Fast and Robust Monte Carlo CDO Sensitivities
and their
Efficient Object Oriented Implementation

Marius G. Rott* Christian P. Fries*
mariusgrott@aol.com email@christian-fries.de

May 31, 2005

(Version 0.9.2)

Abstract

In this paper we present a simple yet generic method for fast and robust Monte-Carlo calculation of sensitivities of *Collateralized Debt Obligations* (CDOs). The method is product independent and only relies on four pricings against modified models. From a modeling perspective the method is also fairly general as it only relies on the availability of a conditional cumulative distribution function for the default time. In our presentation we concentrate on conditional independent loss models as given in [12].

The method we propose in this paper is generic and allows for an equally generic object oriented implementation which is highly efficient with respect to calculation performance and coding time (time to market). We present the design pattern of a stochastic iterator, the *default time iterator*, to create a highly flexible product implementation framework in which any product may become the underlying of any other product. Our benchmark calculations indicate that our method improves calculation time by a factor of around 1000 compared to brute force finite differences. The coding of a new product still remains on a "plug-and-play" level with very short development time.

*Marius Rott and Christian Fries are employees of DZ Bank AG. The views, thoughts and opinions expressed in this paper are those of the authors in their individual capacity and should not in any way be attributed to DZ Bank AG or to the authors as representatives, officer or employee of DZ Bank AG.

Contents

1 Introduction

Due to the high dimensionality of many of the structures, and also the inherent path dependent features of some structures, Monte Carlo [3, 11] has been the method of choice for portfolio credit structures. Additionally, Monte Carlo methods are fast and easy to implement and usually result in pretty generic code without much additional implementation need, when new structures are introduced. The resulting valuation engine is easy to maintain and extend. However, there is a price to pay. Most notably Monte Carlo methods are notoriously slow for the calculation of risk sensitivities, especially for products with discontinuous payoffs. Credit products are by default discontinuous in the sense that usually default just before and just after maturity result in very different payments. This feature is mainly responsible for the inefficiency and instability of Monte Carlo sensitivities for credit structures. Although, they do not really tackle the root of the problem, some improvements can be achieved by accelerating the calculation via variance reduction techniques. There are also general methods that tackle the sensitivity calculation problem directly with some success (e.g. Malliavin Calculus [5] or Likelihood Ratio Method [4]). Nonetheless, recently a number of authors have chosen a different numerical approach. Within a factor reduced model a quasi-analytical solution based on integration over a discretely approximated portfolio loss distribution allows a stable and efficient calculation of prices and risk sensitivities of some products. Unfortunately, these methods are not available for all products and for those available the methods can only be used under some additional simplifying assumptions. Especially path dependent products can not easily be integrated in that framework and then Monte Carlo methods are used in practice. They probably also remain important because there is a tendency in the market to integrate non credit risk factors in new innovative products in an attempt to widen the margin in an environment of decreasing margins for standard products. Integration of new risk factors is probably most easily achieved in a Monte Carlo setup.

In this paper we show how to efficiently calculate CDS par spread deltas for CDOs within a factor reduced model (see [1] for further details). Dependency is introduced through a set of underlying factors affecting the underlying credits through latent default indicator variable. The dependency of each credit on the common factors determines the overall correlation. Usually only a small number of factors is necessary to introduce an acceptable correlation structure. An important feature of the model is that conditional on the realization of the common factors the default indicators, and thus defaults, are mutually independent. As a result the *conditional cumulative default distribution*, that determines the probability of default given the realization of the common factors, is often known analytically. We show how analytical knowledge of it can be used to stabilize Monte Carlo deltas after splitting the calculation into two separate parts. The two parts measure the effect of a shift in default time conditional on default happening before maturity of the product on one hand, and the effect of a shift in default time crossing the maturity of the product on the other. The latter is responsible for the instability of the standard sensitivities calculation methods. Our tests indicate that the speed up achievable over a brute force method can be well above a factor of 1000. This allows to calculate the delta sensitivities in near or even real time. Additional to its methodological efficiency the method presented remains generic in the sense that it does not dependent on any specific product features.

We view the implementation as the ultimate test for the practical relevance of a method. We are mainly interested in two aspects: *development time* and *calculation time*. Often the development time may be reduced by a generic (most likely object oriented) framework, which sometime comes at the expense of an increased calculation time. Whether a method satisfies both objectives, fast development time and fast calculation time, often depends on the generality of the mathematical framework. Ideally it is product or even model independent. For example, calculating sensitivities by *bumping the model* (brute force finite differences) is model and product independent. Its development time is almost zero. However for Monte-Carlo implementations its performance is often unsatisfactory. Calculating sensitivities using

Malliavin Calculus often greatly improves the performance of Monte-Carlo sensitivity calculation, however, it is model dependent and to some extend even product dependent. Hence its implementation is less generic and thus its development time is higher.[1] We present an object oriented design that is adapted to the proposed methodology and realizes a product and almost model independent delta implementation. The design is centered around a general *default time iterator* and combines all the nice features we are interest in from a practical point of view.

1.1 Layout of the Paper

The paper is organized as follows: In Section 2 we will present the CDO Conditional Independent Loss Model and fix notation. Section 3 gives the basics of CDO Monte Carlo evaluation within this model.

Section 4 will present various methods for calculating Monte Carlo sensitivities of CDOs like Brute Force, Likelihood Ratio and the method proposed here, using the conditional cumulative default distribution.

Section 5 gives a brief discussion of the object oriented implementation of the method proposed showing that it is not only robust but also fast in terms of calculation time *and* development time.

Section 6 concludes the paper with a presentation of numerical results.

[1] For an approach that combines the generality of *bumping the model* with the robustness of Malliavin Calculus in a different model context see [6].

2 CDO Conditional Independent Loss Model

In a *Factor reduced Conditional Independent Loss Model* the correlation between the m individual names of the underlying credit pool is introduced by k-factor model for a latent default indicator variable Z_i.

$$Z_i = c_i^T \cdot x + \sqrt{1 - c_i^T \cdot c_i}\epsilon_i \quad i = 1, \ldots, m, \tag{1}$$

where $c_i \in \mathbb{R}^k$ is the vector of factor loadings[2], $x^T = (x_1, \ldots, x_k)$ the vector of systematic factors and ϵ_i the idiosyncratic factor. We start with a model where $x_j, \epsilon_i \sim \mathcal{N}(0,1)$ are independent standard normal distributed, though this distributional assumption may be modified.[3] Each Z_i is also standard normally distributed and, conditional on x, all Z_i are independent (hence the name conditional independent loss model). Given the realization of the default indicator Z_i of the underlying Credit Default Swap (CDS) i the default time τ_i is given implicitly as solution of

$$\Phi(Z_i) = 1 - e^{-\int_0^{\tau_i} \lambda_i(t)\mathrm{d}t}. \tag{2}$$

Here Φ denotes the cumulative standard normal distribution and λ_i is the term structure of risk neutral forward default intensities of credit i. Following the above rule the probability $P(\tau_i \leq T)$ of default of credit i before time T is given by

$$\begin{aligned} P(\tau_i \leq T) &= P(1 - e^{-\int_0^{\tau_i} \lambda_i(t)dt} \leq 1 - e^{-\int_0^{T} \lambda_i(t)dt}) \\ &= P(\Phi(Z_i) \leq 1 - e^{-\int_0^{T} \lambda_i(t)dt}) \\ &= 1 - e^{-\int_0^{T} \lambda_i(t)dt} \end{aligned}$$

Thus individual names default according to a Poisson process with a deterministic term tsructure of default intensities λ_i. In this paper we assume that the underlying products are CDS with liquid market quotes . Assuming deterministic recovery rates one may bootstrap market implied default intensities λ_i from the available market market quotes. Denoting the term structure of CDS par spreads of credit i by s_i, the term structure of default intensities of credit i is thus a function $\lambda_i(s_i)$.

2.1 Coordinate System of the Default State Space

The above model gives the default times τ_i as functionals of underlying random variables. Theses random variables may be viewed as equivalent characterization of the default state in a different coordinate system and in the following we make have use of this observation. We will use the following notation:

- Let $v_1, \ldots, v_m, w_1, \ldots, w_k$ denote i.i.d. uniform random variables on $[0,1]$.
- Let $\epsilon_1, \ldots, \epsilon_m, x_1, \ldots, x_k$ denote the normal random variables given by

$$\epsilon_i := \Phi^{-1}(v_i), \quad x_j := \Phi^{-1}(w_i).$$

- Let Z_i denote normal random variables given by (1).
- Let τ_i denote Poisson random variables implicitly given by (2).

Thus the default times are a functional of uniform random variables $(v, w) \in [0,1]^{m \times k}$. We will make frequent use of the (v, w) space for the efficient calculation of CDS par spread deltas in Section 4.

[2] For an algorithm to determine the factor loadings for a given correlation structure see [1].
[3] See Section 4.5.1 for an example using the t distribution.

Remark

The distributional assumption of the model may be altered by a change of the functional Φ.[4] The approach thus is a special case of *functional modelling*.

[4] Section 4.5.1 shows how to use a t distribution.

3 Product Valuation

Assuming that loss given default is non-random the value of a credit product that depends on losses of the underlying credit pool can be written as

$$V = \int_{\mathbb{R}^m} f(\tau_1, \dots \tau_m) p(\tau_1, \dots \tau_m) d(\tau_1, \dots \tau_m)$$

where f denotes the discounted value of any cash flow of the product defined as a function of the default times τ. The function p denotes the density (simply assuming that it exists) of the default times in our model. Defining the uniformly distributed random variable

$$v_i := \Phi(\epsilon_i) \tag{3}$$

and observing that the default time may be written as a function of default intensities and the realization of the default indicator one can also write the above integral as

$$V = \int_{\mathbb{R}^k} \int_{[0,1]^m} f(x, v, \lambda) dv \prod_{j=1}^{k} \phi(x_j) \mathrm{d}(x_1, \dots, x_k),$$

where we write $f(x, v, \lambda)$ for $f(\tau_1(x, v, \lambda), \dots, \tau_m(x, v, \lambda))$ and $v := (v_1, \dots, v_m)$.

3.1 Monte Carlo Valuation

For the valuation one simply needs to calculate the value of the integral numerically. Because of its high dimension paired with the complexity of the integrand one usually resorts to Monte Carlo techniques. In order to calculate a value for the integral one could use the following recipe:
For $l = 1, \dots, N$ generate a sample path ω_l as follows

(i) Draw a realization $(v_1(\omega_l), \dots, v_m(\omega_l), w_1(\omega_l), \dots, w_k(\omega_l))$ of the $m+k$ i.i.d. uniform random variables v_i $(i = 1, \dots, m)$ and w_j $(j = 1, \dots, k)$.

(ii) Calculate the associated standard normal random variables $\epsilon_i = \Phi^{-1}(v_i)$ $(i = 1, \dots, m)$ and $x_j = \Phi^{-1}(w_j)$ $(j = 1, \dots, k)$.

(iii) Calculate Z_i $(i = 1, \dots, m)$ from (1).

(iv) Calculate the default times τ_i $(i = 1, \dots, m)$ from (2).

Loop N times over step (i)-(iv) to create N independent Monte Carlo paths $\omega_1, \dots, \omega_N$. The value V of the product is then estimated by

$$V(\lambda) \approx \tilde{V}(\lambda) := \frac{1}{N} \sum_{l=1}^{N} f(x(\omega_l), v(\omega_l), \lambda). \tag{4}$$

Together with the estimation of the CDO value one may also estimate the standard deviation as an indication of the Monte Carlo error

$$\tilde{\sigma}_V \approx \frac{1}{N} \sqrt{\sum_{l=1}^{N} (f(x(\omega_l), v(\omega_l), \lambda) - \tilde{V})^2}.$$

3.2 Some Variance Reduction Techniques

In order to improve the speed of convergence one may combine (4) with variance reduction techniques. As examples we indicate how to use importance sampling and stratified sampling. The brute force delta calculation will automatically benefit from any convergence improvement for the pricing.

3.2.1 Importance Sampling

We apply importance sampling of the systematic risk factor x by simply shifting the mean and variance for the distribution of x, using $N(\mu, \sigma)$ instead of $N(0, 1)$. Instead of (4) we then have the estimate[5]

$$\tilde{V} \approx \frac{1}{N} \sum_{k=1}^{N} f(x(\omega_l), v(\omega_l), \lambda) \sigma e^{\frac{1}{2}((\frac{x-\mu}{\sigma})^2 - x^2)},$$

where $x \sim \mathcal{N}(\mu, \sigma)$. The likelihood ratio $LR(x) = \sigma e^{\frac{1}{2}((\frac{x-\mu}{\sigma})^2 - x^2)}$ does not directly dependent on the features of the CDO product. However, a good choice for μ and σ, i.e. a setting that improves the speed of convergence, usually depends on the features of the specific product. See [10] for further details. In [9] it is demonstrated how to apply importance sampling on the conditional default probabilities of individual credits for the calculation of credit-VaR in normal copula models.

3.2.2 Stratified Sampling

We also tested the convergence improvement of using stratified sampled systematic risk factors x. For stratified sampling the state space of the random variable is partitioned into intervals of equal probability mass and the random generator ensures that the frequency of samples in each interval exactly correspond to the probability mass of that interval. Within each interval the random variables are sampled from the associated conditional distribution.

[5] Assuming a one factor model for the moment.

4 CDS Par Spread Sensitivities

We are interested in calculating the sensitivity of V with respect to a shift in direction r of the CDS par spread curve s. For example one may be interested in a one basis point parallel shift, i.e. setting $r(t) \equiv 0.0001$[6]. Let us denote by

$$e(s,r) := \lim_{\xi \to 0} \frac{\lambda(s+\xi r) - \lambda(s)}{\xi} \tag{5}$$

the resulting directional shift in default intensities λ. Then we have

$$\begin{aligned}\nabla_r V(\lambda(s)) &:= \lim_{\xi \to 0} \frac{V(\lambda(s+\xi r)) - V(\lambda(s))}{\xi} \\ &= \lim_{\xi \to 0} \frac{V(\lambda(s)+\xi e) - V(\lambda(s))}{\xi} =: \nabla_e V(\lambda).\end{aligned}$$

In order to calculate the sensitivity $\nabla_r V(\lambda(s))$ we thus first calculate the impact on the default intensities (5). It can easily be approximated by bootstrapping default intensities from the shifted CDS par rate curve, giving a finite difference approximation for e

$$\tilde{e}(s,r) := \frac{\lambda(s+\xi r) - \lambda(s)}{\xi}$$

for some small ξ.

It therefore remains to derive the directional sensitivity $\nabla_{\tilde{e}} V(\lambda)$ of the value V as a function of the default intensity with respect to an directional shift $\tilde{e}$ of the default intensity curve λ. Thus we are interested in calculating

$$\nabla_{\tilde{e}} V(\lambda) = \lim_{\xi \to 0} \frac{V(\lambda(s)+\xi \tilde{e}) - V(\lambda(s))}{\xi}. \tag{6}$$

In the following we will consider different methods for the numerical approximation of (6). Purely for the convenience of the exposition of the numerical results we restricted the test cases in Section 6 to 1 basis point parallel shifts of the CDS par rate spreads, i.e. setting $r \equiv 0.0001$ in (5). This is done purely for the convenience of the exposition within this paper[7].

4.1 Brute Force Approach

Within a Monte Carlo simulation the brute force approach for the derivative bumps the model input, reevaluates and calculates a simple finite difference approximation by

$$\begin{aligned}\nabla_e V(\lambda) \approx \frac{1}{\xi} \Delta_e \tilde{V} &:= \frac{1}{\xi} \Big(\frac{1}{N} \sum_{k=1}^{N} p(x,v,\lambda+\xi e) - \frac{1}{N} \sum_{l=1}^{N} p(x,v,\lambda) \Big) \\ &= \frac{1}{N} \sum_{k=1}^{N} \frac{p(x,v,\lambda+\xi e) - p(x,v,\lambda)}{\xi}.\end{aligned}$$

Not surprisingly the slow speed of convergence, the computational inefficiency and the instability of results render this approach almost useless. Stability may be improved somewhat by applying larger shifts of the CDS par rate curve. Even with a large shift of 1%, sometimes used in practice, the speed of convergence is rather slow. Additionally, the delta now is mixed with higher order effects. As we see from the test results in Section 6 these effects are quite large. For 1% shifts they mostly ruin the sensitivity calculation[8].

[6] It is common to normalize the directional shift vector r to one basis point such that the sensitivity is given as "price change per 1 basis point".

[7] Even for a parallel shift of CDS spread sensitivities the associated shift of the term structure of default intensity will not be parallel.

[8] Brute force methods, best mixed with some variance reduction techniques, may still be valuable for stress test scenarios.

The main problem associated with the above brute force calculation is that the delta is mainly determined by just a few paths of the Monte Carlo simulation. Provided that we apply a positive shift to the CDS par spreads, i.e. default times decrease, the delta with respect to underlying i is mainly determined by the few path that result in additional defaults of underlying i after the shift[9]. As only a few path meet the above condition, convergence is slow unless some additional machinery is put to work. One idea is to use any of the available standard variance reduction techniques, such as stratified sampling, control variates or importance sampling. Any of these techniques (and others) may be easily be applied with some benefit, but it still seems more promising to rely on a direct approach, using variance reduction just for some additional benefit where needed. In fact the method derived below can be combined with any of these techniques for additionally speeding up convergence and we provide some simple examples below.

4.2 The Conditional Cumulative Default Distribution

Referring to the representation of the default time τ_i given in Section 2.1 we have

$$\begin{aligned}
P(\tau_i \leq T \mid x) &= P\left(\Phi(Z_i) \leq P(\tau_i \leq T) \mid x\right) = P\left(Z_i \leq \Phi^{-1}\left(P(\tau_i \leq T)\right) \mid x\right) \\
&= P\left(\varepsilon_i \leq \frac{\Phi^{-1}(P(\tau_i \leq T)) - c_i^T x}{\sqrt{1 - c_i^T c_i}} \mid x\right) \\
&= P\left(v_i \leq \Phi\left[\frac{\Phi^{-1}\left(P(\tau_i \leq T)\right) - c_i^T x}{\sqrt{1 - c_i^T c_i}}\right] \mid x\right) \\
&= \Phi\left(\frac{\Phi^{-1}\left(P(\tau_i \leq T)\right) - c_i^T x}{\sqrt{1 - c_i^T c_i}}\right) \\
&= \Phi\left(\frac{\Phi^{-1}\left(1 - e^{-\int_0^T \lambda_i(t)\mathrm{d}t}\right) - c_i^T x}{\sqrt{1 - c_i^T c_i}}\right).
\end{aligned}$$

Since we will make frequent use this representation we honor the term above with a definition:

Definition 1 (Conditional cumulative default distribution): Let τ_i, x be as in Section 2 then we denote the *conditional cumulative distribution function of* τ_i by

$$q_i(T, \lambda_i | x) := \Phi\left(\frac{\Phi^{-1}\left(1 - e^{-\int_0^T \lambda_i(t)\mathrm{d}t}\right) - c_i^T x}{\sqrt{1 - c_i^T c_i}}\right).$$

q_i may be also viewed as the default time T in (v, w)-coordinates, conditional on x. The function $T \mapsto q_i(T, \lambda_i \mid x)$ is the inverse of $v_i \mapsto \tau_i(v, w)$ for a given $x = \Phi^{-1}(w)$.

Note: We include λ_i as an extra parameter since we are interested in sensitivities with respect to λ. Due to the conditioning on x, q_i does not depend on λ_j for $j \neq i$.

4.3 Likelihood Ratio Method

Following along the lines of [4] one could approximate the derivative as

$$\nabla_e V(\lambda) \approx \frac{1}{N} \sum_{l=1}^{N} f(x(\omega_l), v(\omega_l), \lambda) \cdot (LR_i(\omega_l) - 1),$$

[9] For such a path to be a main contributor to the delta it must also result in an additional payout of the default leg of the CDO, i.e. for a standard single tranche CDO the overall losses in the credit pool must be somewhere near the relevant range between attachment and detachment points of the tranche.

where LR_i denotes the likelihood ratio for the change of measure from the original to the shifted default intensities for underlying credit i. Denoting by τ_i, $\tilde{\tau}_i$ the random default times corresponding to the default intensities λ and $\lambda + \xi e$, the conditional probability of default are given by

$$P(\tau_i \leq T|x) = P(v_i \leq q_i(T, \lambda_i|x)) = q_i(T, \lambda_i|x)$$

and

$$P(\tilde{\tau}_i \leq T|x) = P(v_i \leq q_i(T, \lambda_i + \xi e_i|x)) = q_i(T, \lambda_i + \xi e_i|x)$$

respectively. Taking the derivative with respect to T we get the (conditional) density $\psi(\omega|x)$ of τ_i

$$\phi(\omega|x) = \frac{\mathrm{d}}{\mathrm{d}T} q_i(T, \lambda_i|x)\big|_{T=\tau_i(\omega)}.$$

The derivative $\frac{\mathrm{d}}{\mathrm{d}T} q_i(T, \lambda_i|x)$ may be given in closed form as

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}T} q_i(T, \lambda_i|x) &= \frac{\mathrm{d}}{\mathrm{d}T} \Phi\left[\frac{\Phi^{-1}\left(P(\tau_i \leq T)\right) - c_i^T x}{\sqrt{1 - c_i c_i^T}}\right] \\ &= \phi\left[\frac{\Phi^{-1}(P(\tau_i \leq T)) - c_i^T x}{\sqrt{1 - c_i c_i^T}}\right] \frac{e^{-\int_0^T \lambda(s)ds}\lambda(T)}{\sqrt{1 - c_i c_i^T}\,\phi\left(\Phi^{-1}\left(P(\tau_i \leq T)\right)\right)}\end{aligned}$$

and a similar expression for the density of $\tilde{\tau}_i$. The likelihood ratio for a shift of default intensities of credit i is thus given by

$$\begin{aligned}LR_i(\omega_l) = \frac{\phi\left[\frac{\Phi^{-1}(P(\tilde{\tau}_i \leq \tau_i(\omega_l))) - c_i^T x}{\sqrt{1 - c_i c_i^T}}\right]}{\phi\left[\frac{\Phi^{-1}(P(\tau_i \leq \tau_i(\omega_l))) - c_i^T x}{\sqrt{1 - c_i c_i^T}}\right]} \cdot \frac{\phi\left[\Phi^{-1}\left(P(\tau_i \leq \tau_i(\omega_l))\right)\right]}{\phi\left[\Phi^{-1}\left(P(\tilde{\tau}_i \leq \tau_i(\omega_l))\right)\right]} \cdot \\ \cdot \left(1 + \frac{e(\tau_i(\omega_l))}{\lambda(\tau_i(\omega_l))}\right) \cdot e^{-\int_0^{\tau_i(\omega_l)} e(s)ds}.\end{aligned}$$

The method can also be applied for simultaneous shifts of multiple underlying credits. Using the conditional independence feature of the model, the likelihood ratio is given as the product $LR = \prod_{i=1}^m LR_i$.[10]

4.4 Sensitivity Calculation via Conditional Cumulative Default Distribution

Note that for a given x credit i defaults after effective date[11] T_{start} and before maturity T of the CDO, if and only if

$$\begin{aligned} & & T_{\text{start}} &\leq \tau_i &\leq T \\ \Leftrightarrow & & \Phi^{-1}(P(\tau_i \leq T_{\text{start}})) &\leq Z_i &\leq \Phi^{-1}(P(\tau_i \leq T)) \\ \Leftrightarrow & & \frac{\Phi^{-1}(P(\tau_i \leq T_{\text{start}})) - c_i^T x}{\sqrt{1 - c_i c_i^T}} &\leq \epsilon_i &\leq \frac{\Phi^{-1}(P(\tau_i \leq T)) - c_i^T x}{\sqrt{1 - c_i c_i^T}} \\ \Leftrightarrow & & q_i(T_{\text{start}}, \lambda_i|x) &\leq v_i &\leq q_i(T, \lambda_i|x). \end{aligned}$$

[10] Likelihood LR_i for underlying not shifted are set to 1.
[11] The effective date is relevant only for forward starting transactions.

For now we concentrate on spot starting transactions, i.e. setting T_{start} to 0. We find that underlying credit i defaults before maturity T, if and only if $v_i \sim U[0,1]$ is less or equal to the *conditional default barrier* $q_i(T, \lambda_i|x)$ defined above. Using the default barrier allows to numerically calculate the λ derivative very efficiently. W.l.o.g. we consider a shift of default intensities λ_1 of credit 1. To shorten notation let $\bar{v} := (v_2, \ldots, v_m)$ and $q_1(\lambda_1) := q_1(T, \lambda_1|x)$ and $\frac{\mathrm{d}}{\mathrm{d}\lambda_1} = \nabla_{e_1}$ the directional derivative of the curve λ_1 in direction e_1. Then

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\lambda_1}V(\lambda) &= \frac{\mathrm{d}}{\mathrm{d}\lambda_1}\int_{\mathbb{R}^k}\int_{[0,1]^m} f(x,v,\lambda)\mathrm{d}v \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x \\
&= \int_{\mathbb{R}^k}\int_{[0,1]^{m-1}} \frac{\mathrm{d}}{\mathrm{d}\lambda_1}\int_0^1 f(x,v,\lambda)\mathrm{d}v_1 \,\mathrm{d}\bar{v} \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x \\
&= \int_{\mathbb{R}^k}\int_{[0,1]^{m-1}} \frac{\mathrm{d}}{\mathrm{d}\lambda_1}\Big[\int_0^{q_1(\lambda_1)} f(x,v,\lambda)\mathrm{d}v_1 + \int_{q_1(\lambda_1)}^1 f(x,v,\lambda)\mathrm{d}v_1\Big] \mathrm{d}\bar{v} \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x \\
&= \int_{\mathbb{R}^k}\int_{[0,1]^{m-1}} \Big[(f(x,(q_1(\lambda_1)^-,\bar{v}),\lambda) - f(x,(q_1(\lambda_1)^+,\bar{v}),\lambda))\frac{\mathrm{d}q_1(\lambda_1)}{\mathrm{d}\lambda_1} \\
&\qquad + \int_0^{q_1(T,\lambda_1|x)} \frac{\mathrm{d}f(x,v,\lambda)}{\mathrm{d}\lambda_1}\mathrm{d}v_1\Big] \mathrm{d}\bar{v} \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x,
\end{aligned}$$

where the first integral accounts for a discontinuity of the integrand at $v_1 = q_1$. Using the substitution

$$v_1 \to q_1 \cdot v_1, \quad \int_0^{q_1}\mathrm{d}v_1 \to \int_0^1 \mathrm{d}v_1 q_1 \tag{7}$$

for the inner integral over $\frac{\mathrm{d}f(x,v,\lambda)}{\mathrm{d}\lambda_1}$ and adding an integration $\int_0^1 \mathrm{d}v_1 = 1$ over the part that does not depend on v_1 we finally have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\lambda_1}V(\lambda) = \int_{\mathbb{R}^k}\int_{[0,1]^m} &(f(x,(q_1^-,\bar{v}),\lambda) - f(x,(q_1^+,\bar{v}),\lambda))\frac{\mathrm{d}q_1}{\mathrm{d}\lambda_1} \\
&+\frac{\mathrm{d}f}{\mathrm{d}\lambda_1}(x,(v_1\cdot q_1,\bar{v}),\lambda)\cdot q_1 \,\mathrm{d}v \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x.
\end{aligned}$$

For the derivative $\frac{\mathrm{d}f(x,v,\lambda)}{\mathrm{d}\lambda_1}$ we simply use the a finite difference approximation

$$\frac{\mathrm{d}f}{\mathrm{d}\lambda_1}(x,v,\lambda) \approx \frac{1}{\xi}\cdot\big(f(x,v,\lambda+\xi e_1) - f(x,v,\lambda)\big)$$

for some small ξ and thus get

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\lambda_1}V(\lambda) \approx \int_{\mathbb{R}^k}\int_{[0,1]^m} &\big(f(x,(q_1^-,\bar{v}),\lambda,T) - f(x,(q_1^+,\bar{v}),\lambda)\big)\cdot\frac{\mathrm{d}q_1}{\mathrm{d}\lambda_1} \\
&+\frac{f(x,(v_1\cdot q_1,\bar{v}),\lambda+e_1) - f(x,(v_1\cdot q_1,\bar{v}),\lambda)}{\xi}\cdot q_1 \,\mathrm{d}v \prod_{j=1}^{k}\phi(x_j)\mathrm{d}x.
\end{aligned} \tag{8}$$

Rewriting (8) using the default times $\tau = (\tau_1(x,v_1,\lambda),\ldots,\tau_m(x,v_m,\lambda))$, defining $\bar{\tau} :=$

$(\tau_2,\ldots,\tau_m)$ and the modified[12] default time $\tau_1^{[0,T],\xi} := \tau_1(x,(v_1\cdot q_1,\bar{v}),\lambda+\xi e)$ we have

$$\frac{d}{d\lambda_1}V(\lambda) \approx \int_\Omega \Big(f(T^-,\bar{\tau})-f(T^+,\bar{\tau})\Big)\cdot\frac{\mathrm{d}q_1}{\mathrm{d}\lambda_1} + \Big(f(\tau_1^{[0,T],\xi},\bar{\tau})-f(\tau_1^{[0,T],0},\bar{\tau})\Big)\cdot\frac{q_1}{\xi}\,\mathrm{d}\omega. \quad (9)$$

Thus it turns out that the calculation of $\frac{d}{d\lambda_1}V(\lambda)$ may be performed by four[13] pricing formulas with a weighted payoff and a modified default time τ_1 replaced by T^-, T^+, $\tau_1^{[0,T],\xi}$, $\tau_1^{[0,T],0}$. This insight is of great importance for an efficient implementation since it means that the delta calculation may just reuse the pricing code with a modified default time τ_1 and a modified Monte-Carlo weight.

Using the same notation as for the Monte-Carlo pricing the Monte-Carlo approximation of (9) is

$$\begin{aligned}\frac{d}{d\lambda_1}V(\lambda) \approx \frac{1}{N}\sum_{l=1}^{N}&\Big(f(T^-,\bar{\tau}(\omega_l))-f(T^+,\bar{\tau}(\omega_l))\Big)\cdot\frac{\mathrm{d}q_1}{\mathrm{d}\lambda_1}(\omega_l)\\ &+\frac{f(\tau_1^{[0,T],\xi}(\omega_l),\bar{\tau}(\omega_l))-f(\tau_1^{[0,T],0}(\omega_l),\bar{\tau}(\omega_l))}{\xi}\cdot q_1(\omega_l).\end{aligned} \quad (10)$$

The derivative $\frac{\mathrm{d}q_1(\lambda_1)}{\mathrm{d}\lambda_1}$ can be given in closed form

$$\begin{aligned}\frac{\mathrm{d}q_1(\lambda_1)}{\mathrm{d}\lambda_1} &= \lim_{\xi\to 0}\frac{q_1(x,\lambda_1+\xi e_1)}{d\xi}\\ &= \phi\left[\frac{\Phi^{-1}(P(\tau_1\le T))-c_1^T x}{\sqrt{1-c_1c_1^T}}\right]\frac{(1-P(\tau_1\le T))\int_0^T e_1(s)ds}{\sqrt{1-c_1c_1^T}\,\phi\Big[\Phi^{-1}(P(\tau_1\le T))\Big]}.\end{aligned}$$

Instead of taking the limit for $\xi\to 0$ one could also take the approximation for a small, but fixed shift $\xi\cdot e$ (e.g. $\xi=1$ and e equal to the shift of the default intensities resulting from a 1BP parallel shift of the CDS par rate curve). This would lead to

$$\frac{q_1(x,\lambda)}{d\lambda_1} \approx \phi\left[\frac{\Phi^{-1}(P(\tau_1\le T))-\sum_{j=1}^k c_{1j}x_j}{\sqrt{1-c_1c_1^T}}\right]\frac{(1-P(\tau_1\le T))(1-e^{\int_0^T e(s)ds})}{\sqrt{1-c_1c_1^T}\,\phi\Big[\Phi^{-1}(P(\tau_1\le T))\Big]}.$$

In practice the difference is negligible provided the shifts are sufficiently small.

In (10) we found two contributions to the CDS par spread sensitivity. The first term calculates the effect that is associated with a shift of the default barrier. It is given by the sensitivity of the conditional default barrier q with respect to a change in λ times the change in value of the CDO when the default time of credit 1 crosses the CDO maturity T. The second term is due to a shift in default time while holding the default barrier fixed. Not surprisingly for standard CDOs the first part is the main contributor to the overall delta sensitivity[14]. For the numerical calculation of the second integral it is quite convenient to use a positive shift for the CDS par rate[15]. With a positive shift the default intensities will increase and on each path

[12] Note, that the default time with superscript $^{[0,T]}$ are modified so that credit 1 always defaults before maturity T. Effectively this is achieved by sampling an uniform random variable in the interval $[0,q_1(T,\lambda_1|x)]$ and then proceeding as usual. This modification ensures an efficient calculation of the corresponding integral re-using all relevant information from the original simulation. Effectively one may view the substitution (7) as an importance sampling with respect to the random variable v_1.

[13] This does not mean that the delta is four times slower than pricing with the same number of paths. Since most of the "pricing"-simulation is re-used and only information pertaining to credit 1 changes the direct delta calculation is not slowed down by much.

[14] The second part is still significant and can not be neglected.

[15] For the case that the directional shift of the CDS spreads term structure changes signs, one could to split shift vector in its negative and positive part. Here we simply assume that the shift vector is non negative through out.

defaults after shift will happen earlier compared to the unshifted situation. Thus default times can not leave the interval $[0,T]$ and integration over $[0, q_i(T,\lambda_i|x)]$ scans all of the relevant information. This is important because otherwise jumps in the payout would again destabilize the calculations [16]

The method completely avoids the problems associated with the brute force delta calculation. The jumps of the CDO payout, when defaults lie on different sides of the CDO maturity before and after the CDS par spread shift, do not appear and not surprisingly as a result we will see in Section 6 that speed of convergence of the method greatly benefits. As an additional advantage we do not need to assume a large shift of the par spreads in order to numerically stabilize the results. Shifts of 1BP work very well.

Most of the original simulation is reused for the delta calculation according to (10), so that only little extra computational work is necessary. One simply needs to adjust pathwise the information from the original (for determining the CDO price) simulation. The default time of the credit 1 is replaced with (i) default time just before and just after maturity T or (ii) the adjusted default time after the shift has been applied. For the second part the substitution (7) makes sure that only the defaults for credit 1 before maturity T are sampled. The efficiency gains associated with it by far outweigh the small extra computational work.

The method is completely generic in the sense that it does not dependent on any transaction specifics. We indicate below how this model feature can be used for a very elegant and efficient implementation. Once the method has been implemented for one product it is available for all the others without any additional implementation work. This is very important to keep maintenance costs low and to speed up time to marked.

4.4.1 Adding Standard Variance Reduction

To improve convergence even more we additionally mix in some simple variance reduction techniques. Combining the above result with an importance sampling technique for the integration with respect to x and v_1 we get the following result for a Monte Carlo simulation with N paths

$$\frac{dV}{d\lambda_1} \approx \frac{1}{N}\sum_{l=1}^{N} LR(x(\omega_l))\Big[\big(f(T^-,\bar\tau(\omega_l)) - f(T^+,\bar\tau(\omega_l))\big)\cdot\frac{\mathrm{d}q_1}{\mathrm{d}\lambda_1} + \frac{f(\tau_1^{[0,T],\xi}(\omega_l),\bar\tau(\omega_l)) - f(\tau_1^{[0,T],0}(\omega_l),\bar\tau(\omega_l))}{\xi} q_1(\omega_l)\Big].$$

As a second alternative we also introduce a stratified sampling for the common factors x. Its implementation is independent of the delta calculation method and can also be easily combined with importance sampling.

4.4.2 Forward Starting Transactions

In case we have a forward starting transaction we face the same conceptional problem at the effective date as for the the maturity date. Default happening just before and just after the effective date have a very different impact on the value of the CDO. Not surprisingly the method presented above transfers almost one by one to forward starting transactions by observing[17]

[16] Of course analytically in the limit this problem does vanish. Numerically one could also make the negative shifts of the CDS par rates small enough so that the barrier is never crossed. However, extremely small shifts may again lead to numerical problems and are best avoided. Note also that in the model there is zero probability mass on the boundary $q_i(T,\lambda_i|x)$ so we we do not need to consider that event separately.

[17] For forward starting transaction with short forward periods the effect is rather small and may be neglected. This is due to the small likelihood and small changes of default times for early defaults for a given small default intensity shift.

$$
\begin{aligned}
&\frac{d}{d\lambda_1}\int_0^1 f(x,v,\lambda)dv_1 \\
=\;& \frac{d}{d\lambda_1}\left(\int_0^{q_1(T_{\text{start}})} f(x,v,\lambda)dv_1 + \int_{q_1(T_{\text{start}})}^{q_1(T)} f(x,v,\lambda)dv_1 + \int_{q_1(T)}^1 f(x,v,\lambda)\right) \\
=\;& \left(f(x,(q_1(T,\lambda_1|x)^-,\bar{v}),\lambda) - f(x,(q_1(T,\lambda_1|x)^+,\bar{v}),\lambda)\right)\frac{dq_1(T,\lambda_1)}{d\lambda_1} \\
&+ \left(f(x,(q_1(T_{\text{start}},\lambda_1|x),\bar{v})^-,\lambda) - f(x,(q_1(T_{\text{start}},\lambda_1|x)^+,\bar{v}),\lambda)\right)\frac{dq_1(T_{\text{start}},\lambda_1)}{d\lambda_1} \\
&+ \int_{q_1(T_{\text{start}},\lambda_1|x)}^{q_1(T,\lambda_1|x)} \frac{df(x,v,\lambda)}{d\lambda_1}dv_1.
\end{aligned}
$$

Defining $\tilde{q}_1 := \frac{1}{2}(q_1(T) - q_1(T_{\text{start}}))$ the last integral is best split up into two halves that are then treated separately:

$$
\int_{q_1(T_{\text{start}},\lambda_1)}^{q_1(T,\lambda_1)} \frac{df(x,v,\lambda)}{d\lambda_1}dv_1 = \int_{q_1(T_{\text{start}})}^{q_1(T_{\text{start}})+\tilde{q}_1} \frac{df(x,v,\lambda)}{d\lambda_1}dv_1 + \int_{q_1(T)-\tilde{q}_1}^{q_1(T)} \frac{df(x,v,\lambda)}{d\lambda_1}dv_1.
$$

For the first integral we apply a negative shift to the CDS par yield spreads. In that case default intensities decrease and defaults times of credit 1 always increases. This ensures that for the first integral the default times never cross the effective date[18]. Exactly the opposite is true for the second integral. Going through the derivation above and defining the modified[19] default times $\tau_1^{[T_{\text{start}},\cdot],\xi} := \tau_1(x,(q_1(T_{\text{start}})+v_1\tilde{q}_1,\bar{v}),\lambda+\xi e)$ and $\tau_1^{[\cdot,T],\xi} := \tau_1(x,(q_1(T)-v_1\tilde{q}_1,\bar{v}),\lambda+\xi e)$ now leads to

$$
\begin{aligned}
\frac{d}{d\lambda_1}V(\lambda) \approx \frac{1}{N}\sum_{l=1}^{N}&\Big(f(T^-,\bar{\tau}(\omega_l)) - f(T^+,\bar{\tau}(\omega_l))\Big)\cdot\frac{\mathrm{d}q_1(T)}{\mathrm{d}\lambda_1}(\omega_l) \\
&+\Big(f(T_{\text{start}}^-,\bar{\tau}(\omega_l)) - f(T_{\text{start}}^+,\bar{\tau}(\omega_l))\Big)\cdot\frac{\mathrm{d}q_1(T_{\text{start}})}{\mathrm{d}\lambda_1}(\omega_l) \\
&+\frac{f(\tau_1^{[T_{\text{start}},\cdot],0}(\omega_l),\bar{\tau}(\omega_l)) - f(\tau_1^{[T_{\text{start}},\cdot],-\xi}(\omega_l),\bar{\tau}(\omega_l))}{\xi}\cdot\tilde{q}_1(\omega_l) \\
&+\frac{f(\tau_1^{[\cdot,T],\xi}(\omega_l),\bar{\tau}(\omega_l)) - f(\tau_1^{[\cdot,T],0}(\omega_l),\bar{\tau}(\omega_l))}{\xi}\cdot\tilde{q}_1(\omega_l).
\end{aligned}
$$

4.5 Non-Normal Distributional Assumptions

The method transfers easily to other distributions with known densities. As an example we provide some details for the case of the t distribution.

4.5.1 The t-Distribution

Using a t distribution the k factor model now is given by[20]

[18] They also never cross the maturity date as the upper integration bound provided that the shifts are small and effective date and maturity do not lie very close together.

[19] Now the default times are modified such that default always occurs in the interval $[T_{\text{start}},\cdot]$ for the first integral and $[\cdot,T]$ for the second integral. The mid point $\cdot$ depends on x and is therefore random.

[20] See [1] for further details.

$$\begin{aligned} Y_i &= \sum_{j=1}^{k} c_{ij} x_j + \sqrt{1 - c_i c_i^T}\,\epsilon_i \qquad i = 1, \ldots, m \\ Z_i &= Y_i \sqrt{\frac{\nu}{g}}, \end{aligned} \tag{11}$$

where $X_i \sim \mathcal{N}(0,1)$ and $g \sim \chi^2(\nu)$ with ν degrees of freedom. Assuming that X_i and g are independent, Z_i is t distributed with ν degrees of freedom and a density given by

$$t'_\nu(z) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu\pi}} \left(1 + \frac{z^2}{\nu}\right)^{-1} = \frac{1}{b\left(\frac{\nu}{2}, \frac{1}{2}\right)\sqrt{\nu}} \left(1 + \frac{z^2}{\nu}\right)^{-1}.$$

Analog to above the value of the product is now given by

$$V = \int_0^\infty \int_{\mathbb{R}^k} \int_{[0,1]^m} f(x, v, \lambda) dv \prod_{j=1}^{k} \phi(x_j) dx (\chi^2)'(g) dg$$

and the corresponding Monte Carlo simulation would use the following recipe:
For $l = 1, \ldots, N$ generate a sample path ω_l as follows

(i) Draw a realization $(v_1(\omega_l), \ldots, v_m(\omega_l), w_1(\omega_l), \ldots, w_k(\omega_l), q(\omega_l))$ of the $m + k + 1$ i.i.d. uniform random variables v_i $(i = 1, \ldots, m)$, w_j $(j = 1, \ldots, k)$ and q.

(ii) Calculate a χ^2 distributed random variable $g = (\chi^2)^{-1}(q)$ and standard normal random variables $\epsilon_i = \Phi^{-1}(v_i)$ $(i = 1, \ldots, m)$ and $x_j = \Phi^{-1}(w_j)$ $(j = 1, \ldots, k)$.

(iii) Calculate Z_i $(i = 1, \ldots, m)$ from (11).

(iv) Calculate the default times τ_i $(i = 1, \ldots, m)$ by inverting $t_\nu(Z_i) = 1 - e^{-\int_0^{\tau_i} \lambda_i(t) dt}$.

Loop N times over step (i)-(iv) to create N independent Monte Carlo paths $\omega_1, \ldots, \omega_N$. The value V of the product is then estimated by

$$V(\lambda) \approx \tilde{V}(\lambda) := \frac{1}{N} \sum_{l=1}^{N} f(x(\omega_l), g(\omega_l), v(\omega_l), \lambda). \tag{12}$$

As above we can derive for each credit i a conditional default barrier $q_i(T, \lambda_i | x, g)$, now conditional on the realization of x and g. We find, that credit i defaults, if and only if

$$V_i = \Phi(\epsilon_i) \leq \underbrace{\Phi\left[\frac{\sqrt{\frac{g}{\nu}} t_\nu^{-1}\left(P(\tau_i \leq T)\right) - c_i^T x}{\sqrt{1 - c_i c_i^T}}\right]}_{= q_i(T, \lambda_i | x, g)}.$$

The derivative of $q_i(T, \lambda_i | x, g)$ with respect to λ_i, needed for the direct delta calculation, now is given by

$$\begin{aligned} \frac{dq_i(T, \lambda_i | x, g)}{d\lambda_i} &= \phi\left[\frac{\sqrt{\frac{g}{\nu}} t_\nu^{-1}(P(\tau_1 \leq T)) - c_1^T x}{\sqrt{1 - c_1 c_1^T}}\right] \sqrt{\frac{g}{\nu}} \frac{\left(1 - P(\tau_1 \leq T)\right) \int_0^T e(s) ds}{\sqrt{1 - c_1 c_1^T} t'_\nu\left[t_\nu^{-1}(P(\tau_1 \leq T))\right]} \\ &= \phi\left[\frac{t_\nu^{-1}\left(P(\tau_i \leq T)\right) - c_i^T x}{\sqrt{\frac{\nu}{g}}\sqrt{1 - c_i c_i^T}}\right] \frac{\sqrt{g} b\left(\frac{1}{2}, \frac{\nu}{2}\right)}{\sqrt{1 - c_i c_i^T}} \frac{\left(1 - P(\tau_i \leq T)\right) \int_0^T e(s) ds}{\left[1 + \frac{t^{-1}(P(\tau_i \leq T))^2}{\nu}\right]^{-\frac{\nu+1}{2}}} \end{aligned}$$

and leads to a similar recipe as for the normal method. Again the method is completely generic. It does not depend on any product specifics and can be used with the variance reduction schemes applied above or any other available in the literature. Code changes necessary to introduce a t distribution are limited to just a few lines.

5 Implementation

The proposed method for sensitivity calculation allows for an object oriented implementation which is highly efficient with respect to calculation performance and coding time (time to market). Using the design pattern of a stochastic iterator, the default time iterator, we are able to create a highly flexible product implementation framework in which any product may become the underlying of any other product. In this section we will sketch the basic ideas related to the implementation.[21]

5.1 Reuse of Pricing Code for Delta Calculation

From equation (10) it is clear that the calculation of delta relies on the pricing with modified default times. Thus pricing and delta calculation may share a major portion of the pricing code if one considers the method

```
/**
 * @return Returns the vector of values by modifiyng the models default times of unterlying <code>underlyingID</code>
 */
double[] getValueForGivenDefaultTimes(ConditionalLossModel model, int underlyingID, double[] defaultTimes)
```

which returns the vector of values by path (random variable) where the value has been calculated by taking the given model and forcing the default times of a the underlying specified by `underlyingID` to the provided default times. Using the convention that a non valid `underlyingID` or an empty default time vector will result in no modification of the underlying model the code may be used for pricing as well.

Implementing the method `getValueForGivenDefaultTimes` thus has a striking advantage: For pricing and delta there is only one common location where the product specific pricing code has to be implemented. Once the pricing code for a product is written the delta is available immediately. This again shows that the delta calculation with our method is independent of any product features[22].

Using `getValueForGivenDefaultTimes` the pricing becomes

```
/**
 * @return Returns the value of the product
 */
double getValue(ConditionalLossModel model) {
    double[] valueOnPath = getValueForGivenDefaultTimes(model, 0, null);

    // Calculate value
    double value = 0.0;
    for(int pathIndex = 0; pathIndex<model.getNumberOfPaths(); pathIndex++) value += valueOnPath[pathIndex];

    return value / model.getNumberOfPaths();
}
```

and the calculation of delta is performed through

```
/**
 * @return Returns the delta of the product
 */
double getDelta(ConditionalLossModel model, int underlyingID, double shift) {
    double timeHorizon = tenorStructure[tenorStructure.length-1];

    // Get values with default of underlying at maturity
    double[] valueDefault       = getValueForGivenDefaultTimes(model, underlyingID, timeHorizon);

    // Get values with non default of underlying
    double[] valueNonDefault    = getValueForGivenDefaultTimes(model, underlyingID, timeHorizon+1);

    // Get derivative of default barrier for corresponding underlying
    double[] defaultBarrierDerivative = model.getBarrierDerivative(underlyingID, timeHorizon);

    // Get modified default times
    double[] modifiedDefaultTimesBeforeShift = model.getModifiedDefaultTimes(underlyingID, 0.0);
    double[] modifiedDefaultTimesAfterShift  = model.getModifiedDefaultTimes(underlyingID, shift);

    // Get values with modified default times with and without shift
    double[] valueModDefault
        = getValueForGivenDefaultTimes(model, underlyingID, modifiedDefaultTimesBeforeShift);

    double[] valueModDefaultWithShift
        = getValueForGivenDefaultTimes(model, underlyingID, modifiedDefaultTimesAfterShift);
```

[21] The code fragments shown are given in the Java™ language.

[22] Apart from the conditional cumulative default time distribution function q, the delta calculation could also be viewed as model independent.

```
        // Get default barrier
        double[] defaultBarrier = model.getDefaultBarrier(underlyingID, timeHorizon);

        // Calculate delta and average over values
        double delta = 0.0;
        for(int pathIndex = 0; pathIndex<model.getNumberOfPaths(); pathIndex++) {
            delta +=
              (valueDefault[pathIndex]             - valueNonDefault[pathIndex]) * defaultBarrierDerivative[pathIndex]
            + (valueModDefaultWithShift[pathIndex] - valueModDefault[pathIndex]) * defaultBarrier[pathIndex] / shift;
        }

        return delta / model.getNumberOfPaths();
    }
```

The methods `getValue`, `getValueForGivenDefaultTimes` and `getDelta` are members of the product class which may derive from an abstract product base class. Following the idea of having `getValueForGivenDefaultTimes` as the single place of a products pricing code the methods `getValue` and `getDelta` may be implemented on the level of this abstract product base class.

5.2 Efficient Model Interface: The Default Time Iterator Design Pattern

We propose to implement the actual pricing code of the product by using an iterator design pattern [7] for the default times. This *default time iterator* efficiently provides a sequence of default times and corresponding losses of the underlyings for a given time period. Since default times are stochastic the iterator is stochastic itself, i.e. it carries a path index.[23]

```
public interface DefaultTimeIterator {

    // @brief Modifies the default times of underlying <code>underlyingID</code> to <code>defaultTimes</code>
    void setModifiedDefaultTimes(int underlyingID, double[] defaultTimes);

    // @brief Restricts the iterator to a specific range.  Only default times in (periodStart,periodEnd] will be
 returned
    void setPeriod(double periodStart, double periodEnd);


    void nextEntry(int path);

    boolean hasEntry(int path);

    // @return current entry:  default time
    double getTime(int path);

    // @return current entry:  loss
    double getLoss(int path);

    // @return current entry:  underlying
    int getUnderlyingID(int path);
}
```

This is a stochastic iterator, since it is parametrized by a path index, that as three values in each state (time, loss and underlying). Note that this interface differs slightly from the Java™2 Plattform SE v1.4.2 Iterator in the handling access (getters) and incrementation (nextEntry, hasEntry).

The pricing code of the product then iterates through defaults/losses provided by the default time iterator and calculates (loss triggered) payments depending on product specific information such as subordination or waterfall structures. The modification of defaults of a given underlying, that we need for the delta calculation, may be integrated directly into the default time iterator. This lets the method `getValueForGivenDefaultTimes` (see Listing 1) look even leaner: The use of the default time iterator will then look as follows:

```
            for(int iPath=0; iPath<model.getNumberOfPaths(); iPath++) {
                for(;defaultTimeIterator.hasEntry(iPath) ; defaultTimeIterator.nextEntry(iPath)) {
                    // Get default from underlying model
                    double  time               = defaultTimeIterator.getTime(iPath);
                    double  loss               = defaultTimeIterator.getLoss(iPath);

                    // ...  calculate loss triggered product specific features payments...
                }
            }
```

[23] In other words we have a family of iterators prametrized by the path index.

5.3 Allowing any Product to Become the Underlying of any other: Efficient Implementation of Power CDOs

With the design pattern of a default time iterator, that is provided by the models default time iterator factory, there is a simple way to provide a highly flexible product implementation framework where any product may become the underlying of any other product. To accomplish this only two parts have to be implemented:

1. Any product which potentially may become an underlying provides its own default time iterator, that is synchronized with the model default time iterator.

2. A CDO requests the losses from its underlyings via the underlyings default time iterator.

To ensure efficiency of this approach one exception is made: The underlying CDSes share one common default time iterator, namely the models default time iterator.

The two ideas above are realized by a default time iterator interface (abstract class) which is implemented by any product

```
    private DefaultTimeIterator[]   embededProductsDefaultIterator;
    private double[]                embededProductsNotionals;
```

and a small modification of the pricing code where the loss aggregation becomes

```
            for(int iPath=0; iPath<model.getNumberOfPaths(); iPath++) {
                for(;defaultTimeIterator.hasEntry(iPath) ; defaultTimeIterator.nextEntry(iPath)) {
                    // Get default from underlying model
                    double  time                = defaultTimeIterator.getTime(iPath);
                    double  loss                = defaultTimeIterator.getLoss(iPath);

                    // Add loss of embeded products
                    for(int embededProductIndex=0; embededProductIndex<embededProductsNotionals.length; embededProductIndex++) {
                        // Add loss over embeded product
                        loss += embededProductsNotionals[embededProductIndex] *
                                embededProductsDefaultIterator[embededProductIndex].getLoss(iPath);

                        // Advance emeded products iterator
                        embededProductsDefaultIterator[embededProductIndex].nextEntry(iPath);
                    }

                    // ...  calculate loss triggered product specific features payments...
                }
            }
```

Following this design pattern we create a single product class that may represent a CDO, CDO^2, CDO^3, . . . , while simultaneously being lean and elegant. See Listing 1 for an example.

Listing 1: *A generic single tranche CDO with arbitrary underlyings (e.g. CDS, CDO, CDO^2,...). For notional convenience the payout has been slightly simplified.*

```
public class SingleTrancheCDO {
    private double[]    tenorStructure;
    private double      notional;
    private double      subordination;
    private double      spread;

    private DefaultTimeIterator[]   embededProductsDefaultIterator;
    private double[]                embededProductsNotionals;

    /**
     * @return Returns the vector of values by modifiyng the models default times of unterlying <code>underlyingID</code>
     */
    double[] getValueForGivenDefaultTimes(ConditionalLossModel model, int underlyingID, double[] defaultTimes)
    {
        // Allocate memory for random variables value, notional and subordination
        double[] valueOnPath            = new double[model.getNumberOfPaths()];
        double[] notionalOnPath         = new double[model.getNumberOfPaths()];
        double[] subordinationOnPath    = new double[model.getNumberOfPaths()];

        // Initialize value, notional and subordination
        Arrays.fill(valueOnPath, 0.0);
        Arrays.fill(notionalOnPath, notional);
        Arrays.fill(subordinationOnPath, subordination);

        // Allocate default time iterator
        DefaultTimeIterator defaultTimeIterator = model.getDefaultTimeIterator();

        // Initialize default times modification
        defaultTimeIterator.setModifiedDefaultTimes(underlyingID, defaultTimes);

        // Loop over periods
        for(int iPeriod = 0; iPeriod < tenorStructure.length; iPeriod++) {
            double periodStart     = tenorStructure[iPeriod];
            double periodEnd       = tenorStructure[iPeriod+1];

            double daycountFraction     = periodStart-periodEnd;

            // Restrict the iterator to default times t in (periodStart , periodEnd]
            defaultTimeIterator.setPeriod(periodStart, periodEnd);

            for(int iPath=0; iPath<model.getNumberOfPaths(); iPath++) {
                for(;defaultTimeIterator.hasEntry(iPath) ; defaultTimeIterator.nextEntry(iPath)) {
                    // Get default from underlying model
                    double  time                = defaultTimeIterator.getTime(iPath);
                    double  loss                = defaultTimeIterator.getLoss(iPath);

                    // Add loss of embeded products
                    for(int embededProductIndex=0; embededProductIndex<embededProductsNotionals.length; embededProductIndex++) {
                        // Add loss over embeded product
                        loss += embededProductsNotionals[embededProductIndex] *
                                embededProductsDefaultIterator[embededProductIndex].getLoss(iPath);

                        // Advance emeded products iterator
                        embededProductsDefaultIterator[embededProductIndex].nextEntry(iPath);
                    }

                    // Degrade subordination
                    subordinationOnPath[iPath] -= loss;

                    // If subordination has become negative we have a loss withing this CDO
                    double lossOverSubordination    = -Math.min(subordinationOnPath[iPath],0.0);

                    // Floor subordination at 0.0
                    subordinationOnPath[iPath]      =  Math.max(subordinationOnPath[iPath],0.0);

                    // Cap loss by notional
                    lossOverSubordination = Math.min(lossOverSubordination, notionalOnPath[iPath]);

                    // Degrade notional
                    notionalOnPath[iPath] -= lossOverSubordination;

                    // Consider protection payment
                    valueOnPath[iPath] += lossOverSubordination * model.getDiscountFactor(time);
                }

                // Consider protection fee
                valueOnPath[iPath] -= notionalOnPath[iPath] * spread * daycountFraction * model.getDiscountFactor(periodEnd);
            }
        } // for(iPeriod)

        return valueOnPath;
    }
}
```

6 Numerical Results

In this chapter we provide some numerical results on the methods described above. We demonstrate that the model has been implemented correctly by comparing Monte Carlo prices and deltas for simplified single tranche CDOs against the semi-analytical methods in [1]. Ultimately we are interested in the speed of convergence of our direct delta method compared to other methods that are popular in a Monte Carlo framework. To demonstrate the efficiency the direct delta method we compare the speed of convergence against a brute force delta calculation mixed with some variance reduction techniques. To stress that the direct delta method is generic we use not only a single tranche CDO as a test case, but also one CDO^2 with four underlying single tranche CDOs . As the main results do not depend on the specific market data setting we use somewhat artificial "market data" for our tests.

6.1 Setup of Test Cases

The discount curve we are using is bootstrapped from the market information that resembles the EUR market at the beginning of 2005.

PRODUCT	MATURITY	RATE
Money Market Cash[24]	1 day	2.03%
Money Market Cash	7 days	2.03%
Money Market Cash	1 month	2.08%
Money Market Cash	2 months	2.09%
Money Market Cash	3 months	2.10%
Money Market Cash	4 months	2.12%
Money Market Cash	5 months	2.13%
Money Market Cash	6 months	2.15%
Annual Swap[25]	1 year	2.33%
Annual Swap	2 years	2.53%
Annual Swap	3 years	2.72%
Annual Swap	4 years	2.89%
Annual Swap	5 years	3.05%

The underlying Credit Default Swaps for our test case CDOs are simplified. We use a total of 100 credits, grouped into 5 categories. Credits within each category share the same parameters and we therefore expect that credits in the same group have identical CDS spread deltas. The categories differ in assumed recovery rates, in the assumed CDS par yield curve and their correlation setting. The categories are characterized by the following table

CREDIT GROUP	RECOVERY	CDS CURVE	CORR. GROUP
Group 1	60%	CDS Curve low	1
Group 2	20%	CDS Curve low	1
Group 3	60%	CDS Curve high	1
Group 4	20%	CDS Curve high	1
Group 5	20%	CDS Curve high	2

where the two distinct CDS curves are given in the following table

[24] Using *day count convention* act/360, *day roll convention* modified following.
[25] Using *day count convention* 30/360, *day roll convention* modified following.

CDS CURVE	1 YEAR	3 YEARS	5 YEARS
CDS Curve low	10BP	15BP	20BP
CDS Curve high	20BP	60BP	100BP

and the correlation between the underlying credits is determined by their affiliation to a *correlation group* as indicated in the table above. The correlation between the members of a correlation group as given in the following table.

	CORR GROUP 1	CORR GROUP 2
GROUP 1	20%	0%
GROUP 2	0%	0%

Note, that a one factor model is sufficient to fit the correlation structure perfectly.

The details of the CDO test products are given in the following table[26]. The benchmark CDO use the simplifications and approximations laid out in [1]. With these assumptions [27] a semi-analytical pricing is feasible and can use them as benchmarks for our Monte Carlo implementation if, for consistency reasons, we use the same simplified assumptions. The associated Monte Carlo prices are found under the label BENCH. MC in the table below. Note that the difference between Monte Carlo benchmark prices and the semi-analytical prices are consistent to the reported error of the MC simulation[28].

METHOD	BRUTE FORCE	3 YEARS	5 YEARS
CDS Curve low	10BP	15BP	20BP
CDS Curve high	20BP	60BP	100BP

Besides these simplified benchmark products we also consider the pricing of unmodified products [29]. For all Monte Carlo prices the table also indicates the associated Monte Carlo errors of the price estimation for a simulation with 100,000 paths. The absolute MC error of the simulation is well below 1 BP of the notional for all products.

The underlying of the CDO^2 are the four STCDOs. For the CDO^2 no semi-analytical benchmark is available, so that the table does not include any benchmark prices.

PARAMETER	STCDO 1	STCDO 2	STCDO 3	STCDO 4	CDO^2
Notional Group 1 (mEUR)	40	66.667	0	0	—
Notional Group 2 (mEUR)	40	66.667	0	66.667	—
Notional Group 3 (mEUR)	40	66.667	66.667	66.667	—
Notional Group 4 (mEUR)	40	0	66.667	66.667	—
Notional Group 5 (mEUR)	40	0	66.667	0	—
Pool Notional (mEUR)	4,000	4,000	4,000	4,000	4,000
Tranche Notional (mEUR)	100	100	100	100	200
Subordination	7.5%	7.5%	7.5%	7.5%	2.0%
Maturity in years	5	5	5	5	5
Spread	0.75%	0.50%	2.50%	1.50%	1.50%

to be continued

[26] Some minor information is neglected here. For example we assume that all CDOs are based on quarterly term structure, pay a fixed spread based on act/360 day count convention.

[27] The main simplification is that only loss distributions on the quarterly tenor structure are considered. Therefore all losses need to be mapped to to the quarterly tenor structure.

[28] Further convergence tests showed that the Monte Carlo error vanishes when the number of paths are increased.

[29] As the benchmark products only modify the loss time so that it falls on the quarterly tenor structure, the price difference between the simplified benchmark product and the corresponding STCDO is in general rather small.

DEAL SPECIFICS	STCDO 1	STCDO 2	STCDO 3	STCDO 4	CDO^2
Benchmark MC (EUR)	923,199	276,307	1,099,955	1,967,318	—
Benchmark SA (EUR)	946,718	280,200	1,205,928	2,042,833	—
MC error (EUR)	32,639	22,137	84,094	58,365	—
Relative MC error (BP)	3.3 BP	2.2 BP	8.4 BP	5.8 BP	—
Price (EUR)	872,230	242,409	954,386	1,875,836	87,179
MC error (EUR)	32,754	22,212	77,432	56,892	138,819
Relative MC error (BP)	3.2 BP	2.2 BP	7.7 BP	5.7 BP	6.9 BP

6.2 Credit Spread Sensitivities of a Single Tranche CDO and CDO^2

The following graphic displays the delta (1 BP parallel shift of CDS par yield spreads of single underlying credit) for benchmark STCDO 1 calculated with the semi-analytical benchmark, the brute force delta and our direct delta method using 10,000 paths for the MC methods. It shows that the direct delta method is very close to the correct[30] semi-analytical deltas. The difference is within one standard deviation of the delta estimator as indicated in the graph by error bars. Not surprisingly all deltas of credits within the same group have to be the same. This is nicely reflected in our delta calculation, as the Monte Carlo error is very small and does not obscure the structure. With the brute force method on the other hand it is not even feasible to get an idea of the magnitude of the delta and the structural feature is completely concealed by the Monte Carlo error. For the brute force delta there are only two possible routes to improve.

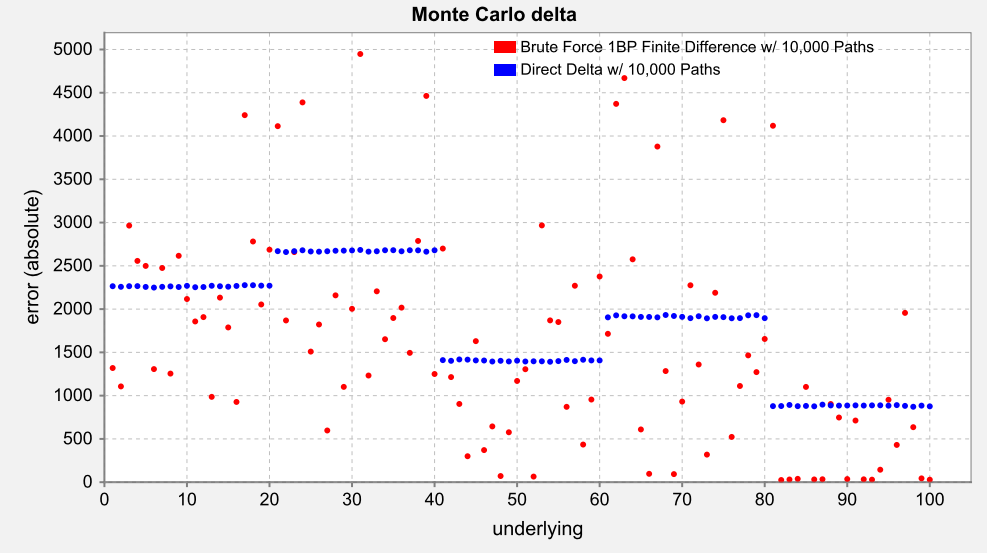

Figure 1: *Single Tranche CDO: CDS Par Spread Deltas 1BP Shift*

Either increase the number of paths or increase the shift applied for the difference quotient in order to improve stability of the calculation. Figure 2 shows the result. Even with 1,000,000 paths the results for 1BP shifts are not as convincing as the deltas calculated with 10,000 paths and our method. Increasing the shift to 100BP the brute force delta calculation stabilizes, but due to significant higher order effects the result is now about 20% off the correct figures. This already indicates that the efficiency gains of our direct delta method are huge. The standard error for the delta calculated with the brute force method calculated with 1,000,000 simulation

[30] If we accept the simplifications necessary for the semi-analytical method it provides the correct results.

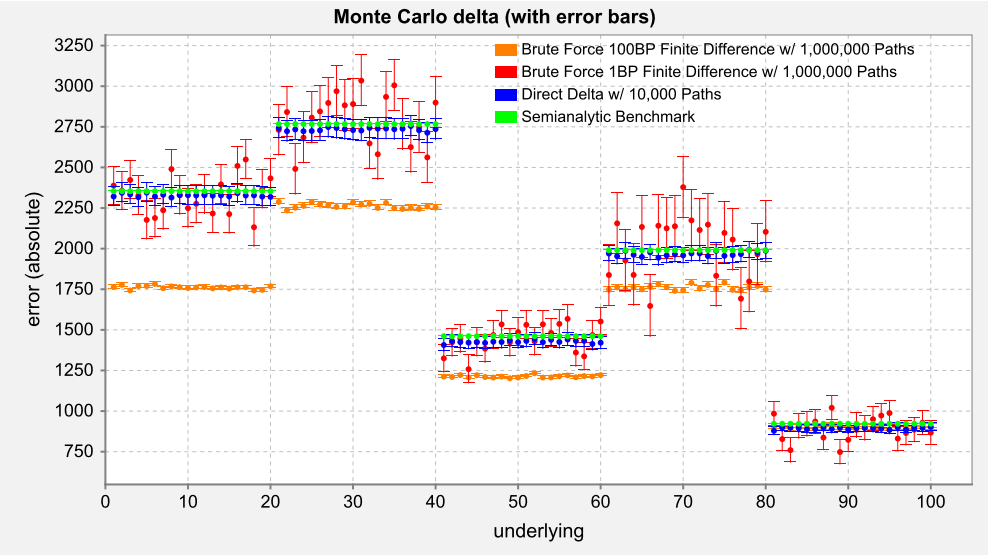

Figure 2: *Benchmark STCDO 1: CDS Par Spread Deltas*

paths lies between 100 and 200, while the direct delta MC error is between 25 and 60 with as little as 10,000 paths. We can therefore expect an efficiency gain by a factor beyond 100. In the following section we further analyze the convergence issues.

6.3 Convergence Properties of CDS Spread Delta Calculations

In this section we compare the convergence speed of the direct delta method outlined above against the standard brute force method, where we additionally applied importance sampling and stratified sampling for both methods. Figure 3 indicates the convergence of the different methods. It shows the Monte Carlo error for the estimation of the CDS delta for STCDO 1 with respect to credit 1 (in group 1). Some improvements can be achieved by the standard variance reduction techniques, but clearly not to the same scale possible with our direct delta method. In line with the square root convergence of MC methods we define *efficiency gain* by the squared quotient of the MC error of the standard brute force method over the method under consideration. For the various calculation methods we found in our tests[31] the following efficiency gains for STCDO 1.

Calculation Method	Sampling Method	Efficiency Gain for Deltas per Credit Group				
		Group 1	Group 2	Group 3	Group 4	Group 5
Brute Force	Standard	1.0	1.0	1.0	1.0	1.0
	Importance(IS)	2.4	2.3	2.0	2.6	2.2
	Stratified(SS)	1.0	1.0	0.9	1.2	1.0
	IS + SS	2.5	2.4	1.9	2.6	2.2
Direct Delta	Standard	273	484	402	781	1,507
	Importance(IS)	969	1,741	1,194	2,405	3,755
	Stratified(SS)	416	746	558	1,156	2,131
	IS + SS	1,148	2,323	1,403	3,154	4,814

[31] Tests for stratified sampling not yet completed.

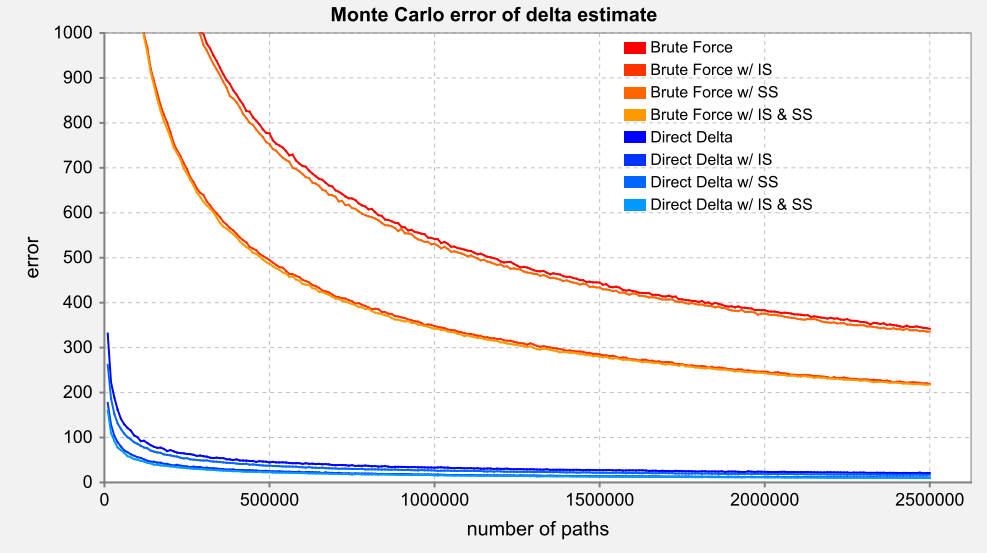

Figure 3: *STCDO 1: Estimated MC Error of CDS Par Spread Deltas (Credit 1)*

In our test case we achieve efficiency gains over the brute force Monte Carlo method of around factor 2 using the standard variance reduction techniques importance sampling and stratified sampling (see also Figure 4, 6). Using our direct delta method, however, efficiency gains are well above a factor of 100 and for some even above 1,000 (see also Figure 5, 7). Combining our method with variance reduction schemes we achieve a efficiency gain by a factors beyond 1,000. With real/near time calculation of deltas become possible. For example the calculation of all 100 CDS deltas for our five test products (totalling 500 sensitivities) with 10,000 simulation paths takes about 190 seconds on a standard PC (3GHz Single Pentium Processor, 1GB RAM). Thus one delta is calculated in less than half second. For 10,000 simulation paths the relative standard error of the delta estimation is around 2.5%. It is important to note that roughly 5,000,000 simulation paths would be necessary in a brute force delta calculation without any variance reduction in order to achieve about the same accuracy. In our experience the direct delta method takes roughly the same computational time as a brute force delta[32]. Thus for the 5 products one would needs to invest roughly 1 day of computational time for calculating all CDS deltas with the same kind of accuracy.

As noted above the direct delta method is completely generic and can be used with any CDO product without any additional implementation. For the CDO^2 we find the following efficiency gains over the standard brute force method.

Calculation Method	Sampling Method	Efficiency Gain for Deltas per Credit Group				
		Group 1	Group 2	Group 3	Group 4	Group 5
Brute Force	Standard	1.0	1.0	1.0	1.0	1.0
	Importance(IS)	2.5	1.9	1.8	1.7	2.1
	Stratified(SS)	1.0	1.0	1.0	1.1	1.0
	IS + SS	2.7	1.9	1.7	1.6	1.8
Direct Delta	Standard	479	2,051	4,244	4,807	3,720
	Importance(IS)	841	1,312	1,022	1,656	2,922
	Stratified(SS)	352	1,312	598	1,137	2,442

to be continued

[32] For the brute force delta the product has to be priced twice, while the direct delta is based on a single simulation.

Calculation Method	Sampling Method	Efficiency Gain for Deltas per Credit Group				
		Group 1	Group 2	Group 3	Group 4	Group 5
	IS + SS	838	1,726	1,256	2,308	3,721

As these results are pretty similar to those for the STCDO1 we are fairly confident that the efficiency gains that we found for our method does transfer to other more complex products as well.

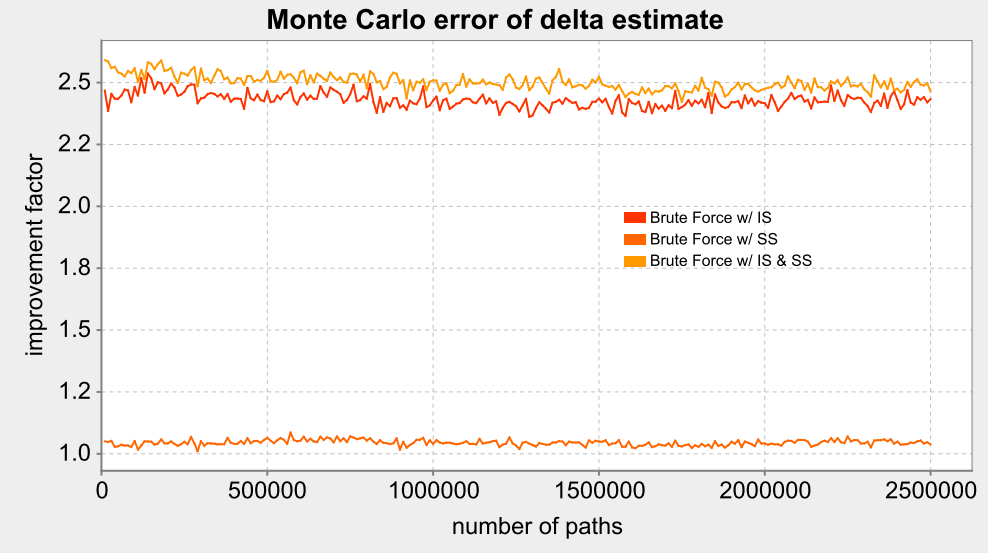

Figure 4: *STCDO 1: Estimated Efficiency Gains of some standard Variance Reduction Schemes for Delta Calculation (Credit Group 1)*

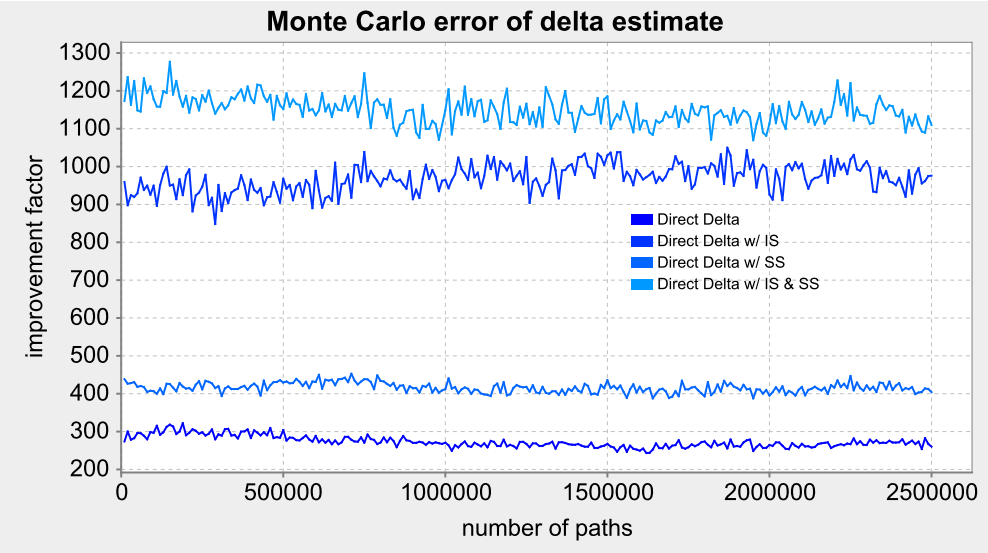

Figure 5: *STCDO 1: Estimated Efficiency Gains of Direct Delta Method (Credit Group 1)*

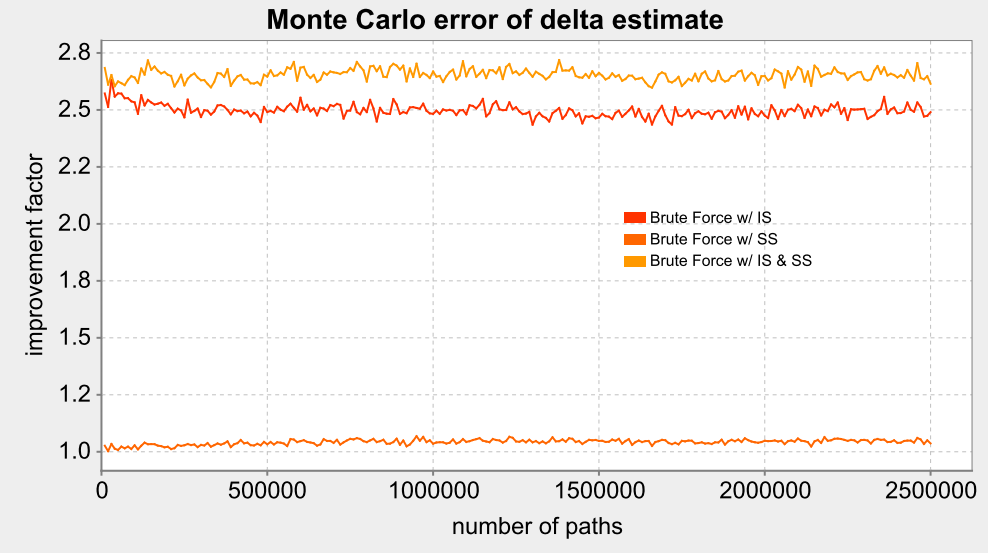

Figure 6: *CDO^2: Estimated Efficiency Gains of some standard Variance Reduction Schemes for Delta Calculation (Credit Group 1)*

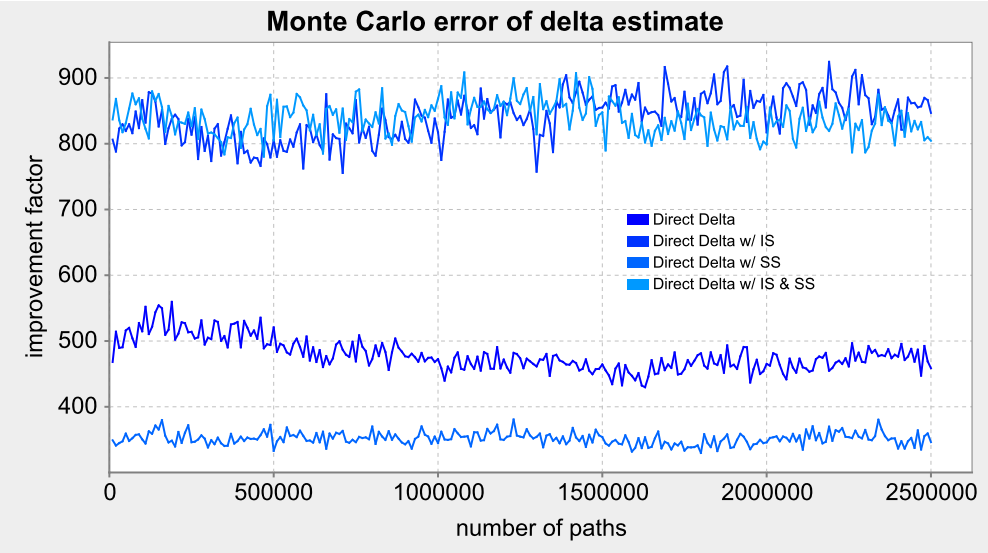

Figure 7: *CDO^2: Estimated Efficiency Gains of Direct Delta Method (Credit Group 1)*

7 Concluding Remarks

In their first days Monte Carlo has been the method of choice for complex basket credit products. So far one of the main shortcomings of that approach has been the slow convergence of its risk sensitivities. This has lead many to investigate other alternative numerical implementations. In this paper we have presented a method that allows to calculate deltas for CDOs very efficiently in a Monte Carlo framework. What's even more, our method is generic, i.e. applicable for arbitrary CDO structures without any need for additional research or implementation efforts, and, as shown explicitly in the code fragments, it leads to a lean implementation. Both of these features are very important from a practical point of view. Finally we note that it should be straight forward, possibly a little tedious, to expand the method to second order sensitivities.

List of Symbols

Symbol	Meaning
k	Number of common factors in the CDO Conditional Loss Model.
m	Number of individual names.
N	Number of monte carlo sample paths.
s_i	Risk neutral credit default spread curve of the i-th individual name.
s	Vector of the risk neutral credit default spread curves.
λ_i	Default intensity function of the i-th individual name, $t \mapsto \lambda_i(t)$.
λ	Vector of default intensities.
τ_i	Default time of the i-th individual name (random variable).
$q_i(t, \lambda_i \mid x)$	Conditional distribution function of τ_i, conditioned on x and for a given value of the i-th default intensity. The function $\tau \mapsto q_i(\tau, \lambda_i \mid x)$ is the inverse of $v_i \mapsto \tau_i(v, w)$ for a given $x = \Phi^{-1}(w)$.
$\mathcal{N}(\mu, \sigma)$	Normal distribution with mean μ and standard deviation σ.
ϕ	Density of the standard normal distribution: $\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2)$.
Φ	Cumulative distribution function of the standard normal distribution: $\Phi(x) = \int_{-\infty}^{x} \phi(\xi)\, \mathrm{d}\xi$.
T	Maturity of the product.
x^T	Vector of common/systematic factors.
c_i	Vector of factor loadings for credit i.
ε_i	Random variable for the idiosyncratic risk of credit i.
V	Value of the CDO product.
$\tilde{V}$	Monte Carlo approximation of the value of the CDO product.
$f(\cdot)$	Function giving the time-0 value of a discounted cashflow of the CDO.
r_i	Directional shift of the credit default term structure s_i.
$e(s, r)$	Directional shift of the default intensity term structure given a CDS spread curve s and a directional shift r. Due to the bootstrapping the shift also depends on the recovery rate of the underlying credit. This dependency is neglected in the notation.
∇_e	Directional derivative in direction e. $\nabla_e V(\lambda) := \lim_{\xi \to 0} \frac{V(\lambda(s)+\xi e) - V(\lambda(s))}{\xi}$
t_ν	The t-distribution with ν degrees of freedom.
$b(z, w)$	The beta function $b(z, w) = \int_0^1 t^{z-1}(1-t)^{w-1} dt$.

References

[1] ANDERSEN, LEIF; SIDENIUS, JAKOB; BASU, SUSANTA: All your hedges in one basket. Risk Magazine 11/2003, 67-72 (2003).

[2] BRASCH, HANS-JÜRGEN: A Note on Efficient Pricing and Risk Calculation of Credit Basket Products. Preprint, 2005. http://defaultrisk.com/pp_crdrv_54.htm.

[3] BOYLE, PHELIM; BOADIE, MARK; GLASSERMAN, PAUL: Monte Carlo methods for security pricing. Journal of Economic Dynamics and Control, 21, 1267-1321 (1997).

[4] BROADIE, MARK; GLASSERMAN, PAUL: Estimating Security Price Derivatives using Simulation. *Management Science*, 1996, Vol. 42, No. 2, 269-285.

[5] FOURNIÉ, ERIC; LASRY JEAN-MICHEL; LEBUCHOUX, JÉRÔME; LIONS, PIERRE-LOUIS; TOUZI, NIZAR: Applications of Malliavin calculus to Monte Carlo methods in finance. Finance Stochastics. 3, 391-412 (1999). Springer- Verlag 1999.

[6] FRIES, CHRISTIAN P.; KAMPEN, JÖRG: Proxy simulation schemes using likelihood ratio weighted Monte Carlo for generic robust Monte-Carlo sensitivities and high accuracy drift approximation (with applications to the LIBOR Market Model). 2005. http://www.christian-fries.de/finmath/proxyscheme

[7] GAMMA, ERICH; HELM, RICHARD; JOHNSON, RALPH E.: Design Patterns. Addison-Wesley Professional, 1997. ISBN 0-2-016-3361-2.

[8] GLASSERMAN, PAUL: Monte Carlo Methods in Financial Engineering. (Stochastic Modelling and Applied Probability). Springer, 2003. ISBN 0-387-00451-3.

[9] GLASSERMAN, PAUL; LI, JINGYI: Importance Sampling for Portfolio Credit Risk. Working paper, Columbia University (2003). http://www2.gsb.columbia.edu/faculty/pglasserman/Other/is_credit.pdf.

[10] JOSHI, MARK S.: Applying Importance Sampling to Pricing Single Tranches of CDOs in a One-Factor Li Model. QUARC, Group Risk Management, Royal Bank of Scotland. Working paper, 2004.

[11] KLOEDEN, PETER E.; PLATEN, ECKHARD: Numerical Solution of Stochastic Differential Equations (Applications of Mathematics. Stochastic Modelling and Applied Probability Vol. 23). Springer Verlag, 1999. ISBN 3-540-54062-8.

[12] LI,D.: On Default Correlation: a Copula Approach (Journal of Fixed Income, 9, 43-45)