

Several alternative methods for calculating sensitivities have been proposed in financial literature (for a review, see, for example, Glasserman, 2004). Among these, the pathwise derivatives method (Broadie & Glasserman, 1996) provides unbiased estimates at a computational cost that may be smaller than the one of bumping. However, in many problems the standard pathwise derivatives method provides limited computational gains, especially when the contract priced has a complex payout (Capriotti, 2010). A much more efficient implementation of the pathwise derivatives method was proposed by Giles & Glasserman (2006) in the context of the Libor market model for European-style payouts, and recently generalised to Bermudan options by Leclerc, Liang & Schneider (2009). These formulations express the calculation of the pathwise derivatives estimator in terms of linear algebra operations, and utilise adjoint methods to reduce the computational complexity by rearranging appropriately the order of the calculations.

Adjoint implementations can be seen as instances of a programming technique known as adjoint algorithmic differentiation (AAD) (Griewank, 2000, and Giles, 2007). In particular, as also discussed in a forthcoming paper (Capriotti & Giles, 2010), AAD can be used as a design paradigm to implement the pathwise derivatives method, or the calculation of the sensitivities of any numerical algorithm, in full generality. In this article, we illustrate these ideas by discussing a specific application: the calculation of correlation risk. We will begin by introducing the main ideas underlying algorithmic differentiation (AD), and the results for the computational efficiency of its two basic approaches: the forward and adjoint modes.

Forward and adjoint algorithmic differentiation

Both the forward and adjoint modes of AD aim at calculating the derivatives of a computer-implemented function. They differ by the direction of propagation of the chain rule through the composition of instructions representing the function. To illustrate this point, suppose we begin with a single input a, and produce a single output z after proceeding through a sequence of steps:

$$a \to \ldots \to u \to v \to \ldots \to z$$

The forward (or tangent) mode of AD defines $\dot{u}$ to be the sensitivity of u to changes in a, that is:

$$\dot{u} \equiv \frac{\partial u}{\partial a}$$

If the intermediate variables u and v are vectors, $\dot{v}$ is calculated by differentiating the dependence of v on u so that:

$$\dot{v}_i = \sum_j \frac{\partial v_i}{\partial u_j} \dot{u}_j$$

Applying this to each step in the calculation, working from left to right, we end up calculating $\dot{z}$, the sensitivity of the output to changes in the input. Note that if we have more than one input, we need to calculate the sensitivity to each one in turn, and so the cost is linear in the number of input variables.

Instead, the adjoint (or backward) mode of AD works from right to left. Using the standard AD notation, $\bar{u}$ is defined to be the sensitivity of the output z to changes in the intermediate vari-

able u, that is:

$$\bar{u}_i \equiv \frac{\partial z}{\partial u_i}$$

Using the chain rule, we get:

$$\frac{\partial z}{\partial u_i} = \sum_j \frac{\partial z}{\partial v_j}\frac{\partial v_j}{\partial u_i}$$

which corresponds to the adjoint mode equation:

$$\bar{u}_i = \sum_j \frac{\partial v_j}{\partial u_i}\bar{v}_j$$

Starting from $\bar{z} = 1$, we can apply this to each step in the calculation, working from right to left, until we obtain $\bar{a}$, the sensitivity of the output to each of the input variables.

In the adjoint mode, the cost does not increase with the number of inputs, but if there is more than one output, then the sensitivities for each output have to considered one at a time and so the cost is linear in the number of outputs. Furthermore, because the partial derivatives depend on the values of the intermediate variables, one first has to compute the original calculation storing the values of all the intermediate variables such as u and v, before performing the adjoint mode sensitivity calculation.

In the above description, each step can be a distinct high-level function, or specific mathematical operations, or even an individual instruction in a computer code. This last viewpoint is the one taken by computer scientists, who have developed tools that take as an input a computer code to perform some high-level function:

$$V = \text{FUNCTION}(U)$$

and produce new routines that will either perform the standard sensitivity analysis:

$$\dot{V} = \text{FUNCTION\_D}(U, \dot{U})$$

with suffix _D for 'dot', or its adjoint counterpart:

$$\bar{U} = \text{FUNCTION\_B}(U, \bar{V})$$

with suffix _B for 'bar'.[1]

One particularly important theoretical result is that the number of arithmetic operations in the adjoint routine FUNCTION_B is at most a factor of four greater than in FUNCTION (Griewank, 2000). As a result, it is possible to show that the execution time of FUNCTION_B is bounded by approximatively four times the cost of execution of the original function FUNCTION. Thus, one can obtain the sensitivity of a single output to an unlimited number of inputs for little more work than the original computation.

While the application of such automatic AD tools to large inhomogeneous pricing software is challenging, the principles of AD can be used as a programming paradigm that can be used to design the forward or adjoint of any algorithm (possibly using automatic AD tools for the implementation of smaller, simpler components). This is especially useful for the most common situations where pricing codes use a variety of libraries written in different languages, possibly linked dynamically. These ideas will be discussed at length in Capriotti & Giles (2010).

AAD and the pathwise derivatives method for correlation risk

In this article, we consider options pricing problems that can be expressed as an expectation value of the form:

$$V = \mathbb{E}_{\mathbb{Q}}\left[P(X)\right] \tag{1}$$

where $X = (X_1, \ldots, X_N)^t$ represents the state vector of N market factors (for example, stock prices, interest rates, foreign exchange pairs, default times, etc), $P(X)$ is the (possibly discounted) payout function of a security contingent on their future realisation, and $\mathbb{Q} = \mathbb{Q}(X)$ represents a risk-neutral probability distribution (Harrison & Kreps, 1979) according to which the components of X are distributed. Although the proposed method easily generalises to other kinds of joint distributions, here we consider a N-dimensional Gaussian copula as a model for the co-dependence between the components of the state vector, namely a joint cumulative density function of the form:

$$\mathbb{Q}(X) = \Phi_N\left(\Phi^{-1}\left(M_1(X_1)\right), \ldots, \Phi^{-1}\left(M_N(X_N)\right); \rho\right) \tag{2}$$

where $\Phi_N(Z_1, \ldots, Z_N; \rho)$ is a N-dimensional multivariate Gaussian distribution with zero mean, and an $N \times N$ positive semidefinite correlation matrix ρ, Φ^{-1} is the inverse of the standard normal cumulative distribution, and $M_i(X_i)$, $i = 1, \ldots, N$ are the marginal distributions of the underlying factors, typically implied from the market prices of liquid securities.

The expectation value in (1) can be estimated by MC by sampling a number N_{MC} of random replicas of the underlying state vector $X[1]$, $\ldots, X[N_{MC}]$, according to the distribution $\mathbb{Q}(X)$, and evaluating the payout $P(X)$ for each of them. This leads to the central limit theorem (Kallenberg, 1997) estimate of the option value V as:

$$V \simeq \frac{1}{N_{MC}} \sum_{i_{MC}=1}^{N_{MC}} P\left(X\left[i_{MC}\right]\right) \tag{3}$$

with standard error $\Sigma/\sqrt{N_{MC}}$, where $\Sigma^2 = E_{\mathbb{Q}}[P(X)^2] - E_{\mathbb{Q}}[P(X)]^2$ is the variance of the sampled payout.

In the Gaussian model above, the dependence between the underlying factors is determined by the correlation of a set of jointly normal random variables $Z = (Z_1, \ldots, Z_N)^t$ distributed according to $\Phi_N(Z_1, \ldots, Z_N; \rho)$. Each Z_i is distributed according to a standard normal distribution so that $\Phi(Z_i)$ is a uniform random variable in (0, 1) and $X_i = M_i^{-1}(\Phi(Z_i))$ is distributed according to M_i. The sampling of the N jointly normal random variables $(Z_1, \ldots, Z_N)$ is efficiently implemented by means of a Cholesky factorisation of the correlation matrix. The Cholesky factorisation produces a lower triangular $N \times N$ matrix C such that $\rho = CC^T$ so that one can write $Z = C\tilde{Z}$ where $\tilde{Z} = (\tilde{Z}_1, \ldots, \tilde{Z}_N)^t$ is a N dimensional vector of independent standard normal random variables. These observations naturally translate into the standard algorithm to generate MC samples of X according to (2), namely:

- **Step 0.** Generate a sample of N independent standard normal variates, $\tilde{Z} = (\tilde{Z}_1, \ldots, \tilde{Z}_N)^t$.
- **Step 1.** Correlate the components of $\tilde{Z}$ by performing the matrix vector product $Z = C\tilde{Z}$.
- **Step 2.** Set $U_i = \Phi(Z_i)$, $i = 1, \ldots, N$.
- **Step 3.** Set $X_i = M^{-1}(U_i)$, $i = 1, \ldots, N$.
- **Step 4.** Calculate the payout estimator $P(X_1, \ldots, X_N)$.

Correlation risk can be obtained in a highly efficient way by implementing the pathwise derivatives method according to the principles of AAD (Capriotti, 2010, and Capriotti & Giles, 2010). It is convenient to first express the expectation value as being over $\mathbb{P}(\tilde{Z})$, the distribution of independent $\tilde{Z}$ used in the MC simulation, so that:

$$V = \mathbb{E}_{\mathbb{Q}}\left[P(X)\right] = \mathbb{E}_{\mathbb{P}}\left[P\left(X\left(\tilde{Z}\right)\right)\right] \tag{4}$$

[1] *To learn more about automatic differentiation tools see, for example,* www.autodiff.org

The point of this subtle change is that $\mathbb{P}(\tilde{Z})$ does not depend on the correlation matrix ρ, whereas $\mathbb{Q}(X)$ does.

The pathwise derivatives method allows the calculation of the sensitivities of the option price V (4) with respect to a set of N_θ parameters $\theta = (\theta_1, \dots, \theta_{N_\theta})$, say:

$$\frac{\partial V(\theta)}{\partial \theta_k} = \frac{\partial}{\partial \theta_k} \mathbb{E}_{\mathbb{P}}\left[P(X)\right] \tag{5}$$

by defining appropriate estimators, say $\bar{\theta}_k(X[i_{MC}])$, that can be sampled simultaneously in a single MC simulation. This can be achieved by observing that whenever the payout function is regular enough (for example, Lipschitz-continuous, see Glasserman, 2004), and the distribution $\mathbb{P}(\tilde{Z})$ does not depend on θ, one can rewrite equation (5) by taking the derivative inside the expectation value, as:

$$\frac{\partial V(\theta)}{\partial \theta_k} = \mathbb{E}_{\mathbb{P}}\left[\frac{\partial P(X)}{\partial \theta_k}\right] \tag{6}$$

The calculation of equation (6) can be performed by applying the chain rule, and calculating the average value of the so-called pathwise derivatives estimator:

$$\frac{\partial P(X)}{\partial \theta_k} = \sum_{i=1}^{N} \frac{\partial P(X)}{\partial X_i} \times \frac{\partial X_i}{\partial \theta_k} \tag{7}$$

The standard pathwise implementation corresponds to a forward mode sensitivity analysis. Applied to steps 1–4 (since the normal variates $\tilde{Z}$ do not depend on any input parameters), this gives for each sensitivity:

- **Step 1f.** Calculate $\dot{Z} = \dot{C}\,\tilde{Z}$ where $\dot{C}$ is the sensitivity of C with respect to a given entry of the correlation matrix.
- **Step 2f.** Set $\dot{U}_i = \phi(Z_i)\dot{Z}_i$, $i = 1, \dots, N$.
- **Step 3f.** Set $\dot{X}_i = \dot{U}_i / m_i(X_i)$, $i = 1, \dots, N$.
- **Step 4f.** Calculate:

$$\dot{P} = \sum_{i=1}^{N} \frac{\partial P}{\partial X_i} \dot{X}_i$$

Here $\phi(x) \equiv \partial\Phi(x)/\partial x$ is the standard normal probability density function and $m_i(x) \equiv \partial M_i(x)/\partial x$ is the probability density function associated with the marginal $M_i(x)$ of the ith random factor.

As anticipated, the computational cost of the forward pathwise derivatives method scales linearly with the number of sensitivities computed N_θ, that is, the same scaling of finite difference approximations of the derivatives $\partial_{\theta_k} E_{\mathbb{Q}}[P(X)]$. As a result, in many situations, typically involving complex payouts, the standard implementation of the pathwise derivatives method offers a limited computational advantage with respect to bumping (Capriotti, 2010).

In contrast, AAD allows in general a much more efficient implementation of the pathwise derivatives estimators (7). Indeed, as an immediate consequence of the computational complexity results introduced in the previous section, it can be shown (Capriotti & Giles, 2010) that AAD allows the simultaneous calculation of the pathwise derivatives estimators for any number of sensitivities at a computational cost that is a small multiple (of order four) of the cost of evaluating the original payout estimator. As a result, one can calculate the MC expectation of an arbitrarily large number of sensitivities at a small fixed cost.

Although AAD can be applied for virtually any model and payout function of interest in computational finance – including path-dependent and Bermudan options – here we will concentrate on the calculation of correlation sensitivities in a Gaussian copula framework. In general, for the reasons mentioned in the previous section, the AAD implementation of the pathwise derivatives method contains a forward sweep – reproducing the steps followed in the calculation of the estimator of the option value $P(X)$ – and a backward sweep. As a result, the adjoint algorithm consists of adjoint counterparts for each of the steps 1–4 above executed in reverse order, plus the adjoint of the Cholesky factorisation.

The first step consists of the evaluation of the adjoint of step 4 of the forward sweep, calculating the derivatives of the payout with respect to the components of the state vector:

$$\bar{X}_k = \frac{\partial P(X)}{\partial X_k} \tag{8}$$

with $k = 1, \dots, N$. These derivatives can be calculated efficiently using AAD, as discussed in Capriotti (2010).

In turn, the adjoint of step 3 of the forward sweep is given by:

$$\bar{U}_k = \bar{M}_k^{-1}\left(U_k, \bar{X}_k\right) = \frac{\bar{X}_k}{m_k\left(X_k\right)} \tag{9}$$

for $k = 1, \dots, N$. The vector $\bar{U}$ is then mapped into the adjoint of the correlated standard normal variables $\bar{Z}$ through the counterpart of step 2:

$$\bar{Z}_k = \bar{\Phi}\left(Z_k, \bar{U}_k\right) = \bar{U}_k \phi\left(Z_k\right) \tag{10}$$

The adjoint of step 1 performing the matrix vector product $Z = C\tilde{Z}$ reads:

$$\bar{C}_{i,j} = \sum_{k=1}^{N} \frac{\partial Z_k}{\partial C_{i,j}} \bar{Z}_k = \tilde{Z}_j \bar{Z}_i \tag{11}$$

or $\bar{C} = \bar{Z}\tilde{Z}^t$. By applying the chain rule, it is straightforward to realise that the adjoint $\bar{w}$ of each intermediate variable w in the succession of steps 0–4 represents the derivative of the payout estimator with respect to w, or $\bar{w} = \partial P/\partial w$. In particular, the quantities $\bar{C}_{i,j}$ calculated at the end of the adjoint of step 1 represent the derivatives of the payout estimator with respect to the entries of the triangular Cholesky matrix C, namely the pathwise estimator (7) with $\theta_k = C_{i,j}$.

In summary, the AAD implementation of the pathwise derivatives estimator consists of steps 1–4 described above (forward sweep) plus the following steps of the backward sweep:

- **Step 5.** Evaluate the payout adjoint $\bar{X}_k = \partial P/\partial X_k$, for $k = 1, \dots, N$.
- **Step 6.** Calculate $\bar{U}_k = \bar{X}_k / m_k(M_k^{-1}(U_k))$, $k = 1, \dots, N$.
- **Step 7.** Calculate $\bar{Z}_k = \bar{U}_k \phi(Z_k)$, $k = 1, \dots, N$.
- **Step 8.** Calculate $\bar{C} = \bar{Z}\tilde{Z}^t$.

At this point in the calculation, there is an interesting complication. The natural AAD approach would average the values of $\bar{C}$ from each of the MC paths. This average $\bar{C}$ can be converted into derivatives with respect to the entries of the correlation matrix by means of the adjoint of the Cholesky factorisation (Smith, 1995), namely a function of the form:

$$\bar{\rho} = \text{CHOLESKY\_B}\left(\rho, \bar{C}\right) \tag{12}$$

providing:

$$\bar{\rho}_{i,j} = \sum_{l,m=1}^{N} \frac{\partial C_{l,m}}{\partial \rho_{i,j}} \bar{C}_{l,m} \tag{13}$$

The pseudo-code for the adjoint Cholesky factorisation is given in figure 1. By inspecting the structure of the pseudo-code it appears clear that its computational cost is just a small multiple (of order two) of the cost of evaluating the original factorisation. Indeed, the adjoint algorithm essentially contains the original Cholesky

1 Adjoint of the Cholesky factorisation

```
Cholesky_b(/*In*/ rho,/*In*/ L_b,/*Out*/ rho_b)
{
      // n = Dimension of Correlation Matrix rho

      // Forward Sweep
      for (unsigned long i = 0; i < n; i++) {
            for (unsigned long j = i; j < n; j++){
                  sum(i, j) = rho(i, j);
                  for (long k = i - 1; k >= 0; k--)
                        sum(i, j) -= L(i, k) * L(j, k);
                        if (i == j){
                              L(i, i) = sqrt(sum(i, j));
                        } else {
                              L(j, i) = sum(i, j) / L(i, i);
                        }
                  }
            }
      }
      // Backward Sweep
      for (long i = n - 1; i >= 0; i--){
            for (long j = n-1; j >= i; j--) {
                  double sum_b =  0.0;
                  if (i == j) {
                        if (sum(i, j) == 0.0) {
                                    sum_b = 0.0;
                        } else {
                              sum_b = L_b(i, j)/( 2.0 * L(i, j));
                        }
                        L_b(i, j) = 0.0;
                  } else {
                        sum_b = L_b(j, i)/L(i, i);
                        L_b(i, i) -= sum(i, j) * sum_b / L(i, i);
                        L_b(j, i) = 0.0;
                  }
                  for (long k = i - 1; k >= 0; k--) {
                        L_b(i, k) -= L(j, k)*sum_b;
                        L_b(j ,k) -= L(i, k)*sum_b;
                  }
                  rho_b(i, j) += sum_b;}
            }
      }
}
```

Note: the forward sweep is an exact replica of the original factorisation

factorisation plus a backward sweep with the same complexity and a similar number of operations.

The complication with this implementation is that it gives an estimate for the correlation risk, but it does not provide a corresponding confidence interval. An alternative approach would be to convert $\bar{C}$ to $\bar{\rho}$ for each individual path, and then calculate the average and standard deviation of $\bar{\rho}$ in the usual way. However, the numerical results will show that this is rather costly. An excellent compromise between these two extremes is to divide the N_{MC} paths into N_b 'bins' of equal size. For each bin, an average value of $\bar{C}$ is calculated and converted into a corresponding value for $\bar{\rho}$. These N_b estimates for $\bar{\rho}$ can then be combined in the usual way to form an overall estimate and confidence interval for the correlation risk.

The computational benefits can be understood by considering the computational costs for both the standard evaluation and the adjoint pathwise derivatives calculation. In the standard evaluation, the cost of the Cholesky factorisation is $O(N^3)$ and the cost of the MC sampling is $O(N_{MC}N^2)$, so the total cost is $O(N^3 + N_{MC}N^2)$. Since N_{MC} is always much greater than N, the cost of the Cholesky factorisation is usually negligible. The cost of the adjoint steps in the MC sampling is also $O(N_{MC}N^2)$, and when using N_b bins the cost of the adjoint Cholesky factorisation is $O(N_bN^3)$. To obtain an accurate confidence interval, but with the cost of the Cholesky factorisation being negligible, requires that N_b is chosen so that $1 \ll N_b \ll N_{MC}/N$. Without binning, that is, using $N_b = N_{MC}$, the cost to calculate the average of the estimators (13) is $O(N_{MC}N^3)$, and so the relative cost compared with the evaluation of the option value is $O(N)$.

The binning procedure described above can be generalised to any situation in which the standard procedure involves a common preprocessing step before any of the path calculations are performed. Other examples would include calibration of model parameters to market prices, or a cubic spline construction of a local volatility surface. In each case, there is a linear relationship between the forward mode sensitivities before and after the preprocessing step, and therefore a linear relationship between the corresponding adjoint sensitivities.

The algorithm described above can be applied whenever the option pricing problem can be formulated as an expectation value over a set of random factors whose distribution is modelled as a Gaussian copula. This includes in general a variety of basket options common across all asset classes, or structured swaps whose coupon depends on a specific observation of a set of correlated rates. In addition, the same ideas can be extended to the simulation of correlated diffusion processes (Capriotti & Giles, 2010).

Numerical tests

As a numerical test ground, we consider the case of basket default options (Chen & Glasserman, 2008). In this context, the random factors X_i represent the default time τ_i of the ith name, for example, the time a specific company in a reference pool of N names fails to pay one of its liabilities as specified by the terms of the contract priced. In particular, in an nth-to-default basket default swap one party (the protection buyer) makes regular payments to a counterparty (the protection seller) at time $T_1, \ldots, T_M \le T$ provided that fewer than n defaults events among the components of the basket are observed before time T_M. On the other hand, if n defaults occur before time T, the regular payments cease and the protection seller makes a payment to the buyer of $(1 - R_i)$ per unit notional, where R_i is the normalised recovery rate of the ith asset. The value at time zero of the basket default swap on a given realisation of the default times $\tau_1, \ldots, \tau_N$ can therefore be expressed as:

$$P(\tau_1,\ldots,\tau_N) = P_{prot}(\tau_1,\ldots,\tau_N) - P_{prem}(\tau_1,\ldots,\tau_N) \qquad (14)$$

that is, as the difference between the so-called protection and premium legs. The value of the protection leg is given by:

$$P_{prot}(\tau_1,\ldots,\tau_N) = (1 - R_n)D(\tau)\mathbb{I}(\tau \le T) \qquad (15)$$

where R_n and τ are the recovery rate and default time of the nth to default, respectively, $D(t)$ is the discount factor for the interval $[0, t]$ (here we assume for simplicity uncorrelated default times and interest rates), and $\mathbb{I}(\tau \le T)$ is the indicator function of the event that the nth default occurs before T. The premium leg reads instead, neglecting for simplicity any accrued payment:

$$P_{prem}(\tau_1,\ldots,\tau_N) = \sum_{k=1}^{L(\tau)} s_k D(T_k) \qquad (16)$$

where $L(\tau) = \max[k \in \{1, \ldots, M\}/T_k < \tau]$, and s_k is the premium payment (per unit notional) at time T_k.

To apply the pathwise derivatives method to the payout above, the indicator functions in (16) and (15) need to be regularised (Glasserman, 2004, and Chen & Glasserman, 2008). One simple and practical way of doing that is to replace the indicator functions with their smoothed counterpart, at the price of introducing a small amount of bias in the Greek estimators. For the problem at hand, as is also generally the case, such bias can be easily reduced to be smaller than the statistical errors that can be obtained for any realistic number of MC iterations N_{MC} (for a more complete discussion of the topic of payout regularisation,

see Capriotti, 2010, Capriotti & Giles, 2010, and Giles, 2009).

The remarkable computational efficiency of AAD is illustrated in figure 2 for the second-to-default swap. Here, we plot the ratio of the computing time required to calculate the value of the option, and all its pairwise correlation sensitivities, and the computing time spent calculating the value alone, as functions of the number of names in the basket. As expected, for standard finite-difference estimators, such a ratio increases quadratically with the number of names in the basket. For medium-sized baskets ($N \equiv 20$), the cost associated with bumping is more than 100 times more expensive than that of AAD.

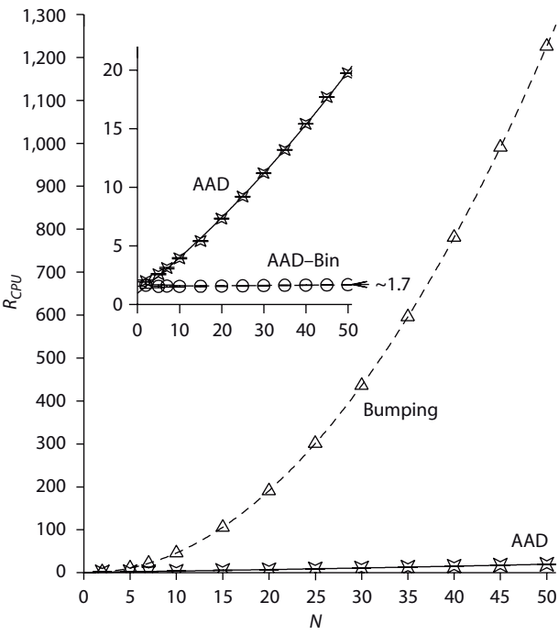
Nevertheless, taking a closer look (see the inset of figure 2), the relative cost of AAD without binning is $O(N)$, for the reasons explained earlier. However, when using $N_b = 20$ bins the cost of the adjoint Cholesky computation is negligible and the numerical results show that all the correlation Greeks can be obtained with a mere 70% overhead compared with the calculation of the value of the option. This results in more than two orders of magnitude savings in computational time for a basket of more than 40 names.

Conclusions

We have shown how adjoint algorithmic differentiation allows an extremely efficient calculation of correlation risk using Monte Carlo. The proposed method relies on using the adjoint mode of algorithmic differentiation to organise the calculation of the pathwise derivatives estimator, and to implement the adjoint counterpart of the Cholesky factorisation. For any number of underlying assets or names in a portfolio, the proposed method allows the calculation of the complete pairwise correlation risk at a computational cost that is at most four times the cost of calculating the option value itself, resulting in remarkable computational savings with respect to bumping. We illustrated the method for a Gaussian copula-based Monte Carlo computation, and we tested it numerically for portfolio default options. In this application, the proposed method is 100 times faster than bumping for 20 names, and 500 times for 40 names. The method generalises immediately to other kinds of elliptic copulas, and to a general diffusive setting. In fact, it is a specific instance of a general AAD approach to the implementation of the pathwise derivatives method that will be discussed in a forthcoming publication (Capriotti & Giles, 2010). ■

Luca Capriotti is vice-president in the Quantitative Strategies group at Credit Suisse. Mike Giles is professor of scientific computing in the University of Oxford Mathematical Institute and a member of the Oxford-Man Institute of Quantitative Finance. They would like to thank Alex Prideaux, Adam Peacock, Matthew Peacock, Jacky Lee and David Shorthouse for useful discussions, and valuable help provided by Mark Bowles and Anca Vacarescu in the initial stages of this project is also gratefully acknowledged. The opinions and views expressed in this paper are those of the authors, and do not necessarily represent those of Credit Suisse Group. Email: luca.capriotti@credit-suisse.com, mike.giles@maths.ox.ac.uk

2 Computation speed



Note: ratios of the CPU time required for the calculation of the option value, and correlation Greeks, and the CPU time spent for the computation of the value alone, as functions of the number of names in the basket, for $N_{MC} = 10^5$. Symbols: bumping (one-sided finite differences) (triangles), AAD without binning (that is, $N_b = N_{MC}$) (stars), AAD with binning ($N_b = 20$) (empty circles). Lines are guides for the eye, and the MC uncertainties are smaller than the symbol sizes

References

Broadie M and P Glasserman, 1996
Estimating security price derivatives using simulation
Management Science 42(2), pages 269–285

Capriotti L, 2010
Fast Greeks by algorithmic differentiation
Forthcoming in Journal of Computational Finance

Capriotti L and M Giles, 2010
Algorithmic differentiation: adjoint Greeks made simple
Working paper

Chen Z and P Glasserman, 2008
Sensitivity estimates for portfolio credit derivatives using Monte Carlo
Finance and Stochastics 12(4), pages 507–540

Giles M, 2007
Monte Carlo evaluation of sensitivities in computational finance
Proceedings of the Eighth HERCMA Conference

Giles M, 2009
Vibrato Monte Carlo sensitivities
In Monte Carlo and Quasi-Monte Carlo Methods 2008, edited by P L'Ecuyer and A Owen, Springer, New York

Giles M and P Glasserman, 2006
Smoking adjoints: fast Monte Carlo Greeks
Risk January, pages 92–96

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

Griewank A, 2000
Evaluating derivatives: principles and techniques of algorithmic differentiation
SIAM, Philadelphia

Harrison J and D Kreps, 1979
Martingales and arbitrage in multiperiod securities markets
Journal of Economic Theory 20(3), pages 381–408

Kallenberg O, 1997
Foundations of modern probability
Springer, New York

Leclerc M, Q Liang and I Schneider, 2009
Fast Monte Carlo Bermudan Greeks
Risk July, pages 84–88

Smith S, 1995
Differentiation of the Cholesky algorithm
Journal of Computational and Graphical Statistics 4(2), pages 134–147