Payoff Scripting Languages: Sung and Unsung Glories and the Next Generation

QuantMinds
Vienna
May 2019

Jesper Andreasen
SaxoBank, Hellerup
kwant.daddy@saxobank.com

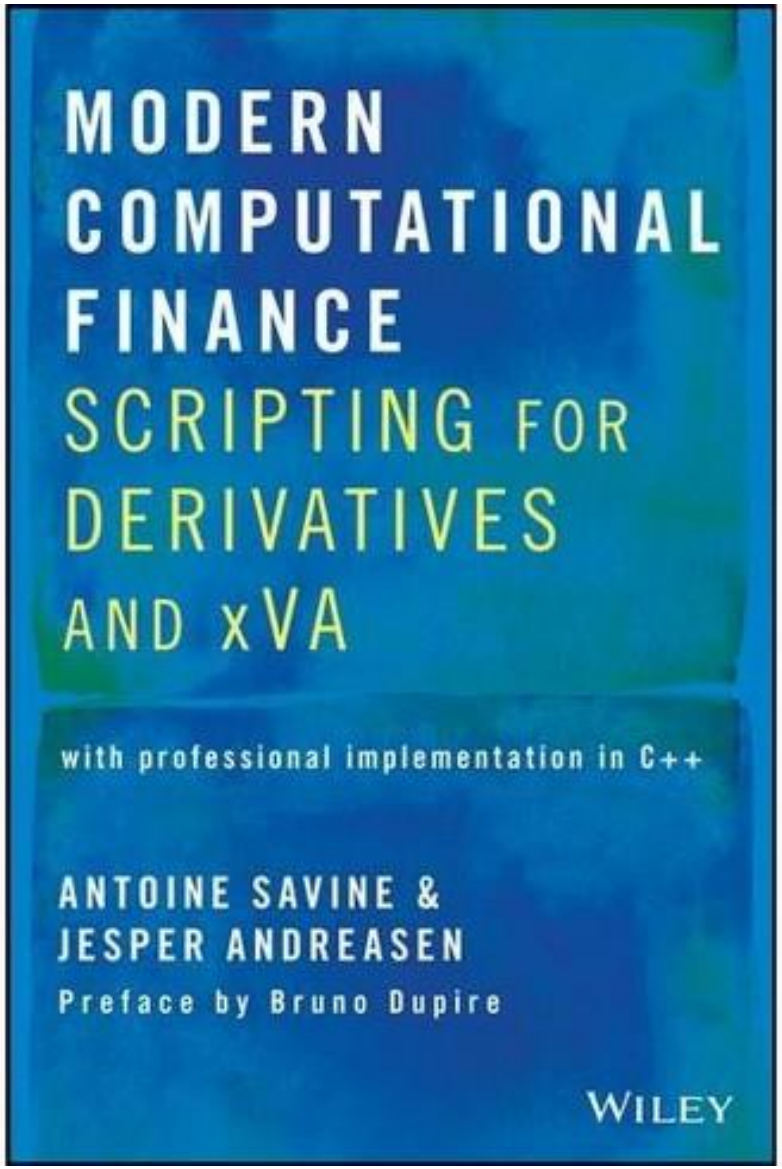
Outline

- Introduction:
  - What is a scripting language?
  - My life with scripts.
  - The mess of banks' IT systems.
- First generation: Exotic derivatives.
- Second generation: XVA, AAD, and capital.
- Third generation: Calculus of transactions, trading strategies, trade life cycle, and front-to-back digital banking.
- Conclusion.

References

- Andreasen, J (2014): “XVA on your iPad Mini”. *Global Derivatives*.

- Andreasen, J and A Savine (Soon): “Scripting for Derivatives and XVA.” *Wiley*.


MODERN
COMPUTATIONAL
FINANCE
SCRIPTING FOR
DERIVATIVES
AND xVA
with professional implementation in C++
ANTOINE SAVINE &
JESPER ANDREASEN
Preface by Bruno Dupire
WILEY

Acknowledgements

- This presentation builds on big stack of largely undocumented work that I have done with various collaborators over the past 20 years.

- Particularly: Antoine Savine, James Pfeffer, Ove Scavenius, Hans-Jørgen Flyger, Brian Huge.

What is a Payoff Scripting Language?

- A better term for it is Domain Specific Language (DSL) developed for a specific purpose.
- In our case the purpose is to describe the cash flows of a derivative (or any financial contract) – and thereby the derivative.
- That and only that.
- So not general purpose languages as C++, VBA, Python, etc.
- In my case the scripting languages are home written in C++ and have over time carried different names: SynTech (GRFP), Thor (BofA), Loke (Nordea), Jive (Danske), Jife (Saxo).

My Life with Scripts

- I have been using scripting languages nearly since the birth of time – 1998 to be precise.
- Initially (1998-2008): scripting used as a flexible tool for pricing (and booking) of exotic derivatives.
- Later (2008-present): scripting used for handling XVA and regulatory capital.
- In fact, the scripting technology has been essential for being able to do XVA with AD on the iPad mini.
- Along the way, the syntax has changed very little but what we do underneath has been constantly expanded.

The Mess

- We all know that IT systems of banks are a mess.
- Daily operations are held together by glue, gaffer tape and a lot of manual work.
- It’s like a factory with a lot of conveyer belts.
- But the conveyer belts are poorly connected and things constantly fall off them. Flow is “fixed” with manual processes.
- To limit costs, banks have moved processes to lower cost countries. But real innovation has been very limited.
- … and a consolidated capital calculation is still a daytrip.

- It may be that clients interact digitally with their bank, and that orders are executed automatically, with the latest and greatest in maths and machine learning.

- … but when it comes to the life cycle of the trade and the controls around it – there is still a lot of binders and faxes and manual hand holding involved.

- This is one of the reasons that approximately 2% of the GDP is eaten by the financial sector.

- In order to solve this, cash flows, transactions, and processes need to be documented. In code -- not on yellow stickers or in binders.

- Scripting is a perfect vehicle for doing this in a consistent manner.

- In the following I will go through what we have done with scripting languages over the past 20 years.
- … and what we are currently doing to expand our scripting language and its potential use.
- Particularly, with respect to cleaning up the operational flow.

1st Generation: Exotic Derivatives

- In the late 90s and early 00s many banks developed tools for describing arbitrary cash flows.

- Typically to be used with Monte-Carlo models for the pricing of exotic derivatives.

- This was to make it possible for traders and structurers to directly price new stuff – without having quants to constantly write and compile new payoff code in C++.

Implementation Problems

- However, the implementations typically suffered from one or more of the problems. For example:

  - The use of full scale interpreter languages (like VBA) combined with C for the simulations => poor speed.

  - The scripting language would be tied to only one model or asset class => poor versatility and lack of generality.

  - Weird syntax, like reverse Polish notation => difficult to use.

  - Only build for pricing => no support for life cycle.

SynTech (1998) [Syntactic Interpreter]

- Antoine Savine's SynTech language (Gen Re FP) was a major step forward:
  - Natural Basic style syntax [still used].
  - Written in C++: statements parsed into trees of C++ objects.
  - Significant amounts of processing (traversing) before actual pricing =>
  - No sacrifice of speed.
  - Could hook with any model: interest rates, equities, fx, finite difference, Monte-Carlo, whatever, …
  - Back-office functionality: fixing and payments.

Syntax

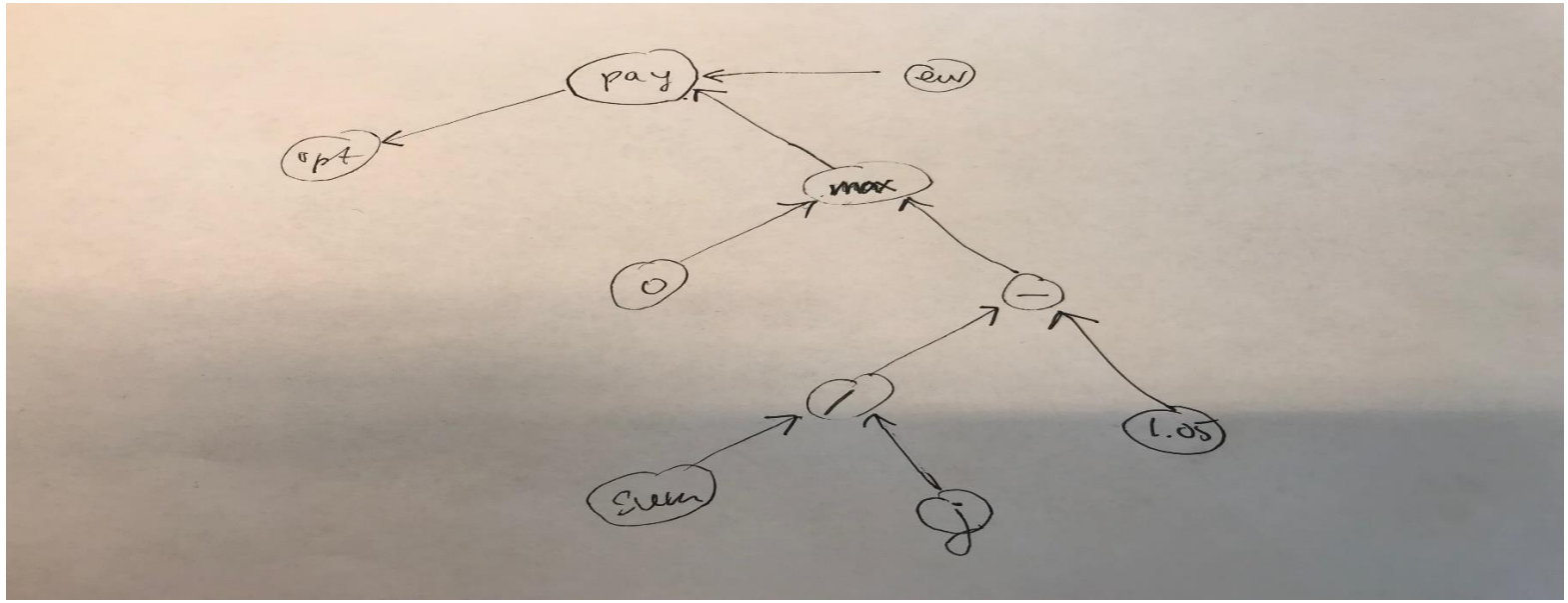
- Here's an example of a script for an Asian quanto option:

start	end	freq	event
01Apr20			s0 = equity(sp500)
01Apr20	01Apr25	3m	sum = sum + equity(sp500)/s0 j = j + 1
01Apr25			opt pay(max(0, sum/j - 1.05), eur)

- So relatively straightforward but not simpler than that.
- Total of 50 keywords: +-*/=<>, max(), min(), …, equity(), libor(), …
- Has stayed close to constant for 20 years.

Trees

- On run-time each statement gets parsed (recursive decent) into a tree of C++ objects. For example:



$$\boldsymbol{opt = opt + num(eur) * max(0, sum/j - 1.05)}$$

Processing

- The trees are traversed to extract (static) information such as
  - Simulation time line.
  - Underlying equity.
  - Payment currency.
  - Etc.
- This information is passed to the model so it can align itself.
- Times, indexes, currencies, etc, are fixed [or at least finite state].

Evaluation

- The value of the stock (equity(sp500) in usd) at the dates (1apr2020, 1jul2020, …, 1apr2025) and the numeraire (eur) is simulated by the model and supplied to the scripting language that evaluates the payoff.

- Note the distinct split of responsibilities:

  - The script handles the payoff. Knows nothing about how it will be evaluated.

  - The model generates the path of the stock and numeraire. But know nothing about the actual payoff, only what it depends on.

- The key observation here is that it is the processing that makes communication between model and script possible.

- … and in turn makes it possible to cleanly split in payoff and model.

Thor (2001) and Loke (2002)

- C++ visitor pattern. Split scripting language code in:
  - Data: Trees of C++ objects. Const after parsing.
  - Processes: C++ visitors to run on the trees. For example: Processing, valuation, early exercise, …
- The benefit of this is that code for processes can produced in parallel.
- Different programmers can write different visitors without stepping on each other's toes.
- It may seem like a small thing but it actually makes the C++ code a lot simpler.

- Advanced pricing features in Thor and Loke:

  - American Monte-Carlo.
  - Including upper bound.
  - Brownian bridge and absorption for barriers.

2nd Generation: Jive (2008)

- At Danske we used Jive for everything.
- Including vanilla swaps, bonds, and later loans and deposits.
- This enabled us to consistently aggregate risk across trades and books for XVA and capital calculations.
- We added a lot of functionality (visitors) to enable aggregation of trades across books and clients.
- Compression and aggregation in Jive: to add up trades and compress (distribute payments) to a predefined coarse date schedule.

- A macro overlay called decoration: to add call or XVA features on top of aggregated and compressed scripts.
- Decoration were directly accessible for the user.
- So that traders could script for example break clauses or collateral options on portfolios and compute the XVA implications.
- Real time and on their iPad minis – of course.
- Same technology used for capital calculations: counterparty credit risk and market risk (FRTB).
- Further, we did AAD in Jive by C++ templatizing the evaluator visitors.

- Also, we integrated branching in Jive.
- So we had a pretty good track on all things associated with calculations.
- But for political and job security reasons, it was never used to the extent it could have been on the mid and back office side.
- For deal capture we used conventional systems and few people would actually know that Jive was used underneath.

3rd Generation: Transactions

- Jive (and earlier) is purely a payoff language.
- It doesn't natively describe transactions and cash positions.
- You can synthetize the ***risk*** of holding a stock -- as a one second cash settled forward contract.
- But you'll need something around Jive to roll that forward contract.
- We need to expand Jive to Jife and include general transactions.

Calculus of Transactions

- Let A,B be two different units, and let a,b be real numbers.
- For units, think gold and Apple shares.
- Then we can define

$$aA+bB=[aA,bB]$$

- … in a vector sense. Transactions are these vectors.
- The present value operator $V(\cdot)$ maps transactions on the real axis and corresponds to division

$$V(aA,bB)=\frac{a}{b}\underbrace{V(A,B)}_{\substack{\text{the value of } A \\ \text{in units of } B}}$$

- … equivalent to a foreign exchange rate.
- Inspired by this we are now ready do our first transaction in Jife.

Script of Transaction

time	event
t	swap rec(2, beer,)
t+1h	swap rec(-5, eur,)
18:00	jesper rec(1, swap, brian)

- Here I have used rec() rather than the conventional pay().
- So at 18:00 Jesper receives 2 beers from Brian and pays him 5 EUR at 19:00.
- What is the pv of Jesper [in eur]?

pv(jesper,eur) = 2*pv(beer,eur) – 5*pv(eur,eur) = 2*pv(beer,eur) – 5

- Note that we need not know what beer or pv(beer,eur) or eur actually is.
- These things can be defined as when/if there is need for it.

Trading Strategies

- We can make it a fair trade by use of the pv() keyword

time	event
18:00	jesper rec(1, swap, brian) brian rec(pv(swap,eur), eur, jesper)

- … or make it contingent

time	event
17:59	dig = pv(swap,eur)>1
18:00	jesper rec(dig, swap, brian) brian rec(dig*pv(swap,eur), eur, jesper)

Back to the Asian Option

- Rewritten in new syntax

start	end	freq	event
01Apr20			s0 = pv(sp500,usd)
01Apr20	01Apr25	3m	sum = sum + pv(sp500,usd)/s0 j = j + 1
01Apr25			opt rec(max(0, sum/j - 1.05), eur,)
17May19			jesper rec(1000, opt, brian) brian rec(1000*pv(opt, sek) + 10, sek, jesper)

- In other words: we will drink some beers and trade some options forward.

- The price of the options will be fair modulo a fee and settled in SEK.

What is the Point?

- Jife allows for a detailed and concise record of all transactions (past and future) that occur in a bank.

- Loans for bicycles, mortgage bonds, equity holdings, exotic options, whatever.

- This means only one core system for operational flows in the bank:

  - fixing, payments, pvs, tax, ...

- Application to aggregate risk management (as with 2nd generation) such as

  - Market risk.
  - XVA.

- Funding balances.
- Regulatory capital calculations.

- But also transaction behavior (past, present and projected) to be used for

  - Market making.
  - Price verification.
  - Anti-money laundering.
  - Audit trail.
  - Customer behavior and commercialization of data.
  - “Simulating the bank”.

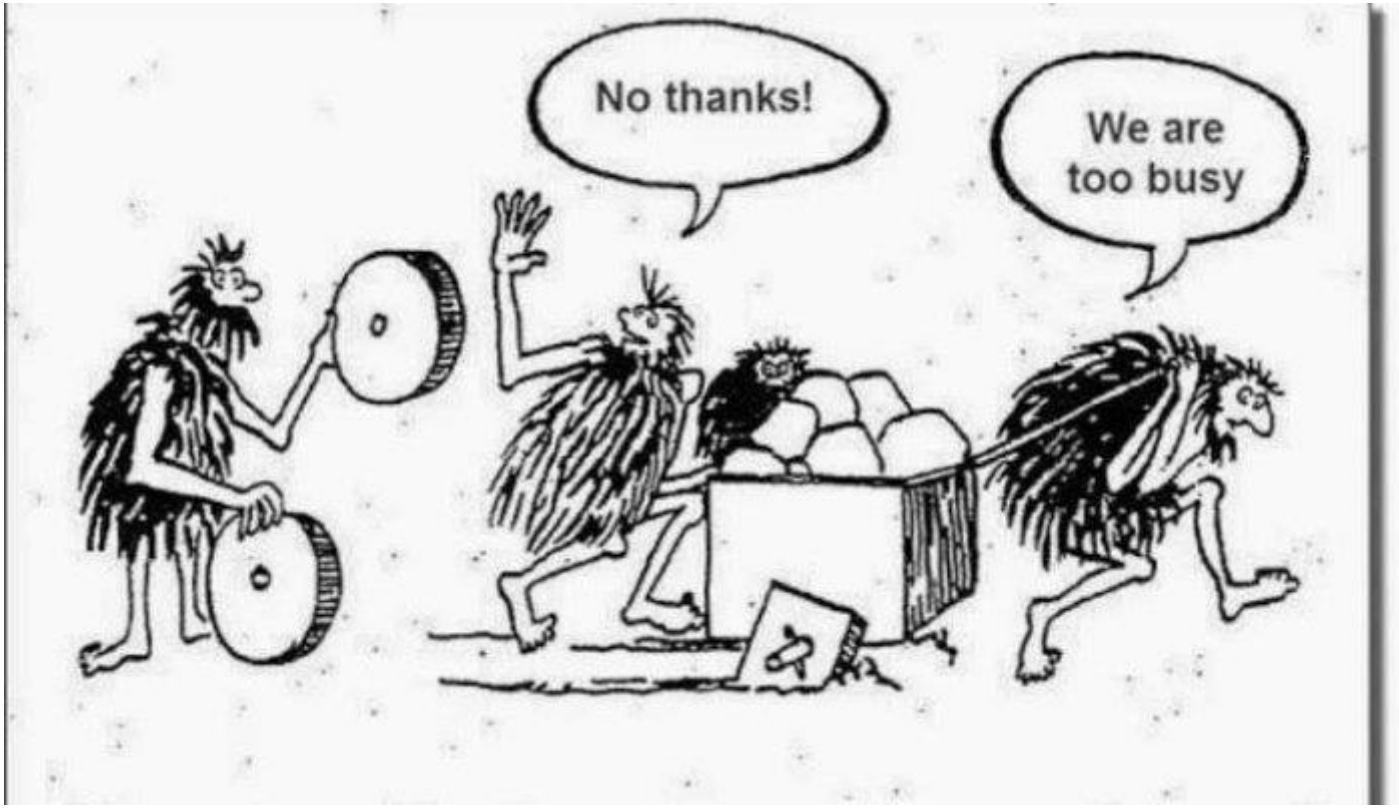
- … and many other things.

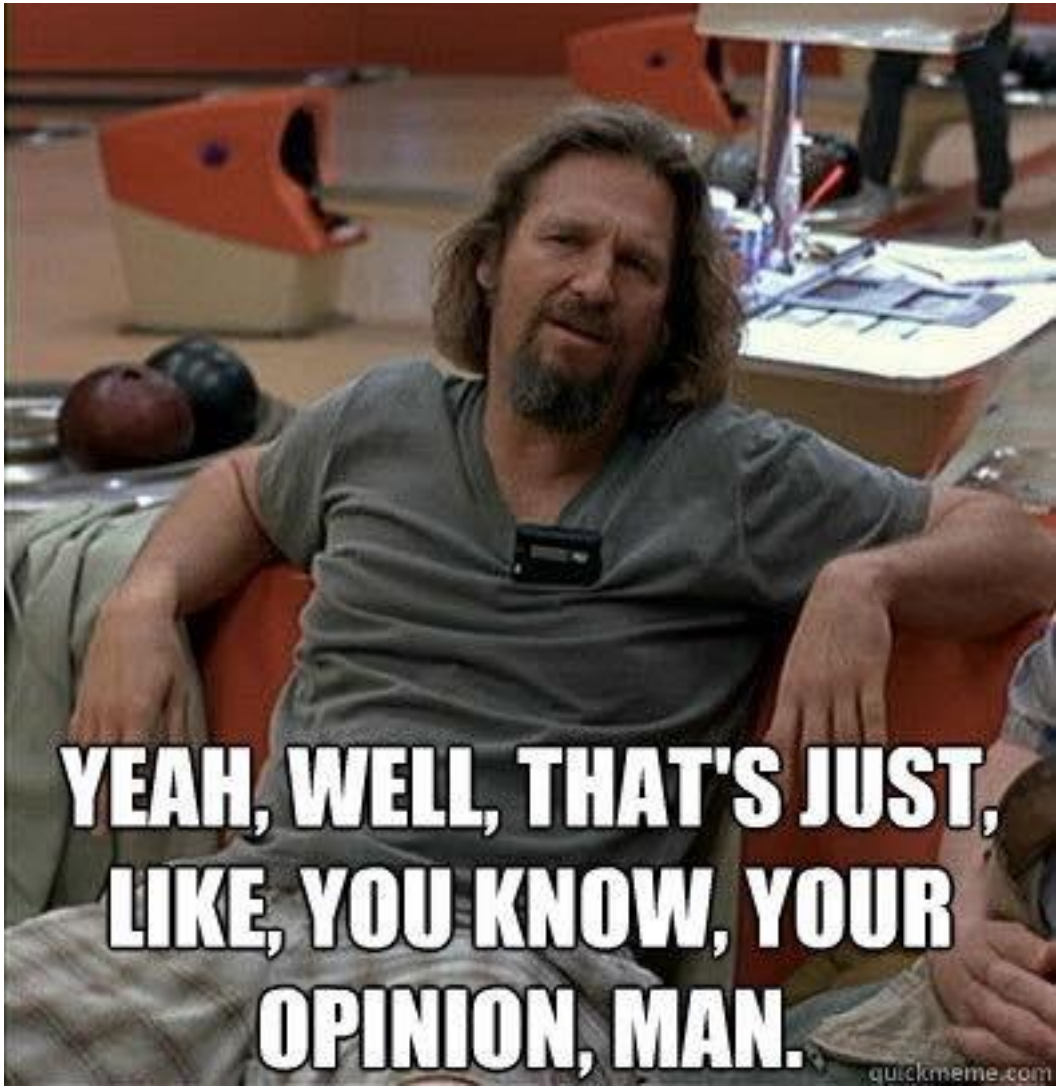
- A lot of structured data to unleash machine learning and AI on.

- The structured approach of the trades, portfolios, and transactions makes it a lot easier to implement it with the latest and greatest in technology

  - Cloud, micro services, blockchain, …

- In other words, Jife can be the back bone of a true digital bank.


No thanks!
We are too busy


YEAH, WELL, THAT'S JUST, LIKE, YOU KNOW, YOUR OPINION, MAN.
quickmeme.com