

FinancePy 0.193

Dominic O’Kane

March 7, 2021

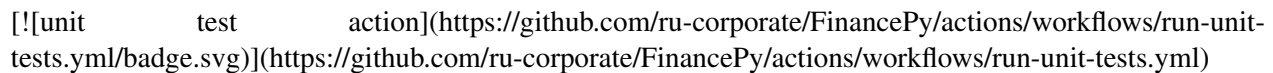
Contents

1	Introduction to FinancePy	3
2	financepy.utils	7
2.1	FinError	9
2.2	FinSolverCG	10
2.3	FinSolvers1D	12
2.4	FinSolversNM	15
3	financepy.market.discount	17
4	financepy.market.volatility	21
5	financepy.products.equity	23
6	financepy.products.credit	25
7	financepy.products.bonds	27
8	financepy.products.rates	29
8.1	FinFixedLeg	32
8.2	FinFloatLeg	34
8.3	FinIborBasisSwap	36
8.4	FinIborBermudanSwaption	38
8.5	FinIborCallableSwap	40
8.6	FinIborCapFloor	41
8.7	FinIborConventions	43
8.8	FinIborDeposit	44
8.9	FinIborDualCurve	46
8.10	FinIborFRA	47
8.11	FinIborFuture	49
8.12	FinIborLMMProducts	51
8.13	FinIborOIS	55
8.14	FinIborSingleCurve	57
8.15	FinIborSwaption	58
8.16	FinOIS	60
8.17	FinOISCurve	63
9	financepy.products.fx	65

10	financepy.models	67
10.1	FinModel	70
10.2	FinModelBlackScholesMCSpeedTests	71
10.3	FinModelMertonCreditMkt	73
10.4	FinModelOptionImpliedDbn	74

Chapter 1

Introduction to FinancePy

[![unit test action](https://github.com/ru-corporate/FinancePy/actions/workflows/run-unit-tests.yml/badge.svg)](https://github.com/ru-corporate/FinancePy/actions/workflows/run-unit-tests.yml)]

Quick Start Guide

FinancePy can be installed from pip using the following command:

‘pip install financepy‘

To upgrade an existing installation type:

‘pip install –upgrade financepy‘

I have encountered problems using Anaconda3-2020.07 due to some Numba and LLVM Lite problems. However Anaconda3-2020.02 works.

Using FinancePy in a Jupyter Notebook

Once financepy has been installed, it is easy to get started.

Just download the project and examine the set of Jupyter Notebooks in the notebooks folder.

A pdf manual describing all of the functions can be found in the project directory.

Overview

FinancePy is a python-based library that is currently in beta version. It covers the following functionality:

- Valuation and risk models for a wide range of equity, FX, interest rate and credit derivatives.

Although it is written entirely in Python, it can achieve speeds comparable to C++ by using Numba. As a result the user has both the ability to examine the underlying code and the ability to perform pricing and risk at speeds which compare to a library written in C++.

The target audience for this library includes:

- Students of finance and students of python
- Academics teaching finance or conducting research into finance
- Traders wishing to price or risk-manage a derivative.
- Quantitative analysts seeking to price or reverse engineer a price.

- Risk managers wishing to replicate and understand price sensitivity.
- Portfolio managers wishing to check prices or calculate risk measures.
- Fund managers wanting to value a portfolio or examine a trading strategy.

Users should have a good, but not advanced, understanding of Python. In terms of Python, the style of the library has been determined subject to the following criteria:

1. To make the code as simple as possible so that those with a basic Python fluency can understand and check the code.
2. To keep all the code in Python so users can look through the code to the lowest level.
3. To offset the performance impact of (2) by leveraging Numba to make the code as fast as possible without resorting to Cython.
4. To make the design product-based rather than model-based so someone wanting to price a specific product can easily find that without having to worry too much about the model – just use the default – unless they want to. For most products, a Monte-Carlo implementation has been provided both as a reference for testing and as a way to better understand how the product functions in terms of payments, their timings and conditions.
5. To make the library as complete as possible so a user can find all their required finance-related functionality in one place. This is better for the user as they only have to learn one interface.
6. To avoid complex designs. Limited inheritance unless it allows for significant code reuse. Some code duplication is OK, at least temporarily.
7. To have good documentation and easy-to-follow examples.
8. To make it easy for interested parties to contribute.

In many cases the valuations should be close to if not identical to those produced by financial systems such as Bloomberg. However for some products, larger value differences may arise due to differences in date generation and interpolation schemes. Over time it is hoped to reduce the size of such differences.

Important Note:

- IF YOU HAVE ANY PRICING OR RISK EXAMPLES YOU WOULD LIKE REPLICATED, SEND SCREENSHOTS OF ALL THE UNDERLYING DATA, MODEL DETAILS AND VALUATION.
- IF THERE IS A PRODUCT YOU WOULD LIKE TO HAVE ADDED, SEND ME THE REQUEST.
- IF THERE IS FUNCTIONALITY YOU WOULD LIKE ADDED, SEND ME A REQUEST.

Contact me at quant@financepy.com.

The Library Design

The underlying Python library is split into a number of major modules:

- **Finutils** - These are utility functions used to assist you with modelling a security. These include dates (FinDate), calendars, schedule generation, some finance-related mathematics functions and some helper functions.
- **Market** - These are modules that capture the market information used to value a security. These include interest rate and credit curves, volatility surfaces and prices.
- **Models** - These are the low-level models used to value derivative securities ranging from Black-Scholes to complex stochastic volatility models.
- **Products** - These are the actual securities and range from Government bonds to Bermudan swaptions.

Any product valuation is the result of the following data design:

- `*VALUATION** = **PRODUCT** + **MODEL** + **MARKET**`

The interface to each product has a `value()` function that will take a model and market to produce a price.

Author

Dominic O’Kane. I am a Professor of Finance at the EDHEC Business School in Nice, France. I have 12 years of industry experience and 10 years of academic experience.

Dependencies

FinancePy depends on Numpy, Numba, Scipy and basic python libraries such as os, sys and datetime.

Changelog

See the changelog for a detailed history of changes.

Contributions

Contributions are very welcome. There are a number of requirements:

- You can use either camel case or snail case. At some point I will try to make the code Pep8 compliant. But not yet.
- Comments are required for every class and function and they should be a clear description.
- At least one test case must be provided for every function.
- Avoid very pythonic constructions. For example a loop is as good as a list comprehension. And with numba it can be faster. Readability is the priority.

License

GPL-3.0 License - See the license file in this folder for details.

Chapter 2

financepy.utils

Introduction

This is a collection of modules used across a wide range of FinancePy functions. Examples include date generation, special mathematical functions and useful helper functions for performing some repeated action

- Date is a class for handling dates in a financial setting. Special functions are included for computing IMM dates and CDS dates and moving dates forward by tenors.
- FinCalendar is a class for determining which dates are not business dates in a specific region or country.
- FinDayCount is a class for determining accrued interest in bonds and also accrual factors in ISDA swap-like contracts.
- FinError is a class which handles errors in the calculations done within FinancePy
- `annual_frequency` takes in an `annual_frequency` type and then returns the number of payments per year `FinGlobalVariables` holds
- FinHelperFunctions is a set of helpful functions that can be used in a number of places
- FinMath is a set of mathematical functions specific to finance which have been optimised for speed using Numba
- FinSobol is the implementation of Sobol quasi-random number generator. It has been speeded up using Numba.
- FinRateConverter converts rates for one compounding `annual_frequency` to rates for a different `annual_frequency` `FinSchedule`
- FinStatistics calculates a number of statistical variables such as mean, standard deviation and variance
- FinTestCases is the code that underlies the test case framework used across FinancePy

FinDayCount

The `year fraction` function can take up to 3 dates, D1, D2 and D3 and a `annual_frequency` in specific cases. The current daycount methods are listed below.

- THIRTY 360 BOND - 30E/360 ISDA 2006 4.16f, German, Eurobond (ISDA 2000)

- THIRTY E 360 - ISDA 2006 4.16(g) 30/360 ISMA, ICMA
- THIRTY E 360 ISDA - ISDA 2006 4.16(h)
- THIRTY E PLUS 360 - A month has 30 days. It rolls D2 to next month if D2 = 31
- ACT ACT ISDA - Splits accrued period into leap and non-leap year portions.
- ACT ACT ICMA - Used for US Treasury notes and bonds. Takes 3 dates and a *annual_{frequency}*. *ACT365F – Denominator is always Fixed at 365, even in a leap year*
- ACT 360 - Day difference divided by 360 - always
- ACT 365L - the 29 Feb is counted if it is in the date range

2.1 FinError

Class: FinError(Exception)

class FinError(Exception):

FinError

Create FinError object by passing a message string.

```
FinError(message: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
message	str	-	-

func_name

return traceback.extract_stack(None, 2)[0][2]

```
func_name():
```

The function arguments are described in the following table.

suppressTraceback

print(sys.tracebacklimit)

```
suppressTraceback():
```

The function arguments are described in the following table.

2.2 FinSolverCG

Class: OptimizeResult(dict)

class OptimizeResult(dict):

polak_ribiere_powell_step

$$xkp1 = xk + \alpha * pk$$

```
polak_ribiere_powell_step(alpha, gfkp1=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
gfkp1	-	-	None

descent_condition

Polak-Ribiere+ needs an explicit check of a sufficient

```
descent_condition(alpha, xkp1, fp1, gfkp1):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
xkp1	-	-	-
fp1	-	-	-
gfkp1	-	-	-

fmin_cg

```
fmin_cg(f, x0, fprime=None, fargs=(), gtol=1e-5, norm=Inf, epsilon=_epsilon,
        maxiter=None, full_output=0, disp=1, retall=0, callback=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
x0	-	-	-
fprime	-	-	None
fargs	-	-	()
gtol	-	-	1e-5
norm	-	-	Inf
epsilon	-	-	_epsilon
maxiter	-	-	None
full_output	-	-	0
disp	-	-	1
retall	-	-	0
callback	-	-	None

vecnorm

if $ord == Inf$:

```
vecnorm(x, ord=2) :
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
ord	-	-	2

2.3 FinSolvers1D

newton_secant

Find a zero from the secant method using the jitted version of Scipy's secant method. Note that 'func' must be jitted via Numba. Parameters ———- *func* : callable and jitted The function whose zero is wanted. It must be a function of a single variable of the form $f(x,a,b,c\dots)$, where $a,b,c\dots$ are extra arguments that can be passed in the 'args' parameter. *x0* : float An initial estimate of the zero that should be somewhere near the actual zero. *args* : tuple, optional(default=()) Extra arguments to be used in the function call. *tol* : float, optional(default=1.48e-8) The allowable error of the zero value. *maxiter* : int, optional(default=50) Maximum number of iterations. *disp* : bool, optional(default=True) If True, raise a RuntimeError if the algorithm didn't converge. Returns ———- *results* : namedtuple A namedtuple containing the following items: *:: root* - Estimated location where function is zero. *function_calls* - Number of times the function was called. *iterations* - Number of iterations needed to find the root. *converged* - True if the routine converged

```
newton_secant(func, x0, args=(), tol=1.48e-8, maxiter=50,
              disp=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-
x0	-	-	-
args	-	-	()
tol	-	-	1.48e-8
maxiter	-	-	50
disp	-	-	True

newton

```
newton(func, x0, fprime=None, args=None, tol=1.48e-8, maxiter=50,
        fprime2=None, x1=None, rtol=0.0, full_output=False, disp=False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-
x0	-	-	-
fprime	-	-	None
args	-	-	None
tol	-	-	1.48e-8
maxiter	-	-	50
fprime2	-	-	None
x1	-	-	None
rtol	-	-	0.0
full_output	-	-	False
disp	-	-	False

brent_max

```
brent_max(func, a, b, args, xtol=1e-5, maxiter=500):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-
a	-	-	-
b	-	-	-
args	-	-	-
xtol	-	-	1e-5
maxiter	-	-	500

bisection

Bisection algorithm. You need to supply root brackets x1 and x2.

```
bisection(func, x1, x2, args, xtol=1e-6, maxIter=100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-
x1	-	-	-
x2	-	-	-
args	-	-	-
xtol	-	-	1e-6
maxIter	-	-	100

minimize_wolfe_powell

Minimize a differentiable multivariate function. Parameters ——— f : function to minimize. The function must return the value of the function (float) and a numpy array of partial derivatives of shape (D,) with respect to X, where D is the dimensionality of the function. X : numpy array - Shape : (D, 1) initial guess. length : int The length of the run. If positive, length gives the maximum number of line searches, if negative its absolute value gives the maximum number of function evaluations. args : tuple Tuple of parameters to be passed to the function f. reduction : float The expected reduction in the function value in the first linesearch (if None, defaults to 1.0) verbose : bool If True - prints the progress of minimize. (default is True) concise : bool If True - returns concise convergence info, only the minimum function value (necessary when optimizing a large number of parameters) (default is False) Return ——— Xs : numpy array - Shape : (D, 1) The found solution. convergence : numpy array - Shape : (i, D+1) Convergence information. The first column is the function values returned by the function being minimized. The next D columns are the guesses of X during the minimization process. If concise = True, convergence information is only the minimum function value. This is necessary only when optimizing a large number of parameters. i : int Number of line searches or function evaluations depending on which was selected. The function returns when either its length is up,

or if no further progress can be made (ie, we are at a (local) minimum, or so close that due to numerical problems, we cannot get any closer) Copyright (C) 2001 - 2006 by Carl Edward Rasmussen (2006-09-08). Converted to python by David Lines (2019-23-08)

```
minimize_wolfe_powel(f, X, length, fargs=(), reduction=None, verbose=False, concise=False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
X	-	-	-
length	-	-	-
fargs	-	-	()
reduction	-	-	None
verbose	-	-	False
concise	-	-	False

2.4 FinSolvers_{NM}

nelder_mead

```
nelder_mead(fun, x0, bounds=np.array([[ ], [ ]).T, args=(), tol_f=1e-10,
            tol_x=1e-10, max_iter=1000, roh=1., chi=2., v=0.5, sigma=0.5):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fun	-	-	-
x0	-	-	-
bounds	-	-	np.array([[], []).T
args	-	-	()
tol_f	-	-	1e-10
tol_x	-	-	1e-10
max_iter	-	-	1000
roh	-	-	1.
chi	-	-	2.
v	-	-	0.5
sigma	-	-	0.5

Chapter 3

financepy.market.discount

Curves

These modules create a family of curve types related to the term structures of interest rates. There are two basic types of curve:

1. Best fit yield curves fitting to bond prices which are used for interpolation. A range of curve shapes from polynomials to B-Splines is available.
2. Discount curves that can be used to present value a future cash flow. These differ from best fits curves in that they exactly refit the prices of bonds or CDS. The different discount curves are created by calibrating to different instruments. They also differ in terms of the term structure shapes they can have. Different shapes have different impacts in terms of locality on risk management performed using these different curves. There is often a trade-off between smoothness and locality.

Best Fit Bond Curves

The first category are BondYieldCurves.

BondYieldCurve

This module describes a curve that is fitted to bond yields calculated from bond market prices supplied by the user. The curve is not guaranteed to fit all of the bond prices exactly and a least squares approach is used. A number of fitting forms are provided which consist of

- Polynomial
- Nelson-Siegel
- Nelson-Siegel-Svensson
- Cubic B-Splines

This fitted curve cannot be used for pricing as yields assume a flat term structure. It can be used for fitting and interpolating yields off a nicely constructed yield curve interpolation curve.

FinCurveFitMethod

This module sets out a range of curve forms that can be fitted to the bond yields. These includes a number of parametric curves that can be used to fit yield curves. These include:

- Polynomials of any degree
- Nelson-Siegel functional form.
- Nelson-Siegel-Svensson functional form.
- B-Splines

Discount Curves

These are curves which supply a discount factor that can be used to present-value future payments.

FinDiscountCurve

This is a curve made from a Numpy array of times and discount factor values that represents a discount curve. It also requires a specific interpolation scheme. A function is also provided to return a survival probability so that this class can also be used to handle term structures of survival probabilities. Other curves inherit from this in order to share common functionality.

FinDiscountCurveFlat

This is a class that takes in a single flat rate.

FinDiscountCurveNS

Implementation of the Nelson-Siegel curve parametrisation.

FinDiscountCurveNSS

Implementation of the Nelson-Siegel-Svensson curve parametrisation.

FinDiscountCurveZeros

This is a discount curve that is made from a vector of times and zero rates.

FinInterpolate

This module contains the interpolation function used throughout the discount curves when a discount factor needs to be interpolated. There are three interpolation methods:

1. **PIECEWISE LINEAR** - This assumes that a discount factor at a time between two other known discount factors is obtained by linear interpolation. This approach does not guarantee any smoothness but is local. It does not guarantee positive forwards (assuming positive zero rates).

2. **PIECEWISE LOG LINEAR** - This assumes that the log of the discount factor is interpolated linearly. The log of a discount factor to time T is $T \times R(T)$ where $R(T)$ is the zero rate. So this is not linear interpolation of $R(T)$ but of $T \times R(T)$.
3. **FLAT FORWARDS** - This interpolation assumes that the forward rate is constant between discount factor points. It is not smooth but is highly local and also ensures positive forward rates if the zero rates are positive.

Chapter 4

financepy.market.volatility

Market Volatility

These modules create a family of curve types related to the market volatility. There are three types of class:

1. Term structures of volatility i.e. volatility as a function of option expiry date.
2. Volatility curves which are smile/skews so store volatility as a function of option strike.
3. Volatility surfaces which hold volatility as a function of option expiry date AND option strike.

The classes are as follows:

equity_{vol}surface

Constructs an equity volatility surface that fits to a grid of market volatilities at a set of strikes and expiry dates. It implements the SVI parameteric form for fitting and interpolating volatilities. It also provides plotting of the volatility curve and surfaces.

FinFXVolSurface

FX volatility as a function of option expiry and strike. This class constructs the surface from the ATM volatility plus a choice of 10 and 25 delta strangles and risk reversals or both. This is done for multiple expiry dates. A number of curve fitting choices are possible including polynomial in delta and SABR.

FinlborCapFloorVol

Libor cap/floor volatility as a function of option expiry (cap/floor start date). Takes in cap (flat) volatility and bootstraps the caplet volatility. This is assumed to be piecewise flat.

FinlborCapFloorVolFn

Parametric function for storing the cap and caplet volatilities based on form proposed by Rebonato.

Chapter 5

financepy.products.equity

Equity Products

This folder contains a set of Equity-related products. It includes:

EquityVanillaOption

Handles simple European-style call and put options on a dividend paying stock with analytical and monte-carlo valuations.

EquityAmericanOption

Handles America-style call and put options on a dividend paying stock with tree-based valuations.

EquityAsianOption

Handles call and put options where the payoff is determined by the average-stock price over some period before expiry.

EquityBasketOption

Handles call and put options on a basket of assets, with an analytical and Monte-Carlo valuation according to Black-Scholes model.

EquityCompoundOption

Handles options to choose to enter into a call or put option. Has an analytical valuation model for European style options and a tree model if either or both options are American style exercise.

EquityDigitalOption

Handles European-style options to receive cash or nothing, or to receive the asset or nothing. Has an analytical valuation model for European style options.

EquityFixedLookbackOption

Handles European-style options to receive the positive difference between the strike and the minimum (put) or maximum (call) of the stock price over the option life.

EquityFloatLookbackOption

Handles an equity option in which the strike of the option is not fixed but is set at expiry to equal the minimum stock price in the case of a call or the maximum stock price in the case of a put. In other words the buyer of the call gets to buy the asset at the lowest price over the period before expiry while the buyer of the put gets to sell the asset at the highest price before expiry. """

EquityBarrierOption

Handles an option which either knocks-in or knocks-out if a specified barrier is crossed from above or below, resulting in owning or not owning a call or a put option. There are eight variations which are all valued.

EquityRainbowOption

TBD

EquityVarianceSwap

TBD

Products that have not yet been implemented include:

- Power Options
- Ratchet Options
- Forward Start Options
- Log Options

Chapter 6

financepy.products.credit

This folder contains a set of credit-related assets ranging from CDS to CDS options, to CDS indices, CDS index options and then to CDS tranches. They are as follows:

- CDS is a credit default swap contract. It includes schedule generation, contract valuation and risk-management functionality.
- CDSBasket is a credit default basket such as a first-to-default basket. The class includes valuation according to the Gaussian copula.
- CDSCurve is a discount curve and survival curve constructed from discount rates and CDS spreads.
- CDSIndexOption is an option on an index of CDS such as CDX or iTraxx. A full valuation model is included.
- CDSIndexPortfolio is a portfolio of CDS contracts.
- CDSOption is an option on a single CDS. The strike is expressed in spread terms and the option is European style. It is different from an option on a CDS index option. A suitable pricing model is provided which adjusts for the risk that the reference credit defaults before the option expiry date.
- CDSTranche is a synthetic CDO tranche. This is a financial derivative which takes a loss if the total loss on the portfolio exceeds a lower threshold K1 and which is wiped out if it exceeds a higher threshold K2. The value depends on the default correlation between the assets in the portfolio of credits. This also includes a valuation model based on the Gaussian copula model.

FinCDSCurve

This is a curve that has been calibrated to fit the market term structure of CDS contracts given a recovery rate assumption and a `FinIborSingleCurve` discount curve. It also contains a `IborCurve` object for discounting. It has methods for fitting the curve and also for extracting survival probabilities.

Chapter 7

financepy.products.bonds

This folder contains a suite of bond-related functionality across a set of files and classes. They are as follows:

- Bond is a basic fixed coupon bond with all of the associated duration and convexity measures. It also includes some common spread measures such as the asset swap spread and the option adjusted spread.
- BondAnnuity is a stream of cash flows that is generated and can be priced.
- BondCallable is a bond that has embedded call and put options. A number of rate models pricing functions have been included to allow such bonds to be priced and risk-managed.
- BondConvertible enables the pricing and risk-management of convertible bonds. The model is a binomial tree implementation of Black-Scholes which allows for discrete dividends, embedded puts and calls, and a delayed start of the conversion option.
- BondFRN enables the pricing and risk-management of a bond with floating rate coupons. Discount margin calculations are provided.
- BondFuture is a bond future that has functionality around determination of the conversion factor and calculation of the invoice price and determination of the cheapest to deliver.
- BondMarket is a database of country-specific bond market conventions that can be referenced. These include settlement days and accrued interest conventions.
- BondMortgage generates the periodic cash flows for an interest-only and a repayment mortgage.
- BondOption is a bond option class that includes a number of valuation models for pricing both European and American style bond options. Models for European options include a Lognormal Price, Hull-White (HW) and Black-Karasinski (BK). The HW valuation is fast as it uses Jamshidians decomposition trick. American options can also be priced using a HW and BK trinomial tree. The details are abstracted away making it easy to use.
- BondPortfolio is a portfolio of bonds.

Conventions

- All interest rates are expressed as a fraction of 1. So 3
- All notionals of bond positions are given in terms of a notional amount.

- All bond prices are based on a notional of 100.0.
- The face of a derivatives position is the size of the underlying position.

Bond Curves

These modules create a family of curve types related to the term structures of interest rates. There are two basic types of curve:

1. Best fit yield curves fitting to bond prices which are used for interpolation. A range of curve shapes from polynomials to B-Splines is available.
2. Discount curves that can be used to present value a future cash flow. These differ from best fits curves in that they exactly refit the prices of bonds or CDS. The different discount curves are created by calibrating to different instruments. They also differ in terms of the term structure shapes they can have. Different shapes have different impacts in terms of locality on risk management performed using these different curves. There is often a trade-off between smoothness and locality.

BondYieldCurve

This module describes a curve that is fitted to bond yields calculated from bond market prices supplied by the user. The curve is not guaranteed to fit all of the bond prices exactly and a least squares approach is used. A number of fitting forms are provided which consist of

- Polynomial
- Nelson-Siegel
- Nelson-Siegel-Svensson
- Cubic B-Splines

This fitted curve cannot be used for pricing as yields assume a flat term structure. It can be used for fitting and interpolating yields off a nicely constructed yield curve interpolation curve.

FinCurveFitMethod

This module sets out a range of curve forms that can be fitted to the bond yields. These includes a number of parametric curves that can be used to fit yield curves. These include:

- Polynomials of any degree
- Nelson-Siegel functional form.
- Nelson-Siegel-Svensson functional form.
- B-Splines

Chapter 8

financepy.products.rates

Funding

This folder contains a set of funding-related products. These reflect contracts linked to funding indices such as Ibors and Overnight index rate swaps (OIS). It includes:

FinIborDeposit

This is the basic Ibor instrument in which a party borrows an amount for a specified term and rate unsecured.

FinInterestRateFuture

This is a class to handle interest rate futures contracts. This is an exchange-traded contract to receive or pay Ibor on a specified future date. It can be used to build the Libor term structure.

FinIborFRA

This is a class to manage Forward Rate Agreements (FRAs) in which one party agrees to lock in a forward Ibor rate.

Swaps

FinFixedIborSwap

This is a contract to exchange fixed rate coupons for floating Ibor rates. This class has functionality to value the swap contract and to calculate its risk.

FinFixedIborSwap - IN PROGRESS

This is a contract to exchange fixed rate coupons for floating Ibor rates. This class has functionality to value the swap contract and to calculate its risk.

FinIborIborSwap - IN PROGRESS

This is a contract to exchange IBOR rates with different terms, also known as a basis swap. This class has functionality to value the swap contract and to calculate its risk.

FinFixedOISwap - IN PROGRESS

This is an OIS, a contract to exchange fixed rate coupons for the overnight index rate. This class has functionality to value the swap contract and to calculate its risk.

FinIborOISwap - IN PROGRESS

This is a contract to exchange overnight index rates for Ibor rates. This class has functionality to value the swap contract and to calculate its risk.

Currency Swaps***FinFixedFixedCcySwap - IN PROGRESS***

This is a contract to exchange fixed rate coupons in two different currencies. This class has functionality to value the swap contract and to calculate its risk.

FinIborIborCcySwap - IN PROGRESS

This is a contract to exchange IBOR coupons in two different currencies. This class has functionality to value the swap contract and to calculate its risk.

FinOIS

This is a contract to exchange the daily compounded Overnight index swap rate for a fixed rate agreed at contract initiation.

FinOISCurve

This is a discount curve that is extracted by bootstrapping a set of OIS rates. The internal representation of the curve are discount factors on each of the OIS dates. Between these dates, discount factors are interpolated according to a specified scheme.

FinIborSingleCurve

This is a discount curve that is extracted by bootstrapping a set of Ibor deposits, Ibor FRAs and Ibor swap prices. The internal representation of the curve are discount factors on each of the deposit, FRA and swap maturity dates. Between these dates, discount factors are interpolated according to a specified scheme - see below.

Options

FinIborCapFloor

FinIborSwaption

This is a contract to buy or sell an option to enter into a swap to either pay or receive a fixed swap rate at a specific future expiry date. The model includes code that prices a payer or receiver swaption with the following models:

- Black's Model - Shifted Black Model - SABR - Shifted SABR - Hull-White Tree Model - Black-Karasinski Tree Model - Black-Derman-Toy Tree Model

FinIborBermudanSwaption

This is a contract to buy or sell an option to enter into a swap to either pay or receive a fixed swap rate at a specific future expiry date on specific coupon dates starting on a designated expiry date. The model includes code that prices a payer or receiver swaption with the following models:

- Hull-White Tree Model - Black-Karasinski Tree Model - Black-Derman-Toy Tree Model

It is also possible to price this using a Ibor Market Model. However for the moment this must be done directly via the Monte-Carlo implementation of the LMM found in `FinModelRatesLMM`.

8.1 FinFixedLeg

Class: FinFixedLeg

class FinFixedLeg:

FinFixedLe

Create the fixed leg of a swap contract giving the contract start date, its maturity, fixed coupon, fixed leg frequency, fixed leg day count convention and notional.

```
FinFixedLe(effective_date: Date, # Date interest starts to accrue
            end_date: (Date, str), # Date contract ends
            leg_type: FinSwapTypes,
            coupon: (float),
            freq_type: FrequencyTypes,
            day_count_type: DayCountTypes,
            notional: float = ONE_MILLION,
            principal: float = 0.0,
            payment_lag: int = 0,
            calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
            bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
            date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_date	Date	Date interest starts to accrue	-
end_date	Date or str	Date contract ends	-
leg_type	FinSwapTypes	-	-
coupon	float or (float	-	-
freq_type	FrequencyTypes	-	-
day_count_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
payment_lag	int	-	0
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

generatePayments

These are generated immediately as they are for the entire

```
generatePayments():
```

The function arguments are described in the following table.

value*PLEASE ADD A FUNCTION DESCRIPTION*

```
value(valuation_date: Date,
      discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	DiscountCurve	-	-

printPayments

Prints the fixed leg dates, accrual factors, discount factors, cash amounts, their present value and their cumulative PV using the last valuation performed.

```
printPayments():
```

The function arguments are described in the following table.

printValuation

Prints the fixed leg dates, accrual factors, discount factors, cash amounts, their present value and their cumulative PV using the last valuation performed.

```
printValuation():
```

The function arguments are described in the following table.

8.2 FinFloatLeg

Class: FinFloatLeg

class FinFloatLeg:

FinFloatLe

Create the fixed leg of a swap contract giving the contract start date, its maturity, fixed coupon, fixed leg frequency, fixed leg day count convention and notional.

```
FinFloatLe(effective_date: Date, # Date interest starts to accrue
            end_date: (Date, str), # Date contract ends
            leg_type: FinSwapTypes,
            spread: (float),
            freq_type: FrequencyTypes,
            day_count_type: DayCountTypes,
            notional: float = ONE_MILLION,
            principal: float = 0.0,
            payment_lag: int = 0,
            calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
            bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
            date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_date	Date	Date interest starts to accrue	-
end_date	Date or str	Date contract ends	-
leg_type	FinSwapTypes	-	-
spread	float or (float	-	-
freq_type	FrequencyTypes	-	-
day_count_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
payment_lag	int	-	0
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

generatePaymentDates

Generate the floating leg payment dates and accrual factors. The coupons cannot be generated yet as we do not have the index curve.

```
generatePaymentDates():
```

The function arguments are described in the following table.

value

Value the floating leg with payments from an index curve and discounting based on a supplied discount curve as of the valuation date supplied. For an existing swap, the user must enter the next fixing coupon.

```
value(valuation_date: Date, # This should be the settlement date
      discount_curve: DiscountCurve,
      index_curve: DiscountCurve,
      firstFixingRate: float=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	This should be the settlement date	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	-
firstFixingRate	float	-	None

printPayments

Prints the fixed leg dates, accrual factors, discount factors, cash amounts, their present value and their cumulative PV using the last valuation performed.

```
printPayments():
```

The function arguments are described in the following table.

printValuation

Prints the fixed leg dates, accrual factors, discount factors, cash amounts, their present value and their cumulative PV using the last valuation performed.

```
printValuation():
```

The function arguments are described in the following table.

8.3 FinIborBasisSwap

Class: FinIborBasisSwap

class FinIborBasisSwap

FinIborBasisSwa

Create a Ibor basis swap contract giving the contract start date, its maturity, frequency and day counts on the two floating legs and notional. The floating leg parameters have default values that can be overwritten if needed. The start date is contractual and is the same as the settlement date for a new swap. It is the date on which interest starts to accrue. The end of the contract is the termination date. This is not adjusted for business days. The adjusted termination date is called the maturity date. This is calculated.

```
FinIborBasisSwa(effective_date: Date, # Date interest starts to accrue
                termination_date_or_tenor: (Date, str), # Date contract ends
                leg1Type: FinSwapTypes,
                leg1FreqType: FrequencyTypes = FrequencyTypes.QUARTERLY,
                leg1DayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
                leg1Spread: float = 0.0,
                leg2FreqType: FrequencyTypes = FrequencyTypes.QUARTERLY,
                leg2DayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
                leg2Spread: float = 0.0,
                notional: float = ONE_MILLION,
                calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
                bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_date	Date	Date interest starts to accrue	-
termination_date_or_tenor	Date or str	Date contract ends	-
leg1Type	FinSwapTypes	-	-
leg1FreqType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
leg1DayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
leg1Spread	float	-	0.0
leg2FreqType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
leg2DayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
leg2Spread	float	-	0.0
notional	float	-	ONE_MILLION
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Value the interest rate swap on a value date given a single Ibor discount curve and an index curve for the

Ibors on each swap leg.

```
value(valuation_date: Date,
      discount_curve: DiscountCurve,
      index_curveLeg1: DiscountCurve = None,
      index_curveLeg2: DiscountCurve = None,
      firstFixingRateLeg1=None,
      firstFixingRateLeg2=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	DiscountCurve	-	-
index_curveLeg1	DiscountCurve	-	None
index_curveLeg2	DiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

printFloatLeg1PV

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
printFloatLeg1PV():
```

The function arguments are described in the following table.

printFloatLeg2PV

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
printFloatLeg2PV():
```

The function arguments are described in the following table.

print_flows

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
print_flows():
```

The function arguments are described in the following table.

8.4 FinIborBermudanSwaption

Class: FinIborBermudanSwaption

class FinIborBermudanSwaption:

FinIborBermudanSwaption

Create a Bermudan swaption contract. This is an option to enter into a payer or receiver swap at a fixed coupon on all of the fixed # leg coupon dates until the exercise date inclusive.

```
FinIborBermudanSwaption(settlement_date: Date,
                        exerciseDate: Date,
                        maturity_date: Date,
                        fixed_legType: FinSwapTypes,
                        exerciseType: FinExerciseTypes,
                        fixedCoupon: float,
                        fixedFrequencyType: FrequencyTypes,
                        fixedDayCountType: DayCountTypes,
                        notional=ONE_MILLION,
                        floatFrequencyType=FrequencyTypes.QUARTERLY,
                        floatDayCountType=DayCountTypes.THIRTY_E_360,
                        calendar_type=CalendarTypes.WEEKEND,
                        bus_day_adjust_type=BusDayAdjustTypes.FOLLOWING,
                        date_gen_rule_type=DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlement_date	Date	-	-
exerciseDate	Date	-	-
maturity_date	Date	-	-
fixed_legType	FinSwapTypes	-	-
exerciseType	FinExerciseTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FrequencyTypes	-	-
fixedDayCountType	DayCountTypes	-	-
notional	-	-	ONE_MILLION
floatFrequencyType	-	-	FrequencyTypes.QUARTERLY
floatDayCountType	-	-	DayCountTypes.THIRTY_E_360
calendar_type	-	-	CalendarTypes.WEEKEND
bus_day_adjust_type	-	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	-	-	DateGenRuleTypes.BACKWARD

value

Value the Bermudan swaption using the specified model and a discount curve. The choices of model are the Hull-White model, the Black-Karasinski model and the Black-Derman-Toy model.

```
value(valuation_date,  
      discount_curve,  
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
discount_curve	-	-	-
model	-	-	-

printSwaptionValue

PLEASE ADD A FUNCTION DESCRIPTION

```
printSwaptionValue():
```

The function arguments are described in the following table.

8.5 FinlborCallableSwap

8.6 FinIborCapFloor

Enumerated Type: FinIborCapFloorModelTypes

This enumerated type has the following values:

- BLACK
- SHIFTED_BLACK
- SABR

Class: FinIborCapFloor()

class FinIborCapFloor():

FinIborCapFloor

Initialise FinIborCapFloor object.

```
FinIborCapFloor(start_date: Date,
                 maturity_date_or_tenor: (Date, str),
                 option_type: FinCapFloorTypes,
                 strikeRate: float,
                 lastFixing: Optional[float] = None,
                 freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
                 day_count_type: DayCountTypes = DayCountTypes.THIRTY_E_360_ISDA,
                 notional: float = ONE_MILLION,
                 calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
                 bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                 date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_date	Date	-	-
maturity_date_or_tenor	Date or str	-	-
option_type	FinCapFloorTypes	-	-
strikeRate	float	-	-
lastFixing	Optional[float]	-	None
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
day_count_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360_ISDA
notional	float	-	ONE_MILLION
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Value the cap or floor using the chosen model which specifies the volatility of the Ibor rate to the cap start date.

```
value(valuation_date, libor_curve, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
libor_curve	-	-	-
model	-	-	-

valueCapletFloorLet

Value the caplet or floorlet using a specific model.

```
valueCapletFloorLet(valuation_date,
                    capletStartDate,
                    capletEndDate,
                    libor_curve,
                    model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
capletStartDate	-	-	-
capletEndDate	-	-	-
libor_curve	-	-	-
model	-	-	-

printLeg

Prints the cap floor payment amounts.

```
printLeg():
```

The function arguments are described in the following table.

8.7 FinIborConventions

Class: FinIborConventions()

class FinIborConventions():

FinIborConventions

PLEASE ADD A FUNCTION DESCRIPTION

```
FinIborConventions(currencyName: str,  
                    indexName: str = "LIBOR"):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
currencyName	str	-	-
indexName	str	-	"LIBOR"

8.8 FinIborDeposit

Class: FinIborDeposit

class FinIborDeposit:

FinIborDeposi

Create a Libor deposit object which takes the start date when the amount of notional is borrowed, a maturity date or a tenor and the deposit rate. If a tenor is used then this is added to the start date and the calendar and business day adjustment method are applied if the maturity date fall on a holiday. Note that in order to calculate the start date you add the spot business days to the trade date which usually today.

```
FinIborDeposi(start_date: Date, # When the interest starts to accrue
              maturity_date_or_tenor: (Date, str), # Repayment of interest
              deposit_rate: float, # MM rate using simple interest
              day_count_type: DayCountTypes, # How year fraction is calculated
              notional: float = 100.0, # Amount borrowed
              calendar_type: CalendarTypes = CalendarTypes.WEEKEND, # Holidays for
                           maturity date
              bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.
                           MODIFIED_FOLLOWING):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_date	Date	When the interest starts to accrue	-
maturity_date_or_tenor	Date or str	Repayment of interest	-
deposit_rate	float	MM rate using simple interest	-
day_count_type	DayCountTypes	How year fraction is calculated	-
notional	float	Amount borrowed	100.0
calendar_type	CalendarTypes	Holidays for maturity date	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.MODIFIED_FOL

value

Determine the value of an existing Libor Deposit contract given a valuation date and a Libor curve. This is simply the PV of the future repayment plus interest discounted on the current Libor curve.

```
value(valuation_date: Date,
      libor_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
libor_curve	-	-	-

print_flows

Print the date and size of the future repayment.

```
print_flows(valuation_date: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-

8.9 FinIborDualCurve

Class: IborDualCurve(DiscountCurve)

class IborDualCurve(DiscountCurve):

IborDualCurve

Create an instance of a FinIbor curve given a valuation date and a set of ibor deposits, ibor FRAs and iborSwaps. Some of these may be left None and the algorithm will just use what is provided. An interpolation method has also to be provided. The default is to use a linear interpolation for swap rates on coupon dates and to then assume flat forwards between these coupon dates. The curve will assign a discount factor of 1.0 to the valuation date.

```
IborDualCurve(valuation_date: Date,
               discount_curve: DiscountCurve,
               iborDeposits: list,
               iborFRAs: list,
               iborSwaps: list,
               interp_type: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES,
               checkRefit: bool = False): # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	DiscountCurve	-	-
iborDeposits	list	-	-
iborFRAs	list	-	-
iborSwaps	list	-	-
interp_type	FinInterpTypes	-	FLAT_FWD_RATES
checkRefit	bool	Set to True to test it works	False

8.10 FinIborFRA

Class: FinIborFRA

class FinIborFRA:

FinIborFR

Create a Forward Rate Agreement object.

```
FinIborFR(start_date: Date, # The date the FRA starts to accrue
          maturity_date_or_tenor: (Date, str), # End of the Ibor rate period
          fraRate: float, # The fixed contractual FRA rate
          day_count_type: DayCountTypes, # For interest period
          notional: float = 100.0,
          payFixedRate: bool = True, # True if the FRA rate is being paid
          calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
          bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.MODIFIED_FOLLOWING
          ):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_date	Date	The date the FRA starts to accrue	-
maturity_date_or_tenor	Date or str	End of the Ibor rate period	-
fraRate	float	The fixed contractual FRA rate	-
day_count_type	DayCountTypes	For interest period	-
notional	float	-	100.0
payFixedRate	bool	True if the FRA rate is being paid	True
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.MODIFIED_FO

value

Determine mark to market value of a FRA contract based on the market FRA rate. We allow the pricing to have a different curve for the Libor index and the discounting of promised cash flows.

```
value(valuation_date: Date,
      discount_curve: DiscountCurve,
      index_curve: DiscountCurve = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	None

maturityDf

Determine the maturity date index discount factor needed to refit the market FRA rate. In a dual-curve world, this is not the discount rate discount factor but the index curve discount factor.

```
maturityDf(index_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
index_curve	-	-	-

print_flows

Determine the value of the Deposit given a Ibor curve.

```
print_flows(valuation_date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-

8.11 FinIborFuture

Class: FinIborFuture

class FinIborFuture:

FinIborFutur

Create an interest rate futures contract which has the same conventions as those traded on the CME. The current date, the tenor of the future, the number of the future and the accrual convention and the contract size should be provided.

```
FinIborFutur(todayDate: Date,
             futureNumber: int, # The number of the future after todayDate
             futureTenor: str = "3M", # '1M', '2M', '3M'
             accrual_type: DayCountTypes = DayCountTypes.ACT_360,
             contractSize: float = ONE_MILLION):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
todayDate	Date	-	-
futureNumber	int	The number of the future after todayDate	-
futureTenor	str	'1M', '2M', '3M'	"3M"
accrual_type	DayCountTypes	-	DayCountTypes.ACT_360
contractSize	float	-	ONE_MILLION

toFRA

Convert the futures contract to a FinIborFRA object so it can be used to bootstrap a Ibor curve. For this we need to adjust the futures rate using the convexity correction.

```
toFRA(futures_price, convexity):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futures_price	-	-	-
convexity	-	-	-

futuresRate

Calculate implied futures rate from the futures price.

```
futuresRate(futures_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futures_price	-	-	-

FRARate

Convert futures price and convexity to a FRA rate using the BBG negative convexity (in percent). This is then divided by 100 before being added to the futures rate.

```
FRARate(futures_price, convexity):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futures_price	-	-	-
convexity	-	-	-

convexity

Calculation of the convexity adjustment between FRAs and interest rate futures using the Hull-White model as described in technical note in link below: <http://www-2.rotman.utoronto.ca/hull/TechnicalNotes/TechnicalNote1.pdf> NOTE THIS DOES NOT APPEAR TO AGREE WITH BLOOMBERG!! INVESTIGATE.

```
convexity(valuation_date, volatility, meanReversion):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
volatility	-	-	-
meanReversion	-	-	-

8.12 FinIborLMMProducts

Class: *FinIborLMMProducts()*

class FinIborLMMProducts():

FinIborLMMProducts

Create a European-style swaption by defining the exercise date of the swaption, and all of the details of the underlying interest rate swap including the fixed coupon and the details of the fixed and the floating leg payment schedules.

```
FinIborLMMProducts(settlement_date: Date,
                    maturity_date: Date,
                    floatFrequencyType: FrequencyTypes = FrequencyTypes.QUARTERLY,
                    floatDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
                    calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
                    bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING
                    ,
                    date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlement_date	Date	-	-
maturity_date	Date	-	-
floatFrequencyType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
floatDayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

simulate1F

Run the one-factor simulation of the evolution of the forward Ibors to generate and store all of the Ibor forward rate paths.

```
simulate1F(discount_curve,
           volCurve: IborCapVolCurve,
           num_paths: int = 1000,
           numeraireIndex: int = 0,
           useSobol: bool = True,
           seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discount_curve	-	-	-
volCurve	IborCapVolCurve	-	-
num_paths	int	-	1000
numeraireIndex	int	-	0
useSobol	bool	-	True
seed	int	-	42

simulateMF

Run the simulation to generate and store all of the Ibor forward rate paths. This is a multi-factorial version so the user must input a numpy array consisting of a column for each factor and the number of rows must equal the number of grid times on the underlying simulation grid. CHECK THIS.

```
simulateMF(discount_curve,
           numFactors: int,
           lambdas: np.ndarray,
           num_paths: int = 10000,
           numeraireIndex: int = 0,
           useSobol: bool = True,
           seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discount_curve	-	-	-
numFactors	int	-	-
lambdas	np.ndarray	-	-
num_paths	int	-	10000
numeraireIndex	int	-	0
useSobol	bool	-	True
seed	int	-	42

simulateNF

Run the simulation to generate and store all of the Ibor forward rate paths using a full factor reduction of the fwd-fwd correlation matrix using Cholesky decomposition.

```
simulateNF(discount_curve,
           volCurve: IborCapVolCurve,
           correlationMatrix: np.ndarray,
           modelType: FinRateModelLMMModelTypes,
           num_paths: int = 1000,
           numeraireIndex: int = 0,
           useSobol: bool = True,
           seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discount_curve	-	-	-
volCurve	IborCapVolCurve	-	-
correlationMatrix	np.ndarray	-	-
modelType	FinRateModelLMMModelTypes	-	-
num_paths	int	-	1000
numeraireIndex	int	-	0
useSobol	bool	-	True
seed	int	-	42

valueSwaption

Value a swaption in the LMM model using simulated paths of the forward curve. This relies on pricing the fixed leg of the swap and assuming that the floating leg will be worth par. As a result we only need simulate Ibors with the frequency of the fixed leg.

```
valueSwaption(settlement_date: Date,
               exerciseDate: Date,
               maturity_date: Date,
               swaptionType: FinSwapTypes,
               fixedCoupon: float,
               fixedFrequencyType: FrequencyTypes,
               fixedDayCountType: DayCountTypes,
               notional: float = ONE_MILLION,
               floatFrequencyType: FrequencyTypes = FrequencyTypes.QUARTERLY,
               floatDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
               calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
               bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
               date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlement_date	Date	-	-
exerciseDate	Date	-	-
maturity_date	Date	-	-
swaptionType	FinSwapTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FrequencyTypes	-	-
fixedDayCountType	DayCountTypes	-	-
notional	float	-	ONE_MILLION
floatFrequencyType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
floatDayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

valueCapFloor

Value a cap or floor in the LMM.

```
valueCapFloor(settlement_date: Date,
               maturity_date: Date,
               capFloorType: FinCapFloorTypes,
               capFloorRate: float,
               frequencyType: FrequencyTypes = FrequencyTypes.QUARTERLY,
               day_count_type: DayCountTypes = DayCountTypes.ACT_360,
               notional: float = ONE_MILLION,
               calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
               bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
               date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlement_date	Date	-	-
maturity_date	Date	-	-
capFloorType	FinCapFloorTypes	-	-
capFloorRate	float	-	-
frequencyType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
day_count_type	DayCountTypes	-	DayCountTypes.ACT_360
notional	float	-	ONE_MILLION
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

8.13 FinIborOIS

Class: FinIborOIS

class FinIborOIS

FinIborOI

Create a Ibor basis swap contract giving the contract start date, its maturity, frequency and day counts on the two floating legs and notional. The floating leg parameters have default values that can be overwritten if needed. The start date is contractual and is the same as the settlement date for a new swap. It is the date on which interest starts to accrue. The end of the contract is the termination date. This is not adjusted for business days. The adjusted termination date is called the maturity date. This is calculated.

```
FinIborOI(effective_date: Date, # Date interest starts to accrue
          termination_date_or_tenor: (Date, str), # Date contract ends
          iborType: FinSwapTypes,
          iborFreqType: FrequencyTypes = FrequencyTypes.QUARTERLY,
          iborDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
          iborSpread: float = 0.0,
          oisFreqType: FrequencyTypes = FrequencyTypes.QUARTERLY,
          oisDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
          oisSpread: float = 0.0,
          oisPaymentLag: int = 0,
          notional: float = ONE_MILLION,
          calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
          bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
          date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_date	Date	Date interest starts to accrue	-
termination_date_or_tenor	Date or str	Date contract ends	-
iborType	FinSwapTypes	-	-
iborFreqType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
iborDayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
iborSpread	float	-	0.0
oisFreqType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
oisDayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
oisSpread	float	-	0.0
oisPaymentLag	int	-	0
notional	float	-	ONE_MILLION
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Value the interest rate swap on a value date given a single Ibor discount curve and an index curve for the Ibors on each swap leg.

```
value(valuation_date: Date,
      discount_curve: DiscountCurve,
      indexIborCurve: DiscountCurve = None,
      indexOISCurve: DiscountCurve = None,
      firstFixingRateLeg1=None,
      firstFixingRateLeg2=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	DiscountCurve	-	-
indexIborCurve	DiscountCurve	-	None
indexOISCurve	DiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

print_flows

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
print_flows():
```

The function arguments are described in the following table.

8.14 FinIborSingleCurve

Class: *IborSingleCurve(DiscountCurve)*

class IborSingleCurve(DiscountCurve):

IborSingleCurve

Create an instance of a FinIbor curve given a valuation date and a set of ibor deposits, ibor FRAs and iborSwaps. Some of these may be left None and the algorithm will just use what is provided. An interpolation method has also to be provided. The default is to use a linear interpolation for swap rates on coupon dates and to then assume flat forwards between these coupon dates. The curve will assign a discount factor of 1.0 to the valuation date. If no instrument is starting on the valuation date, the curve is then assumed to be flat out to the first instrument using its zero rate.

```
IborSingleCurve(valuation_date: Date, # This is the trade date (not T+2)
                 iborDeposits: list,
                 iborFRAs: list,
                 iborSwaps: list,
                 interp_type: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES,
                 checkRefit: bool = False): # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	This is the trade date (not T+2)	-
iborDeposits	list	-	-
iborFRAs	list	-	-
iborSwaps	list	-	-
interp_type	FinInterpTypes	-	FLAT_FWD_RATES
checkRefit	bool	Set to True to test it works	False

8.15 FinIborSwaption

Class: *FinIborSwaption()*

class FinIborSwaption():

FinIborSwaption

Create a European-style swaption by defining the exercise date of the swaption, and all of the details of the underlying interest rate swap including the fixed coupon and the details of the fixed and the floating leg payment schedules. Bermudan style swaption should be priced using the *FinIborBermudanSwaption* class.

```
FinIborSwaption(settlement_date: Date,
                exerciseDate: Date,
                maturity_date: Date,
                fixed_legType: FinSwapTypes,
                fixedCoupon: float,
                fixedFrequencyType: FrequencyTypes,
                fixedDayCountType: DayCountTypes,
                notional: float = ONE_MILLION,
                floatFrequencyType: FrequencyTypes = FrequencyTypes.QUARTERLY,
                floatDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
                calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
                bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlement_date	Date	-	-
exerciseDate	Date	-	-
maturity_date	Date	-	-
fixed_legType	FinSwapTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FrequencyTypes	-	-
fixedDayCountType	DayCountTypes	-	-
notional	float	-	ONE_MILLION
floatFrequencyType	FrequencyTypes	-	FrequencyTypes.QUARTERLY
floatDayCountType	DayCountTypes	-	DayCountTypes.THIRTY_E_360
calendar_type	CalendarTypes	-	CalendarTypes.WEEKEND
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Valuation of a Ibor European-style swaption using a choice of models on a specified valuation date. Models include *FinModelBlack*, *FinModelBlackShifted*, *FinModelSABR*, *FinModelSABRShifted*, *FinModelHW*, *FinModelBK* and *FinModelBDT*. The last two involved a tree-based valuation.

```
value(valuation_date,
      discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
discount_curve	-	-	-
model	-	-	-

cashSettledValue

Valuation of a Ibor European-style swaption using a cash settled approach which is a market convention that used Black's model and that discounts all of the future payments at a flat swap rate. Note that the Black volatility for this valuation should in general not equal the Black volatility for the standard arbitrage-free valuation.

```
cashSettledValue(valuation_date: Date,
                  discount_curve,
                  swap_rate: float,
                  model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
discount_curve	-	-	-
swap_rate	float	-	-
model	-	-	-

printSwapFixedLeg

PLEASE ADD A FUNCTION DESCRIPTION

```
printSwapFixedLeg():
```

The function arguments are described in the following table.

printSwapFloatLeg

PLEASE ADD A FUNCTION DESCRIPTION

```
printSwapFloatLeg():
```

The function arguments are described in the following table.

8.16 FinOIS

Enumerated Type: FinCompoundingTypes

This enumerated type has the following values:

- COMPOUNDED
- OVERNIGHT_COMPOUNDED_ANNUAL_RATE
- AVERAGED
- AVERAGED_DAILY

Class: FinOIS

class FinOIS:

FinOI

Create an overnight index swap contract giving the contract start date, its maturity, fixed coupon, fixed leg frequency, fixed leg day count convention and notional. The floating leg parameters have default values that can be overwritten if needed. The start date is contractual and is the same as the settlement date for a new swap. It is the date on which interest starts to accrue. The end of the contract is the termination date. This is not adjusted for business days. The adjusted termination date is called the maturity date. This is calculated.

```
FinOI(effective_date: Date, # Date interest starts to accrue
      termination_date_or_tenor: (Date, str), # Date contract ends
      fixed_legType: FinSwapTypes,
      fixedCoupon: float, # Fixed coupon (annualised)
      fixedFreqType: FrequencyTypes,
      fixedDayCountType: DayCountTypes,
      notional: float = ONE_MILLION,
      payment_lag: int = 0, # Number of days after period payment occurs
      floatSpread: float = 0.0,
      floatFreqType: FrequencyTypes = FrequencyTypes.ANNUAL,
      floatDayCountType: DayCountTypes = DayCountTypes.THIRTY_E_360,
      calendar_type: CalendarTypes = CalendarTypes.WEEKEND,
      bus_day_adjust_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
      date_gen_rule_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_date	Date	Date interest starts to accrue	-
termination_date_or_tenor	Date or str	Date contract ends	-
fixed_legType	FinSwapTypes	-	-
fixedCoupon	float	Fixed coupon (annualised)	-
fixedFreqType	FrequencyTypes	-	-
fixedDayCountType	DayCountTypes	-	-
notional	float	-	ONE_MILLION
payment_lag	int	Number of days after period payment occurs	0
floatSpread	float	-	0.0
floatFreqType	FrequencyTypes	-	FrequencyTypes.ANNUAL
floatDayCountType	DayCountTypes	-	DayCountTypes.THIRD
calendar_type	CalendarTypes	-	CalendarTypes.WE
bus_day_adjust_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FC
date_gen_rule_type	DateGenRuleTypes	-	DateGenRuleTypes.BA

value

Value the interest rate swap on a value date given a single Ibor discount curve.

```
value(valuation_date: Date,
      oisCurve: DiscountCurve,
      firstFixingRate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
oisCurve	DiscountCurve	-	-
firstFixingRate	-	-	None

pv01

Calculate the value of 1 basis point coupon on the fixed leg.

```
pv01(valuation_date, discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
discount_curve	-	-	-

swap_rate

Calculate the fixed leg coupon that makes the swap worth zero. If the valuation date is before the swap payments start then this is the forward swap rate as it starts in the future. The swap rate is then a forward

swap rate and so we use a forward discount factor. If the swap fixed leg has begun then we have a spot starting swap.

```
swap_rate(valuation_date, oisCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	-	-	-
oisCurve	-	-	-

printFixedLegPV

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
printFixedLegPV():
```

The function arguments are described in the following table.

printFloatLegPV

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
printFloatLegPV():
```

The function arguments are described in the following table.

print_flows

Prints the fixed leg amounts without any valuation details. Shows the dates and sizes of the promised fixed leg flows.

```
print_flows():
```

The function arguments are described in the following table.

8.17 FinOISCurve

Class: OISCurve(DiscountCurve)

class OISCurve(DiscountCurve):

OISCurve

Create an instance of an overnight index rate swap curve given a valuation date and a set of OIS rates. Some of these may be left None and the algorithm will just use what is provided. An interpolation method has also to be provided. The default is to use a linear interpolation for swap rates on coupon dates and to then assume flat forwards between these coupon dates. The curve will assign a discount factor of 1.0 to the valuation date.

```
OISCurve(valuation_date: Date,
         oisDeposits: list,
         oisFRAs: list,
         oisSwaps: list,
         interp_type: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES,
         checkRefit: bool = False): # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuation_date	Date	-	-
oisDeposits	list	-	-
oisFRAs	list	-	-
oisSwaps	list	-	-
interp_type	FinInterpTypes	-	FLAT_FWD_RATES
checkRefit	bool	Set to True to test it works	False

Chapter 9

financepy.products.fx

FX Derivatives

Overview

These modules price and produce the sensitivity measures needed to hedge a range of FX Options and other derivatives with an FX underlying.

FX Forwards

Calculate the price and breakeven forward FX Rate of an FX Forward contract.

FX Vanilla Option

FX Option

This is a class from which other classes inherit and is used to perform simple perturbatory calculation of option Greeks.

FX Barrier Options

FX Basket Options

FX Digital Options

FX Fixed Lookback Option

FX Float Lookback Option

FX Rainbow Option

FX Variance Swap

Chapter 10

financepy.models

Models

Overview

This folder contains a range of models used in the various derivative pricing models implemented in the product folder. These include credit models for valuing portfolio credit products such as CDS Tranches, Monte-Carlo based models of stochastic processes used to value equity, FX and interest rate derivatives, and some generic implementations of models such as a tree-based Hull White model. Because the models are useful across a range of products, it is better to factor them out of the product/asset class categorisation as it avoids any unnecessary duplication.

In addition we seek to make the interface to these models rely only on fast types such as floats and integers and Numpy arrays.

These modules hold all of the models used by FinancePy across asset classes.

The general philosophy is to separate where possible product and models so that these models have as little product knowledge as possible.

Also, Numba is used extensively, resulting in code speedups of between x 10 and x 100.

Generic Arbitrage-Free Models

There are the following arbitrage-free models:

- `FinModelBlack` is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward is lognormally distributed.
- `FinModelBlackShifted` is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward plus a shift is lognormally distributed. CHECK
- `FinModelBachelier` prices options assuming the underlying evolves according to a Gaussian (normal) process.
- `FinSABR Model` is a stochastic volatility model for forward values with a closed form approximate solution for the implied volatility. It is widely used for pricing European style interest rate options, specifically caps and floors and also swaptions.

- `FinSABRShiftedModel` is a stochastic volatility model for forward value with a closed form approximate solution for the implied volatility. It is widely used for pricing European style interest rate options, specifically caps and floors and also swaptions.

The following asset-specific models have been implemented:

Equity Models

- `FinHestonModel`
- `FinHestonModelProcess`
- `FinProcessSimulator`

Interest Rate Models

Equilibrium Rate Models

There are two main short rate models.

- `FinCIRRateModel` is a short rate model where the randomness component is proportional to the square root of the short rate. This model implementation is not arbitrage-free across the term structure.
- `FinVasicekRateModel` is a short rate model that assumes mean-reversion and normal volatility. It has a closed form solution for bond prices. It does not have the flexibility to fit a term structure of interest rates. For that you need to use the more flexible Hull-White model.

Arbitrage Free Rate Models

- `FinBlackKaraskinskiRateModel` is a short rate model in which the log of the short rate follows a mean-reverting normal process. It refits the interest rate term structure. It is implemented as a trinomial tree and allows valuation of European and American-style rate-based options.
- `FinHullWhiteRateModel` is a short rate model in which the short rate follows a mean-reverting normal process. It fits the interest rate term structure. It is implemented as a trinomial tree and allows valuation of European and American-style rate-based options. It also implements Jamshidian's decomposition of the bond option for European options.

Credit Models

- `FinGaussianCopula1FModel` is a Gaussian copula one-factor model. This class includes functions that calculate the portfolio loss distribution. This is numerical but deterministic.
- `FinGaussianCopulaLHPModel` is a Gaussian copula one-factor model in the limit that the number of credits tends to infinity. This is an asymptotic analytical solution.
- `FinGaussianCopulaModel` is a Gaussian copula model which is multifactor model. It has a Monte-Carlo implementation.
- `FinLossDbnBuilder` calculates the loss distribution.

- FinMertonCreditModel is a model of the firm as proposed by Merton (1974).

FX Models

10.1 FinModel

Class: FinModel()

class FinModel():

FinModel

PLEASE ADD A FUNCTION DESCRIPTION

```
FinModel(implementationType,  
         parameterDict: dict):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
implementationType	-	-	-
parameterDict	dict	-	-

10.2 FinModelBlackScholesMCSpeedTests

value_mc1

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc1(s0, t, K, r, q, v, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
t	-	-	-
K	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-

value_mc2

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc2(s0, t, K, r, q, v, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
t	-	-	-
K	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-

value_mc3

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc3(s0, t, K, r, q, v, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
t	-	-	-
K	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-

value_mc4

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc4(s0, t, K, r, q, v, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
t	-	-	-
K	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-

value_mc5

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc5(s0, t, K, r, q, v, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
t	-	-	-
K	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-

10.3 FinModelMertonCreditMkt

Class: *FinModelMertonCreditMkt(FinModelMertonCredit)*

class FinModelMertonCreditMkt(FinModelMertonCredit):

FinModelMertonCreditMkt

Create an object that holds all of the model parameters. These parameters may be vectorised.

```
FinModelMertonCreditMkt(equityValue: (float, list, np.ndarray),
                        bondFace: (float, list, np.ndarray),
                        timeToMaturity: (float, list, np.ndarray),
                        riskFreeRate: (float, list, np.ndarray),
                        assetGrowthRate: (float, list, np.ndarray),
                        equityVolatility: (float, list, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
equityValue	float or list,np.ndarray	-	-
bondFace	float or list,np.ndarray	-	-
timeToMaturity	float or list,np.ndarray	-	-
riskFreeRate	float or list,np.ndarray	-	-
assetGrowthRate	float or list,np.ndarray	-	-
equityVolatility	float or list,np.ndarray	-	-

10.4 FinModelOptionImpliedDbn

optionImpliedDbn

This function calculates the option smile/skew-implied probability density function times the interval width.

```
optionImpliedDbn(s, t, r, q, strikes, sigmas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
r	-	-	-
q	-	-	-
strikes	-	-	-
sigmas	-	-	-