LSM Reloaded

Differentiate xVA on your iPad Mini

Brian Huge
brian.huge@danskebank.dk

Antoine Savine
antoine@asavine.com

June 16, 2017

Abstract

This document reviews the so called least square methodology (LSM) and its application for the valuation and risk of callable exotics and regulatory value adjustments (xVA). We derive valuation algorithms for xVA, both with or without collateral, that are particularly accurate, efficient and practical. These algorithms are based on a reformulation of xVA, designed by Jesper Andreasen and implemented in Danske Bank's award winning systems, that hasn't been previously published in full. We then investigate the matter of risk sensitivities, in the context of Algorithmic Automated Differentiation (AAD). A rather recent addition to the financial mathematics toolbox, AAD is presently generally acknowledged as a vastly superior alternative to the classical estimation of risk sensitivities through finite differences, and the only practical means for the calculation of the large number of sensitivities in the context of xVA. The theory and implementation of AAD, the related check-pointing techniques, and their application to Monte-Carlo simulations are explained in numerous textbooks and articles, including Giles and Glasserman's pioneering Smoking Adjoints. We expose an extension to LSM, and, in particular, we derive an original algorithm that resolves the matters of memory consumption and efficiency in differentiating simulations together with the LSM step.

1 Introduction

LSM (which conveniently stands for both the Longstaff-Schwartz Model of 2001, and the Least-Square Method as it is commonly called, see [23] and [13]) is presently accepted as the best practice for the valuation of callable transactions and xVA with Monte-Carlo simulations, without the need for nested simulations or analytic approximations. As we remind in the first part of Section 2, LSM produces regression proxies for future values as functions of the simulated state variables, essentially turning callable trades and xVA into a purely path-dependent problem that can be subsequently resolved with standard Monte-Carlo methods.

However, that classical use of LSM proxies heavily relies on their accuracy and stability, which may take considerable development effort and excessive CPU power to achieve. This is problematic especially in the context of risk sensitivities, even with AAD, where excessive memory may be an

additional issue[1].

One way around this problem is to restrict the use of LSM proxies to positivity indicators. We call that POI (Proxies Only in Indicators). This approach is not exactly new[2]. But we will demonstrate that a systematic application of POI, in particular to xVA, following the footpaths of Andreasen in [2] not only considerably reduces dependency on proxy accuracy, but also allows to redefine xVA in a computationally friendly way, and often permits to skip the differentiation of the regressions for the production of risk sensitivities.

We explore the application of LSM with POI to xVA and show how to rewrite the xVA problem to produce a valuation algorithm that is particularly efficient, accurate and practical. We also show how this approach can be extended to collateralized xVA with the help of so-called Banching Monte-Carlo simulations to overcome some technical problems raised by the presence of a collateral.

In conjunction with AAD and multi-threaded simulations[3], this implementation of xVA was demonstrated to compute values and risk sensitivities with formidable performance without loss of accuracy[4] and earned Danske Bank the prestigious In-House System of the Year 2015 Risk Award.

All of this is investigated and explained in section 2, which also includes a refresher on classical LSM. Section 3 focuses on the production of risk sensitivities in the context of AAD. In that section, we show that in many cases, there is no need to differentiate the production of proxies. The standard application of path-wise AAD over MC simulations with fixed proxies, also called *cash-flow differentiation* produces the correct sensitivities, so it is not necessary, or advisable, to differentiate regressions or pre-simulations. This also means that differentials of simulations that involve LSM are no different than standard Monte-Carlo differences and essentially come for free.

Finally, for those cases where the differentiation of the regression is necessary or desirable, our section 4 introduces an implementation for this differentiation, one we claim offers maximum performance[5]. In particular, our algorithm makes use of check-pointing techniques[6] to avoid the expensive computation and application of Jakobians, and conducts the differentiation of LSM pre-simulations *path-wise*, which minimizes memory consumption, maximizes cache efficiency, and makes it possible to conduct this differentiation in parallel. Our efforts to efficiently differentiate LSM follow those of [8], [10] and [12], while offering a different approach and implementation. In particular, the key idea of check-pointing LSM so as to perform the differentiation of pre-simulations path-wise is, to our knowledge, without precedent.

In summary, the two main contributions of this paper are:

A reformulation of xVA We clearly derive and explain an approach due to Andreasen that reformulates xVA in a way that is particularly friendly to an efficient, accurate and practical computation. Further, we show how to extend this approach to the case of a *collateralized* netting set, which is non-trivial and resolved with the help of *branching simulations*. That whole

[1] We quickly remind the working principles of AAD in section 4. For details, we refer to textbooks such as [25], or founding articles such as the award winning [17]. For an overview of a basic implementation, we refer to [27], and for a more in-depth exposition, including some material in this document and the application to RWA and KVA, we refer to [15].

[2] We heard it mentioned, among others, by Bruno Dupire and Peter Jaeckel in the early 2000s and in [7]. Andreasen's iPad Mini series [2], [3], [4], [5], [6] took it to a whole new level with a *systematic* approach for xVA.

[3] As well as the scripting of cash flows, including compression and decoration, see [2] and [9] for details.

[4] Although actual implementation on iOS remains untested, full risks of CVA for large netting sets were computed in less than a minute locally on a MacBook Pro in live demonstrations at Global Derivatives in 2014, 2015 and 2016 by Flyger, Huge and Savine of Danske Bank.

[5] As Jesper Andreasen would put it, "Differentiate LSM on an iPad Mini."

[6] Details in AAD material.

approach was designed by Andreasen and Huge for Danske Bank, where it was implemented by Flyger and Savine. It has not been previously published in full, although the approach was presented at Global Derivatives in 2015 by Andreasen, see [1].

A check-pointing algorithm through LSM We resolve memory and efficiency problems arising in the differentiation of LSM. We show that LSM can be differentiated in 3 successive check-pointed steps, where the first and the third steps are performed path-wise, which limits memory consumption to one path and begs for parallel processing. The second step is analytical. The systematic use of *check-pointing* avoids the costly production and multiplication of Jakobians. The whole algorithm is original.

In addition, we show how the so-called SVD regression (Numerical Recipes, chapter 15.4) brings stability, and also performance, to LSM, where it should be systematically used in place of standard regression. We leave the implementation of the SVD to Numerical Recipes, but we clearly show why and how it is implemented for LSM. We also discuss its differentiation and derive check-pointed differentials analytically using adjoint calculus.

2 Effective LSM

2.1 Classical LSM

Callable instruments are best valued with finite difference and other lattice based methods. However, when the underlying transaction features floored or capped coupons or any form of path dependency such as averaging or a global barrier, or when the dimensionality of the model is too high to accommodate a lattice, like multi-factor and/or non-Markov interest rate models, such as Heath-Jarrow-Mortons [19] or Brace-Gatarek-Musiela [11] or extensions of these models, then lattices are unpractical and we fall back on to Monte-Carlo simulations.

The trouble is that contrarily to lattices, Monte-Carlo simulates future paths only for the state variables of the model, not the present values (PV) of the products. It is only by averaging payoffs across completed simulations that todays PVs are produced. Future PVs are not available *within the simulations*, hence, simulations cannot deal with early termination or other forms of exercise. The party that holds the right to exercise does so when the *continuation value* (the PV of the transaction *immediately* after the exercise date if it is not exercised there) is greater than the exercise value (typically 0 or some rebate or intrinsic value[7]). The PV *immediately* before the exercise date is therefore the maximum of the continuation and the exercise values, hence, in order to proceed, we need some way of estimating continuation values on exercise dates within the simulations. One could obtain these with nested simulations (Monte-Carlo simulations *within each simulation*), a terribly inefficient brute force solution that is not to be considered for a serious implementation. The financial community had been trying to resolve that matter with varying degrees of success until [23] was published and quickly adopted as best practice.

LSMs simple idea is to estimate the unavailable future PVs with *regression proxies*. LSM proxies are functions of the state variables being simulated, these functions being constructed through regressions. Because proxies are functions of the state variables, they can be computed within the simulations, solving the problem of the unavailability of future PVs.

[7] In this paper, and without loss of generality, we always consider 0 rebate on exercise. Hence, exercise occurs when the continuation value is negative, and continuation occurs when it is positive.

We remind the LSM methodology in what follows. Note that it is model and instrument agnostic and is best implemented independently of any particular model or instrument. This means that LSM is implemented once and for all, and may be used with any combination of models (that feed LSM with state variables on event dates and regression variables on exercise dates) and products (that compute pathwise PVs given the paths for the state variables on the event dates).

A simulation model always simulates the evolution of a number of *state variables* on all future *event dates* (dates where the transaction has a fixing or an exercise). In Black-Scholes or a local volatility model *a la Dupire*, the unique state variable is the underlying spot price. In a Markov interest rate model *a la Cheyette*, state variables would be the limited number of Markov variables out of which all interest rates may be deduced deterministically on event dates. In a non-Markov interest rate model like Heath-Jarrow-Morton or Brace-Gatarek-Musiela, the state variables are the entire collection of forward rates that constitute the underlying yield curve. With Stochastic Volatility extensions of these models, the instantaneous volatility (when Markov as is standard) is an additional state variable. In all cases, the market variables relevant for the path-wise computation of payoffs at event dates are known, deterministic functions of the state variables at that date[8]. We note I the number of state variables being simulated and $S_t = (S_{ti})_{1 \leq i \leq I}$ the values of these state variables at time t.

2.1.1 One-time callable instruments

We start with instruments that can be called once at some exercise date T^{ex} in the future, like European swaptions[9]. The *underlying* transaction pays the cash-flows $(CF_k)_{1 \leq k \leq K}$ on dates $(T_k)_{1 \leq k \leq K} > T^{ex}$.

In all that follows, we ignore discounting/numeraire considerations in order to simplify the notations. As always, values are expectations and future values are conditional expectations, *implicitly* under a martingale measure associated with a numeraire, and *implicitly* all cash-flows are discounted to the value date with that numeraire. We however ignore that in our notations, such that the future values of the underlying transaction at the date T^{ex} is simply denoted:

$$V_{T^{ex}} = E_{T^{ex}} \left[\sum_{T_k > T^{ex}} CF_k \right]$$

This is also the value of the callable transaction immediately after the exercise date, provided it is not terminated there. For every simulation number $m \in \{1, \ldots, M\}$, the model provides the values of its state variables $S^m_{T^{ex}}$ (a vector of size I) at the exercise date, from which we calculate a vector (of size $J+1$ with indexing starting at 0) of *regression* variables $X^m_{T^{ex}}$ where $X^m_{T^{ex}0} = 1, X^m_{T^{ex}j} = f_j\left(S^m_{T^{ex}}\right)$ with the convention $f_0 \equiv 1$.

The regression variables may be the state variables themselves (plus the unit variable) or some functions of them. For instance, in the case of a standard European swaption, a natural regression variable is the PV of the underlying swap on the exercise date, something any interest rate model should provide as a function of its state variables. For a standard Bermuda, it is wise to use the swap rate to the next exercise date as a second regression variable, since the exercise for this date can be seen as the option to purchase the Bermuda starting at the next exercise date, for the PV

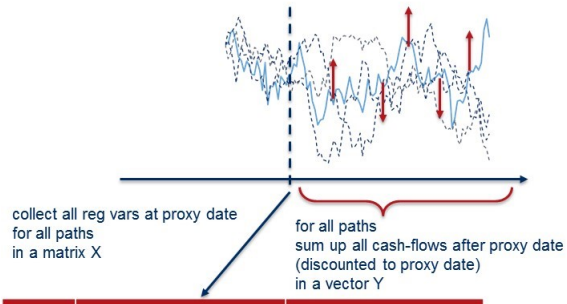
[8] Or even the state variables themselves, in which case the function is simply the identity.

[9] European swaptions typically dont need LSM, the PV of the underlying swap being a known function of the state variables. However, this may not be the case if the coupons of the underlying swap are capped, floored or subject to some path dependency such as averaging or a barrier. Besides, this is an exercise we need to go through as a warm-up.

of the cash-flows to the next exercise date. With stochastic volatility, the volatility should be used as a regression variable too for a Bermuda or a one-time callable trade with some optionality in the coupons such as caps or floors.

We may also perform a quadratic regression by choosing not only the state variables, but also their squares and pair wise products as additional regression variables. In the same spirit, we can make the regressions polynomial or use any basis functions of the state variables.

Further, within the same simulation number m, we run the simulation up to the final date T_K of the last cash-flow payment and produce the PV at the exercise date, along that path, of all the subsequent payments. We denote that PV $Y^m = \sum_{T_k > T^{ex}} CF_k^m$.



path	X matrix of reg vars	Y vector of PVs
1	$1 \quad X_1^1 \quad \dots \quad X_J^1$	Y^1
2	$1 \quad X_1^2 \quad \dots \quad X_J^2$	Y^2
...	... ...	...
M	$1 \quad X_1^M \quad \dots \quad X_J^M$	Y^M

regression: $\beta = (X^T X)^{-1}(X^T Y)$

proxy: $\tilde{V}_T = \beta \cdot X_T$

true PV: $V_T = E_T\left[\sum_{T_k > T} CF_k\right]$ (ignoring discount)

if reg vars are well chosen:

$$V_T = E_T\left[\sum_{T_k > T} CF_k\right] \approx E\left[\sum_{T_k > T} CF_k | X_T\right] \approx \beta \cdot X_T = \tilde{V}_T$$

Having performed these steps for all simulations, we collect a matrix X of size $M \times (J+1)$ of all the regression variables within all the simulations. The first column of X is $\mathbf{1}$, the unit regression variable in all scenarios. We also collect a vector Y of size M of the corresponding PVs of all subsequent cash-flows along the same paths.

Our goal is to estimate the value $V_{T^{ex}} = E_{T^{ex}}\left[\sum_{T_k > T^{ex}} CF_k^m\right] = E_{T^{ex}}\left[Y^m\right]$ of the future cash-flows at T^{ex} within the simulation number m. Because the dynamics of the state variables is Markov, we know that it is a function of the state variables $V_{T^{ex}} = g\left(S_{T^{ex}}\right)$, although not an explicit one[10]. LSM makes it explicit by performing a regression of Y over X, such that we have a *regression proxy* for the future value:

$$\widetilde{V}_{T^{ex}}^m = \sum_{j=0}^{J} \beta_j^{ex} X_{T^{ex}j}^m = \beta^{ex} \cdot X_{T^{ex}}^m$$

path-wise pre-simulations The LSM step's only purpose is to compute the regression coefficients. It is conducted in 2 stages:

1. Pre-simulations generate a number of Monte-Carlo paths. For each path, we collect the value

[10] Unless the cash-flows themselves carry path-dependency, like averaging or a barrier, a case that we deal with later in this subsection.

of regression variables at the exercise date, as a row in the matrix X. We also compute the PV to the exercise date of all subsequent cash-flows, and store it as an entry in the vector Y. The only purpose of this step is to produce X and Y.

2. Regressions produce coefficients as a function of the entire matrix X and the vector Y.

It is to be noted that the vast majority of the CPU time is spent in step 1. Step 2 typically takes negligible time. In the expensive step 1, each path is generated and evaluated separately to produce a row in X and an entry in Y. This makes it easy to distribute the paths generated in step 1 over multiple calculation nodes or cores. For that reason, we always strive to disentangle complex calculations such that most of them can be conducted path-wise, so we maximize opportunities for parallel processing.

SVD Regression We may compute the regression coefficients with the standard formula:

$$\beta = \left(X^T X\right)^{-1} \left(X^T Y\right)$$

This is a classic result, and one that is trivial to implement. It is also easy to differentiate, either analytically or automatically through standard AAD. However, this may not be good enough for our purpose. The covariance matrix $\left(X^T X\right)$ of the regression variables can easily end up singular or close depending on the choice of regression variables. This is especially the case when the number of regression variables exceeds the number of state variables, or when a low number of pre-simulations does not produce enough non linearity between the regression variables, so that the columns of X end up close to collinear, resulting in unstable regression results.

Although this problem generally vanishes as we increase the number of pre-simulations, that solution is not acceptable. In Section 2.2.1, we strive to reduce the reliance of the end result to the accuracy of proxies, so we may aggressively reduce the number of basis functions and pre-simulations. We certainly don't want to worry about collinearity when doing so.

Our preferred solution is to replace the standard regression by the so-called *SVD regression*. SVD regression is somewhat more complicated to conduct and differentiate than standard regression. We explain SVD regression in appendix A, and its differentiation in Section 4.2.1.

SVD regression produces stable results even when regression variables are collinear or close. Otherwise, its results coincide with those of standard regression. For that reason, we *systematically* use SVD regression, and never worry about collinearity when deciding on the number of pre-simulations.

SVD regression is implemented with the formula:

$$\beta = UD\Sigma V^T Y$$

where $X = VDU^T$ is the SVD of X *excluded for singular values that are null or lower than some threshold (called svd cut)* ϵ, $\Sigma_{jj} = \frac{1}{D_{jj}^2 + \lambda^2}$, λ is the Tikhonov parameter (small norm preference). See our appendix A for more detail. Computer code for the SVD can be found in [26] (Chapter 2.6) alongside a full description and an enlightening discussion (Chapter 15.4). Wikipedia's article on Tikhonov regularization and [7] provide further information and discussion on SVD regression, Tikhonov's small norm preference and application to the valuation of callable transactions with Monte-Carlo.

After regression is complete and the vector (of size $J+1$) of coefficients β is known, we may proceed with valuation, using proxies $\widetilde{V}^m_{T^{ex}} = \beta^{ex} \cdot X^m_{T^{ex}}$ in place of future values within the simulations. Therefore LSM is best implemented as a *pre-simulation* step, which sole purpose is to compute the regression coefficients β. After that, the *main simulation* step may proceed and use β to produce proxies within simulations whenever future values are needed. The regression step *turns the callable trade into a path-dependent* transaction, one that can be valued with standard Monte-Carlo simulations.

Importantly also, the main simulations should use a different *random seed* to pre-simulations in order to avoid a *foresight bias*.

Note finally that the *true* future value $V^m_{T^{ex}} = g\left(S^m_{T^{ex}}\right)$ is a function of the state variables and the regression variables $X_{T^{ex}j} = f_j\left(S_{T^{ex}}\right)$ are also functions (possibly basis functions) of state variables. Hence, the proxy can be made to approach the true value to arbitrary precision (with a performance cost), depending on the dimension and choice of the basis functions.

However, when the underlying transaction is path-dependent, that is no longer the case, because then, the true value of the underlying transaction at a future date is no longer a function of the state variables alone, but also of some transaction-specific path-dependency. For instance, if the transaction has a barrier, then its value is 0 if the barrier was previously hit, something that can't be deduced from the state variables alone. In this case, a regression proxy that only uses state variables may be severely inaccurate.

In the case of callable exotics, this problem is easily solved by adding the path-dependency variable (hit indicator for barriers, current average for Asians, etc.) among the regression variables (and possibly basis functions of it too, basically treat the path dependency as another state variable). Unfortunately, as we will see next, that solution does not practically apply to xVA.

2.1.2 Multi callable instruments

We now move on to transactions that may be exercised on multiple dates, such as Bermudas. For such instruments, we need to estimate, for each exercise date, the future *continuation value*. The continuation value is the value of the callable transaction immediately after that exercise date, assuming it is not terminated there. If the continuation value is negative, the transaction is exercised at that exercise date, therefore, its value on the exercise date is 0. The continuation value is the sum of the values of future cash-flows *up to the next exercise date*, assuming no prior exercise occurred, and the value of the callable on the next exercise date, including the possibility that it could be exercised there. Hence, all this must be computed recursively, from the last exercise to the first one.

On the last L'th exercise date, assuming no prior exercise, we have a one-time call and we proceed as previously to produce the coefficients β^L such that in every scenario m, the continuation value is estimated with $\widetilde{C}^m_{T^{ex}_L} = \beta^L \cdot X^m_{T^{ex}_L}$. Further, a proxy to the future value of the callable transaction on that date is $\widetilde{V}^m_{T^{ex}_L} = \max\left(0, \widetilde{C}^m_{T^{ex}_L}\right)$.

Assuming now we computed β^{l+1} and all the $\widetilde{C}^m_{T^{ex}_{l+1}} = \beta^{l+1} \cdot X^m_{T^{ex}_{l+1}}, 1 \leq m \leq M$, we can recursively compute β^l and $\widetilde{C}^m_{T^{ex}_l} = \beta^l \cdot X^m_{T^{ex}_l}, 1 \leq m \leq M$ by regression of $Y_{T^{ex}_l}$ over $X_{T^{ex}_l}$ where:

- $Y_{T^{ex}_l} = \left(Y^m_{T^{ex}_l}\right), 1 \leq m \leq M$ is the vector of continuation payoffs by scenario: $Y^m_{T^{ex}_l} =$

$$\sum_{T_l^{ex} < T_p \leq T_{l+1}^{ex}} CF_p^m + \max\left(0, \widetilde{C}_{T_{l+1}^{ex}}^m\right).$$

- $X_{T_l^{ex}}$ is the matrix whose rows are simulations and columns are regression variables at time T_l^{ex}. Note that the number of state variables and how they are computed out of state variables may differ for each exercise.

The only difference with the one-time callable case is therefore that proxies estimate continuation values and not simply the PV of all future cash-flows. The production of the regression coefficients for each call date requires proxies for the next call date. Hence, regression coefficients must be produced recursively, from the last call date to the first one.

To this extent, the application to xVA is somewhat simpler, since in that case, like in the one-time callable case, proxies are used as an estimate of the PV of all future cash-flows, without any form of recursiveness. In the case of xVA, we have multiple proxies, but their production does not depend on one another, contrarily to the multi callable case.

2.1.3 xVA

Although LSM was designed for callable exotics, it found another major application in the domain of xVA. We focus on CVA here without loss of generality. A CVA on an uncollateralized netting set is, by definition:

$$\begin{aligned} & E\left[\sum_p \left(1 - R_{T_p}\right) 1_{\{T_{p-1} \leq \tau < T_p\}} \max\left(0, V_{T_p}\right)\right] \\ \approx \quad & E\left[\sum_p \left(1 - R_{T_p}\right) 1_{\{T_{p-1} \leq \tau < T_p\}} \max\left(0, \widetilde{V}_{T_p}\right)\right] \end{aligned}$$

where τ is the default time, R the recovery rate and V_{T_p} is the PV of the netting set at exposure time T_p, that is the PV at that time, of all future cash-flows from all the transactions in the netting set.

In the frequent special case where recovery is deterministic and default is independent from the drivers of the netting set, this definition is simplified into:

$$CVA = E\left[\sum_p \left[1 - R\left(0, T_p\right)\right] \left[S\left(0, T_{p-1}\right) - S\left(0, T_p\right)\right] \max\left(0, V_{T_p}\right)\right]$$

where R is the recovery assumed today for future default dates, and S is today's, that is unconditional, survival probability to some future date. The definition of the CVA, and xVA in general, is much simpler in this case, since the only random variable in the expectation is V_{T_p}. It is clear that the CVA in this case can be further defined as:

$$CVA = \sum_p \left[1 - R\left(0, T_p\right)\right] \left[S\left(0, T_{p-1}\right) - S\left(0, T_p\right)\right] E\left[\max\left(0, V_{T_p}\right)\right]$$

where the terms outside of the expectation are deterministic multipliers, and the only terms that require simulations are $E\left[\max\left(0, V_{T_p}\right)\right]$. These terms are called *exposures* and denoted:

$$e\left(T\right) = E\left[\max\left(0, V_T\right)\right]$$

Further, it is also customary to consider continuous exposures and define the default intensity as:

$$\lambda(0,T) \equiv -\frac{\partial \log S(0,T)}{\partial T} \Leftrightarrow S(0,T) = \exp - \int_0^T \lambda(0,u)\, du$$

In this case, the CVA is written as an integral over continuous exposure times:

$$CVA = \int [1 - R(0,t)]\, S(0,t)\, \lambda(0,t)\, e(t)\, dt$$

In this document, we choose to write CVA as a discrete sum and conduct calculations in the general case, where our results thankfully apply. When we derive these results, we also show their simplified form in the independent credit case.

Some of the transactions in the netting set may be optional, path-dependent, exotic or callable, so costly nested simulations or inaccurate approximations would be necessary without LSM. LSM pre-simulations provide proxies for V on the *exposure dates* T_p, so that the main-simulations can value CVA using those proxies instead of the unavailable future values[11].

For the case of a *collateralized* netting set, we model the collateral as a function of the PV of the netting set at the collateral fixing date. Define the margin period of risk as θ which could be 10 days, and the margin date as $T_p^{mrg} = T_p - \theta$. For instance, the CVA on a fully collateralized netting set would be:

$$\begin{aligned} & E\left[\sum_p \left(1 - R_{T_p}\right) 1_{\{T_{p-1} \leq \tau < T_p\}} \max\left(0, V_{T_p} - V_{T_p^{mrg}}\right)\right] \\ \approx \quad & E\left[\sum_p \left(1 - R_{T_p}\right) 1_{\{T_{p-1} \leq \tau < T_p\}} \max\left(0, \widetilde{V}_{T_p} - \widetilde{V}_{T_p^{mrg}}\right)\right] \end{aligned}$$

or with independent credit:

$$CVA = \sum_p [1 - R(0,T_p)]\, [S(0,T_{p-1}) - S(0,T_p)]\, E\left[\max\left(0, V_{T_p} - V_{T_p^{mrg}}\right)\right]$$

The only change is in the exposure, which is now defined as:

$$e(T) = E\left[\max\left(0, V_T - V_T^{mrg}\right)\right]$$

Once again, a pre-simulation regression step provides the regression coefficient for the exposure dates and the collateral fixing dates, so that the main simulations can use proxies for the future values of the netting set on those dates without the need for nested simulations.

2.1.4 xVA with callable or path-dependent transactions

The computation of the CVA becomes somewhat more complicated when we have path-dependent or callable transactions in the netting set.

[11] Although in this case, we deal with potentially thousands of cash-flows depending on hundreds of state variables simulated given thousands of market parameters. See [5] for techniques for coping with this complexity and perform fast, accurate valuations and risk on light hardware. We expose some core components in this paper, although other technologies are necessary, such as script visitors, compression (also exposed in [9]) and decoration. We also need an arbitrage-free model that effectively simulates a potentially large number of underlying rates, exchange rates, equity indices, etc. Such models are called *hybrid models* and their implementation is well known since the early 2000s.

Let us consider the problem of computing an exposure[12]

$$e(T) = \max(0, V_T)$$

on a netting set that contains callable transactions. This is not a problem when the exposure date T is *anterior* to exercises. We deal with the exercises *first*, computing the proxy coefficients for continuation values. Then, we regress the PV of the netting set at the exposure date, taking into account the value of callable transactions on their exercise date, like we did for the multi-callable case.

If we have an exercise date *before* the exposure date $T > T^{ex}$, however, we run into a problem that is not easily resolved in practice. LSM essentially turns callable trades into path-dependent transactions. A callable becomes a (discretely monitored) barrier that is hit on the 1^{st} exercise date where the proxy to the continuation value is negative. Therefore, the path-wise value of the netting set at the exposure date depends not only on state variables on that date, but also on whether callable transactions in the netting set had been terminated or not prior to exposure. We have the same problem with all other path-dependent exotics with fixings prior to the exposure date. For instance, if we have an Asian transaction in the netting set, an exposure proxy that only considers the value of the state variables at the exposure dates ignores the cumulated average to that date and, therefore, cannot accurately represent the true path-wise value of the underlying netting set.

In the case of callable exotics, we easily resolved this problem by including path-dependencies *as additional state variables*. But we can't practically do that for CVA on netting sets that may include thousands of different exotics. That means that the proxy may end up severely mis-estimating the real PV at the exposure date. This is an open problem and a serious limitation at the time of writing. We don't have a solution to offer other than brute force with different sets of regression variables for each path-dependent or callable transaction in the netting set, eventually summing them all to produce a proxy to the value of the netting set. This is a solution that works, but a rather impractical and inefficient one.

However, the variation explained next shows that systematically uses proxies *only in indicator functions* considerably weakens the reliance of the final result on the accuracy of proxies, and therefore alleviates problems arising from proxy inaccuracies, including in this context.

2.2 Effective LSM

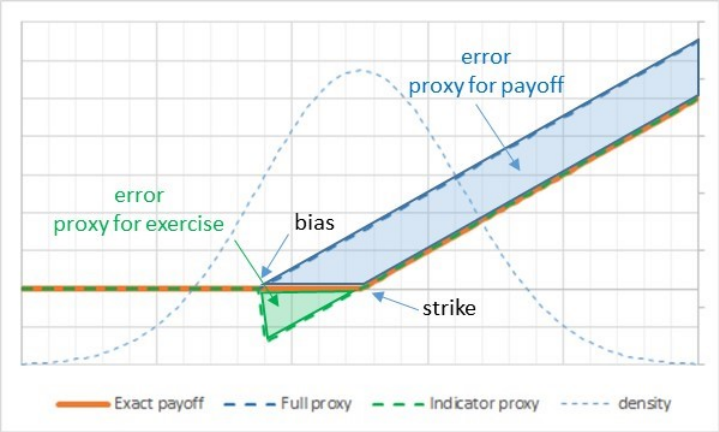
2.2.1 POI: Proxies Only in Indicators

We follow Andreasens POI approach to LSM. This is also something that has been mentioned and implemented by other people independently, but it is Andreasen who first realized the consequences of a systematic application of this approach, especially in the context of xVA, and had it implemented in Danske Banks award winning xVA system. POI only concerns how the main simulations use proxies. It doesnt modify how pre-simulations *produce* the proxies. Hence, the pre-simulation and regression steps explained in the previous section remain unchanged[13].

[12] We focus on exposures here rather than CVA for simplicity. CVA is simply the integral of exposures against credit.

[13] Although one of the points of this approach is to decrease the dependency on the accuracy of proxies so the production of proxies may be performed more approximately and hence, significantly faster. In addition, proxies themselves can be estimated more accurately and more practically using POI as explained in Section 4.3

As a simple first example, consider a call option on V with some strike κ. Its payoff is $\max(0, V-\kappa)$.



Suppose we have a biased proxy $\widetilde{V} = V + \mu$. If we use the proxy to estimate the payoff, we end up pricing with the wrong strike: $\max\left(0, \widetilde{V} - \kappa\right) = \max\left[0, V - (\kappa - \mu)\right]$. With a Gaussian approximation on V, we have an error of magnitude $\Delta \cdot \mu \approx \mu \Phi\left[\frac{V-\kappa}{\sqrt{\operatorname{var} V}}\right]$, hence in the order of the bias, μ. Now, if we somehow manage to use the proxy only for testing positivity (and we show next how exactly this is achieved in the context of callable exotics and xVA) then we are pricing with the right strike, the only mistake being to exercise the option out of the money in certain scenarios. In this case, we end up pricing

$$1_{\left\{\widetilde{V}>\kappa\right\}}(V-\kappa) = \underbrace{(V-\kappa)}_{\text{right payoff}} \underbrace{1_{\{V>\kappa-\mu\}}}_{\text{wrong exercise}}$$

and we have an error of the more acceptable magnitude

$$\mu\left(\Phi\left[\frac{V-\kappa}{\sqrt{\operatorname{var} V}}\right] - \Phi\left[\frac{V-\kappa-\mu}{\sqrt{\operatorname{var} V}}\right]\right) \simeq \mu^2 \frac{\varphi\left[\frac{V-\kappa}{\sqrt{\operatorname{var} V}}\right]}{\sqrt{\operatorname{var} V}}$$

of order μ^2.

2.2.2 One-time callable instruments

The classical LSM valuation of a one-time callable transaction is performed in the main simulation through:

$$V_0 = E\left[\max\left(0, V_{T^{ex}}\right)\right] \approx \frac{1}{N}\sum_{n=1}^{N} \max\left(0, V_{T^{ex}}^n\right) \approx \frac{1}{N}\sum_{n=1}^{N} \max\left(0, \widetilde{V}_{T^{ex}}^n\right)$$

This valuation directly uses the proxy to determine the payoff at the exercise date. Consequently, its accuracy is directly related to the accuracy of the proxy. That means that a linear regression over state variables may not be accurate enough, and a polynomial regression of a higher degree, or a higher dimensional set of basis functions, may be required. In addition, a large number of pre-simulations may be needed to ensure an acceptable convergence of the regression coefficients. In

fact it is shown in [18] that the number of simulations needed grows exponentially in the number of basis functions.

All that, in turn, comes at a considerable performance cost, and higher order regressions may produce instabilities that can be corrected but with considerable effort.

In addition, in order to produce risk sensitivities for V_0, the proxy and the regression coefficients would need to be differentiated with respect to the parameters of the model, again, at a cost. Although we provide an efficient LSM differentiation algorithm in our Section 4, it is best to avoid it when unnecessary.

Finally, we saw that the proxy may be inaccurate to a large degree, irrespectively of the order of the regression, when the underlying cash-flows are path-dependent, something that we can't correct in the case of the xVA without a considerable complication and performance penalty.

A simple solution to these problems is to use the proxy to the determination of the exercise but not the payoff as follows:

$$\begin{aligned}
V_0 &= E\left[\max\left(0, V_{T^{ex}}\right)\right] \\
&= E\left[1_{\{V_{T^{ex}}>0\}} V_{T^{ex}}\right] \\
&= E\left[1_{\{V_{T^{ex}}>0\}} E_{T^{ex}}\left(\sum_{T_k>T^{ex}} CF_k\right)\right] \\
&= E\left[1_{\{V_{T^{ex}}>0\}} \sum_{T_k>T^{ex}} CF_k\right] \\
\text{Monte-Carlo} \rightarrow &\approx \frac{1}{N}\sum_{n=1}^{N}\left[1_{\{V^n_{T^{ex}}>0\}} \sum_{T_k>T^{ex}} CF^n_k\right] \\
\text{LSM} \rightarrow &\approx \frac{1}{N}\sum_{n=1}^{N}\left[1_{\{\widetilde{V}^n_{T^{ex}}>0\}} \sum_{T_k>T^{ex}} CF^n_k\right]
\end{aligned}$$

This solution is extremely simple. Instead of computing the payoff as a function of the proxy, we use it to determine whether exercise occurs or not at the exercise date, and, if not, evaluate the value of the remaining cash-flows within the simulation itself. Note that we can rewrite the final result from the previous equation as:

$$\frac{1}{N}\sum_{n=1}^{N}\left[\sum_{T_k>T^{ex}} 1_{\{\widetilde{V}^n_{T^{ex}}>0\}} CF^n_k\right]$$

Which means that in order to value the *callable* transaction, we value path-wise the cash-flows of the *underlying* transaction, but *discounted* for exercise. This is something that will be leveraged to a large degree for multi-callable deals and xVA next.

The main benefit is to considerably weaken the dependency on the accuracy of the proxy. Not only is it used only to test for positivity, but also the path-wise sensitivity of the end result to the proxies is now 0, meaning that inaccuracies in the proxy have little impact on the value (within limits):

$$\frac{\partial\left(1_{\{\widetilde{V}^n_{T^{ex}}>0\}}\sum_{T_k>T^{ex}} CF^n_k\right)}{\partial \widetilde{V}^n_{T^{ex}}} = \delta\left(\widetilde{V}^n_{T^{ex}}\right)\sum_{T_k>T^{ex}} CF^n_k$$

where δ is the Dirac delta function. We find

$$\begin{aligned} N \cdot E\left[\frac{\partial V_0}{\partial \widetilde{V}^n_{T^{ex}}}\right] &= E\left[\frac{\partial\left(1_{\{\widetilde{V}^n_{T^{ex}}>0\}}\sum_{T_k>T^{ex}} CF^n_k\right)}{\partial \widetilde{V}^n_{T^{ex}}}\right] \\ &= E\left[\delta\left(\widetilde{V}^n_{T^{ex}}\right)\sum_{T_k>T^{ex}} CF^n_k\right] \\ &= E\left[\delta\left(\widetilde{V}^n_{T^{ex}}\right)E_{T^{ex}}\left(\sum_{T_k>T^{ex}} CF^n_k\right)\right] \\ &= E\left[\delta\left(\widetilde{V}^n_{T^{ex}}\right)V^n_{T^{ex}}\right] \\ &= \varphi_{\widetilde{V}^n_{T^{ex}}}(0)E\left[V^n_{T^{ex}}\middle|\widetilde{V}^n_{T^{ex}}=0\right] \end{aligned}$$

where $\varphi_{\widetilde{V}^n_{T^{ex}}}(0)$ is the density of $\widetilde{V}^n_{T^{ex}}$. Hence, the sensitivity of the value to the proxy is 0 at the point of accuracy. That means that the PV is not sensitive to limited errors in the proxy, hence, there is no necessity for expensive high dimensional regressions or a large number of pre-simulations. In addition, this alleviates the problems linked to potentially severe inaccuracies of proxies in the context of CVA with path-dependent exotics.

Finally, another considerable benefit is that there is no need to differentiate through the LSM regression process in this case. The sensitivity to the proxy, hence also to the regression coefficients, is 0 when the proxy is accurate, and it amounts to unwanted numerical noise when it isn't. In addition, since AAD differentials of binary functions are always 0, AAD will do the right thing *automatically*. Pre-simulations need not be instrumented with AAD and may run without the AAD overhead, while the sensitivity to proxies in the AAD-instrumented main simulations are automatically ignored. Therefore, AAD overhead is limited to main simulations, where all risks can be computed to analytical precision for the cost of 4 to 8 valuations, just like for any other path-dependent transaction. This point is further investigated in Section 3.

2.2.3 Multi callable instruments

This analysis immediately applies to multi-callable instruments too, and the same benefits hold.

$$\begin{aligned}
V_0 &= E\left[\max\left(0, C_{T_1^{ex}}\right)\right] \\
&= E\left[1_{\left\{C_{T_1^{ex}}>0\right\}} C_{T_1^{ex}}\right] \\
&= E\left[1_{\left\{C_{T_1^{ex}}>0\right\}} \left(\sum_{T_1^{ex}<T_k\leq T_2^{ex}} CF_k + C_{T_2^{ex}}\right)\right] \\
&\vdots \\
&= E\left[\sum_k \left(\prod_{T_l^{ex}<T_k} 1_{\left\{C_{T_l^{ex}}>0\right\}}\right) CF_k\right] \\
&\approx \frac{1}{N}\sum_{n=1}^{N}\left[\sum_k \left(\underbrace{\prod_{T_l^{ex}<T_k} 1_{\left\{\widetilde{C}^n_{T_l^{ex}}>0\right\}}}_{\eta^n_{T_k}}\right) CF_k^n\right] \\
&= \frac{1}{N}\sum_{n=1}^{N}\left[\sum_k \eta^n_{T_k} CF_k^n\right]
\end{aligned}$$

Note that once again, we value the multi-callable deal by evaluating the cash-flows of its underlying transaction, and discounting them with a stochastic, path-dependent process η_t that accounts for exercise. Over the the n'th path that process is defined by:

$$\eta_t^n = \prod_{T_l^{ex}<t} 1_{\left\{\widetilde{C}^n_{T_l^{ex}}>0\right\}} \Leftrightarrow \begin{cases} \eta_0^n = 1 \\ \eta^n_{T_l^{ex}+} = 1_{\left\{\widetilde{C}^n_{T_l^{ex}}>0\right\}} \eta^n_{T_l^{ex}-} \end{cases}$$

This is a piece-wise constant process that updates itself at exercise dates, jumping to 0 in case of an exercise. Its simulation is trivial and inexpensive.

This also reinforces the view that LSM pre-simulations turn a callable transaction into a proxy path-dependent trade. To ignore the differentiation of LSM regressions in the production of risk sensitivities boils down to computing risks for the proxy path-dependent transaction, which, as we demonstrate in Section 3, has the same sensitivities as the callable trade when LSM proxies are accurate, the difference amounting to unwanted numerical noise otherwise.

2.2.4 xVA without collateral

Andreasen [2] demonstrated that the same can be said of xVA. As previously, we focus on CVA without loss of generality and start without collateral:

$$\begin{aligned} CVA &= E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \max\left(0, V_{T_p}\right)\right] \\ &= E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{V_{T_p}>0\right\}} V_{T_p}\right] \\ &= E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{V_{T_p}>0\right\}} E_{T_p}\left(\sum_{T_k>T_p} CF_k\right)\right] \\ &= E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{V_{T_p}>0\right\}} \sum_{T_k>T_p} CF_k\right] \end{aligned}$$

Reversing the order of the sums to sum over cash-flows first:

$$CVA = E\left[\sum_k CF_k \sum_{T_p<T_k} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{V_{T_p}>0\right\}}\right]$$

And using LSM proxies:

$$\begin{aligned} CVA &\approx E\left[\sum_k CF_k \underbrace{\sum_{T_p<T_k} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{\widetilde{V}_{T_p}>0\right\}}}_{=\eta_{T_k}}\right] \\ &= E\left[\sum_k \eta_{T_k} CF_k\right] \end{aligned}$$

where

$$\eta_{T_k} = \sum_{T_p<T_k} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{\widetilde{V}_{T_p}>0\right\}}$$

or, in the independent credit case:

$$\eta_{T_k} = \sum_{T_p<T_k} \left[1-R\left(0,T_p\right)\right]\left[S\left(0,T_{p-1}\right)-S\left(0,T_p\right)\right] 1_{\left\{\widetilde{V}_{T_p}>0\right\}}$$

Note that in this case, the only non-determinism in η comes from the indicator for positive future PVs (using proxies).

Once again, we can value the CVA by valuing the underlying netting set, while applying a stochastic path-dependent discounting to its cash-flows. This discounting process η_t starts at 0, is constant between exposure dates, and updates itself on exposure dates with:

$$\eta_{T_{p+1}} = \eta_{T_p} + \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_p\leq\tau<T_{p+1}\}} 1_{\left\{\widetilde{V}_{T_p}>0\right\}}$$

or, in the independent credit case:

$$\eta_{T_{p+1}} = \eta_{T_p} + \left[1 - R\left(0, T_p\right)\right]\left[S\left(0, T_p\right) - S\left(0, T_{p+1}\right)\right] 1_{\left\{\widetilde{V}_{T_p} > 0\right\}}$$

This is where Andreasens approach shines brightest, allowing xVA to be computed by applying a special discount process to the cash-flows of the underlying netting set. Where nested simulations would have been necessary without LSM, and the naive use of LSM could produce large inaccuracies, we can now evaluate xVA accurately for pretty much the cost of the valuation of the underlying netting set, simply by applying a stochastic, path-dependent discounting to its cash-flows, where the discounting process is subject to a simple update on exposure dates.

The resulting algorithm is accurate, efficient and practical. The accuracy comes from POI, which alleviates the reliance of the end result to the accuracy of proxies. That means that proxies may be produced with aggressive optimizations, which also increases the performance of the pre-simulation step. The main performance gain comes from the ability to value an xVA by valuying the cash-flows in its netting set, simply applying a path-dependent discounting, which follows a simple process. Main simulations can also be conducted path-wise, and different paths can be easily evaluated in parallel. Finally, the algorithm is extremely practical, since the valuation scripts for the cash-flows simply need to be *decorated* with the special discounting to produce the xVA[14].

2.2.5 xVA with collateral

Perhaps surprisingly, this methodology does not extend by itself to the collateralized case. We consider a fully collateralized netting set, a period margin of risk of θ, and compute the exposure for some date T and margin date $T^{mrg} = T - \theta$ (after we are done with exposures, we will easily produce the complete CVA formula by integration against credit):

$$\begin{aligned} e\left(T\right) &= E\left[\max\left(0, V_T - V_{T-\theta}\right)\right] \\ &= E\left[1_{\{V_T > V_{T^{mrg}}\}} V_T\right] - E\left[1_{\{V_T > V_{T^{mrg}}\}} V_{T^{mrg}}\right] \end{aligned}$$

The left hand side lends itself to our previous reformulation without difficulty:

$$\begin{aligned} E\left[1_{\{V_T > V_{T^{mrg}}\}} V_T\right] &= E\left[1_{\{V_T > V_{T^{mrg}}\}} E_T\left(\sum_{T_k > T} CF_k\right)\right] \\ &= E\left[1_{\{V_T > V_{T^{mrg}}\}} \sum_{T_k > T} CF_k\right] \\ &= E\left[\sum_{T_k > T} 1_{\{V_T > V_{T^{mrg}}\}} CF_k\right] \\ &\approx E\left[\sum_{T_k > T} 1_{\left\{\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}} CF_k\right] \end{aligned}$$

[14] Drawing a parallel with classical credit analysis, it is well known that if default is modelled by a poisson process (or in more general a Cox process see [22]) with intensity h_t then a positive cash-flow should be discounted with an intensity adjusted rate. What may be less known is that for cash-flows that can be both negative and positive the adjusted rate should take the total value into account, see [14] for details. In this case a 0-recovery defaultable claim (disregarding our own credit risk DVA) can be valued recursively as

$$V_t = E_t\left[\sum_k e^{-\int_t^{T_k} h_u 1_{\{V_u > 0\}}\, du} CF_k\right] \approx E_t\left[\sum_k e^{-\int_t^{T_k} h_u 1_{\{\widetilde{V}_u > 0\}}\, du} CF_k\right]$$

This is another way to see that the proxy is only used for discounting **not** the cash-flows.

The right hand side, however, is trickier. Assume that there is no payment within the margin period of risk[15], then we can write as before:

$$E\left[1_{\{V_T>V_{T^{mrg}}\}}V_{T^{mrg}}\right] = E\left[1_{\{V_T>V_{T^{mrg}}\}}E_{T^{mrg}}\left(\sum_{T_k>T} CF_k\right)\right]$$

But we cannot move $1_{\{V_T>V_{T^{mrg}}\}}$ in and out of the conditional expectation because it is not T^{mrg} measurable. What we can write is this:

$$\begin{aligned}
& E\left[1_{\{V_T>V_{T^{mrg}}\}}V_{T^{mrg}}\right] \\
= \quad & E\left[E_{T^{mrg}}\left[1_{\{V_T>V_{T^{mrg}}\}}E_{T^{mrg}}\left(\sum_{T_k>T} CF_k\right)\right]\right] \\
= \quad & E\left[E_{T^{mrg}}\left(1_{\{V_T>V_{T^{mrg}}\}}\right)E_{T^{mrg}}\left(\sum_{T_k>T} CF_k\right)\right] \\
\underset{\text{LSM}}{\approx} \quad & E\left[E_{T^{mrg}}\left(1_{\left\{\widetilde{V}_T>\widetilde{V}_{T^{mrg}}\right\}}\right)E_{T^{mrg}}\left(\sum_{T_k>T} CF_k\right)\right] \qquad (1)
\end{aligned}$$

This is as far as we can go, in general

$$E\left[1_{\{V_T>V_{T^{mrg}}\}}V_{T^{mrg}}\right] \neq E\left[\sum_{T_k>T} \eta_{T_k}CF_k\right]$$

for some process η_t adapted to the filtration generated by our model. That means the algorithm we derived in the uncollateralized case cannot be transposed here. At least not in the context of standard Monte-Carlo simulations.

But it very much can in the context of *branching* simulations.

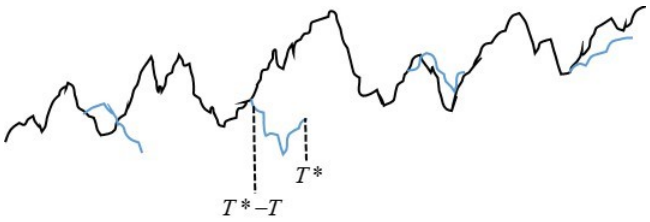
Branching simulations Branching Monte-Carlo simulations were recently produced by Henry-Labordere in [20], under the inspiration of 40 year old work by McKean from [24]. Andreasen and Huge successfully applied this technique to resolve the problem of collateralized CVA. Subsequently, systematic support for branching was implemented in Danske Banks award winning xVA systems by Savine and extended to produce RWA and KVA without nested simulation by Flyger, as was demonstrated live in Global Derivatives in 2015 and 2016 by computing full risk sensitivities for a RWA on a large netting set locally on MacBook Pro within a couple of minutes.

Branching Monte-Carlos consist in the simulation, in every scenario, of not only one evolution of the world, but also a number of branches, where each branch sticks out of the main path at a particular point in time (for our purpose, the margin dates) and starts a parallel simulation, that follows the same dynamics that the main branch but independently from it.

Note that branching in this sense is very different from nested simulations, since, for every simulation, we only simulate one parallel branch sticking out of each collateral fixing date, and terminate this parallel branch on the corresponding exposure date, typically 10 days later.

[15]This can be ensured by pre-processing cash-flows to align their schedules, a process referred to as compression and out of the scope of this article. We refer to [5] and [9]. We will assume in all that follows that there is no payment within the margin period of risk, simply because that simplifies notations and equations. There is no theoretical difficulty (but no practical need either) to soften that assumption.

'Hairy' Simulation



Given that our Monte-Carlo simulates some multi-dimensional diffusion for the state variables of the model $dS = a(S,t)\,dt + b(S,t)\,dW$[16], a secondary branch sticking out at time T_B, noted ${}^B S$, is part of the same scenario, and simulated in such a way that:

1. For $t \leq T_B$ ${}^B S_t^n = S_t^n$.
2. For $t > T_B$ $d^B S = a\left({}^B S, t\right) dt + b\left({}^B S, t\right) dW^B$ where W^B is a standard multi-dimensional Brownian Motion *independent from W*.

While this looks (and is) extra work for the developer and the CPU both, these branching simulations elegantly and effectively resolve our collateralized exposure problem: consider a branch sticking out at the collateral fixing date T^{mrg}, and note

$${}^B\widetilde{V}_T$$

the LSM proxy to V *read on that Branch*. That means that

$${}^B\widetilde{V}_T = \beta \cdot f\left({}^B S_T\right)$$

is produced with the same regression coefficients, but with regression variables picked on the branch. Then, by construction, ${}^B\widetilde{V}_T$ and $\widetilde{V}_T$ are independent and identically distributed *conditionally to the filtration at* T^{mrg}. Hence we continue (1) as follows:

$$\begin{aligned}
& E\left[1_{\{V_T > V_{T^{mrg}}\}} V_{T^{mrg}}\right] \\
\approx\ & E\left\{E_{T^{mrg}}\left[1_{\left\{\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}}\right] E_{T^{mrg}}\left[\sum_{T_k > T} CF_k\right]\right\} \\
\text{identical distr.} \rightarrow\ = \ & E\left\{E_{T^{mrg}}\left[1_{\left\{{}^B\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}}\right] E_{T^{mrg}}\left[\sum_{T_k > T} CF_k\right]\right\} \\
\text{independence} \rightarrow\ = \ & E\left\{E_{T^{mrg}}\left[1_{\left\{{}^B\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}} \sum_{T_k > T} CF_k\right]\right\} \\
=\ & E\left[1_{\left\{{}^B\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}} \sum_{T_k > T} CF_k\right]
\end{aligned}$$

And, finally, we have the formula for our exposure:

$$e(T) = E\left[\left(1_{\left\{\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}} - 1_{\left\{{}^B\widetilde{V}_T > \widetilde{V}_{T^{mrg}}\right\}}\right) \sum_{T_k > T} CF_k\right]$$

[16] The technique easily extends to jumps too.

Notice, that $1_{\{V_T>V_{T^{mrg}}\}} - 1_{\{{}^B V_T>V_{T^{mrg}}\}}$ is ± 1 with equal probability and 0 otherwise. This may look wrong since $e(T)$ is supposed to be positive. However, this variable is multiplied by V_T so the positive contributions, $V_T > V_{T^{mrg}}$, are bigger than the negative once, $V_T < V_{T^{mrg}}$. In math terms let $\Delta V_t = V_t - V_s, \Delta^B V_t = {}^B V_t - {}^B V_s$ then

$$\begin{aligned} & e(t) \\ = \; & E\left[\left(1_{\{\Delta V_t>0\}} - 1_{\{\Delta^B V_t>0\}}\right) V_t\right] \\ = \; & E\left[\left(1_{\{\Delta V_t>0,\Delta^B V_t<0\}} - 1_{\{\Delta V_t<0,\Delta^B V_t>0\}}\right) V_t\right] \\ \geq \; & E\left[\left(1_{\{\Delta V_t>0,\Delta^B V_t<0\}} - 1_{\{\Delta V_t<0,\Delta^B V_t>0\}}\right) V_s\right] \end{aligned}$$

since

$$\begin{aligned} E\left[1_{\{\Delta V_t>0,\Delta^B V_t<0\}} V_s\right] &= E\left[E_s\left[1_{\{\Delta V_t>0,\Delta^B V_t<0\}}\right] V_s\right] \\ &= E\left[\Pr_s\left(\Delta V_t > 0, \Delta^B V_t < 0\right) V_s\right] \\ \text{independence} \rightarrow \; &= E\left[\Pr_s\left(\Delta V_t > 0\right) \Pr_s\left(\Delta^B V_t < 0\right) V_s\right] \\ \text{identical distr.} \rightarrow \; &= E\left[\Pr_s\left(\Delta^B V_t > 0\right) \Pr_s\left(\Delta V_t < 0\right) V_s\right] \\ \text{independence} \rightarrow \; &= E\left[\Pr_s\left(\Delta^B V_t > 0, \Delta V_t < 0\right) V_s\right] \\ &= E\left[E_s\left[1_{\{\Delta^B V_t>0,\Delta V_t<0\}}\right] V_s\right] \\ &= E\left[1_{\{\Delta V_t<0,\Delta^B V_t>0\}} V_s\right] \end{aligned}$$

we have $e(t) \geq 0$.

Let *crd* subscript in the expectation stand for conditional to the evolution of the credit, default time and recovery then using proxies in indicators, we get:

$$\begin{aligned} & CVA \\ \approx \; & E\left[\sum_p \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} E_{crd}\left[e\left(T_p\right)\right]\right] \\ = \; & E\left[\sum_p \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \sum_{T_k>T_p} \left(1_{\left\{\widetilde{V}_{T_p}>\widetilde{V}_{T_p^{mrg}}\right\}} - 1_{\left\{{}^{B_p}\widetilde{V}_{T_p}>\widetilde{V}_{T_p^{mrg}}\right\}}\right) CF_k\right] \\ = \; & E\left[\sum_k CF_k \sum_{T_p<T_k} \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \left(1_{\left\{\widetilde{V}_{T_p}>\widetilde{V}_{T_p^{mrg}}\right\}} - 1_{\left\{{}^{B_p}\widetilde{V}_{T_p}>\widetilde{V}_{T_p^{mrg}}\right\}}\right)\right] \\ = \; & E\left[\sum_k \eta_{T_k} CF_k\right] \end{aligned}$$

where

$$\eta_{T_p} = \sum_{T_q<T_p} \left(1 - \mathrm{R}_{T_q}\right) 1_{\{T_{q-1}\leq\tau<T_q\}} \left(1_{\left\{\widetilde{V}_{T_q}>\widetilde{V}_{T_q^{mrg}}\right\}} - 1_{\left\{{}^{B_q}\widetilde{V}_{T_q}>\widetilde{V}_{T_q^{mrg}}\right\}}\right)$$

is a collateral-adjusted discounting process, very similar to what we derived in the uncollateralized case. In the independent credit case:

$$\eta_{T_p} = \sum_{T_q<T_p} \left[1 - R\left(0,T_q\right)\right] \left[S\left(0,T_{q-1}\right) - S\left(0,T_q\right)\right] \left(1_{\left\{\widetilde{V}_{T_q}>\widetilde{V}_{T_q^{mrg}}\right\}} - 1_{\left\{{}^{B_q}\widetilde{V}_{T_q}>\widetilde{V}_{T_q^{mrg}}\right\}}\right)$$

This discounting process that starts at 0 and trivially follows the update equation:

$$\eta_{T_{p+1}} = \eta_{T_p} + \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1} \leq \tau < T_p\}} \left(1_{\left\{ \widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}} - 1_{\left\{ {}^{B_p}\widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}} \right)$$

or, in the independent credit case:

$$\eta_{T_{p+1}} = \eta_{T_p} + \left[1 - R\left(0, T_p\right)\right] \left[S\left(0, T_p\right) - S\left(0, T_{p+1}\right)\right] \left(1_{\left\{ \widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}} - 1_{\left\{ {}^{B_p}\widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}} \right)$$

All the benefits from the uncollateralized case apply, and that makes it possible to compute all kind of collateralized xVA, RWA and KVA without the need for nested simulations and avoid using the proxy for the cash-flows.

The difference is that the simulations must now incorporate branching. However, we only need one branch per scenario per exposure date, and these branches are only as long as the margin period of risk, and assuming exposure dates are distant by more than that, branches dont overlap. All this means that the CPU time for a Monte-Carlo with Branching should be less than twice the time of a standard Monte-Carlo with the same number of simulations.

However, the Monte-Carlo estimation of

$$1_{\left\{ \widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}} - 1_{\left\{ {}^{B_p}\widetilde{V}_{T_p} > \widetilde{V}_{T_p^{mrg}} \right\}}$$

introduces significant variance and forces to run a lot more simulations than in the uncollateralized case to achieve a similar degree of accuracy[17]. This problem may be easily and significantly alleviated by using smooth binary or call spread functions in place of the indicators:

$$1_{\{x>0\}} \approx \left(\frac{1}{\varepsilon}x + \frac{1}{2}\right) 1_{\left\{x > -\frac{\varepsilon}{2}\right\}} - \left(\frac{1}{\varepsilon}x - \frac{1}{2}\right) 1_{\left\{x > \frac{\varepsilon}{2}\right\}}$$

See for instance [28].

Another advisable improvement is to use the same regression coefficients (but with different values for the regression variables) for $\widetilde{V}_{T_p^{mrg}}, \widetilde{V}_{T_p}, {}^{B_p}\widetilde{V}_{T_p}$, which is correct as long as no cash-flow is paid within the margin period of risk.

3 Cash-flow differentials and full sensitivities

Our Section 2 exposed an efficient LSM implementation that only uses proxies in indicators. Regression coefficients are produced in a prior pre-simulation step, and used by main simulations to produce proxies as required. These proxies are used (only) as arguments to indicator functions that subject the payment of subsequent cash-flows to positivity.

From here on, we assume that valuation proceeds in this way, and explore the consequences for the derivation of risk sensitivities. Section 4 explains in detail how this whole process, including

[17] Assume Z_1, Z_2 are symmetric distributed, uncorrelated variables with mean 0 and some variance then $Var\left(1_{\{Z_i>0\}}\right) = \frac{1}{2} - \left(\frac{1}{2}\right)^2 = \frac{1}{4}$ and $Var\left(1_{\{Z_1>0\}} - 1_{\{Z_2>0\}}\right) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$. Hence for these 2 variables to have the same Monte-Carlo noise we need twice as many paths for the latter.

pre-simulations and main simulations, can be efficiently differentiated. In the meantime, this section investigates when and to what extent such complete differentiation is necessary or even desirable.

To understand this point, suppose that, for the purpose of computing risk sensitivities, we ignore the LSM step and proceed to compute derivatives path-wise over the main simulations with AAD[18]. AAD, by nature, ignores differentials of indicators[19]. We only use proxies in indicators for valuation. Therefore, what AAD path-wise differentials over the main simulations effectively produce are *differentials of expected cash-flows* with fixed proxies, also called cash-flow differentials.

This section investigates in what circumstances, and to what extent, it is correct to produce risk sensitivities in this way. In other terms, in what situations are cash-flow differentials a correct estimation of the complete differentials?

In such situations, the differentiation can be conducted simply by path-wise AAD over the main simulations and without extra work. In particular, it is not necessary to differentiate the LSM step. It follows that the LSM code doesn't need to be *instrumented for AAD* and can run in standard mode without any AAD overhead. We will show that this is the case in many contexts of practical relevance, in particular for all callable exotics, and xVA, both with or without collateral, where the underlying netting set does not contain callable transactions.

In those situations where cash-flow differentials are *not* a correct estimate of the complete sensitivities, the whole process LSM + main simulations must be differentiated, which is not trivial to achieve in a speed and memory efficient way. We resolve that problem in Section 4, and describe an algorithm that uses path-wise AAD, check-pointing and matrix adjoint calculus to achieve high speed differentiation and low memory consumption without sacrificing accuracy. This being said, it is best to skip those calculations altogether in the situations where we can get away with simple path-wise AAD over the main simulations only.

We will characterize the contexts where cash-flow differentials are sufficient or not, and, in particular, review the different situations investigated in Section 2: once and multi callable exotics, xVA with or without collateral, on a netting set that may or may not itself contain callable exotics.

We will be using a simple and generally well known result that stems directly from derivation and Dirac mass integration rules. If some value can be written in the form:

$$E\left[X\left(a\right)\max\left(0,C\left(a\right)\right)\right]=E\left[X\left(a\right)C\left(a\right)1_{\left\{C(a)>0\right\}}\right]$$

where C and X, in addition to being random, depend on some parameter a. Further, if we use a proxy for C, but only in the indicator, that is we estimate the value as:

$$E\left[X\left(a\right)C\left(a\right)1_{\left\{\widetilde{C}(a)>0\right\}}\right]\equiv w\left(a\right)$$

Then the derivative of our estimate with respect to a is:

$$\begin{aligned} w'\left(a\right) &= E\left[\left(XC\right)'\left(a\right)1_{\left\{\widetilde{C}(a)>0\right\}}+X\left(a\right)C\left(a\right)\widetilde{C}'\left(a\right)\delta\left(\widetilde{C}\left(a\right)\right)\right] \\ &= \underbrace{E\left[\left(XC\right)'\left(a\right)1_{\left\{\widetilde{C}(a)>0\right\}}\right]}_{LHS}+\underbrace{E\left[X\left(a\right)C\left(a\right)\widetilde{C}'\left(a\right)\delta\left(\widetilde{C}\left(a\right)\right)\right]}_{RHS} \end{aligned}$$

[18]That means that we differentiate each path in isolation, and average sensitivities in the end. With AAD, that means that each path calculation generates and evaluates its own tape and then destroys it. This results in a low memory consumption and cache efficiency, since the tape records calculations for only one path at a time. In addition, we can trivially differentiate different paths in parallel. For these reasons, it is generally recognized that path-wise AAD differentiation is a very efficient means to differentiate Monte-Carlo simulations. In Section 4, we strive to conduct as much as possible of the whole differentiation path-wise.

[19]unless indicators are smoothed as discussed in Section 4.

where we have used that the Dirac delta function is the derivative of the indicator. The left hand side (LHS) is precisely the result of path-wise differentiation with fixed proxies. The right hand side (RHS) can be developed further into:

$$RHS = \varphi_{\widetilde{C}(a)}(0)\, E\left[X(a)\,\widetilde{C}'(a)\,C(a)\middle|\,\widetilde{C}(a)=0\right]$$

where $\varphi_{\widetilde{C}(a)}$ is the density of the proxy. We can see that if our proxy is accurate[20], $E\left[\cdot C\middle|\widetilde{C}=0\right]=0$ and the whole RHS collapses to 0, meaning that the LHS is indeed a correct estimator of the complete sensitivity.

In summary, we derived the (rather trivial, and generally known) result (2), which states that, provided that the proxy is an accurate representation of its target around 0:

$$\frac{\partial}{\partial a}E\left[XC1_{\{\widetilde{C}>0\}}\right]=E\left[\frac{\partial(XC)}{\partial a}1_{\{\widetilde{C}>0\}}\right] \tag{2}$$

That results holds only when the *trigger* (proxy in the indicator) is a proxy to the *payoff multiplier* (a multiplier of the indicator). In other terms, when what we exercise on is a proxy to what we get on exercise. This explains why the result holds for an option, including a callable transaction, an exposure, and therefore an xVA. But the result does not hold when the trigger is not a proxy to the payoff, like an option that delivers Nikkei if SnP hits 2,500. In other terms, in general:

$$\frac{\partial}{\partial a}E\left[XD1_{\{\widetilde{C}>0\}}\right]\neq E\left[\frac{\partial(XD)}{\partial a}1_{\{\widetilde{C}>0\}}\right]$$

The result (2) also relies on the accuracy of the proxy, or at least its capacity to correctly represent its target around 0. In reality, this is rarely the case, especially since POI decreased the dependency on the accuracy of the proxy for valuation purposes and gave us license to aggressively optimize the computation of proxies at the expense of their accuracy. Whether that means that we cannot rely on this result is discussed in length next.

3.1 When is differentiation unnecessary?

3.1.1 One-time callable instruments

We consider, as in Section 2, a transaction that pays the cash-flows $(CF_k)_{1\leq k\leq K}$ on dates $(T_k)_{1\leq k\leq K} > T^{ex}$ after the exercise date, provided the transaction is not terminated at that date. Its value is

$$E\left[\max\left(0,V_{T^{ex}}\right)\right]=E\left[1_{\{V_{T^{ex}}>0\}}V_{T^{ex}}\right]$$

where the continuation value $V_{T^{ex}}$ is the value of the transaction immediately after the exercise date, assuming the transaction is not terminated there. It is the future value of all subsequent cash-flows:

$$V_{T^{ex}}=E_{T^{ex}}\left[\sum_{T_k>T^{ex}}CF_k\right]$$

[20]Note that for this result to hold true, we only need the distribution of the true value, conditional to its proxy being null, to be narrowly confined around 0. In other words, the proxy must correctly represent to true value *around 0, that is, on the exercise frontier in the case of callables and on the edge between asset and debt in the case of xVA.*

We estimate this value with the help of the proxy, but only in the indicator, as explained in Section 2:

$$V_0 \equiv E\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right] = E\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} \sum_{T_k>T^{ex}} CF_k\right]$$

By a direct application of (2), we find that:

$$\frac{\partial V_0}{\partial a} = E\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} \frac{\partial V_{T^{ex}}}{\partial a}\right] = E\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} \sum_{T_k>T^{ex}} \frac{\partial CF_k}{\partial a}\right]$$

(assuming the proxy is accurate). This equation proves that in the case of one-time callables, the complete sensitivities coincide with cash-flow differentials, and also clarifies why they are called this.

When the proxy is inaccurate, that result does not hold. However, in that case, we argue that the difference is essentially an artifact from the approximations committed in the pre-simulation step, and therefore, is better left ignored in the interest of accuracy and stability.

Our argument is not indisputable. A counterargument, for instance, is that sensitivities computed in this way may fail to explain trading profit and loss (PnL). In practice, after a market move, values are recomputed, and the whole valuation process, including pre-simulations, is repeated with the new market variables. Hence, to correctly predict the change in value after a market move, we would need complete sensitivities.

This debate is open. While we understand and appreciate the PnL explain argument, we still believe that it is best to produce sensitivities as close as possible to the true theoretical ones, leaving the approximations and inaccuracies of the pre-simulation step out of the differentials. To some extent, and somewhat ironically, our estimation of the sensitivity may fail to explain actual differences because it is more accurate than our valuation. Does this mean that we must account for the valuation inaccuracy in sensitivities when we can easily ignore it? We think not. Readers who agree with us can trust cash-flow differentials and rely on path-wise AAD differences over the main simulations to produce them for free. Those who believe that PnL explain dictates the production of complete sensitivities will find a particularly efficient means of doing so in Section 4.

3.1.2 Multi callable instruments

The previous analysis extends to multiple exercises, In this case, we have (see Section 2):

$$V_0 = E\left[\sum_k \left(\prod_{T_p^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_p^{ex}}>0\right\}}\right) CF_k\right]$$

where $\widetilde{C}_{T_p^{ex}}$ is the regression proxy for the continuation value:

$$C_{T_p^{ex}} = E_{T_p^{ex}}\left[\sum_k \left(\prod_{T_p^{ex}<T_q^{ex}<T_k} 1_{\left\{C_{T_q^{ex}}>0\right\}}\right) CF_k\right]$$

Differentiating for some parameter a, we get:

$$\begin{aligned}\frac{\partial V_0}{\partial a} &= \underbrace{E\left[\sum_k \left(\prod_{T_p^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_p^{ex}}>0\right\}}\right)\frac{\partial}{\partial a}CF_k\right]}_{LHS} \\ &+ \underbrace{E\left[\sum_k CF_k\frac{\partial}{\partial a}\left(\prod_{T_p^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_p^{ex}}>0\right\}}\right)\right]}_{RHS}\end{aligned}$$

We see that the LHS is the cash-flow differential, but we have a bit more work to demonstrate that the RHS is null when the proxies are accurate. First, we have:

$$RHS = E\left[\sum_k CF_k \sum_{T_p^{ex}<T_k}\frac{\partial\widetilde{C}_{T_p^{ex}}}{\partial a}\delta\left(\widetilde{C}_{T_p^{ex}}\right)\prod_{T_q^{ex}<T_k, q\neq p} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\right]$$

Then, we swap the order of the sums so that the outermost sum operates on exercises, and get:

$$RHS = \sum_p r_p$$

where

$$\begin{aligned} & r_p \\ = \; & E\left[\prod_{T_q^{ex}<T_p^{ex}} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\frac{\partial\widetilde{C}_{T_p^{ex}}}{\partial a}\delta\left(\widetilde{C}_{T_p^{ex}}\right)\sum_k CF_k\prod_{T_p^{ex}<T_q^{ex}}^{T_q^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\right] \\ = \; & E\left[\prod_{T_q^{ex}<T_p^{ex}} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\frac{\partial\widetilde{C}_{T_p^{ex}}}{\partial a}\delta\left(\widetilde{C}_{T_p^{ex}}\right)E_{T_p^{ex}}\left(\sum_k CF_k\prod_{T_p^{ex}<T_q^{ex}}^{T_q^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\right)\right]\end{aligned}$$

and, modulo the accuracy of proxies:

$$\begin{aligned} & E_{T_p^{ex}}\left(\sum_k CF_k\prod_{T_p^{ex}<T_q^{ex}<T_k} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\right) \\ \approx \; & E_{T_p^{ex}}\left(\sum_k CF_k\prod_{T_p^{ex}<T_q^{ex}<T_k} 1_{\left\{C_{T_q^{ex}}>0\right\}}\right) \\ = \; & C_{T_p^{ex}}\end{aligned}$$

Hence:

$$r_p \approx E\left[\frac{\partial\widetilde{C}_{T_p^{ex}}}{\partial a}\delta\left(\widetilde{C}_{T_p^{ex}}\right)C_{T_p^{ex}}\prod_{T_q^{ex}<T_p^{ex}} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\right]$$

Applying Dirac mass integration rules:

$$r_p \approx \varphi_{\widetilde{C}_{T_p^{ex}}}E\left(\frac{\partial\widetilde{C}_{T_p^{ex}}}{\partial a}C_{T_p^{ex}}\prod_{T_q^{ex}<T_p^{ex}} 1_{\left\{\widetilde{C}_{T_q^{ex}}>0\right\}}\middle|\,\widetilde{C}_{T_p^{ex}}=0\right)$$

where the $\varphi_{\widetilde{C}_{T_p^{ex}}}$ are the densities for the proxies. When proxies are accurate, $E\left(\cdot C_{T_p^{ex}}\middle|\widetilde{C}_{T_p^{ex}}=0\right) = 0$, therefore all the r_p, and hence, the whole RHS, collapse to 0.

3.1.3 xVA without collateral

The same result also holds for xVA when the underlying netting set has no callable transactions. Path-dependent transactions fit (although they are subject to proxy inaccuracy, see Section 2). Contrarily to callable deals, the presence of path-dependent exotics in the netting set does not cause cash-flow differentials to differ from complete sensitivities.

We will see later that this result also holds for collateralized netting sets. For now, we focus on uncollateralized netting sets. Without loss of generality, we restrict ourselves to CVA, which is, by definition:

$$CVA = \sum_p \underbrace{E\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \max\left(0, V_{T_p}\right)\right]}_{=e(T_p)}$$

Using proxies for indicators as explained in Section 2, we get:

$$e\left(T_p\right) = E\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} 1_{\left\{\widetilde{V}_{T_p}>0\right\}} V_{T_p}\right]$$

where $\widetilde{V}_{T_p}$ is the regression proxy for the value of the netting set at that time:

$$V_{T_p} = E_{T_p}\left[\sum_{T_k>T_p} CF_k\right]$$

The expression for $e\left(T_p\right)$ satisfies the requirements for (2), hence (modulo proxy accuracy as usual):

$$\begin{aligned}
&\frac{\partial e\left(T_p\right)}{\partial a} \\
= \;& E\left\{1_{\left\{\widetilde{V}_{T_p}>0\right\}} \frac{\partial}{\partial a}\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} V_{T_p}\right]\right\} \\
= \;& E\left\{1_{\left\{\widetilde{V}_{T_p}>0\right\}} \frac{\partial}{\partial a}\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \sum_{T_k>T_p} CF_k\right]\right\}
\end{aligned}$$

and, trivially:

$$\begin{aligned}
&\frac{\partial CVA}{\partial a} \\
= \;& \sum_p \frac{\partial e\left(T_p\right)}{\partial a} \\
= \;& E\left[\sum_p 1_{\left\{\widetilde{V}_{T_p}>0\right\}} \frac{\partial}{\partial a}\left(\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}} \sum_{T_k>T_p} CF_k\right)\right] \\
= \;& E\left[\sum_p 1_{\left(\widetilde{V}_{T_p}>0\right\}} \left\{\begin{array}{c} \frac{\partial}{\partial a}\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}}\right] \sum_{T_k>T_p} CF_k \\ + \left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\leq\tau<T_p\}}\right] \frac{\partial}{\partial a}\left(\sum_{T_k>T_p} CF_k\right) \end{array}\right\}\right]
\end{aligned}$$

we now swap the order of the sums and obtain:

$$\begin{aligned} &= E\left[\sum_k \sum_{T_p<T_k} 1_{\left\{\widetilde{V}_{T_p}>0\right\}} \left(\begin{array}{l} \dfrac{\partial}{\partial a}\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}}\right] CF_k \\ +\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}}\right] \dfrac{\partial}{\partial a} CF_k \end{array} \right)\right] \\ &= E\left[\sum_k \sum_{T_p<T_k} 1_{\left\{\widetilde{V}_{T_p}>0\right\}} \frac{\partial}{\partial a}\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} CF_k\right]\right] \end{aligned}$$

If we compare this result to our estimate of the value from Section 2:

$$CVA = E\left[\sum_k \left(\sum_{T_p<T_k} 1_{\left\{\widetilde{V}_{T_p}>0\right\}} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} \right) CF_k\right]$$

it is clear that sensitivities coincides with the cash-flow differentials. The difference is that this time, it is not only the cash-flows that are differentiated, but the cash-flows *discounted for credit*. Crucially, the LSM proxies remain out of the differentials. The credit part of the discounting process must be differentiated, but not the LSM part. Intuitively, the CDS sensitivities of the CVA are a major part of its risk, and something that must be part of the risk report. That does not change the fact that AAD over main simulations with fixed proxies produces the correct sensitivities, including to credit, without a need to differentiate the LSM step. [21]

3.2 xVA with collateral

Collateral does not affect this result. In this case, the CVA is calculated with:

$$\begin{aligned} & CVA \\ &= E\left[\sum_p \left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} \max\left(0, V_{T_p} - V_{T_p^{mrg}}\right)\right]\right] \\ &\approx \sum_p E\left[\left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} 1_{\left\{\widetilde{V}_{T_p}-\widetilde{V}_{T_p^{mrg}}>0\right\}} \left(V_{T_p} - V_{T_p^{mrg}}\right)\right] \end{aligned}$$

The terms inside the sum satisfy the requirements for (2), and it follows that (modulo proxy accuracy) sensitivities coincide with cash-flow differentials. Note that in this case, the proxies on the secondary branch are fixed too, and since the only use of the branch is for proxies and indicators, branching does not affect differentiation in any way.

We investigated all cases of interest in our paper, except the case of an xVA on a netting set that itself contains callable transactions, and concluded that cash-flow differentials are sufficient in all these cases. This is a particularly nice result, because it means that we don't need to differentiate or instrument LSM at all, and that sensitivities essentially come for free. What about the remaining case, that of an xVA on a netting set that contains callable transactions?

[21] The credit indicator may need smoothing so it is correctly differentiated. Note that practical implementations usually don't simulate default times but forward default probabilities, which removes the credit indicator and avoids the need for a special treatment.

3.3 When is differentiation *necessary*?

In the case of an xVA on a netting set that itself contains callable transactions, cash-flow differentials are *not* sufficient. In this case, the contribution of proxies to sensitivities is real and material and cannot be ignored. We show why this is the case, and derive a formula for what is missed. That simply identifies the problem, leaving the solution to Section 4.

xVA with callable transactions in the netting set

To examine the problem in its simplest form, we consider an exposure on a one time callable transaction. With a full xVA on a set of transactions including multi callable trades, the problem is not different and only makes exposition more complicated.

We consider a netting set consisting in a transaction that pays the cash flows $(CF_k)_{T_k \leq T^{ex}}$ before the exercise date and the cash flows $(CCF_k)_{T_k > T^{ex}}$ after the exercise date unless it is terminated there (we double the C in the notation to emphasize that these cash-flows may be called).

Exposure before exercise We start with the easy case where we compute an exposure for that netting set, for a date T prior to exercise. In this case, we have:

$$\begin{aligned} & e(T) \\ = \; & E\left[\max\left(0, V_T\right)\right] \\ \approx \; & E\left[1_{\left\{\widetilde{V}_T > 0\right\}}\left(\sum_{T < T_k \leq T^{ex}} CF_k + 1_{\left\{\widetilde{V}_{T^{ex}} > 0\right\}} \sum_{T_k > T^{ex}} CCF_k\right)\right] \end{aligned}$$

where

$$\widetilde{V}_{T^{ex}} \approx V_{T^{ex}} = E_{T^{ex}}\left[\sum_{T_k > T^{ex}} CCF_k\right]$$

is the proxy to the PV of the transaction immediately after the exercise date providing it is not terminated. This is also the future PV of all subsequent cash flows. We call this proxy the *exercise proxy*. And the *exposure proxy* $\widetilde{V}_T$ is the proxy to the value of the transaction on the exposure date:

$$\begin{aligned} V_T \; &= \; E_T\left[\sum_{T < T_k \leq T^{ex}} CF_k + 1_{\{V_{T^{ex}} > 0\}} \sum_{T_k > T^{ex}} CCF_k\right] \\ &\approx \; E_T\left[\sum_{T < T_k \leq T^{ex}} CF_k + 1_{\left\{\widetilde{V}_{T^{ex}} > 0\right\}} \sum_{T_k > T^{ex}} CCF_k\right] \end{aligned}$$

During pre-simulations, $\widetilde{V}_{T^{ex}}$ is computed first, so that the exposure proxy incorporates the possibility of a subsequent exercise. As explained in Section 2 for multi callable deals, proxies are computed recursively, in reverse chronological order.

We immediately note that:

$$e(T) = E\left[1_{\left\{\widetilde{V}_T > 0\right\}} V_T\right]$$

This expression satisfies the requirements of (2) so the exposure proxy has no contribution to $\frac{\partial e(T)}{\partial a}$. Further, we also have:

$$\begin{aligned} & e\left(T\right) \\ = \; & E\left[1_{\left\{\widetilde{V}_T>0\right\}}\left(\sum_{T<T_k\leq T^{ex}} CF_k + 1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right)\right] \\ = \; & E\left[1_{\left\{\widetilde{V}_T>0\right\}} \sum_{T<T_k\leq T^{ex}} CF_k\right] + E\left[1_{\left\{\widetilde{V}_T>0\right\}} 1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right] \end{aligned}$$

The exercise proxy only appears in the right hand side of the addition, which is also of the form required by (2), hence, it too bears no contribution to $\frac{\partial e(T)}{\partial a}$.

Therefore, for exposures prior to exercise dates, our previous conclusion holds. Cash-flow derivatives with fixed proxies produce the correct complete sensitivities.

Exposure after exercise That conclusion, however, does not hold for exposure dates *after* the exercise. In this case,

$$e\left(T\right) = E\left[1_{\left\{\widetilde{V}_T>0\right\}} 1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} \sum_{T_k>T} CCF_k\right] = E\left[1_{\left\{\widetilde{V}_T>0\right\}} 1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_T\right]$$

It appears that (2) is satisfied for the exposure proxy (which therefore bears no contribution to sensitivities) but not for the exercise proxy. The exercise proxy appears in an indicator, of which the exercise value is *not* a factor. This is a Nikkei exercised on SnP type of situation. That means that we are authorized to compute differentials with fixed exposure proxy but not exercise proxy:

$$\frac{\partial e\left(T\right)}{\partial a} = E\left[1_{\left\{\widetilde{V}_T>0\right\}} \frac{\partial\left(1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_T\right)}{\partial a}\right]$$

We can rewrite this expression as:

$$\begin{aligned} \frac{\partial e\left(T\right)}{\partial a} \; = \; & E\left[1_{\left\{\widetilde{V}_T>0\right\}} \frac{\partial\left(1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right)}{\partial a}\right] \\ - \; & E\left[1_{\left\{\widetilde{V}_T>0\right\}} \frac{\partial}{\partial a}\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\left(V_{T^{ex}} - V_T\right)\right]\right] \end{aligned}$$

We may apply (2) to the left hand side of the difference so we can fix the exercise proxy there, which yields:

$$\begin{aligned}
&\frac{\partial e\left(T\right)}{\partial a}\\
=\;& E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial V_{T^{ex}}}{\partial a}\right]\\
-\;& E\left[1_{\left\{\widetilde{V}_T>0\right\}}\frac{\partial}{\partial a}\left[1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\left(V_{T^{ex}}-V_T\right)\right]\right]\\
=\;& E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial V_T}{\partial a}\right]+E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial\left(V_{T^{ex}}-V_T\right)}{\partial a}\right]\\
-\;& E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial\left(V_{T^{ex}}-V_T\right)}{\partial a}\right]-E\left[1_{\left\{\widetilde{V}_T>0\right\}}\left(V_{T^{ex}}-V_T\right)\frac{\partial\widetilde{V}_{T^{ex}}}{\partial a}\delta\left(\widetilde{V}_{T^{ex}}\right)\right]\\
=\;& E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial V_T}{\partial a}\right]-E\left[1_{\left\{\widetilde{V}_T>0\right\}}\left(V_{T^{ex}}-V_T\right)\frac{\partial\widetilde{V}_{T^{ex}}}{\partial a}\delta\left(\widetilde{V}_{T^{ex}}\right)\right]
\end{aligned}$$

Injecting the formulas for future values, we get:

$$\begin{aligned}
\frac{\partial e\left(T\right)}{\partial a} &= E\left[1_{\left\{\widetilde{V}_T>0\right\}}1_{\left\{\widetilde{V}_{T^{ex}}>0\right\}}\frac{\partial}{\partial a}\sum_{T_k>T}CCF_k\right]\\
&- E\left[\delta\left(\widetilde{V}_{T^{ex}}\right)\frac{\partial\widetilde{V}_{T^{ex}}}{\partial a}1_{\left\{\widetilde{V}_T>0\right\}}\sum_{T^{ex}<T_k\leq T}CCF_k\right]
\end{aligned}$$

The left hand side is the cash-flow differential. The *non-zero* right hand side is the error committed, the part of the sensitivity that is not captured by cash-flow differentiation.

We can see that the *exposure proxy* is not responsible for the error. It is only the *exercise proxy* that has a non-zero contribution to sensitivities. The intuition here is that exercise refers to the PV of all subsequent cash flows, whereas the exposure computation is only interested in those ulterior to the exposure date. Hence the cash-flows in between: $\sum_{T_{ex}<T_k\leq T}CCF_k$ generate an error, a difference between the correct sensitivities and the cash-flow differentials. This error is equal to:

$$\begin{aligned}
&err\\
=\;& -E\left[\delta\left(\widetilde{V}_{T^{ex}}\right)\frac{\partial\widetilde{V}_{T^{ex}}}{\partial a}1_{\left\{\widetilde{V}_T>0\right\}}\sum_{T^{ex}<T_k\leq T}CCF_k\right]\\
=\;& -\varphi_{\widetilde{V}_{T^{ex}}}\left(0\right)\Pr\left(\widetilde{V}_T>0\middle|\,\widetilde{V}_{T^{ex}}=0\right)\\
& E\left[\frac{\partial\widetilde{V}_{T^{ex}}}{\partial a}\sum_{T^{ex}<T_k\leq T}CCF_k\middle|\,\widetilde{V}_T>0,\widetilde{V}_{T^{ex}}=0\right]
\end{aligned}$$

While this term could itself be estimated with Monte-Carlo simulations, this is not practical for xVA (which are integrals of exposures) over large netting sets (that may contain multiple trades with multiple exercises each). It is best to keep this formula in mind for the estimate of the magnitude of the error, but estimate sensitivities by conducting a complete differentiation as explained next.

4 LSM and MC Differentiation

4.1 Check-pointing and main simulations

4.1.1 Introduction to AAD and check-pointing

Differentiation through AAD is arguably the strongest addition to Computational Finance since the Global Crisis. Differentials that previously took multiple valuations while bumping all relevant market variables one by one (up to thousands of variables in the case of xVA) may now be computed together with the value for the cost of about 4 to 8 valuations. We list textbooks, articles and presentations in our bibliography that readers may use to get themselves acquainted with this major technology. In short, AAD records every mathematical operation involved in a calculation, together with its result, into a *tape*[22]. Once the calculation is complete, AAD applies the chain rule over the tape in the reverse order from the calculation, in order to *back-propagate* the sensitivities (also called *adjoints*) of the final result to all the intermediary steps, including all inputs. Before this *adjoint propagation* takes place, the tape is *seeded* with the adjoint of the final result, which is its sensitivity to itself, which is trivially 1.

More formally, a calculation is decomposed into elementary steps: addition, multiplication, exponential, ... $x_i = f_i\left(x_j, j \in E_i\right), 1 \leq i \leq N$ where x is the result of the operation f and E is the set of the indices of its arguments. Arguments must be computed before they are used, so their indices are lower than the index of the operation that uses them. For instance, when f_i a multiplication, E_i is the set of the 2 indices less than i of the arguments being multiplied. Initial inputs have no arguments, and, ultimately, we compute the sensitivities of the final result x_N to these. We define adjoints (the partial derivatives)

$$\bar{x}_i \equiv \frac{\partial x_N}{\partial x_i}$$

Then, obviously, $\bar{x}_N = \frac{\partial x_N}{\partial x_N} = 1$ and from the chain rule:

$$\bar{x}_j = \sum_{\{i|j\in E_i\}} \frac{\partial x_i}{\partial x_j} \bar{x}_i$$

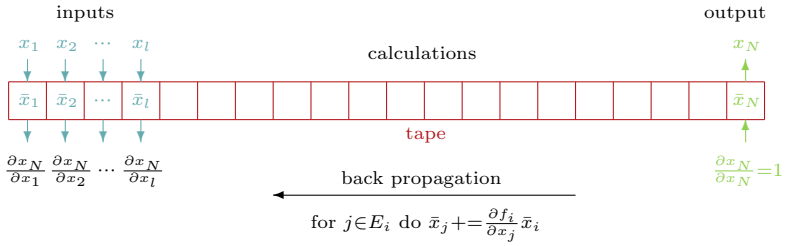
which provides an algorithm for the computation of all adjoints, last to first:

$$\begin{cases} \text{set } \bar{x}_N = 1, \bar{x}_{N-1} = \ldots = \bar{x}_1 = 0 \\ \text{for } i = N-1, ..., 1 \text{ do:} \qquad \text{for } j \in E_i \text{ do } \bar{x}_j \mathrel{+}= \dfrac{\partial f_i}{\partial x_j} \bar{x}_i \end{cases}$$

It is easy to see that adjoints are computed in constant time over all inputs, which is the whole point, and that the $\frac{\partial f_i}{\partial x_j}$ of elementary operations are known and analytic. This back-propagation is conducted after the calculation is complete and its tape recorded as shown in the figure below.

The biggest obstacle to the systematic implementation of AAD, besides the skill and effort required for the correct instrumentation of the code, is the vast amount of memory consumed by this technique, that is around 5GB per second per core in the valuation process. In addition, even though modern computers routinely include 128 to 256GB of cheap memory, it is essential for performance

[22] This is best implemented with *operator overloading*, where all calculations are performed with a custom number type, for which all mathematical operations are overloaded so they are put on tape when they are performed.



that the tape should be contained within the L3 cache of modern processors, typically up to 50MB. Fortunately, a technique known as *check-pointing* allows to reduce memory consumption by orders of magnitude by decomposing the valuation process into successive steps, and conducting AAD differentiation sequentially (in the reverse order) over the different steps. The maximum memory requirement decreases to 5GB/sec/core *over the longest step alone* instead of the whole computation. Hence, provided that the valuation process can be divided in a large number of small steps, memory consumption may be significantly subdued.

In the context of standard Monte-Carlo simulations, check-pointing is implemented trivially by computing derivatives path by path. Differentials are produced independently (and possibly in parallel) over each path, and averaged in the end. Memory consumption is limited to one path, which, typically, takes less than ten milliseconds on a single core, even for xVA on large netting sets. Hence, the worst case memory consumption is limited to a very manageable 50MB. This is known as path-wise AAD differentiation, is now a best practice among advanced Investment Banks, and is explained in detail in multiple textbooks, papers, theses and professional presentations, including some of our own [17], [25], [27] and [15].

We have seen in section 3 that, in the context of LSM, such standard path-wise differentials with fixed proxies, also called cash flow differentials, actually produce in many cases, on their own and without the need for further work, the correct sensitivities. In particular, AAD always implicitly fixes indicators when computing differentials, hence, the differentiation of LSM enabled simulations essentially comes for free. However, in those cases where it is necessary or desirable to differentiate the whole valuation process, including the production of the regression coefficients, some further work is required. In particular, it is not trivial to decompose the LSM step, because the computation of regression coefficients requires all pre-simulations to be generated simultaneously and results to be produced in an entangled way. We will need to somehow disentangle these steps in order to efficiently implement check-pointing through the differentiation of LSM.

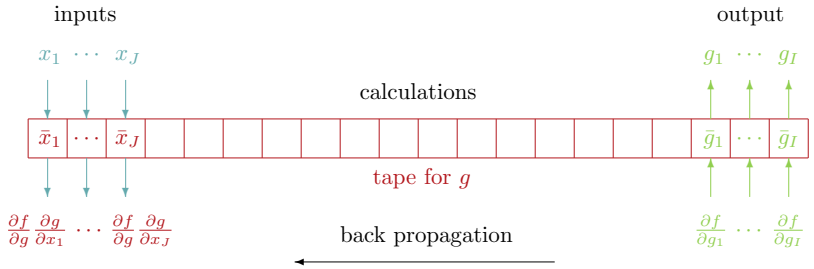
The algorithm we present here is essentially a specialization of the check-point methodology. In the same way that AAD conducts differentiation by propagating derivatives -or *adjoints* over every mathematical operation in the reverse order from valuation, check-pointing consists in the propagation of derivatives over the steps involved in a valuation process, also in the reverse order.

To quickly introduce check-pointing, say we want to differentiate the real valued function $h(x) = f(x, g(x))$ where f is a real valued function (in our context, the main simulations, that produce a value out of model parameters and regression coefficients), and g is a vector valued function (in our context, the LSM step, itself divided into pre-simulations and regressions, that produces regression coefficients out of model parameters).

Where f is too long to allow a memory and cache efficient differentiation within a self contained AAD instrumentation, we can use the chain rule $\frac{\partial h}{\partial x} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$, first compute the sensitivities

of f, that is $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial g}$, next compute the Jakobian $\frac{\partial g}{\partial x}$ and apply the matrix product $\frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$ to complete the computation.

The problem is that Jakobians computation with AAD is linear in the number of results, and matrix product is cubic. Check-pointing avoids these costly computations by directly computing $\frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$ given $\frac{\partial f}{\partial g}$ in constant time. How exactly this is achieved is explained in detail in AAD literature. In short, we differentiate f first, computing $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial g}$. In our case, this is conducted with path-wise AAD over the main simulations. Then, we evaluate g with AAD instrumentation, building the tape of all mathematical operations involved. Here comes check-pointing: we *seed* this tape with the adjoints for the results, that is, the set of the $\frac{\partial f}{\partial g}$, which are already known at this point, and compute all adjoints backwards through the tape, which produces $\frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$ in constant time. Note that the Jakobian of g is never computed and remains unavailable, although the end result incorporates it correctly. Note that in order to evaluate f first, the result of g must be known. Therefore, in order to implement check-pointing, the entire calculation must be performed first without any AAD instrumentation, and the results of intermediary steps must be stored so they can be reused as inputs to the following steps. This adds an overhead of 1 valuation, which is more than fine given the order of magnitude we gain from skipping Jakobian calculations. We describe check-pointing in further detail in what follows, in the context of LSM.



The remarkable performance of the differentiation algorithm we describe next comes in part from a systematic recourse to check-pointing, which avoids the costly computation and application of Jakobians. We also strive to conduct as much as possible of the computation *path-wise*, which limits memory usage, ensures cache efficiency, and permits to easily conduct computations for different paths in parallel. Finally, we compute check-pointed differentials through the SVD regression analytically.

In the rest of the document, we detail the steps involved in this differentiation, and show how performance is achieved, starting with the main simulations, which we called f in the example above. The subsection after that shows how to check-point the results into the LSM step (which we called g), and in particular, through its two main components, which are pre-simulations and regressions (differentiated in the reverse order, regressions first). All computations are detailed in the simple case with a single proxy (like a one time callable), and the final subsection shows how to generalize to the case of multiple interrelated proxies (like a multi callable).

4.1.2 Differentiating through main simulations

Check-pointing consists in computing differentials of successive steps in a valuation algorithm *in the reverse order*. Hence, the main simulations, which are the last step in the valuation process, are also the first step in the differentiation process.

This subsection investigates the differentiation of main simulations, under the assumption that LSM proxies are only used in indicators, as described in section 2.

Main simulations produce the value V of a transaction, book, netting set or xVA, given some parameters a, and our ultimate goal is compute differentials of V to these parameters a. Main simulations produce values *given* the regression coefficients β that were produced within a previous LSM step. Further, $V = \frac{1}{N}\sum_n V^n$ is the average PV across simulations, hence:

$$\frac{\partial V}{\partial a} = \frac{1}{N}\sum_n \frac{\partial V^n}{\partial a}$$

We choose, for now, to express the PV for simulation number n as a function of the parameters a and the proxies $\widetilde{C}^n$ computed in that path. That allows us to isolate sensitivities to parameters over main simulations with fixed proxies:

$$V^n = \nu^n\left(a, \widetilde{C}^n\right)$$

Note that V^n is also a function of the random numbers used for the generation of that path number n. That, however, does not matter, since we compute differentials *with fixed random numbers*. The random numbers, therefore, act as an exogenous input, one that does not participate in differentials. In AAD lingo, we call them *inactive*, and in coding lingo, we say that they are not instrumented.

Evidently:

$$\frac{\partial V^n}{\partial a} = \frac{\partial \nu^n}{\partial a} + \frac{\partial \nu^n}{\partial \widetilde{C}^n}\frac{\partial \widetilde{C}^n}{\partial a}$$

Here, $\frac{\partial \nu^n}{\partial a}$ are the path-wise differentials with fixed proxies, also called cash-flow differentials. Further, since proxies are only used in indicators:

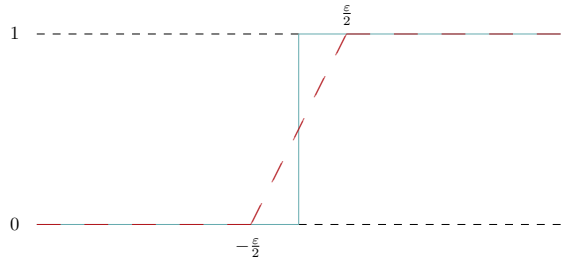
$$\frac{\partial \nu^n}{\partial \widetilde{C}^n} = \frac{\partial \nu^n}{\partial 1_{\{\widetilde{C}^n>0\}}}\frac{\partial 1_{\{\widetilde{C}^n>0\}}}{\partial \widetilde{C}^n}$$

AAD does not differentiate control flow, hence, $\frac{\partial 1_{\{\widetilde{C}^n>0\}}}{\partial \widetilde{C}^n} = 0$ and it follows that $\frac{\partial V^n}{\partial a} = \frac{\partial \nu^n}{\partial a}$. AAD produces sensitivities that are simply cash-flow differentials with fixed proxies. Section 3 demonstrated that this is indeed correct in many cases. However, in those cases where a complete sensitivity is required or desired, we need the main simulations to produce correct path-wise sensitivities $\frac{\partial \nu^n}{\partial \widetilde{C}^n}$ to the proxies. That means that we must differentiate the main simulations correctly through indicators.

Indicator functions are discontinuous, therefore it is well known that Monte-Carlo estimators of their sensitivities are, at best, unstable, even with traditional finite differences. With AAD, they are simply 0. It is therefore a strong market practice to *smooth* these indicator functions, for instance for digital payoffs, barrier options or other transactions that involve discontinuous payoffs. [28] offers an in-depth investigation of such smoothing, and, in particular, shows how to automatically embed it in cash-flow scripting using *fuzzy logic*. For our purpose here, it suffices to note that in order for the main simulations to produce accurate, unbiased, stable values for $\frac{\partial 1_{\{\widetilde{C}^n>0\}}}{\partial \widetilde{C}^n}$, we must use *smooth indicators* in place of indicators[23]:

$$1_{\{x>0\}} \approx \left(\frac{1}{\varepsilon}x + \frac{1}{2}\right)1_{\left\{x>-\frac{\varepsilon}{2}\right\}} - \left(\frac{1}{\varepsilon}x - \frac{1}{2}\right)1_{\left\{x>\frac{\varepsilon}{2}\right\}}$$

[23] which is just like using call spreads to approximate digital options.



Note that this point is also made in [12]. For the purpose of differentiation, all indicators must be smoothed, something we will assume is the case from there on and in all that follows.

Provided that the main simulations use smooth indicators in place of indicators, their path-wise AAD differentiation produces the cash-flow derivatives $\frac{\partial \nu^n}{\partial a}$ (with fixed proxies) *and the proxy derivatives* $\frac{\partial \nu^n}{\partial \widetilde{C}^n}$ (with fixed parameters).

In practice, the main simulations are fed with the regression coefficients β produced in a prior pre-simulation LSM step, and produce proxies within simulations, as a function of these regressions coefficients, as well as the simulated regression variables, which themselves depend on the parameters a, and the (irrelevant) random numbers. Hence, in fine, main simulations that are fed with the β and use proxies in *smoothed* indicators to produce, through path-wise AAD, differentials to parameters a (including differentials of proxies with fixed regression coefficients) and to the regression coefficients β:

$$V = \frac{1}{N}\sum_n V^n = \frac{1}{N}\sum_n v^n(a,\beta) = v(a,\beta)$$

where v^n produces the PV for path number n out of the parameters a and the regression coefficients β. Path-wise AAD produces the sensitivities:

$$\frac{\partial v}{\partial a} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial a} \qquad \frac{\partial v}{\partial \beta} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial \beta}$$

where we note that $\frac{\partial v}{\partial a}$ are the cash-flow differentials with *fixed regression coefficients* and no longer fixed proxies.

Further, the β are produced in a prior LSM step, as a function of the parameters a. Hence, our (ultimate) goal is to compute:

$$\frac{\partial V}{\partial a} = \underbrace{\frac{\partial v}{\partial a}}_{\substack{CF \text{ sensitivity} \\ \text{fixed } \beta \\ \text{main sim}}} + \underbrace{\underbrace{\frac{\partial v}{\partial \beta}}_{\substack{\beta \text{ sensitivity} \\ \text{main sim}}} \underbrace{\frac{\partial \beta}{\partial a}}_{\text{from LSM}}}_{\text{contribution of the LSM step}}$$

Path-wise AAD over the main simulations (provided they use proxies in smoothed indicators) provides $\frac{\partial v}{\partial a}$, as well as $\frac{\partial v}{\partial \beta}$. The last step to close the computation consists in turning $\frac{\partial v}{\partial \beta}$ into $\frac{\partial v}{\partial \beta}\frac{\partial \beta}{\partial a}$ by check-pointing it into the LSM step. Once this calculation is complete, we will have successfully computed $\frac{\partial V}{\partial a}$.

4.2 Differentiating through LSM: single proxy

The rest of the document is dedicated to the calculation of $\frac{\partial v}{\partial \beta}\frac{\partial \beta}{\partial a}$ given the sensitivity to regression coefficients $\frac{\partial v}{\partial \beta}$.

We start with a one-time callable transaction so as to explain the method in a context that is not complicated by the presence of multiple proxies. Note that the case of an exposure (on a netting set that does not contain callable trades) also involves a single proxy. Hence, this case is also covered in this simplified context. Further, an xVA is essentially an integral of exposures against credit. Even though xVA requires multiple proxies for the different exposure dates, the proxies are computed *independently on one another* in the sense that the production of the regression coefficients for any proxy does not depend on other regression coefficients or proxies. On the contrary, for multi callable deals, or xVA on callable trades, the computation of regression coefficients for a given proxy depends on proxies for subsequent dates, introducing a complication that we investigate and resolve in a dedicated final subsection. For now, we note that the simplified context of our analysis here is sufficient for one time callables, but also for xVA, as long as the underlying netting set has no callable deals. In this case, we only need to repeat the steps for each proxy and sum-up the resulting differentials.

We remind that LSM produces regression coefficients in 2 steps: first, pre-simulations produce the regression variables X and PVs Y for all paths[24], given parameters a (and irrelevant random numbers). Then, the regression step (that is best conducted as an SVD regression for stability, accuracy, but also efficiency, as explained in section 2) produces the regression coefficients β from X and Y alone. We remind that X is the matrix of all regression variables (in columns) for all pre-simulations (in rows), while Y is the vector of the PVs to the call/exposure date of the subsequent cash-flows, as computed within the corresponding pre-simulation.

We apply check-pointing to this process, going over the second step (regressions) first, and the first step (pre-simulations) next. In Figure 1 we have illustrated the check-point'ed process. On the left we have the valuation. The parameters a are input to the pre-simulation from where we produce the regression variables and PVs X^m, Y^m, path-wise. The entire matrix X and vector Y are then input in the (SVD) regression to compute the regression coefficients β. Finally, we use the parameters a and the regression coefficients β in the main simulation to calculate the value of the transaction, book or regulatory quantity.

On the right the calculation order is reversed to calculate differentials. First, we differentiate through main simulations, computing $\frac{\partial v}{\partial a}, \frac{\partial v}{\partial \beta}$ with path-wise AAD as explained in the previous subsection. The resulting $\frac{\partial v}{\partial \beta}$ is then check-pointed into the differentiation of the regression/SVD which can be done analytically or in the case of standard regression with standard AAD. We detail this step in section 4.2.1. Its outputs are all the sensitivities wrt the regression variables and PVs from each pre-simulation path. We check-point those into the pre-simulation step, where we use path-wise AAD differentials to compute the contribution of each pre-simulation path to the sensitivities. Finally, we add the resulting contributions together to compute the overall contribution of LSM to sensitivities, which we add to the cash-flow sensitivities computed through main simulations.

[24]Here, we are talking about *pre-simulation paths*, which are different from the main simulation paths.

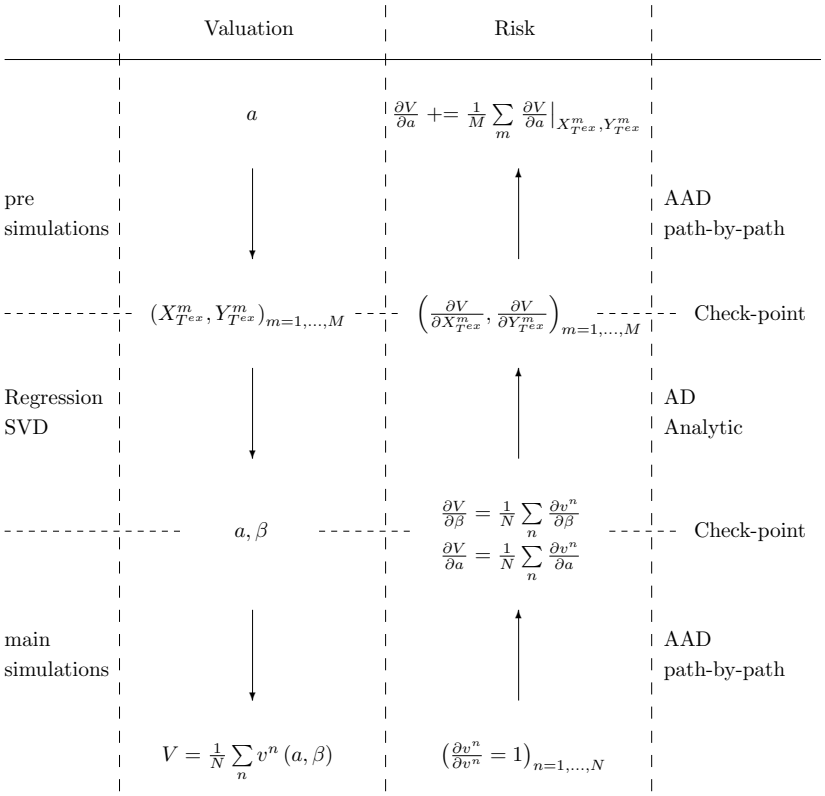


Figure 1: Check-point'ed AD process

4.2.1 Differentiating through regressions

In this subsection, we focus on the differentiation of the regression step, the last step in LSM calculation, hence the first one in the LSM differential. During valuation, the regression step turns the matrix X and the vector Y of all pre-simulated regression varibales and PVs, into a vector of regression coefficients β. During differentiation, we compute the differential of this step by turning $\frac{\partial V}{\partial \beta}$ (which we know from the differentials of main simulations) into the full set of $\frac{\partial V}{\partial X}$ and $\frac{\partial V}{\partial Y}$.

We first consider the case of a standard regression, where this step is trivial, and then the case of an SVD regression, where it is much more complicated. However, as has been mentioned in section 2, SVD is an essential piece for an efficient implementation of LSM, hence, its differentiation is a required step, and the standard case is only given for reference. In the following we will follow [16] and [21] so for details we refer to these papers. Define the derivatives of the final result V wrt each element of β by $\bar{\beta}$. Anything with a bar $\bar{x}$ denotes an adjoint, that is $\frac{\partial V}{\partial x}$.

We will be using the following result from basic adjoint caculus. Let A, B, C be matrices and f a matrix function such that $C = f(A, B)$ then by definition and the chain rule

$$\begin{aligned} \operatorname{Tr}\left(\bar{C}^T dC\right) &\equiv \sum_{i,j} \bar{C}_{ij} dC_{ij} \\ &= dV \\ &= \sum_{i,j} \bar{A}_{ij} dA_{ij} + \bar{B}_{ij} dB_{ij} \\ &= \operatorname{Tr}\left(\bar{A}^T dA\right) + \operatorname{Tr}\left(\bar{B}^T dB\right) \end{aligned} \tag{3}$$

where $\operatorname{Tr} A = \sum_i A_{ii}$ is the trace of a square matrix. Since we also have

$$dC = \frac{\partial f}{\partial A} dA + \frac{\partial f}{\partial B} dB$$

we get the relations

$$\bar{A} = \left(\frac{\partial f}{\partial A}\right)^T \bar{C} \qquad\qquad \bar{B} = \left(\frac{\partial f}{\partial B}\right)^T \bar{C}$$

which is a recipe for working backwards from outputs to inputs.

Standard regression We remind the standard regression formula $\beta = \left(X^T X\right)^{-1} \left(X^T Y\right)$. An application of the previous recipe gives us $\bar{X}, \bar{Y}$ as a function of $\bar{\beta}$. First,

$$d\beta = \left(X^T X\right)^{-1} \left(dX^T Y + X^T dY - dX^T X\beta - X^T dX\beta\right)$$

Second, we use (3) to find

$$\begin{aligned} & \operatorname{Tr}\left(\bar{\beta}^T d\beta\right) \\ = \; & \operatorname{Tr}\left(\bar{\beta}^T \left(X^T X\right)^{-1} \left(dX^T Y + X^T dY - dX^T X\beta - X^T dX\beta\right)\right) \\ = \; & \operatorname{Tr}\left(\left(Y - X\beta\right) \bar{\beta}^T \left(X^T X\right)^{-1} dX^T - \beta\bar{\beta}^T \left(X^T X\right)^{-1} X^T dX\right) \\ & + \operatorname{Tr}\left(\bar{\beta}^T \left(X^T X\right)^{-1} X^T dY\right) \\ = \; & \operatorname{Tr}\left(\left(\left(X^T X\right)^{-1} \bar{\beta} \left(Y^T - \beta^T X^T\right) - \beta\bar{\beta}^T \left(X^T X\right)^{-1} X^T\right) dX\right) \\ & + \operatorname{Tr}\left(\bar{\beta}^T \left(X^T X\right)^{-1} X^T dY\right) \end{aligned}$$

which gives us the result

$$\begin{aligned}\bar{X} &= (Y - X\beta)\,\bar{\beta}^T \left(X^T X\right)^{-1} - X\left(X^T X\right)^{-1}\bar{\beta}\beta^T \\ \bar{Y} &= X\left(X^T X\right)^{-1}\bar{\beta}\end{aligned}$$

Notice, that this computation can also be performed with standard check-pointed AAD by taping the calculation $\beta = \left(X^T X\right)^{-1}\left(X^T Y\right)$, seeding the tape with $\bar{\beta}$, and run the tape backwards to produce $\bar{X}$ and $\bar{Y}$.

SVD regression Unfortunately, the standard regression approach will not work in general as explained in section 2. We instead use the SVD approach. Consider the SVD $X = VDU^T$. Then, $\beta = UD\Sigma V^T Y$.

It would therefore be natural to compute $\bar{V}, \bar{D}, \bar{\Sigma}, \bar{U}, \bar{Y}$ as functions of $\bar{\beta}$ and check-point that result into SVD to compute $\bar{X}$ as a function of $\bar{V}, \bar{D}, \bar{\Sigma}, \bar{U}$. The problem is that the SVD procedure is *not* differentiable in general. Even though the whole *regression* is differentiable, each step of it is not. The problem is that when X has repeated singular values the differentials have singularities. Consider the example

$$X = \begin{bmatrix} a & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix}$$

then for $a = 1$ the singular values are $\frac{3}{4}, \frac{3}{4}, \frac{3}{2}$. In Figure 2 we have shown an example where the differential $\frac{\partial U_{22}}{\partial X_{21}}$ has a singularity at $a = 1$. In this example the regression is clearly differentiable

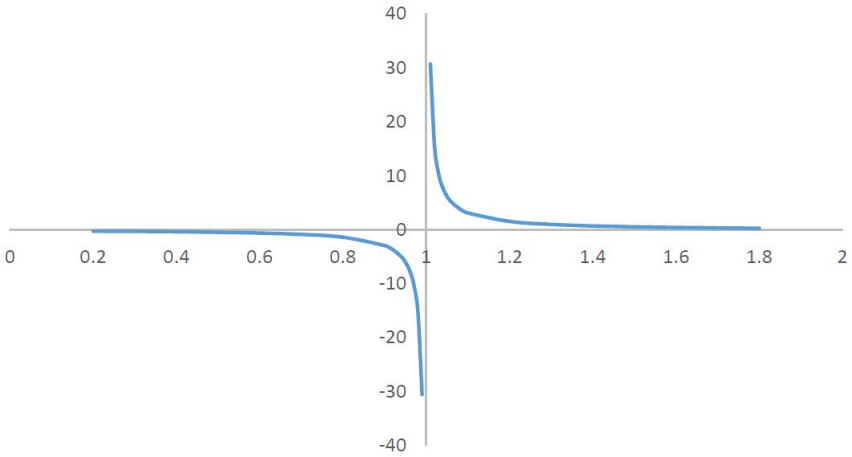


Figure 2: $\frac{\partial U_{22}}{\partial X_{21}}$ as a function of a.

even though the SVD process is not.

In [21] the entire SVD regression process is differentiated and the results are reported here. First, [21] differentiates the SVD regression assuming distinct singular values by using an extended version of results from [16]. This can be done by following the adjoint calculus recipe to compute $\bar{X}, \bar{Y}$ as functions of $\bar{\beta}$. Second, it is verified that these functions have no singularities for common singular values. Hence, by continuity these functions are also valid for common singular values.

Finally, the differential of the SVD regression is as follows. We use the SVD of X that excludes singular values close to zero:

$$X = VDU^T$$

where $V \in \mathbb{R}^{M\times\Omega}, D \in \mathbb{R}^{\Omega\times\Omega}, U \in \mathbb{R}^{J+1\times\Omega}$ with $D_{jj} > \epsilon$ for every j. Then,

$$\beta = UD\Sigma V^T Y$$

where $\Sigma \in \mathbb{R}^{\Omega\times\Omega}$ is a diagonal matrix with $\Sigma_{jj} = \frac{1}{D_{jj}^2+\lambda^2}$. Define the matrices $W \in \mathbb{R}^{\Omega\times\Omega}, \Gamma \in \mathbb{R}^{\Omega\times\Omega}$ as

$$\begin{aligned} W &= U^T\bar{\beta}Y^TV \\ \Gamma &= \Sigma\left(\lambda^2 W^T - DWD\right)\Sigma \end{aligned}$$

then the results we need are

$$\begin{aligned} \bar{X} &= V\Gamma U^T + \left(I_M - VV^T\right)Y\bar{\beta}^T U\Sigma U^T + V\Sigma V^T Y\bar{\beta}^T\left(I_{J+1} - UU^T\right) \\ \bar{Y} &= VD\Sigma U^T\bar{\beta} \\ \bar{\lambda} &= -2\lambda\beta^T U\Sigma U^T\bar{\beta} \end{aligned}$$

Notice, that we also get a sensitivity to the Tikhonov parameter (for free). As mentioned in Section 2 the Tikhonov parameter may depend on X, Y. We will not go into this in detail just mention that this can easily be done by AAD of the function $\lambda(X, Y)$ with input $\bar{\lambda}$. This is another use of check-pointing.

4.2.2 Differentiating through pre-simulations

We now focus on check-pointing the matrix $\frac{\partial V}{\partial X}$ and the vector $\frac{\partial V}{\partial Y}$ resulting from the above step, into the pre-simulations that produce the regression variables X and PVs Y, for all (pre-simulation) paths, given the parameters a (and irrelevant random numbers), so as to produce the contribution of the LSM step to the sensitivities to a (which we then add to the sensitivities coming from the main simulations to complete the computation). Importantly, we show how this step, which is the most time consuming step in the production and differentiation of LSM, can be conducted *path-wise* over pre-simulations, which limits memory consumption and gives the opportunity to conduct this step in parallel.

The differentiation of the regression step produced $\bar{X}, \bar{Y}$ where the m'th row is $\frac{\partial V}{\partial X^m}, \frac{\partial V}{\partial Y^m}$, the sensitivities of the final result to the regression variables and PVs generated through the pre-simulations for the path number m. X^m and Y^m are the products of the pre-simulation number m, which means that they are a function of the parameters a and the (irrelevant for differentiation) random numbers used for the generation of that path. We call g^m this function:

$$\begin{bmatrix} X^m & Y^m \end{bmatrix} = g^m(a)$$

Then the contribution of the LSM step to $\frac{\partial V}{\partial a}$ is

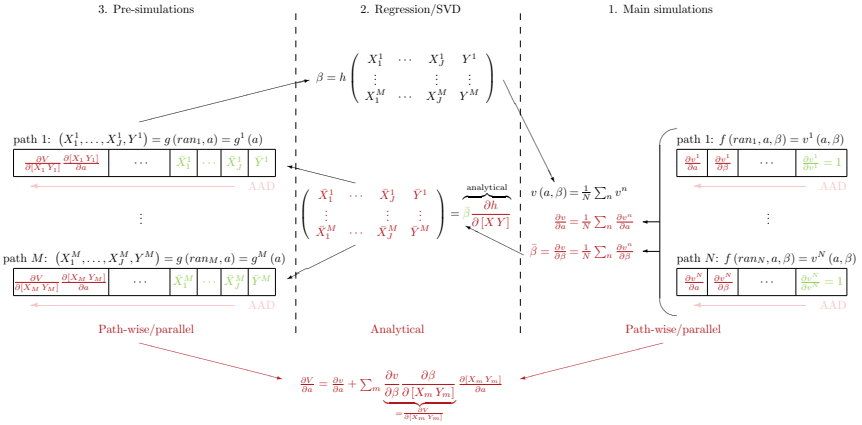
$$\sum_m \begin{bmatrix} \frac{\partial V}{\partial X^m} & \frac{\partial V}{\partial Y^m} \end{bmatrix} \frac{\partial g^m}{\partial a}$$

And the final result we are after is:

$$\underbrace{\underbrace{\underbrace{\frac{\partial v}{\partial a}}_{\substack{\text{main sim}\\ \text{fixed beta}}} + \sum_m \underbrace{\underbrace{\left[\frac{\partial V}{\partial X^m} \quad \frac{\partial V}{\partial Y^m}\right]}_{\text{pre sim path } m} \frac{\partial g^m}{\partial a}}_{\text{full contribution of the LSM step}}}_{\text{complete sensitivity}}$$

All we need to do is compute $\left[\frac{\partial V}{\partial X^m} \quad \frac{\partial V}{\partial Y^m}\right] \frac{\partial g^m}{\partial a}$ given $\frac{\partial V}{\partial X^m}$ and $\frac{\partial V}{\partial Y^m}$, which we have from the regression step. This computation can be made independently path by path, which makes it suitable for parallel processing.

For every path, the computation may be performed with standard check-pointed AAD. That means that we run the generation of the path number m (the function g^m) with AAD instrumentation, producing the tape that records the sequence of mathematical operations that constitutes g^m, from inputs a to outputs X^m and Y^m. At that point, we *seed* that tape with the adjoints of the outputs $\frac{\partial V}{\partial X^m}$, $\frac{\partial V}{\partial Y^m}$ and run backwards to produce the required adjoints $\frac{\partial V}{\partial a}$.



This concludes the production of sensitivities in the case where we have one single proxy. The 2 successive steps described in this subsection: regression and pre-simulation, effectively incorporate the contribution of the LSM step to the total sensitivity $\frac{\partial V}{\partial a}$ given $\frac{\partial v}{\partial \beta}$ that results from the differentiation of the main simulation, as described in the previous subsection.

Our algorithm is particularly efficient, both in terms of speed and memory. The regression step is essentially differentiated analytically, while the time consuming pre-simulation step is differentiated path-wise. In addition, the systematic use of check-points conducts this computation without the need to produce expensive Jakobians, instead seeding AAD tapes with previously derived adjoints, which is conducted in constant time and increases efficiency by orders of magnitude.

These steps are sufficient in the cases where a single proxy is needed, like one-time callable exotics, or the exposure on a netting set that does not contain callable trades. They are also sufficient in the cases where multiple proxies are needed, but are derived independently of one another, like for xVA, which requires one proxy for each exposure date, without any form of interdependence between the proxies. On the contrary, in the case of multi-callable trades, or exposures/xVA on callable trades, regression coefficients for different proxies are computed recursively, last to first, and the production of each proxy relies on the regression coefficients for subsequent dates, which must

have been produced in a prior step, hence the recursion. For those cases, we must introduce another check-pointed step, which we describe next.

4.3 Extension to multiple interrelated proxies

We left for the end the case where multiple proxies are interrelated as in multi-callables or xVA on callables. This is not a particularly difficult extension. We only postponed it so we could explain the more complex parts of this section in a simplified context.

We first consider a twice callable trade, or, equivalently, a T_1 exposure on a trade that is callable at $T_2 > T_1$. In both cases, the derivation of β_2 is exactly the same as in the one-time call case, but the derivation of β_1 now depends on β_2.

Remember from 2.1.2 that β_1 is derived by regression/SVD from the matrix X_1 of regression variables pre-simulated for T_1 across all paths, and the vector Y_1 of PVs by path:

$$Y_1^m = \sum_{T_1 < T_p \leq T_2} CF_p^m + \max\left(0, \tilde{C}_2^m\right)$$

We now know better so we dont use the proxy other than in indicators (which we also always smooth) so that the actual formula for Y becomes:

$$Y_1^m = \sum_{T_1 < T_p \leq T_2} CF_p^m + 1_{\left\{\tilde{C}_2^m > 0\right\}} \sum_{T_2 < T_p} CF_p^m$$

Replacing the proxy by its formula:

$$Y_1^m = \sum_{T_1 < T_p \leq T_2} CF_p^m + 1_{\left\{\beta_2 \cdot X_2^m > 0\right\}} \sum_{T_2 < T_p} CF_p^m$$

Notice, it appears explicitly that the production of Y_1, and hence, that of β_1, depends on β_2. Recall from 4.2.2 that in the simple case, $\beta_1 = h\left(X_1, Y_2\right)$ (where h is the regression/SVD) and $\left\{X^m, Y^m\right\} = g^m\left(a\right)$ so ultimately $\beta_1 = f_1\left(a\right)$. Well, in the more general case, $\left\{X_1^m, Y_1^m\right\} = g^m\left(a, \beta_2\right)$ so $\beta_1 = f_1\left(a, \beta_2\right)$. This is why the betas must be computed recursively, starting with the last one (which is the only one that does not depend on any other and is derived as in the simple case) and ending with the first one. This is all described in 2.1.2 except we now use a more a precise estimation for Y_1 by using proxies in indicators only, something we did not know back in 2.1.2.

Unsurprisingly, when it comes to computing differentials, the order is reversed and we start with $\frac{\partial V}{\partial \beta_1}$. Note that the differentials of the main simulations are unchanged, so we obtain $\frac{\partial v}{\partial \beta_1}$ and $\frac{\partial v}{\partial \beta_2}$ from the main simulations just as before, and as explained in 4.1.2.

Following the exact same steps described in 4.2, we obtain $\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial a} = \frac{\partial v}{\partial \beta_1}\frac{\partial f_1}{\partial a}$, which is the contribution of the T_1-LSM step to the total sensitivity with fixed β_2, but, this time, we also produce an additional $\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2} = \frac{\partial v}{\partial \beta_1}\frac{\partial f_1}{\partial \beta_2}$.

At this point, we need to update the sensitivity of the end result to β_2 by incorporating its contribution through β_1:

$$\frac{\partial V}{\partial \beta_2} = \frac{\partial v}{\partial \beta_2} + \frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2} = \frac{\partial v}{\partial \beta_2} + \frac{\partial v}{\partial \beta_1}\frac{\partial f_1}{\partial \beta_2}$$

We repeat the steps from 4.2 with β_2, which itself only depends on a: $\beta_2 = f_2(a)$. The only difference is that we now seed this calculation with the total $\frac{\partial V}{\partial \beta_2}$, not only the part that comes from the main simulations $\frac{\partial v}{\partial \beta_2}$, but also incorporating the contribution $\frac{\partial v}{\partial \beta_1}\frac{\partial f_1}{\partial \beta_2}$ from the β_1 step.

Finally, our total sensitivity to a is the sum of the sensitivities computed over the main simulations with fixed betas, the contribution from β_1 with fixed β_2, and the contribution of β_2 including the part coming from β_1:

$$\frac{\partial V}{\partial a} = \underbrace{\frac{\partial v}{\partial a}}_{\substack{\text{main sim}\\ \text{fixed betas}}} + \underbrace{\frac{\partial v}{\partial \beta_1}}_{\text{main sim}}\underbrace{\frac{\partial f_1}{\partial a}}_{T_1\text{-LSM}} + \left(\underbrace{\frac{\partial v}{\partial \beta_2}}_{\text{main sim}} + \underbrace{\frac{\partial v}{\partial \beta_1}\frac{\partial f_1}{\partial \beta_2}}_{T_1\text{-LSM}}\right)\underbrace{\frac{\partial \beta_2}{\partial a}}_{T_2\text{-LSM}}$$

In the general case, we add to the sensitivity, the contributions of the regression coefficients for the dates $T_1, \ldots, T_L$, recursively, starting with β_1 and ending with β_L, in the reverse order to when those coefficients were computed. Each coefficient β_l is a function of a and of the coefficients that come after T_l, that is:

$$\beta_l = f_l(a, \beta_{l+1}, \ldots, \beta_L)$$

and

$$\beta_L = f_L(a)$$

Following the steps from 4.2 we derive $\frac{\partial V}{\partial \beta_l}\frac{\partial f_l}{\partial a}$ and add this contribution to the sensitivities, and also $\frac{\partial V}{\partial \beta_l}\frac{\partial f_l}{\partial \beta_{l+1}}, \ldots, \frac{\partial V}{\partial \beta_l}\frac{\partial f_l}{\partial \beta_L}$ and update $\frac{\partial V}{\partial \beta_{l+1}}, \ldots, \frac{\partial V}{\partial \beta_L}$ accordingly, adding the contribution from β_l to the sensitivity to subsequent coefficients, which we in turn differentiate and update the total sensitivity to a and to the coefficients after that, and so on, recursively. This backward recursive reasoning is at the heart of the check-pointed AAD technology, and we see well why the same recursion that is used for the production of the coefficients is also used for their differentiation, only in the reverse order.

Note that the check-pointing described here does not slow down the algorithm in any way. Obviously, the steps described in 4.2 must be repeated for all dates in any case. For a general implementation, it is therefore best to always differentiate recursively, first to last.

5 Conclusion

We investigated the application of LSM to the valuation of callable exotics and xVA, focusing on an effective and practical implementation. In particular, we showed how POI (using LSM proxies only in indicators) reduced dependency on proxies, and how a systematic application of POI, in particular to xVA, allowed us to reformulate the xVA problem in a computationally friendly way. We showed that to value a callable exotic or an xVA, we essentially value the underlying cash-flow while applying a path-dependent discounting. The resulting algorithm is accurate because we limited dependency on proxies with POI. It is efficient because we only need to simulate the discount process while valuing cash-flows. All this may be conducted path-wise and in parallel. We also extended these results to collateral with the help of so-called branching Monte-Carlo simulations. Finally, we showed how the SVD alternative to regressions helps not only stability but also efficiency.

Moving on to AAD differentiation, we showed how a systematic use of check-pointing, not only limited memory consumption and allowed cache efficiency, but also avoided the costly computation

and application of Jakobians. In addition, just like valuation, the differences of the main simulations and pre-simulations could be conducted path-wise, further limiting memory consumption and optimizing cache efficiency, while begging for parallel processing across the simulated paths.

The resulting algorithms use the best of known and new technologies to value and differentiate xVA with maximum performance, even on light hardware. As Andreasen likes to put it, "Differentiate xVA on an iPad Mini."

The outstanding issue left for future research is a more elegant way to handle path dependency for xVA with netting sets including many exotics. We mentioned either the brute force solution to use different regression sets for each path dependency or to ignore the path dependency. With POI we were able to significantly reduce the dependency on the proxy hence for most netting sets we will propose the latter approach to ignore the path dependency. However, none of these solutions is fully satisfactory, and that challenge remains open.

Appendix A SVD

SVD offers a stable alternative for the regression step to the classical $\beta = \left(X^T X\right)^{-1}\left(X^T Y\right)$ for the solution of:

$$\beta = \arg\min_b \|Xb - Y\|^2$$

The solution $\beta = \left(X^T X\right)^{-1}\left(X^T Y\right)$ assumes that $\left(X^T X\right)^{-1}$ exists. The matrix $\left(X^T X\right)$ is the covariance matrix between the regression variables, which are, in the context of LSM, functions $(f_j)_{0\leq j\leq J}$ of the state variables $(S_i)_{1\leq i\leq I}$, hence, if some of the basis functions are linearly dependent (or close to), the covariance matrix is singular and not invertible.

To resolve this issue and produce a stable solution even in the presence of collinearity, we define the singular value decomposition of $X = \hat{V}\hat{D}\hat{U}^T$ where $\hat{V} \in \mathbb{R}^{M\times J+1}$ with orthogonal columns such that $\hat{V}^T\hat{V} = I_{J+1}$, $\hat{U} \in \mathbb{R}^{J+1\times J+1}$ is a square orthogonal matrix such that $\hat{U}^T\hat{U} = \hat{U}\hat{U}^T = I_{J+1}$ and $\hat{D} \in \mathbb{R}^{J+1\times J+1}$ is a diagonal matrix with the singular values. With this decomposition we can write the coefficients

$$\begin{aligned}
\beta &= \left(X^T X\right)^{-1} X^T Y \\
&= \left(\hat{V}\hat{D}\hat{U}^T\hat{U}\hat{D}\hat{V}^T\right)^{-1}\hat{V}\hat{D}\hat{U}^T Y \\
&= \hat{V}\hat{D}^{-2}\hat{V}^T\hat{V}\hat{D}\hat{U}^T Y \\
&= \hat{U}\hat{D}^{-1}\hat{V}^T Y
\end{aligned}$$

Note that the inverse is well defined if and only if the singular values are all different from 0. An algorithm for calculating the SVD matrices, $\hat{U}, \hat{D}, \hat{V}$, can be found in [26].

If one or more of the singular values are 0 then the SVD identifies in which directions multi collinearity occurs. Assume that we have Ω singular values which are not null, i.e. Ω is the effective number of basis functions. Then, we divide our decomposition in the following way

$$\hat{V} = \begin{bmatrix} V & V^\perp \end{bmatrix}, \hat{D} = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}, \hat{U} = \begin{bmatrix} U & U^\perp \end{bmatrix}$$

where $V \in \mathbb{R}^{M\times\Omega}$, $U \in \mathbb{R}^{J+1\times\Omega}$, $D \in \mathbb{R}^{\Omega\times\Omega}$ Also, define

$$\alpha = \begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix} \equiv \begin{bmatrix} U^T\beta \\ \left(U^{\perp}\right)^T\beta\end{bmatrix} = \hat{U}^T\beta$$

Then $X\hat{U} = \begin{bmatrix} VD & 0\end{bmatrix}$ and we have the transformed regression formula $Y = X\beta = X\hat{U}\hat{U}^T\beta = \hat{V}\hat{D}\alpha = VD\alpha_1$. Hence, transforming the basis $(f_j)_{j=0,\dots,J}$ with the orthogonal basis $\hat{U}$ identifies the linear dependencies. We can find the coefficients α in the new basis. Redefine the problem as

$$\alpha_1 = \arg\min_a \|Y - VDa\|^2$$

hence, we get the solution

$$\alpha = \begin{bmatrix} \alpha_1 \\ 0 \end{bmatrix} = \begin{bmatrix} D^{-1}V^TY \\ 0 \end{bmatrix}$$

which we know exists. In the original basis we get the coefficients

$$\beta = \hat{U}\alpha = UD^{-1}V^TY$$

Note that even though we have null singular values, we are still able to solve the problem and invert $\hat{D}$. We introduce an SVD cut parameter $\epsilon > 0$ and set any singular value less than ϵ to 0, hence, $\hat{D}^{-1}_{jj} = \frac{1_{\{\hat{D}_{jj}>\epsilon\}}}{\hat{D}_{jj}}$.

In order to further stabilize the regression, we use the so-called Tikhonov regularization, that assigns a preference to solutions with small norms. Instead of the original minimization problem, we solve:

$$\beta = \arg\min_b \|Xb - Y\|^2 + \lambda\|b\|^2$$

where $\lambda > 0$ is some constant that may depend on X, Y. The solution to this problem is

$$\beta = \hat{U}\hat{D}\hat{\Sigma}\hat{V}^TY = UD\Sigma V^TY$$

where $\hat{\Sigma} \in \mathbb{R}^{J+1\times J+1}$ is a diagonal matrix with $\hat{\Sigma}_{jj} = \frac{1}{\hat{D}^2_{jj}+\lambda^2}1_{\left\{\hat{D}_{jj}>\epsilon\right\}}$. Also

$$\hat{\Sigma} = \begin{bmatrix}\Sigma & 0 \\ 0 & 0\end{bmatrix}$$

with $\Sigma \in \mathbb{R}^{\Omega\times\Omega}$.

In conclusion, the solution for the regression is written in the preferred SVD form as:

$$\beta = UD\Sigma V^TY$$

where $X = VDU^T$ is the SVD of X excluded for singular values distant from null by less than some *SVD cut* ϵ, $\Sigma_{jj} = \frac{1}{D^2_{jj}+\lambda^2}$, and λ is the Tikhonovs parameter (small norm preference).

References

[1] J. Andreasen. Branching processes for xva. Global Derivatives, 2015, Presentation.

[2] J. Andreasen. Cva on an ipad mini. Global Derivatives, 2014, Presentation.

[3] J. Andreasen. Cva on an ipad mini, part 1: Intro. Aarhus Kwant Factory PhD Course, 2014.

[4] J. Andreasen. Cva on an ipad mini, part 2: The beast. Aarhus Kwant Factory PhD Course, 2014.

[5] J. Andreasen. Cva on an ipad mini, part 3: Xva algorithms. Aarhus Kwant Factory PhD Course, 2014.

[6] J. Andreasen. Cva on an ipad mini, part 4: Cutting the it edge. Aarhus Kwant Factory PhD Course, 2014.

[7] J. Andreasen and O. Scavenius. Bermuda monte-carlo. Bank of America working paper, 2006.

[8] A. Antonov. Algorithmic differentiation for callable exotics. *SSRN preprint*, 2016.

[9] A. Antonov and D. Brecher. Exposure and cva for large portfolios of vanilla swaps: The thin-out optimization. *SSRN preprint*, 2012.

[10] A. Antonov, S. Issakov, M. Konikov, A. McClelland, and S. Mechkov. Pv and xva greeks for callable exotics by algorithmic differentiation. *SSRN preprint*, 2016.

[11] A. Brace, D. Gatarek, and M. Musiela. The market model of interest rate dynamics. *Mathematical Finance*, 7(2):127–154, 1997.

[12] L. Capriotti, Y. Jiang, and A. Macrina. Aad and least squares monte carlo: Fast bermudan-style options and xva greeks. Available at SSRN: https://ssrn.com/abstract=2842631 or http://dx.doi.org/10.2139/ssrn.2842631, 2016.

[13] J. F. Carriere. Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics*, 19(1):19–30, 1996.

[14] D. Duffie and M. Huang. Swap rates and credit quality. *The Journal of Finance*, 51(3):921–949, 1996.

[15] H.-J. Flyger, B. Huge, and A. Savine. Practical implementation of aad for derivatives risk management, xva and rwa. Global Derivatives, 2015.

[16] M. Giles. An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation. An extended version of a paper that appeared in the proceedings of AD2008, the 5th International Conference on Automatic Differentiation.

[17] M. Giles and P. Glasserman. Smoking adjoints: Fast evaluation of greeks in monte carlo calculations. *Risk*, 2006.

[18] P. Glasserman and B. Yu. Number of paths versus number of basis functions in american option pricing. *Ann. Appl. Probab.*, 14(4):2090–2119, 11 2004.

[19] D. Heath, R. Jarrow, and A. Morton. Bond pricing and the term structure of interest rates: A new methodology for contingent claim valuation. *Ecnonometrica*, 60(1):77–105, 1992.

[20] P. Henry-Labordere. Counterparty risk valuation: A marked branching diffusion approach. *The Review of Financial Studies*, 2012. Available at SSRN: https://ssrn.com/abstract=1995503 or http://dx.doi.org/10.2139/ssrn.1995503.

[21] B. Huge. Ad of matrix calculations - regression and cholesky. Danske Bank working paper.

[22] D. Lando. On cox processes and credit risky securities. *Review of Derivatives Research*, 2(2):99–120, 1998.

[23] F. A. Longstaff and E. S. Schwartz. Valuing american options by simulation: A simple least-square approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

[24] H. McKean. Application of brownian motion to the equation of kolmogorov-petrovskii-piskunov. *Communications on Pure and Applied Mathematics*, 28(3):323–331, 1975.

[25] U. Naumann. *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. SIAM, 2012.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[27] A. Savine. Automatic differentiation for financial derivatives. Global Derivatives, 2014.

[28] A. Savine. Stabilize risks of discontinuous payoffs with fuzzy logic. Global Derivatives, 2015.