Financial Cash-Flow Scripting: Beyond Valuation

Antoine Savine
antoine@asavine.com

xVA, capital and other regulatory calculations: Computational challenges

Challenge	Industry standard	Danske Bank
Thousands of heterogeneous transactions: aggregation, compression and manipulation	?	Scripting
Heavy, high dimensional Monte-Carlo: hybrid models with multi-factor rates	Distributed parallelism	Multithreaded simulations
Sensitivity to thousands market variables	Distributed bumping, Increasingly AAD	AAD
Future PVs within simulations	Nested simulations, Closed-form approximations, Regression (LSM) proxies	Regression proxies
Accurate LSM proxies	Large number of simulations	Machine Learning: auto-regularization, deep ANNs
Mitigate inaccuracy of LSM proxies	None	[illegible]I (only in indicators)

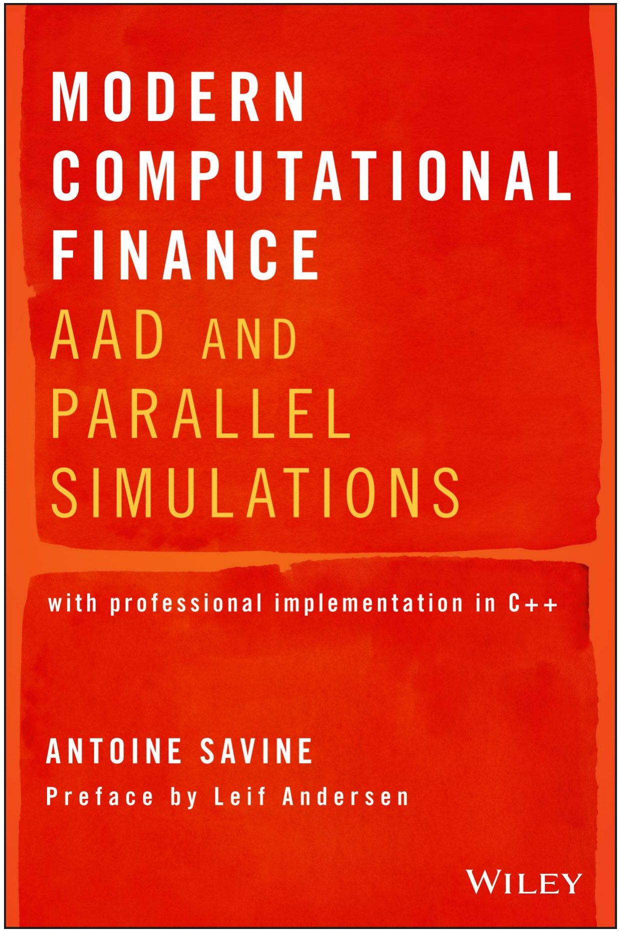
Risk Awards 2015

Regulatory calculations on light hardware

- 2 Options:
  1. Stitch exisiting FO systems, accumulate hardware, use large data centres
  2. Rethink FO systems, write new algorithms to calculate accurately and quickly on light hardware

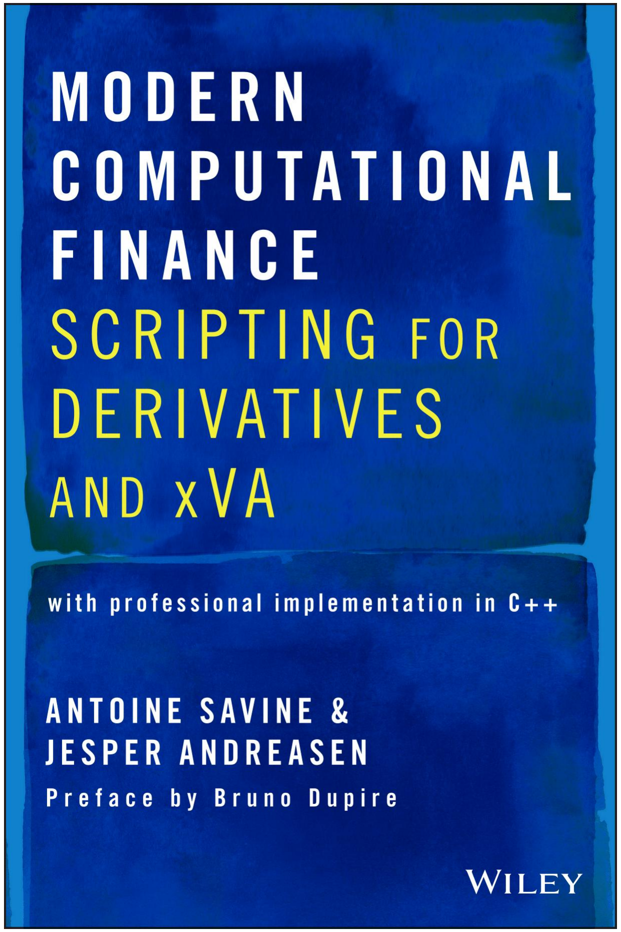
- Danske Bank invested on the second option
  - Wrote FO and XVA system around:
    model hierarchies, cash-flow scripting, multithreaded simulations, LSM and AAD
  - Publically demonstrated sizeable netting set XVA on a laptop in seconds, full risk in minutes
  - Gave series of talks titled ”XVA on iPad Mini”
  - Won the In-House System of the Year 2015 Risk award
  - Now publishing our solutions

Modern Computational Finance (Wiley, 2018)



- All about AAD
  - Complete from mathematics to efficient implementation
  - Detailed coverage of use in finance
  - Textbook style with pedagogical explanations
  - With complete, professional C++ code
- Parallel simulations
  - How to write multithreaded programs in C++
  - How to develop generic, efficient, parallel MC libraries
  - Connection with AAD
- Curriculum for Numerical Finance, MSc Mathematics-Economics, Copenhagen University

Modern Computational Finance (Wiley, 2018)



All about scripting

- Textbook style with complete, pedagogical explanations
- Covers implementation in C++
- And use for risk management and XVA
- With complete, professional C++ code

Modern Computational Finance (Wiley, 2018)

Modern Computational Finance

LSM and other algorithms

For XVA, capital and regulatory calculations

Jesper Andreasen, Brian Huge
and Antoine Savine

Wiley, xMas 2018

- All about LSM
- Effective XVA calculation and differentiation

Introduction to Scripting

Financial Simulations and Modular Library Design

Models
Produce Scenarios

Scenario
Market Variables on Event Dates

Products
Cash Flows function(al)s of Scenario

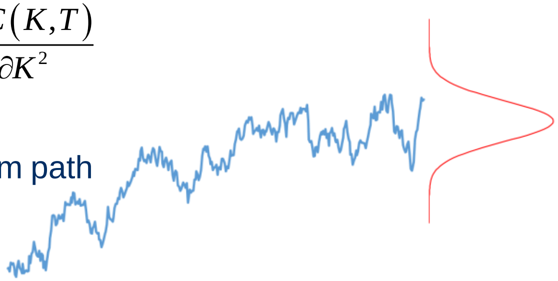
Linear Models
1 scenario : realizes forwards

Smile Models
All call prices for expiry T $C(K,T)$
Scenarios: 1 point from density $f_T(x) = \dfrac{\partial^2 C(K,T)}{\partial K^2}$

Dynamic Models
Monte-Carlo simulations, scenario = random path

- Black-Scholes $dS = S\sigma dW$
- Bachelier $dS = \sigma dW$
- Dupire $dS = \sigma(S,t) dW$
- Stoch. Vol. $dS = S\sqrt{v}dW, dv = -k(v - v_0)dt + \alpha\sqrt{v}dZ$
- SLV $dS = \sigma(S,t)\sqrt{v}dW$
- Merton $dS = S\sigma dW + JdN - \lambda dt$

- Rates (HJM/BGM)
- Multi-underlying
- Multi-currency ➔ quanto effects
- Hybrid models: joint paths for rates, currencies and equities



call	
Date	Event
expiry	opt pays max (0 , spot() - strike)
barrier	
Date	Event
trade date	alive = 1
monitoring sched	if spot() > barrier then alive = 0 endIf
expiry	opt pays alive * max (0 , spot() - strike)
autocallable (simplified)	
Date	Event
trade date	ref = spot() alive = 1
1y	if spot() > ref then opt pays 110 alive = 0 endIf
2y	if alive and spot() > ref then opt pays 120 alive = 0 endIf

3y	if alive then
	if spot() > ref then opt pays 130 else opt pays 50 endIf
	endIf

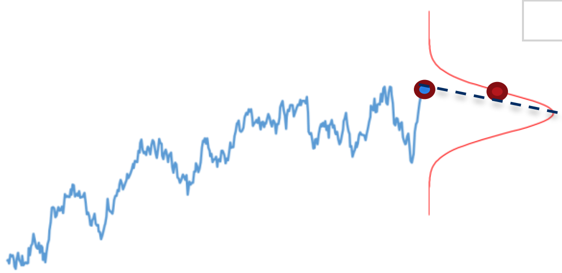
Loose Coupling and Product Scripting

- Loose Coupling
  - Models = dynamics for state and market variables, know nothing about products
    Responsibility: produce scenarios
  - Products = cash flows as functions of path, know nothing about dynamics
    Responsibility: compute payoff = sum of (numeraire discounted) cash flows, given scenario
  - Value = expected payoff = approx. average among scenarios
  - Communication only through scenarios
    - Before simulations: Product communicates event dates and nature of market variables
    - During simulations: Model generates scenarios = market variables on event dates
- Scripting
  - A (programming?) language for describing the cash-flows of the Product
  - Model agnostic
  - Let users *define* products in real time
  - Mix and match products and models on the fly

How does it work?

Model

Scenario



Evaluator

-depth first visitor-

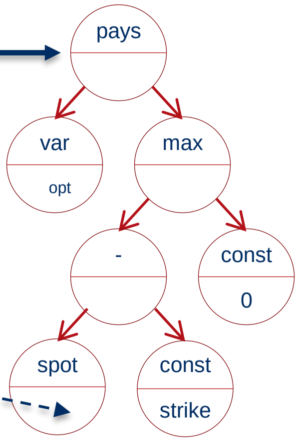
Visit	Node	Stack
1	spot	spot
2	const strike	strike, spot
3	-	strike - spot
4	const 0	0, strike - spot
5	max	max(0, strike - spot)
6	var	&opt, max(0, strike - spot)
7	pays	[empty]
	opt += max(0, strike - spot) ⁄ numeraire	

Product

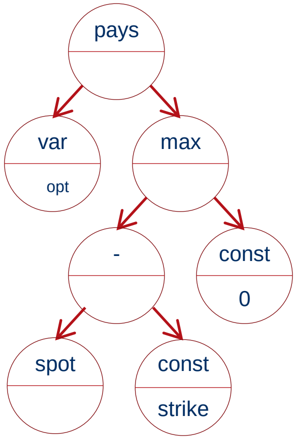
Call	
Date	Event
expiry	opt pays max (0, spot() - strike)

Parser

Expression Tree



Visitors



- Expression Trees
  - Store the structure of cash flows in their DNA
  - Are not black boxes, trees can be traversed – or *visited* – in many ways
- Evaluation against a path = traversal of the tree
  - Depth first: leaves to top
  - Visit to nodes conduct calculations that determine payoff
  - Picking market variables on the path
  - And storing (intermediate) results on the stack
- Evaluation = one particular example of traversal
- Visitor = object that traverses nodes, (generally) depth first
  - Collects information
  - (Possibly modifies nodes)
  - Maintains state

Visitors: Pre Processors

- Pre-processors do their work *before* simulations take place

- Pre-processors collect relevant information about the Product and:
  - Perform all computations to prepare and optimize simulation-time calculations
  - Conduct ”administrative” work: memory allocation, schedule generation, …
  - Index all product and market variables so they are directly accessed in memory during simulations
    - Example: ”x = x + y” becomes ”v[0] = v[0] + v[1]”

- Pre-processors is what make script valuation virtually the same speed as hard coded payoffs

Visitors: Readers

Visitors don’t work with the script (as a string) but with the tree

Some visitors traverse trees and identify:

- Market variables for event dates ➔ scenario definition + indexing

- Non linearities, path dependencies ➔ model selection
- Dependecies between product variables ➔ optimize LSM
- Value domain of product variables and expressions ➔ required for fuzzy logic

- Schedule of fixings and payments ➔ helps middle office processing
- And a lot more

Visitors: Modifiers

Other visitors modify trees directly :

For example, we use 3 visitors to compute xVA

- **Aggregate** cash-flows from transactions within a netting set
- **Compress** cash-flows to align fixings and payments on a master schedule
- **Decorate** transaction scripts to compute xVA and other regulatory calculations

Scripting is not only for Structuring and Exotics

- Not even that convenient for exotics
  - Model still needs to be specified and calibrated so it is relevant for a particular Exotic
  - Although (in theory) visitors can analyze cash-flows and select/calibrate model accordingly
  - And we now have models that calibrate (decently) to everything
- What scripting offers is
  - A consistent representation of all transactions
  - Down to cash flows
  - That can be read and modified by visitors
- Scripts are best suited as an *internal* representation of transactions
- This versatility is generally not well known
  - No publication on the subject
  - No active development in most houses
  - Tendency to limit usage to the structuring of exotics

Script Examples: equity/commodity/forex

Call

STRIKE	120	MATURITY	opt pays CALL (Spot(), STRIKE)
MATURITY	01-Jun-16		
CALL(S,K)	max(0, S-K)		

Barrier

INDEX	fx(EUR, USD)	BARSTART	vAlive = 1
STRIKE	100	Start: BARSTART	If INDEX > BARRIER then vAlive = 0 endIf
BARRIER	120	End: BAREND	
PAYOFF(S, K)	max(0, S-K)	Freq: BARFREQ	
MATURITY	01-Jun-16	Fixing: end	
BARSTART	01-Jun-15	MATURITY	pOpt pays vAlive * PAYOFF(INDEX, STRIKE)
BAREND	01-Jun-16		
BARFREQ	1m		

Basket

loops

TRADEDATE	01-Jun-15	TRADEDATE	loop (0, NSTOCKS) s0[ii] = spot(stocks[ii]) endLoop
EXPIRY	01-Jun-16	EXPIRY	loop (0, NSTOCKS)
STRIKE	0.1		s1 = spot(stocks[ii])
			perf = perf + weights[ii] * (s1 – s0[ii]) / s0[ii]
STOCKS	**WEIGHTS**		endLoop
STOCK1	0.2		opt pays max(0, perf – STRIKE)
STOCK2	0.1		
STOCK3	0.15		
STOCK4	0.25		
STOCK5	0.3		

vectors

predefined vectors

Script Examples: Rates

Swap

STARTDATE	01-Jun-18	Start: STARTDATE	swap pays - libor(StartPeriod, EndPeriod, FLBASIS, FLIDX)
ENDDATE	01-Jun-28	End: ENDDATE	* cvg(StartPeriod, EndPeriod, FLBASIS)
FLFREQ	3m	Freq:FLFREQ	on EndPeriod
FLBASIS	act/360	Fixing: start - 2bd	
FLIDX	L3M	Start: STARTDATE	swap pays CPN * cvg(StartPeriod, EndPeriod, FIXBASIS)
FIXFREQ	1y	End: ENDDATE	on EndPeriod
FIXBASIS	30/360	Freq:FIXFREQ	
CPN	2.00%	Fixing: start - 2bd	

Cap

STARTDATE	01-Jun-18	Start: STARTDATE	cap pays
ENDDATE	01-Jun-28	End: ENDDATE	max(0, libor(StartPeriod, EndPeriod, FLBASIS, FLIDX) - STRIKE)
FLFREQ	3m	Freq:FLFREQ	* cvg(StartPeriod, EndPeriod, FLBASIS)
FLBASIS	act/360	Fixing: start - 2bd	on EndPeriod
FLIDX	L3M		
STRIKE	2.00%		

Swap with cap

STARTDATE	01-Jun-18	Start: STARTDATE	swap pays -
ENDDATE	01-Jun-28	End: ENDDATE	min(CAP, libor(StartPeriod, EndPeriod, FLBASIS, FLIDX))
FLFREQ	3m	Freq:FLFREQ	* cvg(StartPeriod, EndPeriod, FLBASIS)
FLBASIS	act/360	Fixing: start - 2bd	on EndPeriod
FLIDX	L3M	Start: STARTDATE	swap pays CPN * cvg(StartPeriod, EndPeriod, FIXBASIS)
FIXFREQ	1y	End: ENDDATE	on EndPeriod
FIXBASIS	30/360	Freq:FIXFREQ	
CAP	2.00%	Fixing: start - 2bd	

Script Examples: LSM

		Swaption	
TRADEDATE	01-Jun-15	Start: STARTDATE	flCpn = libor(StartPeriod, EndPeriod, FLBASIS, FLIDX)
STARTDATE	01-Jun-15	End: ENDDATE	* cvg(StartPeriod, EndPeriod, FLBASIS)
ENDDATE	01-Jun-25	Freq: FLFREQ	
FLFREQ	3m	Fixing: start-2bd	if vAlive = 1 then swaption pays −flCpn on EndPeriod endIf
FLBASIS	act/360	Start: STARTDATE	fixCPn = CPN * cvg(StartPeriod, EndPeriod, FIXBASIS)
FLIDX	L3M	End: ENDDATE	
FIXFREQ	1y	Freq: FIXFREQ	if vAlive = 1 then swaption pays fixCpn on EndPeriod endIf
FIXBASIS	30/360	Fixing: start-2bd	
CPN	2%	TRADEDATE	vAlive = 1
		STARTDATE-2BD	terminate vAlive when PV(swaption) < 0
		Bermuda	
TRADEDATE	01-Jun-15	Start: STARTDATE	flCpn = libor(StartPeriod, EndPeriod, FLBASIS, FLIDX)
STARTDATE	01-Jun-15	End: ENDDATE	* cvg(StartPeriod, EndPeriod, FLBASIS)
ENDDATE	01-Jun-25	Freq: FLFREQ	
FLFREQ	3m	Fixing: start-2bd	if vAlive = 1 then swaption pays −flCpn on EndPeriod endIf
FLBASIS	act/360	Start: STARTDATE	fixCPn = CPN * cvg(StartPeriod, EndPeriod, FIXBASIS)
FLIDX	L3M	End: ENDDATE	
FIXFREQ	1y	Freq: FIXFREQ	if vAlive = 1 then swaption pays fixCpn on EndPeriod endIf
FIXBASIS	30/360	Fixing: start-2bd	
CPN	2%	TRADEDATE	vAlive = 1
		Start: STARTDATE+2Y	
		End: ENDDATE	terminate vAlive when PV(swaption) < 0
		Freq: FIXFREQ	
		Fixing: end -10bd	
		Exotic Bermuda with Path-Dependence	
TRADEDATE	01-Jun-15	Start: STARTDATE	flCpn = libor(StartPeriod, EndPeriod, FLBASIS, FLIDX)
STARTDATE	01-Jun-15	End: ENDDATE	* cvg(StartPeriod, EndPeriod, FLBASIS)
ENDDATE	01-Jun-25	Freq: FLFREQ	
FLFREQ	3m	Fixing: start-2bd	maxCpn = max(flCpn, maxCpn)
FLBASIS	act/360		
FLIDX	L3M		if vAlive = 1 then swaption pays −maxCpn on EndPeriod endIf
FIXFREQ	1y	Start: STARTDATE	fixCPn = CPN * cvg(StartPeriod, EndPeriod, FIXBASIS)
FIXBASIS	30/360	End: ENDDATE	
CPN	3%	Freq: FIXFREQ	if vAlive = 1 then swaption pays fixCpn on EndPeriod endIf
		Fixing: start-2bd	
		TRADEDATE	vAlive = 1
		Start: STARTDATE+2Y	
		End: ENDDATE	terminate vAlive when PV(swaption, maxCpn) < 0
		Freq: FIXFREQ	
		Fixing: end -10bd	

LSM Regression proxies

LSM

− Founding Papers

Carriere, 1996

Longstaff & Schwartz, 2001

− Modern Implementation Differentiation and Application to xVA

Andreasen, Huge & Savine Wiley, 2018

xVA

xVA

- (Idealized, one-sided, uncollateralized) CVA =
  contingent to default
  0 strike put
  on the future value of the netting set

- Payoff = $\sum_{\text{exposure date } p} \underbrace{1_{\{T_{p-1} \le \tau < T_p\}}}_{\text{default indicator}} \underbrace{\underbrace{\left(1 - R_{T_p}\right)}_{\text{recovery}} \underbrace{\max\left(0, V_{T_p}\right)}_{\text{exposure}}}_{\text{loss given default (LGD)}}$

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

- Where $V_{T_p} = E_{T_p}{}^{RN}\left[\sum_{T_k > T_P} CF_k\right]$

Rewriting xVA

- (Idealized) CVA $CVA = E^{RN}\left[\sum_p \left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}\max\left(0,V_{T_p}\right)\right], V_{T_p} = E_{T_p}^{RN}\left[\sum_{T_k>T_P} CF_k\right]$
- Using $(x)^+ \equiv 1_{\{x>0\}}x$ $CVA = E^{RN}\left[\sum_p \left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}1_{\{V_{T_p}>0\}}V_{T_p}\right]$
- Injecting V (not in indicator) $CVA = E^{RN}\left\{\sum_p \left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}1_{\{V_{T_p}>0\}}E_{T_p}^{RN}\left[\sum_{T_k>T_P} CF_k\right]\right\}$
- Using boxed expectations $CVA = E^{RN}\left\{\sum_p \left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}1_{\{V_{T_p}>0\}}\sum_{T_k>T_P} CF_k\right\}$
- Reversing sums $CVA = E^{RN}\left\{\sum_k \left(\sum_{T_P\le T_k}\left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}1_{\{V_{T_p}>0\}}\right)CF_k\right\}$

$$= E^{RN}\left\{\sum_k \eta_{T_k} CF_k\right\} \text{ where } \eta_{T_k} = \sum_{T_P\le T_k}\left(1-R_{T_p}\right)1_{\{T_{p-1}\le\tau<T_p\}}1_{\{V_{T_p}>0\}} \text{ or } \begin{cases}\eta_0 = 0\\ \eta_{T_{i+1}} = \eta_{T_i} + \left(1-R_{T_{i+1}}\right)1_{\{T_i\le\tau<T_{i+1}\}}1_{\{V_{T_{i+1}}>0\}}\end{cases}$$

- CVA = value of the cash flows discounted by path-dependent process η
- Note future PV V
  - Only intervenes in the equation for η
  - And only inside positivity indicator

POI: Proxies only in Indicators

- Using LSM proxies directly in $CVA \approx E^{RN}\left[\sum_p \left(1-\mathrm{R}_{T_p}\right) 1_{\left\{T_{p-1}\le\tau<T_p\right\}} \max\left(0,\tilde{V}_{T_p}\right)\right]$
  - Accuracy of CVA depends on the accuracy of proxy (to the 1st order both in bias and stderr)

proxy error $\quad \varepsilon \equiv \tilde{V} - V$

1st order exposure error

$$E\left(\tilde{V}^+\right) = E\left[\left(V+\varepsilon\right)^+\right] \approx E\left[V^+ + 1_{\{V>0\}}\varepsilon\right] = E\left(V^+\right) + \underbrace{E\left[1_{\{V>0\}}\varepsilon\right]}_{\substack{=P(V>0)E(\varepsilon)+corr\left(1_{\{V>0\}},\varepsilon\right)\sqrt{P(V>0)\left[1-P(V>0)\right]Var(\varepsilon)} \\ \approx P(V>0)E(\varepsilon)+corr\sqrt{P(V>0)}std(\varepsilon) \\ \neq 0}}$$

  - Either spend massive CPU time in LSM to produce accurate proxies or accept the inaccuracy

- If we use proxies *only in indicators*
  - Accuracy of CVA *independent* (to 1the 1st order) on the accuracy of the proxy

proxy error $\quad \varepsilon \equiv \tilde{V} - V$

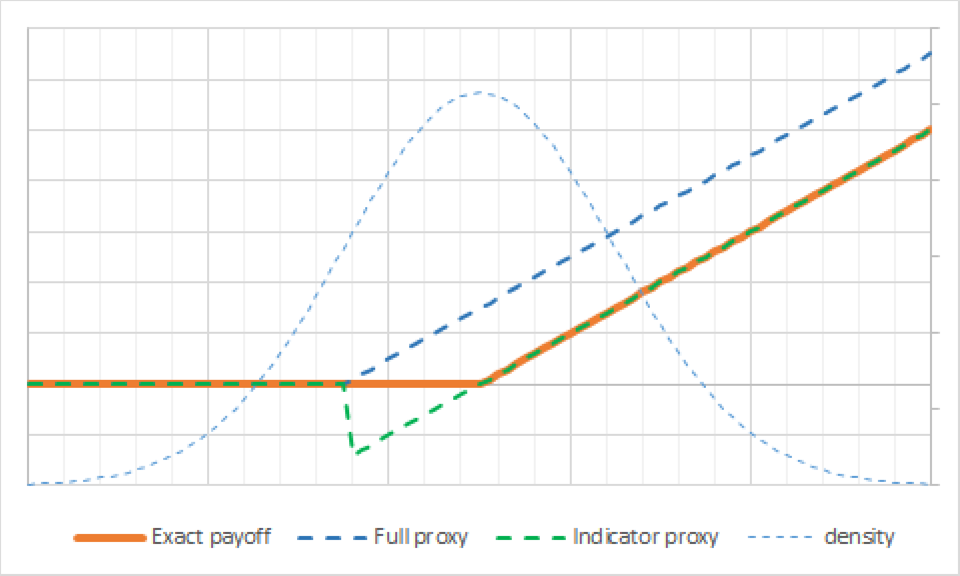
1st order exposure error

$$E\left(1_{\{\tilde{V}>0\}} V\right) = E\left[1_{\{V+\varepsilon>0\}} V\right] \approx E\left[1_{\{V>0\}} V + \delta(V) V \varepsilon\right] = E\left(V^+\right) + \underbrace{E\left[\delta(V) V \varepsilon\right]}_{=0}$$

  - We have an accurate CVA even with quick proxies

POI: illustration

- Example: biased proxy
  - 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
  - Proxy payoff: $1_{\{\tilde{V} > K\}}\left(V - K\right)$
  - Error magnitude: $dens\left(V == K\right) \cdot \mu^{2}$, order μ^{2}



Putting it all together

- CVA = value of cash-flows discounted with path-dependent process η

$$CVA = E^{RN}\left\{\sum_k \eta_{T_k} CF_k\right\} \text{ where } \eta_{T_k} = \sum_{T_p \le T_k} \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1} \le \tau < T_p\}} 1_{\{V_{T_p} > 0\}}$$

- Using LSM proxies in place of future PVs: $\eta_{T_k} \approx \sum_{T_p \le T_k} \left(1 - \mathrm{R}_{T_p}\right) 1_{\{T_{p-1} \le \tau < T_p\}} 1_{\{\tilde{V}_{T_p} > 0\}}$
- We end up with an algorithm that is:
  - **Accurate:** (to the 1^{st} order) proxies are only used in indicators
  - **Efficient:** value xVA for the cost of netting set valuation (+ simulation of discounting process)
  - **Practical:** CF valuation scripts are *decorated* so payments are discounted by η
- The same holds for some other xVAs
  - Immediate for DVA, FVA
  - With adjustments for RWA, kVA (see Flyger-Huge-Savine, 2015-2016)

xVA with Collateral

- Fully collateralized CVA with MPR θ:

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

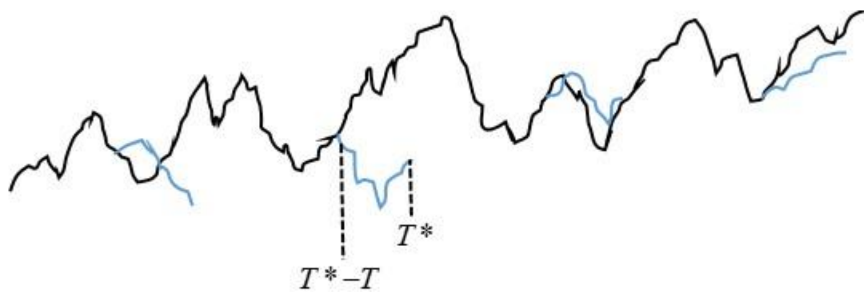
- That **cannot** be directly written as $CVA = E\left[\sum_k \eta_{T_k} CF_k\right]$
  - Consider one exposure: $e(T^*) = E\left[\max\left(0, V_{T^*} - V_{T^*-\vartheta}\right)\right] = E\left[1_{\{V_{T^*} > V_{T^*-\vartheta}\}} V_{T^*}\right] - E\left[1_{\{V_{T^*} > V_{T^*-\vartheta}\}} V_{T^*-\vartheta}\right]$
  - 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 Simulations

- 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 is **not** like 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 *picked 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 filtration at the margin date $T^{mrg} = T - \vartheta$, the main and secondary branches 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

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

- And the collateralized CVA

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

- All the comments from the uncollateralized case apply
  - Essentially the same algorithm
  - Except the discounting process is simulated with branches

Practical xVA: Aggregation

- Before computing xVA, all transactions in a netting set must be aggregated
- This is generally a conundrum
- Scripted transactions are *relatively* easy to merge
- Note we need a proper visitor
  - Merging scripts is not enough
  - We must also duplicate all payments on a single variable that represents the netting set

01-Sep-15	swap1 pays n1 * libor (3Sep2015, 3m, act/360, L3) * cvg11 on 3Dec2015
01-Dec-15	swap1 pays n1 * libor (3Dec2015, 3m, act/360, L3) * cvg12 on 3Mar2016

15-Oct-15	swap2 pays n2 * libor (17Oct2015, 3m, act/360, L3) * cvg2 on 17Jan2016

01-Sep-15	cpn = n1 * libor (3Sep2015, 3m, act/360, L3) * cvg11, swap1 pays cpn on 3Dec2015, ns pays cpn on 3Dec2015
15-Oct-15	cpn = n2 * libor (17Oct2015, 3m, act/360, L3) * cvg2, swap2 pays cpn on 17Jan2016, ns pays cpn on 17Jan2016
01-Dec-15	cpn = n1 * libor (3Dec2015, 3m, act/360, L3) * cvg12, swap1 pays cpn on 3Mar2016, ns pays cpn on 3Mar2016

Practical xVA: Compression

- Avoid linear growth of CPU and memory load with number of transactions
- Align event dates on a master schedule
- De-interpolate fixings and payments on surrounding dates in the master schedule
- Aggregate notionals
- We call this process “compression”
  - Conducted by specialized (and complicated) visitors
  - Fully automated leveraging on the visitor design

01-Sep-15	cpn = n1 * libor (3Sep2015, 3m, act/360, L3) * cvg11, swap1 pays cpn on 3Dec2015, ns pays cpn on 3Dec2015
15-Oct-15	cpn = n2 * libor (17Oct2015, 3m, act/360, L3) * cvg2, swap2 pays cpn on 17Jan2016, ns pays cpn on 17Jan2016
01-Dec-15	cpn = n1 * libor (3Dec2015, 3m, act/360, L3) * cvg12, swap1 pays cpn on 3Mar2016, ns pays cpn on 3Mar2016

↓

01-Sep-15	cpn = (n1 + 0.5 * n2) * libor (3Sep2015, 3m, act/360, L3) * cvg11, ns pays cpn on 3Dec2015
01-Dec-15	cpn = (n1 + 0.5 * n2) * libor (3Dec2015, 3m, act/360, L3) * cvg12, ns pays cpn on 3Mar2016

Practical xVA: Decoration

- We have our (compressed) schedule for all cash flows in the NS:

CFDate1	ns pays …
	ns pays …
	…
	ns pays …
…	…
CFDateN	ns pays …
	ns pays …
	…
	ns pays …

- We want to compute:

$$CVA = E\left[\sum_k \eta_{T_k} CF_k\right]$$

$$\eta_0 = 0, \eta_{T_{i+1}} = \eta_{T_i} + \left[1 - R_{T_i}\right]\left[S\left(T_i\right) - S\left(T_{i+1}\right)\right]1_{\left\{V_{T_i}>0\right\}}$$

Decoration (2)

- We duplicate all payments, multiply them by η:

CFDateI	ns pays ...
	ns pays ...
	...
	ns pays ...
	cva pays nu * ...
	cva pays nu * ...
	...
	cva pays nu * ...

- And add an additional schedule for the simulation of η:

Start: TODAY	if PV(ns) > 0 then
End: FINALDATE	nu = nu + (1 – rec(StartPeriod))
Freq: CVAFREQ	* (surv(StartPeriod) – surv(EndPeriod))
Fixing: end	endIf

- This is conducted by a dedicated visitor called *decorator*

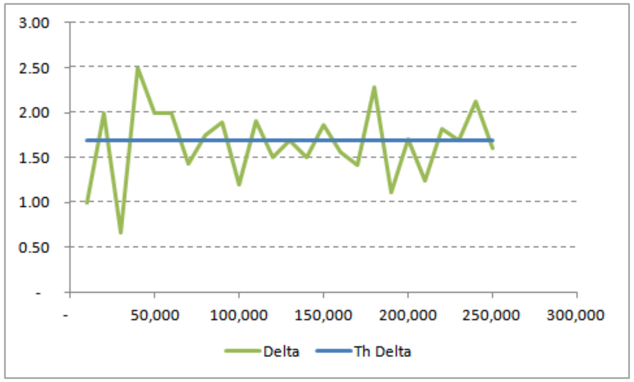
Fuzzy Logic

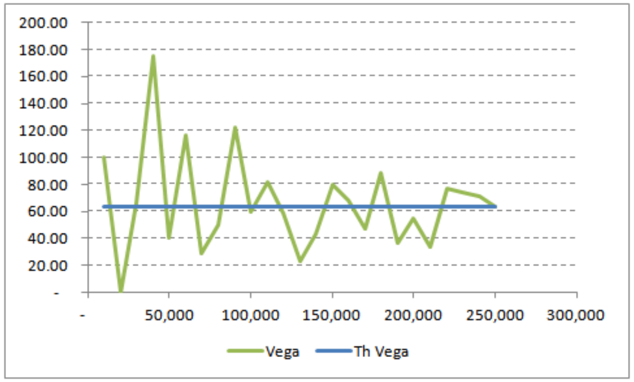
Discontinuous payoffs with Monte-Carlo

- Discontinuous profiles produce unstable risks with Monte-Carlo
  - Example: 110 1y Digital in Black-Scholes MC, vol = 20%
  - 10,000 to 250,000 simulations
  - Seed changed between pricings
  - Decent convergence on price, risk all over the place

Black-Scholes	
Today	10-Apr-15
Spot	100.00
Vol	20%
Rate	0%

10-Apr-16	if spot() > 110 then dig pays 100 endIf



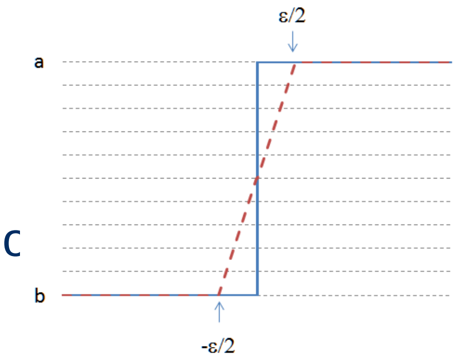


Payoff smoothing

- Payoff smoothing is a simple (and yet surprinsingly effective) method:

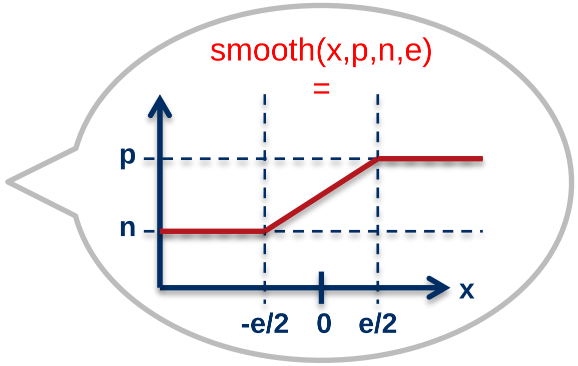
Replace discontinuous payoffs by "close" continuous ones

  - Digital → Call spread
  - Barrier → Smooth barrier



- One benefit: work on payoffs only, no work required on mc
- One problem: must identify and smooth all discontinuities
- Today, this is a manual process:

Traders rewrite payoffs one by one
using a "call-spread" or "smooth" function:

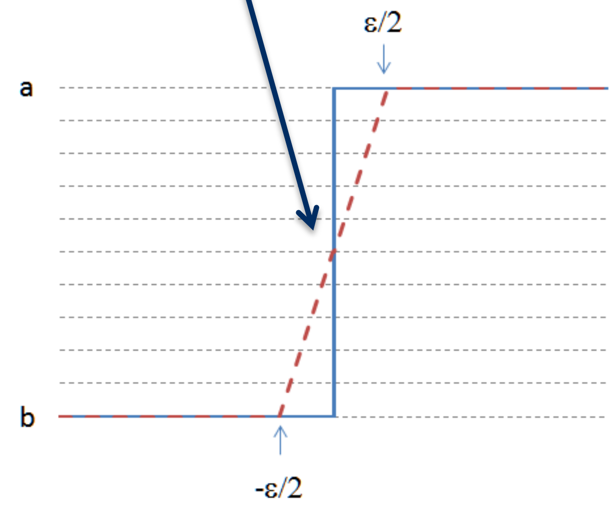


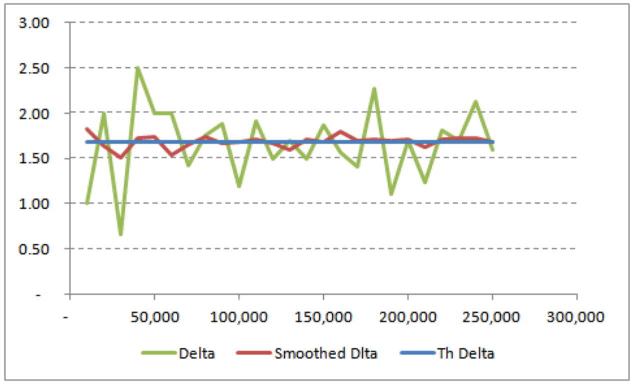
Smoothing a digital

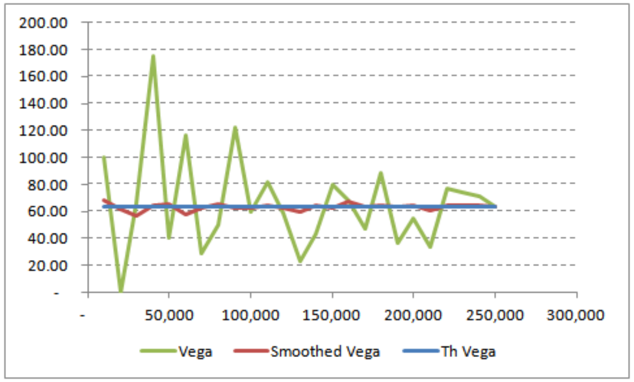
10-Apr-16	if spot() > 110 then dig pays 100 endIf

10-Apr-16	dig pays smooth(spot() - 110, 100, 0, 1)

- Digital → Call Spread
- Dig = $-\frac{\delta C}{\delta K}$
  → CS = finite difference *continuous* approximation
- Alt. interpretation: spread the notional evenly across strikes between (K-e/2,K+e/2)

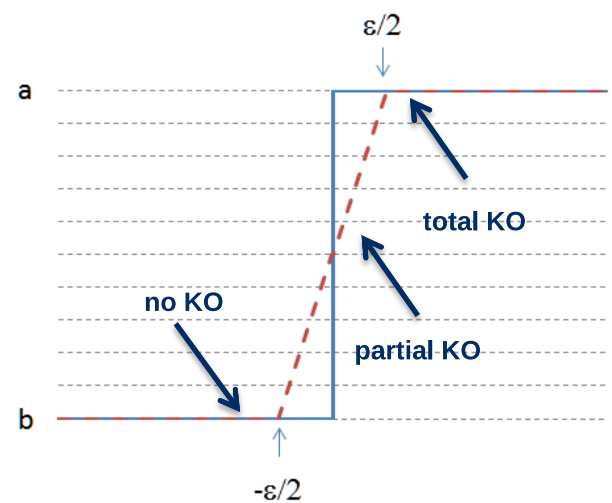






Smoothing a barrier

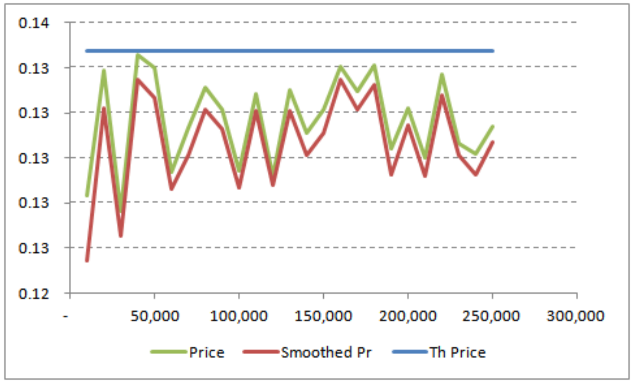
- Spread notional across barriers in (B-ε/2,B+ε/2)
- Example: 1y 110-120 RKO in Black-Scholes (20%)
- Weekly monitoring, barrier shifted by 0.60 std
- Smooth barrier: knock-out part of the notional depending on how far we land in (B-ε/2,B+ε/2)

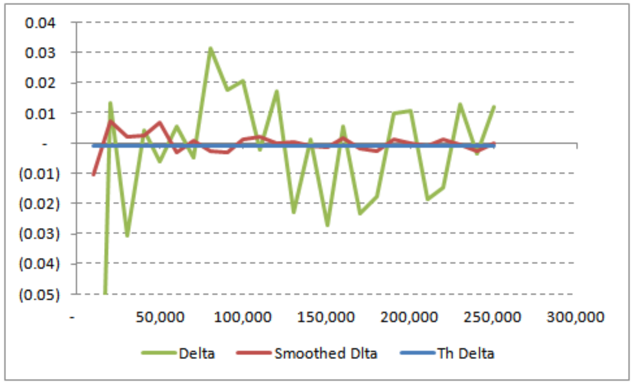


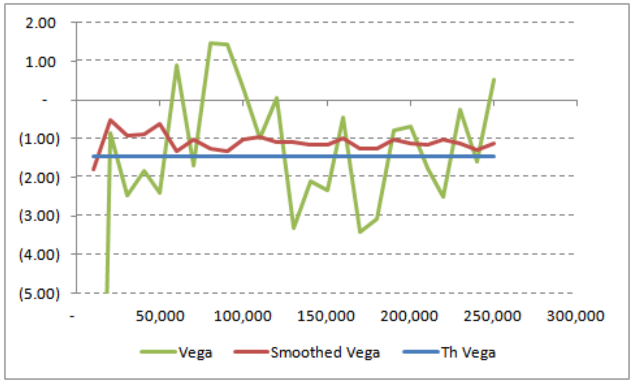
10-Apr-15			alive = 1
10-Apr-15	10-Apr-16	weekly	if spot() > 118.36 then alive = 0 endif
10-Apr-16			rko pays alive * max(spot() - 110, 0)

➡

10-Apr-15			alive = 1
10-Apr-15	10-Apr-16	weekly	alive = alive * smooth(spot() - 118.36, 0, 1, 1)
10-Apr-16			rko pays alive * max(spot() - 110, 0)







Smoothing everything

- Many transactions with discontinuous profile besides digitals and barriers
- Popular example: simplified Autocallable
  - If spot declines 3 years in a row, lose 50% of notional
  - Otherwise make 10% per annum until spot raises above initial level

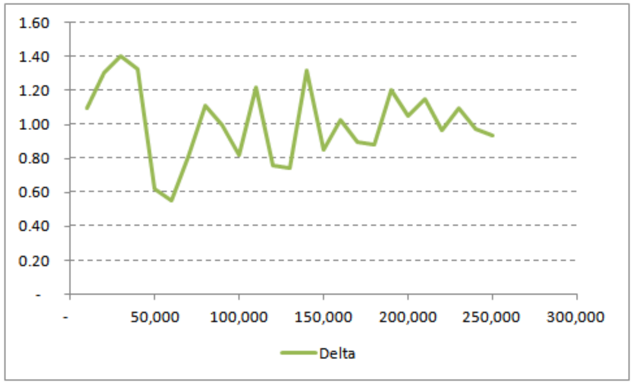
10-Apr-15	ref = 100 alive = 1
10-Apr-16	if spot() > ref then autocall pays 110 alive = 0 endIf
10-Apr-17	if spot() > ref then autocall pays alive * 120 alive = 0 endIf
10-Apr-18	if spot() > ref then autocall pays alive * 130 else autocall pays alive * 50 endIf

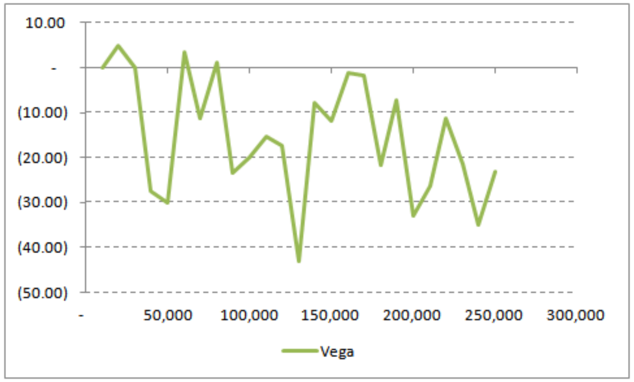
- All those products can’t be reliably risk managed unless smoothed
- Traders must manually apply smoothing for every (type of) transaction
- We describe that process, assuming a scripting platform

Identifying discontinuities

- In programming, incl. scripting, discontinuities come from *control flow*
  - *If this then that* pattern
  - Calls to discontinuous functions (meaning themselves implement *if this then that* statements)
- Autocallable:
  - 3 control flows
  - Unstable risk

10-Apr-15	ref = 100 alive = 1
10-Apr-16	if spot() > ref then autocall pays 110 alive = 0 endIf
10-Apr-17	if spot() > ref then autocall pays alive * 120 alive = 0 endIf
10-Apr-18	if spot() > ref then autocall pays alive * 130 else autocall pays alive * 50 endIf



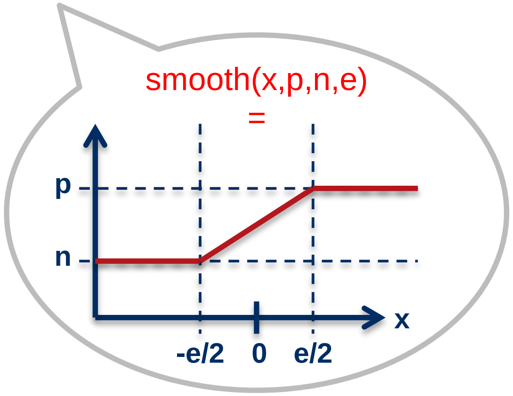


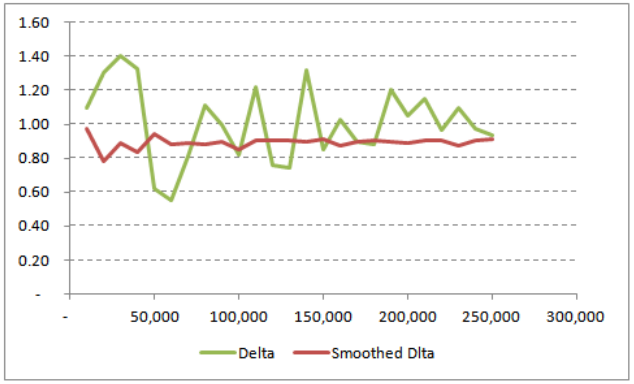
Manual smoothing

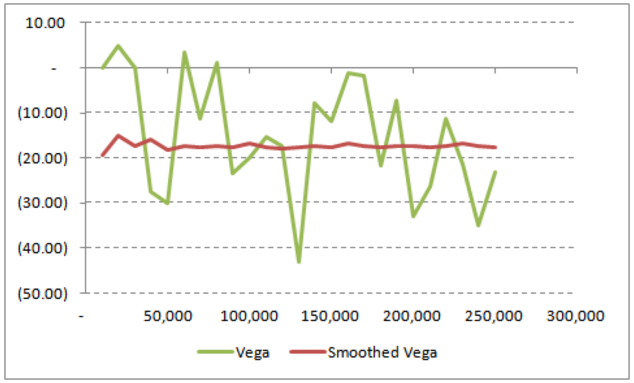
- Replace *if this then that* by *smooth* (remindeer: smooth = ”call spread” function)

10-Apr-15	ref = 100 alive = 1
10-Apr-16	if spot() > ref then autocall pays 110 alive = 0 endIf
10-Apr-17	if spot() > ref then autocall pays alive * 120 alive = 0 endIf
10-Apr-18	if spot() > ref then autocall pays alive * 130 else autocall pays alive * 50 endIf

10-Apr-15	ref = 100 alive = 1
10-Apr-16	autocall pays smooth(spot() - ref, 110, 0, 1) alive = smooth(spot() - ref, 0, 1, 1)
10-Apr-17	autocall pays alive * smooth(spot() - ref, 120, 0, 1) alive = alive * smooth(spot() - ref, 0, 1, 1)
10-Apr-18	autocall pays alive * smooth(spot() - ref, 130, 50, 1)







- Does the job, but hard to write and read and error prone, even in simple cases

Automatic smoothing

- Automatically turn
  - Nice, easy, natural scripts

10-Apr-15	ref = 100 alive = 1
10-Apr-16	if spot() > ref then autocall pays 110 alive = 0 endIf
10-Apr-17	if spot() > ref then autocall pays alive * 120 alive = 0 endIf
10-Apr-18	if spot() > ref then autocall pays alive * 130 else autocall pays alive * 50 endIf

  - Into smoothed scripts that produce stable risks

10-Apr-15	ref = 100 alive = 1
10-Apr-16	autocall pays smooth(spot() - ref, 110, 0, 1) alive = smooth(spot() - ref, 0, 1, 1)
10-Apr-17	autocall pays alive * smooth(spot() - ref, 120, 0, 1) alive = alive * smooth(spot() - ref, 0, 1, 1)
10-Apr-18	autocall pays alive * smooth(spot() - ref, 130, 50, 1)

- Visitor technology means :
  Program knows the script and can apply smoothing the same way traders do
- In practice, rule based algorithms invalidated by counter examples

Fuzzy Logic to the rescue

- The one realization that brings everything together:
  smoothing = use of Fuzzy Logic
- In classical (sharp) logic, a condition is *true* or *false*
  - Schrodinger’s cat is dead or alive
- In fuzzy logic, a condition is true *to a degree* = degree of truth or DT
  - Schrodinger’s cat is dead *and* alive
  - For instance, it can be alive with DT 65% and dead with DT 35%
- Note: DT is not a probability
  ➔ Financial interpretation: condition applies to DT% of the notional
  - A barrier is 65% alive means 35% of the notional knocked out, the rest is still going
  - It does not mean that it is hit with proba 35%, which makes no sense on a given scenario

Fuzzy Logic

- Invented in the 1960s by Lotfi Zadeh
- Not to smooth financial risks
- But to build the first expert systems

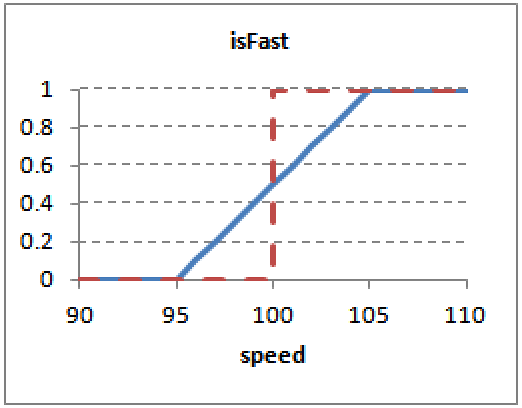
- Automate expert opinions
  - Expert: it is dangerous for the engine to be fast and hot
  - Programmer: what do you mean by ”fast” and ”hot”?
  - Expert: say fast = ”>100mph” and hot = ”>50C”

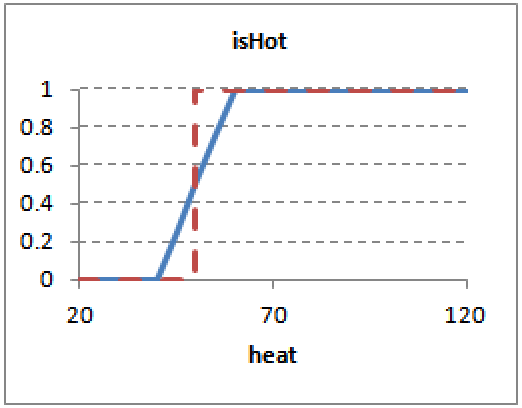
  ➔Code: if speed > 100 and heat > 50 then danger = true else danger = false endIf

  ➔Makes no sense: (99.9,90) → safe, (100.1,50.1) → dangerous !!

More Fuzzy Logic

- With fuzzy logic:
  - isFast and isHot are DTs = smooth functions resp. of speed and heat valued in [0,1]





  - DTs combine like booleans $DT[C_1 \text{ } and \text{ } C_2] = DT_1 \cdot DT_2$, $DT[C_1 \text{ or } C_2] = DT_1 + DT_2 - DT_1 \cdot DT_2$ alt. $DT[C_1 \text{ } and \text{ } C_2] = \min(DT_1, DT_2)$, $DT[C_1 \text{ or } C_2] = \max(DT_1, DT_2)$

➔Code: isDangerous = fuzzyAnd(isFast(speed), isHot(heat))

➔Results make sense: (99.9,90) → 50% dangerous, (100.1,50.1) → 25% dangerous

Smoothing = Fuzzy Logic

- Remarkable result:
  established smoothing exactly is application of Fuzzy Logic
  - ✓ Digital smoothing = apply fuzzy logic to *if spot>strike then pay 1*
  - ✓ Barrier smoothing = apply fuzzy logic to *if spot>barrier then alive = 0*
- That explains, in hindsight, why smoothing performs so remarkably well
  - ✓ Not a hack
  - ✓ (In hindsight) based on established scientific theory
- And gives us the means to correctly, automatically smooth everything
  - ✓ Smoothing everything = apply fuzzy logic to the evaluation of *all* conditions
  - ✓ Without modifying scripts in any way

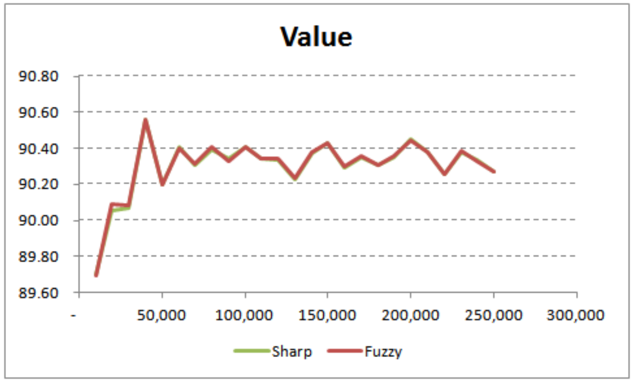
We derive the evaluator into a fuzzy evaluator that overrides visits to if nodes and conditions and evaluates them with Fuzzy Logic

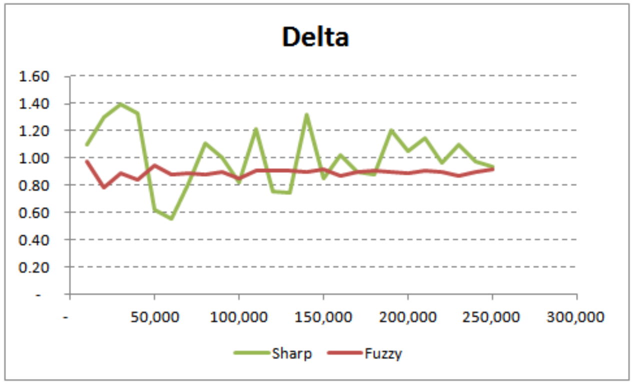
Evaluation of ”if this then that” with fuzzy logic

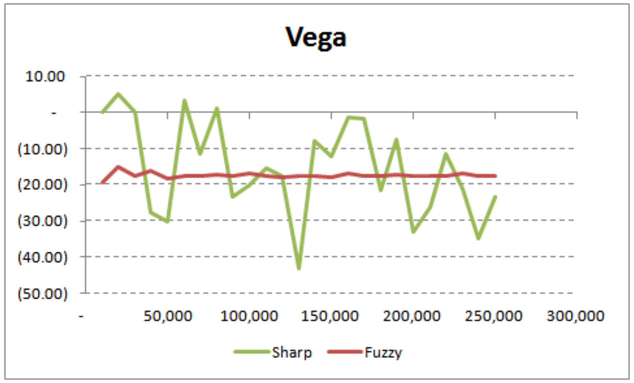
- Evaluation with sharp logic
  1. Evaluate condition C
  2. If C is true, evaluate statements between then and else or endIf
  3. If C is false, evaluate statements between else and endIf if any
- Evaluation with fuzzy logic
  1. Record state S0 (all variables and products) just before the ”if” statement
  2. Evaluate the *degree of truth* (DT) of the condition
  3. Evaluate ”if true” statements
  4. Record resulting state S1
  5. Restore state to S0
  6. Evaluate ”else” statements if any
  7. Record resulting state S2
  8. Set final state S = DT * S1 + (1-DT) * S2

Fuzzy Logic : results

- No modification to scripts
- Override evaluation of *if this then that* statements, conditions and combinators
- Effectively stabilises risks sensitivities with minimal PV impact
- (With proper optimization) just as fast as normal
- Easily flick between sharp (pricing) and fuzzy (risk) evaluation







Thank You

scriptingCentral@asavine.com