foveSIMM
a formally verified SIMM

In a nutshell

- **OpenSIMM**: ISDA margin model for non-cleared derivatives
  - does the model *always* perform properly?

- **formal methods**: *proof* for *all* cases, not *testing* of *many*
  - faster, cheaper, surer, eases regulatory approval
  - Intel, Facebook, Airbus, NASA, Amazon, DARPA, Microsoft

- **foveSIMM:** a proof-of-concept formally verified SIMM
  - *proves* margin payments *always* increase in the confidence level, are defined, scale with portfolio size and with confidence level
  - *generates* performant, executable Scala code

OpenSIMM

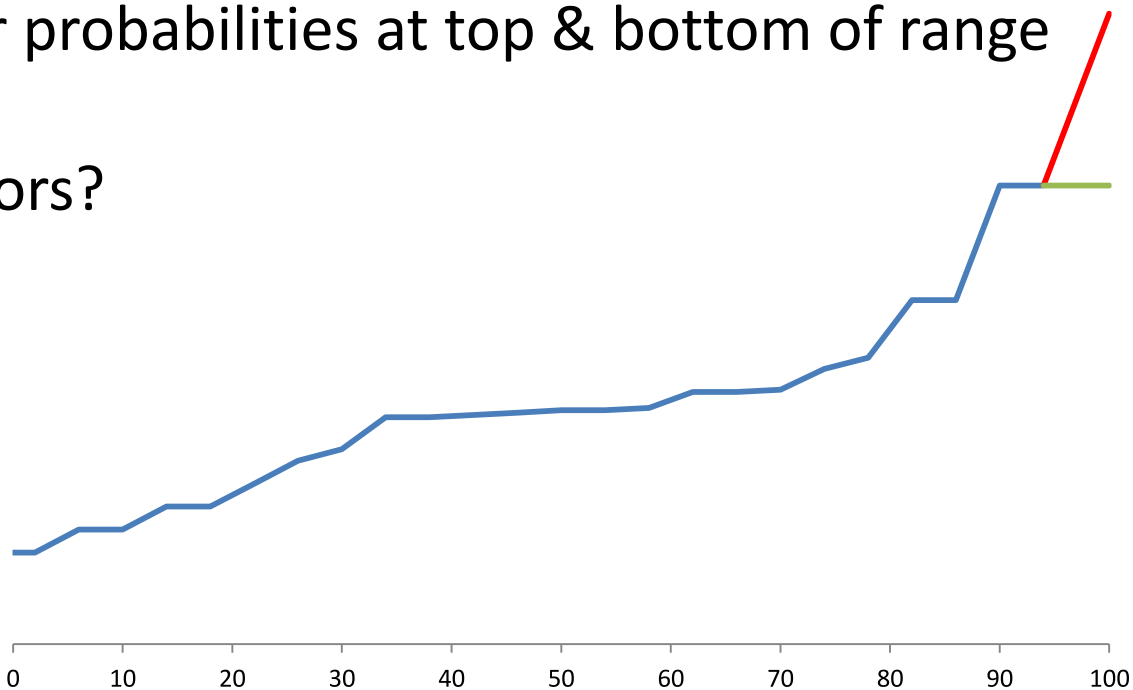
- reference implementation of ISDA's Standard Initial Margin Model (SIMM)
- written in Java 8 (heavily using its functional features)
- version used commercially is not open source

Trusting OpenSIMM

- built around a **percentile function**, $f(L, p)$
  - L, a list of shocks
  - p, a probability

- OpenSIMM fails for probabilities at top & bottom of range

- are there other errors?



Traditional development cycle (**small model** or **small change**)

1. quant builds a prototype (c. 1 month)
2. functional specs, business requirements (c. 2 months)
3. developers write code (c. 2 months)
4. testing (c. 1 month)
5. validation (c. 1 month)
    - shows correct performance on test cases
6. regulatory approval (c. 3-9 months)
    - shows correct performance on test cases

- scales poorly: new cases and interactions to test

Formal development cycle

1. quant prototype is functional, extracts production code
    - writes definitions, theorems (properties of the model)
    - with prover, writes *proofs* for *all* cases

2. validators, regulators can focus on definitions, theorems
    - prover: general purpose, open source, small core

- scales well: more libraries mean more proven results
    - wrote boundedness proof strategy in c. 1 minute
    - prover then found proof automatically in c. 20 seconds

- faster, cheaper & better risk management

Preliminary theorems

1. **bounded**: $f(L,p)$ is always less than its value at $p = \bar{p}$

2. **monotonic**: $f(L,p)$ weakly increases over $p \in [0,1]$

- thus, $f(L,p)$ is *defined* for *all* $p \in [0,1]$
  (Java's was undefined above top, $\bar{p}$ & below bottom, $\underline{p}$)

3. **homogeneity**: $f(L,p)$ scales with the portfolio size

$$f(\alpha \times L, p) = \alpha \times f(L, p)$$

e.g. doubling portfolio, *always* doubles margin

Lipschitz continuity theorem

- knowing the margin payment at p_1 bounds it at *any* p_2

$$f(L, p_2) - f(L, p_1) \leq k \times (p_2 - p_1)$$

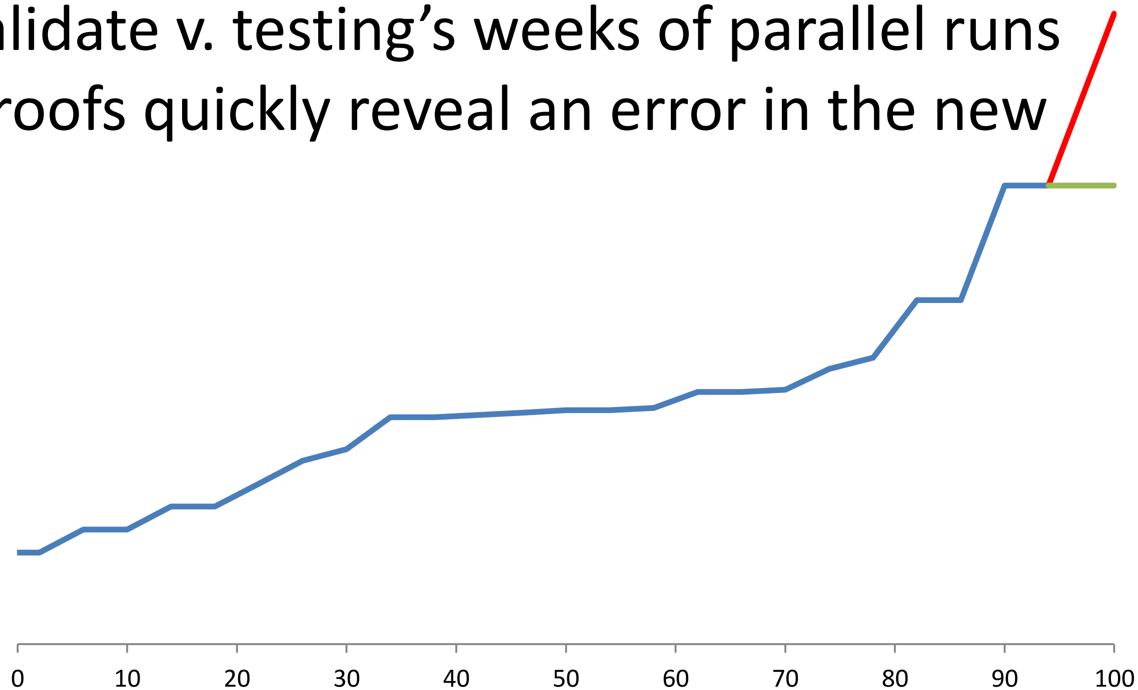
- typically, a model change like this would require
  - 3 weeks of parallel runs
  - back/stress-testing on *all* relevant portfolios

- no need to re-compute: it's proven

Validating a model change

- horizontal extension to $f(L, p)$ is very conservative

- we change the definition of $f(L, p)$ and re-run the proofs
  - losses increase above $\bar{p}$ at the *worst* historic rate
  - 10 hours to re-validate v. testing's weeks of parallel runs
  - problems with proofs quickly reveal an error in the new definition



Comparing executable code

OpenSIMM's Java	verified Scala
c. 1,700 lines of code	c. 3,000 lines of code
<1 sec run time	<1 sec run time
excludes data files c. ½ are comments	includes data files code unoptimised

- can also generate executable code in Haskell, OCaml, SML

sorry

- a proof fails if any step in it fails

- can insert a **sorry** statement in the code
  - prover skips that step, accepting it as true
  - eases modular code development

- can then have operational risk discussion:
  1. take the risk, and use the unverified code
  2. use traditional software engineering to mitigate the risk
  3. remove it by verifying it

to do list

1. confirm the formulae for the IR asset class
2. read data from same input files as OpenSIMM
   - use OpenSIMM's I/O routines, plugging Scala into Java
3. refine Isabelle code
   - use existing library results, not our own
   - eliminate 'sorry' statements
4. optimise the executable Scala code
5. 'sign' the executable code to certify its origins