LSM Reloaded - Differentiate xVA on your iPad Mini

Presentation · May 2017

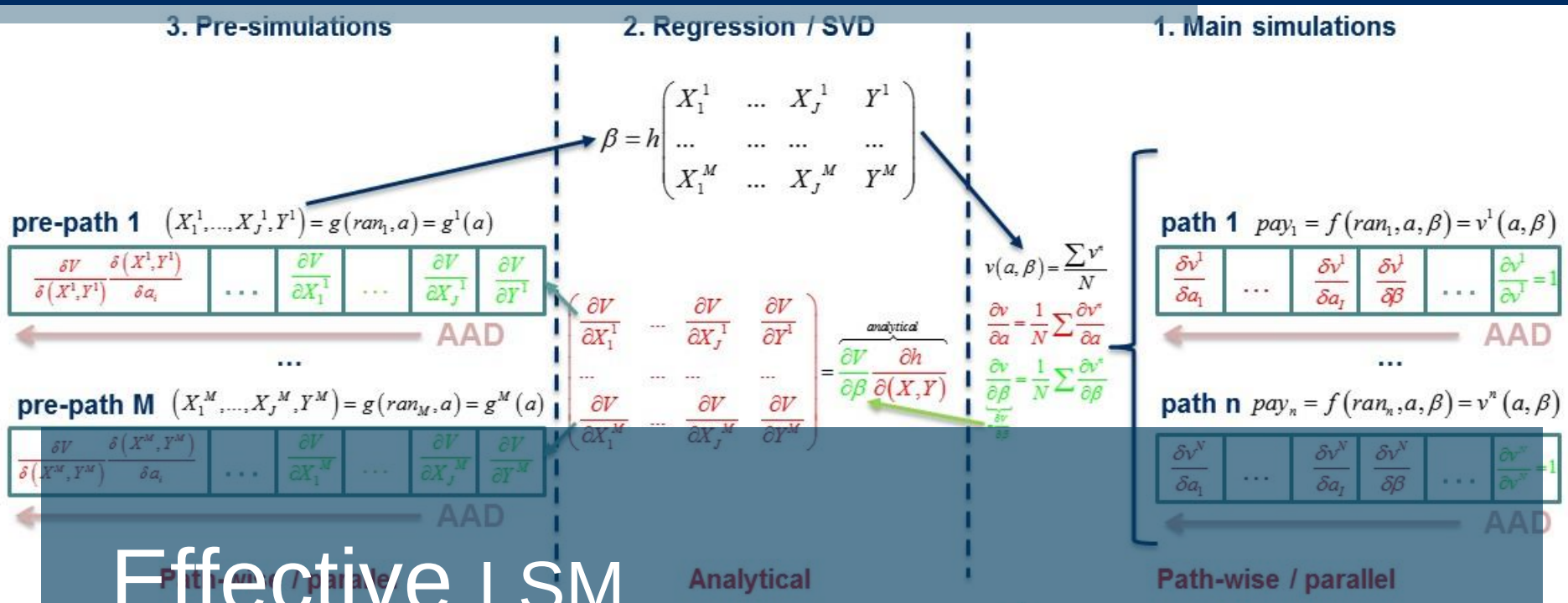
CITATIONS
0

READS
442

2 authors:

Brian Norsk Huge
Danske Bank
23 PUBLICATIONS **130** CITATIONS
SEE PROFILE

Antoine Savine
DANSKE BANK
20 PUBLICATIONS **11** CITATIONS
SEE PROFILE



Effective LSM

High performance computation and AAD differentials
For exotics, xVA and other regulatory calculations

Brian Huge and Antoine Savine

Paper: LSM Reloaded

- This talk is essentially a summary of our paper: LSM Reloaded

- References include
  - Obviously Longstaff-Schwarz, 2001
  - Referencing Carriere, 1996
  - Numerous works by Giles & Glasserman
  - Jesper Andreasen, 2014-2016
  - Antonov, 2016 and Capriotti, 2016
  - Henry-Labordere, 2012 and McKean, 1975
  - And also Dupire, Jaeckel, Naumann
  - And the compulsory Black, Scholes, Heath, Jarrow, Morton, Brace, Gatarek, Musiela, Press, Teukolsky, Vetterling, Flannery, etc.

LSM Reloaded
Differentiate xVA on your iPad Mini

Brian Huge
brian.huge@danskebank.dk

Antoine Savine
antoine@asavine.com

May 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 Giles and Glasserman's pioneering publication 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.

xVA

- CVA: loss due to default of a counterparty who owes positive PV
  - Formula $CVA = E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} \max\left(0, V_{T_p}\right)\right]$
  - Means we are short a (contingent to default) put on the entire netting set
  - Hence, CVA is a *hybrid option* that is valued and hedged under risk-neutral measure
  - Models are hybrid models with joint arbitrage free RN dynamics on all the relevant market variables
  - Implementation is Monte-Carlo due to dimensionality
  - Other xVA with similar definitions: DVA, FVA, and with some adjustments: RWA, KVA
- Main challenge for computation: "payoff" refers to future PV of the netting set
  - How to compute that? Nested simulations? Approximations?
  - Notice the similarity with a callable transaction: $V_0 = E\left[\left(V_{T^{ex}}\right)^+\right]$
  - Golden standard: just like for callable transactions, use *LSM proxies*
- Many other challenges
  - How to simulate in decent time a large number of market variables, factors and cash-flows?
  - How to aggregate and compress all cash-flows from all transactions in a netting set?

LSM: Longstaff-Schwartz Method

- Published in 2001 to resolve callable exotics with Monte-Carlo simulations
  - Efficient lattice methods impractical in high dimension or with model/product path dependence
  - Fall back to Monte-Carlo simulations
  - MC simulates state variables not transaction PVs ➔ cannot account for early exercise
  - LSM solution: build, then use **regression proxies** to PVs as functions of state variables
  - Best practice and key component in derivative systems
- Adopted as a key component for xVA and other regulatory calculations
  - xVA also needs future PVs within simulations ➔ LSM applies naturally
  - Alternatives: nested simulations, approx. ➔ Vastly inferior for performance, accuracy, practicality
- We introduce a particularly effective approach to xVA with LSM
  - Responsible in part for our In House System of the Year 2015 award
  - Also applies (with adjustments) to RWA, kVA, even FRTB, PRIIP, etc.

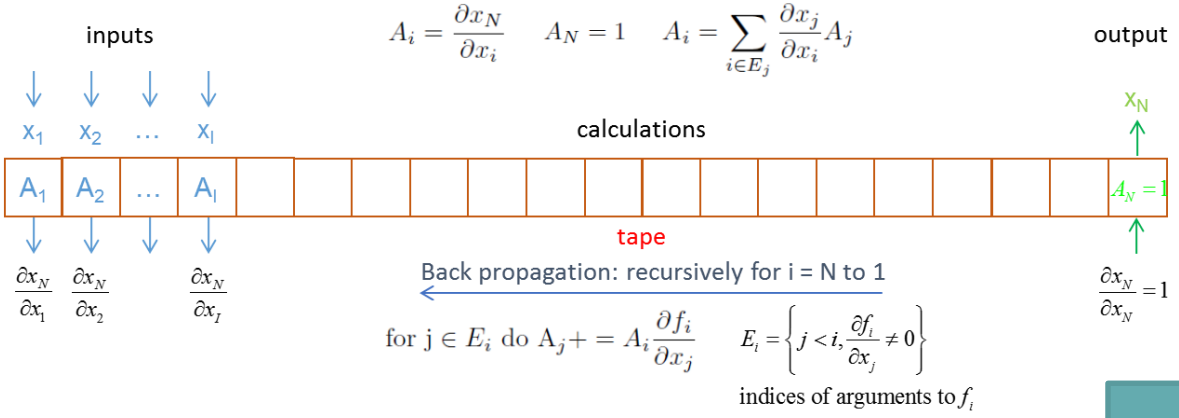
AAD: Automatic Algorithmic Differentiation

AAD: a vastly superior technology for the computation of sensitivities

- Classical sensitivity computation: finite difference or “bump and re-run”
  - Run valuation repeatedly, bumping inputs one by one (up to thousands for xVA)
- AAD: compute all sensitivities together with valuation **in constant time**
  - Total CPU cost: 4-8 valuations
  - Parallelism / Multi-threading friendly
- How? Record operations and results that constitute valuation on a *tape*
- Apply chain rule backwards over the tape to compute all sensitivities
- Cost is constant in the number of sensitivities ➔ viable for xVA

$$x_i = f_i\left(x_j, j \in E_i\right), \underbrace{E_i = \left\{ j < i, \frac{\partial f_i}{\partial x_j} \neq 0 \right\}}_{\text{arguments of } f_i}$$

$$A_i \equiv \frac{\partial x_N}{\partial x_i}, A_N = 1, A_i = \sum_{i \in E_j} \frac{\partial f_j}{\partial x_i} A_j$$



Check-pointing

- AAD memory consumption
  - Tape records all mathematical operations
  - Consumes around 5GB/sec/core in valuation
  - For maximum performance, must be contained within ~50MB (L3 cache)
  - Limited to valuations under 1/100 sec?
- Solution for standard Monte-Carlo: path-wise differentials
  - Compute AAD differentials one path at a time and then average
  - One path always takes (way) less than 1/100 sec
  - Bonus: paths can easily be differentiated in parallel
- General solution: check-pointing
  - Divide valuation in a (large) number of (small) steps
  - Differentiate steps recursively with AAD, last to first, applying chain rule to results
  - Like AAD but at the level of valuation steps rather than elementary mathematical operations
  - We illustrate check-pointing in the context of LSM

LSM Differentiation

- **Valuation**: run LSM to produce regression coefficients, then run standard MC
- **Differentiation**: why even a challenge? Why not just run AAD over LSM + MC?
  - Valuation takes up to minutes for large xVA ➔ Insane memory consumption, cache inefficiency
  - With MC we have now classical path-wise differentials
  - LSM step alone may take up to a minute ➔ Must be check-pointed
  - Check-pointing LSM is not trivial: all paths simulated at once with entangled results
  - Parts of LSM are not trivially differentiable
- And yet it is crucial to get it right
  - AAD is the only viable way to compute regulatory sensitivities
  - LSM is best practice for regulatory calculations
- LSM differentiation with AAD a hot topic with high stakes

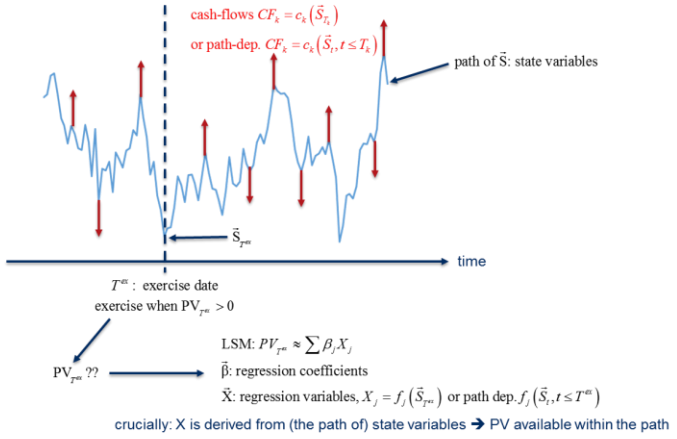
  The current subject of heated debates and vast amounts of publications and talks
- We present an original solution featuring high performance and low memory

Summary

- Effective LSM implementation
  - LSM refresher for callable exotics
  - POI: Proxies Only in Indicators
  - **Take out 1:**
    xVA reformulation for an efficient computation

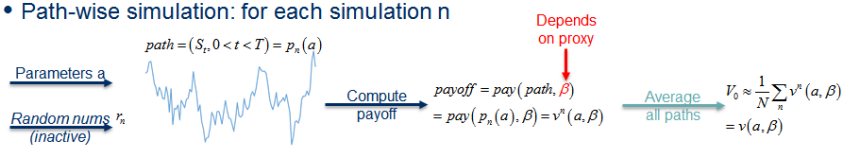
- Effective LSM differentiation
  - Cash-flow differentials: path-wise, fixed proxies
  - Are CF differentials enough or what do we miss?
  - **Take out 2: LSM differentiation algorithm fast, memory efficient, mainly parallel**
  - Extension to multiple interrelated proxies

LSM refresher: one proxy

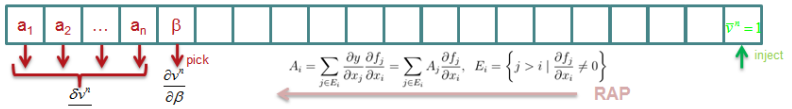


Main simulation differentials: AAD path-wise

- Path-wise simulation: for each simulation n



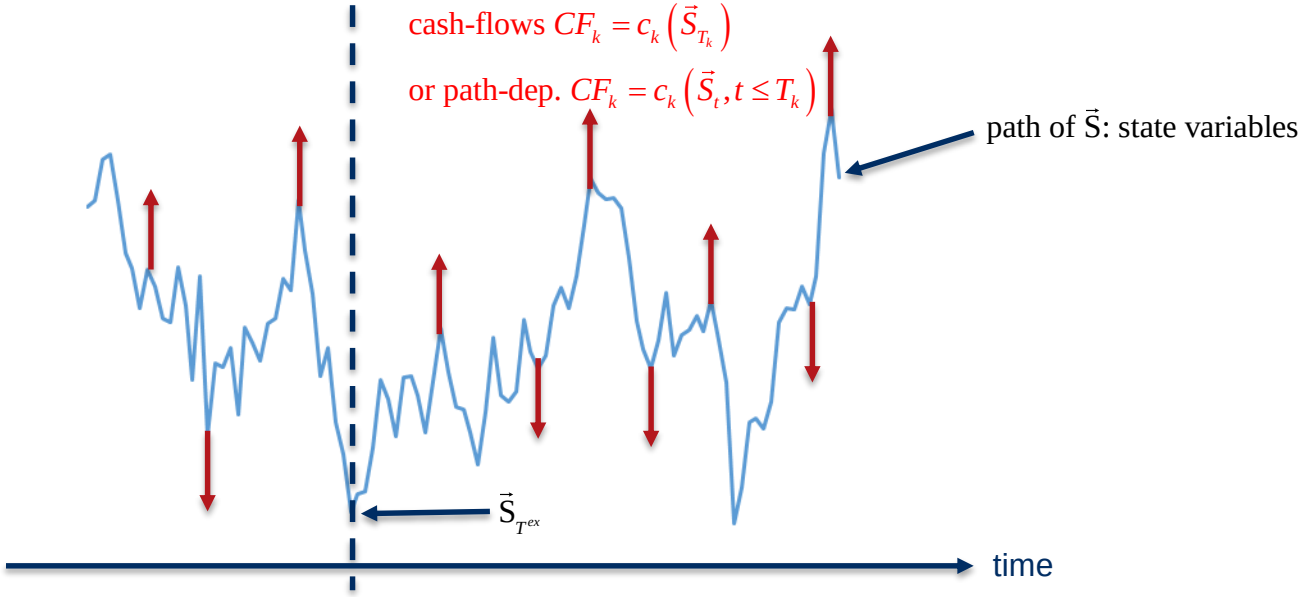
- AAD differential (see e.g. Savine, 2014):
  - Compute $v^n(a, \beta)$ with AAD instrumentation, build adjoint tape
  - Seed result with $\bar{v}^n = 1$, apply chain rule (reverse adjoint propagation), pick $\bar{a} = \frac{\partial v^n}{\partial a}, \bar{\beta} = \frac{\partial v^n}{\partial \beta}$



- Average $\frac{\partial v}{\partial a} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial a}, \frac{\partial v}{\partial \beta} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial \beta}$

Effective LSM implementation

LSM refresher: one proxy



T^{ex} : exercise date

exercise when $PV_{T^{ex}} > 0$

$PV_{T^{ex}}$?? → LSM: $PV_{T^{ex}} \approx \sum \beta_j X_j$

$\vec{\beta}$: regression coefficients

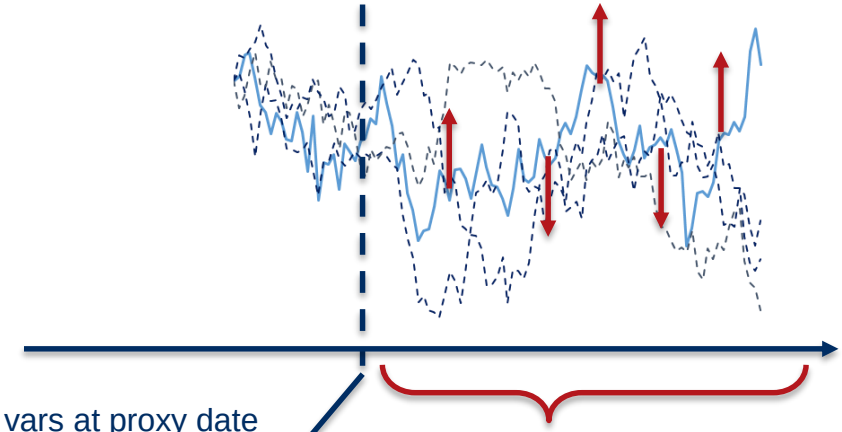
$\vec{X}$: regression variables, $X_j = f_j\left(\vec{S}_{T^{ex}}\right)$ or path dep. $f_j\left(\vec{S}_t, t \le T^{ex}\right)$

crucially: X is derived from (the path of) state variables ➔ PV available within the path

LSM refresher: pre-simulations

- LSM: use regression proxies where path-wise future PV are needed
  - Regression: $\tilde{V}_T \approx \sum \beta_j X_j$, Regression variables: $X_j = f_j\left(\vec{S}_T\right)$ or $f_j\left(\vec{S}_t, t \le T\right)$
    - Can be SVs, pair-wise products (quadratic regression), any polynomial or basis functions
    - And/or some analytical PV (like a swap in an IR model) or path-dependency (average so far)
  - State variables simulated ➔ proxy to future PV available within simulation
  - Resolve problem of future path-wise PVs for modelling exercise, and also exposure, xVA, …
- We first need to produce the regression coefficients $\vec{\beta}$
- We do so by *regression* over *pre-simulations*
  - Simulate a number of scenarios before starting the actual (main) valuation simulations
  - Use these pre-simulations to perform a regression of the future PV over regression variables
  - Which produces $\vec{\beta}$
  - We call this part LSM step – its only goal is to produce $\vec{\beta}$
  - Essentially, LSM transforms the problem into that of a path-dependent
  - Which, then, can be valued with standard Monte-Carlo methods in main simulations

LSM refresher: regression



collect all reg vars at proxy date
for all paths
in a matrix X

for all paths
sum up all cash-flows after proxy date
(discounted to proxy date)
in a vector Y

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

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:

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

LSM refresher: recursion

- With multiple independent proxy dates
  - For example xVA: multiple exposure dates, no dependence between proxies
  - We just repeat the regression step for each proxy date

- But with interdependent proxy dates…
  - Multi-callable: the path-wise PV (Y) depends on future exercises, future proxies and future betas
  - Exposure on callable trade: exposure proxy depends on future exercise proxies

- … We must regress recursively
  - Perform the regression step for the last proxy date first – **assuming no prior exercise**
  - Then for each proxy date, last to first – still assuming no prior exercise:
    - Produce Y path-wise, taking into account future exercises using future proxies
    - Perform the regression for that proxy date and move on to the previous one

LSM refresher: recursive mathematics

- Notations
  - Simulated State Variables $\left(\vec{S}_t\right)$ ➔ Regression Variables $\vec{X}_l^{\,m} = \vec{f}_l\left(\vec{S}_t^{\,m}, t \le T_l\right)$, Cash-Flows $CF_k^{\,m} = c_k\left(\vec{S}_t^{\,m}, t \le T_k\right)$
  - m: path index in pre-simulations, l: proxy date index, k: cash-flow date index

- Computations in the case of a multi-callable
  - Path-wise PVs (ignoring discounting) $Y_l^m = \underbrace{\sum_{T_l < T_k \le T_{l+1}} CF_k^{\,m}}_{\text{cf to next exercise}} + \overset{\text{sum over future exercises}}{\sum_{l \le li}} \left[\underbrace{\left(\prod_{l < lj \le li} 1_{\left\{\tilde{V}_{lj}^{\,m} > 0\right\}} \right)}_{\substack{\text{do not count cf} \\ \text{after exercise}}} \underbrace{\sum_{T_{li} < T_k \le T_{li+1}} CF_k^{\,m}}_{\text{cf between ex dates}} \right]$
  - Matrices $X_l = \begin{bmatrix} 1 & \vec{X}_l^{\,1\,T} \\ 1 & \vec{X}_l^{\,2\,T} \\ \dots & \dots \\ 1 & \vec{X}_l^{\,m\,T} \end{bmatrix}, Y = \begin{bmatrix} Y_l^1 \\ Y_l^2 \\ \dots \\ Y_l^m \end{bmatrix}$ ➔ Regression Coefficients $\beta_l = \left(X^{\,T}{}_l X_l\right)^{-1}\left(X_l{}^T Y_l\right)$
  - Proxies $\tilde{V}_l^{\,m} = \vec{\beta}_l \cdot \vec{X}_l^{\,m}$
  - These equations are clearly recursive, to be computed from last proxy date (l=L) to first (l=1)

- Also applies to exposure (xVA) on callables

SVD regression

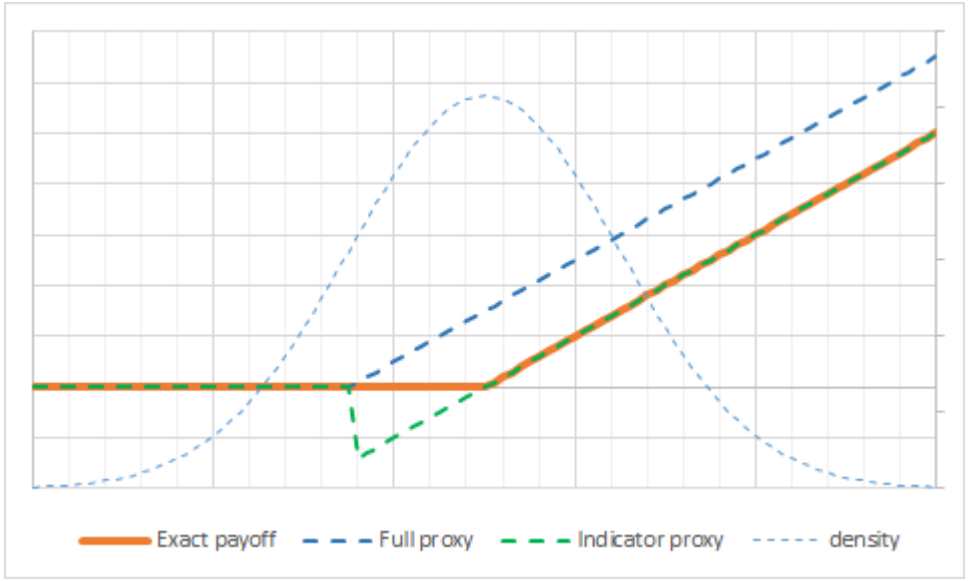
- The standard regression formula $\beta = \left(X^T X\right)^{-1}\left(X^T Y\right)$
  - Relies on $\left(X^T X\right)$ being invertible
  - When the columns of X are collinear (or close to, as is often the case) this is not satisfied
  - Which generates instabilities and inaccuracies
  - That tend to disappear with an increased number of pre-simulations M (rows of X)
  - At the expense of performance
- Instead we always use SVD regression $\beta = UD\Sigma V^T Y$
  - Where $X = VDU^T$ is the SVD decomposition of X excluding singular values close to 0
  - Sigma is the diagonal matrix $\Sigma_{jj} = \frac{1}{D_{jj}^2 + \lambda^2}$ and λ is Tikhonov's small norm preference
  - Resilient to collinearity, even when the number of rows of X is small
- Which improves not only stability, but also performance
  - We can reduce the number of paths without worrying about collinearity
- See Numerical Recipes, chp. 2.6 and 15.4 and our appendix

POI: Proxies Only in Indicators

- We systematically restrict the usage of proxies to indicators
  - To produce accurate proxies is extremely expensive
    - Large number of pre-simulations
    - Large number of regression variables / basis functions $\vec{X}_l^{\,m} = \vec{f}_l\left(\vec{S}_t^{\,m}, t \le T_l\right)$
  - We want to aggressively optimize the production of proxies and regression coefficients
  - Even at the expense of the accuracy of proxies
  - ➢ We want to minimize the reliance of the final result to the accuracy of proxies
- This is achieved by using proxies only in indicators
- Which has considerable additional benefits
  - Reformulate xVA problem in a particularly effective way for computation
    - Even with collateral
  - Make the differentiation of LSM essentially free in most cases
    - And very effective otherwise

POI: illustration

- Example
  - Call on V, strike K
  - Our proxy is biased: $\tilde{V} = V + \mu$
- Strategy 1: use proxy in payoff
  - Wrong strike: $\left(\tilde{V} - K\right)^+ = \left[V - \left(K - \mu\right)\right]^+$
  - Error magnitude: $\Delta\mu$, order μ
- Strategy 2: use proxy in indicator
  - How? See next: $1_{\{\tilde{V} > K\}}\left(V - K\right)$
  - Error magnitude: $dens\left(V == K\right) \cdot \mu^2$, order μ^2



One time callable exotics

- Using proxy directly: $V_0 = E\left[\left(\tilde{V}_{T^{ex}}\right)^+\right]$
  - Obviously, result heavily dependent on proxy
- Using proxy in indicator: $V_0 = E\left[1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right], V_{T^{ex}} = E_{T^{ex}}\left[\sum_{T_k>T^{ex}} CF_k\right] \rightarrow V_0 = E\left[1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} \sum_{T_k>T^{ex}} CF_k\right]$
- Limits dependence on proxy
  - We denote $\varepsilon \equiv \tilde{V}_{T^{ex}} - V_{T^{ex}}$ the proxy inaccuracy
  - Then the error due to the proxy inaccuracy is $E\left[\underbrace{\left(1_{\left\{V_{T^{ex}}+\varepsilon>0\right\}} - 1_{\left\{V_{T^{ex}}>0\right\}}\right)}_{\equiv e(\varepsilon)} V_{T^{ex}}\right]$
  - And we note that (obviously) $E\left[e(0)\right] = 0$
  - But also $E\left[e'(0)\right] = 0$
  - ➢ (to the 1st order) the proxy inaccuracy does not matter to the result
- In addition note that: $V_0 = E\left[\sum_{T_k>T^{ex}} 1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} CF_k\right]$
  - To value the callable transactions, we value its cash-flows, discounted for exercise
  - That seems trivial, but we beautifully generalize it next, including for xVA

Multi-callable exotics

- From there on, we assume that proxies are only used in indicators
- Most of what follows does not hold otherwise!

- For multi-callables trades, that means that: $V_0 \approx E\left[\sum_k \left(\underbrace{\prod_{T^{ex}{}_l < T_k} 1_{\left\{\tilde{V}_{T^{ex}{}_l} > 0\right\}}}_{\eta_{T_k}}\right) CF_k\right]$
  - To value the callable trade, we value its cash-flows…
  - …Discounted to account for exercise by a process (η_t) …
  - …That starts at 1 and jumps to 0 on the first exercise…
  - …That is the 1st time where the proxy is negative on a call date
- We illustrate that
  - LSM effectively turned the (multi-) callable into a path-dependent
  - This path-dependence turns out to be a stochastic discounting process for cash-flows
  - Amazingly, we can say the exact same thing for xVA

xVA

- With POI we rewrite CVA (or any xVA) following Andreasen, 2014
  - By definition: $CVA = E\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} \max\left(0, V_{T_p}\right)\right] \approx \left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} 1_{\{\tilde{V}_{T_p}>0\}} V_{T_p}\right]$
  - And $V_{T_p} = E_{T_p}\left(\sum_{T_k>T_i} CF_k\right)$
  - After some reversal and re-arranging we find that: $CVA = E\left[\sum_k \eta_{T_k} CF_k\right], \eta_{T_k} = \sum_{T_p<T_k} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} 1_{\{\tilde{V}_{T_p}>0\}}$
- To value a CVA
  - We value the cash-flows in its netting set…
  - …Discounted by the (stochastic, path-dependent) process $\eta_{T_k} = \sum_{T_p<T_k} \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} 1_{\{\tilde{V}_{T_p}>0\}}$
  - …(Which is the only place that uses proxies), and that obeys the simple update rule

$$\eta_0 = 1, \eta_{T_{p+1}} = \eta_{T_p} + \left(1-\mathrm{R}_{T_p}\right) 1_{\{T_{p-1}\le\tau<T_p\}} 1_{\{\tilde{V}_{T_p}>0\}}$$

- Which sheds a new light on xVA and produces a valuation algorithm that is:
  - **Accurate:** proxies are only used in indicators in the discounting process
  - **Efficient:** value xVA for the cost of cash-flow valuation + simulation of discounting process
    path-wise simulation lends itself to parallel processing
  - **Practical:** CF valuation scripts simply need to be *decorated* so they are discounted by η

Collateralized xVA

- For instance fully collateralized CVA with MPR θ:

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

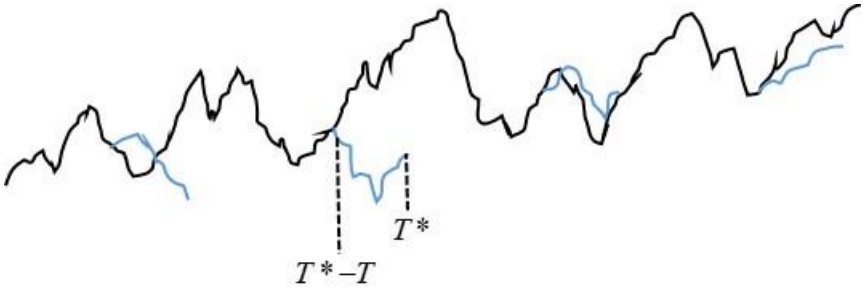
- That **cannot** be directly written as $CVA = E\left[\sum_k \eta_{T_k} CF_k\right]$
  - Consider one exposure: $e(T^*) = E\left[\max\left(0, V_{T^*} - V_{T^*-\vartheta}\right)\right] = E\left[1_{\{V_{T^*}>V_{T^*-\vartheta}\}} V_{T^*}\right] - E\left[1_{\{V_{T^*}>V_{T^*-\vartheta}\}} V_{T^*-\vartheta}\right]$
  - The LHS can be rewritten as previously $LHS \approx E\left[\sum_{T_k>T^*} 1_{\{\tilde{V}_{T^*}>\tilde{V}_{T^*-\vartheta}\}} CF_k\right]$
  - But the not the RHS because $1_{\{V_{T^*}>V_{T^*-\theta}\}}$ is **not** T*- θ –measurable so

$$RHS \approx E\left\{E_{T^*-\vartheta}\left[1_{\{\tilde{V}_{T^*}>\tilde{V}_{T^*-\vartheta}\}}\right] E_{T^*-\vartheta}\left(\sum_{T_k>T^*} CF_k\right)\right\}$$

Branching to the rescue

- Branching (McKean 1975, Henry-Labordere 2012)
  - For each path, also generate one Branch that “sticks out” at margin date…
  - And starts a parallel path that obeys the same SDE but independently from the main path

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.



- Branching does **not** involve nested simulations
  - For every path, we only generate **one** Branch (per margin date) as part of that simulation
  - Averaging is correctly performed by the outer expectation operator

Branching Proxies

- Consider a proxy read on a secondary branch $^{B}\tilde{V}_T = \beta \cdot f\left(^{B}S_T\right)$
  - Constructed with the same regression coefficients, but regression variables simulated on the branch
  - Conditionally to the filtration at the margin date, $T^{mrg} = T - \vartheta$ the main and secondary proxies are:
    - Independent
    - Identically distributed
- We can now compute the RHS from the collateralized exposure equation

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

Branching solution

- We finally get the complete formula for the collateralized exposure

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

- And the collateralized CVA

$$CVA = E\left[\sum_k \eta_{T_k} CF_k\right] \qquad \eta_{T_p} = \sum_{T_q < T_p} (1 - \mathrm{R}_{T_q})\, 1_{\{T_{q-1} \le \tau < T_q\}} \left(1_{\{\tilde{V}_{T_q} > \tilde{V}_{T_q}^{mrg}\}} - 1_{\{{}^{B_q}\tilde{V}_{T_q} > \tilde{V}_{T_q}^{mrg}\}}\right)$$

- All the comments from the uncollateralized case apply
  - Essentially the same algorithm: accurate, efficient, practical
  - Except the discounting process is simulated with Branches ➔ Up to 2x slower
  - In addition, it is recommended to smooth indicators to reduce variance (more on that later)

Effective LSM implementation: conclusion

- We implement LSM with:
  - A prior LSM step that
    - Is itself divided into pre-simulation + regression
    - Produces regression coefficients
    - Turns the problem of callables and xVA into standard path dependence
    - Uses SVD for stability and performance
  - Followed by a main (standard) Monte-Carlo step that
    - Computes the value of that path dependent
    - Uses POI to decrease reliance on proxy accuracy
- And we found that:
  - In all cases of interest: callable exotics, xVA
  - We always end up valuing the underlying cash-flows
  - And discounting them with a simple path-dependent process that uses the proxies
  - The resulting algorithm is accurate, efficient and practical

Effective LSM differentiation

Cash-flow differentials

- Suppose for the purpose of computing risk sensitivities:
  - We ignore the LSM step
  - And use (now standard) path-wise AAD over the main simulations
- We know that:
  - AAD does not differentiate control flow ➔ Differential of indicators is 0
  - And since we only use proxies in indicators
  - What we obtain are path-wise differentials **with fixed proxies**
- Also called “cash-flow differentials”
  - Because we differentiate cash-flows but fix the regression coefficients, proxies and indicators
  - Comes essentially for free if we already have path-wise AAD in place
  - But are they correct?
    Are cash-flow differentials same as complete differentials or do they miss something
  - The answer is “that depends…”

Cash flow differentials: one time callable exotics

- We remind that: $V_0 = E\left[1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} V_{T^{ex}}\right], V_{T^{ex}} = E_{T^{ex}}\left[\sum_{T_k > T^{ex}} CF_k\right] \rightarrow V_0 = E\left[1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} \sum_{T_k > T^{ex}} CF_k\right]$

- Taking the differential to some parameter a: $\frac{\partial V_0}{\partial a} = E\left[1_{\left\{\tilde{V}_{T^{ex}}>0\right\}} \sum_{T_k > T^{ex}} \frac{\partial CF_k}{\partial a}\right] + E\left[\delta\left(\tilde{V}_{T^{ex}}\right) V_{T^{ex}}\right]$
  - The LHS is the cash-flow differential
  - The RHS is equal to: $E\left[\delta\left(\tilde{V}_{T^{ex}}\right) V_{T^{ex}}\right] = dens\left(\tilde{V}_{T^{ex}} == 0\right) E\left[V_{T^{ex}} \middle| \tilde{V}_{T^{ex}} = 0\right]$
  - When the proxy is accurate in the sense the $E\left[V_{T^{ex}} \middle| \tilde{V}_{T^{ex}} = 0\right] = 0$, RHS = 0
  - That means that the complete risk sensitivity and the cash-flow differential are one and the same
- What if the proxy is inaccurate?
  - Then the risk sensitivity is different from the cash-flow differential
  - Our argument: the difference amounts to unwanted numerical noise
  - Counter-argument: accurate PnL explain requires complete sensitivity, incl. numerical artefact
  - We “agree with ourselves” while acknowledging some validity in the counter argument
  - Where a complete differential is desirable, we provide effective means of conducting it next

Cash flow differentials: other cases

- Along the same lines we show that CF differentials are also sufficient for:
  - Multi-callable exotics
  - Exposures
  - xVA, with or without collateral **but without callable transactions in the netting set**
  - And for anything where
    - The trigger in the indicator: what we exercise on
    - Is a proxy to the payoff: what we get on exercise

$$\frac{\partial}{\partial a}E\left[DX1_{\{\tilde{X}>0\}}\right]=E\left[\frac{\partial(DX)}{\partial a}1_{\{\tilde{X}>0\}}\right] \text{ but } \frac{\partial}{\partial a}E\left[DY1_{\{\tilde{X}>0\}}\right]\neq E\left[\frac{\partial(DY)}{\partial a}1_{\{\tilde{X}>0\}}\right]$$

- But it misses a significant part of the sensitivity
  - When the trigger is different from the payoff, like an option exercised on S&P that delivers the Nikkei
  - **That includes xVA on callable exotics – as the only case of interest here**
  - And then, we **must** differentiate the whole process, including LSM pre-simulations and regressions

Illustration: exposure on callable exotics

- We illustrate the problem with a T_2 exposure on an one time T_1 callable exotic:
  - Rather trivially: $e = E\left[1_{\{V_{T_1}>0\}} V_{T_2}^{+}\right] \approx E\left[1_{\{\tilde{V}_{T_1}>0\}} 1_{\{\tilde{V}_{T_2}>0\}} V_{T_2}\right], V_{T_2} = E_{T_2}\left[\sum_{T_k>T_2} CF_k\right]$
  - We have 2 proxies involved, in 2 indicators
    - The exercise proxy $\tilde{V}_{T_1}$
    - The exposure proxy $\tilde{V}_{T_2}$
  - And the payoff is V_{T_2} , to which the exposure proxy is a proxy but the exercise proxy is not
- The trigger on the exercise is different for the payoff
- Therefore cash-flow differentials are **not** sufficient
- In fact, we show that the error
  – difference between CF differential and complete sensitivity - is

$$-dens\left(\tilde{V}_{T_1}==0\right)\Pr\left(\tilde{V}_{T_2}>0\middle|V_{T_1}=0\right)E\left[\frac{\partial \tilde{V}_{T_1}}{\partial a}\underbrace{\left(V_{T_1}-V_{T_2}\right)}_{=\sum_{T_1<T_k\le T_2} CF_k}\middle|\tilde{V}_{T_1}=0,\tilde{V}_{T_2}>0\right]$$

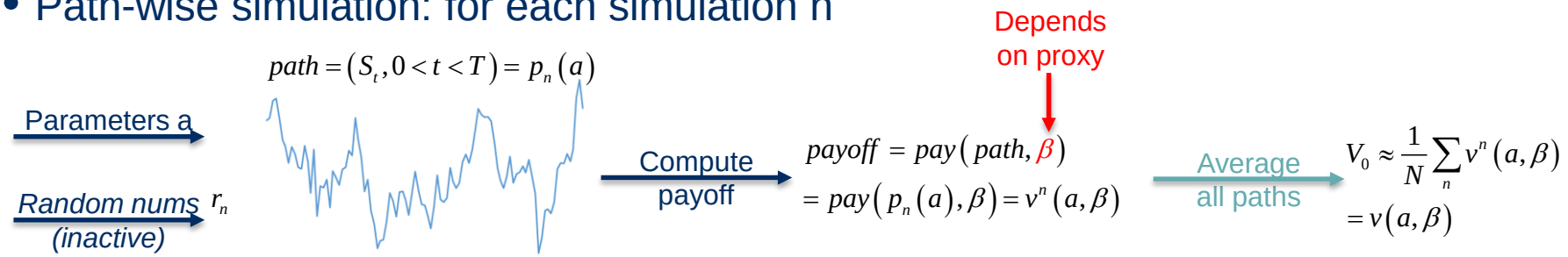
It is directly related to the cash-flows that are part of V_{T_1} (the exercise trigger) but not of V_{T_2} (the payoff)

Check-Pointed Differentials: one proxy

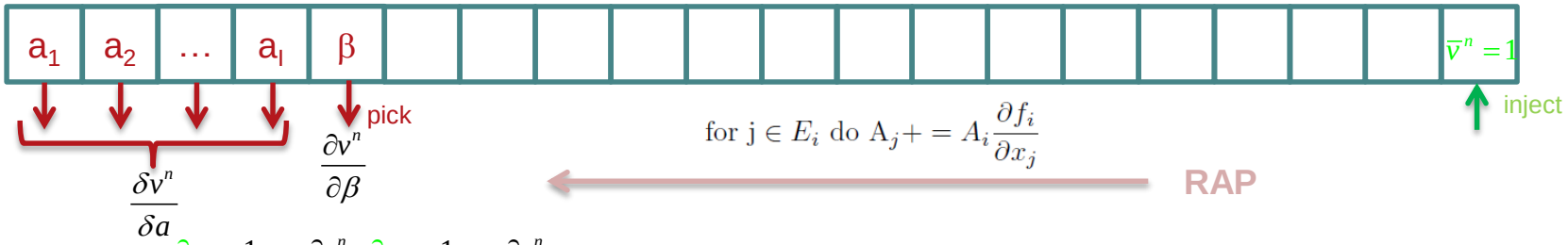
- Reminder: valuation takes 3 steps:
  1. **Pre-simulations**: for all pre-sim paths m, produce regression variables X and path-wise PVs Y as functions of parameters a and (irrelevant, inactive) random numbers
  $$\left[X^m, Y^m\right] = g_m(a)$$
  2. **Regression**: standard or SVD, produce regression coefficients out of all the Xs and Ys
  $$\beta = h(X, Y) \text{ where } X = \left(X^m, 1 \le m \le M\right), Y = \left(Y^m, 1 \le m \le M\right)$$
  3. **Main simulations**: produce and average path-wise payoffs as functions of a and the β from 1, 2
  $$V = v(a, \beta) = \frac{1}{N}\sum_n v^n(a, \beta)$$
- Check-pointed differential in reverse order: $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta}\sum_m \frac{\partial h}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$
  1. (3) Compute $\frac{\partial v}{\partial a}$ and $\frac{\partial v}{\partial \beta}$ with standard AAD **path-wise: low mem consumption, parallel**
  2. (2) Compute $\frac{\partial V}{\partial\{X, Y\}} = \frac{\partial v}{\partial \beta}\frac{\partial h}{\partial\{X, Y\}}$ ➔ Using efficient **check-pointing**, not costly Jakobians
  3. (1) Perform **check-pointed path-wise** AAD differentials of g_m to find $\frac{\partial V}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

Main simulation differentials: AAD path-wise

- Path-wise simulation: for each simulation n



- AAD differential (see e.g. Savine, 2014):
  - Compute $v^n(a,\beta)$ with AAD instrumentation, build tape
  - Seed result with $\bar{v}^n = 1$, apply chain rule (reverse adjoint propagation), pick $\bar{a} = \frac{\partial v^n}{\partial a}, \bar{\beta} = \frac{\partial v^n}{\partial \beta}$



- Average $\frac{\partial v}{\partial a} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial a}, \frac{\partial v}{\partial \beta} = \frac{1}{N}\sum_n \frac{\partial v^n}{\partial \beta}$

Main simulation differentials: efficiency

- AAD: constant (x4-8 valuation) in sensitivities (number of a's)

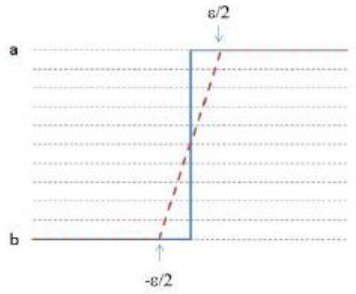
- Path-wise computation
  - ✓ Subdued memory consumption: one path at a time, worst case 0.01s ➔ 50MB
  - ✓ Cache efficient
  - ✓ Embarrassingly parallel problem ➔ screams for parallelisation over paths
    - ✓ Reminder: AAD is friendly to parallel processing, see e.g. Savine, 2014

Main simulation differentials: smooth indicators

- There is one problem
  - AAD does not differentiate indicators
  - Proxies are only used in indicators
  - Regression coefficients are only used to compute proxies
  - Hence we always find that $\frac{\partial v}{\partial \beta} = 0$

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

- Solution: always use smooth indicators
  - Replace “digitals” by “call-spreads”
  - Removes discontinuities
  - Corrects and stabilizes sensitivities
  - See Savine, 2016 for the full story
  - Produces the correct $\frac{\partial v}{\partial \beta}$
  - And $\frac{\partial v}{\partial a}$ are now path-wise derivatives with **fixed coefficients** – not fixed proxies



LSM differentials

- Valuation
  1. **Pre-simulations**: $\left[X^m, Y^m\right] = g_m(a)$
  2. **Regression**: $\beta = h(X, Y)$ where $X = \left(X^m, 1 \le m \le M\right), Y = \left(Y^m, 1 \le m \le M\right)$
  3. **Main simulations**: $V = v(a, \beta) = \frac{1}{N}\sum_n v^n(a, \beta)$

LSM

- Differential: $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta}\sum_m \frac{\partial h}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

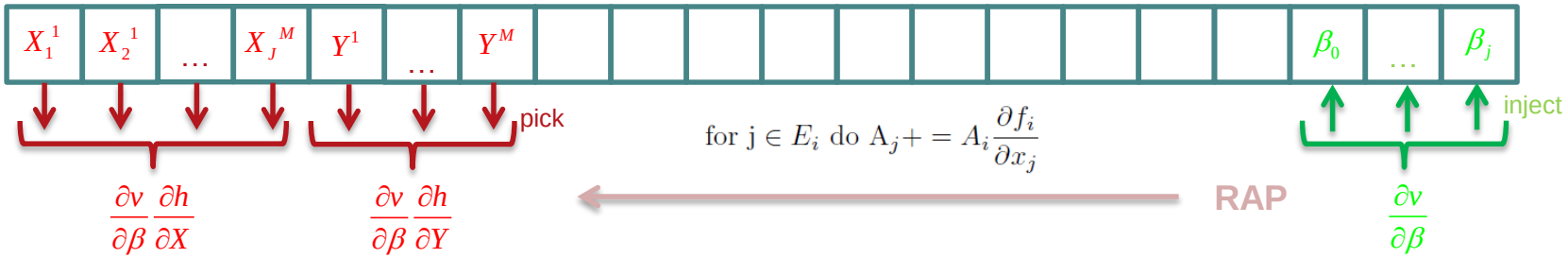
✓ (3) Compute $\frac{\partial v}{\partial a}$ and $\frac{\partial v}{\partial \beta}$

2. (2) Compute $\frac{\partial V}{\partial\{X, Y\}} = \frac{\partial v}{\partial \beta}\frac{\partial h}{\partial\{X, Y\}}$
3. (1) Compute $\frac{\partial V}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

LSM

Regression differentials: standard regression

- Standard regression: $\beta = h(X,Y) = (X^T X)^{-1}(X^T Y)$
- We compute $\frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$
  - Natural solution (J: number of coefficients / variables, M: number of pre-simulations)
    - Compute the (J+1) x M x (J+1) Jakobian $\frac{\partial h}{\partial \{X,Y\}}$ → cost ~ (J+1) with AAD, M x (J+1) otherwise
    - Multiply by the (J+1) vector $\frac{\partial v}{\partial \beta}$ → cost ~ (J+1)^2 x M
    - Expensive and inefficient
  - Check-pointing: compute $\frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$ knowing $\frac{\partial v}{\partial \beta}$ **in constant time**
    - Compute $h(X,Y)$ with AAD instrumentation, build AAD tape
    - Seed result with $\frac{\partial v}{\partial \beta}$, apply chain rule (reverse adjoint propagation), pick $\frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$



Standard regression differentials: comments

- For reference only: use SVD in practice
- Adjoint calculations over $\beta = h(X,Y) = \left(X^T X\right)^{-1}\left(X^T Y\right)$ can also be performed analytically
  - We call that “analytical AD”
  - See Huge, AD of matrix calculations, 2016
  - Analytical AD is faster, but AAD as presented is maintenance free
- The illustrated check-pointing technique is extremely useful
  - Efficient: avoids expensive computation and application of Jakobians
  - Computes $\frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$ given $\frac{\partial v}{\partial \beta}$ without computing $\frac{\partial h}{\partial \{X,Y\}}$ or multiplying matrices
  - We use that same technique next for the differential of the pre-simulation step

Regression differentials: SVD

- For SVD regression we will compute $\frac{\partial v}{\partial \beta}$ $\xrightarrow{\text{Analytical AD}}$ $\frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$
- If SVD is used we have 2 steps
  - Calculate the SVD of $X = VDU^T$
  - Calculate the solution $\beta = UD\Sigma V^T Y$
- This looks easy enough using check-pointing
  - AAD the matrix operations $\beta = UD\Sigma V^T Y$
  - Check-point into SVD
- Unfortunately SVD is not AAD'able
  - For common singular values SVD is not unique
  - The orthogonal basis can be rotated
- SVD is `Analytical AD`able for **distinct** singular values

Regression differentials: SVD

- For distinct singular values we can show

$$W = U^T \frac{\partial v}{\partial \beta} Y^T V$$

$$\Gamma = \Sigma\left(\lambda^2 W^T - DWD\right)\Sigma$$

$$\frac{\partial v}{\partial X} = V\Gamma U^T + \left(I - VV^T\right) Y \frac{\partial v}{\partial \beta}^T U\Sigma U^T + V\Sigma V^T Y \frac{\partial v}{\partial \beta}^T \left(I - UU^T\right)$$

$$\frac{\partial v}{\partial Y} = VD\Sigma U^T \frac{\partial v}{\partial \beta}$$

$$\left(\frac{\partial v}{\partial \lambda} = -2\lambda\beta^T U\Sigma U^T \frac{\partial v}{\partial \beta}\right)$$

- Also valid for common singular values hence by continuity this is also general
- We have $\frac{\partial v}{\partial \beta}$ $\xrightarrow{\text{Analytical AD}}$ $\frac{\partial v}{\partial \{X,Y\}} = \frac{\partial v}{\partial \beta}\frac{\partial h}{\partial \{X,Y\}}$

Pre-simulation differentials

- Valuation
  1. **Pre-simulations**: $\left[X^m, Y^m\right] = g_m(a)$
  2. **Regression**: $\beta = h(X,Y)$ where $X = \left(X^m, 1 \le m \le M\right), Y = \left(Y^m, 1 \le m \le M\right)$
  3. **Main simulations**: $V = v(a,\beta) = \frac{1}{N}\sum_n v^n(a,\beta)$

} LSM

- Differential: $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta}\sum_m \frac{\partial h}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

✓ (3) Compute $\frac{\partial v}{\partial a}$ and $\frac{\partial v}{\partial \beta}$

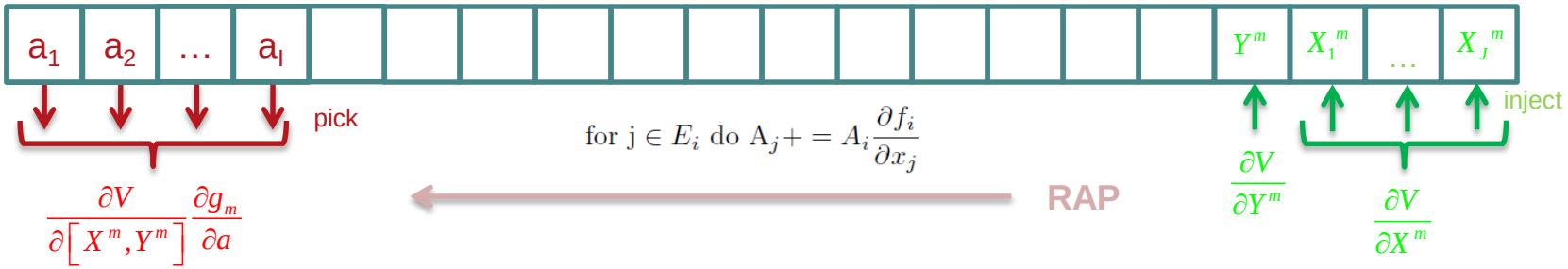
✓ (2) Compute $\frac{\partial V}{\partial\{X,Y\}} = \frac{\partial v}{\partial \beta}\frac{\partial h}{\partial\{X,Y\}}$

2. (1) Compute $\frac{\partial V}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

} LSM

Pre-simulation differentials

- We compute **path-wise**, that is for each pre-sim path m: $\frac{\partial V}{\partial \left[X^m, Y^m \right]} \frac{\partial g_m}{\partial a}$
- Again, we avoid expensive Jakobian computation and matrix product
- Instead, we check-point to produce the sensitivity in **constant time**
- This step (which is the most expensive) can be easily conducted in parallel
  - Compute $\left[X^m, Y^m \right] = g_m(a)$ with AAD instrumentation, build adjoint tape
  - Seed result with $\frac{\partial V}{\partial \left[X^m, Y^m \right]}$, apply chain rule (reverse adjoint propagation), pick $\frac{\partial V}{\partial \left[X^m, Y^m \right]} \frac{\partial g_m}{\partial a}$



Complete differential

- Valuation
  1. **Pre-simulations**: $\left[X^m, Y^m\right] = g_m(a)$
  2. **Regression**: $\beta = h(X, Y)$ where $X = \left(X^m, 1 \le m \le M\right), Y = \left(Y^m, 1 \le m \le M\right)$
  3. **Main simulations**: $V = v(a, \beta) = \frac{1}{N}\sum_n v^n(a, \beta)$

LSM

- Differential: $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta}\sum_m \frac{\partial h}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

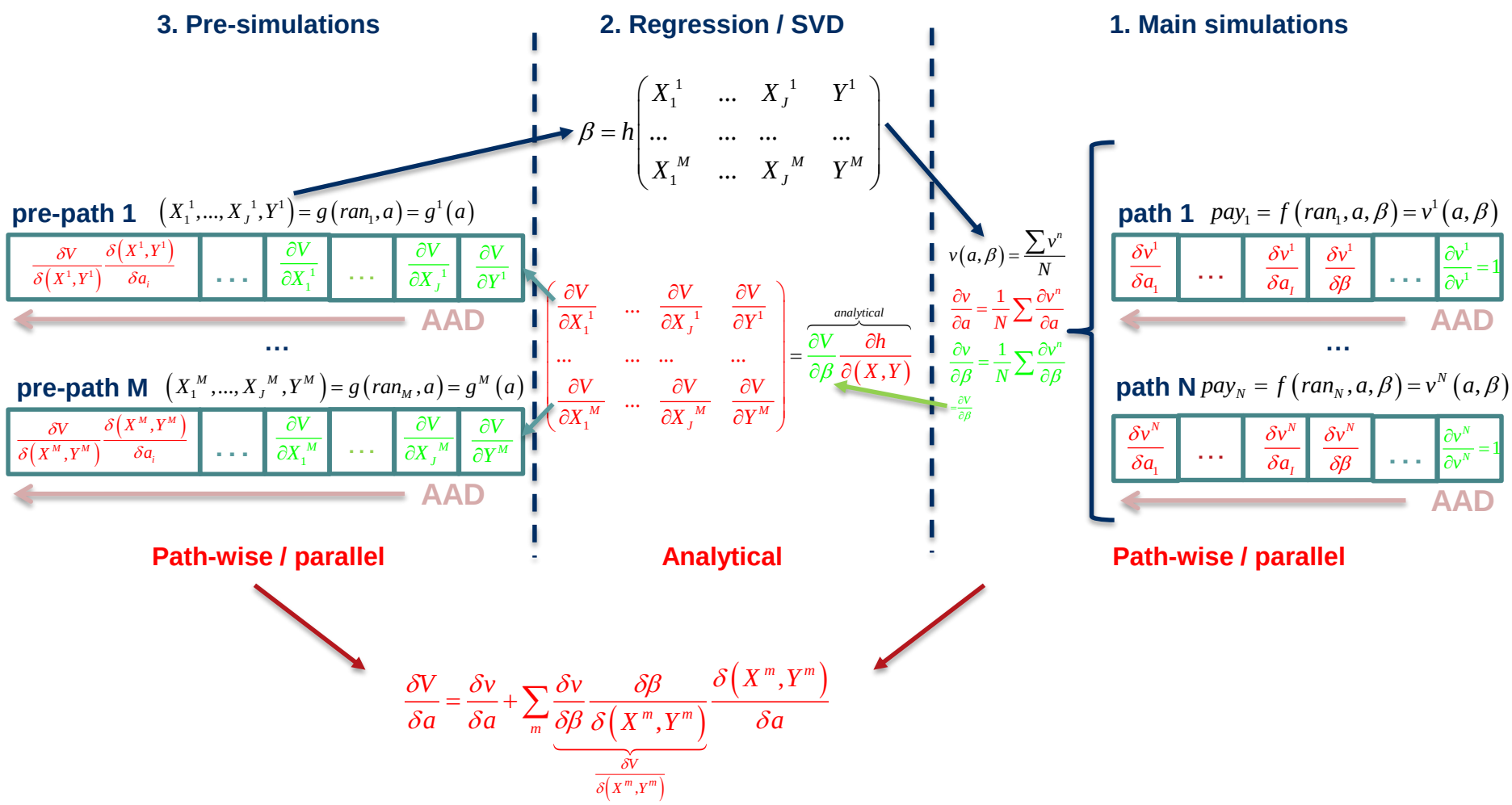
  ✓ (3) Compute $\frac{\partial v}{\partial a}$ and $\frac{\partial v}{\partial \beta}$

  ✓ (2) Compute $\frac{\partial V}{\partial\{X, Y\}} = \frac{\partial v}{\partial \beta}\frac{\partial h}{\partial\{X, Y\}}$

  ✓ (1) Compute $\frac{\partial V}{\partial\left[X^m, Y^m\right]}\frac{\partial g_m}{\partial a}$

LSM

Single proxy case: check-pointing summary



Check-Pointed Differentials: multiple proxies

- Easy case: unrelated proxies
  - Example: xVA
  - Multiple proxies, one per exposure date
  - No interdependence: proxies are produced and used independently on one another
  - In this case, we just repeat the steps for each proxy, adding up their contributions

$$\text{one proxy: } \frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta} \sum_m \frac{\partial h}{\partial \left[X^m, Y^m \right]} \frac{\partial g_m}{\partial a}$$

$$\text{multiple proxies: } \frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \sum_l \frac{\partial v}{\partial \beta_l} \sum_m \frac{\partial h_l}{\partial \left[X_l^m, Y_l^m \right]} \frac{\partial g_{l,m}}{\partial a}$$

- Harder case: interrelated proxies
  - Example: multi-callable exotics, exposure/xVA on callable exotics
  - Multiple proxies, which production depends on one another
  - Reminder: the formula to produce β_l depends on future proxies/betas $(\beta_k, k > l)$
  - Hence, we must introduce one further check-pointed step

2 interrelated proxies

- Twice callable trade at T_1 and $T_2 > T_1$
  - Or a T_1 exposure on a T_2 callable
  - In both cases, β_2 is produced independently of β_1 and $\frac{\partial v}{\partial \beta_2}\sum_m \frac{\partial h_2}{\partial\left[X_2^m, Y_2^m\right]}\frac{\partial g_{2,m}}{\partial a}$ is produced as previously
  - But β_1 depends on β_2 and is computed after β_2
  - Unsurprisingly, we need to differentiate in the *reverse* order, β_1 first and then β_2

- Reminder: with one proxy $\frac{\partial V}{\partial a} = \underbrace{\frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta}}_{\text{main sim}}\frac{\partial \beta}{\partial a}$ LSM step

- With 2 proxies: $\frac{\partial V}{\partial a} = \underbrace{\frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta_1}}_{\text{main sim}}\frac{\partial \beta_1}{\partial a}$ T_1 LSM step $+ \left(\frac{\partial v}{\partial \beta_2} + \underbrace{\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2}}_{\text{main sim } T_1 \text{ LSM step}} \right) \frac{\partial \beta_2}{\partial a}$ T_2 LSM step

extra sensitivity to β_2 from contribution to β_1

Algorithm: 2 interrelated proxies

1. Differentiate main simulations – reminder: use smooth indicators
   - We get $\frac{\partial v}{\partial a}$ $\frac{\partial v}{\partial \beta_1}$ $\frac{\partial v}{\partial \beta_2}$

2. Check-point $\frac{\partial v}{\partial \beta_1}$ into $\beta_1 = lsm(a, \beta_2)$ → see LSM differentials: regression + pre-sim
   - The difference is β_1 is now a function of not only a, but also β_2
   - We get $\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial a}$ $\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2}$
   - We *update* the sensitivity to β_2 to incorporate that part $\frac{\partial V}{\partial \beta_2} = \underbrace{\frac{\partial v}{\partial \beta_2}}_{\text{main sim}} + \underbrace{\frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2}}_{\substack{\text{main sim } T_1 \text{ LSM step} \\ \text{extra sensitivity to } \beta_2 \\ \text{from contribution to } \beta_1}}$

3. Check-point $\frac{\partial V}{\partial \beta_2}$ into $\beta_2 = lsm(a)$
   - We get $\frac{\partial V}{\partial \beta_2}\frac{\partial \beta_2}{\partial a} = \left(\frac{\partial v}{\partial \beta_2} + \frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2}\right)\frac{\partial \beta_2}{\partial a}$

1. Add up contributions from 1, 2, 3 $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a} + \frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial a} + \left(\frac{\partial v}{\partial \beta_2} + \frac{\partial v}{\partial \beta_1}\frac{\partial \beta_1}{\partial \beta_2}\right)\frac{\partial \beta_2}{\partial a}$

Generalized algorithm: L interrelated proxies

1. Differentiate main simulations – reminder: use smooth indicators
   - We get $\frac{\partial v}{\partial a}$ $\frac{\partial v}{\partial \beta_1}$ $\frac{\partial v}{\partial \beta_2}$ … $\frac{\partial v}{\partial \beta_L}$
   - Start with $\frac{\partial V}{\partial a} = \frac{\partial v}{\partial a}$ $\frac{\partial V}{\partial \beta_1} = \frac{\partial v}{\partial \beta_1}$ … $\frac{\partial V}{\partial \beta_L} = \frac{\partial v}{\partial \beta_L}$

2. Recursively (l from 1 to L) check-point $\frac{\partial V}{\partial \beta_l}$ into $\beta_l = lsm(a, \beta_{l+1}, ..., \beta_L)$
   - We get $\frac{\partial V}{\partial \beta_l}\frac{\partial \beta_l}{\partial a}$ $\frac{\partial v}{\partial \beta_l}\frac{\partial \beta_l}{\partial \beta_{l+1}}$ … $\frac{\partial v}{\partial \beta_l}\frac{\partial \beta_l}{\partial \beta_L}$
   - So we update the sensitivity to a $\frac{\partial V}{\partial a} += \frac{\partial V}{\partial \beta_l}\frac{\partial \beta_l}{\partial a}$
   - And the sensitivities to $\beta_{l+1}, ..., \beta_L$ $\frac{\partial V}{\partial \beta_{l+1}} += \frac{\partial V}{\partial \beta_l}\frac{\partial \beta_l}{\partial \beta_{l+1}}; ...; \frac{\partial V}{\partial \beta_L} += \frac{\partial V}{\partial \beta_l}\frac{\partial \beta_l}{\partial \beta_L}$
   - Repeat

Effective LSM differentials: conclusion

- Differentiation often comes for free
- When this is not the case we compute LSM differentials efficiently
  - Avoid expensive production and application of Jakobians with systematic *check-pointing*
  - Conduct differentiation of main and pre simulations path-wise
    - Memory and cache efficient
    - Embarrassingly parallel
  - SVD differential is essentially analytical

Conclusion

- Effective valuation with LSM
  - Path-wise for parallel processing
  - Systematic application of POI ➔ reduces reliance on proxies
  - xVA as a stochastic discounting
  - Extension to collateral with branching

- Effective differentiation with LSM
  - Systematic check-pointing avoids expensive production and application of Jakobians
  - Also divides computation into steps manageable for memory and cache
  - Conduct differentials of pre simulations path-wise for memory, cache and parallel processing
  - SVD differential is essentially analytical

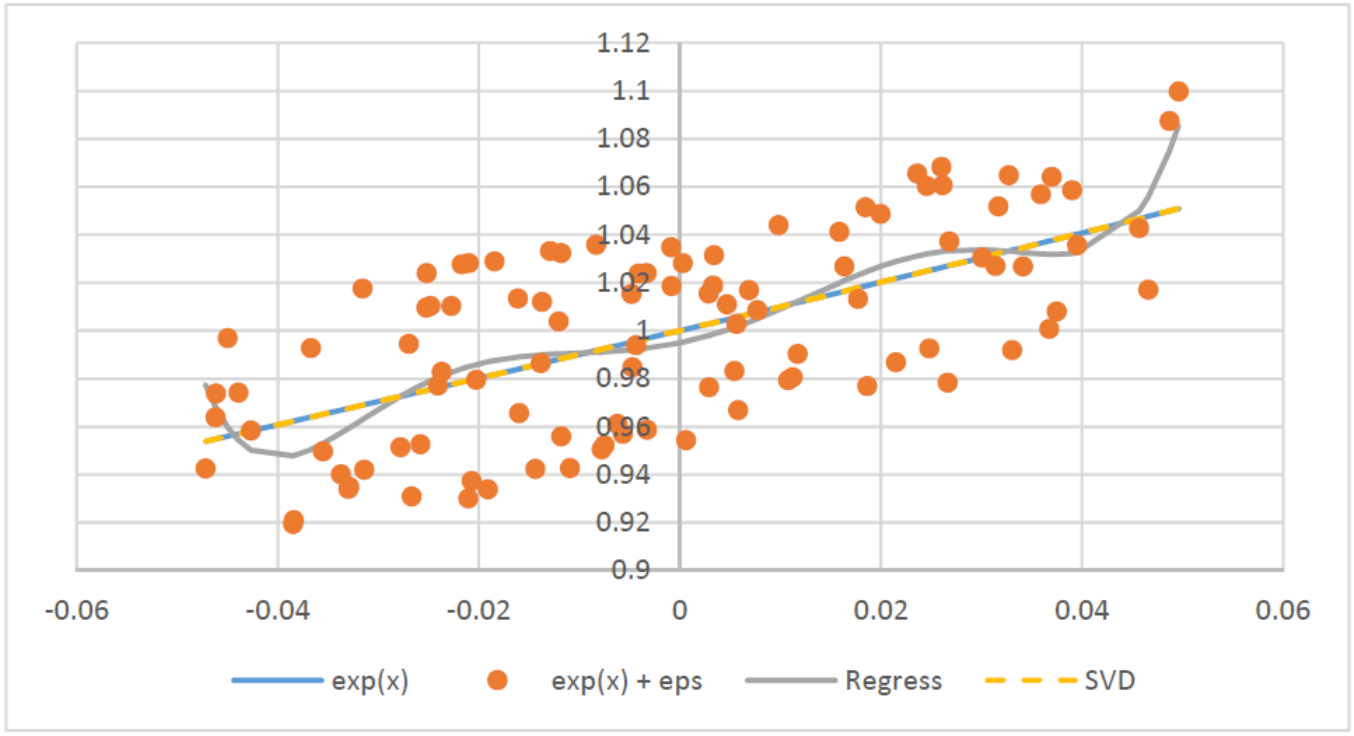
Appendix: SVD regression

- The least squares problem $\beta = \arg\min_{b} \left\| Y - Xb \right\|^2$

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

- Collinearity (linearity between columns in X)
  ➔ $\left(X^T X \right)$ singular and **not** invertible
- Occurs when simulations do not distinguish between 2 or more basis functions
  - Simulations fit equally well or equally bad
  - Not enough simulations to distinguish basis functions
- SVD to the rescue
  (Numerical Recipes chap 15.4 offers a nice discussion of this point)

SVD regression: example



Singular Value Decomposition: crash course

- Any matrix $A \in \mathbb{R}^{M \times N}$ with $M \geq N$ can be written as the product of
  - a column orthogonal matrix $V \in \mathbb{R}^{M \times N}$
  - a diagonal matrix $D \in \mathbb{R}^{N \times N}$ with positive or 0 elements (the singular values)
  - an orthogonal matrix $U \in \mathbb{R}^{N \times N}$

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} = \begin{bmatrix} \\ V \\ \\ \end{bmatrix} \begin{bmatrix} D_{11} & & 0 \\ & \ddots & \\ 0 & & D_{MM} \end{bmatrix} \begin{bmatrix} \\ U^T \\ \\ \end{bmatrix}$$

- An algorithm for finding V, D, U can be found in ”Numerical Recipes: The Art of Scientific Computing” (chap 2.6) alongside a nice intro for SVD

SVD: crash course (2)

- Define $X = \hat{V}\hat{D}\hat{U}^T, \hat{V} = \begin{bmatrix} V & V^\perp \end{bmatrix}, \hat{D} = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}, \hat{U} = \begin{bmatrix} U & U^\perp \end{bmatrix}, D_{ii} > \varepsilon > 0$
- Transform the problem with the orthogonal basis $b = \hat{U}a$
- The transformed least squares problem $\min_b \|Y - Xb\|^2 = \min_a \|Y - \hat{V}\hat{D}a\|^2$

  with solution $\alpha_1 = \left(DV^TVD\right)^{-1}\left(DV^TY\right) = D^{-1}V^TY \quad = \min_{a_1} \|Y - VDa_1\|^2$

- To get the solution for the original problem we transform back

$$\beta = \hat{U}\alpha = \hat{U}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = UD^{-1}V^TY + U^\perp\alpha_2 = UD^{-1}V^TY$$

- We choose $\alpha_2 = 0$, which exactly excludes the redundancies in X

SVD: stabilisation

- SVD cut parameter $\varepsilon > 0$ so all singular $\hat{D}_{ii} < \varepsilon$ will be considered 0
  - Stabilises the regression functions
- Introduce a small norm preference parameter (Tikhonov) $\lambda \geq 0$
- Instead we solve

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

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

$$\Sigma = \left(D^2 + \lambda^2 I \right)^{-1}$$

- λ may be a function $\lambda = \lambda(X, Y)$, see e.g. Wikipedia for a discussion on choosing λ

SVD: conclusion

- SVD is an alternative to regression $\beta = \left(X^T X\right)^{-1}\left(X^T Y\right)$ that is
  - **Stable:** resilient to colinearity
  - **Efficient:** no need to increase number of pre-sims to avoid colinearity

- To implement SVD
  1. Perform SVD on X (see Numerical Recipes) $X = VDU^T$
  2. Denote $\Sigma_{jj} = \frac{1}{D_{jj}^2 + \lambda^2} 1_{\{D_{jj} > \varepsilon\}}$ where λ is Tikhonov's small norm preference and ε is the SVD cut
  3. Regression coefficients are produced with the formula $\beta = UD\Sigma V^T Y$

Thank you

brian.huge@danskebank.dk

antoine@asavine.com