Python for computational finance

Alvaro Leitao Rodriguez

TU Delft - CWI

June 24, 2016

1. Why Python for computational finance?

2. QuantLib

3. Pandas

Why Python for computational finance?

- Everything we have already seen so far.
- Flexibility and interoperability.
- Huge python community.
- Widely used in financial institutions.
- Many mature financial libraries available.

QuantLib

- Open-source library.
- It is implemented in C++.
- Object-oriented paradigm.
- Bindings and extensions for many languages: Python, C#, Java, Perl, Ruby, Matlab/Octave, S-PLUS/R, Mathematica, Excel, etc.
- Widely used: d-fine, Quaternion, DZ BANK, Deloitte, Banca IMI, etc.

QuantLib - Advantages

- It is free!! Es gratis!! Het is gratis!!
- Source code available.
- Big community of programmers behind improving the library.
- For researchers (us), benchmark results and performance.
- Common framework.
- Avoid worries about basic implementations.
- Pre-build tools: Black-Scholes, Monte Carlo, PDEs, etc.
- Good starting point for object-oriented concepts.

QuantLib - Disadvantages

- Learning curve.
- Immature official documentation: only available for C++.
- Some inconsistencies between C++ and Python.

QuantLib - Python resources

- QuantLib Python examples.
- QuantLib Python Cookbook (June 2016) by Luigi Ballabio.
- Videoblogs:
  - Introduction to QuantLib (8 parts).
  - The QuantLib notebooks by Luigi Ballabio.
- Blogs:
  - IPython notebooks – a Swiss Army Knife for Quants by Matthias Groncki: https://ipythonquant.wordpress.com/
  - QuantLib Python Tutorials With Examples by Gouthaman Balaraman: http://gouthamanbalaraman.com/blog/quantlib-python-tutorials-with-examples.html
- QuantLib User Meeting (every year).

QuantLib - Modules

- Date and time calculations.
- Financial instruments.
- Stochastic processes.
- Pricing engines.
- Mathematical tools.
- Many others: term structures, indexes, currencies, etc.
- Webpage: http://quantlib.org/

QuantLib - Date

- Constructors:
  - Date(*ndays*). Integer *ndays* is the number of days. $ndays = 0$ corresponds to 31-12-1899.
  - Date(*day*, *month*, *year*). *day* and *year* are integers. *month* is either an integer or enumerate (January, . . . , December).
- Date arithmetic: $+, -, + =, - =$.
- Define a period: Period(*num*, *units*). Number of units, *num*, and $units \in \{Days, Weeks, Months, Years\}$.
- Useful methods: weekday(), dayOfMonth(), dayOfYear(), month(), year().
- Other methods: Date.todaysDate(), minDate(), maxDate(), isLeap(*year*), endOfMonth(*date*), isEndOfMonth(*date*), nextWeekday(*date*, *weekday*), nthWeekday(*n*, *weekday*, *month*, *year*).

QuantLib - Calendar class

- Calendar: holidays, business days and weekends for different countries.
- Many available: UK, Germany, United States, TARGET, etc.
- Also special exchange calendars.
- You can construct your own calendar.
- Useful methods:
  - isBusinessDay(*date*): checks if *date* is a business day.
  - isHoliday(*date*): checks if *date* is a holiday.
  - isEndOfMonth(*date*): checks if *date* is the end of the month.
  - endOfMonth(*date*): returns the last business date in the month.

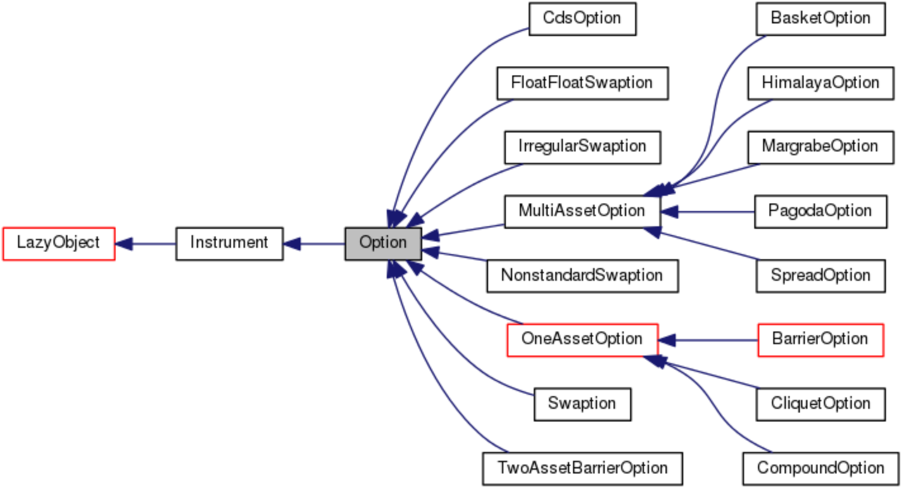
QuantLib - Day count

- Day count conventions: Actual360, Actual365Fixed, ActualActual, Business252, Thirty360, etc.
- Useful methods: dayCount(*date*1, *date*2), yearFraction(*date*1, *date*2).
- Example: `/QuantLib_examples/1-Date.py`

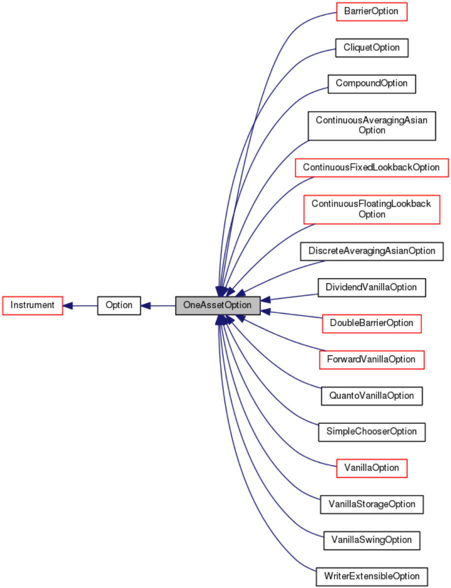
QuantLib - Financial instruments

- Classes defining Options, Bonds, Swaps, Swaptions, etc.
- Useful methods that inherit all the subclasses:
  - NPV(): returns the net present value of the instrument.
  - errorEstimate(): returns the error estimate on the NPV when available.
  - setPricingEngine (*pricingEngine*): set the pricing engine to be used.
  - isExpired(): bool if the option is expired.
- For Options, two main classes: OneAssetOption and MultiAssetOption.

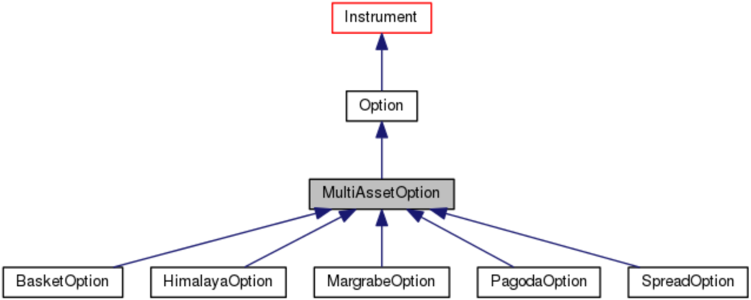
QuantLib - Option class



QuantLib - OneAssetOption class



QuantLib - MultiAssetOption class

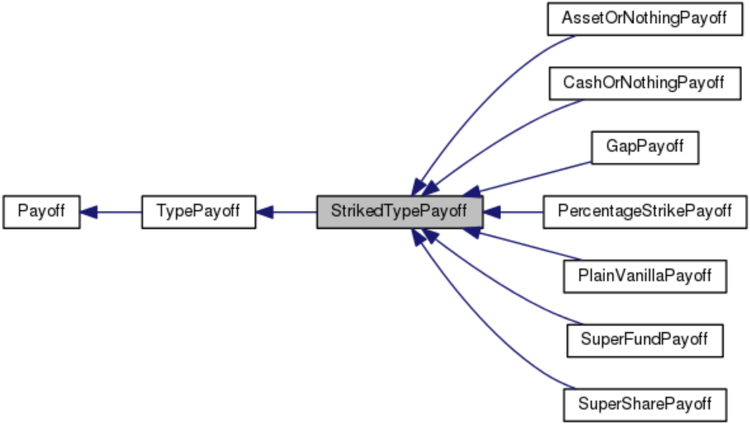


QuantLib - Option class

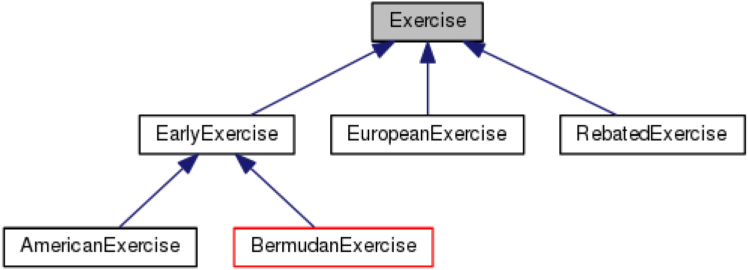
- Constructor: Option(*payoff*, *exercise*).
- Enumerated type: $\{Call = 1, Put = -1\}$.
- For any option, we need to define the payoff and the exercise type.
- Classes Payoff and Exercise.

QuantLib - Payoff class

- Focus on *Striked* type payoffs.



QuantLib - Exercise class



QuantLib - Exercise class

- Enumerated type: $\{American = 0, Bermudan = 1, European = 2\}$
- Most useful classes:
  - EuropeanExercise(*date*): the maturity date is provided.
  - AmericanExercise(*inicialDate*, *finalDate*, *payoffAtExpiry*): the last argument is a boolean indicating if the payment is done immediately of at maturity.
  - BermudanExercise(*dates*, *payoffAtExpiry*) : *dates* is a vector of Date objects.
- Example: `/QuantLib_examples/2-Financial_instruments.py`

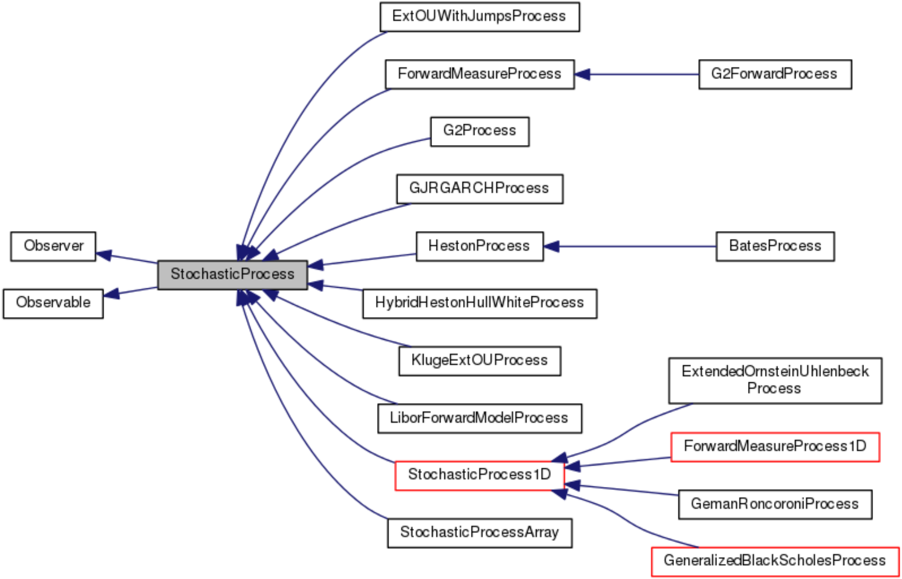
QuantLib - Stochastic processes

- *StochasticProcess* class models a d-dimensional Ito process:

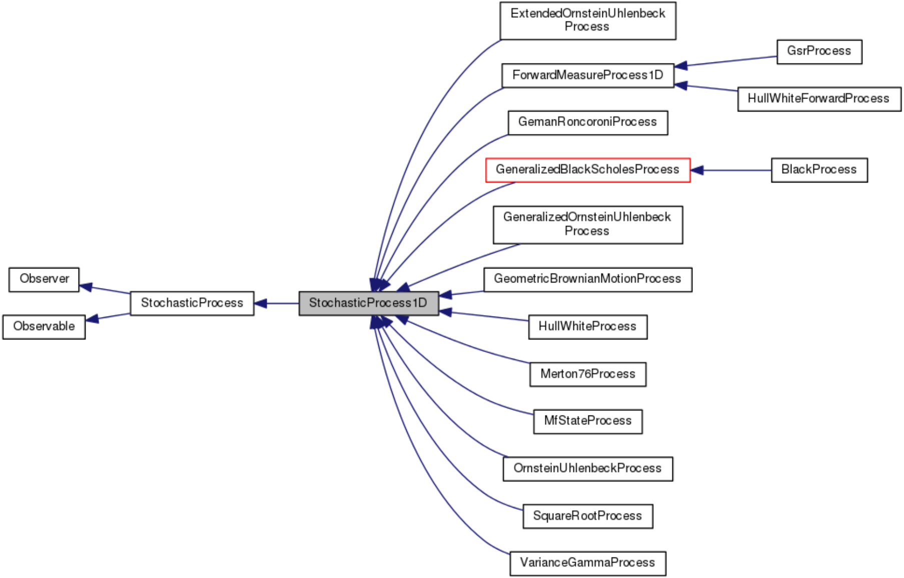
$$\mathrm{d}S_t = \mu(t, S_t)\mathrm{d}t + \sigma(t, S_t)\mathrm{d}W_t$$

- It has a public class *discretization* to model the discretized version.
- Useful methods that all the subclasses inherit:
  - size(): returns the number of dimensions of the stochastic process.
  - initialValues(): returns the initial values of the state variables.
  - drift(t, S_t): returns the drift part of the equation, i.e., $\mu(t, S_t)$.
  - diffusion(t, S_t): returns the drift part of the equation, i.e., $\sigma(t, S_t)$.
  - expectation($t0$, S_0, Δt): returns the expectation (discrete process).
  - stdDeviation($t0$, S_0, Δt): returns the standard deviation (discrete process).
  - covariance($t0$, S_0, Δt): returns covariance (discrete process).
  - evolve($t0$, S_0, Δt, ΔW): returns $\mathbb{E}[S_{t0+\Delta t}|S_0] + \sigma(S_{t0+\Delta t}|S_0)\Delta W$.
  - apply(S_0, $\mathrm{d}S$): returns $S_0 + \mathrm{d}S$.
- Example: `/QuantLib_examples/3-Stochastic_processes.py`

QuantLib - StochasticProcess class



QuantLib - StochasticProcess1D class



QuantLib - Pricing engines

- Compilation of classes modelling pricing engines.
- Constructor: Stochastic process + engine arguments.
- Grouped into several modules:
  - Asian option engines.
  - Barrier option engines.
  - Basket option engines.
  - Cap/floor engines.
  - Cliquet option engines.
  - Forward option engines.
  - Quanto option engines.
  - Swaption engines.
  - Vanilla option engines.
- Depending on the solution technique:
  - Analytical engines.
  - Monte Carlo (MC) engines.
  - Binomial engines.
  - Finite-Differences (FD) engines.
  - Fourier Transform (FFT) engines.
  - Integral engines.

QuantLib - Vanilla option engines

- Analytic engines classes:
  - *AnalyticEuropeanEngine*, *AnalyticHestonEngine*, *AnalyticDigitalAmericanEngine*, *JumpDiffusionEngine*, etc.
- Monte Carlo (MC) engines classes:
  - *MCEuropeanEngine*, *MCAmericanEngine*, *MCEuropeanHestonEngine*, *MCDigitalEngine*, etc.
- Binomial engines classes:
  - *BinomialVanillaEngine*.
- Finite-Differences (FD) engines classes:
  - *FDEuropeanEngine*, *FDBermudanEngine*, *FDAmericanEngine*, *FdHestonHullWhiteVanillaEngine*, etc.
- Fourier Transform (FFT) engines classes:
  - *FFTVarianceGammaEngine*.
- Integral engines classes:
  - *IntegralEngine*, *VarianceGammaEngine*.
- Example: `/QuantLib_examples/4-European_pricing.py`

QuantLib - Mathematical tools

- Integration:
    - *TrapezoidIntegral*, *GaussLobattoIntegral*, etc.
- Solvers:
    - *Bisection*, *Newton*, *FiniteDifferenceNewtonSafe*, etc.
- Interpolation:
    - *LinearInterpolation*, *LogLinearInterpolation*, *CubicNaturalSpline*, etc.
- Matrix:
    - *Matrix*, *Array*, etc.
- Optimizer:
    - *ConjugateGradient*, *SteepestDescent*, *LevenbergMarquardt*, etc.
- Random numbers:
    - *MersenneTwisterUniformRng*, *BoxMullerGaussianRng*, etc.
- Statistical distributions:
    - *NormalDistribution*, *CumulativePoissonDistribution*, *InverseCumulativeStudent*, etc.
- Example: `/QuantLib_examples/5-Math_tools.py`

QuantLib - Examples

- Hands-on:
  - /QuantLib_examples/6-Heston.py
  - /QuantLib_examples/7-Heston_calibration.py
  - /QuantLib_examples/8-Implied_volatility.py
- Extra: /QuantLib_examples/9-HullWhite_simulation.py

Pandas

- Python tool for data manipulation and analysis.
- It stands for *PANel DAta*.
- Open-source → Free/Gratis/Gratis.
- It is built on top of *Numpy*.
- Highly optimized: expensive parts in Cython.
- Very well documented.
- Widely used for financial applications.
- Webpage: http://pandas.pydata.org/

Pandas - Some features

- Easy handling of missing data (represented as NaN).
- Size mutability: columns can be inserted and deleted.
- Automatic and explicit data alignment.
- Make it easy to convert Python and NumPy data structures into Pandas objects.
- Intuitive merging and joining data sets.
- Flexible reshaping and pivoting of data sets.
- Hierarchical labeling of axes (possible to have multiple labels per tick).
- IO tools for loading data from flat files (CSV and delimited), Excel files, databases, etc.
- Time series-specific functionality.

Pandas - Resources

- Documentation: http://pandas.pydata.org/pandas-docs/stable/
- Many sources of information:
  - Tutorials.
  - Video tutorials.
  - Online courses.
- Book: Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython by Wes McKinney.

Pandas

- Data structures.
  - *Series*.
  - *DataFrame*.
- Visualization.
- Time series.
- Data reader.

Pandas - Data structures - Series

- One-dimensional indexed (labelled) structure.
- Series(*data*, *index*):
  - *data* can be a list, dictionary, numpy *narray*, etc.
  - *index* is a list of labels (optional, by default $0, 1, \dots$).
- As *narray*, Series can be viewed, accessed, sliced, compared ($>$, $==$, etc), etc.
- Series handling: *max*, *min*, *sort*, etc.
- Statistics: *mean*, *std*, etc.
- Interoperability with NumPy element-wise math operations.
- Also the dictionary function: *in*, *get*, etc.
- Main difference: Series automatically align the data based on the label.
- Naming the series.
- Example: `/Pandas_examples/1-Series.py`

Pandas - Data structures - DataFrame

- 2-dimensional indexed (labelled) data structure.
- Like a Excel or SQL table.
- DataFrame(*data*, *index*, *columns*):
  - *data* can be a dictionary of lists, dictionaries, narrays or Series.
  - *data* can be a list of dictionaries.
  - *data* can be a 2D narray.
  - *data* can be a Series object.
  - *index* is a list of labels for rows (optional).
  - *columns* is a list of labels for columns (optional).
- The column of a DataFrame object is a Series object.

Pandas - Data structures - DataFrame

- Data alignment is intrinsic: the link between labels and data can not be broken unless done so explicitly.
- Many operations for accessing, addition, selection, alignment, etc.
- Interoperability with NumPy element-wise math operations.
- Operations between *DataFrame* and *Series*: by default, the *Series* index is aligned on the *DataFrame* columns (broadcasting row-wise).
- Higher dimensions: *Panel*, *Panel4D* and *PanelND* (experimental).
- Example: `/Pandas_examples/2-DataFrame.py`

Pandas - Data structures

- Viewing:
  - *head(n)/tail(n)*: returns the *n* first/last elements.
  - *index/columns*: returns the index/column of the structure.
  - *describe()*: returns statistical measures (mean, std, quantiles, etc.).
- Getting/Setting:
  - *['C']*: returns the column called *'C'*.
  - *[n:m]*: returns all the columns between *n* and *m* (slicing).
  - *loc['r']*: returns the row called *'r'*. Slicing (:) also available.
  - *at['r', 'C']*: returns the value at row *'r'* and *'C'*.
  - *iloc[i]*: same as *loc['r']* but using position *i*. Slicing (:) also available.
  - *ix['i']*: works with indexes or positions.
- Operations:
  - *mean()*, *std()*, *cumsum()*, *T*, etc.
  - *apply(f)*: applies *f* .
- Others:
  - Merging: *concat*, *join*, *append*, etc.
  - Grouping: *groupby*.
  - Reshaping: *stack/unstack*, *pivot_table*

Pandas - Visualization

- Pandas provides advanced visualization tools.
- Based on *Matplotlib*, easier to use.
- Several methods: *line* (line plot), *bar* (bars), *hist* (histograms), *box* (boxplots), *kde* (density), *area* (areas), *scatter* (scatter plots), *hexbin* (hexagonal bins) and *pie* (pie plots).
- The returning value is a *Matplotlib Axes* object.
- Highly customizable (color, legends, size, orientation, scales, etc).
- Other functions for special plots like Andrews curves, scatter matrix, density plot, autocorrelation plot, bootstrap plot, etc.
- *Matplotlib* can be also used (pandas structures act as Numpy arrays).
- Example: `/Pandas_examples/3-Visualization.py`

Pandas - Time series

- Pandas is suitable tool for time series data (financial data).
- Functionalities to:
  - generate sequences of fixed-frequency dates.
  - convert time series to a particular frequency.
  - compute “relative” dates based on various non-standard time increments.
- Based on the *datetime64* type of NumPy.
- Nanosecond precision.
- Main components: *Timestamp* and *Period*.
- List of *Timestamp*/*Period*: *DatetimeIndex* and *PeriodIndex*.
- Conversion from list-like structures, strings, integers, etc. into *DatetimeIndex*: *to_datetime(list)*.
- Used as indexes in *Series* and *DataFrame* objects.

Pandas - Time series (cont.)

- Generating ranges: *date_range(start, periods, freq)* and *bdate_range(start, periods, freq)*.
- Functionalities:
  - Optimized accessing, slicing, alignment manipulations.
  - Partial indexing: 'year', 'month', etc.
  - Truncation.
- Conversions between Timestamp and Period: *to_period* and *to_timestamp*.
- Many other functionalities: resampling, time zone handling, DateOffsets (implicit and explicit), etc.
- Click to documentation (DateOffsets).
- Example: `/Pandas_examples/4-TimeSeries.py`

Pandas - Data reader

- Functions to extract (financial) data from Internet sources.
- It returns a *DataFrame* object.
- The downloaded data is cached: the subsequent accesses will be faster.
- Currently supported sources: Yahoo! Finance, Google Finance, St.Louis FED (FRED), Kenneth French's data library, World Bank and Google Analytics.
- Useful functions: *DataReader(name, source, start, end)* and *Options(name, source)* (experimental, only Yahoo! Finance).
- Specific requests to avoid the download all the data: *get_call_data*, *expirity_dates*, etc.
- A lot of information from the World Bank (*wb* package): *search*, *download*, country codes, etc.
- Example: `/Pandas_examples/5-DataReader.py`