Danske Bank


# Stabilize risks of discontinuous payoffs with Fuzzy Logic

Antoine Savine

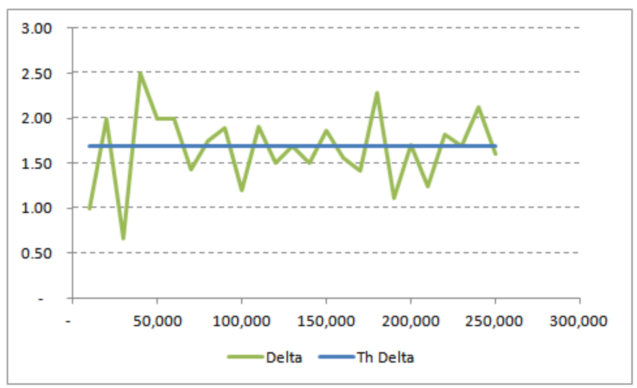


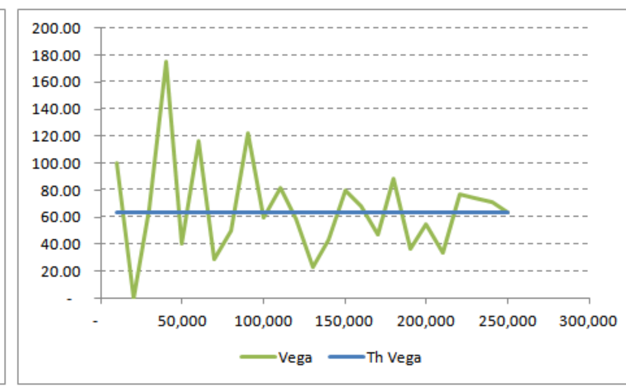
# Discontinuous payoffs with Monte-Carlo

- 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 |



# Malliavin's calculus

- The industry developed 2 families of methods to deal with this problem:
- One is the Likelihood Ratio Method (LRM) and more general Malliavin's calculus
  - Apply

$$\frac{\partial}{\partial a_i} E\vec{a}\left[f\left(\vec{X}\right)\right] = \frac{\partial}{\partial a_i}\int_{\vec{X}} f\left(\vec{X}\right)\varphi\left(\vec{a},\vec{X}\right)d\vec{X}$$

$$= \int_{\vec{X}} f\left(\vec{X}\right)\varphi_{a_i}\left(\vec{a},\vec{X}\right)d\vec{X} = \int_{\vec{X}} f\left(\vec{X}\right)\frac{\varphi_{a_i}\left(\vec{a},\vec{X}\right)}{\varphi\left(\vec{a},\vec{X}\right)}\varphi\left(\vec{a},\vec{X}\right)d\vec{X}$$

$$= \int_{\vec{X}} f\left(\vec{X}\right)\left[\log\varphi\left(\vec{a},\vec{X}\right)\right]_{a_i}\varphi\left(\vec{a},\vec{X}\right)d\vec{X} = E\left\{f\left(\vec{X}\right)\left[\log\varphi\left(\vec{a},\vec{X}\right)\right]_{a_i}\right\}$$

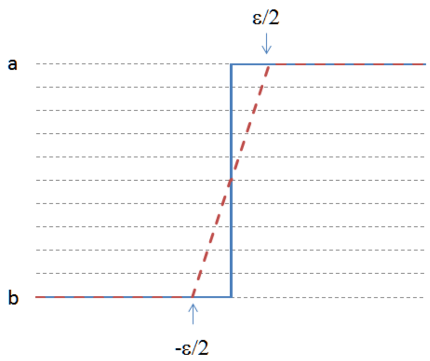
  - Compute $\left[\log\varphi\left(\vec{a},\vec{X}\right)\right]_{a_i}$ pathwise
  - No need to differentiate a discontinuous payoff, differentiate log-likelihood instead
  - Tractability, efficiency and stability depend on model



# Payoff smoothing

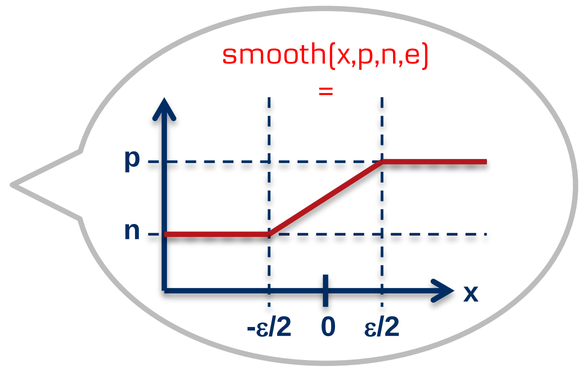
- Payoff smoothing is considerably simpler (and surprinsingly effective):

*Replace discontinuous payoffs by ”close” continuous ones*




  - Digital → Call spread
  - Barrier → Smooth barrier
- Benefit: work on payoffs only, no work required on models
- Problem: to identify (and smooth) all discontinuities
  we need to know exactly *how the payoff is calculated*
- Today, this is a manual process:

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

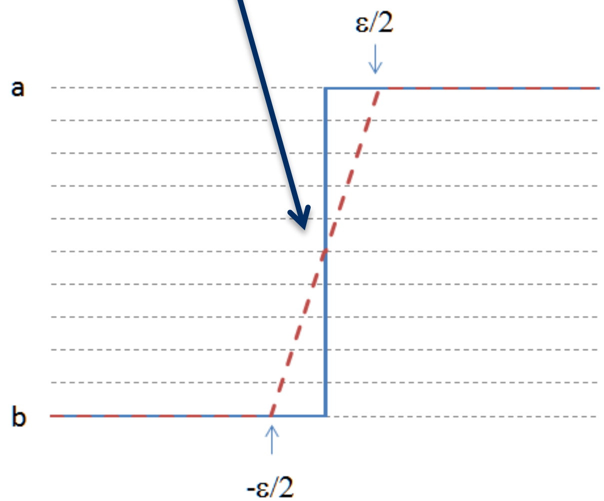


# Smoothing a digital

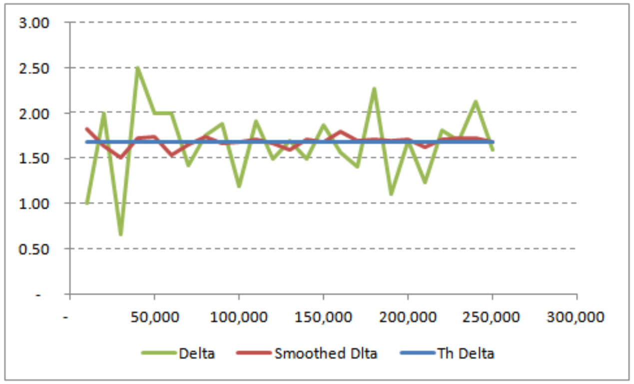
10-Apr-16 | if spot() > 110 then dig pays 100 endIf

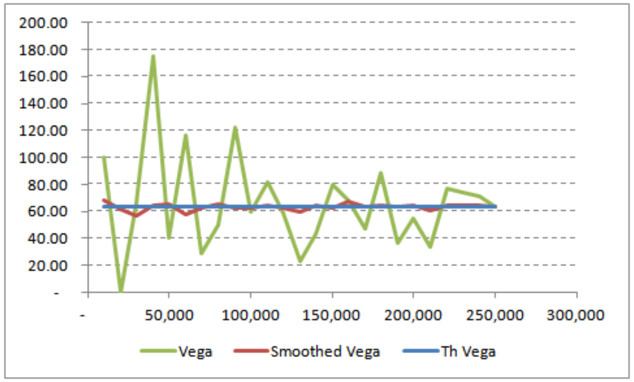
10-Apr-16 | dig pays smooth( spot() - 110, 100, 0, 1)

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



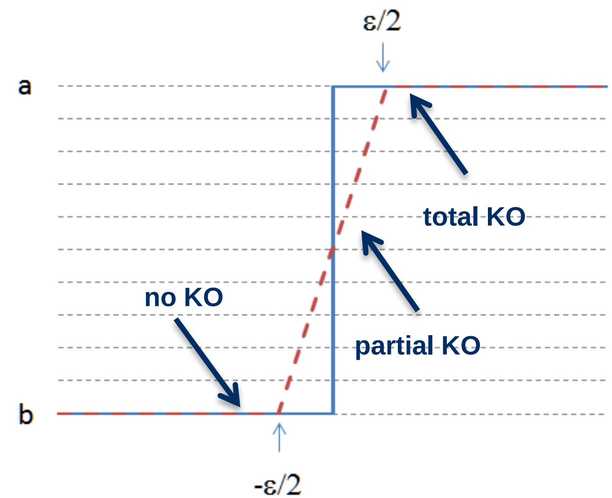




# Smoothing a barrier

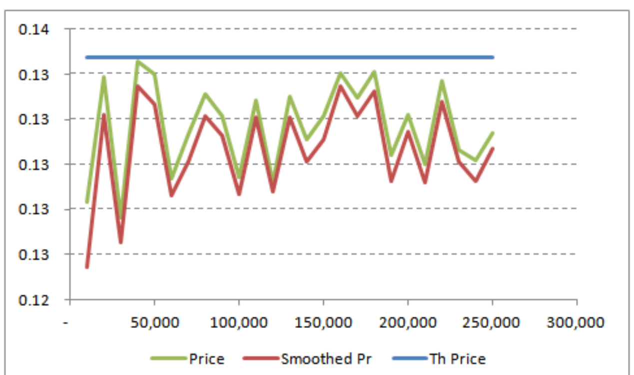
- Spread notional across barriers in (B-ε/2,B+ε/2)
- Example: 1y 110-120 RKO in Black-Scholes (20%)
- Weekly monitoring, barrier shifted by 0.60 std
- Smooth barrier: knock-out part of the notional depending on how far we land in (B-ε/2,B+ε/2)

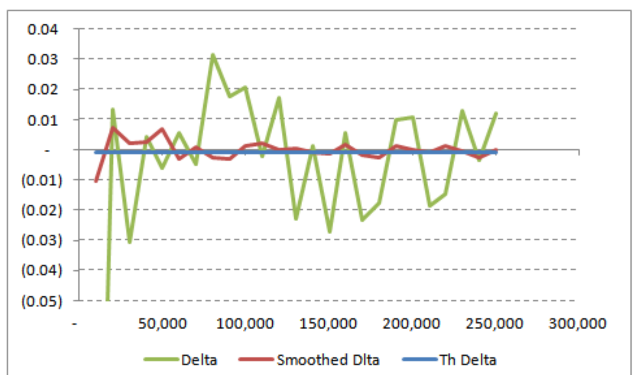


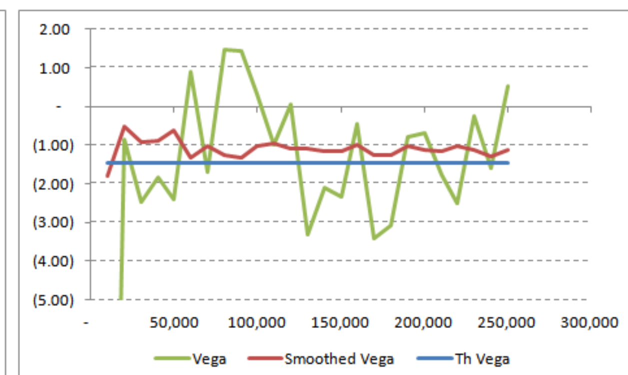

| | | | |
|---|---|---|---|
| 10-Apr-15 | | | alive = 1 |
| 10-Apr-15 | 10-Apr-16 | weekly | if spot() > 118.36 then alive = 0 endif |
| 10-Apr-16 | | | rko pays alive * max( spot() - 110, 0) |

➡

| | | | |
|---|---|---|---|
| 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

- How can we generalize digital/barrier smoothing to *any* payoff?

- We need to identify (and smooth) all discontinuities
- Hence, we need to know exactly how the payoff is calculated
- We need access to the ”source code” of the payoff

- ”Hard coded” payoffs only allow evaluation against scenarios: ”compiled code”
- But *scripted* payoffs provide full information: ”source code”

# Scripting Languages

- Simple ”programming” language optimized for the description of cash-flows
- Describe the cash-flows in any transaction: vanillas, exotics, even CVA/KVA
- And then evaluate the script with some model
- 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 |



# The many benefits of scripting cash-flows

- Run-time pricing and risks of transactions
  - Users create/edit a transaction by scripting its cash-flows
  - Then produce value and risk by sending the script to a model
- Homogeneous representation of all cash flows in all transactions
  - Software can manipulate, merge and compress cash-flows across transactions
  - Crucial for ”derivatives on netting sets” like VAR/CVA/XVA/KVA/RWA
- Crucially for us, provide ”source code” cash-flow information to the software
  - So the software can, among (many) others:
    - Obviously, evaluate cash-flows in different scenarios
    - Detect contingency, path-dependence or early exit and select the appropriate model
    - Optimise the calculation of value and risk sensitivities
    - Detect (and smooth) discontinuities

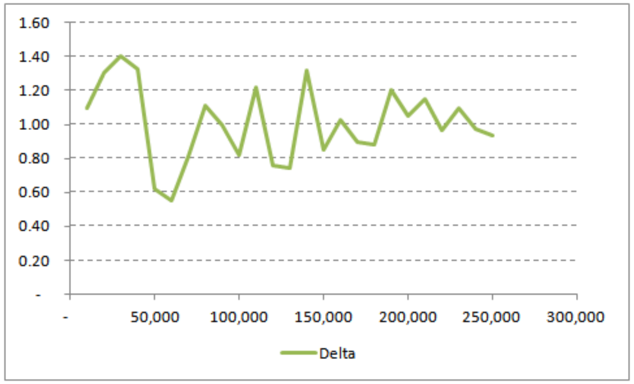


# Identifying discontinuities

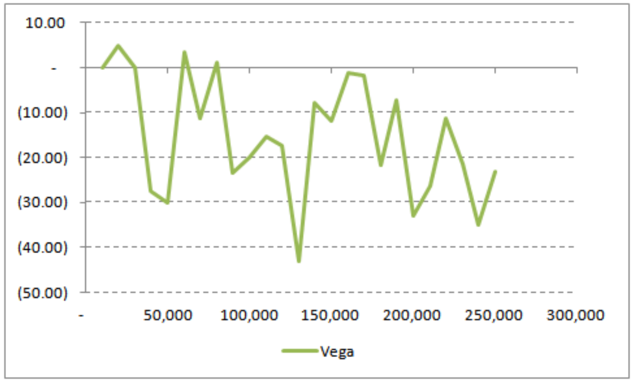
- In programming, incl. scripting, discontinuities come from *control flow*
  - *If this then that* pattern
  - Call to discontinuous functions (meaning themselves implement *if this then that* statements)
- Autocallable:
  - 3 control flows
  - Unstable risk

| | |
|---|---|
| 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 |









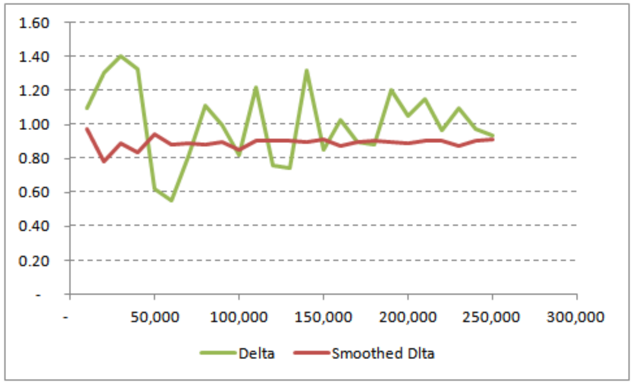


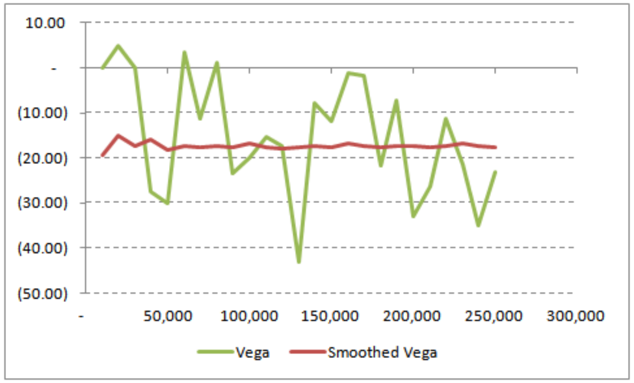
# Manual smoothing

- Replace *if this then that* by *smooth*

| | |
|---|---|
| 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) |

- We have access to the script → we can identify control flows
- We only need to smooth them all

# Fuzzy Logic to the rescue


- Turns out we don't need to transform the script at all!
- We just need to *evaluate* differently
  the conditions and the conditional statements
- 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 65% alive and 35% dead
- 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
- Not to smooth financial risks
- But to build the first expert systems

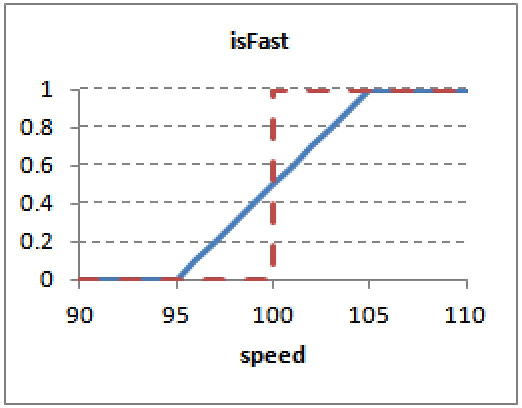
- Automate expert opinions
  - Expert: it is dangerous for the engine to go fast and be hot
  - Programmer: what do you mean by ”fast” and ”hot”?
  - Expert: say fast = ”>100mph” and hot = ”>50C”

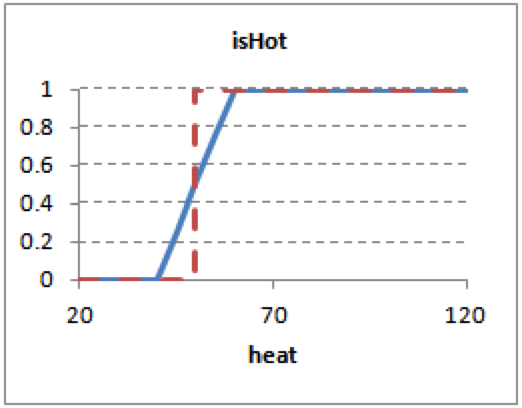
  ➔Code: if speed > 100 and heat > 50 then danger = true else danger = false endIf

  ➔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

$$DT[C_1 \ and \ C_2] = DT_1 \cdot DT_2$$
$$DT[C_1 \text{ or } C_2] = DT_1 + DT_2 - DT_1 \cdot DT_2$$

alt.

$$DT[C_1 \ and \ C_2] = \min(DT_1, DT_2)$$
$$DT[C_1 \text{ or } C_2] = \max(DT_1, DT_2)$$

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

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



# Back to payoff smoothing

- Digital smoothing = apply fuzzy logic to *if spot>strike then pay 1*
- Barrier smoothing = apply fuzzy logic to *if spot>barrier then alive = 0*

- Classical smoothing *exactly* corresponds to
  the replacement of sharp logic with fuzzy logic in the evaluation of conditions

- Reversely, we can systematically apply fuzzy logic to evaluate all conditions
- Fuzzy logic for control flow effectively removes discontinuities
- And stabilises risk sensitivities

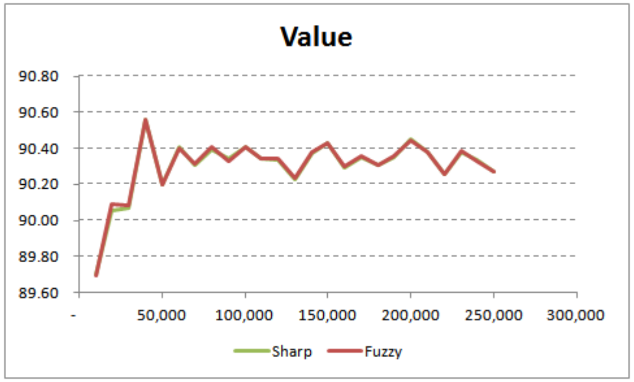
# Evaluation of ”if this then that” with fuzzy logic

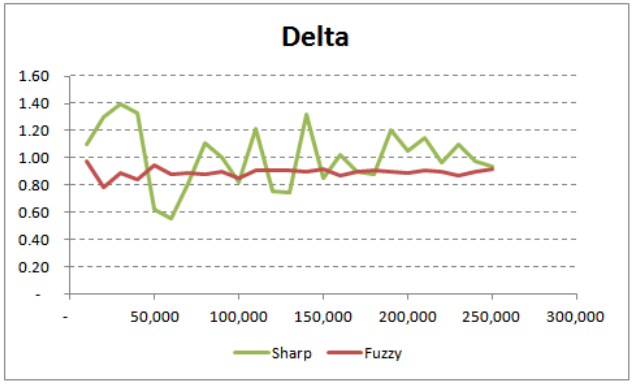
- Evaluation with sharp logic
  1. Evaluate condition C
  2. If C is true, evaluate statements between then and else or endIf
  3. If C is false, evaluate statements between else and endIf if any
- Evaluation with fuzzy logic
  1. Record state S0 (all variables and products) just before the ”if” statement
  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

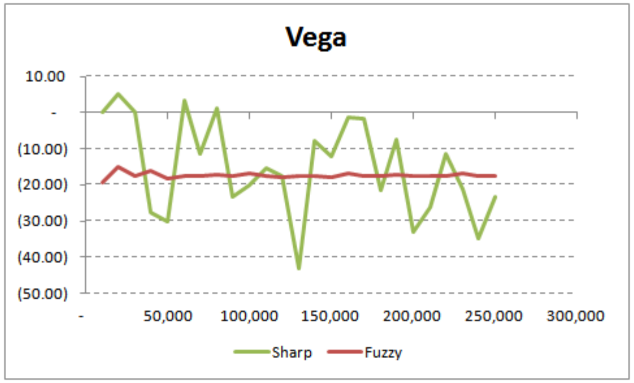


# Fuzzy Logic implementation

- No change in scripts
- Change the evaluation of *if this then that* statements only
- 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



# Advanced Fuzzy Logic: condition domains

- Consider the script

| | |
|---|---|
| 10-Apr-16 | digPays = 0 |
| 10-Apr-16 | if spot() > 110 then digPays = 1 endIf |
| 10-Apr-16 | if digPays > 0 then DigOpt pays 100 endIf |

- The variable *digPays* is valued in {0,1}
- To apply fuzzy logic / call spread to *digPays > 0* makes no sense
- Evidently the DT of *digPays > 0* is *digPays* itself

- This is easy to correct but we must first compute the domains of all conditions
- Fortunately, we have access to the script
- We can write code that traverses the script and calculates condition domains
- This is not as hard as it sounds



# Advanced Fuzzy Logic: beware the cat


- We are used to conditions being true or false
  - Schrodinger’s cat is dead or alive
  - The cat cannot be dead and alive
  - If the cat is alive then it can’t be dead, etc.

- We may write scripts based on such logic, like for instance:

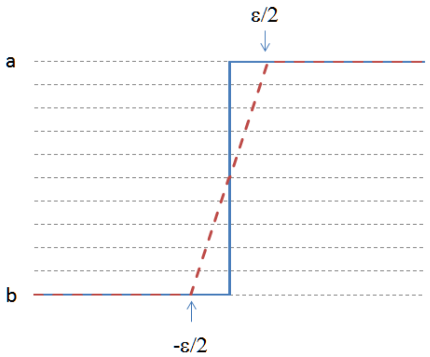
| |
|---|
| x=100 |
| if spot() > 100 then x = 1 endIf |
| if spot() <= 100 then x = 0 endIf |

*X has to be overwritten by either 0 or 1 right?*

- With fuzzy logic, where spot > 100 to *a degree* and <= 100 to *a degree*
  - The logic that underlies the script is not applicable
  - And the result is a bias in the value by orders of magnitude!
  - With fuzzy logic, best stick to the decription of cash-flows and refrain from injecting additional ”logic”

# Advanced Fuzzy Logic: what epsilon?

- Fuzzy logic applies a ”call spread” to conditions:
  - Too wide → risk of valuation bias
  - Too narrow → unstable risks
  - What is a reasonable size?




- Typical ε for a condition *expr* > *0* = fraction of the std dev of *expr*
- Alternative = set ε such that Proba ( - ε/2 < expr < ε/2) = target

- Solution: use *pre-simulations* to estimate the first 2 moments of all conditions
- And set their epsilon accordingly