FAST GAMMA COMPUTATIONS FOR CDO TRANCHES

MARK JOSHI AND CHAO YANG

Abstract. We demonstrate how to compute first- and second-order sensitivities of portfolio credit derivatives such as synthetic collateralized debt obligation (CDO) tranches using algorithmic Hessian methods developed in Joshi and Yang (2010) in a single-factor Gaussian copula model. Our method is correct up to floating point error and extremely fast. Numerical result shows that, for an equity tranche of a synthetic CDO with 125 names, we are able to compute the whole Gamma matrix with computational times measured in seconds.

1. Introduction

The credit derivatives market has experienced an incredible growth in market size over the last decade. Although the market has shrunk after the financial crisis, effective pricing and risk management of credit derivatives is still of high importance to both buyers and sellers of such financial products.

The industry standard to price portfolio credit derivatives is to use copula methods to model joint defaults of the underlying assets. Li (2000) used Gaussian copula with a small number of latent factors to price credit derivatives in a Monte Carlo simulation. The main benefit from using factor models is that the dimensionality of the problem becomes the number of latent factors rather than the number of assets, thus it leads to substantial time reductions in the model implementation. To further improve numerical efficiency, Andersen et al (2003) and Laurent and Gregory (2005) introduced efficient semi-analytic methods to price portfolio credit derivatives. Under such approaches, the pay-off of portfolio credit derivatives can be written as functions of the portfolio loss distribution at different times. Efficient recursive methods are then developed to compute the portfolio loss distribution in conjunction with the copula modelling.

These methods have been extended to the computation of Greeks. Andersen et al (2003) developed recursive formulas for first-order Greeks, and O'Kane (2008) adopted a related finite-differencing approach. Both approaches involved developing the full loss distribution and then backing out the loss distribution with one less name. This procedure can be numerically unstable (see O'Kane (2008)), and the need to develop the full loss distribution is unappealing when only a small part is required to price the tranche.

In this paper, we present a generic methodology to compute first- and second-order sensitivities for multi-name credit derivatives. Our method is numerically stable and correct up to floating point error; in addition, it does not require the construction of the full portfolio loss distribution, thus it is able to achieve substantial reductions in computational times.

Our approach is based on the theory of algorithmic differentiation, see Griewank and Walther (2008), and follows on from work by Giles and Glasserman (2006) on deltas for the LIBOR market model, and Joshi and Yang (2010) for the Gamma matrix. In particular, we use the result that adjoint methods allow the computation of the gradient of a function with 4 times the number of floating point operations required to compute the function. We use the adaptation of this presented in Joshi and Yang (2010) that the Gamma matrix (i.e. the Hessian) can be computed in $AM + B$

Date: October 8, 2010.

1991 *Mathematics Subject Classification.* 91B24, 60G40, 60G44. JEL Classification: G13.

Key words and phrases. portfolio credit derivatives, copula modelling, semi-analytic method, algorithmic differentiation.

times the number of operations where M is the maximum number of state variables required to compute the function, and A, B are constants that only depend on the set of floating point operations allowed. These methods rely on the idea that given the Hessian of a function f, one can find the Hessian of $f \circ G$ in a simple fashion provided the function G is sufficiently elementary. We therefore simply have to divide the computation into a sequence of elementary functions.

We concentrate on equity tranches of CDOs in this paper. However, mezzanines can be handled as a difference of two equity tranches and senior tranche can be regarded as the complement. We also note that the extension to nth-to-default basket swap is straight-forward. Numerical results show that our method is both fast and accurate. In particular, we are able to calculate Hessian matrices for synthetic CDO tranches that have 125 underlying names in seconds.

The paper is organized as follows. We review the classical copula model and the semi-analytic method in section 2. In section 3, we briefly review credit derivatives pricing and, in particular, algorithms used to price synthetic CDO tranches as in Andersen et al (2003) and Laurent and Gregory (2005). We introduce relevant algorithmic differentiation results from Joshi and Yang (2010) in section 4. In section 5, we give detailed algorithms of computing synthetic CDO tranche Greeks. We present numerical testing results in section 6. Concluding remarks are given in section 7.

2. Copula Modelling and the Semi-analytic Method

We follow the Li model in this paper. We introduce necessary notations and briefly review the recursive algorithm used to construct the portfolio loss distribution as in Andersen et al. (2003).

2.1. **Notations.** We denote by τ_i the default time for an obligor i and let $\lambda_i(t)$ be the (deterministic) default intensity for obligor i, for $i = 1, \ldots, m$.

2.2. **Gaussian copula modelling of default times.** The (marginal) distribution function of the random variable (RV) τ_i (or the default probability for obligor i by time t) is given by

$$q_i(t) = \mathbb{P}\Big(\tau_i \leqslant t\Big) = 1 - \exp\bigg(-\int_0^t \lambda_i(s)\mathrm{d}s\bigg), \tag{2.1}$$

for $i = 1, \ldots, m$. These one-dimensional RVs are connected to each other by a multivariate Gaussian copula; that is,

$$F\Big(t_1, \ldots, t_m\Big) = \Phi_m\Big(\Phi^{-1}\big(q_1(t_1)\big), \ldots, \Phi^{-1}\big(q_m\big(t_m\big)\big); \rho\Big), \tag{2.2}$$

where F is the joint distribution function of the RVs τ_i, Φ is the standard Gaussian distribution function and Φ_m is the multivariate Gaussian distribution function with a specified correlation matrix ρ. Under this framework, the default times can be easily simulated from Gaussian RVs in a Monte Carlo simulation (for example, see Li (2000)).

2.3. **Semi-analytic approach.** We work in a single-factor Li model: let V and $\big\{\epsilon_i\big\}_{i=1}^m$ be independent standard Gaussian RVs. We introduce the following latent standard Gaussian RV

$$X_i = \alpha_i V + \sqrt{1 - \alpha_i^2}\epsilon_i \tag{2.3}$$

to model the default time of obligor i via the relationship

$$\mathbb{P}\Big(\tau_i \leqslant t\Big) = \mathbb{P}\Big(X_i \leqslant C_i(t)\Big), \tag{2.4}$$

where

$$C_i(t) = \Phi^{-1}\big(q_i(t)\big). \tag{2.5}$$

For details, see chapter 13 of O'Kane (2008).

2.3.1. *Conditional default probability.* We denote by $q_i(t|V)$ the probability that obligor i will default by time t conditional on the first factor V, then

$$q_i(t|V) = \mathbb{P}\Big(X_i \leqslant C_i(t)\Big|V\Big) = \Phi\left(\frac{\Phi^{-1}\big(q_i(t)\big) - \alpha_i V}{\sqrt{1-\alpha_i^2}}\right). \tag{2.6}$$

2.3.2. *Conditional portfolio loss distribution.* We denote by $f_k^{(i)}(t|V)$ the probability of there being k defaults in a portfolio with i obligors by time t conditional on the systemic factor V, for $k = 0, 1, \ldots, i$ and $i = 0, 1, \ldots, m$. Because the obligors' defaults are independent conditional on V, we introduce the following recursive algorithm to construct the (conditional) loss distribution:

$$f_k^{(i)}(t|V) = \begin{cases} f_0^{(i-1)}(t|V)\big(1-q_i(t|V)\big), & \text{if } k = 0; \\ f_k^{(i-1)}(t|V)\big(1-q_i(t|V)\big) + f_{k-1}^{(i-1)}(t|V)q_i(t|V), & \text{if } k = 1, \ldots, i-1; \\ f_{i-1}^{(i-1)}(t|V)q_i(t|V), & \text{if } k = i. \end{cases} \tag{2.7}$$

We assume that each obligor has the same constant recovery rate so that we do not refine the loss distribution through a loss unit as in Andersen et al (2003), but the extension would be straight-forward.

3. Synthetic CDO Tranche Pricing

In general, the price of a multi-name credit derivative can be regarded as a real-valued vector function, $\mathbf{P}$, of the time-dependent default intensities:

$$\text{Price} = \int_{\mathbb{R}} \mathbf{P}\big(\lambda_1, \ldots, \lambda_m\big|V\big)\varphi(v)\mathrm{d}v, \tag{3.1}$$

where λ_i are deterministic default intensities and φ is the standard Gaussian density function. In what follows, we will be interested in sensitivities of the price with respect to parallel shifts, $\{s_i\}$, of the default intensities. Thus, we define the following function $\mathbf{F} : \mathbb{R}^m \to \mathbb{R}$ with $\mathbf{F}(s_1, \ldots, s_m) = \mathbf{P}(\lambda_1 + s_1, \ldots, \lambda_m + s_m)$ and we rewrite the price as

$$\text{Price} = \int_{\mathbb{R}} \mathbf{F}\big(s_1, \ldots, s_m\big|V\big)\varphi(v)\mathrm{d}v. \tag{3.2}$$

3.1. **Notations.** K_a denotes the attachment point of the tranche and K_b denotes the detachment point. $T_{(a,b)} = m(K_b - K_a)$ is the tranche size (with unit notional value). The coupon payment dates are T_j, $j = 1, \ldots, n$. We denote the constant recovery rate for each obligor by R. Let $P_t(0) = e^{-rt}$, where r is the risk-free rate, be the current price of a zero coupon bond maturing at time t; if no confusion arise, we write $P_j(0) = P_{T_j}(0)$ for $j = 1, \ldots, n$.

3.2. **Tranche loss.** We denote by $L_{(a,b)}(t)$ the tranche loss at time t and

$$L_{(a,b)}(t) = \min\Big\{L(t), mK_b\Big\} - \min\Big\{L(t), mK_a\Big\}, \tag{3.3}$$

where $L(t) = (1-R)\sum_{i=1}^m \mathbb{I}_{\{\tau_i \leqslant t\}}$ is the portfolio loss at time t.

3.3. **Tranche pricing.** The underlyings of the synthetic CDO are credit default swaps (CDSs) written on different obligors. Thus, the price of the synthetic CDO tranche is characterized by the following two legs:

- The value of the premium leg is given by

$$V_{\text{prem}} = \sum_{j=1}^n \Delta_j P_j(0)\Big(T_{(a,b)} - L_{(a,b)}(T_j)\Big), \tag{3.4}$$

where $\Delta_j = T_j - T_{j-1}$.

- The value of the protection leg is given by

$$V_{\text{prot}} = \int_0^{T_n} P_t(0)\mathrm{d}L_{(a,b)}(t). \tag{3.5}$$

Integration by parts, see Laurent and Gregory (2005), gives

$$\begin{aligned} V_{\text{prot}} &= P_n(0)L_{(a,b)}(T_n) + \int_0^{T_n} rP_t(0)L_{(a,b)}(t)\mathrm{d}t \\ &\simeq P_n(0)L_{(a,b)}(T_n) + \frac{r}{2}\Big(\sum_{j=1}^{n-1}\big(\Delta_j + \Delta_{j+1}\big)P_j(0)L_{(a,b)}(T_j) + \Delta_n P_n(0)L_{(a,b)}(T_n)\Big), \end{aligned} \tag{3.6}$$

where we use the trapezoidal rule to approximate the definite integral.

3.3.1. *CDO tranche price and expected tranche loss.* Given a market spread, the (conditional) CDO tranche price is given by

$$\mathbf{F}\big(s_1,\ldots,s_m\big|V\big) = \mathbb{E}\Big(V_{\text{prot}} - \text{spread}\cdot V_{\text{prem}}\Big|V\Big). \tag{3.7}$$

It is clear that the key variables that we need to compute in pricing are the expected tranche losses $\mathbb{E}\big(L_{(a,b)}(t)\big|V\big)$, which is a simple function of the portfolio loss distribution at time t:

$$\mathbb{E}\Big(L_{(a,b)}(t)\Big|V\Big) = \sum_{k=0}^{m} g_{(a,b)}(k) f_k^{(m)}(t|V), \tag{3.8}$$

where

$$g_{(a,b)}(k) = \min\Big\{k(1-R), mK_b\Big\} - \min\Big\{k(1-R), mK_a\Big\}.$$

3.4. **Efficient equity tranche pricing.** However, it is not necessary to construct the full loss distribution when we price equity or mezzanine tranches. We first show how to do this for an equity tranche with detachment K_b. Define an integer by

$$C_b = \left\lfloor \frac{mK_b}{1-R} \right\rfloor. \tag{3.9}$$

The expected equity tranche loss at time t can be re-written as

$$\begin{aligned} \mathbb{E}\Big(L_{(0,b)}(t)\Big|V\Big) &= \sum_{k=0}^{C_b} k(1-R) f_k^{(m)}(t|V) + \sum_{k=C_b+1}^{m} T_{(0,b)} f_k^{(m)}(t|V) \\ &= T_{(0,b)} - \sum_{k=0}^{C_b} f_k^{(m)}(t|V)\Big(T_{(0,b)} - k(1-R)\Big). \end{aligned} \tag{3.10}$$

That is, we only need to construct the first $C_b + 1$ probability mass functions for the portfolio loss RV. Thus we discard all higher terms when performing the recursion (2.7).

4. Adjoint algorithmic Differentiation and Hessian Updating

Efficient Greek computation techniques have been studied extensively in recent years. Joshi and Yang (2010) developed algorithmic Hessian methods to compute second-order Greeks. We briefly review their results here.

4.1. **Algorithmic Hessian methods.** The main results of Joshi and Yang (2010) are dedicated to computing the Hessian matrix of a twice-differentiable function

$$f : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R},$$
$$f : (\underline{x}, \alpha) \mapsto y,$$

with respect to the $\underline{x}$ variables. We shall regard the variables $\underline{x}$ as the state variables of our problem, and the variables α as auxiliary variables.

The key idea of algorithmic Hessian methods is to decompose f into a sequence of elementary operations such that the Hessian matrix can be updated (or overwritten) in a fast and efficient backwards fashion.

4.2. **Elementary operation.** Let π_j denote the projection onto the jth coordinate thus

$$\pi_j(\underline{x}, \alpha) = x_j.$$

Definition 4.1. *We shall say that $\mathbf{F}_K$ defines an elementary operation if there exists i such that*

$$\pi_j \mathbf{F}_K = x_j, \quad \text{for } j \neq i,$$

and $h = \pi_i \mathbf{F}_K$ maps to one of the following values for some j, r, s, t and u:

1. *addition: $x_i + \alpha_r x_j$;*
2. *multiplication: $(x_i + \alpha_r)x_j - \alpha_r$;*
3. *shifting: $l(x_j)$ with $l : \mathbb{R} \to \mathbb{R}$ a twice-differentiable function.*

Note that we only list operations that are relevant to this paper, for a detailed list see Joshi and Yang (2010).

4.3. **Gradient and Hessian updating.** We have two mappings: $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ and $F : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}^m$. Suppose that the map g has gradient ∇_g and Hessian $\underline{\mathrm{H}}_g$. In what follows, we give recursive algorithms on how to overwrite ∇_g and $\underline{\mathrm{H}}_g$ to obtain the gradient vector and the Hessian matrix of the composite mapping $g \circ F$, $\nabla_{g\circ F}$ and $\underline{\mathrm{H}}_{g\circ F}$, respectively.

4.3.1. *Addition.* The gradient vector for the addition operation is updated as follows:

$$\begin{cases} \nabla_{g\circ F}(r) = \nabla_g(r), & r \neq j; \\ \nabla_{g\circ F}(j) = \nabla_g(j) + \alpha_r \nabla_g(i). \end{cases} \tag{4.1}$$

The Hessian matrix is updated as follows:

$$\begin{cases} \underline{\mathrm{H}}_{g\circ F}(r,s) = \underline{\mathrm{H}}_g(r,s), & r,s \neq j; \\ \underline{\mathrm{H}}_{g\circ F}(r,j) = \underline{\mathrm{H}}_g(r,j) + \alpha_r \underline{\mathrm{H}}_g(r,i), & r \neq j; \\ \underline{\mathrm{H}}_{g\circ F}(j,j) = \alpha_r^2 \underline{\mathrm{H}}_g(i,i) + 2\alpha_r \underline{\mathrm{H}}_g(i,j) + \underline{\mathrm{H}}_g(j,j). \end{cases} \tag{4.2}$$

4.3.2. *Multiplication.* The gradient vector for the multiplication operation is updated as follows:

$$\begin{cases} \nabla_{g\circ F}(r) = \nabla_g(r), & r \neq i,j; \\ \nabla_{g\circ F}(i) = x_j \nabla_g(i); \\ \nabla_{g\circ F}(j) = (x_i + \alpha_r)\nabla_g(j). \end{cases} \tag{4.3}$$

The Hessian matrix is updated as follows:

$$\begin{cases} \underline{\mathrm{H}}_{g\circ F}(r,s) = \underline{\mathrm{H}}_g(r,s), \quad r,s \neq i,j; \\ \underline{\mathrm{H}}_{g\circ F}(r,i) = x_j \underline{\mathrm{H}}_g(r,i), \quad r \neq i,j; \\ \underline{\mathrm{H}}_{g\circ F}(r,j) = (x_i+\alpha_r)\underline{\mathrm{H}}_g(r,j), \quad r \neq i,j; \\ \underline{\mathrm{H}}_{g\circ F}(i,i) = x_j^2 \underline{\mathrm{H}}_g(i,i); \\ \underline{\mathrm{H}}_{g\circ F}(i,j) = x_j \underline{\mathrm{H}}_g(i,j) + \big[x_j(x_i+\alpha_r) - \alpha_r\big]\underline{\mathrm{H}}_g(i,i) + \nabla_g(i); \\ \underline{\mathrm{H}}_{g\circ F}(j,j) = \underline{\mathrm{H}}_g(j,j) + 2(x_i+\alpha_i)\underline{\mathrm{H}}_g(i,j) + (x_i+\alpha_i)^2 \underline{\mathrm{H}}_g(i,i). \end{cases} \tag{4.4}$$

4.3.3. *Shifting operation.* The gradient vector for the shifting operation is updated as follows:

$$\begin{cases} \nabla_{g\circ F}(r) = \nabla_g(r), \quad r \neq i,j; \\ \nabla_{g\circ F}(i) = 0; \\ \nabla_{g\circ F}(j) = \nabla_g(j) + l'(x_j)\nabla_g(i). \end{cases} \tag{4.5}$$

The Hessian matrix is updated as follows:

$$\begin{cases} \underline{\mathrm{H}}_{g\circ F}(r,s) = \underline{\mathrm{H}}_g(r,s), \quad r,s \neq i,j; \\ \underline{\mathrm{H}}_{g\circ F}(r,i) = 0, \quad \forall r; \\ \underline{\mathrm{H}}_{g\circ F}(r,j) = \underline{\mathrm{H}}_g(r,j) + l'(x_j)\underline{\mathrm{H}}_g(r,i), \quad r \neq i,j; \\ \underline{\mathrm{H}}_{g\circ F}(j,j) = \underline{\mathrm{H}}_g(j,j) + 2l'(x_j)\underline{\mathrm{H}}_g(i,j) + l'(x_j)^2 \underline{\mathrm{H}}_g(i,i) + l''(x_j)\nabla_g(i). \end{cases} \tag{4.6}$$

5. Greek Estimations for Synthetic CDO Tranches

In this paper, we will study efficient algorithms to compute first- and second-order Greeks of synthetic CDO tranches. However, our techniques can be easily applied to nth-to-default basket swaps.

For simplicity, we shall consider computing the conditional sensitivities of an equity tranche with detachment point K_b given the common single factor V as in (2.3):

$$\frac{\partial \mathbf{F}}{\partial s_i}\big(\underline{0}\big|V\big) \quad \text{and} \quad \frac{\partial^2 \mathbf{F}}{\partial s_i \partial s_j}\big(\underline{0}\big|V\big), \tag{5.1}$$

where $\mathbf{F}$ is defined in (3.7) and s_i are parallel shifts in $\lambda_i(t)$ for all t. We use the Gauss-Hermite quadrature rule to compute all the unconditional sensitivities.

5.1. Elementary operations in synthetic CDO pricing.

The state variables are

$$\underline{x} = \begin{pmatrix} s_1 & \cdots & s_m & f_0 & \cdots & f_{C_b} & q & l & v \end{pmatrix}^\top, \tag{5.2}$$

where the first m entries are the parallel shifts in default intensities, the next C_b+1 entries represent the portfolio loss distribution at each coupon dates with C_b given by (3.9), and the last three entries are auxiliary variables used in the pricing algorithm.

We decompose the mapping $\mathbf{F}$ as in (3.7) into the following set of vector mappings:

$$\mathbf{F}\big(\underline{x}\big|V\big) = \mathcal{P} \circ \mathcal{G}_n \circ \mathcal{G}_{n-1} \circ \cdots \circ \mathcal{G}_1\big(\underline{x}\big|V\big), \tag{5.3}$$

where each vector function, $\mathcal{G}_j$, updates the tranche value using the expected tranche loss $\mathbb{E}L_{(0,b)}(T_j)$ and $\mathcal{P}$ is the project onto the last entry of $\underline{x}$. Each of the mappings $\mathcal{G}_j$ can be further decomposed into the following set of sub-vector mappings:

$$\mathcal{G}_j = \mathcal{V}_j \circ \mathcal{L}_j \circ \mathcal{H}_m^j \circ \mathcal{H}_{m-1}^j \circ \cdots \circ \mathcal{H}_1^j, \tag{5.4}$$

where the vector functions $\mathcal{V}_j$ updates the tranche value, $\mathcal{L}_j$ computes the expected tranche loss and $\mathcal{H}_i^j$ are given by

$$\mathcal{H}_i^j = \mathcal{D}_i \circ \mathcal{D}_{i-1} \circ \cdots \circ \mathcal{D}_0 \circ \mathcal{Q}_i^j \tag{5.5}$$

with $\mathcal{D}_k$ updating the conditional portfolio loss distribution and $\mathcal{Q}_i^j$ updating the conditional default probability.

Having identified the elementary operations[1] that are used in pricing the equity tranche of a synthetic CDO, we can use the efficient algorithms in section 4 to update gradient vectors and Hessian matrices recursively.

5.1.1. *Sub-sub-mapping* $\mathcal{Q}_i^j(\underline{x}) = \underline{y}$. This mapping performs the following operation on q:

$$y_{m+C_b+2} = l(x_j) = \Phi\bigg(\frac{\Phi^{-1}(1 - e^{-x_j T_j} p_i(T_j)) - \alpha_i V}{\sqrt{1-\alpha_i^2}}\bigg), \tag{5.6}$$

where $p_i(T_j) = 1 - q_i(T_j)$ with $q_i(T_j)$ given by (2.1).

Note that this operation belongs to the *shifting elementary operations*, and the algorithms to update the gradient vector and the Hessian matrix are given in section 4. We only give the first- and second-order partial derivatives here:

$$l'(x_j) = \varphi\Big(h_i\big(T_j|V\big)\Big) \cdot C_i\big(T_j\big) \cdot T_j, \tag{5.7}$$

$$l''(x_j) = \varphi\Big(h_i\big(T_j|V\big)\Big) \cdot \Big(D_i\big(T_j\big) - C_i\big(T_j\big)^2 \cdot h_i\big(T_j|V\big)\Big) \cdot T_j^2, \tag{5.8}$$

where

$$h_i\big(T_j|V\big) = \frac{\Phi^{-1}\big(1 - \widehat{p}_i(T_j)\big) - \alpha_i V}{\sqrt{1-\alpha_i^2}}, \tag{5.9}$$

$$\widehat{p}_i\big(T_j\big) = e^{-s_j T_j} p_i\big(T_j\big), \tag{5.10}$$

$$C_i\big(T_j\big) = \sqrt{\frac{2\pi}{1-\alpha_i^2}} \cdot e^{\Phi^{-1}\left(1-\widehat{p}_i(T_j)\right)^2/2} \cdot \widehat{p}_i(T_j), \tag{5.11}$$

$$D_i\big(T_j\big) = C_i\big(T_j\big)\bigg(\sqrt{2\pi} \cdot \Phi^{-1}\Big(1 - \widehat{p}_i\big(T_j\big)\Big) \cdot e^{\Phi^{-1}\left(1-\widehat{p}_i(T_j)\right)^2/2} \cdot \widehat{p}_i\big(T_j\big) - 1\bigg). \tag{5.12}$$

5.1.2. *Sub-sub-mapping* $\mathcal{D}_k(\underline{x}) = \underline{y}$. This mapping updates the $\{f_k\}$ via

$$y_{m+k} = \begin{cases} x_m\big(1 - x_{m+C_b+2}\big), & \text{if } k = 0; \\ x_{m+i} \cdot x_{m+C_b+2}, & \text{if } k = i; \\ x_{m+k}\big(1 - x_{m+C_b+2}\big) + x_{m+k-1} \cdot x_{m+C_b+2}, & \text{else.} \end{cases} \tag{5.13}$$

The first case is a projection then followed by a multiplication operation. The second case is simply a multiplication. The remaining cases are the combination of the first two cases.

5.1.3. *Sub-sub-mapping* $\mathcal{L}_j(\underline{x}) = \underline{y}$. The (conditional) expected tranche loss is updated via

$$y_{m+C_b+3} = T_{(0,b)} - \sum_{k=0}^{C_b} x_{m+k}\big(T_{(0,b)} - k(1-R)\big). \tag{5.14}$$

The first operation puts $y_{m+C_b+3} = T_{(0,b)}(1 - x_m)$, which is the shifting operation, and each subsequent one adds one more term.

[1]The elementary operations $\mathcal{V}_j$, $\mathcal{L}_j$, $\mathcal{D}_k$ and $\mathcal{Q}_i^j$ are $\mathbb{R}^{m+C_b+4}$ to $\mathbb{R}^{m+C_b+4}$ vector functions.

5.1.4. *Sub-sub-mapping* $\mathcal{V}_j(\underline{x}) = \underline{y}$. We update the equity tranche value using (3.7)

$$y_{m+C_b+4} = \begin{cases} \big(r(T_1+\Delta_2)/2 + \text{spread}T_1\big)P_1(0)x_{m+C_b+3}, & \text{if } j=1; \\ x_{m+C_b+4} + \big((r/2+\text{spread})\Delta_n + 1\big)P_n(0)x_{m+C_b+3} - \text{spread}\cdot \text{A}_n, & \text{if } j=n; \\ x_{m+C_b+4} + \big(r(\Delta_j+\Delta_{j+1})/2 + \text{spread}\Delta_j\big)P_j(0)x_{m+C_b+3}, & \text{else,} \end{cases} \tag{5.15}$$

where $A_n = \sum_{j=1}^{n} \Delta_j P_j(0)$. These operations are similar to those in $\mathcal{L}_j$: the first one is a shifting operation and the rest belong to the addition operation.

5.2. **Mezzanine tranche sensitivities.** We do not develop algorithms for estimating sensitivities of the mezzanine tranche since the mezzanine tranche can be treated as a difference of two equity tranches. This makes particular sense in the base correlation framework where only equity tranches with different detachment points are marked to the market.

5.3. n**th-to-default basket swap sensitivities.** The elementary operations in (5.4) for nth-to-default basket swaps are exactly the same as those for synthetic CDO tranches except the mapping $\mathcal{V}_j$, which is again a simple function of the expected tranche losses and portfolio loss distributions at different coupon dates. Hence, extension to computing sensitivities of default basket swaps is trivial.

6. Numerical Testing

6.1. **Market data.** We consider pricing an equity tranche of a synthetic CDO with detachment points 3%, 7%, 10% and 15% and varying number of underlying names. The maturity of the synthetic CDO is 5 years with quarterly coupon payments so that $\Delta_j = 0.25$, $j = 1, \ldots, 20$. The (deterministic) default intensities of the underlying names are defined by

$$\lambda_i(t) = \begin{cases} \lambda_1^i, & t \in [0, T_1); \\ \lambda_j^i, & t \in [T_{j-1}, T_j), \quad j = 2, \ldots, 20, \end{cases}$$

for $i = 1, \ldots, m$. For testing purposes, we set $\lambda_j^i = 0.02$ for all i and j. The constant recovery is set to 40%. The market spread for equity tranches is set to 6%. We assume that the off-diagonal entries of the correlation matrix are constant and set $\alpha_i = \sqrt{0.5}$ for all i. We use a 30-point Gauss-Hermite quadrature rule to estimate the price (3.2) and the Greeks (5.1).

6.2. **Results and discussion.** We compute the price, all the the first-order and second-order sensitivities to parallel shifts using single-threaded C++ code on a notebook computer[2]. Although we carry out numerical tests on CDOs with different number of obligors, we focus the case where we have 125 underlying names in the following discussion.

6.2.1. *Accuracy tests.* We compute sensitivities using two methods: the finite-differencing[3] method and the adjoint method. Table 6.1 lists the numerical values for the following two sums:

$$\sum_{i=1}^{125} \frac{\partial \text{Price}}{\partial s_i} \quad \text{and} \quad \sum_{i=1}^{125} \frac{\partial^2 \text{Price}}{\partial s_i^2},$$

where Price is the price of an equity tranche of a synthetic CDO with 125 names. The main observation is that the adjoint numbers are extremely close to the finite-differencing numbers.

[2]Intel Core i5 CPU 2.40GHz with 4GB RAM.

[3]The bump size in all finite-differencing calculations is set to 1 basis point.

	Finite-differencing		Adjoint method	
detachment point	Delta sum	Gamma sum	Delta sum	Gamma sum
3%	59.2667	1887.6486	59.2668	1887.6378
7%	123.6344	2774.3058	123.6346	2774.3005
10%	177.1553	2662.4165	177.1556	2662.4113
15%	225.7445	2419.2850	225.7449	2419.2779

TABLE 6.1. Numerical values for the sums of all the Deltas and all the diagonal entries of the Hessian matrix for different equity tranches of a synthetic CDO with 125 names, computed using the finite-differencing method and the adjoint method.

m	5	10	25	50	75	100	125
pricing	0.0003	0.0005	0.0015	0.0038	0.0063	0.0112	0.0144
delta	0.0002	0.0004	0.0013	0.0038	0.0070	0.0144	0.0191
gamma	0.0008	0.0021	0.0203	0.1323	0.3807	1.0012	1.8564
ratio/m	0.57	0.39	0.53	0.69	0.80	0.89	1.03

TABLE 6.2. Computational times for computing the price, first-order sensitivities and second-order sensitivities for a $(0\%, 3\%)$ equity tranche with different number of underlying names. Note that ratio/m is the ratio of the time taken to compute Gammas to the time taken to price the tranche divided by the corresponding number of underlying names.

6.2.2. *Timing tests.* We list the computational times (in seconds) of pricing and Greek estimation for equity tranches with different detachment points in tables 6.2 to 6.5:

- We first note that the pricing algorithm is extremely fast. It only takes 0.02 seconds to price the 3% equity tranche and 0.05 seconds to price the 15% tranche.
- To compute all the 125 finite-difference Deltas, the computational time for doing so will be 125 times of the original pricing time (if we use forward difference estimates). However, it only takes up to 1.49 times (the 15% equity tranche) of the original pricing time if we use algorithmic differentiation.
- We have $7,875$ Gammas to estimate. If we use central difference estimates, it will take $31,250$ times[4] of the original pricing time to achieve this. Surprisingly, it only takes up to 195 times to estimate all the Gammas with algorithmic Hessian methods. In particular, it takes less than 2 seconds to estimate all the Gammas for a 3% equity tranche.
- Finally we notice that the ratios are increasing in m. The reason might be that the higher the detachment points are, the more portfolio loss distributions are needed. Hence, we have more state variables so that it takes more time to update Hessian matrices.

7. Conclusion

We have successfully developed generic methods to compute sensitivities of multi-name credit derivatives in a single-factor Gaussian copula model. Our methods largely follow from the ideas used in Joshi and Yang (2010): we decompose the pricing mapping into elementary vector operations, whose gradient vector and Hessian matrix can be updated recursively using efficient algorithms.

These algorithms result in substantial reductions in computational times. Compared with traditional finite-difference methods, our method is able to achieve hundreds of times speed-ups.

[4] 2 extra pricings for diagonal Gammas and 4 extra pricings for cross Gammas.

m	5	10	25	50	75	100	125
pricing	0.0003	0.0007	0.0018	0.0054	0.0101	0.0170	0.0257
delta	0.0002	0.0005	0.0016	0.0074	0.0135	0.0230	0.0342
gamma	0.0008	0.0044	0.0320	0.2702	0.9056	2.2199	4.1446
ratio/m	0.56	0.68	0.70	1.01	1.19	1.31	1.29

TABLE 6.3. Computational times for computing the price, first-order sensitivities and second-order sensitivities for a $(0\%, 7\%)$ equity tranche with different number of underlying names. Note that ratio/m is the ratio of the time taken to compute Gammas to the time taken to price the tranche divided by the corresponding number of underlying names.

m	5	10	25	50	75	100	125
pricing	0.0003	0.0007	0.0023	0.0066	0.0132	0.0216	0.0325
delta	0.0002	0.0005	0.0022	0.0085	0.0174	0.0297	0.0468
gamma	0.0008	0.0045	0.0549	0.4099	1.3117	3.0015	5.8680
ratio/m	0.55	0.67	0.96	1.23	1.32	1.39	1.44

TABLE 6.4. Computational times for computing the price, first-order sensitivities and second-order sensitivities for a $(0\%, 10\%)$ equity tranche with different number of underlying names. Note that ratio/m is the ratio of the time taken to compute Gammas to the time taken to price the tranche divided by the corresponding number of underlying names.

m	5	10	25	50	75	100	125
pricing	0.0003	0.0007	0.0030	0.0083	0.0182	0.0311	0.0468
delta	0.0003	0.0006	0.0031	0.0120	0.0230	0.0487	0.0697
gamma	0.0017	0.0069	0.0810	0.6336	2.0266	4.7892	9.1302
ratio/m	0.95	0.93	1.08	1.52	1.49	1.54	1.56

TABLE 6.5. Computational times for computing the price, first-order sensitivities and second-order sensitivities for a $(0\%, 15\%)$ equity tranche with different number of underlying names. Note that ratio/m is the ratio of the time taken to compute Gammas to the time taken to price the tranche divided by the corresponding number of underlying names.

REFERENCES

[1] L. Andersen, J. Sidenius and S. Basu. (2003). All your hedges in one basket. Risk (November), 67–72.
[2] M. Giles and P. Glasserman. (2006). Smoking adjoints: fast Monte Carlo Greeks. Risk, January issue, 92–96.
[3] A. Griewank and B. Walter. (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* SIAM.
[4] M. Joshi and C. Yang. (2010). Algorithmic Hessians and the Fast Computation of Cross-Gamma Risk. SSRN working paper. Retrieved from: http://ssrn.com/abstract=1626547.
[5] P. Laurent and J. Gregory. (2005). Basket default swaps, CDOs, and factor copulas. Journal of Risk, **7**(4), 103–122.
[6] D. Li. (2000). On default correlation: a copula function approach. Journal of Fixed Income, **9**, 43–54.
[7] D. O'Kane. (2008). *Modelling Single-name and Multi-name Credit Derivatives.* Wiley.

CENTRE FOR ACTUARIAL STUDIES, DEPARTMENT OF ECONOMICS, UNIVERSITY OF MELBOURNE, VICTORIA 3010, AUSTRALIA
E-mail address: mark@markjoshi.com
E-mail address: c.yang7@pgrad.unimelb.edu.au