FinancePy

A Python library for Financial Security Valuation and Risk-Management

PyConf Hyderabad 2020

Dr. Dominic O'Kane

This Version: 5 Dec 2020

Overview

- What is FinancePy ?
  - Quick Introduction to Financial Derivatives
- Design of **FinancePy**
  - Model, Market, Product paradigm
- Case Studies
  - Bond Analysis
  - Option Pricing
  - Performance Test
- Alternatives
- Where to start
- Contributing
- Conclusions

What is FinancePy ?

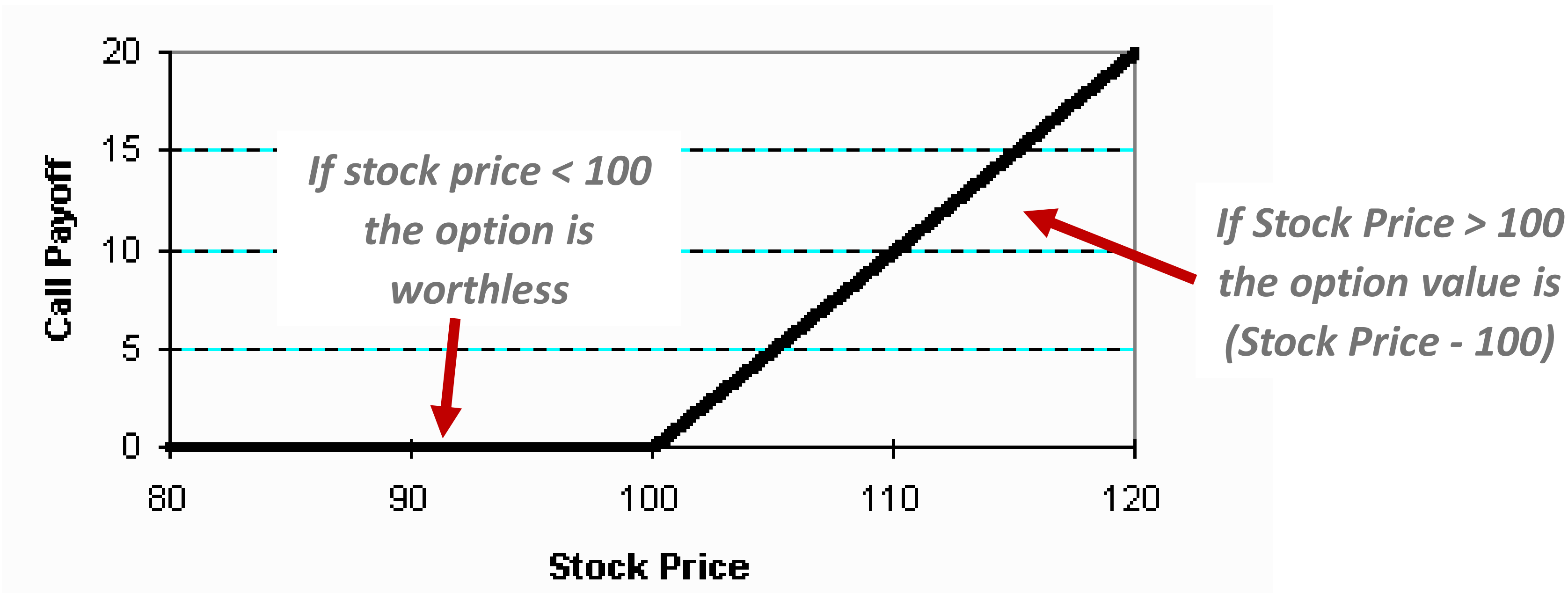
- FinancePy is a Python-based library for the valuation of financial securities, **with a special focus on financial derivatives**
- Find it at https://github.com/domokane/FinancePy
- Developed by me – finance academic with industrial background
- Contributions from Fergal O'Kane and Gurram Poorna Prudhvi
- Handles a broad range of asset classes including:
  - bonds
  - equities
  - currencies
  - interest rates
  - inflation
- And derivatives on all of these

What are Derivatives ?

- Derivatives are financial contracts with payoffs linked to the price of market assets, e.g., stocks prices, interest rates, FX rates
- They are called "derivatives" because their value is **derived** from the price of the underlying asset
- The financial derivatives market is huge – more than $700 trillion
- Derivatives enable …
  - Banks to protect themselves against changes in funding rates
  - Companies to protect themselves against changes in FX rates
  - Farmers to fix the price of their harvest in advance
  - Start-ups to pay a lower salary but give employees upside!
  - Investors to take a specific risk-return profiles
- The simplest (non-trivial) derivative is the Call Option…

What is a Call Option ?

- Investor wants exposure to a stock for a year but does not want to lose too much money if the stock price falls
- The investor buys a call option - they have the **option** to buy the stock in one year at a **strike** price equal to today's price of 100



- No downside - the investor has to pay something for this option!
- FinancePy can determine the option price.

FinancePy Design

FinancePy Design

- Finutils
  - Basic functionality used across the library
- Market
  - Holders, processors of market data
- Models
  - Quantitative valuation model library
- Products
  - Financial securities including derivatives as Python classes

FinUtils

- **There are a lot of market conventions used in finance**
- We ensure these are followed as exactly as possible in FinancePy
- FinDate
  - In finance, dates are key to determining valuation
  - There are certain key dates (CDS, IMM dates)
- FinCalendar
  - Need to know all holiday dates in NY, Europe, London, …
- FinSchedule
  - Need to calculate series of cashflow payment dates in accordance with market conventions

Products

Bonds

- FinBond
- FinBondAnnuity
- FinBondConvertible
- FinBondEmbeddedOption
- FinBondFRN
- FinBondFuture
- FinBondMortgage
- FinBondOption

Credit

- FinCDS
- FinCDSBasket
- FinCDSCurve
- FinCDSIndexOption
- FinCDSIndexPortfolio
- FinCDSOption
- FinCDSTranche

Funding

- FinFixedLeg
- FinFloatLeg
- FinIborBasisSwap
- FinIborCallableSwap
- FinIborDeposit
- FinIborFuture
- FinIborFRA
- FinIborSwap
- FinIborCapFloor
- FinIborSwaption
- FinIborSingleCurve
- FinIborDualCurve
- FinIborOIS
- FinOIS
- FinOISCurve
- FinIborBermudanSwaption

Equity

- FinEquityAmericanOption
- FinEquityAsianOption
- FinEquityBarrierOption
- FinEquityBasketOption
- FinEquityChooserOption
- FinEquityCliquetOption
- FinEquityCompoundOption
- FinEquityDigitalOption
- FinEquityFixedLookbackOption
- FinEquityFloatLookbackOption
- FinEquityRainbowOption
- FinEquityOneTouchOption
- FinEquityVanillaOption
- FinEquityVarianceSwap

FX

- FinFXForward
- FinFXVanillaOption
- FinFXBarrierOption
- FinFXBasketOption
- FinFXRainbowOption
- FinFXDigitalOption
- FinFXFixedLookbackOption
- FinFXFloatLookbackOption
- FinFXVarianceSwap

Inflation

- FinInflationBond
- FinInflationSwap

Commodities

- FinSwingContract (TBC)

- Each of these is a Python class under **Products**

Market

- Discounting future cashflows correctly is essential

Discount Curves

- FinDiscountCurve
- FinDiscountCurveFlat
- FinDiscountCurveNS
- FinDiscountCurveNSS
- FinDiscountCurvePoly
- FinDiscountCurvePWF
- FinDiscountCurvePWL
- FinDiscountCurveZeros

- Managing the volatility assumptions for options is key

Volatility

- FinEquityVolCurve
- FinFXVolSurface
- FinIborCapVolCurve
- FinIborCapVolCurveFn

Models

- Models are not product-specific

Lognormal

- FinGBMProcess
- FinModelBlack
- FinModelBlackScholes
- FinModelBlackScholesAnalytical
- FinModelBlackScholesShifted
- FinModelCRRTree

Credit

- FinModelGaussianCopula
- FinModelLossDbnBuilder
- FinModelLHPlus
- FinModelMertonCredit
- FinModelMertonCreditMkt

Rates

- FinModelRatesBDT
- FinModelRatesBK
- FinModelRatesCIR
- FinModelRatesHL
- FinModelRatesLMM

Normal

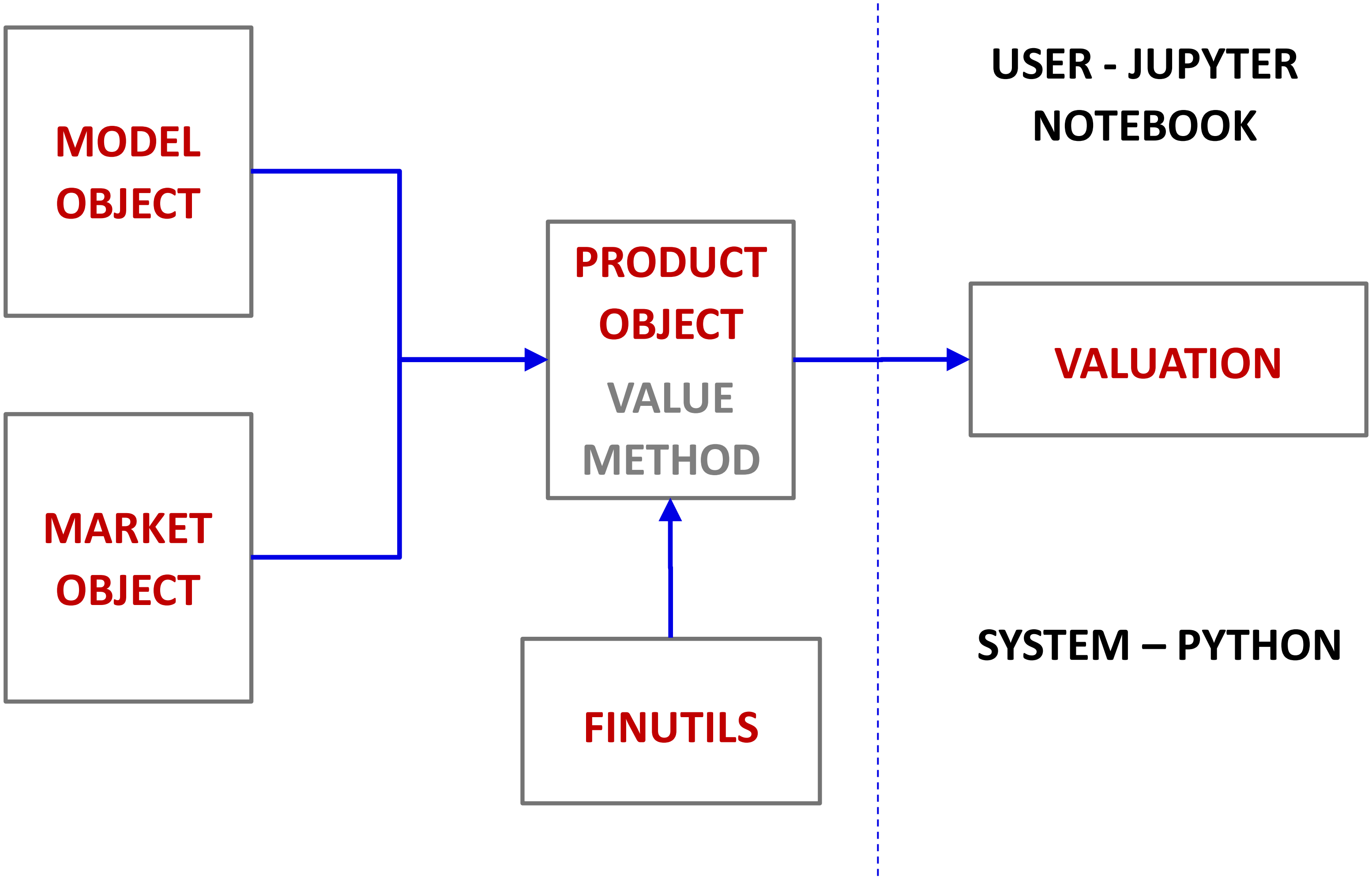
- FinModelBachelier
- FinModelRatesVasicek

Stochastic Vol

- FinModelHeston
- FinModelSABR

Design

- Stage I: Create a Product Object e.g. a call option
- Stage II: Value the product by passing in a model and market



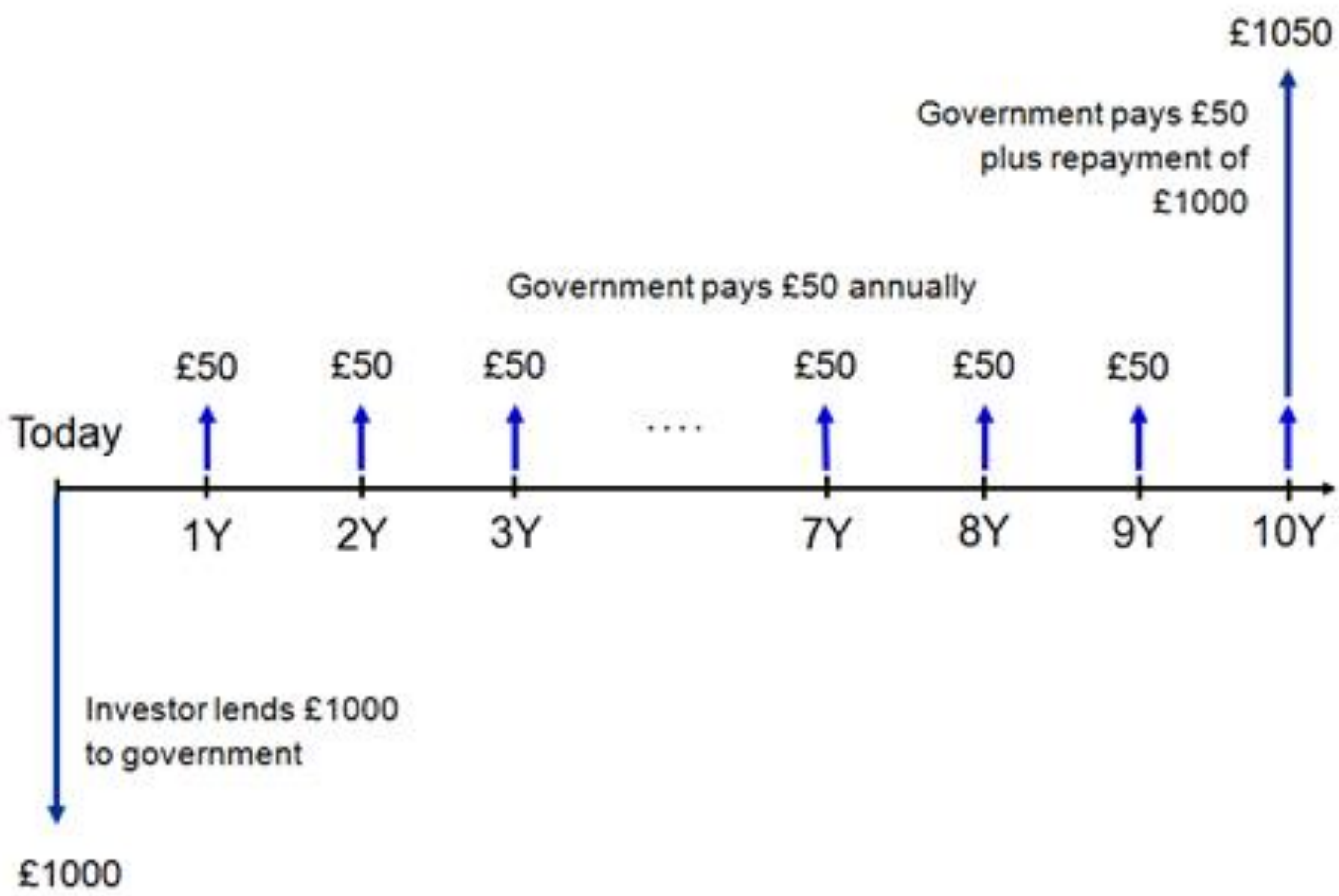
Highlights

- Bond Yield curve fitting with multiple parametric forms
- Building of IBOR discount curve
- Two curve-based pricing using OIS and associated derivatives
- Single factor Trinomial Trees for interest rate option pricing
- Multi-factor Libor Market Model
- Convertible bond pricing model
- Valuation of Synthetic CDO tranches
- Full calibration to FX volatility surface
- Multi-process simulator with stochastic volatility
- Variance reduction methods for path dependent options
- and lots more …

Case Study:
Analysing a Bond

A Government Bond

- Bonds are how governments borrow
- Example of a 10 year government bond with a 5% coupon



Starting the Jupyter Notebook

- Open the notebook “Case Study – Analysing a Bond”

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from financepy.finutils import *
from financepy.products.bonds.FinBond import *
```

```
####################################################################
# FINANCEPY BETA Version 0.186 - This build:  01 Dec 2020 at 13:21 #
#      This software is distributed FREE & WITHOUT ANY WARRANTY    #
# For info and disclaimer - https://github.com/domokane/FinancePy  #
#      Send any bug reports or comments to quant@financepy.com     #
####################################################################
```

- I always load **Numpy** and **Matplotlib** for processing and plotting
- I import all the **FinUtils** modules – dates and conventions
- I only import the **FinBond** product – loading all the products is slow and unnecessary

Creating a FinBond

- We need to use FinDate and some Enum types

```
issueDate = FinDate(15, 5, 2020)
```

```
maturityDate = FinDate(15, 5, 2030)
```

```
coupon = 0.050 # This means 5%
```

```
freqType = FinFrequencyTypes.ANNUAL
```

```
accrualType = FinDayCountTypes.ACT_ACT_ICMA
```

```
faceAmount = ONE_MILLION
```

```
bond = FinBond(issueDate, maturityDate, coupon, freqType, accrualType, faceAmount)
```

Use enums for categorical values

We create the FinBond object

Examining a Product: A Bond

- Every object is printable

Object type

```
print(bond)
```

```
OBJECT TYPE: FinBond
ISSUE DATE: 15-MAY-2010
MATURITY DATE: 15-MAY-2030
COUPON: 0.05
FREQUENCY: FinFrequencyTypes.ANNUAL
ACCRUAL TYPE: FinDayCountTypes.ACT_ACT_ICMA
FACE AMOUNT: 1000000
```

- This helps us when checking and can be used for reporting

Documentation

- If you need help, type **help(FinBond) …**
- Or there is a 339-page auto-generated user guide in the project!

Contents

7.1 FinBond

Enumerated Type: FinYTMCalcType

This enumerated type has the following values:
- UK_DMO
- US_STREET
- US_TREASURY

Class: FinBond(object)

Class for fixed coupon bonds and performing related analytics. These are bullet bonds which means they have regular coupon payments of a known size that are paid on known dates plus a payment of par at maturity.

FinBond

Create FinBond object by providing the issue date, maturity Date, coupon frequency, annualised coupon, the accrual convention type, face amount and the number of ex-dividend days.

```
FinBond(issueDate: FinDate,
        maturityDate: FinDate,
        coupon: float,  # Annualised bond coupon
        freqType: FinFrequencyTypes,
        accrualType: FinDayCountTypes,
        faceAmount: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issueDate	FinDate	-	-
maturityDate	FinDate	-	-
coupon	float	Annualised bond coupon	-
freqType	FinFrequencyTypes	-	-
accrualType	FinDayCountTypes	-	-
faceAmount	float	-	100.0

fullPriceFromYTM

Calculate the full price of bond from its yield to maturity. This function is vectorised with respect to the yield input. It implements a number of standard conventions for calculating the YTM.

```
fullPriceFromYTM(settlementDate: FinDate,
                 ytm: float,
                 convention: FinYTMCalcType = FinYTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Type Checking Inputs

- Want to protect users from accidentally typing in wrong input
- We do type-checking of inputs using the **type hints** feature

```python
class FinBond(object):
    ''' Class for fixed coupon bonds and performing related analytics. These
    are bullet bonds which means they have regular coupon payments of a known
    size that are paid on known dates plus a payment of par at maturity. '''

    def __init__(self,
                 issueDate: FinDate,
                 maturityDate: FinDate,
                 coupon: float,  # Annualised bond coupon
                 freqType: FinFrequencyTypes,
                 accrualType: FinDayCountTypes,
                 faceAmount: float = 100.0):
        ''' Create FinBond object by providing the issue date, maturity Date,
        coupon frequency, annualised coupon, the accrual convention type, face
        amount and the number of ex-dividend days. '''

        checkArgumentTypes(self.__init__, locals())
```

TYPE CHECKING

- Uses a function that throws an exception if there is a type error
- Only do this for product classes and other user-facing functions

Example: Display of Cash Flows

- We can obtain the list of cash flows as of a given settlement date

```
settlementDate = FinDate(6, 12, 2020)
```

```
bond.printFlows(settlementDate)
```

```
15-MAY-2021      50000.00
15-MAY-2022      50000.00
15-MAY-2023      50000.00
15-MAY-2024      50000.00
15-MAY-2025      50000.00
15-MAY-2026      50000.00
15-MAY-2027      50000.00
15-MAY-2028      50000.00
15-MAY-2029      50000.00
15-MAY-2030    1050000.00
```

5% paid on $1m means $50k per payment

At maturity you get $50k plus $1m

Example: The Yield To Maturity

- An important metric is the yield (YTM) - calculated from the price
- It needs to be **exactly right** as it is used to trade on
- Can calculate the YTM according to different conventions

```
cleanPrice = 102.20
```

Yield to maturity using different conventions

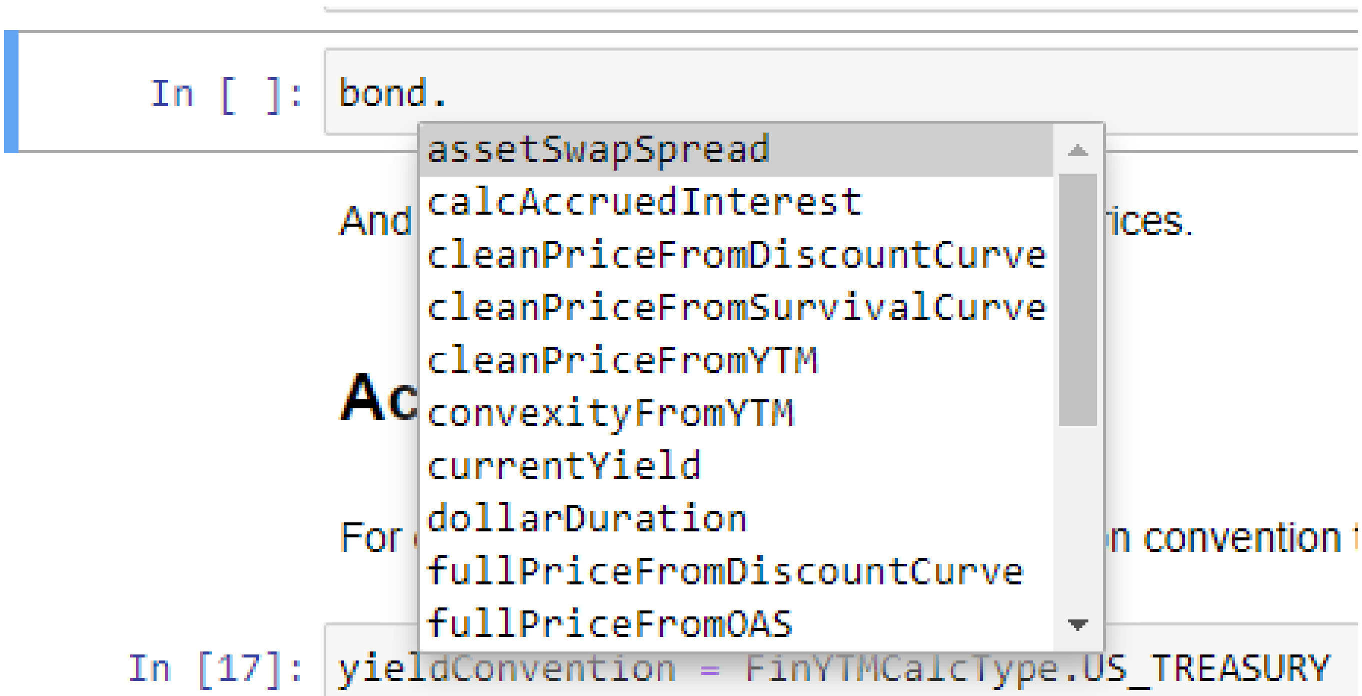
```
for ytmCalcType in FinYTMCalcType:
    y = bond.yieldToMaturity(settlementDate, cleanPrice, ytmCalcType)
    print("%30s %12.7f" % (ytmCalcType, y*100))
```

```
       FinYTMCalcType.UK_DMO    4.7022358
    FinYTMCalcType.US_STREET    4.7022358
  FinYTMCalcType.US_TREASURY    4.6986405
```

- This is a complex calculation that involves solving a 1D equation
- But the answer is instant – we use a Numpy function to solve

Comprehensive set of Bond Functions

- There are lots of bond-specific functions
- These are all revealed using the tab key



- Good way for the user to see the available functionality

Case Study:
Valuing an Option

What is an Equity Vanilla Option ?

- Notebook – "Case study Equity Vanilla Option.ipynb"
- As we saw earlier, an option is described by:
  - The expiry date
  - The strike price
  - The type of option (call or put)
- It's called a "**Vanilla**" option as it's the most common type of option and to distinguish it from other types of option
- Here I just consider an options on equities (company stocks)
- So FinEquityVanillaOption is the product name.

Creating a FinEquityVanillaOption

- We need to specify the expiry date and the strike price

```
expiryDate = FinDate(1, 6, 2021)
```

```
strikePrice = 100.0
```

We now create the option object

```
callOption = FinEquityVanillaOption(expiryDate, strikePrice, FinOptionTypes.EUROPEAN_CALL)
```

```
print(callOption)
```

```
OBJECT TYPE: FinEquityVanillaOption
EXPIRY DATE: 01-JUN-2021
STRIKE PRICE: 100.0
OPTION TYPE: FinOptionTypes.EUROPEAN_CALL
NUMBER: 1.0
```

Enum for option type

- The "number" of underlying shares is a default argument
- If we don't supply it, it equals 1.0

How do we value it ?

- We use a model called “Black-Scholes”
- This is widely accepted as the market standard
- I won’t try to explain why or derive it here !
- The Black-Scholes option valuation model needs to know:
  - The number of years to the expiry date
  - The stock price today
  - Interest rate
  - Dividend rate
  - Volatility of the stock price

Valuation of an Option

- The valuation inputs are as follows

```
valueDate = FinDate(6, 12, 2020)
```

```
stockPrice = 90.0
```

```
dividendYield = 0.01
```

- Interest rates are very important. I have created several ways of representing the structure of interest rates.
- These are known as **FinDiscountCurves** objects
- The simplest is to assume that interest rates curves are flat
- I call this the **FinDiscountCurveFlat** object !

```
interestRate = 0.02
```

```
discountCurve = FinDiscountCurveFlat(valueDate, interestRate, FinFrequencyTypes.ANNUAL)
```

Valuation of an Option … continued

- The final input is a model – in BS this is just a volatility parameter

```
volatility = 0.20
model = FinModelBlackScholes(volatility)
```

- The valuation takes in all the inputs including the model

```
callOption.value(valueDate, stockPrice, discountCurve, dividendYield, model)
```

```
1.801680685204456
```

- The option costs $1.80
- Even though it is the market standard, Black-Scholes is not the only model that can be used to value an option
- **Making the model an object allows us to switch models easily**

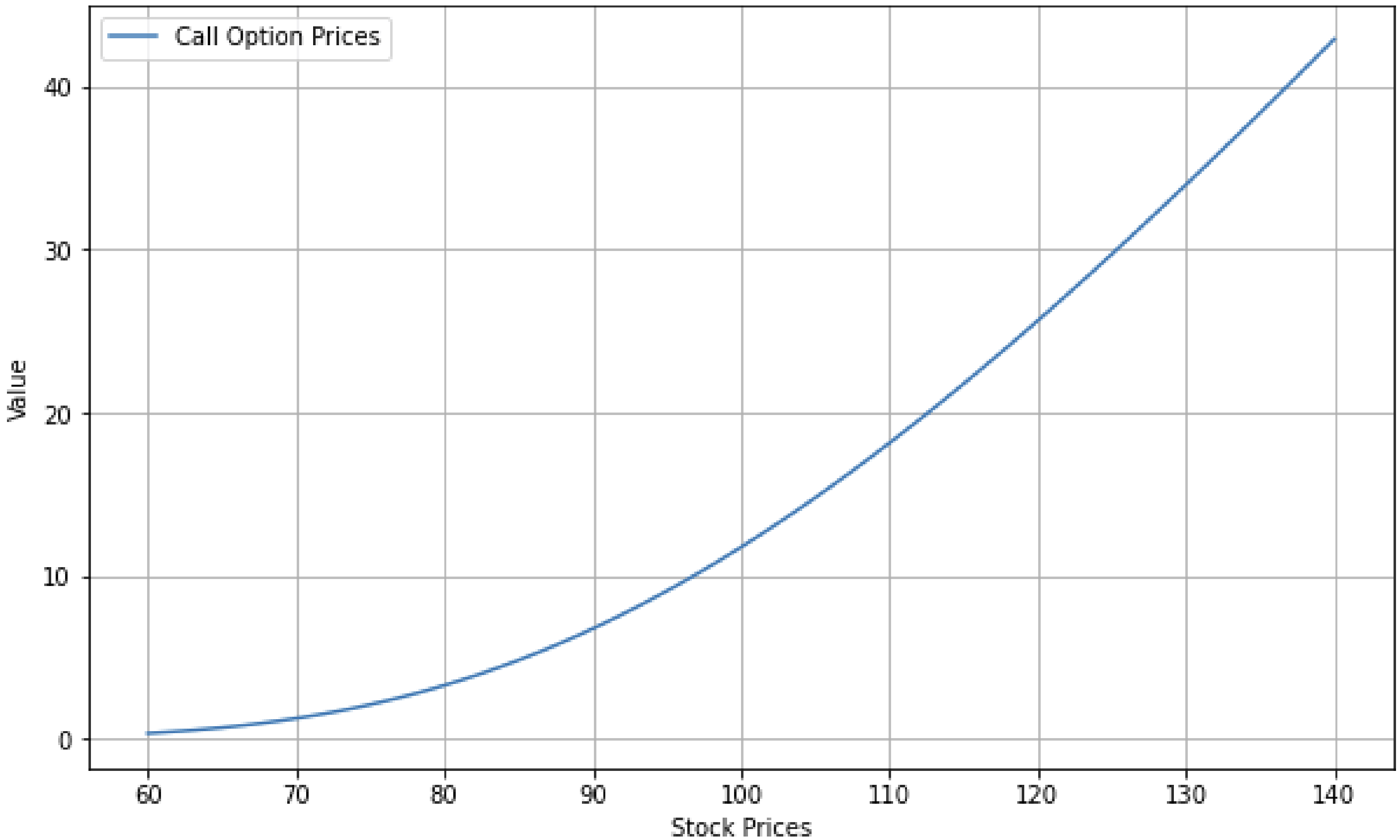
Vectorisation: Makes Analysis Easier

- There are lots of inputs to an option valuation
- Varying an input usually means writing a loop
- FinancePy avoids this pain by doing vectorization
- Some of this is automatic thanks to Numpy
- Some of it is hand-written internally
- In the notebook you will see a lot of examples

Vectorisation Example

```
stockPrices = np.linspace(60,140,100)
values = callOption.value(valueDate, stockPrices, discountCurve, dividendYield, model)
```

```
plt.figure(figsize=(10,6))
plt.plot(stockPrices,values, label="Call Option Prices")
plt.xlabel("Stock Prices")
plt.ylabel("Value")
plt.legend()
plt.grid()
```



Case Study:
MC Optimization using Numpy and Numba

Numpy and Numba

- Numpy
  - Numpy is an open-source numerical library
  - Uses compiled code to perform complex calculations quickly
  - Vectorised calculations are faster than Python
  - Need to know how to vectorise calculations
- Numba
  - Numba is an open-source JIT compiler
  - It uses the LLVM compiler library and it means that speeds can approach that of C and Fortran
  - It also facilitates parallel processing
  - Just add a decorator to the Python function
- Hat-tip to Yves Hilpisch whose excellent book **Python for Finance** first alerted me to the power of Numba

Monte Carlo Option Pricing

- Some options have complex payoffs and cannot be priced using a closed-form equation so instead we have to use "Monte Carlo"

  1. Simulate many thousands of "paths" - future stock prices drawn from the correct distribution
  2. Determine the payoff of the option on the expiry date
  3. Average over the payoffs
  4. Discount the average back to today to get the price

- The price gets more accurate with more paths – ***we want as many paths as possible!***
- Is Python fast enough to compete with C++ quant libraries ?

Pure Python

- We draw random numbers using a Numpy function
- We then calculate the final stock price and the option payoff
- Finally, we average and discount the payoff

```
def valueMC1(s0, t, K, r, q, v, numPaths, seed):

    vsqrtt = v * sqrt(t)
    ss = s0 * exp((r - q - v*v / 2.0) * t)

    np.random.seed(seed)
    g = np.random.standard_normal(numPaths)

    payoff = 0.0
    for i in range(0, numPaths):
        s = ss * exp(+g[i] * vsqrtt)
        payoff += max(s - K, 0.0)

    v = payoff * np.exp(-r * t) / numPaths
    return v
```

Generate random numbers

Loop over paths

Average payoff and discount it

Using Numpy

- We can vectorise several stages of the algorithm

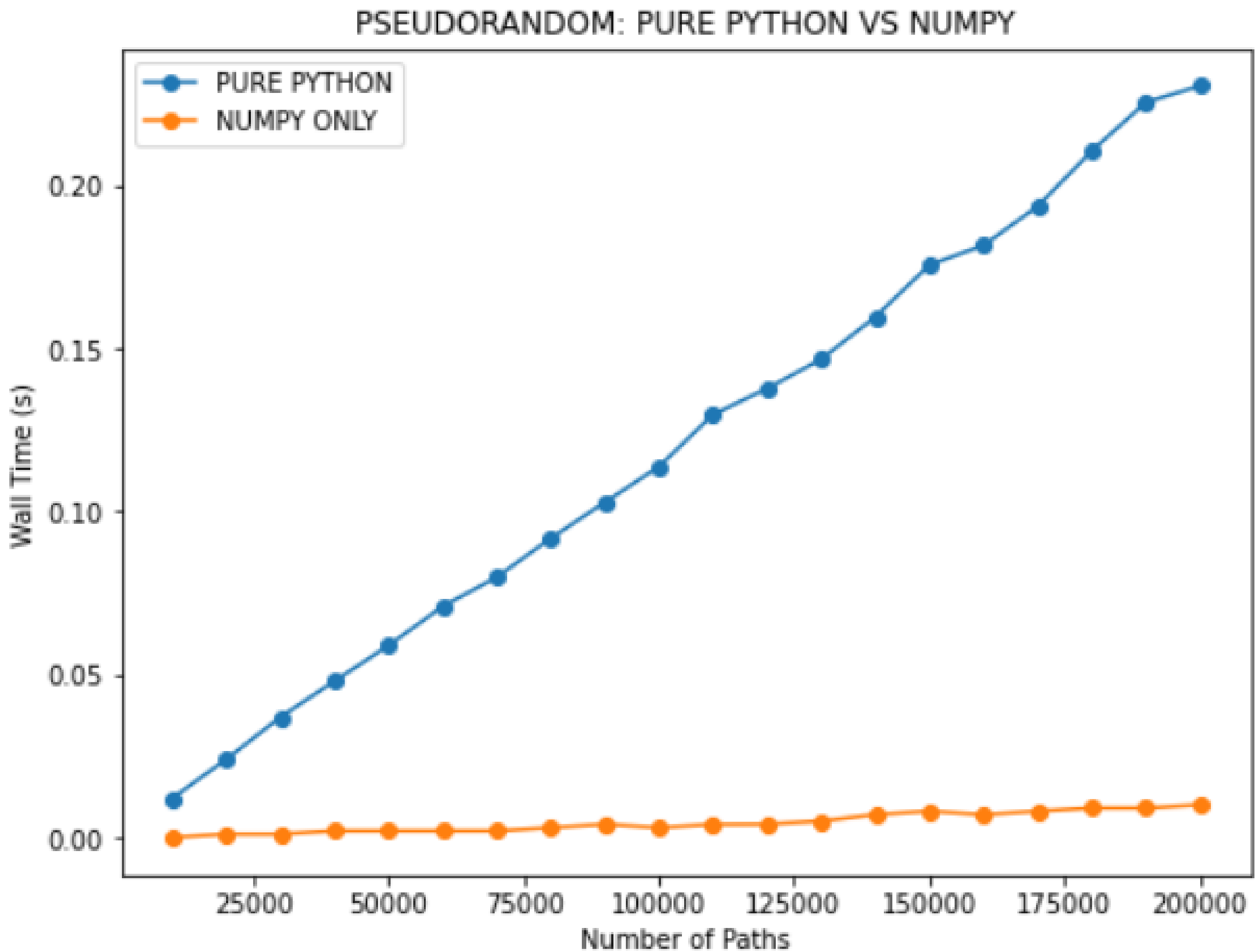
```
def valueMC2(s0, t, K, r, q, v, numPaths, seed):

    np.random.seed(seed)
    g = np.random.standard_normal(numPaths)
    vsqrtt = v * np.sqrt(t)
    s = s0 * exp((r - q - v*v / 2.0) * t)

    s = s * np.exp(g * vsqrtt)
    payoff = np.maximum(s - K, 0.0)
    averagePayoff = np.mean(payoff)

    v = averagePayoff * np.exp(-r * t)
    return v
```

Generation of vector of stock prices, payoffs and averaging all done using vectorisation

- Sorry for not using snake_case (too much C++) 😉

Numpy vs Pure Python

- The impact is huge – a speed increase of about x 25



How to use Numba

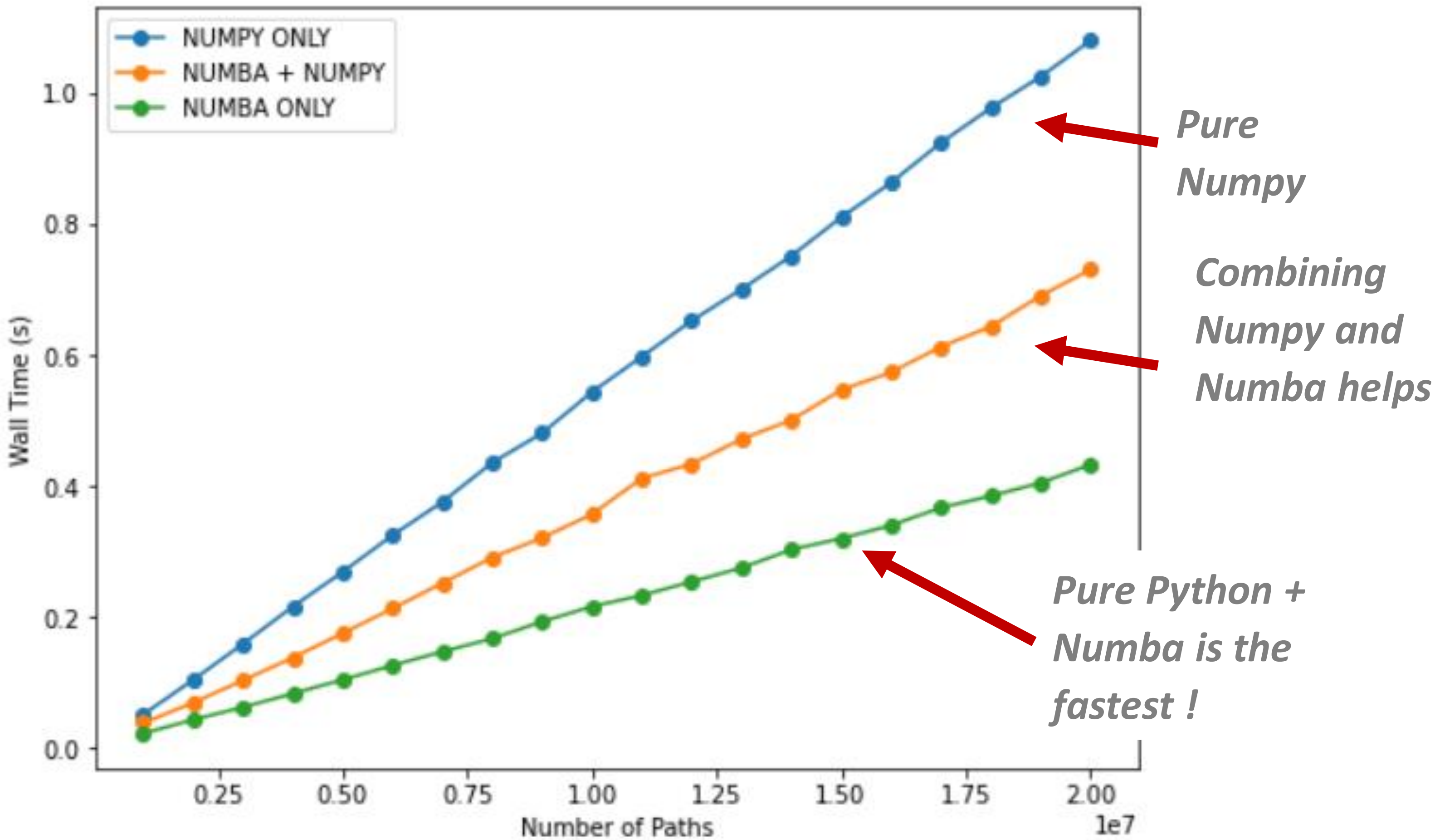
- We import Numba and add a decorator to the pure Python

```
@njit(float64(float64, float64, float64, float64, float64, float64,
              int64, int64), cache=True, fastmath=True)
def valueMC3(s0, t, K, r, q, v, numPaths, seed):

    vsqrtt = v * sqrt(t)
    ss = s0 * exp((r - q - v*v / 2.0) * t)

    np.random.seed(seed)
    g = np.random.standard_normal(numPaths)

    payoff = 0.0
    for i in range(0, numPaths):
        s = ss * exp(+g[i] * vsqrtt)
        payoff += max(s - K, 0.0)

    v = payoff * np.exp(-r * t) / numPaths
    return v
```

Decorator that shows function signature, caches the compiled code and uses lower precision for faster math

- I **also** added a Numba decorator to the Numpy function.

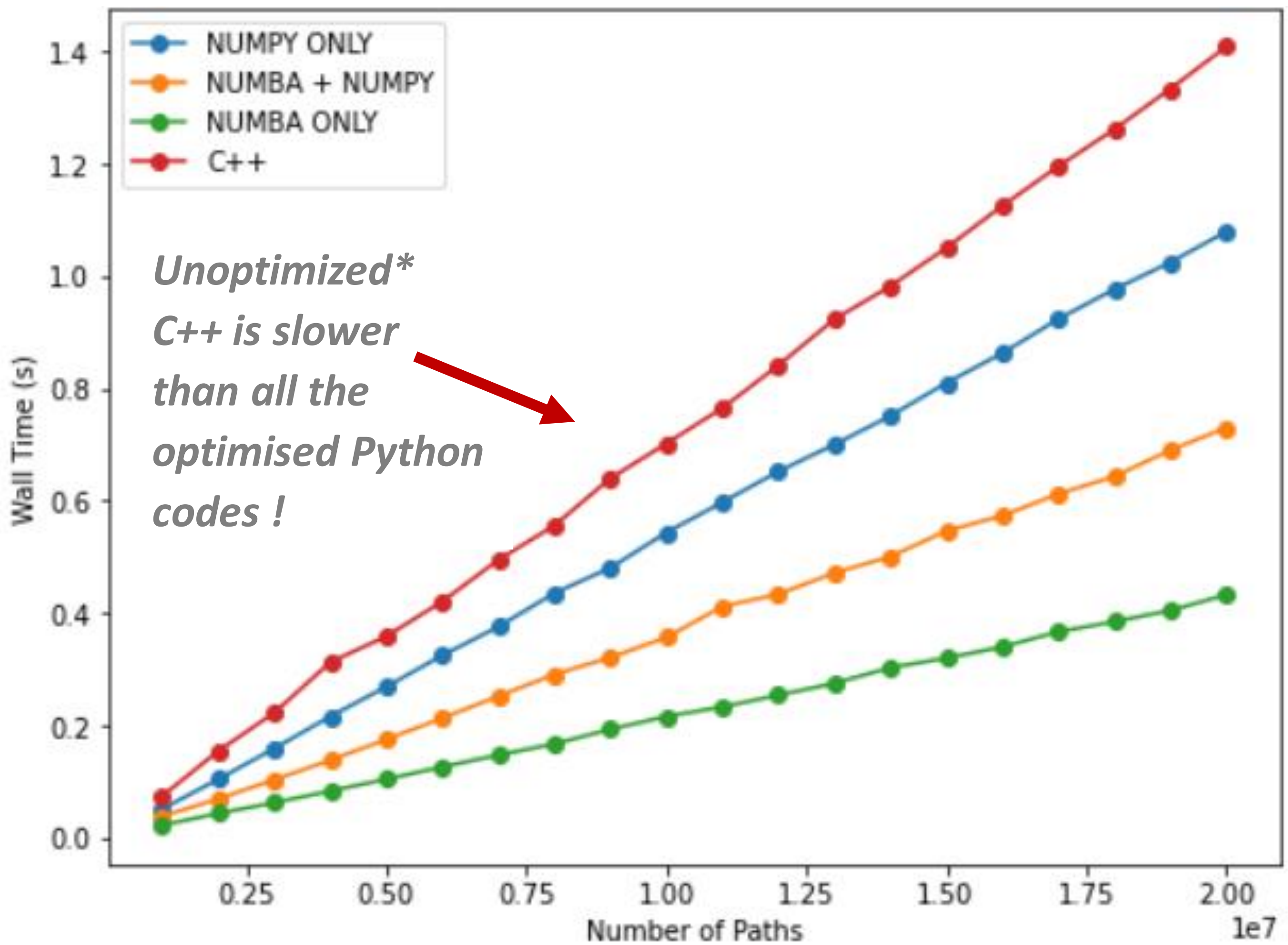
Timings – From 2 to 20M paths



- Pure Python + Numba is the fastest of the three
- **Note that all 3 give identical results – same random sequence**

Comparison against Unoptimized C++

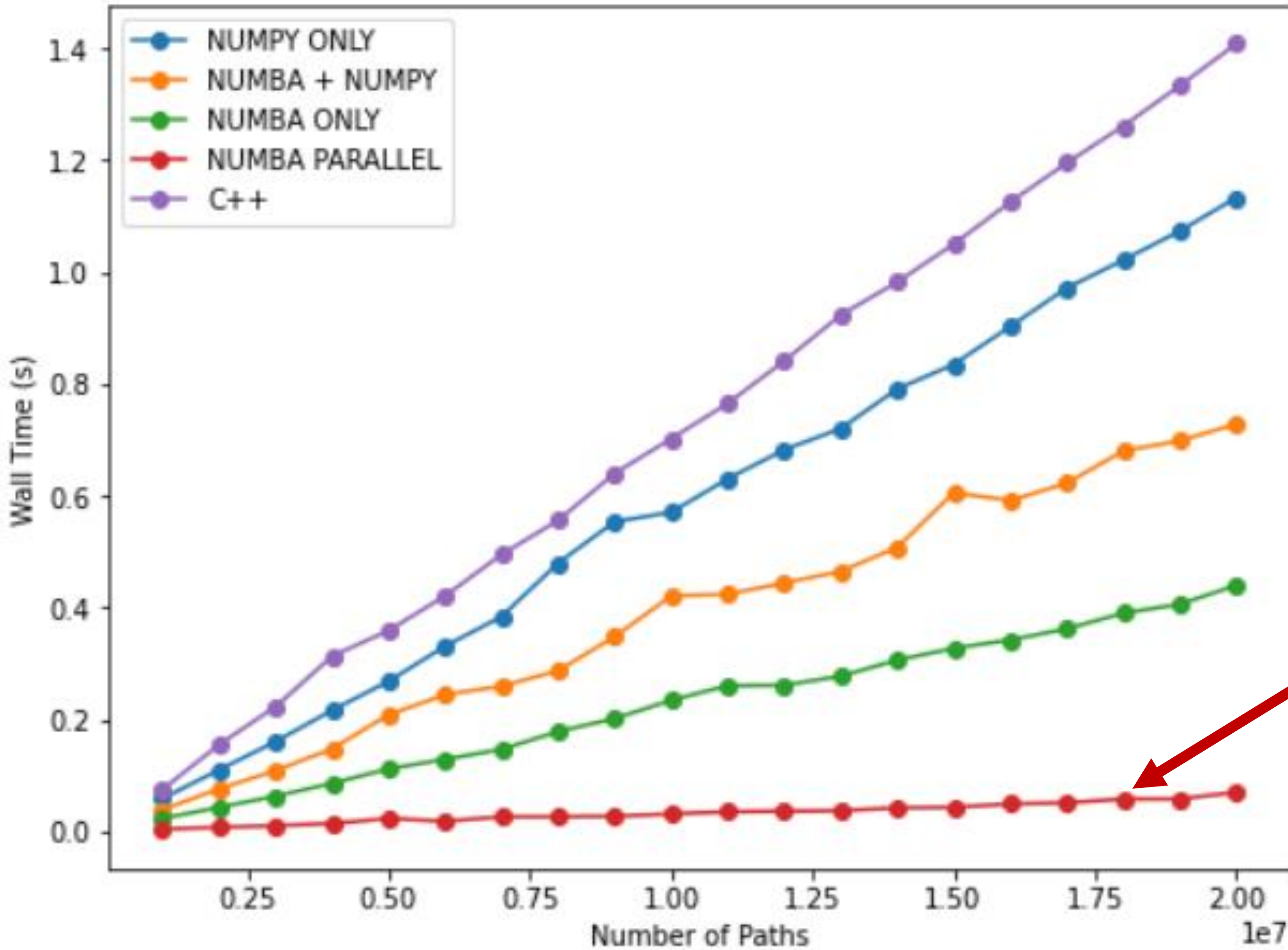
- I did a comparison against C++ (release, O2 optimized)



* Random number generator from Numerical Recipes has not been optimized.

Last Optimisation - Parallel

- I set decorator flag **parallel=True** and change **range** to **prange**



Parallel is 6-7x faster !

- Numba Parallel is 6-7 times faster than Numba (I have 8 cores)

Final Remarks

Target Audience

- Students and Professors
  - A way to teach/learn about finance and python
- Traders
  - A tool for checking a price
- Quantitative analyst
  - Analyzing a derivative to see if the price is fair
- Risk managers
  - Sensitivity analysis of derivative price
- Investor
  - Scenario testing an investment strategy
- Finance academic or researcher
  - Designing a new model or pricing algorithm

Alternative: MATLAB's FIT

- One alternative is MATLAB's Financial Instruments Toolbox (FIT)
- Annual license for MATLAB is $960 and Financial Toolbox is $820
- Very comprehensive – been in development for many years
- Fast for matrix calculations and vectorization
- Good documentation online plus online community for support
- The code is a “black box”
- API is not intuitive IMHO
- Not so easy to integrate into existing systems

Alternative: QUANTLIB and OTHERS

- Free open-source C++ library
- Comprehensive and tested – been in development for 10+ years
- Fast as it’s coded in C++
- C++ can be linked into existing systems
- The code is complex for a newbie to understand
- Too complex to use as a tool for teaching finance
- Need advanced understanding of library to add new code

Others:

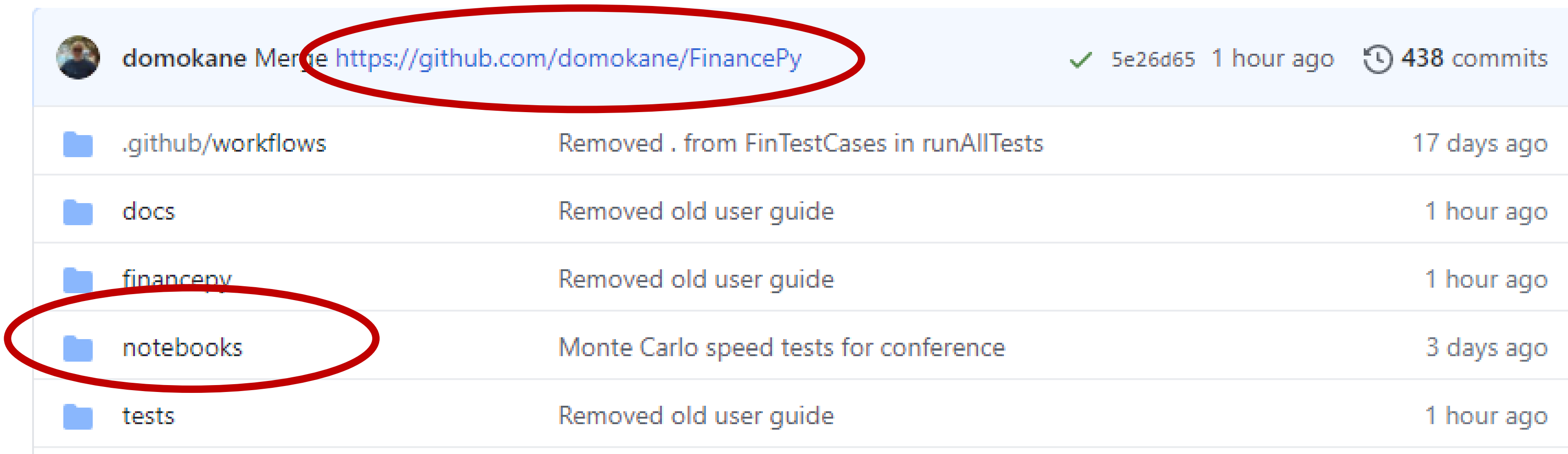
- OpenGamma looks good but it is in Java
- 3rd party vendors – expensive and black boxes

To get started …

- To install FinancePy, use **pip**

```
pip install financepy
```

- Download the repo and look in the notebooks folder



- There are about 70-80 notebooks that you can choose from
- They are organised by product type
- Look forward to your feedback !

Contributors

- Contact me if you wish to contribute at quant@financepy.com
- A few tasks are Python based
- Most require some knowledge of finance + derivative pricing
- You can see a list of issues on the github repository
- Some plans for future work:
  - More products – commodities, MBS, securitized products
  - Models – rates, seasonality (inflation + commodities)
  - Market calibration for complex rate derivatives
  - Payoff Language for structured products
  - Counterparty Value Adjustment valuation
  - Extensive testing!

Conclusions

- Final comments about FinancePy:
  - **Transparent** - Open source and documented ✓
  - **Low Cost** - Free ✓
  - **Comprehensive** - One library for a broad range of products ✓
  - **Responsive** - Code can be changed and released quickly ✓
  - **Design** – Product focused makes it more industry focused ✓
  - **Interface** - Leverage Jupyter Notebook ✓
  - **Fast** - Can compete with C++ ✓
  - **Python** - The only fully Python finance pricing library ✓
- Hope it will gain more users and develop to become a useful toolkit for those needing a finance library
- Thanks for your interest!